Documentação do SUSE Edge

Documentação do SUSE Edge

Data de Publicação: 2025-07-09

https://documentation.suse.com

Conteúdo

Documentação do SUSE Edge 3.3.1

Boas-vindas à documentação do SUSE Edge. Você vai encontrar uma visão geral de alto nível da arquitetura, guias de início rápido, designs validados, orientação de como usar os componentes, integrações de terceiros e melhores práticas de gerenciamento da infraestrutura e das cargas de trabalho de computação de borda.

1 O que é o SUSE Edge?

O SUSE Edge é uma solução completa, amplamente validada, de estreita integração e desenvolvida com um propósito específico para resolver os desafios exclusivos da implantação da infraestrutura e dos aplicativos nativos de nuvem na borda. Ele tem como objetivo principal oferecer uma plataforma persistente, porém bastante flexível, altamente escalável e segura que abrange a criação de imagem da implantação inicial, o provisionamento e a integração de nós, a implantação de aplicativos, a observabilidade e as operações do ciclo de vida completo. A plataforma foi criada do zero, com base no melhor software de código aberto, e condiz com os nossos mais de 30 anos de história como provedores de plataformas SUSE Linux seguras, estáveis e certificadas e com a nossa experiência em oferecer gerenciamento Kubernetes altamente escalável e repleto de recursos com o nosso portfólio Rancher. O SUSE Edge é fundamentado nesses recursos para entregar funcionalidades que atendem a inúmeros segmentos de mercado, como varejo, medicina, transporte, logística, telecomunicações, manufatura inteligente e IoT industrial.

2 Filosofia do design

A solução foi criada com a ideia de que não existe uma plataforma de borda do tipo "tamanho único" porque os requisitos e as expectativas dos clientes variam de maneira significativa. As implantações de borda nos levam a resolver (e sempre aprimorar) alguns dos problemas mais desafiadores, como escalabilidade massiva, disponibilidade de rede restrita, limitações de espaço físico, novas ameaças à segurança e vetores de ataque, variações na arquitetura de hardware e nos recursos de sistema, a necessidade de implantar e estabelecer interface com infraestruturas e aplicativos legados e soluções de clientes com durações estendidas. Como muitos desses desafios são diferentes das mentalidades tradicionais, por exemplo, a implantação de infraestrutura e aplicativos em centros de dados ou na nuvem pública, precisamos analisar o design de modo muito mais detalhado e repensar várias hipóteses que antes eram comuns.

Por exemplo, consideramos importante o minimalismo, a modularidade e a facilidade das operações. O minimalismo é essencial nos ambientes de borda, já que as chances de um sistema falhar aumentam de acordo com a sua complexidade. Quando há centenas, ou até centenas de milhares de locais, os sistemas complexos vão apresentar falhas igualmente complexas. A modularidade em nossa solução oferece mais opções aos usuários e elimina a complexidade desnecessária da plataforma implantada. Precisamos também levar em consideração a facilidade das operações. As pessoas tendem a cometer erros quando repetem um processo milhares de vezes, portanto, a plataforma deve garantir que possíveis erros sejam recuperáveis, acabando com a necessidade de visitas técnicas, mas também buscar a consistência e a padronização.

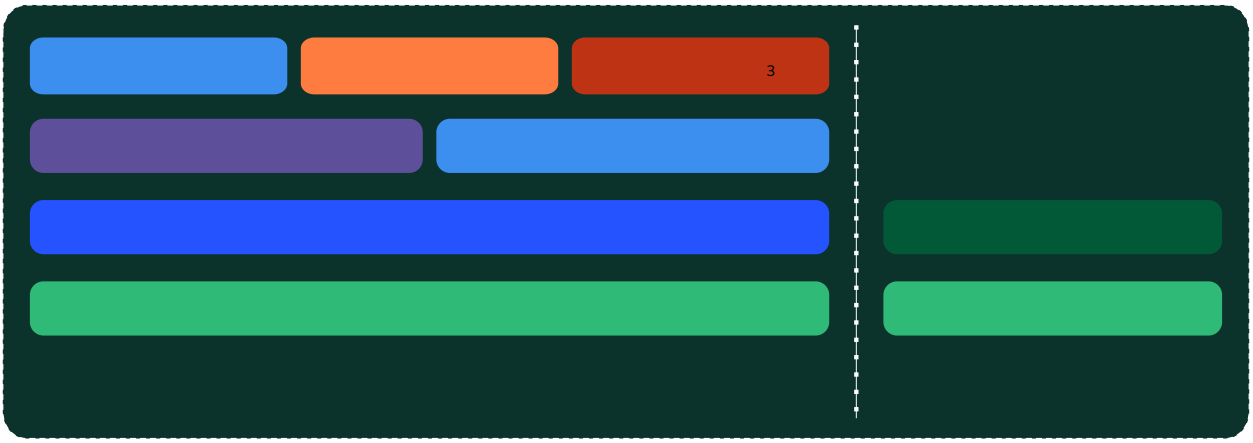
3 Arquitetura de alto nível

A arquitetura de alto nível do SUSE Edge está dividida em duas categorias principais, os chamados clusters de "gerenciamento" e "downstream". O cluster de gerenciamento é responsável pelo gerenciamento remoto de um ou mais clusters downstream, apesar de ser reconhecido que, em determinadas situações, os clusters downstream precisam operar sem gerenciamento remoto, por exemplo, quando um site de borda não tem conectividade externa e precisa operar de maneira independente. No SUSE Edge, os componentes técnicos usados na operação de ambos os clusters de gerenciamento e downstream são muito comuns, embora possam apresentar diferenças referentes a especificações de sistema e aos aplicativos nos quais residem, ou seja, o cluster de gerenciamento executa aplicativos que possibilitam as operações de ciclo de vida e gerenciamento de sistemas, enquanto os clusters downstream atendem aos requisitos para executar os aplicativos de usuário.

3.1 Componentes usados no SUSE Edge

O SUSE Edge é composto de componentes existentes tanto do SUSE quanto do Rancher, junto com outros recursos e componentes desenvolvidos pela equipe do Edge, para enfrentarmos as restrições e as complexidades previstas na computação de borda. Veja abaixo uma explicação dos componentes usados nos clusters de gerenciamento e downstream, com um diagrama simplificado de alto nível (observe que não se trata de uma lista completa):

3.1.1 Cluster de gerenciamento



- **Gerenciamento**: trata-se da parte centralizada do SUSE Edge usada para gerenciar o provisionamento e o ciclo de vida dos clusters downstream conectados. Normalmente, o cluster de gerenciamento inclui os seguintes componentes:
  - Gerenciamento multicluster com o Rancher Prime (*Capítulo 5, Rancher*), que oferece um dashboard comum para integração de clusters downstream e gerenciamento do ciclo de vida contínuo da infraestrutura e dos aplicativos, além do isolamento total de locatários e das integrações de `IDP` (provedor de identidade), um amplo mercado de integrações e extensões de terceiros e uma API independente de fornecedor.
  - Gerenciamento de sistemas Linux com o SUSE Multi-Linux Manager, que permite o gerenciamento automatizado de patches e configurações do sistema operacional Linux subjacente (*SUSE Linux Micro (*Capítulo 9, SUSE Linux Micro*)) executado nos clusters downstream. Quando esse componente está conteinerizado, observe que ele precisa ser executado em um sistema separado dos outros componentes de gerenciamento, devidamente identificado no diagrama acima como "Linux Management".

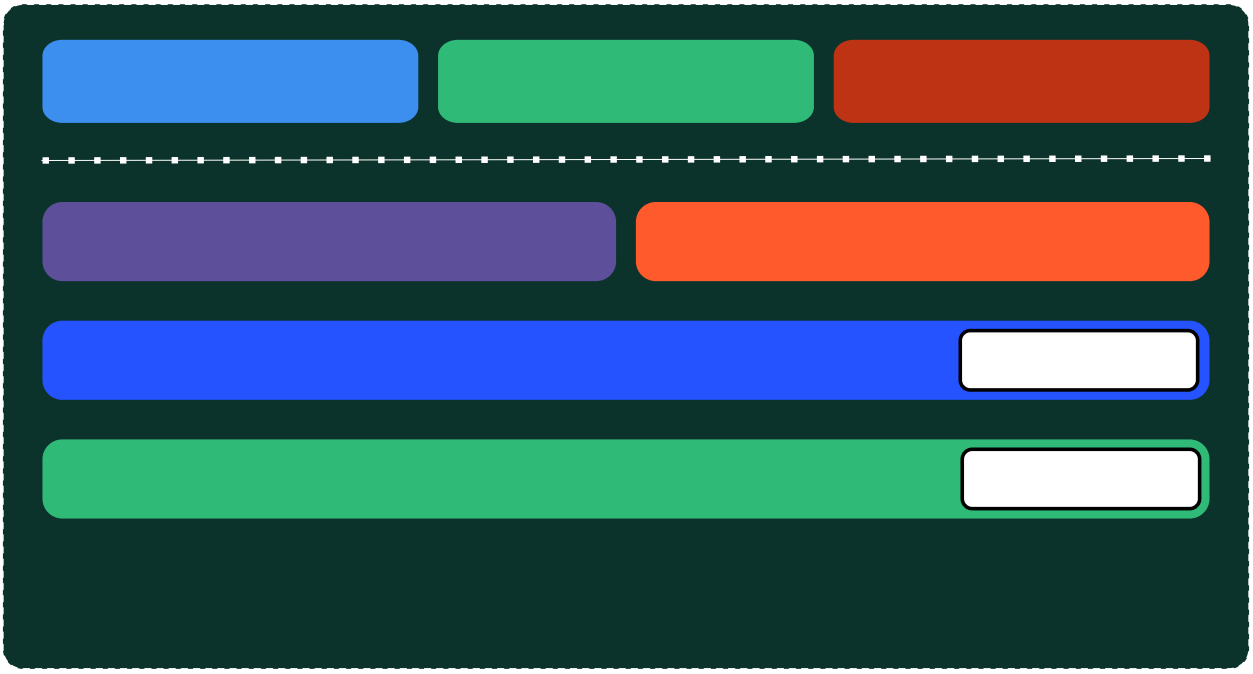
- Um controlador dedicado de gerenciamento do ciclo de vida (*Capítulo 23, Controller de upgrade*) encarregado dos upgrades dos componentes do cluster de gerenciamento para uma determinada versão do SUSE Edge.

- Integração remota do sistema ao Rancher Prime com Elemental (*Capítulo 13, Elemental*), que permite a vinculação posterior dos nós de borda conectados aos clusters Kubernetes e a implantação de aplicativos, por exemplo, pelo GitOps.

- Suporte opcional completo ao ciclo de vida e gerenciamento bare metal com os provedores de infraestrutura Metal3 (*Capítulo 10, Metal*^{3}), MetalLB (*Capítulo 19, MetalLB*) e `CAPI` (Cluster API), que permite o provisionamento total, de ponta a ponta, dos sistemas bare metal com recursos de gerenciamento remoto.

- Um mecanismo opcional do GitOps chamado Fleet (*Capítulo 8, Fleet*) que gerencia o provisionamento e o ciclo de vida dos clusters downstream e dos aplicativos que residem neles.
- O cluster de gerenciamento conta com o SUSE Linux Micro (*Capítulo 9, SUSE Linux Micro*) como o alicerce do sistema operacional de base e com o RKE2 (*Capítulo 16, RKE2*) como a distribuição Kubernetes que oferece suporte aos aplicativos do cluster de gerenciamento.

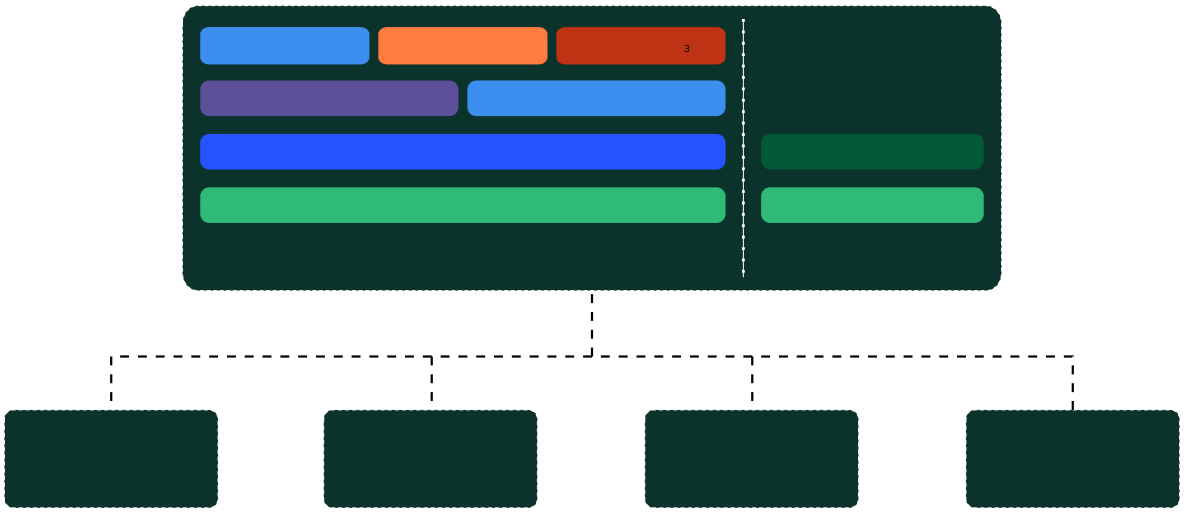
3.1.2 Clusters downstream



- **Downstream**: trata-se da parte distribuída do SUSE Edge usada para executar as cargas de trabalho dos usuários no Edge, ou seja, o software que é executado no próprio local de borda, e que normalmente inclui os seguintes componentes:
  - Uma seleção de distribuições Kubernetes, com distribuições seguras e leves como K3s (*Capítulo 15, K3s*) e RKE2 (*Capítulo 16, RKE2*) (RKE2 é protegido, certificado e otimizado para uso em órgãos governamentais e setores regulamentados).
  - SUSE Security (*Capítulo 18, SUSE Security*) para habilitar recursos de segurança, como verificação de vulnerabilidade de imagens, inspeção detalhada de pacotes e proteção contra ameaças e vulnerabilidades em tempo real.

- Armazenamento de software em blocos com o SUSE Storage (*Capítulo 17, SUSE Storage*), que possibilita o armazenamento em blocos leve, persistente, resiliente e escalável.
- Um sistema operacional Linux leve, otimizado para contêiner e protegido com o SUSE Linux Micro (*Capítulo 9, SUSE Linux Micro*), que oferece um sistema operacional imutável e altamente resiliente para execução de contêineres e máquinas virtuais na borda. O SUSE Linux Micro está disponível para as arquiteturas AArch64 e AMD64/Intel 64 e também suporta o `kernel Real-Time` para aplicativos sensíveis à latência (por exemplo, os casos de uso de telecomunicações).
- Para os clusters conectados (ou seja, os que têm conectividade com o cluster de gerenciamento), dois agentes são implantados: Agente do sistema Rancher, para gerenciar a conectividade com o Rancher Prime, e venv-salt-minion, para receber as instruções do SUSE Multi-Linux Manager e aplicar as atualizações de software Linux. Eles não são obrigatórios para o gerenciamento de clusters desconectados.

3.2 Conectividade



A imagem acima apresenta uma visão geral de alto nível da arquitetura dos clusters downstream **conectados** e sua relação com o cluster de gerenciamento. É possível implantar o cluster de gerenciamento em uma ampla variedade de plataformas de infraestrutura subjacentes, em instalações tanto no local quanto na nuvem, dependendo da disponibilidade de rede entre os

clusters downstream e o cluster de gerenciamento de destino. O único requisito para que isso funcione é que a API e o URL de callback estejam acessíveis pela rede que conecta os nós do cluster downstream à infraestrutura de gerenciamento.

É importante reconhecer que existem mecanismos em que a conectividade é estabelecida e que são diferentes do mecanismo de implantação do cluster downstream. Há uma explicação mais detalhada sobre isso na próxima seção; mas, para um entendimento geral, há três mecanismos principais para estabelecer os clusters downstream conectados como um cluster "gerenciado":

1. Os clusters downstream são implantados primeiro no modo "desconectado" (por exemplo, pelo Edge Image Builder (*Capítulo 11, Edge Image Builder*)) e depois são importados para o cluster de gerenciamento se ou quando a conectividade permite.
2. Os clusters downstream são configurados para usar o mecanismo de integração incorporado (por exemplo, pelo Elemental (*Capítulo 13, Elemental*)) e são automaticamente registrados no cluster de gerenciamento na primeira inicialização, o que permite a vinculação posterior da configuração do cluster.
3. Os clusters downstream foram provisionados com os recursos de gerenciamento bare metal (CAPI + Metal^3) e são automaticamente importados para o cluster de gerenciamento após a implantação e configuração do cluster (pelo operador Rancher Turtles).

Nota

É recomendada a implementação de vários clusters de gerenciamento para acomodar a grande escala de implantações, otimizar os ambientes geograficamente dispersos conforme as expectativas de largura de banda e de latência e minimizar as interrupções em caso de paradas ou upgrades do cluster de gerenciamento. Você encontra os limites de escalabilidade dos clusters de gerenciamento e os requisitos do sistema atuais aqui (https://ranchermanager.docs.rancher.com/v2.11/getting-started/installation-and-upgrade/installation-requirements).

4 Padrões de implantação de borda comuns

Devido à variedade de ambientes operacionais e requisitos de ciclo de vida, implementamos o suporte a alguns padrões de implantação que se adaptam de maneira flexível aos segmentos de mercado e casos de uso em que o SUSE Edge atua. Documentamos um guia de início rápido para cada um desses padrões de implantação para ajudar você a se familiarizar com a plataforma

SUSE Edge de acordo com as suas necessidades. Os três padrões de implantação com suporte atualmente estão descritos a seguir, com um link para a respectiva página do guia de início rápido.

4.1 Provisionamento de rede direcionado

O provisionamento de rede direcionado é por onde você fica sabendo dos detalhes do hardware em que deseja implantar e tem acesso direto à interface de gerenciamento fora da banda para orquestrar e automatizar todo o processo de provisionamento. Nesse cenário, nossos clientes esperam uma solução capaz de provisionar sites de borda completamente automatizados de um local centralizado, indo muito além da criação de uma imagem de inicialização, minimizando as operações manuais no local de borda: basta montar, ligar e conectar as redes necessárias ao hardware físico, e o processo de automação liga a máquina por meio do gerenciamento fora da banda (por exemplo, pela API Redfish) e cuida do provisionamento, da integração e da implantação da infraestrutura sem intervenção do usuário. A chave para que esse processo funcione é que os administradores conheçam os sistemas e saibam qual hardware está em que local, e que a implantação seja realizada centralmente.

Esta solução é a mais robusta, já que você interage diretamente com a interface de gerenciamento do hardware, trabalha com um hardware conhecido e tem menos restrições de disponibilidade de rede. Quanto às funcionalidades, a solução usa amplamente a Cluster API e o Metal^3 para o provisionamento automatizado, de bare metal a sistema operacional, Kubernetes e aplicativos em camadas, e oferece a opção de se vincular aos demais recursos de gerenciamento do ciclo de vida comuns de pós-implantação do SUSE Edge. O início rápido da solução está disponível em *Capítulo 1, Implantações automatizadas de BMC com Metal*^{3}.

4.2 Provisionamento de rede "phone home"

Pode acontecer de você trabalhar em um ambiente onde o cluster de gerenciamento central não pode gerenciar o hardware diretamente (por exemplo, a rede remota é protegida por firewall ou não há uma interface de gerenciamento fora da banda; o que é comum em hardware do tipo "PC" mais encontrado na borda). Nesse cenário, oferecemos ferramentas para provisão remota de clusters e suas cargas de trabalho sem a necessidade de saber para onde o hardware é enviado quando é iniciado. É isto que a maioria das pessoas pensa sobre computação de borda: são milhares, ou dezenas de milhares, de sistemas um tanto desconhecidos que são iniciados nos locais de borda e se comunicam de maneira segura com a central ("phone home"), validando

suas identidades e recebendo as instruções do que devem fazer. Nesse caso, nossos requisitos são o provisionamento e o gerenciamento do ciclo de vida com muito pouca intervenção do usuário, além de pré-criar a imagem da máquina de fábrica ou simplesmente anexar uma imagem de inicialização, por exemplo, por USB, e ligar o sistema. O principal desafio nesse caso é abordar a escala, a consistência, a segurança e o ciclo de vida desses dispositivos no mundo real.

Esta solução oferece um ótimo nível de flexibilidade e consistência no modo como os sistemas são provisionados e integrados, seja qual for seu local, tipo ou especificação ou quando são ligados pela primeira vez. O SUSE Edge oferece total flexibilidade e personalização do sistema pelo Edge Image Builder, além de aproveitar a oferta de recursos de registro Elemental do Rancher para integração de nós e provisionamento do Kubernetes, junto com o SUSE Multi-Linux Manager para aplicação de patches no sistema operacional. O início rápido da solução está disponível no *Capítulo 2, Integração remota de host com o Elemental*.

4.3 Provisionamento com base na imagem

Para os clientes que precisam trabalhar em ambientes independentes, air-gapped ou com limitações de rede, o SUSE Edge oferece uma solução que permite gerar mídias de instalação totalmente personalizadas com todos os artefatos de implantação necessários para habilitar clusters Kubernetes de alta disponibilidade na borda, tanto de um quanto de vários nós, incluindo qualquer carga de trabalho ou componentes adicionais em camadas que sejam necessários. Tudo isso sem precisar de conectividade de rede com ambientes externos e sem a intervenção de uma plataforma de gerenciamento centralizada. A experiência do usuário é muito parecida com a solução "phone home", no que se refere à mídia de instalação fornecida aos sistemas de destino, mas a solução é "iniciada no local". Nesse cenário, é possível conectar os clusters resultantes ao Rancher para gerenciamento contínuo (ou seja, passar do modo de operação "desconectado" para "conectado" sem precisar de reconfiguração ou reimplantação significativa) ou continuar a operação de forma isolada. Observe que, em ambos os casos, é possível usar o mesmo mecanismo consistente para automatizar as operações de ciclo de vida.

Além disso, é possível usar a solução para criar rapidamente clusters de gerenciamento que podem hospedar a infraestrutura centralizada que sustenta os modelos de provisionamento de rede tanto "direcionado" quanto "phone home", já que pode ser a forma mais rápida e simples de provisionar todos os tipos de infraestrutura de borda. Essa solução faz uso intensivo dos recursos do SUSE Edge Image Builder para criar mídias de instalação totalmente personalizadas e autônomas. O início rápido está disponível no *Capítulo 3, Clusters independentes com o Edge Image Builder*.

5 Validação da pilha do SUSE Edge

Todas as versões do SUSE Edge têm componentes estreitamente integrados e validados na íntegra que são lançados juntos. Como parte do trabalho constante de integração e validação de pilha que testa a integração entre os componentes e garante que o desempenho do sistema atenda às expectativas em cenários de falha forçada, a equipe do SUSE Edge publica todas as execuções e os resultados dos testes. Os resultados e todos os parâmetros de entrada estão disponíveis em ci.edge.suse.com (https://ci.edge.suse.com).

6 Lista completa de componentes

A lista completa de componentes, com um link para a descrição de alto nível de cada um e como são usados no SUSE Edge, está disponível abaixo:

- Rancher (*Capítulo 5, Rancher*)
- Extensões do Rancher Dashboard (*Capítulo 6, Extensões do Rancher Dashboard*)
- Rancher Turtles (*Capítulo 7, Rancher Turtles*)
- SUSE Multi-Linux Manager
- Fleet (*Capítulo 8, Fleet*)
- SUSE Linux Micro (*Capítulo 9, SUSE Linux Micro*)
- Metal^3 (*Capítulo 10, Metal^3*)
- Edge Image Builder (*Capítulo 11, Edge Image Builder*)
- NetworkManager Configurator (*Capítulo 12, Rede de borda*)
- Elemental (*Capítulo 13, Elemental*)
- Akri (*Capítulo 14, Akri*)
- K3s (*Capítulo 15, K3s*)
- RKE2 (*Capítulo 16, RKE2*)
- SUSE Storage (*Capítulo 17, SUSE Storage*)
- SUSE Security (*Capítulo 18, SUSE Security*)

- MetalLB (*Capítulo 19, MetalLB*)
- KubeVirt (*Capítulo 21, Edge Virtualization*)
- System Upgrade Controller (*Capítulo 22, System Upgrade Controller*)
- Controller de upgrade (*Capítulo 23, Controller de upgrade*)

I Guias de início rápido

Estes são os guias de início rápido

1 Implantações automatizadas de BMC com Metal^3

Metal^3 é um projeto da CNCF (https://metal3.io/) que oferece recursos de gerenciamento de infraestrutura bare metal para Kubernetes.

O Metal^3 inclui recursos nativos do Kubernetes para gerenciar o ciclo de vida de servidores bare metal que suportam gerenciamento com protocolos fora da banda, como o Redfish (https://www.dmtf.org/standards/redfish).

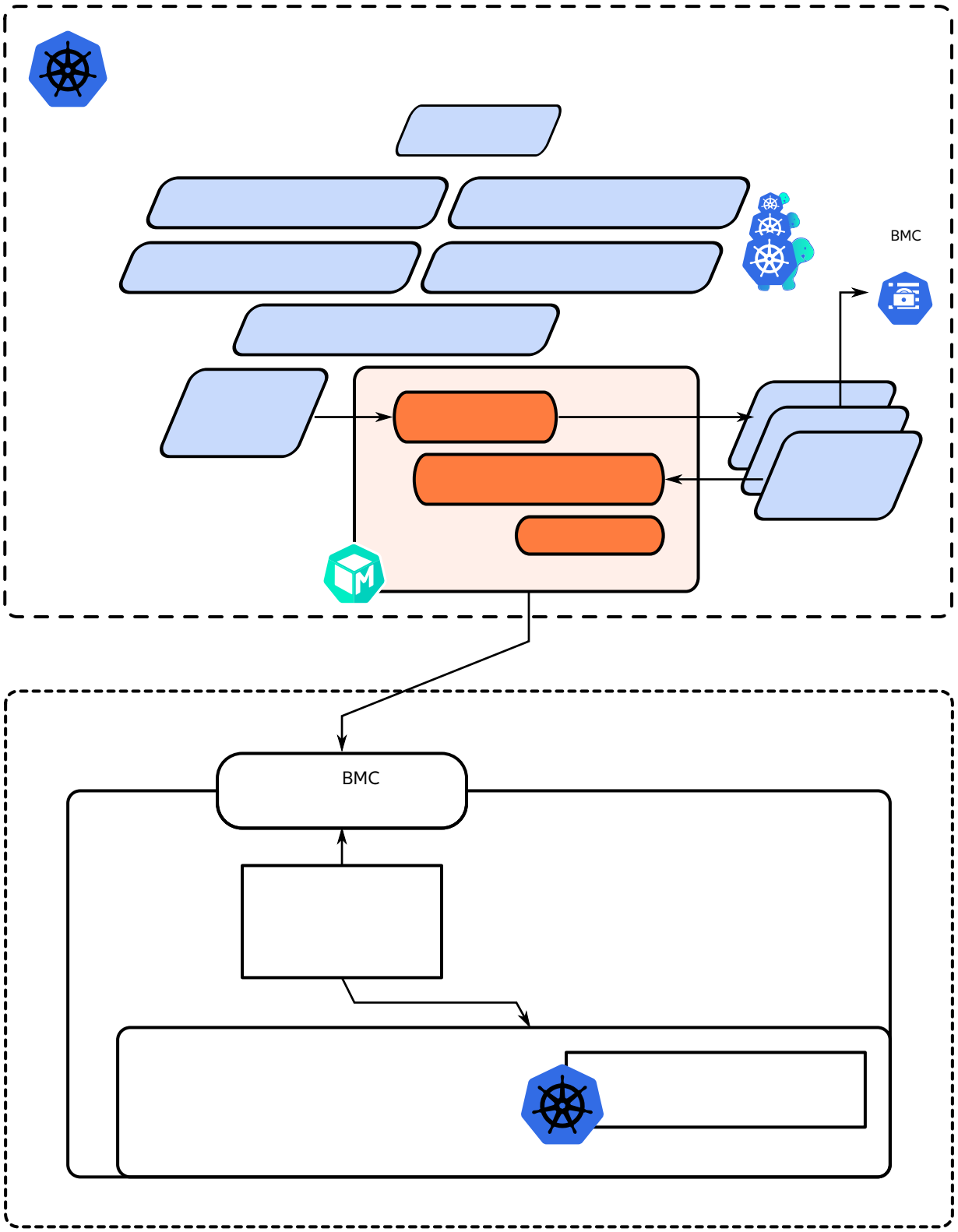
Ele também conta com suporte a Cluster API (CAPI) (https://cluster-api.sigs.k8s.io/), que permite gerenciar os recursos da infraestrutura de vários provedores por meio de APIs amplamente adotadas e independentes de fornecedor.

1.1 Por que usar este método

Este método é útil em cenários com hardware de destino que permite o gerenciamento fora da banda e quando se deseja um fluxo de gerenciamento de infraestrutura automatizado.

O cluster de gerenciamento é configurado para oferecer APIs declarativas que permitem o gerenciamento de inventário e de estado dos servidores bare metal do cluster downstream, incluindo inspeção automatizada, limpeza e provisionamento/desprovisionamento.

1.2 Arquitetura de alto nível



1.3 Pré-requisitos

Há algumas restrições específicas relacionadas ao hardware e à rede de servidor do cluster downstream:

- Cluster de gerenciamento
  - É preciso ter conectividade de rede com a API de gerenciamento/BMC do servidor de destino
  - É preciso ter conectividade com a rede do plano de controle do servidor de destino
  - Para clusters de gerenciamento de vários nós, é necessário um endereço IP reservado adicional
- Hosts para serem controlados
  - É preciso ter suporte a gerenciamento fora da banda pelas interfaces do Redfish, iDRAC ou iLO
  - É preciso ter suporte à implantação por mídia virtual (não há suporte a PXE no momento)
  - É preciso ter conectividade de rede com o cluster de gerenciamento para acessar as APIs de provisionamento do Metal[3]

Algumas ferramentas são necessárias e podem ser instaladas no cluster de gerenciamento ou em um host que possa acessá-lo.

- Kubectl (https://kubernetes.io/docs/reference/kubectl/kubectl/), Helm (https://helm.sh) e Clusterctl (https://cluster-api.sigs.k8s.io/user/quick-start.html#install-clusterctl)
- Um tempo de execução do contêiner, como Podman (https://podman.io) ou Rancher Desktop (https://rancherdesktop.io)

Faça download do arquivo de imagem de sistema operacional `SL-Micro.x86_64-6.1-Base-GM.raw` pelo SUSE Customer Center (https://scc.suse.com/) ou pela página de download da SUSE (https://www.suse.com/download/sle-micro/).

1.3.1 Configurar o cluster de gerenciamento

As etapas básicas de instalação de um cluster de gerenciamento e uso do Metal^3 são:

1. Instalar um cluster de gerenciamento RKE2
2. Instalar o Rancher
3. Instalar um provedor de armazenamento (opcional)
4. Instalar as dependências do Metal^3
5. Instalar as dependências da CAPI pelo Rancher Turtles
6. Criar uma imagem do sistema operacional SLEMicro para hosts do cluster downstream
7. Registrar CRs de BareMetalHost para definir o inventário de bare metal
8. Criar um cluster downstream definindo recursos da CAPI

Este guia considera que já existe um cluster RKE2 e que o Rancher (incluindo o cert-manager) foi instalado, por exemplo, usando o Edge Image Builder (*Capítulo 11, Edge Image Builder*).

Dica

Estas etapas também podem ser totalmente automatizadas conforme descrito na documentação do cluster de gerenciamento (*Capítulo 40, Configurando o cluster de gerenciamento*).

1.3.2 Instalando as dependências do Metal^3

O cert-manager deve ser instalado e executado caso não tenha sido junto com a instalação do Rancher.

É necessário instalar um provedor de armazenamento persistente. O SUSE Storage é recomendado, mas também é possível usar o `local-path-provisioner` em ambientes de desenvolvimento/PoC. A instrução abaixo considera que StorageClass foi marcada como padrão (https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/); do contrário, será necessária uma configuração adicional para o gráfico do Metal^3.

Um IP adicional é necessário, que pode ser gerenciado por MetalLB (https://metallb.universe.tf/) para fornecer um endpoint consistente aos serviços de gerenciamento do Metal[3]. Esse IP deve fazer parte da sub-rede do plano de controle e ser reservado para configuração estática (e não fazer parte de um pool DHCP).

Dica

Se o cluster de gerenciamento é um nó único, é possível evitar o requisito de IP flutuante adicional gerenciado por MetalLB. Consulte a *Seção 1.6.1, "Configuração de nó único"*.

1. Vamos instalar primeiro o MetalLB:

```
helm install \
  metallb oci://registry.suse.com/edge/charts/metallb \
  --namespace metallb-system \
  --create-namespace
```

2. Na sequência, definimos um `IPAddressPool` e `L2Advertisement` usando o IP reservado, especificado abaixo como `STATIC_IRONIC_IP`:

```
export STATIC_IRONIC_IP=<STATIC_IRONIC_IP>

cat <<-EOF | kubectl apply -f -
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: ironic-ip-pool
  namespace: metallb-system
spec:
  addresses:
  - ${STATIC_IRONIC_IP}/32
  serviceAllocation:
    priority: 100
    serviceSelectors:
    - matchExpressions:
      - {key: app.kubernetes.io/name, operator: In, values: [metal3-ironic]}
EOF
```

```
cat <<-EOF | kubectl apply -f -
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: ironic-ip-pool-l2-adv
```

```
  namespace: metallb-system
spec:
  ipAddressPools:
  - ironic-ip-pool
EOF
```

3. Agora é possível instalar o Metal[3]:

```
helm install \
  metal3 oci://registry.suse.com/edge/charts/metal3 \
  --namespace metal3-system \
  --create-namespace \
  --set global.ironicIP="$STATIC_IRONIC_IP"
```

4. A execução do contêiner init pode levar cerca de dois minutos nesta implantação, portanto, garanta que todos os pods estejam em execução antes de continuar:

```
kubectl get pods -n metal3-system
NAME                                                READY   STATUS    RESTARTS
 AGE
baremetal-operator-controller-manager-85756794b-fz98d   2/2     Running   0
 15m
metal3-metal3-ironic-677bc5c8cc-55shd               4/4     Running   0
 15m
metal3-metal3-mariadb-7c7d6fdbd8-64c7l              1/1     Running   0
 15m
```

Atenção

Não avance para as etapas seguintes até que todos os pods no namespace `metal3-system` estejam em execução.

1.3.3 Instalando as dependências de API do cluster

As dependências de API do cluster são gerenciadas pelo gráfico Helm do Rancher Turtles:

```
cat > values.yaml <<EOF
rancherTurtles:
  features:
    embedded-capi:
      disabled: true
    rancher-webhook:
      cleanup: true
EOF
```

```
helm install \
  rancher-turtles oci://registry.suse.com/edge/charts/rancher-turtles \
  --namespace rancher-turtles-system \
  --create-namespace \
  -f values.yaml
```

Após algum tempo, os pods do controlador deverão estar em execução nos namespaces `capi-system`, `capm3-system`, `rke2-bootstrap-system` e `rke2-control-plane-system`.

1.3.4 Preparar a imagem do cluster downstream

O Kiwi (*Capítulo 28, Criando imagens atualizadas do SUSE Linux Micro com o Kiwi*) e o Edge Image Builder (*Capítulo 11, Edge Image Builder*) são usados para preparar a imagem base do SLEMicro modificado que foi provisionado nos hosts do cluster downstream.

Neste guia, abordamos a configuração mínima necessária para implantar o cluster downstream.

1.3.4.1 Configuração da imagem

Nota

Primeiro siga o *Capítulo 28, Criando imagens atualizadas do SUSE Linux Micro com o Kiwi* para gerar uma imagem nova como a primeira etapa necessária para criar clusters.

Ao executar o Edge Image Builder, um diretório é montado com base no host, portanto, é necessário criar uma estrutura de diretórios para armazenar os arquivos de configuração usados para definir a imagem de destino.

- `downstream-cluster-config.yaml` é o arquivo de definição da imagem. Consulte o *Capítulo 3, Clusters independentes com o Edge Image Builder* para obter mais detalhes.
- Quando o download da imagem base é feito, ela é compactada com `xz`, que deve ser descompactada com `unxz` e copiada/movida para a pasta `base-images`.
- A pasta `network` é opcional. Consulte a *Seção 1.3.5.1.1, "Script adicional para configuração de rede estática"* para obter mais detalhes.
- O diretório custom/scripts contém scripts que são executados na primeira inicialização. Atualmente, um script `01-fix-growfs.sh` é necessário para redimensionar a partição raiz do sistema operacional na implantação.

```
├── downstream-cluster-config.yaml
├── base-images/
│   └ SL-Micro.x86_64-6.1-Base-GM.raw
├── network/
│   └ configure-network.sh
└── custom/
    └ scripts/
        └ 01-fix-growfs.sh
```

1.3.4.1.1 Arquivo de definição da imagem do cluster downstream

O `downstream-cluster-config.yaml` é o arquivo de configuração principal para a imagem do cluster downstream. Veja a seguir um exemplo mínimo de implantação por meio do Metal[3]:

```
apiVersion: 1.2
image:
  imageType: raw
  arch: x86_64
  baseImage: SL-Micro.x86_64-6.1-Base-GM.raw
  outputImageName: SLE-Micro-eib-output.raw
operatingSystem:
  time:
    timezone: Europe/London
    ntp:
      forceWait: true
      pools:
        - 2.suse.pool.ntp.org
      servers:
        - 10.0.0.1
        - 10.0.0.2
  kernelArgs:
    - ignition.platform.id=openstack
    - net.ifnames=1
  systemd:
    disable:
      - rebootmgr
      - transactional-update.timer
      - transactional-update-cleanup.timer
  users:
    - username: root
      encryptedPassword: $ROOT_PASSWORD
      sshKeys:
      - $USERKEY1
  packages:
    packageList:
```

```
      - jq
  sccRegistrationCode: $SCC_REGISTRATION_CODE
```

Em que `$SCC_REGISTRATION_CODE` é o código de registro copiado do SUSE Customer Center (https://scc.suse.com/), e a lista de pacotes contém o `jq`, que é obrigatório.

`$ROOT_PASSWORD` é a senha criptografada do usuário root, que pode ser útil para teste/depuração. É possível gerá-la com o comando `openssl passwd -6 PASSWORD`.

Para os ambientes de produção, a recomendação é usar as chaves SSH que podem ser adicionadas ao bloco de usuários substituindo a `$USERKEY1` pelas chaves SSH reais.

Nota

O `net.ifnames=1` habilita a Nomenclatura de interface de rede previsível (https://documentation.suse.com/smart/network/html/network-interface-predictable-naming/index.html).

Isso corresponde à configuração padrão do gráfico do Metal[3], mas a configuração deve corresponder ao valor `predictableNicNames` do gráfico.

Veja também que o `ignition.platform.id=openstack` é obrigatório. Sem esse argumento, a configuração do SUSE Linux Micro pelo ignition vai falhar no fluxo automatizado do Metal[3].

A seção `time` é opcional, mas sua configuração é altamente recomendada para evitar possíveis problemas com certificados e divergência de relógio. O valor usado neste exemplo é somente para fins ilustrativos. Ajuste-o de acordo com seus requisitos específicos.

1.3.4.1.2 Script growfs

Atualmente, é necessário um script personalizado (`custom/scripts/01-fix-growfs.sh`) para expandir o sistema de arquivos de acordo com o tamanho do disco na primeira inicialização após o provisionamento. O script `01-fix-growfs.sh` contém as seguintes informações:

```
#!/bin/bash
growfs() {
  mnt="$1"
  dev="$(findmnt --fstab --target ${mnt} --evaluate --real --output SOURCE --noheadings)"
  # /dev/sda3 -> /dev/sda, /dev/nvme0n1p3 -> /dev/nvme0n1
  parent_dev="/dev/$(lsblk --nodeps -rno PKNAME "${dev}")"
  # Last number in the device name: /dev/nvme0n1p42 -> 42
```

```
  partnum="$(echo "${dev}" | sed 's/^.*[^0-9]\([0-9]\+\)$/\1/')"
  ret=0
  growpart "$parent_dev" "$partnum" || ret=$?
  [ $ret -eq 0 ] || [ $ret -eq 1 ] || exit 1
  /usr/lib/systemd/systemd-growfs "$mnt"
}
growfs /
```

Nota

Adicione seus próprios scripts personalizados para execução durante o processo de provisionamento usando a mesma abordagem. Para obter mais informações, consulte o *Capítulo 3, Clusters independentes com o Edge Image Builder*.

1.3.4.2 Criação de imagem

Depois que a estrutura de diretórios for preparada de acordo com as seções anteriores, execute o seguinte comando para criar a imagem:

```
podman run --rm --privileged -it -v $PWD:/eib \
 registry.suse.com/edge/3.3/edge-image-builder:1.2.1 \
 build --definition-file downstream-cluster-config.yaml
```

Ele cria o arquivo de imagem de saída chamado `SLE-Micro-eib-output.raw`, com base na definição descrita acima.

Depois disso, a imagem de saída deverá estar disponível em um servidor web, ou o contêiner de servidor de mídia habilitado pelo gráfico do Metal3 (*Nota*) ou um outro servidor acessível localmente. Nos exemplos a seguir, chamamos esse servidor de `imagecache.local:8080`.

Nota

Na implantação de imagens EIB em clusters downstream, é necessário também incluir a soma sha256 da imagem no objeto `Metal3MachineTemplate`. É possível gerá-la com este comando:

```
sha256sum <image_file> > <image_file>.sha256
# On this example:
sha256sum SLE-Micro-eib-output.raw > SLE-Micro-eib-output.raw.sha256
```

1.3.5 Adicionando o inventário de BareMetalHost

Para registrar servidores bare metal para implantação automatizada, é necessário criar dois recursos: um segredo com as credenciais de acesso ao BMC e um recurso BareMetalHost do Metal[3] com a definição da conexão com o BMC e outros detalhes:

```
apiVersion: v1
kind: Secret
metadata:
  name: controlplane-0-credentials
type: Opaque
data:
  username: YWRtaW4=
  password: cGFzc3dvcmQ=
---
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: controlplane-0
  labels:
    cluster-role: control-plane
spec:
  online: true
  bootMACAddress: "00:f3:65:8a:a3:b0"
  bmc:
    address: redfish-virtualmedia://192.168.125.1:8000/redfish/v1/Systems/68bd0fb6-
d124-4d17-a904-cdf33efe83ab
    disableCertificateVerification: true
    credentialsName: controlplane-0-credentials
```

Observe o seguinte:

- O nome de usuário/senha do segredo deve ser codificado com base64. Observe que ele não deve incluir novas linhas à direita (por exemplo, usar `echo -n`, não apenas `echo`!)
- É possível definir o rótulo `cluster-role` agora ou mais tarde, durante a criação do cluster. No exemplo abaixo, esperamos `control-plane` ou `worker`
- `bootMACAddress` deve ser um MAC válido correspondente à NIC de plano de controle do host

- O endereço `bmc` é a conexão com a API de gerenciamento de BMC. As seguintes opções são suportadas:
  - `redfish-virtualmedia://<ENDEREÇO IP>/redfish/v1/Systems/<ID DO SISTEMA>`: mídia virtual Redfish, por exemplo, SuperMicro
  - `idrac-virtualmedia://<ENDEREÇO IP>/redfish/v1/Systems/System.Embedded.1`: iDRAC da Dell
- Consulte os documentos da API upstream (https://github.com/metal3-io/baremetal-operator/blob/main/docs/api.md) para obter mais detalhes sobre a API BareMetalHost

1.3.5.1 Configurando IPs estáticos

O exemplo do BareMetalHost acima considera que o DHCP configura a rede do plano de controle, mas nos cenários em que a configuração manual é necessária, como no caso dos IPs estáticos, é possível definir uma configuração adicional, conforme descrito abaixo.

1.3.5.1.1 Script adicional para configuração de rede estática

Ao criar a imagem base com o Edge Image Builder, na pasta `network`, crie o arquivo `configure-network.sh` a seguir.

Esse procedimento consome os dados da unidade de configuração na primeira inicialização e configura a rede do host usando a ferramenta NM Configurator (https://github.com/suse-edge/nm-configurator).

```
#!/bin/bash

set -eux

# Attempt to statically configure a NIC in the case where we find a network_data.json
# In a configuration drive

CONFIG_DRIVE=$(blkid --label config-2 || true)
if [ -z "${CONFIG_DRIVE}" ]; then
  echo "No config-2 device found, skipping network configuration"
  exit 0
fi

mount -o ro $CONFIG_DRIVE /mnt
```

```
NETWORK_DATA_FILE="/mnt/openstack/latest/network_data.json"

if [ ! -f "${NETWORK_DATA_FILE}" ]; then
  umount /mnt
  echo "No network_data.json found, skipping network configuration"
  exit 0
fi

DESIRED_HOSTNAME=$(cat /mnt/openstack/latest/meta_data.json | tr ',{}' '\n' | grep
 '\"metal3-name\"' | sed 's/.*\"metal3-name\": \"\(.*\)\"/\1/')
echo "${DESIRED_HOSTNAME}" > /etc/hostname

mkdir -p /tmp/nmc/{desired,generated}
cp ${NETWORK_DATA_FILE} /tmp/nmc/desired/_all.yaml
umount /mnt

./nmc generate --config-dir /tmp/nmc/desired --output-dir /tmp/nmc/generated
./nmc apply --config-dir /tmp/nmc/generated
```

1.3.5.1.2 Segredo adicional com a configuração de rede do host

É possível definir um segredo adicional com os dados no formato nmstate (https://nmstate.io/) suportado pela ferramenta NM Configurator (*Capítulo 12, Rede de borda*) para cada host.

Depois disso, o segredo será referenciado no recurso `BareMetalHost` pelo campo de especificação `preprovisioningNetworkDataName`.

```
apiVersion: v1
kind: Secret
metadata:
  name: controlplane-0-networkdata
type: Opaque
stringData:
  networkData: |
    interfaces:
    - name: enp1s0
      type: ethernet
      state: up
      mac-address: "00:f3:65:8a:a3:b0"
      ipv4:
        address:
        - ip:  192.168.125.200
          prefix-length: 24
        enabled: true
        dhcp: false
    dns-resolver:
```

```
      config:
        server:
        - 192.168.125.1
    routes:
      config:
      - destination: 0.0.0.0/0
        next-hop-address: 192.168.125.1
        next-hop-interface: enp1s0
---
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: controlplane-0
  labels:
    cluster-role: control-plane
spec:
  preprovisioningNetworkDataName: controlplane-0-networkdata
# Remaining content as in previous example
```

Nota

Em algumas situações, é possível omitir o endereço MAC. Consulte a *Seção 12.5.8, "Configurações unificadas de nós"* para obter mais detalhes.

1.3.5.2 Preparação do BareMetalHost

Depois de criar o recurso BareMetalHost e os segredos associados conforme descrito acima, será acionado um fluxo de trabalho de preparação do host:

- Uma imagem de disco RAM será iniciada depois de conectar a mídia virtual ao BMC do host de destino
- O disco RAM inspeciona os detalhes do hardware e prepara o host para provisionamento (por exemplo, limpando os discos de dados anteriores)
- Ao término desse processo, os detalhes do hardware no campo `status.hardware` do BareMetalHost serão atualizados e poderão ser verificados

Esse processo pode levar bastante tempo, mas quando for concluído, você verá que o estado do BareMetalHost deve mudar para `available`:

```
% kubectl get baremetalhost
NAME             STATE       CONSUMER   ONLINE   ERROR   AGE
```

```
controlplane-0   available              true             9m44s
worker-0         available              true             9m44s
```

1.3.6 Criando clusters downstream

Agora criamos os recursos de Cluster API que definem o cluster downstream e os recursos de Machine que provisionam e, depois, iniciam os recursos do BareMetalHost para formar um cluster RKE2.

1.3.7 Implantação do plano de controle

Para implantar o plano de controle, definimos um manifesto YAML semelhante ao mostrado abaixo, que contém os seguintes recursos:

- O recurso Cluster define o nome do cluster, as redes e o tipo de provedor de plano de controle/infraestrutura (neste caso, RKE2/Metal3)
- Metal3Cluster define o endpoint do plano de controle (IP do host para nó único, endpoint LoadBalancer para vários nós; neste exemplo, foi considerado o nó único)
- RKE2ControlPlane define a versão RKE2 e a configuração adicional necessária durante a inicialização do cluster
- Metal3MachineTemplate define a imagem do sistema operacional que será aplicada aos recursos do BareMetalHost, e hostSelector define quais BareMetalHosts serão consumidos
- Metal3DataTemplate define o metaData adicional que será passado para o BareMetalHost (veja que o networkData não é suportado na solução Edge)

Nota

Para simplificar, este exemplo considera um plano de controle de nó único em que o BareMetalHost está configurado com o IP `192.168.125.200`. Para ver exemplos mais avançados de vários nós, consulte o *Capítulo 42, Provisionamento de rede direcionado totalmente automatizado*.

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: sample-cluster
```

```
  namespace: default
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - 192.168.0.0/18
    services:
      cidrBlocks:
        - 10.96.0.0/12
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: RKE2ControlPlane
    name: sample-cluster
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3Cluster
    name: sample-cluster
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3Cluster
metadata:
  name: sample-cluster
  namespace: default
spec:
  controlPlaneEndpoint:
    host: 192.168.125.200
    port: 6443
  noCloudProvider: true
---
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
metadata:
  name: sample-cluster
  namespace: default
spec:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3MachineTemplate
    name: sample-cluster-controlplane
  replicas: 1
  version: v1.32.4+rke2r1
  rolloutStrategy:
    type: "RollingUpdate"
    rollingUpdate:
      maxSurge: 0
  agentConfig:
    format: ignition
```

```
    kubelet:
      extraArgs:
        - provider-id=metal3://BAREMETALHOST_UUID
    additionalUserData:
      config: |
        variant: fcos
        version: 1.4.0
        systemd:
          units:
            - name: rke2-preinstall.service
              enabled: true
              contents: |
                [Unit]
                Description=rke2-preinstall
                Wants=network-online.target
                Before=rke2-install.service
                ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
                [Service]
                Type=oneshot
                User=root
                ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
                ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/$(jq -r .uuid /mnt/
openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
                ExecStart=/bin/sh -c "echo \"node-name: $(jq -r .name /mnt/openstack/
latest/meta_data.json)\" >> /etc/rancher/rke2/config.yaml"
                ExecStartPost=/bin/sh -c "umount /mnt"
                [Install]
                WantedBy=multi-user.target
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3MachineTemplate
metadata:
  name: sample-cluster-controlplane
  namespace: default
spec:
  template:
    spec:
      dataTemplate:
        name: sample-cluster-controlplane-template
      hostSelector:
        matchLabels:
          cluster-role: control-plane
      image:
        checksum: http://imagecache.local:8080/SLE-Micro-eib-output.raw.sha256
        checksumType: sha256
        format: raw
        url: http://imagecache.local:8080/SLE-Micro-eib-output.raw
```

```
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3DataTemplate
metadata:
  name: sample-cluster-controlplane-template
  namespace: default
spec:
  clusterName: sample-cluster
  metaData:
    objectNames:
      - key: name
        object: machine
      - key: local-hostname
        object: machine
      - key: local_hostname
        object: machine
```

Depois de adaptado ao seu ambiente, aplique o exemplo usando o `kubectl` e monitore o status do cluster por meio do `clusterctl`.

```
% kubectl apply -f rke2-control-plane.yaml

# Wait for the cluster to be provisioned
% clusterctl describe cluster sample-cluster
NAME                                                READY  SEVERITY  REASON  SINCE
 MESSAGE
Cluster/sample-cluster                              True                     22m
├─ClusterInfrastructure - Metal3Cluster/sample-cluster  True                     27m
├─ControlPlane - RKE2ControlPlane/sample-cluster        True                     22m
│ └─Machine/sample-cluster-chflc                       True                     23m
```

1.3.8 Implantação de worker/computação

De maneira semelhante à implantação do plano de controle, definimos um manifesto YAML que contém os seguintes recursos:

- MachineDeployment define o número de réplicas (hosts) e o provedor de inicialização/infraestrutura (neste caso, RKE2/Metal3)
- RKE2ConfigTemplate descreve a versão RKE2 e a configuração da primeira inicialização para inicializar o host do agente

- Metal3MachineTemplate define a imagem do sistema operacional que será aplicada aos recursos do BareMetalHost, e o seletor de host define quais BareMetalHosts serão consumidos
- Metal3DataTemplate define os metadados adicionais que serão passados para o BareMetalHost (veja que o `networkData` não é suportado atualmente)

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineDeployment
metadata:
  labels:
    cluster.x-k8s.io/cluster-name: sample-cluster
  name: sample-cluster
  namespace: default
spec:
  clusterName: sample-cluster
  replicas: 1
  selector:
    matchLabels:
      cluster.x-k8s.io/cluster-name: sample-cluster
  template:
    metadata:
      labels:
        cluster.x-k8s.io/cluster-name: sample-cluster
    spec:
      bootstrap:
        configRef:
          apiVersion: bootstrap.cluster.x-k8s.io/v1alpha1
          kind: RKE2ConfigTemplate
          name: sample-cluster-workers
      clusterName: sample-cluster
      infrastructureRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
        kind: Metal3MachineTemplate
        name: sample-cluster-workers
      nodeDrainTimeout: 0s
      version: v1.32.4+rke2r1
---
apiVersion: bootstrap.cluster.x-k8s.io/v1alpha1
kind: RKE2ConfigTemplate
metadata:
  name: sample-cluster-workers
  namespace: default
spec:
  template:
    spec:
```

```
      agentConfig:
        format: ignition
        version: v1.32.4+rke2r1
        kubelet:
          extraArgs:
            - provider-id=metal3://BAREMETALHOST_UUID
        additionalUserData:
          config: |
            variant: fcos
            version: 1.4.0
            systemd:
              units:
                - name: rke2-preinstall.service
                  enabled: true
                  contents: |
                    [Unit]
                    Description=rke2-preinstall
                    Wants=network-online.target
                    Before=rke2-install.service
                    ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
                    [Service]
                    Type=oneshot
                    User=root
                    ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
                    ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/$(jq -r .uuid /
mnt/openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
                    ExecStart=/bin/sh -c "echo \"node-name: $(jq -r .name /mnt/openstack/
latest/meta_data.json)\" >> /etc/rancher/rke2/config.yaml"
                    ExecStartPost=/bin/sh -c "umount /mnt"
                    [Install]
                    WantedBy=multi-user.target
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3MachineTemplate
metadata:
  name: sample-cluster-workers
  namespace: default
spec:
  template:
    spec:
      dataTemplate:
        name: sample-cluster-workers-template
      hostSelector:
        matchLabels:
          cluster-role: worker
      image:
        checksum: http://imagecache.local:8080/SLE-Micro-eib-output.raw.sha256
```

```
        checksumType: sha256
        format: raw
        url: http://imagecache.local:8080/SLE-Micro-eib-output.raw
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3DataTemplate
metadata:
  name: sample-cluster-workers-template
  namespace: default
spec:
  clusterName: sample-cluster
  metaData:
    objectNames:
      - key: name
        object: machine
      - key: local-hostname
        object: machine
      - key: local_hostname
        object: machine
```

Quando o exemplo acima for copiado e adaptado ao seu ambiente, ele poderá ser aplicado usando o `kubectl` e, depois, o status do cluster poderá ser monitorado com o `clusterctl`

```
% kubectl apply -f rke2-agent.yaml

# Wait for the worker nodes to be provisioned
% clusterctl describe cluster sample-cluster
NAME                                                READY  SEVERITY  REASON  SINCE
 MESSAGE
Cluster/sample-cluster                              True                     25m
├─ClusterInfrastructure - Metal3Cluster/sample-cluster  True                     30m
├─ControlPlane - RKE2ControlPlane/sample-cluster    True                     25m
│ └─Machine/sample-cluster-chflc                    True                     27m
└─Workers
  └─MachineDeployment/sample-cluster                True                     22m
    └─Machine/sample-cluster-56df5b4499-zfljj       True                     23m
```

1.3.9 Desprovisionamento do cluster

É possível desprovisionar o cluster downstream excluindo os recursos que foram aplicados nas etapas de criação acima:

```
% kubectl delete -f rke2-agent.yaml
% kubectl delete -f rke2-control-plane.yaml
```

Esse procedimento aciona o desprovisionamento dos recursos do BareMetalHost, o que pode levar bastante tempo. Depois disso, eles deverão voltar para o estado disponível:

```
% kubectl get bmh
NAME             STATE            CONSUMER                              ONLINE   ERROR
 AGE
controlplane-0   deprovisioning   sample-cluster-controlplane-vlrt6   false
 10m
worker-0         deprovisioning   sample-cluster-workers-785x5        false
 10m

...

% kubectl get bmh
NAME             STATE       CONSUMER   ONLINE   ERROR   AGE
controlplane-0   available              false            15m
worker-0         available              false            15m
```

1.4 Problemas conhecidos

- O controlador de gerenciamento de endereços IP (https://github.com/metal3-io/ip-address-manager) upstream não é suportado no momento porque ainda não é compatível com a nossa seleção de ferramentas de configuração de rede e conjunto de ferramentas de primeira inicialização no SLEMicro.
- De modo semelhante, os recursos IPAM e os campos networkData de Metal3DataTemplate não são suportados no momento.
- Apenas há suporte para implantação por redfish-virtualmedia.

1.5 Alterações planejadas

- Habilitar suporte de recursos e configuração do IPAM pelos campos networkData

1.6 Recursos adicionais

A documentação do SUSE Edge for Telco (*Capítulo 37, SUSE Edge for Telco*) tem exemplos de uso mais avançado do Metal[3] para casos de uso de telecomunicações.

1.6.1 Configuração de nó único

Para ambientes de teste/PoC em que o cluster de gerenciamento é um nó único, é possível evitar o requisito de IP flutuante adicional gerenciado por MetalLB.

Nesse modo, o endpoint para as APIs do cluster de gerenciamento é o IP do cluster de gerenciamento, portanto, ele deve ser reservado ao usar DHCP ou configurado estaticamente para garantir que não seja alterado. Mencionado abaixo como `<MANAGEMENT_CLUSTER_IP>`.

Para possibilitar esse cenário, os valores do gráfico do Metal[3] necessários são estes:

```
global:
  ironicIP: <MANAGEMENT_CLUSTER_IP>
metal3-ironic:
  service:
    type: NodePort
```

1.6.2 Desabilitando TLS para anexo de ISO de mídia virtual

Alguns fornecedores de servidor verificam a conexão SSL ao anexar imagens ISO de mídia virtual ao BMC, o que pode causar um problema porque os certificados gerados para a implantação do Metal[3] são autoassinados. Para solucionar esse problema, desabilite o TLS apenas para anexo de disco de mídia virtual com os valores do gráfico do Metal[3] a seguir:

```
global:
  enable_vmedia_tls: false
```

Uma solução alternativa é configurar os BMCs com o certificado de CA. Nesse caso, você pode ler os certificados do cluster usando o `kubectl`:

```
kubectl get secret -n metal3-system ironic-vmedia-cert -o yaml
```

Depois disso, o certificado poderá ser configurado no console de BMC do servidor, embora o processo para isso seja específico do fornecedor (e pode não ser possível para todos os fornecedores e, consequentemente, exigir o uso do sinalizador `enable_vmedia_tls`).

1.6.3 Configuração do armazenamento

Para ambientes de teste/PoC em que o cluster de gerenciamento é um nó único, não há necessidade de armazenamento persistente; mas em casos de uso de produção, a recomendação é instalar o SUSE Storage (Longhorn) no cluster de gerenciamento para que as imagens relacionadas ao Metal[3] persistam durante a reinicialização/reprogramação de um pod.

Para habilitar o armazenamento persistente, os valores do gráfico do Metal[3] necessários são estes:

```
metal3-ironic:
  persistence:
    ironic:
      size: "5Gi"
```

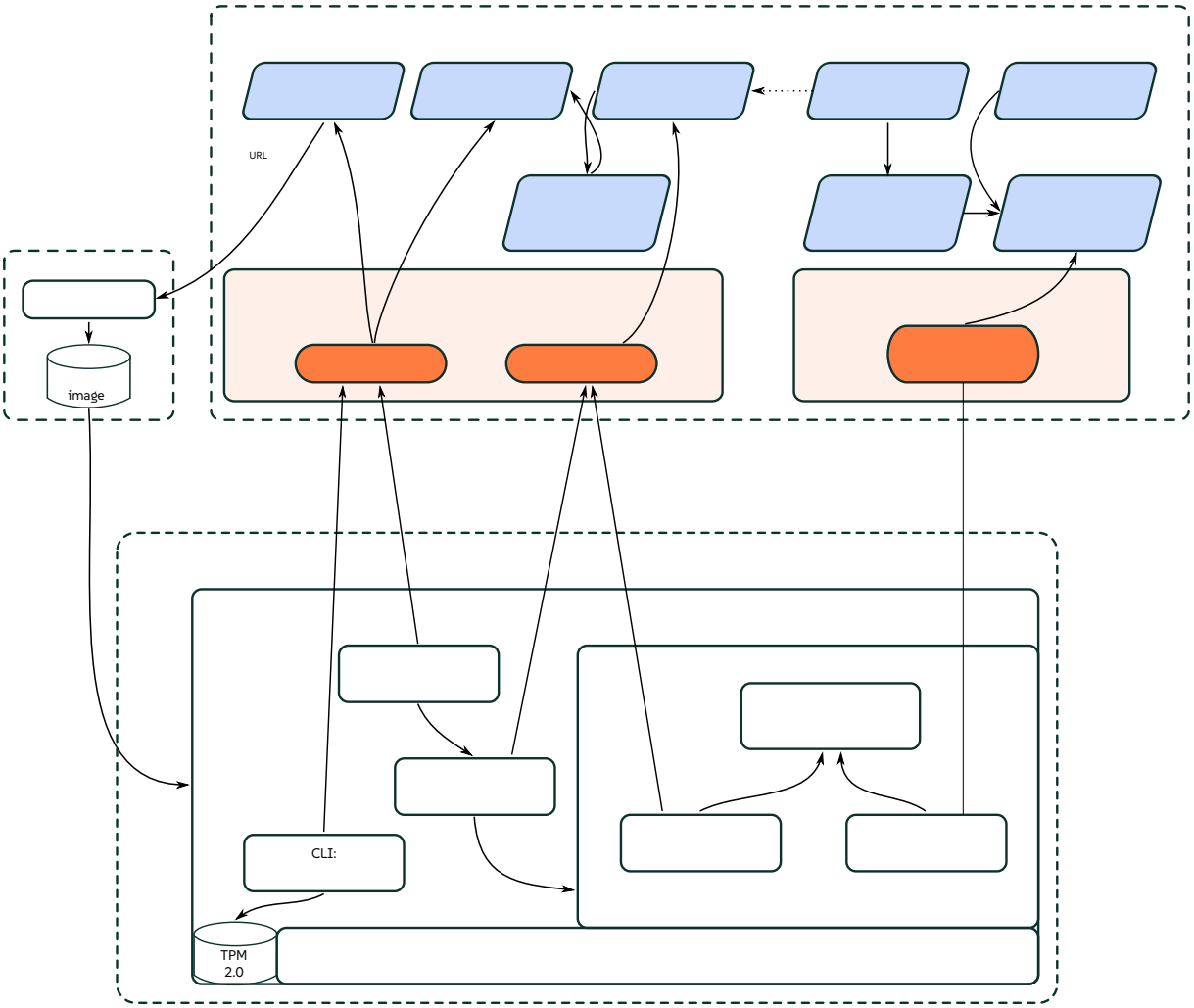
A documentação para cluster de gerenciamento do SUSE Edge for Telco (*Capítulo 40, Configurando o cluster de gerenciamento*) tem mais detalhes sobre como configurar um cluster de gerenciamento com armazenamento persistente.

2 Integração remota de host com o Elemental

Esta seção apresenta a solução de "provisionamento de rede 'phone home'" como parte do SUSE Edge, em que usamos o Elemental para auxiliar na integração de nós. O Elemental é uma pilha de software que permite o registro remoto de hosts e o gerenciamento de sistema operacional nativo de nuvem e totalmente centralizado com o Kubernetes. Na pilha do SUSE Edge, usamos o recurso de registro do Elemental para fazer a integração remota de hosts no Rancher, o que permite integrar os hosts a uma plataforma de gerenciamento centralizada e, nela, implantar e gerenciar os clusters Kubernetes junto com os componentes em camadas, os aplicativos e o ciclo de vida deles, tudo de um único local.

Essa abordagem pode ser útil em cenários em que os dispositivos que você deseja controlar não estão na mesma rede que o cluster de gerenciamento, ou não têm uma integração por controlador de gerenciamento fora da banda que permita um controle mais direto, e quando você inicializa vários sistemas "desconhecidos" na borda e precisa integrá-los e gerenciá-los com segurança e em escala. Esse cenário é comum nos casos de uso de varejo, IoT industrial ou em outros ambientes de baixo controle da rede em que os dispositivos são instalados.

2.1 Arquitetura de alto nível



2.2 Recursos necessários

Veja a seguir uma descrição dos requisitos mínimos de sistema e de ambiente para execução por meio desta inicilização rápida:

- Um host para o cluster de gerenciamento centralizado (o mesmo que hospeda o Rancher e o Elemental):
    - Espaço em disco mínimo de 8 GB RAM e 20 GB para desenvolvimento ou teste (consulte aqui (https://ranchermanager.docs.rancher.com/v2.11/getting-started/installation-and-upgrade/installation-requirements#hardware-requirements) para uso em produção)
- O provisionamento de um nó de destino, ou seja, o dispositivo de borda (é possível usar uma máquina virtual para fins de demonstração ou teste)
    - Mínimo de 4GB de RAM, 2 núcleos de CPU e disco de 20 GB
- Um nome de host "resolvable" para o cluster de gerenciamento ou um endereço IP estático para usar com um serviço do tipo sslip.io
- Um host para montar a mídia de instalação usando o Edge Image Builder
    - SLES 15 SP6, openSUSE Leap 15.6 ou outro sistema operacional compatível em execução com suporte ao Podman
    - Com o Kubectl (https://kubernetes.io/docs/reference/kubectl/kubectl/), o Podman (https://podman.io) e o Helm (https://helm.sh) instalados
- Uma unidade flash USB de origem para inicialização (em caso de hardware físico)
- Uma cópia baixada da imagem ISO mais recente do SelfInstall do SUSE Linux Micro 6.1 disponível aqui (https://www.suse.com/download/sle-micro/).

Nota

Os dados existentes nas máquinas de destino serão substituídos como parte do processo. Faça um backup dos dados em dispositivos de armazenamento USB e discos conectados aos nós de implantação de destino.

Este guia foi criado com um droplet da DigitalOcean para hospedar o cluster upstream e com um Intel NUC como dispositivo downstream. Para montar a mídia de instalação, o SUSE Linux Enterprise Server foi usado.

2.3 Criar o cluster de inicialização

Para começar, crie um cluster capaz de hospedar o Rancher e o Elemental. Esse cluster precisa ser roteável na rede à qual os nós downstream estão conectados.

2.3.1 Criar o cluster Kubernetes

Se você usa um hiperescalador (como Azure, AWS ou Google Cloud), a maneira mais fácil de configurar um cluster é usar as ferramentas incorporadas dele. Para manter este guia conciso, não detalhamos o processo de cada uma dessas opções.

Se você está instalando em um serviço de hospedagem bare metal, ou algum outro, e precisa também providenciar a própria distribuição Kubernetes, recomendamos usar o RKE2 (https://docs.rke2.io/install/quickstart).

2.3.2 Configurar o DNS

Antes de continuar, você precisa configurar o acesso ao cluster. Assim como na configuração do próprio cluster, a configuração do DNS será diferente dependendo do local onde ele está hospedado.

Dica

Se você não deseja configurar registros DNS (por exemplo, quando se trata apenas de um servidor de teste efêmero), uma alternativa é usar um serviço como o sslip.io (https://sslip.io). Com ele, você resolve qualquer endereço IP com `<endereço>.sslip.io`.

2.4 Instalar o Rancher

Para instalar o Rancher, você precisa acessar a API Kubernetes do cluster que acabou de criar. Esse procedimento muda dependendo da distribuição Kubernetes que é usada.

Para o RKE2, o arquivo kubeconfig será gravado em `/etc/rancher/rke2/rke2.yaml`. Salve-o como `~/.kube/config` no sistema local. Talvez você tenha que editar o arquivo para incluir o endereço IP ou o nome de host roteável externamente correto.

Instale o Rancher facilmente com os comandos apresentados na documentação do Rancher (https://ranchermanager.docs.rancher.com/v2.11/getting-started/installation-and-upgrade/install-upgrade-on-a-kubernetes-cluster):

Instale o cert-manager (https://cert-manager.io):

```
helm repo add jetstack https://charts.jetstack.io
helm repo update
helm install cert-manager jetstack/cert-manager \
 --namespace cert-manager \
 --create-namespace \
 --set crds.enabled=true
```

Em seguida, instale o próprio Rancher:

```
helm repo add rancher-prime https://charts.rancher.com/server-charts/prime
helm repo update
helm install rancher rancher-prime/rancher \
  --namespace cattle-system \
  --create-namespace \
  --set hostname=<DNS or sslip from above> \
  --set replicas=1 \
  --set bootstrapPassword=<PASSWORD_FOR_RANCHER_ADMIN> \
  --version 2.11.2
```

Nota

Se a finalidade do sistema é de produção, use o cert-manager para configurar um certificado real (como aquele da Let's Encrypt).

Procure o nome de host que você configurou e faça login no Rancher com a `bootstrapPassword` usada. Você será guiado por um curto processo de configuração.

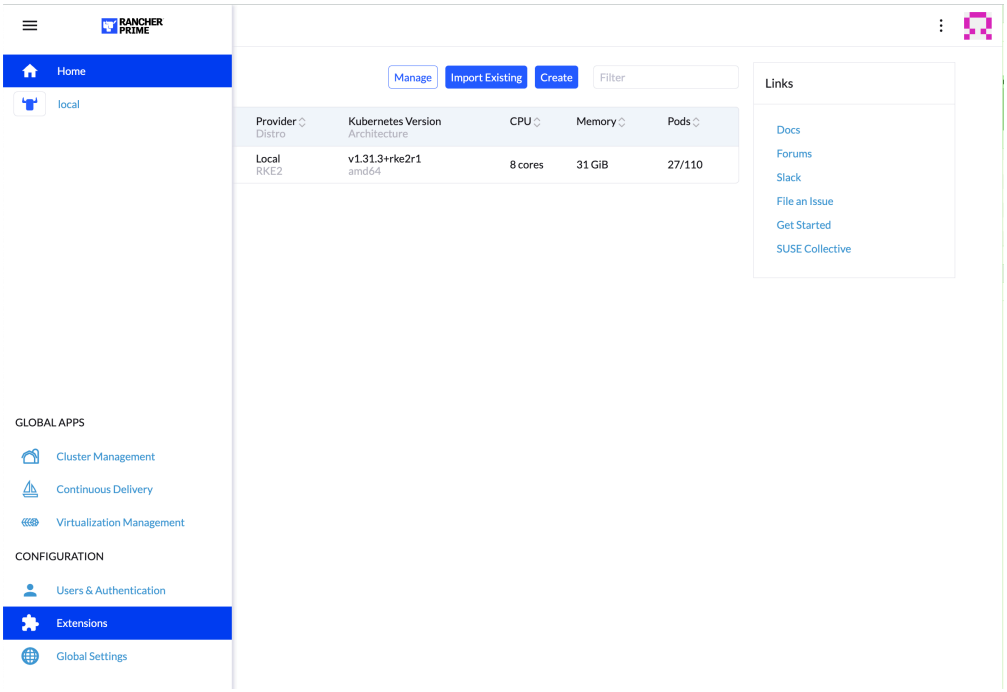
2.5 Instalar o Elemental

Com o Rancher instalado, agora você pode instalar o operador Elemental e as CRDs necessárias. O gráfico Helm do Elemental foi publicado como um artefato OCI, portanto, a instalação é um pouco mais simples do que a dos outros gráficos. É possível instalá-lo do mesmo shell usado para instalar o Rancher ou no navegador dentro do shell do Rancher.

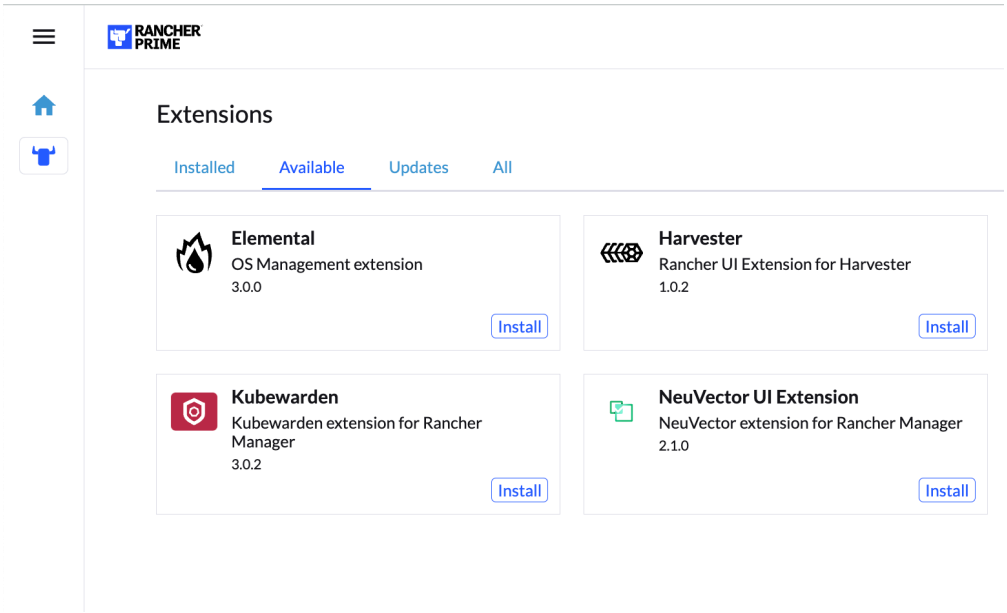
```
helm install --create-namespace -n cattle-elemental-system \
 elemental-operator-crds \
 oci://registry.suse.com/rancher/elemental-operator-crds-chart \
 --version 1.6.8

helm install -n cattle-elemental-system \
 elemental-operator \
 oci://registry.suse.com/rancher/elemental-operator-chart \
 --version 1.6.8
```

2.5.1 Instalar a extensão de IU do Elemental (opcional)

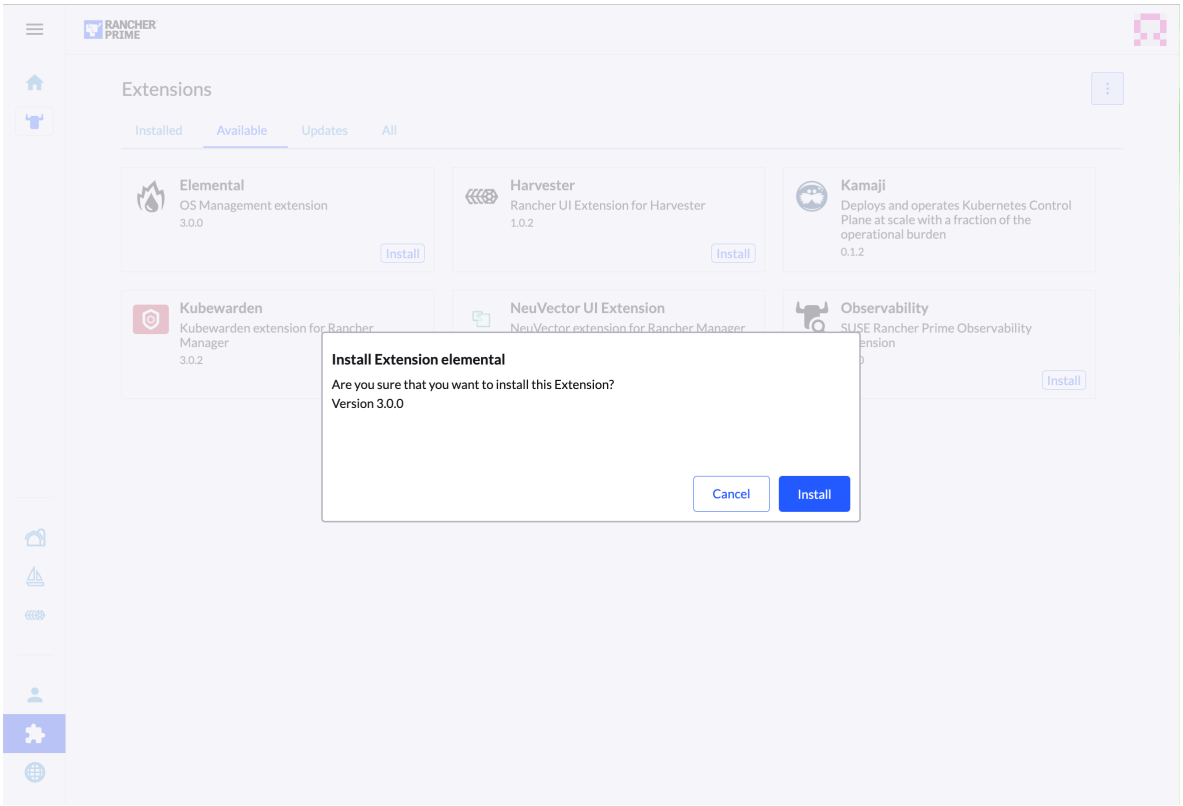
1. Para usar a IU do Elemental, faça login na instância do Rancher e clique no menu de três linhas na parte superior esquerda:



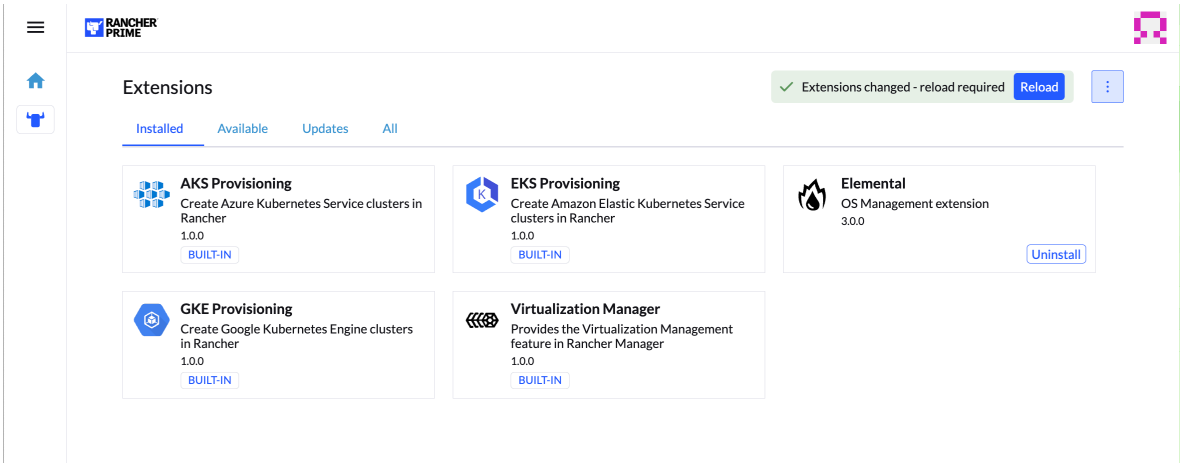
2. Na guia "Available" (Disponível) nesta página, clique em "Install" (Instalar) no cartão Elemental:



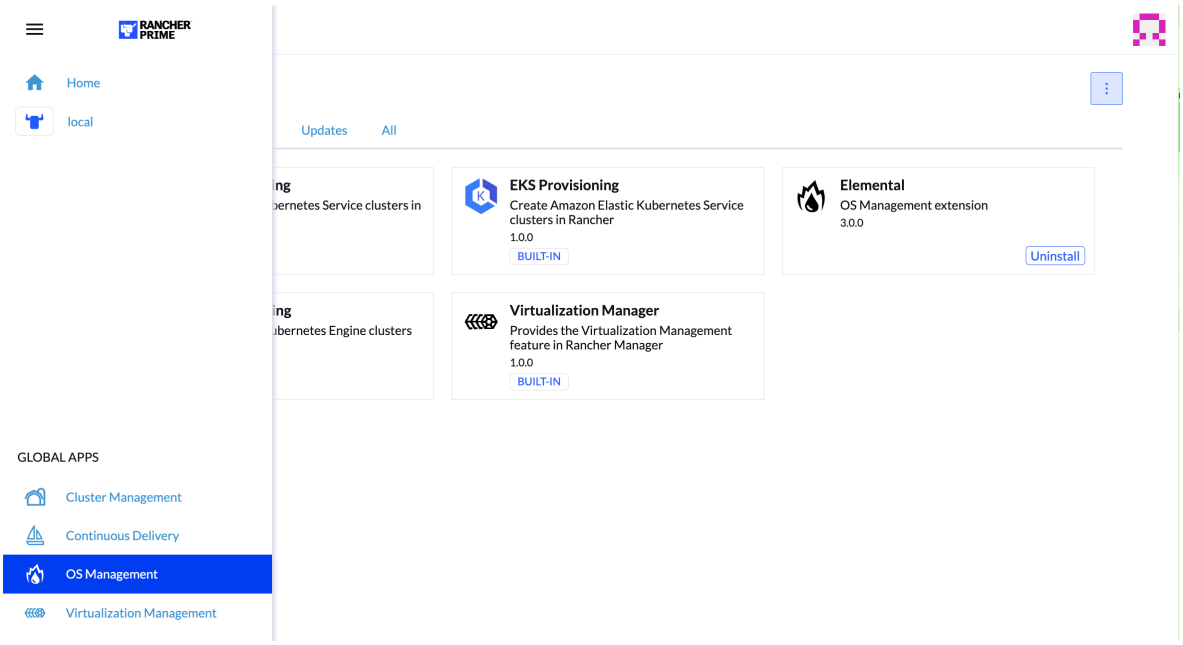
3. Confirme que você deseja instalar a extensão:



4. Após a instalação, será solicitado que você recarregue a página.



5. Após o recarregamento, acesse a extensão do Elemental pelo app global "OS Management".



2.6 Configurar o Elemental

Para simplificar, recomendamos definir a variável `$ELEM` como o caminho completo do local desejado para o diretório de configuração:

```
export ELEM=$HOME/elemental
mkdir -p $ELEM
```

Para que as máquinas sejam registradas no Elemental, precisamos criar um objeto `MachineRegistration` no namespace `fleet-default`.

Vamos criar uma versão básica desse objeto:

```
cat << EOF > $ELEM/registration.yaml
apiVersion: elemental.cattle.io/v1beta1
kind: MachineRegistration
metadata:
  name: ele-quickstart-nodes
  namespace: fleet-default
spec:
  machineName: "\${System Information/Manufacturer}-\${System Information/UUID}"
  machineInventoryLabels:
    manufacturer: "\${System Information/Manufacturer}"
    productName: "\${System Information/Product Name}"
EOF
```

```
kubectl apply -f $ELEM/registration.yaml
```

Nota

O comando `cat` insere um caractere de escape de barra (\) para cada `$` para que o Bash não os utilize como gabarito. Remova as barras se estiver copiando manualmente.

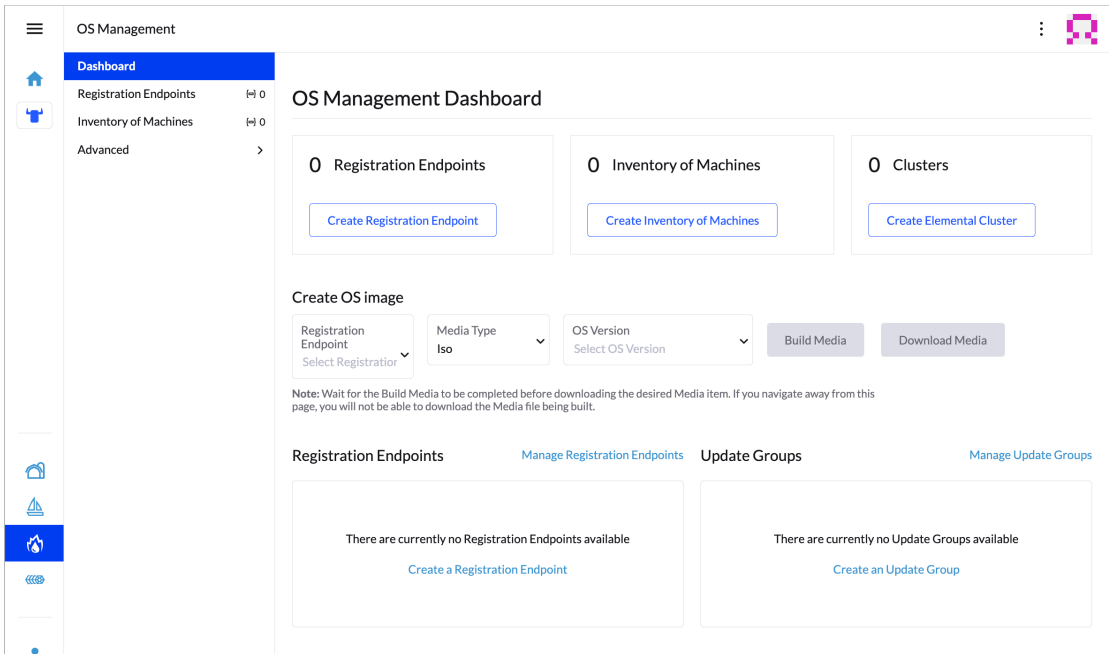
Depois que o objeto é criado, encontre e observe o endpoint que foi atribuído:

```
REGISURL=$(kubectl get machineregistration ele-quickstart-nodes -n fleet-default -o
 jsonpath='{.status.registrationURL}')
```

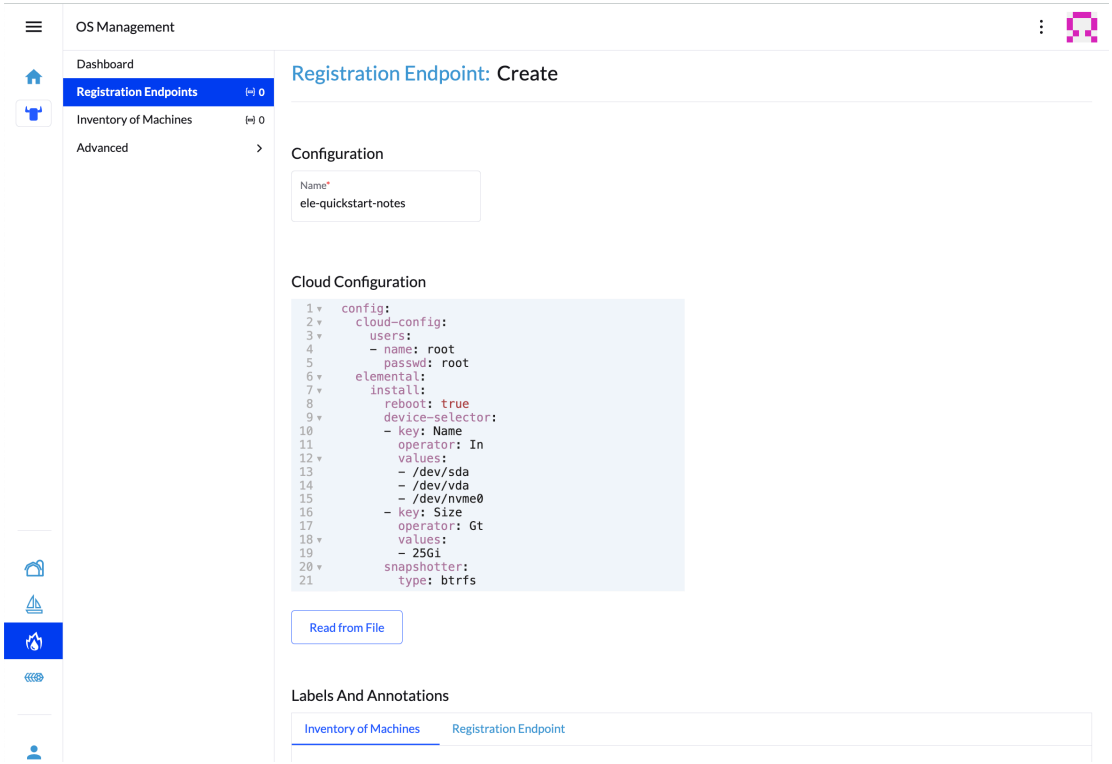
Você também pode fazer isso pela IU.

Extensão de IU

1. Na extensão OS Management, clique em "Create Registration Endpoint" (Criar endpoint de registro):



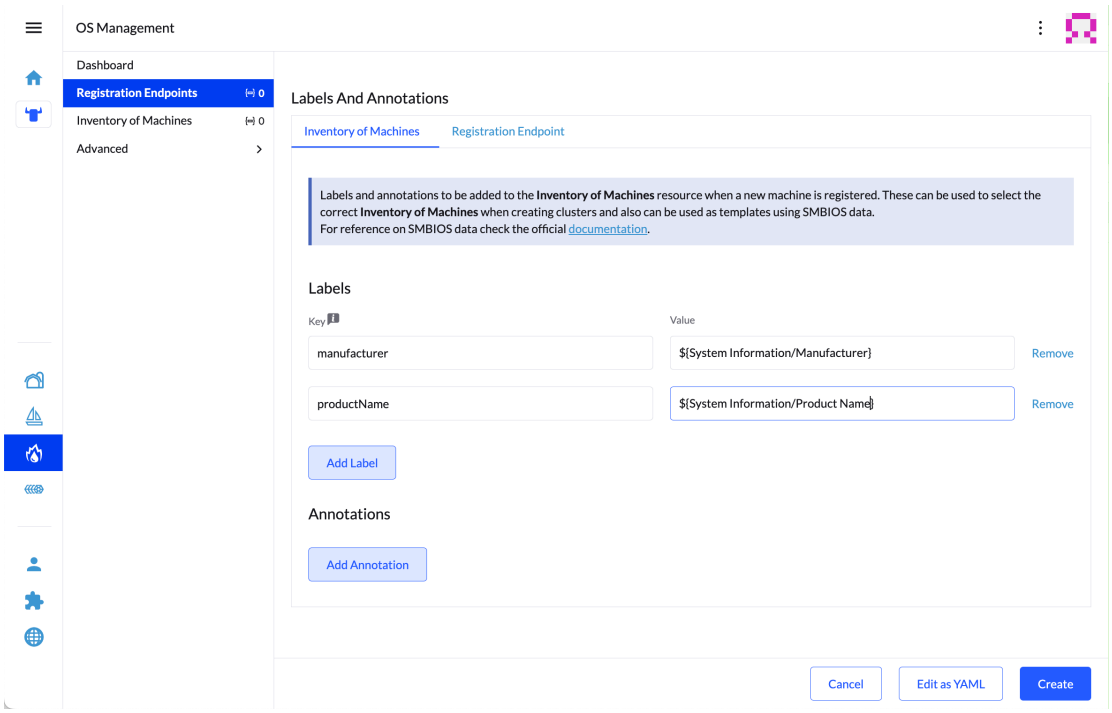
2. Dê um nome para esta configuração.



Nota

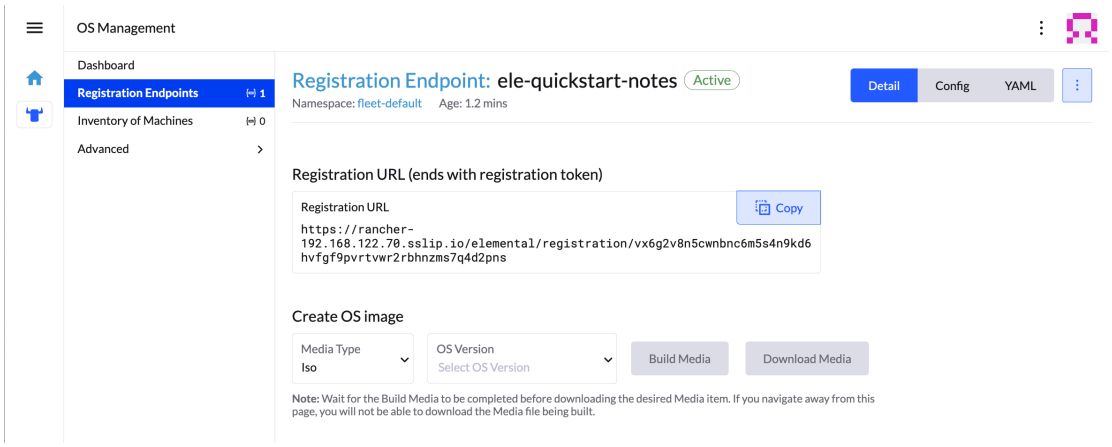
Você pode ignorar o campo Cloud Configuration (Configuração de nuvem) já que os dados nele serão substituídos ao executar as etapas a seguir com o Edge Image Builder.

3. Em seguida, role para baixo e clique em "Add Label" (Adicionar rótulo) para cada rótulo que deseja incluir no recurso criado quando a máquina é registrada. Isso é útil para distinguir as máquinas.



4. Clique em "Create" (Criar) para salvar a configuração.

5. Após a criação do registro, você deverá ver o URL dele listado e poderá clicar em "Copy" (Copiar) para copiar o endereço:



Dica

Se você clicou fora dessa tela, pode clicar em "Registration Endpoints" (Endpoints de registro) no menu esquerdo e, depois, no nome do endpoint que acabou de criar.

Esse URL será usado na próxima etapa.

2.7 Criar a imagem

A versão atual do Elemental conta com um método para criar a própria mídia de instalação. No SUSE Edge 3.3.1, fazemos isso com o Kiwi e o Edge Image Builder, portanto, o sistema resultante é desenvolvido com o SUSE Linux Micro (https://www.suse.com/products/micro/) como o sistema operacional de base.

Dica

Para saber mais detalhes sobre o Kiwi, siga o processo do construtor de imagens do Kiwi (*Capítulo 28, Criando imagens atualizadas do SUSE Linux Micro com o Kiwi*) para primeiro criar novas imagens e, para o Edge Image Builder, consulte o Guia de Introdução do Edge Image Builder (*Capítulo 3, Clusters independentes com o Edge Image Builder*) e também a documentação do componente (*Capítulo 11, Edge Image Builder*).

Em um sistema Linux com o Podman instalado, crie os diretórios e insira a imagem base criada pelo Kiwi:

```
mkdir -p $ELEM/eib_quickstart/base-images
cp /path/to/{micro-base-image-iso} $ELEM/eib_quickstart/base-images/
mkdir -p $ELEM/eib_quickstart/elemental
```

```
curl $REGISURL -o $ELEM/eib_quickstart/elemental/elemental_config.yaml
```

```
cat << EOF > $ELEM/eib_quickstart/eib-config.yaml
apiVersion: 1.2
image:
    imageType: iso
    arch: x86_64
    baseImage: SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso
    outputImageName: elemental-image.iso
operatingSystem:
  time:
    timezone: Europe/London
    ntp:
      forceWait: true
      pools:
```

```
        - 2.suse.pool.ntp.org
      servers:
        - 10.0.0.1
        - 10.0.0.2
  isoConfiguration:
    installDevice: /dev/vda
  users:
    - username: root
      encryptedPassword: \$6\$jHugJNNd3HElGsUZ\
$eodjVe4te5ps44SVcWshdfWizrP.xAyd71CVEXazBJ/.v799/WRCBXxfYmunlBO2yp1hm/zb4r8EmnrrNCF.P/
  packages:
    sccRegistrationCode: XXX
EOF
```

Nota

- A seção `time` é opcional, mas sua configuração é altamente recomendada para evitar possíveis problemas com certificados e divergência de relógio. O valor usado neste exemplo é somente para fins ilustrativos. Ajuste-o de acordo com seus requisitos específicos.
- A senha não codificada é `eib`.
- O `sccRegistrationCode` é necessário para fazer download e instalar os RPMs exigidos das fontes oficiais (se preferir, faça sideload manual dos RPMs `elemental-register` e o `elemental-system-agent`).
- O comando `cat` insere um caractere de escape de barra (`\`) para cada `$` para que o Bash não os utilize como gabarito. Remova as barras se estiver copiando manualmente.
- O dispositivo de instalação será apagado durante a instalação.

```
podman run --privileged --rm -it -v $ELEM/eib_quickstart/:/eib \
 registry.suse.com/edge/3.3/edge-image-builder:1.2.1 \
 build --definition-file eib-config.yaml
```

Se você está inicializando um dispositivo físico, precisa gravar a imagem em uma unidade flash USB, o que pode ser feito com o comando:

```
sudo dd if=/eib_quickstart/elemental-image.iso of=/dev/<PATH_TO_DISK_DEVICE>
 status=progress
```

2.8 Inicializar os nós downstream

Agora que já criamos a mídia de instalação, podemos inicializar os nós downstream com ela.

Para cada um dos sistemas que você quer controlar usando o Elemental, adicione a mídia de instalação e inicialize o dispositivo. Após a instalação, ele será reinicializado e se registrará.

Se você usa a extensão de IU, deve ver o nó listado em "Inventory of Machines" (Inventário de máquinas).

Nota

Não remova o meio de instalação antes de ver o prompt de login. Durante a primeira inicialização, os arquivos ainda são acessados no dispositivo USB.

2.9 Criar clusters downstream

Precisamos criar dois objetos ao provisionar um novo cluster usando o Elemental.

Linux

O primeiro é o `MachineInventorySelectorTemplate`. Esse objeto permite especificar um mapeamento entre os clusters e as máquinas no inventário.

1. Crie um seletor que corresponda qualquer máquina no inventário com um rótulo:

```
cat << EOF > $ELEM/selector.yaml
apiVersion: elemental.cattle.io/v1beta1
kind: MachineInventorySelectorTemplate
metadata:
  name: location-123-selector
  namespace: fleet-default
spec:
  template:
    spec:
      selector:
        matchLabels:
          locationID: '123'
EOF
```

2. Aplique o recurso ao cluster:

```
kubectl apply -f $ELEM/selector.yaml
```

3. Obtenha o nome da máquina e adicione o rótulo correspondente:

```
MACHINENAME=$(kubectl get MachineInventory -n fleet-default | awk 'NR>1 {print
 $1}')

kubectl label MachineInventory -n fleet-default \
 $MACHINENAME locationID=123
```

4. Crie um recurso de cluster K3s simples de nó único e aplique-o ao cluster:

```
cat << EOF > $ELEM/cluster.yaml
apiVersion: provisioning.cattle.io/v1
kind: Cluster
metadata:
  name: location-123
  namespace: fleet-default
spec:
  kubernetesVersion: v1.32.4+k3s1
  rkeConfig:
    machinePools:
      - name: pool1
        quantity: 1
        etcdRole: true
        controlPlaneRole: true
        workerRole: true
        machineConfigRef:
          kind: MachineInventorySelectorTemplate
          name: location-123-selector
          apiVersion: elemental.cattle.io/v1beta1
EOF

kubectl apply -f $ELEM/cluster.yaml
```

Extensão de IU

A extensão de IU permite o uso de alguns atalhos. Observe que o gerenciamento de vários locais pode envolver um trabalho manual excessivo.

1. Como já foi feito, abra o menu esquerdo de três linhas e selecione "OS Management". Você será levado de volta à tela principal para gerenciar seus sistemas Elemental.

2. Na barra lateral esquerda, clique em "Inventory of Machines" (Inventário de máquinas). Essa ação abre o inventário das máquinas que foram registradas.

3. Para criar um cluster dessas máquinas, selecione os sistemas desejados, clique na lista suspensa "Actions" (Ações) e em "Create Elemental Cluster" (Criar cluster Elemental). Isso abre a caixa de diálogo Cluster Creation (Criação de cluster) e cria, ao mesmo tempo, um MachineSelectorTemplate para usar em segundo plano.

4. Nessa tela, configure o cluster que deseja criar. Neste início rápido, o K3s v1.30.5+k3s1 foi selecionado, e as demais opções foram mantidas como estavam.

Dica

Pode ser necessário rolar para abaixo para ver mais opções.

Após a criação dos objetos, você deverá ver um novo cluster Kubernetes dar arranque (partida) usando o nó com o qual você acabou de instalar.

2.10 Redefinição de nó (opcional)

O SUSE Rancher Elemental permite executar uma "redefinição de nó", que pode ser acionada quando um cluster inteiro é excluído do Rancher, um único nó é excluído do cluster ou um nó é excluído manualmente do inventário de máquinas. Esse recurso é útil para redefinir e limpar recursos órfãos e recuperar automaticamente o nó limpo no inventário de máquinas para que possa ser reutilizado. Isso não está habilitado por padrão e, portanto, um sistema que foi removido não será limpo (ou seja, os dados não serão removidos, e os recursos do cluster Kubernetes continuarão operando nos clusters downstream), será necessária uma intervenção manual para limpar os dados e registrar a máquina novamente no Rancher pelo Elemental.

Para habilitar essa funcionalidade por padrão, você precisa garantir que o `MachineRegistration` a habilite explicitamente adicionando `config.elemental.reset.enabled: true`, por exemplo:

```
config:
  elemental:
    registration:
      auth: tpm
    reset:
      enabled: true
```

Na sequência, todos os sistemas registrados com esse `MachineRegistration` receberão automaticamente a anotação `elemental.cattle.io/resettable: 'true'` na respectiva configuração. Para fazer isso manualmente em nós individuais, por exemplo, porque obteve um `MachineInventory` existente que não tem essa anotação ou já implantou os nós, você pode modificar o `MachineInventory` e adicionar a configuração `resettable`, por exemplo:

```
apiVersion: elemental.cattle.io/v1beta1
kind: MachineInventory
metadata:
  annotations:
    elemental.cattle.io/os.unmanaged: 'true'
    elemental.cattle.io/resettable: 'true'
```

No SUSE Edge 3.1, o operador Elemental insere um marcador no sistema operacional que aciona o processo de limpeza automaticamente; ele interrompe todos os serviços do Kubernetes, remove todos os dados persistentes, desinstala todos os serviços do Kubernetes, limpa os diretórios restantes do Kubernetes/Rancher e força o novo registro no Rancher por meio da configuração original `MachineRegistration` do Elemental. Isso é feito automaticamente, não há necessidade de intervenção manual. O script chamado está disponível em `/opt/edge/elemental_node_cleanup.sh` e é acionado pelo `systemd.path` depois de inserir o marcador, portanto, sua execução é imediata.

Atenção

Com o uso do `resettable`, a funcionalidade assume que o comportamento desejado ao remover um nó/cluster do Rancher é limpar os dados e forçar um novo registro. A perda de dados é certa nessa situação, portanto, use essa funcionalidade apenas se tiver certeza de que deseja executar a redefinição automática.

2.11 Próximas etapas

Veja alguns recursos recomendados para pesquisa depois de usar este guia:

- Automação de ponta a ponta no *Capítulo 8, Fleet*
- Mais opções de configuração de rede no *Capítulo 12, Rede de borda*

3 Clusters independentes com o Edge Image Builder

O Edge Image Builder (EIB) é uma ferramenta que simplifica o processo de geração de imagens de disco personalizadas e prontas para inicialização (CRB) para inicializar máquinas, mesmo em cenários totalmente air-gapped. O EIB é usado para criar imagens de implantação para uso em todas as três áreas de implantação do SUSE Edge, já que é flexível o suficiente para oferecer desde as menores personalizações, como adicionar um usuário ou definir o fuso horário, até uma imagem amplamente configurada que comporta, por exemplo, configurações de redes complexas, implantações de clusters Kubernetes de vários nós, implantações de cargas de trabalho de clientes e registros na plataforma de gerenciamento centralizado pelo Rancher/ Elemental e SUSE Multi-Linux Manager. O EIB é executado como uma imagem de contêiner, o que o torna incrivelmente portátil entre as plataformas e garante que todas as dependências necessárias sejam autossuficientes, exercendo um impacto muito mínimo sobre os pacotes instalados do sistema usado para operar a ferramenta.

Nota

Nos cenários de vários nós, o EIB implanta automaticamente o MetalLB e o Endpoint Copier Operator para que os hosts provisionados que usam a mesma imagem criada ingressem em um cluster Kubernetes de maneira automática.

Para obter mais informações, leia a introdução do Edge Image Builder (*Capítulo 11, Edge Image Builder*).

Atenção

O Edge Image Builder 1.2.1 suporta a personalização de imagens no SUSE Linux Micro 6.1. As versões mais antigas, como o SUSE Linux Enterprise Micro 5.5 ou 6.0, não são compatíveis.

3.1 Pré-requisitos

- Uma máquina host de build AMD64/Intel 64 (física ou virtual) com o SLES 15 SP6.
- O mecanismo de contêiner Podman
- Uma imagem ISO SelfInstall do SUSE Linux Micro 6.1 criada pelo procedimento do construtor Kiwi (*Capítulo 28, Criando imagens atualizadas do SUSE Linux Micro com o Kiwi*)

Nota

Para fins que não sejam de produção, é possível usar o openSUSE Leap 15.6 ou o openSUSE Tumbleweed como máquina host de build. Outros sistemas operacionais podem funcionar, desde que um tempo de execução do contêiner compatível esteja disponível.

3.1.1 Obtendo a imagem do EIB

A imagem de contêiner do EIB é publicamente disponível e pode ser baixada do registro do SUSE Edge executando o seguinte comando no host de build da imagem:

```
podman pull registry.suse.com/edge/3.3/edge-image-builder:1.2.1
```

3.2 Criando o diretório de configuração de imagem

Como o EIB é executado dentro de um contêiner, precisamos montar um diretório de configuração do host, o que permite especificar a configuração desejada e, durante o processo de criação, o EIB tem o acesso aos arquivos de entrada necessários e artefatos auxiliares. Esse diretório deve seguir uma estrutura específica. Vamos criá-lo imaginando que ele existe em seu diretório pessoal com o nome "eib":

```
export CONFIG_DIR=$HOME/eib
mkdir -p $CONFIG_DIR/base-images
```

Na etapa anterior, criamos um diretório "base-images" que hospedará a imagem de entrada do SUSE Linux Micro 6.1. Vamos garantir que a imagem seja copiada para o diretório de configuração:

```
cp /path/to/SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso $CONFIG_DIR/base-images/
slemicro.iso
```

Nota

Durante a execução do EIB, a imagem base original **não** é modificada. Uma versão nova e personalizada é criada com a configuração desejada na raiz do diretório de configuração do EIB.

Neste momento, o diretório de configuração deve ter a seguinte aparência:

```
└── base-images/
    └── slemicro.iso
```

3.3 Criando o arquivo de definição de imagem

O arquivo de definição descreve grande parte das opções configuráveis disponíveis no Edge Image Builder. Um exemplo completo das opções é apresentado aqui (https://github.com/suse-edge/edge-image-builder/blob/release-1.2/pkg/image/testdata/full-valid-example.yaml), e recomendamos que você leia o guia de imagens de criação upstream (https://github.com/suse-edge/edge-image-builder/blob/release-1.2/docs/building-images.md) para ver exemplos mais detalhados do que vamos executar a seguir. Vamos começar com um arquivo de definição bem básico para a nossa imagem de sistema operacional:

```
cat << EOF > $CONFIG_DIR/iso-definition.yaml
apiVersion: 1.2
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
EOF
```

Essa definição especifica que estamos gerando uma imagem de saída para um sistema baseado em AMD64/Intel 64. A imagem que será usada como base para modificação posterior é uma imagem `iso` chamada `slemicro.iso`, que esperamos que esteja no local `$CONFIG_DIR/base-images/slemicro.iso`. Ela também descreve que, depois que o EIB terminar de modificar a imagem, a imagem de saída se chamará `eib-image.iso` e, por padrão, residirá em `$CONFIG_DIR`.

Agora a estrutura do nosso diretório deve ter esta aparência:

```
├── iso-definition.yaml
└── base-images/
    └── slemicro.iso
```

Nas seções a seguir, vamos analisar alguns exemplos de operações comuns:

3.3.1 Configurando usuários do sistema operacional

O EIB permite pré-configurar usuários com informações de login, como senhas ou chaves SSH, incluindo a definição de uma senha de root fixa. Como parte deste exemplo, vamos corrigir a senha de root, e o primeiro passo é usar o `OpenSSL` para criar uma senha criptografada unidirecional:

```
openssl passwd -6 SecurePassword
```

A saída será semelhante a esta:

```
$6$G392FCbxVgnLaFw1$Ujt00mdpJ3tDHxEg1snBU3GjujQf6f8kvopu7jiCBIhRbRvMmKUqwcmXAKggaSSKeUUOEtCP3ZUoZQY7zTX
```

Em seguida, podemos adicionar uma seção chamada `operatingSystem` ao arquivo de definição com uma matriz `users` dentro dela. O arquivo resultante terá esta aparência:

```
apiVersion: 1.2
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword:
 $6$G392FCbxVgnLaFw1$Ujt00mdpJ3tDHxEg1snBU3GjujQf6f8kvopu7jiCBIhRbRvMmKUqwcmXAKggaSSKeUUOEtCP3ZUoZQY7zT
```

Nota

É possível também adicionar outros usuários, criar diretórios pessoais, definir IDs de usuário, adicionar autenticação por chave SSH e modificar informações de grupo. Consulte o guia de criação de imagens upstream (https://github.com/suse-edge/edge-image-builder/blob/release-1.2/docs/building-images.md) para ver mais exemplos.

3.3.2 Configurando o horário do sistema operacional

A seção `time` é opcional, mas sua configuração é altamente recomendada para evitar possíveis problemas com certificados e divergência do relógio. O EIB configura o chronyd e o `/etc/localtime` dependendo desses parâmetros.

```
operatingSystem:
  time:
    timezone: Europe/London
    ntp:
      forceWait: true
      pools:
        - 2.suse.pool.ntp.org
      servers:
        - 10.0.0.1
        - 10.0.0.2
```

- O `timezone` especifica o fuso horário no formato "região/localidade" (por exemplo, "Europa/Londres"). É possível ver a lista completa executando o comando `timedatectl list-timezones` no sistema Linux.
- ntp: define atributos relacionados à configuração do NTP (usando o chronyd).
- forceWait: solicita que o chronyd tente sincronizar as fontes de horário antes de iniciar outros serviços, com um tempo limite de 180 segundos.
- pools: especifica uma lista de pools que o chronyd usará como fontes de dados (usando o `iburst` para melhorar o tempo gasto na sincronização inicial).
- servers: especifica uma lista de servidores que o chronyd usará como fontes de dados (usando o `iburst` para melhorar o tempo gasto na sincronização inicial).

Nota

Os valores apresentados neste exemplo são apenas para fins ilustrativos. Ajuste-os de acordo com seus requisitos específicos.

3.3.3 Adicionando certificados

Arquivos de certificado com a extensão ".pem" ou ".crt" armazenados no diretório `certificates` serão instalados no armazenamento de certificados global do sistema do nó:

```
.
├── definition.yaml
└── certificates
    ├── my-ca.pem
    └── my-ca.crt
```

Consulte o guia "Securing Communication with TLS Certificate" (https://documentation.suse.com/smart/security/html/tls-certificates/index.html#tls-adding-new-certificates) (Protegendo a comunicação com o certificado TLS) para obter mais informações.

3.3.4 Configurando pacotes RPM

Um dos principais recursos do EIB é o mecanismo para adicionar outros pacotes de software à imagem, dessa forma, o sistema pode aproveitar os pacotes instalados assim que a instalação é concluída. O EIB permite que os usuários especifiquem o seguinte:

- Pacotes por nome em uma lista na definição da imagem
- Repositórios de rede nos quais pesquisar os pacotes
- Credenciais do SUSE Customer Center (SCC) para pesquisar os pacotes listados em repositórios oficiais da SUSE
- Por um diretório `$CONFIG_DIR/rpms`, fazer sideload dos RPMs personalizados que não existem nos repositórios de rede
- Pelo mesmo diretório (`$CONFIG_DIR/rpms/gpg-keys`), chaves GPG para habilitar a validação de pacotes de terceiros

Em seguida, o EIB é executado por um processo de resolução de pacote no momento da criação da imagem, usando a imagem base como entrada, e tenta obter e instalar todos os pacotes fornecidos, especificados pela lista ou fornecidos localmente. O EIB faz download de todos os pacotes, incluindo as dependências, em um repositório existente na imagem de saída, e instrui o sistema a instalá-los durante o processo da primeira inicialização. A execução desse processo durante a criação da imagem garante que os pacotes sejam devidamente instalados na plataforma desejada, por exemplo, o nó de borda, durante a primeira inicialização. Isso também é vantajoso em ambientes nos quais você deseja fazer bake dos pacotes adicionais na imagem, em vez de extraí-los pela rede durante a operação, por exemplo, em ambientes air-gapped ou de rede restrita.

Como um exemplo simples para demonstrar esse procedimento, vamos instalar o pacote RPM `nvidia-container-toolkit` encontrado no repositório NVIDIA mantido por fornecedor terceirizado:

```
  packages:
    packageList:
      - nvidia-container-toolkit
    additionalRepos:
      - url: https://nvidia.github.io/libnvidia-container/stable/rpm/x86_64
```

O arquivo de definição resultante terá esta aparência:

```
apiVersion: 1.2
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword:
 $6$G392FCbxVgnLaFw1$Ujt00mdpJ3tDHxEg1snBU3GjujQf6f8kvopu7jiCBIhRbRvMmKUqwcmXAKggaSSKeUUOEtCP3ZUoZQY7zT
  packages:
    packageList:
      - nvidia-container-toolkit
    additionalRepos:
      - url: https://nvidia.github.io/libnvidia-container/stable/rpm/x86_64
```

O exemplo acima é simples, mas para uma totalidade, faça download da chave de assinatura do pacote NVIDIA antes de gerar a imagem:

```
$ mkdir -p $CONFIG_DIR/rpms/gpg-keys
```

```
$ curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey > $CONFIG_DIR/rpms/gpg-keys/nvidia.gpg
```

Atenção

A adição de outros RPMs por esse método tem como objetivo incluir componentes de terceiros compatíveis ou pacotes fornecidos (e mantidos) pelo usuário. Esse mecanismo não deve ser usado para adicionar pacotes que não costumam ser suportados no SUSE Linux Micro. Se ele for usado para adicionar componentes de repositórios openSUSE (que não são suportados), inclusive de versões ou pacotes de serviço mais recentes, você poderá acabar com um configuração sem suporte, principalmente quando a resolução de dependência resulta na substituição de partes importantes do sistema operacional, mesmo que o sistema resultante pareça funcionar conforme o esperado. Se você tiver qualquer dúvida, entre em contato com seu representante SUSE para obter ajuda para determinar se a configuração desejada é suportada.

Nota

Um guia mais completo com exemplos adicionais está disponível no guia de instalação de pacotes upstream (https://github.com/suse-edge/edge-image-builder/blob/release-1.2/docs/installing-packages.md).

3.3.5 Configurando o cluster Kubernetes e as cargas de trabalho dos usuários

Outro recurso do EIB é a capacidade de usá-lo para automatizar a implantação de clusters Kubernetes altamente disponíveis, tanto de nó único quanto de vários nós, que são "inicializados no local" (ou seja, não exigem a coordenação de nenhuma forma de infraestrutura de gerenciamento centralizado). O principal motivador dessa abordagem são as implantações air-gapped, ou ambientes de rede restrita, mas ela também é útil para inicializar rapidamente clusters independentes, mesmo com acesso à rede total e irrestrito disponível.

Com esse método, é possível implantar o sistema operacional personalizado e especificar a configuração do Kubernetes, componentes adicionais em camadas por gráficos Helm e cargas de trabalho de usuários por manifestos fornecidos pelo Kubernetes. No entanto, o princípio do projeto por trás do uso desse método é que já consideramos que o usuário deseja se desconectar e, portanto, os itens especificados na definição da imagem serão inseridos na imagem, o que inclui

as cargas de trabalho fornecidas pelo usuário. O EIB garantirá que todas as imagens descobertas necessárias, conforme as definições especificadas, sejam copiadas localmente e enviadas pelo registro de imagens incorporado ao sistema implantado resultante.

Neste exemplo, vamos usar nossa definição de imagem existente e especificar uma configuração do Kubernetes (não há uma lista dos sistemas e suas funções, portanto, vamos considerar um nó único), que vai instruir o EIB a provisionar um cluster Kubernetes RKE2 de nó único. Para mostrar a automação da implantação das cargas de trabalho fornecidas pelo usuário (por manifesto) e dos componentes em camadas (por Helm), vamos instalar o KubeVirt por meio do gráfico Helm do SUSE Edge e o NGINX por meio de um manifesto do Kubernetes. A configuração adicional que precisamos para anexar a definição da imagem existente é:

```
kubernetes:
  version: v1.32.4+rke2r1
  manifests:
    urls:
      - https://k8s.io/examples/application/nginx-app.yaml
  helm:
    charts:
      - name: kubevirt
        version: 303.0.0+up0.5.0
        repositoryName: suse-edge
    repositories:
      - name: suse-edge
        url: oci://registry.suse.com/edge/charts
```

O arquivo de definição completo resultante agora tem esta aparência:

```
apiVersion: 1.2
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword:
 $6$G392FCbxVgnLaFw1$Ujt00mdpJ3tDHxEg1snBU3GjujQf6f8kvopu7jiCBIhRbRvMmKUqwcmXAKggaSSKeUUOEtCP3ZUoZQY7zT
  packages:
    packageList:
      - nvidia-container-toolkit
    additionalRepos:
      - url: https://nvidia.github.io/libnvidia-container/stable/rpm/x86_64
kubernetes:
```

```
version: v1.32.4+k3s1
manifests:
  urls:
    - https://k8s.io/examples/application/nginx-app.yaml
helm:
  charts:
    - name: kubevirt
      version: 303.0.0+up0.5.0
      repositoryName: suse-edge
  repositories:
    - name: suse-edge
      url: oci://registry.suse.com/edge/charts
```

Nota

Há outros exemplos de opções, como implantações de vários nós, rede personalizada e opções/valores do gráfico Helm, disponíveis na documentação upstream (https://github.com/suse-edge/edge-image-builder/blob/release-1.2/docs/building-images.md).

3.3.6 Configurando a rede

No último exemplo deste início rápido, vamos configurar a rede que será criada quando um sistema for provisionado com a imagem gerada pelo EIB. É importante entender que, exceto se uma configuração de rede for especificada, o DHCP usará o modelo padrão em todas as interfaces descobertas no momento da inicialização. No entanto, ela nem sempre é a configuração desejada, principalmente se o DHCP não estiver disponível e você precisar fazer configurações estáticas, configurar estruturas de rede mais complexas, como vínculos, LACP e VLANs, ou substituir determinados parâmetros, por exemplo, nomes de host, servidores DNS e rotas.

O EIB permite fazer configurações por nó (em que o sistema em questão é identificado exclusivamente por seu endereço MAC) ou uma substituição para especificar uma configuração idêntica para cada máquina, o que é mais útil quando não se sabe os endereços MAC do sistema. O EIB usa uma ferramenta adicional chamada Network Manager Configurator, ou `nmc` na forma abreviada, que foi desenvolvida pela equipe do SUSE Edge para que a configuração de rede personalizada seja aplicada com base no esquema de rede declarativo nmstate.io (https://nmstate.io/) e, no momento da inicialização, identificará o nó em que está sendo inicializado e aplicará a configuração de rede desejada antes da ativação de quaisquer serviços.

Agora vamos aplicar uma configuração de rede estática a um sistema com uma interface única descrevendo o estado desejado da rede em um arquivo específico do nó (com base no nome de host desejado) no diretório `network` exigido:

```
mkdir $CONFIG_DIR/network

cat << EOF > $CONFIG_DIR/network/host1.local.yaml
routes:
  config:
  - destination: 0.0.0.0/0
    metric: 100
    next-hop-address: 192.168.122.1
    next-hop-interface: eth0
    table-id: 254
  - destination: 192.168.122.0/24
    metric: 100
    next-hop-address:
    next-hop-interface: eth0
    table-id: 254
dns-resolver:
  config:
    server:
    - 192.168.122.1
    - 8.8.8.8
interfaces:
- name: eth0
  type: ethernet
  state: up
  mac-address: 34:8A:B1:4B:16:E7
  ipv4:
    address:
    - ip: 192.168.122.50
      prefix-length: 24
    dhcp: false
    enabled: true
  ipv6:
    enabled: false
EOF
```

Atenção

O exemplo acima foi configurado para a sub-rede padrão `192.168.122.0/24` considerando que o teste é executado em uma máquina virtual. Adapte-o de acordo com o seu ambiente, lembrando do endereço MAC. Como é possível usar a mesma imagem

para provisionar vários nós, a rede configurada pelo EIB (via `nmc`) depende da capacidade de identificar exclusivamente o nó por seu endereço MAC. Como resultado, durante a inicialização, o `nmc` aplicará a configuração de rede correta a cada máquina. Isso significa que você precisa saber os endereços MAC dos sistemas nos quais deseja instalar. Como alternativa, o comportamento padrão é confiar no DHCP, mas você pode usar o gancho `configure-network.sh` para aplicar uma configuração comum a todos os nós. Consulte o guia de rede (*Capítulo 12, Rede de borda*) para obter mais detalhes.

A estrutura do arquivo resultante deverá ter esta aparência:

```
├── iso-definition.yaml
├── base-images/
│   └── slemicro.iso
└── network/
    └── host1.local.yaml
```

A configuração de rede que acabamos de criar será analisada, e os arquivos de conexão necessários do NetworkManager serão automaticamente gerados e inseridos na nova imagem de instalação que o EIB criará. Esses arquivos serão aplicados durante o provisionamento do host, resultando na configuração de rede completa.

Nota

Consulte o componente de rede de borda (*Capítulo 12, Rede de borda*) para ver uma explicação mais abrangente da configuração acima e exemplos desse recurso.

3.4 Criando a imagem

Agora que temos uma imagem base e uma definição da imagem para o EIB usar, vamos criar a imagem. Para isso, simplesmente usaremos o `podman` para chamar o contêiner do EIB com o comando "build", especificando o arquivo de definição:

```
podman run --rm -it --privileged -v $CONFIG_DIR:/eib \
registry.suse.com/edge/3.3/edge-image-builder:1.2.1 \
build --definition-file iso-definition.yaml
```

A saída do comando deve ter esta aparência:

```
Setting up Podman API listener...
```

```
Downloading file: dl-manifest-1.yaml 100% (498/498 B, 9.5 MB/s)
Pulling selected Helm charts... 100% (1/1, 43 it/min)
Generating image customization components...
Identifier ................... [SUCCESS]
Custom Files ................. [SKIPPED]
Time ......................... [SKIPPED]
Network ...................... [SUCCESS]
Groups ....................... [SKIPPED]
Users ........................ [SUCCESS]
Proxy ........................ [SKIPPED]
Resolving package dependencies...
Rpm .......................... [SUCCESS]
Os Files ..................... [SKIPPED]
Systemd ...................... [SKIPPED]
Fips ......................... [SKIPPED]
Elemental .................... [SKIPPED]
Suma ......................... [SKIPPED]
Populating Embedded Artifact Registry... 100% (3/3, 10 it/min)
Embedded Artifact Registry ... [SUCCESS]
Keymap ....................... [SUCCESS]
Configuring Kubernetes component...
The Kubernetes CNI is not explicitly set, defaulting to 'cilium'.
Downloading file: rke2_installer.sh
Downloading file: rke2-images-core.linux-amd64.tar.zst 100% (657/657 MB, 48 MB/s)
Downloading file: rke2-images-cilium.linux-amd64.tar.zst 100% (368/368 MB, 48 MB/s)
Downloading file: rke2.linux-amd64.tar.gz 100% (35/35 MB, 50 MB/s)
Downloading file: sha256sum-amd64.txt 100% (4.3/4.3 kB, 6.2 MB/s)
Kubernetes ................... [SUCCESS]
Certificates ................. [SKIPPED]
Cleanup ...................... [SKIPPED]
Building ISO image...
Kernel Params ................ [SKIPPED]
Build complete, the image can be found at: eib-image.iso
```

A imagem ISO criada é armazenada em `$CONFIG_DIR/eib-image.iso`:

```
├── iso-definition.yaml
├── eib-image.iso
├── _build
│   └── cache/
│       └── ...
│   └── build-<timestamp>/
│       └── ...
├── base-images/
│   └── slemicro.iso
└── network/
    └── host1.local.yaml
```

Cada build cria uma pasta com marcação de data e hora em `$CONFIG_DIR/_build/`, que inclui os registros do build, os artefatos usados durante o build e os diretórios `combustion` e `artefacts` com todos os scripts e artefatos adicionados à imagem CRB.

O conteúdo do diretório deve ter esta aparência:

```
├── build-<timestamp>/
│   ├── combustion/
│   │   ├── 05-configure-network.sh
│   │   ├── 10-rpm-install.sh
│   │   ├── 12-keymap-setup.sh
│   │   ├── 13b-add-users.sh
│   │   ├── 20-k8s-install.sh
│   │   ├── 26-embedded-registry.sh
│   │   ├── 48-message.sh
│   │   ├── network/
│   │   │   ├── host1.local/
│   │   │   │   └── eth0.nmconnection
│   │   │   └── host_config.yaml
│   │   ├── nmc
│   │   └── script
│   ├── artefacts/
│   │   ├── registry/
│   │   │   ├── hauler
│   │   │   ├── nginx:<version>-registry.tar.zst
│   │   │   ├── rancher_kubectl:<version>-registry.tar.zst
│   │   │   └── registry.suse.com_suse_sles_15.6_virt-operator:<version>-registry.tar.zst
│   │   ├── rpms/
│   │   │   └── rpm-repo
│   │   │       ├── addrepo0
│   │   │       │   ├── nvidia-container-toolkit-<version>.rpm
│   │   │       │   ├── nvidia-container-toolkit-base-<version>.rpm
│   │   │       │   ├── libnvidia-container1-<version>.rpm
│   │   │       │   └── libnvidia-container-tools-<version>.rpm
│   │   │       ├── repodata
│   │   │       │   ├── ...
│   │   │       └── zypper-success
│   │   └── kubernetes/
│   │       ├── rke2_installer.sh
│   │       ├── registries.yaml
│   │       ├── server.yaml
│   │       ├── images/
│   │       │   ├── rke2-images-cilium.linux-amd64.tar.zst
│   │       │   └── rke2-images-core.linux-amd64.tar.zst
│   │       ├── install/
│   │       │   ├── rke2.linux-amd64.tar.gz
│   │       │   └── sha256sum-amd64.txt
```

```
│   │       └── manifests/
│   │           ├── dl-manifest-1.yaml
│   │           └── kubevirt.yaml
│   ├── createrepo.log
│   ├── eib-build.log
│   ├── embedded-registry.log
│   ├── helm
│   │   └── kubevirt
│   │       └── kubevirt-0.4.0.tgz
│   ├── helm-pull.log
│   ├── helm-template.log
│   ├── iso-build.log
│   ├── iso-build.sh
│   ├── iso-extract
│   │   └── ...
│   ├── iso-extract.log
│   ├── iso-extract.sh
│   ├── modify-raw-image.sh
│   ├── network-config.log
│   ├── podman-image-build.log
│   ├── podman-system-service.log
│   ├── prepare-resolver-base-tarball-image.log
│   ├── prepare-resolver-base-tarball-image.sh
│   ├── raw-build.log
│   ├── raw-extract
│   │   └── ...
│   └── resolver-image-build
│       └──...
└── cache
    └── ...
```

Se houver falha no build, `eib-build.log` será o primeiro registro com as informações. A partir disso, ele vai direcionar você para o componente com falha para depuração.

Neste ponto, você deve ter uma imagem pronta para uso que:

1. Implantará o SUSE Linux Micro 6.1.
2. Configurará a senha de root.
3. Instalará o pacote `nvidia-container-toolkit`.
4. Configurará um registro de contêiner incorporado para enviar o conteúdo localmente.
5. Instalará o RKE2 de nó único.
6. Configurará a rede estática.

7. Instalará o KubeVirt.
8. Implantará um manifesto fornecido pelo usuário.

3.5 Depurando o processo de criação da imagem

Se o processo de criação da imagem falhar, consulte o guia de depuração upstream (https://github.com/suse-edge/edge-image-builder/blob/release-1.2/docs/debugging.md).

3.6 Testando a imagem recém-criada

Para obter instruções de como testar a imagem CRB recém-criada, consulte o guia de teste de imagem upstream (https://github.com/suse-edge/edge-image-builder/blob/release-1.2/docs/testing-guide.md).

4 SUSE Multi-Linux Manager

O SUSE Multi-Linux Manager está incluído no SUSE Edge para oferecer a automação e o controle que mantêm o sistema operacional SUSE Linux Micro subjacente sempre atualizado em todos os nós da implantação de borda.

O objetivo deste início rápido é agilizar ao máximo para você a instalação e configuração do SUSE Multi-Linux Manager para fornecer as atualizações de sistema operacional para seus nós de borda. O guia de início rápido não aborda tópicos como dimensionar armazenamento, criar e gerenciar canais de software adicionais para fins de preparação ou gerenciar usuários, grupos de sistemas e organizações para implantações de maior porte. Para uso em produção, é altamente recomendável familiarizar-se com a documentação completa do SUSE Multi-Linux Manager (https://documentation.suse.com/suma/5.0/en/suse-manager/index.html).

As seguintes etapas são necessárias para preparar o SUSE Edge para uso eficaz do SUSE Multi-Linux Manager:

- Implantar e configurar o SUSE Multi-Linux Manager Server.
- Sincronizar os repositórios de pacotes do SUSE Linux Micro.
- Criar grupos de sistemas.
- Criar chaves de ativação.
- Usar o Edge Image Builder para preparar a mídia de instalação para registro do SUSE Multi-Linux Manager.

4.1 Implantar o SUSE Multi-Linux Manager Server

Se você já tem uma instância da versão mais recente do SUSE Multi-Linux Manager 5.0 em execução, pode ignorar esta etapa.

Você pode executar o SUSE Multi-Linux Manager Server em um servidor físico dedicado, como uma máquina virtual em seu próprio hardware, ou na nuvem. São fornecidas imagens pré-configuradas da máquina virtual para o SUSE Multi-Linux Server para nuvens públicas suportadas.

Neste início rápido, usamos a imagem "qcow2" `SUSE-Manager-Server.x86_64-5.0.2-Qcow-2024.12.qcow2` para AMD64/Intel 64, que você encontra em https://www.suse.com/download/suse-manager/ ou no SUSE Customer Center. Essa imagem funciona como uma máquina virtual em hipervisores do tipo KVM. Verifique e use sempre a versão mais recente da imagem para novas instalações.

Você também pode instalar o SUSE Multi-Linux Manager Server em qualquer outra arquitetura de hardware compatível. Nesse caso, escolha a imagem correspondente à arquitetura.

Depois de fazer download da imagem, crie uma máquina virtual que atenda pelo menos às seguintes especificações mínimas de hardware:

- 16 GB de RAM
- 4 núcleos físicos ou virtuais
- um dispositivo de blocos adicional com, no mínimo, 100 GB

Com a imagem qcow2, não há necessidade de instalar o sistema operacional. É possível anexar a imagem diretamente como a partição raiz.

Configure a rede para que os nós de borda possam acessar o SUSE Multi-Linux Manager Server posteriormente com um nome de host que inclua o nome de domínio completo e qualificado ("FQDN", Fully Qualified Domain Name)!

Quando você inicializa o SUSE Multi-Linux Manager pela primeira vez, precisa fazer algumas configurações iniciais:

- Selecionar o layout do seu teclado
- Aceitar o contrato de licença
- Selecionar seu fuso horário
- Inserir a senha de root do sistema operacional

As etapas seguintes precisam ser realizas como usuário "root":

Para a etapa seguinte, são necessários dois códigos de registro, que você encontra no SUSE Customer Center:

- Seu código de registro do SLE Micro 5.5
- Seu código de registro da extensão do SUSE Multi-Linux Manager

Registre o SUSE Linux Micro:

```
transactional-update register -r <REGCODE> -e <your_email>
```

Registre o SUSE Multi-Linux Manager:

```
transactional-update register -p SUSE-Manager-Server/5.0/x86_64 -r <REGCODE>
```

A string do produto depende da arquitetura de hardware. Por exemplo, se você usa o SUSE Multi-Linux Manager em um sistema Arm de 64 bits, a string é "SUSE-Manager-Server/5.0/aarch64".

Reinicializar

Atualize o sistema:

```
transactional-update
```

Reinicialize para aplicar as atualizações (se houver).

O SUSE Multi-Linux Manager é oferecido por um contêiner gerenciado pelo Podman. O comando `mgradm` cuida da instalação e da configuração para você.

Atenção

É muito importante configurar o nome de host no SUSE Multi-Linux Manager Server com um nome de domínio completo e qualificado ("FQDN", Fully Qualified Domain Name) que os nós de borda que você deseja gerenciar possam resolver apropriadamente na rede!

Antes de instalar e configurar o contêiner do SUSE Multi-Linux Manager Server, você precisa preparar o dispositivo de blocos adicional que já incluiu. Para isso, é necessário saber o nome que a máquina virtual atribuiu ao dispositivo. Por exemplo, se o dispositivos de blocos é `/dev/vdb`, configure-o para uso no SUSE Multi-Linux Manager executando o seguinte comando:

```
mgr-storage-server /dev/vdb
```

Implante o SUSE Multi-Linux Manager:

```
mgradm install podman <FQDN>
```

Insira a senha para o certificado de CA. Essa senha deve ser diferente das suas senhas de login. Em geral, você não precisa inseri-la posteriormente, mas é bom anotá-la.

Insira a senha do usuário "admin". Esse é o usuário inicial para fazer login no SUSE Multi-Linux Manager. Você pode criar outros usuários com direitos completos ou restritos no futuro.

4.2 Configurar o SUSE Multi-Linux Manager

Após a conclusão da implantação, faça login na IU da web do SUSE Multi-Linux Manager usando o nome de host que você já forneceu. O usuário inicial é "admin". Use a senha informada na etapa anterior.

Para a etapa seguinte, você precisa das credenciais da sua organização, que estão na segunda subguia da guia "Usuários" da organização no SUSE Customer Center. Com essas credenciais, o SUSE Multi-Linux Manager pode sincronizar todos os produtos que você assinou.

Selecione *Admin* › *Assistente de configuração*.

Na guia `Credenciais da Organização`, crie uma nova credencial com seu `Nome de usuário` e `Senha` que constam no SUSE Customer Center.

Vá para a guia seguinte `Produtos SUSE`. Aguarde o término da primeira sincronização de dados com o SUSE Customer Center.

Depois que a lista for preenchida, use o filtro para mostrar apenas "Micro 6". Marque a caixa do SUSE Linux Micro 6.1 relativa à arquitetura de hardware em que seus nós de borda serão executados (`x86_64` ou `aarch64`).

Clique em `Adicionar produtos` para adicionar o repositório ("canal") de pacotes principal do SUSE Linux Micro e, automaticamente, o canal para as ferramentas do cliente SUSE Manager como um subcanal.

Dependendo da sua conexão com a Internet, a primeira sincronização levará algum tempo. Você já pode iniciar as etapas seguintes:

Em `Sistemas > Grupos do Sistema`, crie pelo menos um grupo em que seus sistemas ingressarão quando forem integrados. Os grupos são importantes para categorizar os sistemas de modo que você possa aplicar uma configuração ou ações a todo o conjunto de sistemas de uma vez. Por concepção, eles são similares aos rótulos no Kubernetes.

Clique em `+ Criar Grupo`.

Insira um nome abreviado, por exemplo, "Nós de borda", e uma descrição longa.

Em `Sistemas > Chaves de ativação`, crie pelo menos uma chave de ativação. Pense nas chaves de ativação como perfis de configuração que são automaticamente aplicados aos sistemas depois de integrados ao SUSE Multi-Linux Manager. Para adicionar determinados nós de borda a grupos diferentes ou usar outra configuração, você pode criar chaves de ativação separadas para eles e usá-las no Edge Image Builder para criar uma mídia de instalação personalizada.

Um caso de uso avançado comum das chaves de ativação é para atribuir clusters de teste aos canais de software com as atualizações mais recentes, e os clusters de produção aos canais de software que apenas recebem essas atualizações mais recentes depois que você as testou no cluster de teste.

Clique em `+ Criar chave`.

Escolha uma descrição resumida, por exemplo, "Nós de borda". Insira um nome exclusivo que identifique a chave, por exemplo, "edge-x86_64" para os nós de borda com arquitetura de hardware AMD64/Intel 64. Um prefixo de número é automaticamente adicionado à chave. Para a organização padrão, o número é sempre "1". Se você criar mais organizações no SUSE Multi-Linux Manager e gerar chaves para elas, esse número poderá ser diferente.

Se você não criou canais de software clonados, pode manter a configuração do canal base como "padrão do SUSE Manager". Isso atribui automaticamente o repositório de atualização correto do SUSE aos nós de borda.

Como "canal filho", selecione o controle deslizante para "incluir recomendações" referente à arquitetura de hardware para a qual a chave de ativação é usada. Dessa forma, o "SUSE-Manager-Tools-For-SL-Micro-6.1" será adicionado ao canal.

Na guia "Grupos", adicione o grupo criado. Todos os nós integrados por meio dessa chave de ativação serão automaticamente adicionados a esse grupo.

4.3 Criar uma imagem de instalação personalizada com o Edge Image Builder

Para usar o Edge Image Builder, você precisa apenas de um ambiente para iniciar o contêiner baseado em Linux com o Podman.

Na verdade, para configuração mínima de laboratório, podemos usar a mesma máquina virtual na qual o SUSE Multi-Linux Manager Server está em execução. Verifique se você tem espaço em disco suficiente na máquina virtual! Essa configuração não é recomendada para uso em produção. Consulte a *Seção 3.1, "Pré-requisitos"* para saber quais sistemas operacionais host foram testados com o Edge Image Builder.

Faça login no host do SUSE Multi-Linux Manager Server como root.

Acesse o contêiner do Edge Image Builder:

```
podman pull registry.suse.com/edge/3.3/edge-image-builder:1.2.1
```

Crie o diretório `/opt/eib` e o subdiretório `base-images`:

```
mkdir -p /opt/eib/base-images
```

Neste início rápido, usamos a variante de "autoinstalação" da imagem do SUSE Linux Micro. Posteriormente, essa imagem poderá ser gravada em uma unidade removível USB física que você pode usar para instalação em servidores físicos. Se o servidor tem a opção de anexar remotamente as ISOs de instalação por um BMC (Baseboard Management Controller), você também pode adotar essa abordagem. Por fim, também é possível usar essa imagem com grande parte das ferramentas de virtualização.

Para pré-carregar a imagem diretamente em um nó físico ou iniciá-la de uma VM, você também pode usar a variante de imagem "bruta".

Você encontra essas imagens no SUSE Customer Center ou em https://www.suse.com/download/sle-micro/

Faça download ou copie a imagem `SL-Micro.x86_64-6.1-Default-SelfInstall-GM.install.iso` no diretório `base-images` e dê o nome "slemicro.iso" a ela.

A criação de imagens AArch64 em hosts de build baseados no Arm é uma prévia de tecnologia no SUSE Edge 3.3.1. É muito provável que ela funcione, mas ainda não tem suporte. Se você quer tentar, precisa ter o Podman em uma máquina Arm de 64 bits e substituir "x86_64" em todos os exemplos e trechos de código por "aarch64".

Em `/opt/eib`, crie um arquivo chamado `iso-definition.yaml`. Essa é a definição do seu build para o Edge Image Builder.

Este é um exemplo simples para instalar o SL Micro 6.1, definir uma senha de root e o mapa do teclado, iniciar a IU gráfica do Cockpit e registrar o nó no SUSE Multi-Linux Manager:

```
apiVersion: 1.0
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
  - username: root
    createHomeDir: true
    encryptedPassword: $6$aaBTHyqDRUMY1HAp$pmBY7.qLtoVlCGj32XR/Ogei4cngc3f4OX7fwBD/
gw7HWyuNBOKYbBWnJ4pvrYwH2WUtJLKMbinVtBhMDHQIY0
  keymap: de
  systemd:
    enable:
```

```
    - cockpit.socket
packages:
  noGPGCheck: true
suma:
  host: ${fully qualified hostname of your SUSE Multi-Linux Manager Server}
  activationKey: 1-edge-x86_64
```

O Edge Image Builder também configura a rede, instala automaticamente o Kubernetes no nó e até implanta aplicativos por gráficos Helm. Consulte o *Capítulo 3, Clusters independentes com o Edge Image Builder* para ver exemplos mais completos.

Para `baseImage`, especifique o nome real da ISO no diretório `base-images` que deseja usar.

Neste exemplo, a senha de root é "root". Consulte a *Seção 3.3.1, "Configurando usuários do sistema operacional"* para criar um hash para a senha segura que deseja usar.

Defina o mapa do teclado com o layout desejado para o sistema após a instalação.

Nota

Usamos a opção `noGPGCheck: true` porque não vamos fornecer uma chave GPG para verificar pacotes RPM. Um guia completo com uma configuração mais segura recomendada para uso em produção está disponível no guia de instalação de pacotes upstream (https://github.com/suse-edge/edge-image-builder/blob/release-1.2/docs/installing-packages.md).

Como já foi mencionado várias vezes, o host do SUSE Multi-Linux Manager exige um nome de host completo e qualificado que possa ser resolvido na rede em que os nós de borda serão inicializados.

O valor de `activationKey` precisa corresponder à chave que você criou no SUSE Multi-Linux Manager.

Para criar uma imagem de instalação que registre automaticamente os nós de borda no SUSE Multi-Linux Manager após a instalação, você também precisa preparar dois artefatos:

- pacote Salt minion para instalar o agente de gerenciamento para o SUSE Multi-Linux Manager
- certificado de CA do servidor SUSE Multi-Linux Manager

4.3.1 Fazer download do pacote venv-salt-minion

Em `/opt/eib`, crie o subdiretório `rpms`.

Faça download do pacote `venv-salt-minion` do seu servidor SUSE Multi-Linux Manager para esse diretório. Você pode obtê-lo pela IU da web localizando o pacote em `Software > Lista de canais` e fazendo download dele do canal SUSE-Manager-Tools, ou fazer download dele do "repositório de inicialização" do SUSE Multi-Linux Manager com uma ferramenta como curl:

```
curl -O http://${HOSTNAME_OF_SUSE_MANAGER}/pub/repositories/slmicro/6/1/bootstrap/x86_64/
venv-salt-minion-3006.0-3.1.x86_64.rpm
```

O nome do pacote real pode ser diferente se uma nova versão já foi lançada. Se houver vários pacotes para seleção, escolha sempre o mais recente.

4.4 Fazer download do certificado de CA do SUSE Multi-Linux Manager

Em `/opt/eib`, crie o subdiretório `certificates`.

Faça download do certificado de CA do SUSE Multi-Linux Manager para esse diretório:

```
curl -O http://${HOSTNAME_OF_SUSE_MANAGER}/pub/RHN-ORG-TRUSTED-SSL-CERT
```

Atenção

Você deve renomear o certificado para `RHN-ORG-TRUSTED-SSL-CERT.crt`. Depois disso, o Edge Image Builder garantirá que ele seja instalado e ativado no nó de borda durante a instalação.

Agora execute o Edge Image Builder:

```
cd /opt/eib
podman run --rm -it --privileged -v /opt/eib:/eib \
registry.suse.com/edge/3.3/edge-image-builder:1.2.1 \
build --definition-file iso-definition.yaml
```

Se você usou outro nome para o arquivo de definição YAML e quer usar uma versão diferente do Edge Image Builder, precisa adaptar o comando de acordo.

Após o término da criação, a ISO de instalação estará no diretório `/opt/eib` como `eib-image.iso`.

II Componentes

Lista de componentes do Edge

5 Rancher

Consulte a documentação do Rancher em https://ranchermanager.docs.rancher.com/v2.11.

> O Rancher é uma plataforma de gerenciamento Kubernetes avançada e de código-fonte aberto que simplifica a implantação, as operações e o monitoramento de clusters Kubernetes em vários ambientes. Se você gerencia clusters no local, na nuvem ou na borda, o Rancher oferece uma plataforma unificada e centralizada para todas as suas necessidades do Kubernetes.

5.1 Principais recursos do Rancher

- **Gerenciamento multicluster:** a interface intuitiva do Rancher permite gerenciar cluster Kubernetes de qualquer lugar: nuvens públicas, centro de dados privados e locais de borda.
- **Segurança e conformidade:** o Rancher impõe políticas de segurança, controle de acesso com base em função (RBAC, Role-Based Access Control) e padrões de conformidade de acordo com o seu cenário do Kubernetes.
- **Operações de cluster simplificadas:** o Rancher automatiza o provisionamento de clusters, faz upgrade e soluciona problemas, simplificando as operações do Kubernetes para equipes de todos os tamanhos.
- **Catálogo de aplicativos centralizado:** o catálogo de aplicativos do Rancher oferece uma variedade de gráficos Helm e operadores do Kubernetes, o que facilita a implantação e o gerenciamento de aplicativos conteinerizados.
- **Entrega contínua:** o Rancher oferece suporte a GitOps e pipelines de CI/CD, o que automatiza e simplifica os processos de entrega dos aplicativos.

5.2 Uso do Rancher no SUSE Edge

O Rancher oferece várias funcionalidades importantes para a pilha do SUSE Edge:

5.2.1 Gerenciamento centralizado do Kubernetes

Nas implantações de borda típicas com vários clusters distribuídos, o Rancher atua como um plano de controle central para gerenciar os clusters Kubernetes. Ele oferece uma interface unificada para provisionamento, upgrade, monitoramento e solução de problemas, simplificando as operações e garantindo a consistência.

5.2.2 Implantação de clusters simplificada

O Rancher simplifica a criação de clusters Kubernetes no sistema operacional leve SUSE Linux Micro, o que facilita a distribuição da infraestrutura de borda com recursos avançados do Kubernetes.

5.2.3 Implantação e gerenciamento de aplicativos

O catálogo de aplicativos integrado do Rancher simplifica a implantação e o gerenciamento de aplicativos conteinerizados nos clusters SUSE Edge, facilitando a implantação de cargas de trabalho de borda.

5.2.4 Imposição de segurança e políticas

O Rancher oferece ferramentas de governança com base em política, controle de acesso com base em função (RBAC, Role-Based Access Control) e integração com provedores de autenticação externos. Isso ajuda a manter a segurança e a conformidade das implantações do SUSE Edge, o que é crítico em ambientes distribuídos.

5.3 Melhores práticas

5.3.1 GitOps

O Rancher inclui o Fleet como componente incorporado que permite gerenciar configurações de clusters e implantações de aplicativos com um código armazenado em git.

5.3.2 Observabilidade

O Rancher inclui ferramentas incorporadas de monitoramento e registro, como Prometheus e Grafana, para insights abrangentes sobre a integridade e o desempenho do seu cluster.

5.4 Instalando com o Edge Image Builder

O SUSE Edge usa o *Capítulo 11, Edge Image Builder* para personalizar as imagens base do sistema operacional SUSE Linux Micro. Leia a *Seção 27.6, "Instalação do Rancher"* para instalação air-gapped do Rancher em clusters Kubernetes provisionados pelo EIB.

5.5 Recursos adicionais

- Documentação do Rancher (https://rancher.com/docs/)
- Rancher Academy (https://www.rancher.academy/)
- Rancher Community (https://rancher.com/community/)
- Gráficos Helm (https://helm.sh/)
- Operadores do Kubernetes (https://operatorhub.io/)

6 Extensões do Rancher Dashboard

As extensões permitem que usuários, desenvolvedores, parceiros e clientes estendam e aprimorem a IU do Rancher. O SUSE Edge oferece as extensões de dashboard KubeVirt e Akri.

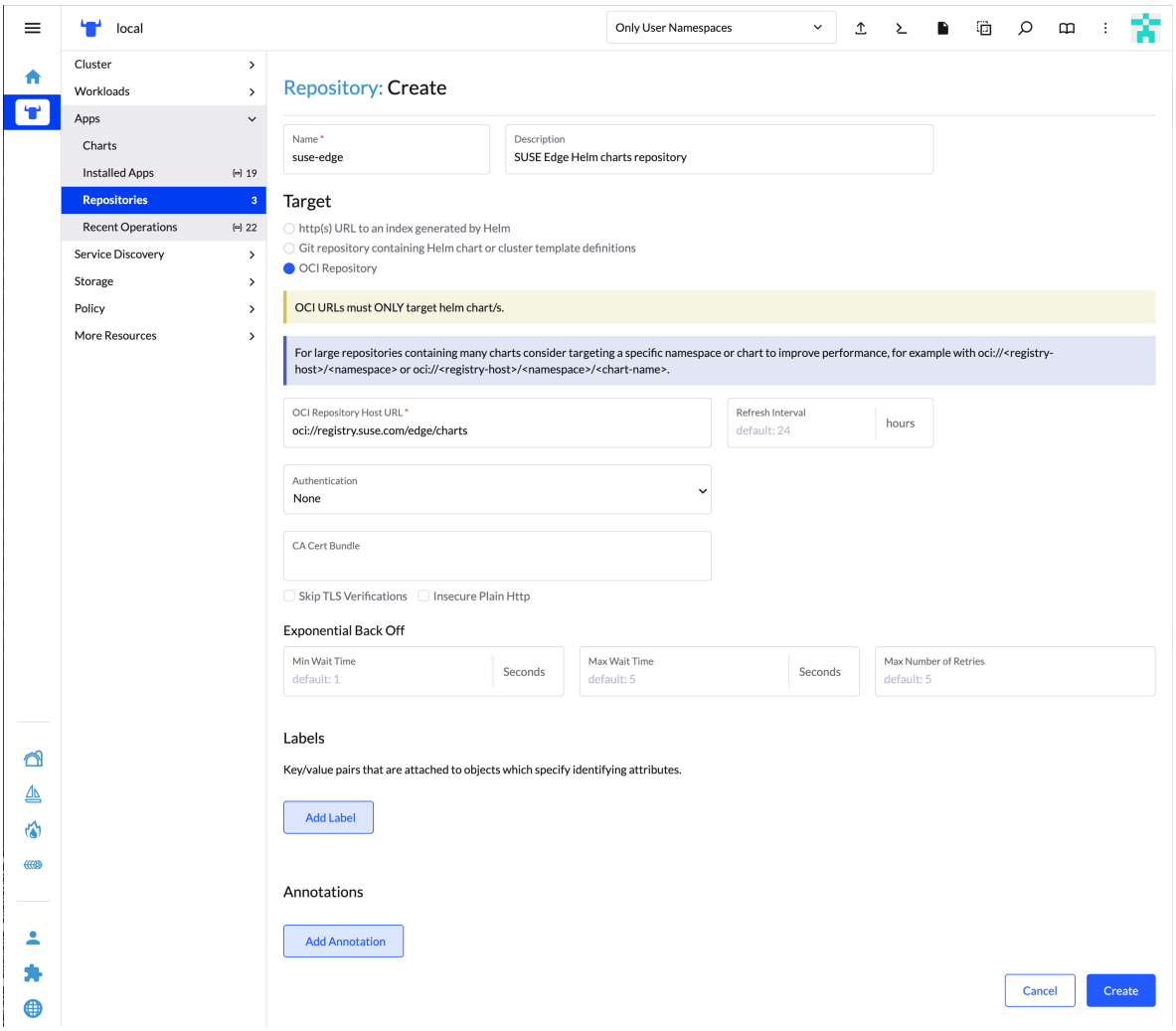
Consulte a `documentação do Rancher` para obter informações gerais sobre as extensões do Rancher Dashboard.

6.1 Instalação

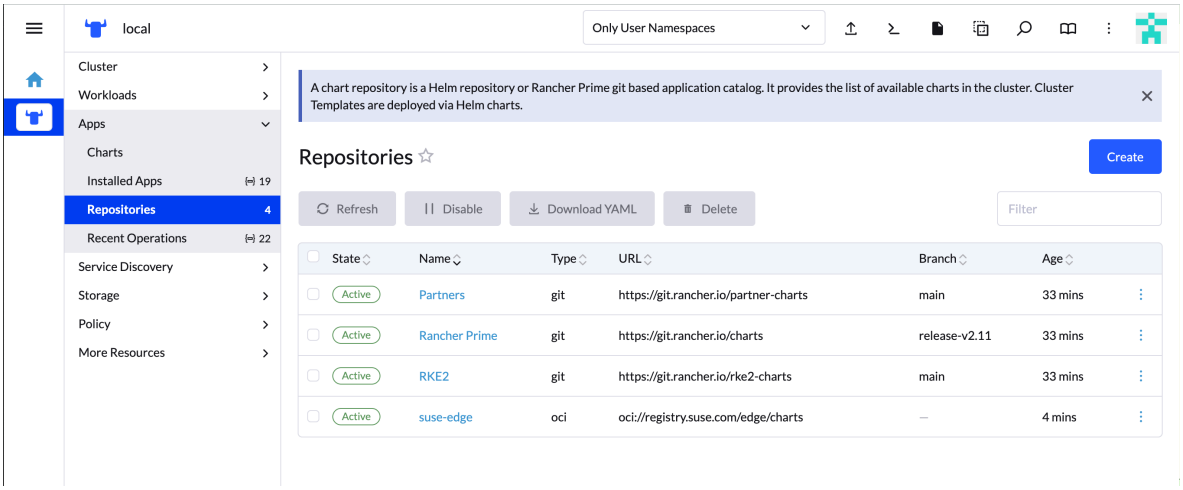
Todos os componentes do SUSE Edge 3.3.1, incluindo as extensões de dashboard, são distribuídos como artefatos OCI. Para instalar as extensões do SUSE Edge, você pode usar a IU do Rancher Dashboard, o Helm ou o Fleet:

6.1.1 Instalando a IU do Rancher Dashboard

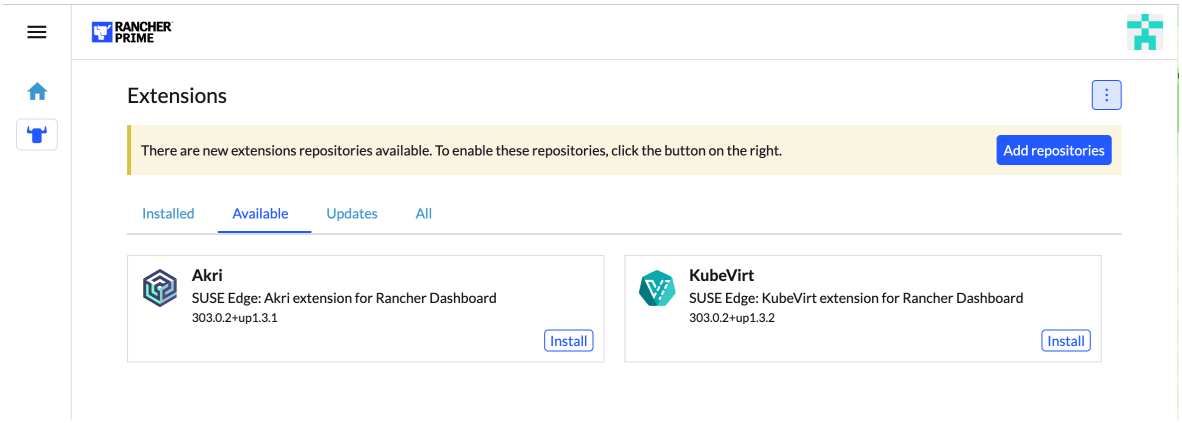
1. Clique em **Extensions** (Extensões) na seção **Configuration** (Configuração) da barra lateral de navegação.
2. Na página de extensões, clique no menu de três pontos na parte superior direita e selecione **Manage Repositories** (Gerenciar repositórios).
   Cada extensão é distribuída por seu próprio artefato OCI. Elas estão disponíveis no repositório de gráficos Helm do SUSE Edge.
3. Na página **Repositories** (Repositórios), clique em `Create` (Criar).
4. No formulário, especifique o nome e o URL do repositório e clique em `Create` (Criar).
   URL do repositório de gráficos Helm do SUSE Edge: `oci://registry.suse.com/edge/charts`



5. Veja que o repositório de extensões foi adicionado à lista com o estado `Active` (Ativo).



6. Volte para **Extensions** (Extensões) na seção **Configuration** (Configuração) da barra lateral de navegação.
   Na guia **Available** (Disponível), veja as extensões disponíveis para instalação.



7. No cartão da extensão, clique em `Install` (Instalar) e confirme a instalação.
   Depois que a extensão for instalada, a IU do Rancher solicitará que a página seja recarregada, conforme descrito na `página da documentação de instalação de extensões do Rancher`.

6.1.2 Instalando com o Helm

```
# KubeVirt extension
helm install kubevirt-dashboard-extension oci://registry.suse.com/edge/charts/kubevirt-
dashboard-extension --version 303.0.2+up1.3.2 --namespace cattle-ui-plugin-system

# Akri extension
helm install akri-dashboard-extension oci://registry.suse.com/edge/charts/akri-dashboard-
extension --version 303.0.2+up1.3.1 --namespace cattle-ui-plugin-system
```

Nota

É preciso instalar as extensões no namespace `cattle-ui-plugin-system`.

Nota

Após a instalação de uma extensão, a IU do Rancher Dashboard deverá ser recarregada.

6.1.3 Instalando com o Fleet

A instalação de extensões de dashboard com o Fleet requer a definição de um recurso `gitRepo` que aponte para um repositório Git com um ou mais arquivos de configuração de bundle `fleet.yaml` personalizados.

```
# KubeVirt extension fleet.yaml
defaultNamespace: cattle-ui-plugin-system
helm:
  releaseName: kubevirt-dashboard-extension
  chart: oci://registry.suse.com/edge/charts/kubevirt-dashboard-extension
  version: "303.0.2+up1.3.2"
```

```
# Akri extension fleet.yaml
defaultNamespace: cattle-ui-plugin-system
helm:
  releaseName: akri-dashboard-extension
  chart: oci://registry.suse.com/edge/charts/akri-dashboard-extension
  version: "303.0.2+up1.3.1"
```

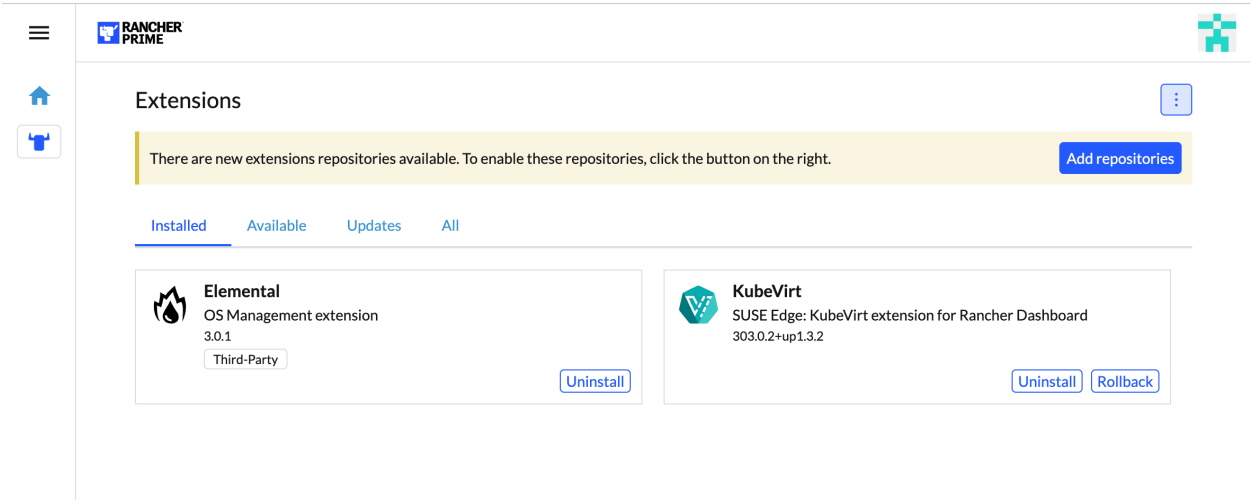
Nota

A propriedade `releaseName` é necessária e precisa corresponder ao nome da extensão para instalá-la corretamente.

```
cat <<- EOF | kubectl apply -f -
apiVersion: fleet.cattle.io/v1alpha1
metadata:
  name: edge-dashboard-extensions
  namespace: fleet-local
spec:
  repo: https://github.com/suse-edge/fleet-examples.git
  branch: main
  paths:
  - fleets/kubevirt-dashboard-extension/
  - fleets/akri-dashboard-extension/
EOF
```

Para obter mais informações, consulte o *Capítulo 8, Fleet* e o repositório `fleet-examples`.

Após a instalação das extensões, elas serão listadas na seção **Extensions** (Extensões) nas guias **Installed** (Instaladas). Como não são instaladas via Apps/Marketplace, elas são marcadas com o rótulo `Third-Party` (Terceiros).



6.2 Extensão de dashboard KubeVirt

A extensão KubeVirt fornece gerenciamento de máquinas virtuais básicas para a IU do Rancher Dashboard. Seus recursos estão descrito na *Seção 21.7.2, "Usando a extensão KubeVirt do Rancher Dashboard"*.

6.3 Extensão de dashboard Akri

Akri é uma interface de recursos do Kubernetes que permite expor facilmente dispositivos folha heterogêneos (por exemplo, câmeras IP e dispositivos USB) como recursos em um cluster Kubernetes, além de oferecer suporte à exposição de recursos de hardware incorporados, como GPUs e FPGAs. A Akri detecta continuamente os nós com acesso a esses dispositivos e programa as cargas de trabalho com base neles.

Com a extensão de dashboard Akri, você pode usar a interface de usuário do Rancher Dashboard para gerenciar e monitorar dispositivos folha e executar cargas de trabalho após a descoberta desses dispositivos.

Há uma descrição mais detalhada dos recursos da extensão na *Seção 14.5, "Extensão Akri do Rancher Dashboard"*.

7 Rancher Turtles

Consulte a documentação do Rancher Turtles em https://documentation.suse.com/cloudnative/cluster-api/

> O Rancher Turtles é um operador do Kubernetes que proporciona integração entre o Rancher Manager e a Cluster API (CAPI), com o objetivo de levar suporte completo para CAPI ao Rancher.

7.1 Principais recursos do Rancher Turtles

- Importar automaticamente clusters CAPI para o Rancher instalando o Rancher Cluster Agent nos clusters provisionados pela CAPI.
- Instalar e configurar as dependências do controlador CAPI pelo CAPI Operator (https://cluster-api-operator.sigs.k8s.io/).

7.2 Uso do Rancher Turtles no SUSE Edge

A pilha do SUSE Edge oferece um gráfico Helm agrupador que instala o Rancher Turtles com uma configuração específica que habilita:

- Os principais componentes do controlador da CAPI
- Os componentes do provedor de plano de controle e de inicialização do RKE2
- Os componentes do provedor de infraestrutura do Metal3 (*Capítulo 10, Metal*^{3})

Há suporte apenas para os provedores padrão instalados pelo gráfico agrupador. Os provedores alternativos de plano de controle, inicialização e infraestrutura não são suportados atualmente como parte da pilha do SUSE Edge.

7.3 Instalando o Rancher Turtles

É possível instalar o Rancher Turtles seguindo o Guia de Início Rápido do Metal3 (*Capítulo 1, Implantações automatizadas de BMC com Metal*^{3}) ou a documentação do cluster de gerenciamento (*Capítulo 40, Configurando o cluster de gerenciamento*).

7.4 Recursos adicionais

- Documentação do Rancher (https://rancher.com/docs/)
- Manual da Cluster API (https://cluster-api.sigs.k8s.io/)

8 Fleet

Fleet (https://fleet.rancher.io) é um mecanismo de gerenciamento e implantação de contêineres desenvolvido para oferecer aos usuários mais controle sobre o cluster local e monitoramento constante por GitOps. O foco do Fleet não está apenas na capacidade de ajustar a escala, mas também em proporcionar aos usuários um alto grau de controle e visibilidade para monitorar exatamente o que está instalado no cluster.

O Fleet gerencia implantações do Git de arquivos YAML brutos no Kubernetes, gráficos Helm, Kustomize ou qualquer combinação dos três. Seja qual for a fonte, todos os recursos são dinamicamente convertidos em gráficos Helm, e o Helm é usado como mecanismo para implantar todos os recursos no cluster. Como resultado, os usuários aproveitam um alto grau de controle, consistência e auditabilidade dos clusters.

Para obter informações sobre o funcionamento do Fleet, consulte a arquitetura do Fleet (https://ranchermanager.docs.rancher.com/v2.11/integrations-in-rancher/fleet/architecture).

8.1 Instalando o Fleet com o Helm

O Fleet está incorporado ao Rancher, mas também é possível instalá-lo (https://fleet.rancher.io/installation) como aplicativo independente em qualquer cluster Kubernetes que usa o Helm.

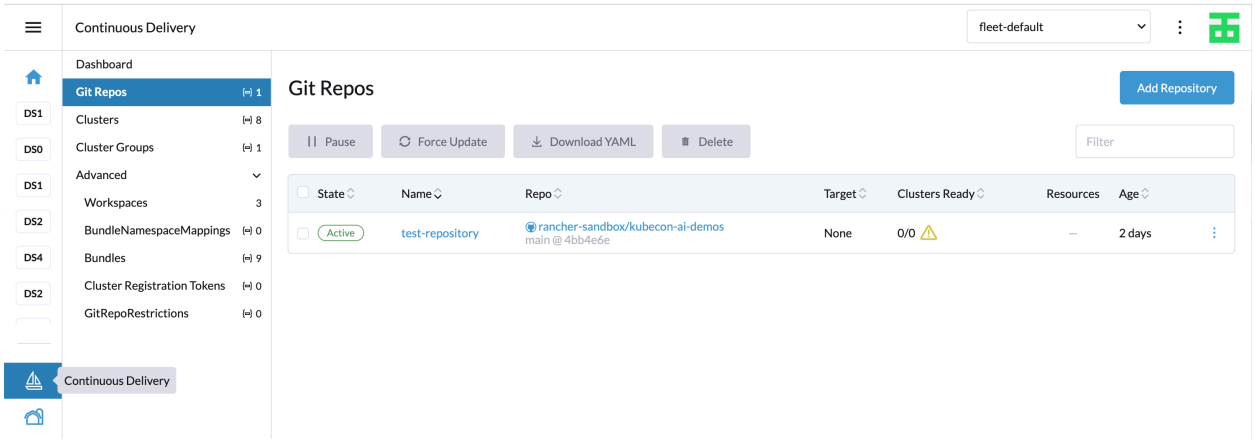
8.2 Usando o Fleet com o Rancher

O Rancher usa o Fleet para implantar aplicativos em clusters gerenciados. A entrega contínua com o Fleet introduz o GitOps em escala, o que foi projetado para gerenciar aplicativos executados em uma grande quantidade de clusters.

O Fleet se destaca como parte integrante do Rancher. O agente Fleet é automaticamente implantado em clusters gerenciados com o Rancher, como parte do processo de instalação/importação, e o cluster fica imediatamente disponível para gerenciamento pelo Fleet.

8.3 Acessando o Fleet na IU do Rancher

O Fleet vem pré-instalado no Rancher e é gerenciado pela opção **Continuous Delivery** (Entrega contínua) na IU do Rancher.



A seção Continuous Delivery (Entrega contínua) consiste nos seguintes itens:

8.3.1 Dashboard

Uma página de visão geral de todos os repositórios do GitOps em todos os espaços de trabalho. São mostrados apenas os espaços de trabalho com repositórios.

8.3.2 Repositórios Git

Uma lista de repositórios do GitOps no espaço de trabalho selecionado. Escolha o espaço de trabalho ativo na lista suspensa na parte superior da página.

8.3.3 Clusters

Uma lista de clusters gerenciados. Por padrão, todos os clusters gerenciados pelo Rancher são adicionados ao espaço de trabalho `fleet-default`. O espaço de trabalho `fleet-local` inclui o cluster (de gerenciamento) local. Nele, é possível `Pause` (Pausar) ou `Force update` (Forçar a atualização) dos clusters ou mover o cluster para outro espaço de trabalho. A edição do cluster permite atualizar rótulos e anotações usados para agrupamento de clusters.

8.3.4 Grupos de clusters

A seção "Cluster Groups" (Grupos de clusters) permite o agrupamento personalizado de clusters no espaço de trabalho usando os seletores.

8.3.5 Avançado

A seção "Advanced" (Avançado) permite gerenciar espaços de trabalho e outros recursos do Fleet relacionados.

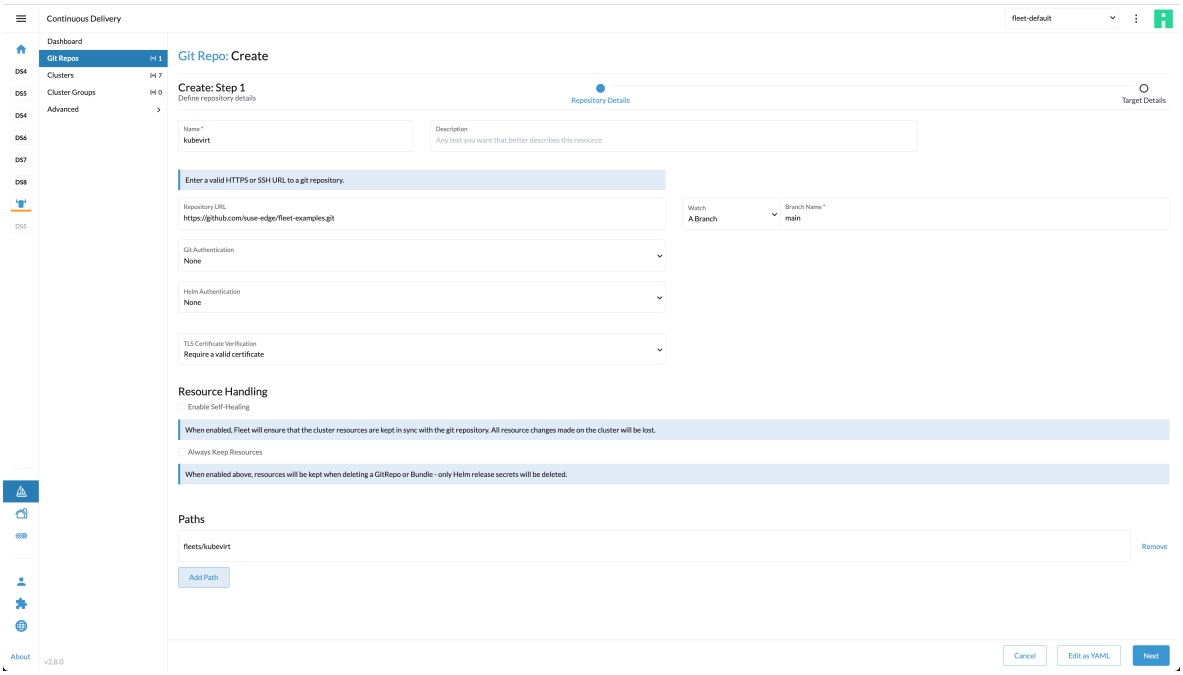
8.4 Exemplo de instalação do KubeVirt com o Rancher e o Fleet usando o Rancher Dashboard

1. Crie um repositório Git com o arquivo `fleet.yaml`:

```
defaultNamespace: kubevirt
helm:
  chart: "oci://registry.suse.com/edge/charts/kubevirt"
  version: "303.0.0+up0.5.0"
  # kubevirt namespace is created by kubevirt as well, we need to take ownership of
 it
  takeOwnership: true
```

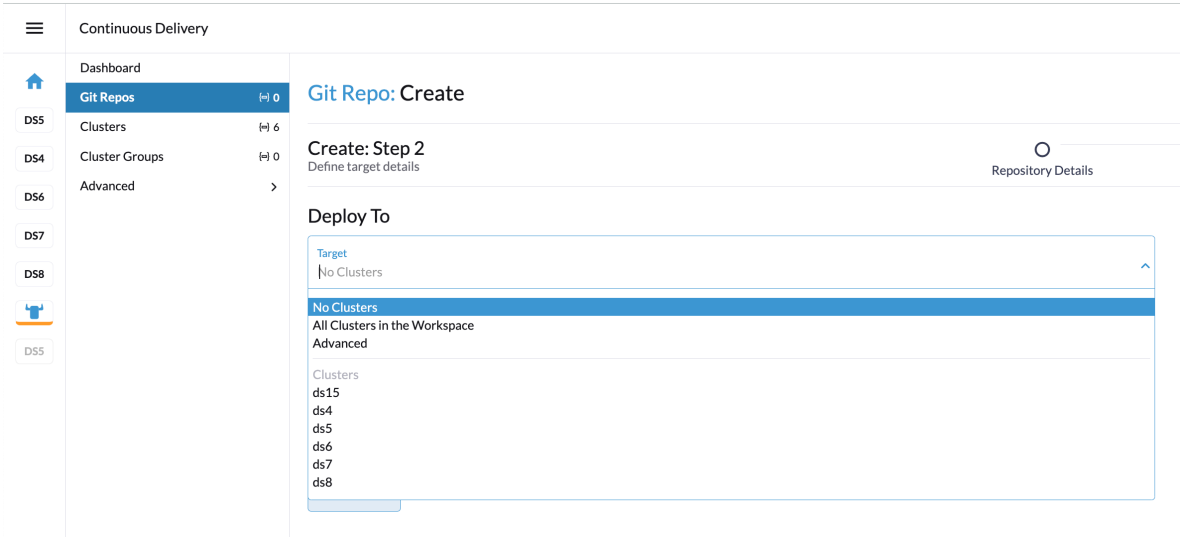
2. No Rancher Dashboard, navegue até # > **Continuous Delivery** > **Git Repos** (Entrega contínua > Repositórios Git) e clique em `Add Repository` (Adicionar repositório).

3. O assistente de criação de repositório orienta na criação do repositório Git. Insira o **Name** (Nome), **Repository URL** (URL do repositório), mencionando o repositório Git criado na etapa anterior, e selecione a ramificação ou revisão apropriada. No caso de um repositório mais complexo, especifique os **Paths** (Caminhos) para usar vários diretórios em um único repositório.



4. Clique em `Next` (Avançar).

5. Na próxima etapa, você poderá definir o local de implantação das cargas de trabalho. A seleção de cluster inclui várias opções básicas: nenhum cluster, todos os clusters ou escolher diretamente um cluster gerenciado ou grupo de clusters específico (se definido). A opção "Advanced" (Avançado) permite editar os seletores direto pelo YAML.

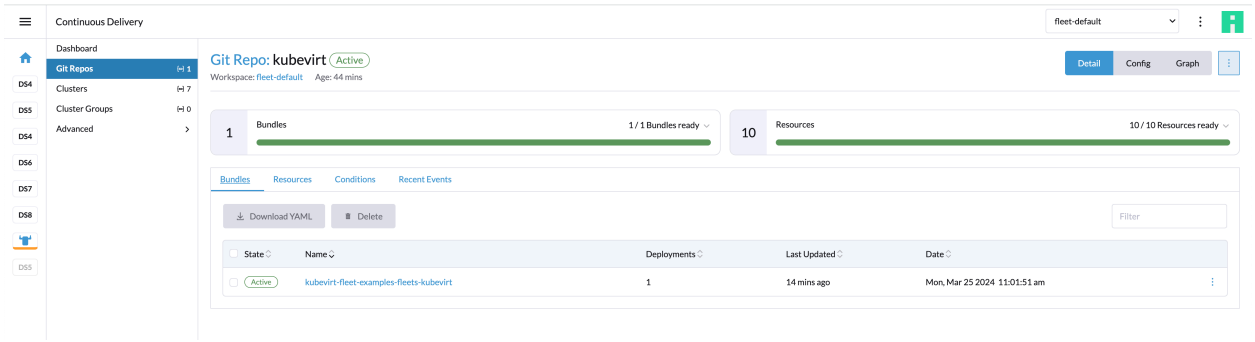


6. Clique em `Create` (Criar). O repositório é criado. A partir de agora, as cargas de trabalho são instaladas e sincronizadas nos clusters correspondentes à definição do repositório.

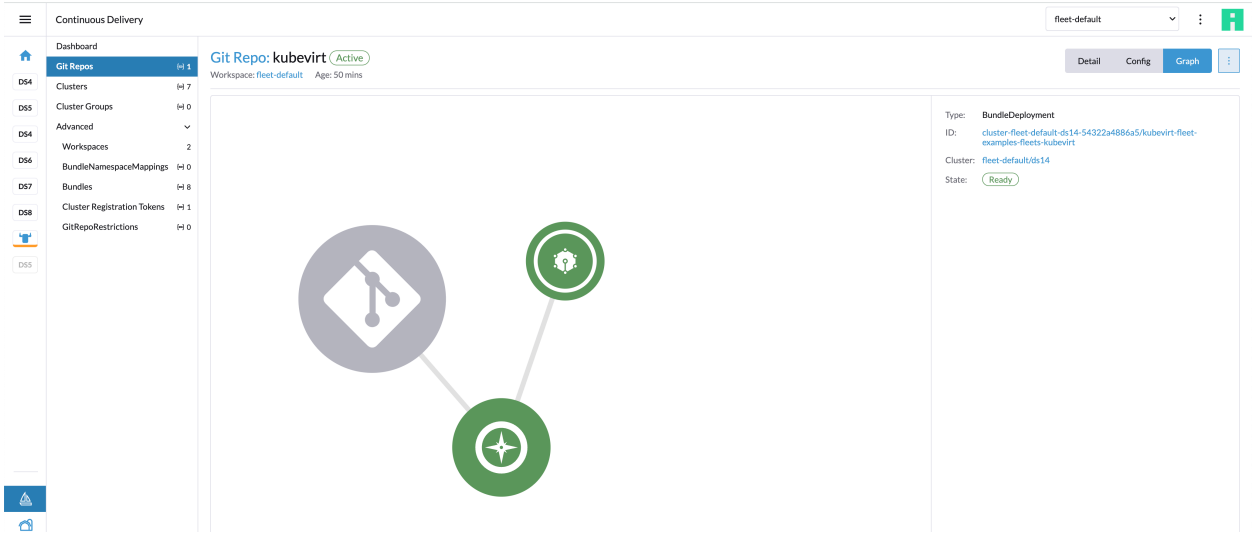
8.5 Depurando e solucionando problemas

A seção de navegação "Advanced" (Avançado) apresenta visões gerais dos recursos do Fleet de nível mais baixo. Bundle (https://fleet.rancher.io/ref-bundle-stages) é um recurso interno usado para orquestração dos recursos do Git. Quando um repositório Git é verificado, ele gera um ou mais bundles.

Para localizar os bundles relevantes a um repositório específico, vá até a página de detalhes do repositório Git e clique na guia `Bundles`.



Para cada cluster, o bundle é aplicado a um recurso BundleDeployment que foi criado. Para ver os detalhes do BundleDeployment, clique no botão `Graph` (Gráfico) na parte superior direita da página de detalhes do repositório Git. É carregado um gráfico de **Repo** > **Bundles** > **BundleDeployments** (Repositório > Bundles > BundleDeployments). Clique no BundleDeployment no gráfico para ver seus detalhes e clique no `Id` para conferir o YAML do BundleDeployment.



Para obter informações adicionais com dicas de solução de problemas no Fleet, acesse aqui (https://fleet.rancher.io/troubleshooting).

8.6 Exemplos do Fleet

A equipe do Edge mantém um repositório (https://github.com/suse-edge/fleet-examples) com exemplos de instalação de projetos do Edge com o Fleet.

O projeto do Fleet inclui o repositório fleet-examples (https://github.com/rancher/fleet-examples) que abrange todos os casos de uso da estrutura de repositório Git (https://fleet.rancher.io/gitrepo-content).

9 SUSE Linux Micro

Consulte a documentação oficial do SUSE Linux Micro (https://documentation.suse.com/sle-micro/6.1/)

> O SUSE Linux Micro é um sistema operacional leve e seguro para a borda. Ele combina os componentes protegidos da empresa do SUSE Linux Enterprise com os recursos que os desenvolvedores desejam em um sistema operacional moderno e imutável. Como resultado, você tem uma plataforma de infraestrutura confiável com a mais alta conformidade, porém fácil de usar.

9.1 Como o SUSE Edge usa o SUSE Linux Micro?

O SUSE Linux Micro é usado como o sistema operacional de base para a nossa pilha de plataforma. Desse modo, contamos com uma base segura, estável e mínima para criação.

O SUSE Linux Micro é exclusivo na questão do uso de instantâneos do sistema de arquivos (Btrfs) para facilitar as reversões se houver qualquer erro no upgrade. Esse recurso protege os upgrades remotos de toda a plataforma mesmo sem acesso físico em caso de problemas.

9.2 Melhores práticas

9.2.1 Mídia de instalação

O SUSE Edge usa o Edge Image Builder (*Capítulo 11, Edge Image Builder*) para pré-configurar a imagem de autoinstalação do SUSE Linux Micro.

9.2.2 Administração local

O SUSE Linux Micro vem com o Cockpit, que permite o gerenciamento local do host por meio de um aplicativo web.

Por padrão, esse serviço está desabilitado, mas é possível habilitar o serviço `cockpit.socket` do systemd para iniciá-lo.

9.3 Problemas conhecidos

- No momento, não existe um ambiente de área de trabalho disponível no SUSE Linux Micro, mas há uma solução conteinerizada em desenvolvimento.

10 Metal^3

Metal^3 (https://metal3.io/) é um projeto da CNCF que oferece recursos de gerenciamento de infraestrutura bare metal para Kubernetes.

O Metal^3 inclui recursos nativos do Kubernetes para gerenciar o ciclo de vida de servidores bare metal que suportam gerenciamento com protocolos fora da banda, como o Redfish (https://www.dmtf.org/standards/redfish).

Ele também conta com suporte a Cluster API (CAPI) (https://cluster-api.sigs.k8s.io/), que permite gerenciar os recursos da infraestrutura de vários provedores por meio de APIs amplamente adotadas e independentes de fornecedor.

10.1 Como o SUSE Edge usa o Metal^3?

Este método é útil em cenários com hardware de destino que permite o gerenciamento fora da banda e quando se deseja um fluxo de gerenciamento de infraestrutura automatizado.

Esse método fornece APIs declarativas que permitem o gerenciamento de inventário e de estado dos servidores bare metal, incluindo inspeção, limpeza e provisionamento/desprovisionamento automatizados.

10.2 Problemas conhecidos

- No momento, não há suporte para o controlador de gerenciamento de endereços IP (https://github.com/metal3-io/ip-address-manager) upstream, pois ele ainda não é compatível com a nossa seleção de ferramentas de configuração de rede.
- Da mesma forma, os recursos do IPAM e os campos networkData do Metal3DataTemplate não são suportados.
- Apenas há suporte para implantação por redfish-virtualmedia.

11 Edge Image Builder

Consulte o repositório oficial (https://github.com/suse-edge/edge-image-builder).

O Edge Image Builder (EIB) é uma ferramenta que simplifica a geração das imagens de disco personalizadas e prontas para inicialização (CRB) para máquinas de inicialização. Essas imagens permitem a implantação de ponta a ponta da pilha de software completa do SUSE com uma única imagem.

O EIB cria imagens CRB para todos os cenários de provisionamento e ainda demonstra um enorme valor para as implantações air-gapped com redes limitadas ou totalmente isoladas.

11.1 Como o SUSE Edge usa o Edge Image Builder?

O SUSE Edge usa o EIB para configuração rápida e simplificada de imagens personalizadas do SUSE Linux Micro para diversos cenários, incluindo inicialização de máquinas virtuais e bare metal com:

- Implantações totalmente air-gapped do Kubernetes K3s/RKE2 (nó único e vários nós)
- Implantações totalmente air-gapped de gráficos Helm e manifestos do Kubernetes
- Registro no Rancher pela API Elemental
- Metal^3
- Rede personalizada (por exemplo, IP estático, nome de host, VLANs, vinculação etc.)
- Configurações personalizadas de sistema operacional (por exemplo, usuários, grupos, senhas, chaves SSH, proxies, NTP, certificados SSL personalizados etc.)
- Instalação air-gapped de pacotes RPM no nível do host e sideloaded (com resolução de dependências)
- Registro do SUSE Multi-Linux Manager para gerenciamento de sistema operacional
- Imagens de contêiner incorporadas
- Argumentos de linha de comando do Kernel
- Unidades do Systemd para habilitação/desabilitação no momento da inicialização
- Scripts e arquivos personalizados para tarefas manuais

11.2 Introdução

A documentação completa sobre uso e teste do Edge Image Builder está disponível aqui (https://github.com/suse-edge/edge-image-builder/tree/release-1.2/docs).

Consulte também o *Capítulo 3, Clusters independentes com o Edge Image Builder* que apresenta um cenário de implantação básica.

Depois que você se familiarizar com a ferramenta, encontre mais informações úteis em nossa página de dicas de truques (../tips/eib.adoc).

11.3 Problemas conhecidos

- O EIB cria gabaritos dos gráficos Helm para isolá-los e analisa todas as imagens dentro dos gabaritos. Se um gráfico Helm não incluir todas as suas imagens dentro de um gabarito e, em vez disso, fizer sideload das imagens, o EIB não poderá isolar essas imagens automaticamente. Nesse caso, a solução é adicionar manualmente as imagens não detectadas à seção `embeddedArtifactRegistry` do arquivo de definição.

12 Rede de borda

Esta seção descreve a abordagem de configuração de rede na solução SUSE Edge. Vamos mostrar como configurar o NetworkManager no SUSE Linux Micro de maneira declarativa e explicar como as ferramentas relacionadas são integradas.

12.1 Visão geral do NetworkManager

O NetworkManager é uma ferramenta que gerencia a conexão de rede principal e outras interfaces de conexão.

O NetworkManager armazena as configurações de rede como arquivos de conexão que contêm o estado desejado. Essas conexões são armazenadas como arquivos no diretório `/etc/NetworkManager/system-connections/`.

Os detalhes sobre o NetworkManager estão disponíveis na documentação do SUSE Linux Micro (https://documentation.suse.com/sle-micro/6.1/html/Micro-network-configuration/index.html).

12.2 Visão geral do nmstate

O nmstate é uma biblioteca muito usada (com uma ferramenta CLI auxiliar) que oferece uma API declarativa para configurações de rede usando um esquema predefinido.

Os detalhes sobre o nmstate estão disponíveis na documentação upstream (https://nmstate.io/).

12.3 Inserir: NetworkManager Configurator (nmc)

As opções de personalização de rede no SUSE Edge são obtidas por uma ferramenta CLI chamada NetworkManager Configurator, ou *nmc* na forma abreviada, que aproveita a funcionalidade da biblioteca nmstate e, desse modo, é totalmente capaz de configurar endereços IP estáticos, servidores DNS, VLANs, vínculos, pontes etc. Essa ferramenta permite gerar configurações de rede com base em estados desejados predefinidos e aplicá-las a vários nós de maneira automatizada.

Os detalhes sobre o NetworkManager Configurator (nmc) estão disponíveis no repositório upstream (https://github.com/suse-edge/nm-configurator).

12.4 Como o SUSE Edge usa o NetworkManager Configurator?

O SUSE Edge utiliza o *nmc* para as personalizações de rede em diversos modelos de provisionamento:

- Configurações de rede personalizadas em cenários de provisionamento de rede direcionado (*Capítulo 1, Implantações automatizadas de BMC com Metal*[3])
- Configurações estáticas declarativas em cenários de provisionamento com base em imagem (*Capítulo 3, Clusters independentes com o Edge Image Builder*)

12.5 Configurando com o Edge Image Builder

O Edge Image Builder (EIB) é uma ferramenta que permite configurar vários hosts com uma única imagem de sistema operacional. Nesta seção, vamos mostrar como usar uma abordagem declarativa para descrever os estados da rede desejados, como eles são convertidos nas respectivas conexões do NetworkManager e, em seguida, aplicados durante o processo de provisionamento.

12.5.1 Pré-requisitos

Se você está seguindo este guia, já deve ter os seguintes itens disponíveis:

- Host físico (ou máquina virtual) AMD64/Intel 64 com SLES 15 SP6 ou openSUSE Leap 15.6
- Runtime de contêiner disponível (por exemplo, Podman)
- Cópia da imagem RAW do SUSE Linux Micro 6.1 disponível aqui (https://www.suse.com/download/sle-micro/)

12.5.2 Obtendo a imagem de contêiner do Edge Image Builder

A imagem de contêiner do EIB está disponível publicamente e pode ser baixada do registro SUSE Edge executando o seguinte comando:

```
podman pull registry.suse.com/edge/3.3/edge-image-builder:1.2.1
```

12.5.3 Criando o diretório de configuração de imagem

Vamos começar pela criação do diretório de configuração:

```
export CONFIG_DIR=$HOME/eib
mkdir -p $CONFIG_DIR/base-images
```

Agora vamos mover a cópia da imagem base baixada para o diretório de configuração:

```
mv /path/to/downloads/SL-Micro.x86_64-6.1-Base-GM.raw $CONFIG_DIR/base-images/
```

Nota

O EIB nunca modifica a entrada da imagem base, ele cria uma nova com as respectivas modificações.

Neste momento, o diretório de configuração deve ter a seguinte aparência:

```
└── base-images/
    └── SL-Micro.x86_64-6.1-Base-GM.raw
```

12.5.4 Criando o arquivo de definição de imagem

O arquivo de definição descreve a maioria das opções configuráveis que o Edge Image Builder suporta.

Vamos começar com um arquivo de definição bem básico para nossa imagem de sistema operacional:

```
cat << EOF > $CONFIG_DIR/definition.yaml
apiVersion: 1.2
image:
  arch: x86_64
  imageType: raw
  baseImage: SL-Micro.x86_64-6.1-Base-GM.raw
  outputImageName: modified-image.raw
operatingSystem:
  users:
    - username: root
      encryptedPassword: $6$jHugJNNd3HElGsUZ
$eodjVe4te5ps44SVcWshdfWizrP.xAyd71CVEXazBJ/.v799/WRCBXxfYmunlBO2yp1hm/zb4r8EmnrrNCF.P/
```

```
EOF
```

A seção `image` é obrigatória e especifica a imagem de entrada, sua arquitetura e tipo, além do nome da imagem de saída. A seção `operatingSystem` é opcional e contém a configuração para permitir o login nos sistemas provisionados com nome de usuário/senha `root/eib`.

Nota

Você pode usar sua própria senha criptografada executando `openssl passwd -6 <senha>`.

Neste momento, o diretório de configuração deve ter a seguinte aparência:

```
├── definition.yaml
└── base-images/
    └── SL-Micro.x86_64-6.1-Base-GM.raw
```

12.5.5 Definindo as configurações de rede

As configurações de rede desejadas não fazem parte do arquivo de definição da imagem que acabamos de criar. Vamos agora preenchê-las no diretório especial `network/`. Para criá-lo:

```
mkdir -p $CONFIG_DIR/network
```

Como já foi mencionado, a ferramenta NetworkManager Configurator (*nmc*) espera uma entrada no formato de esquema predefinido. Você encontra como configurar uma ampla variedade de opções de rede na documentação de exemplos upstream do NMState (https://nmstate.io/examples.html).

Esse guia explica como configurar a rede em três nós diferentes:

- Um nó que usa duas interfaces Ethernet
- Um nó que usa vínculo de rede
- Um nó que usa ponte de rede

Atenção

O uso de configurações de rede completamente diferentes não é recomendado em builds de produção, especialmente na configuração de clusters Kubernetes. Em geral, as configurações de rede devem ser homogêneas entre os nós ou, pelo menos, entre as funções em um determinado cluster. Este guia inclui diversas opções apenas para servir como referência de exemplo.

Nota

No exemplo abaixo, foi considerada a rede padrão `libvirt` com um intervalo de endereços IP `192.168.122.1/24`. Ajuste-a de acordo se for diferente em seu ambiente.

Vamos criar os estados desejados para o primeiro nó, que será chamado de `node1.suse.com`:

```
cat << EOF > $CONFIG_DIR/network/node1.suse.com.yaml
routes:
  config:
    - destination: 0.0.0.0/0
      metric: 100
      next-hop-address: 192.168.122.1
      next-hop-interface: eth0
      table-id: 254
    - destination: 192.168.122.0/24
      metric: 100
      next-hop-address:
      next-hop-interface: eth0
      table-id: 254
dns-resolver:
  config:
    server:
      - 192.168.122.1
      - 8.8.8.8
interfaces:
  - name: eth0
    type: ethernet
    state: up
    mac-address: 34:8A:B1:4B:16:E1
    ipv4:
      address:
        - ip: 192.168.122.50
          prefix-length: 24
```

```
      dhcp: false
      enabled: true
    ipv6:
      enabled: false
  - name: eth3
    type: ethernet
    state: down
    mac-address: 34:8A:B1:4B:16:E2
    ipv4:
      address:
        - ip: 192.168.122.55
          prefix-length: 24
      dhcp: false
      enabled: true
    ipv6:
      enabled: false
EOF
```

Nesse exemplo, definimos um estado desejado de duas interfaces Ethernet (eth0 e eth3), os endereços IP solicitados, o roteamento e a resolução DNS.

Atenção

Garanta que os endereços MAC de todas as interfaces Ethernet sejam listados. Eles são usados durante o processo de provisionamento como identificadores dos nós e servem para determinar quais configurações devem ser aplicadas. É dessa forma que podemos configurar vários nós usando uma única imagem ISO ou RAW.

O próximo é o segundo nó que será chamado de `node2.suse.com` e usará vínculo de rede:

```
cat << EOF > $CONFIG_DIR/network/node2.suse.com.yaml
routes:
  config:
    - destination: 0.0.0.0/0
      metric: 100
      next-hop-address: 192.168.122.1
      next-hop-interface: bond99
      table-id: 254
    - destination: 192.168.122.0/24
      metric: 100
      next-hop-address:
      next-hop-interface: bond99
      table-id: 254
dns-resolver:
  config:
```

```
    server:
      - 192.168.122.1
      - 8.8.8.8
interfaces:
  - name: bond99
    type: bond
    state: up
    ipv4:
      address:
        - ip: 192.168.122.60
          prefix-length: 24
      enabled: true
    link-aggregation:
      mode: balance-rr
      options:
        miimon: '140'
      port:
        - eth0
        - eth1
  - name: eth0
    type: ethernet
    state: up
    mac-address: 34:8A:B1:4B:16:E3
    ipv4:
      enabled: false
    ipv6:
      enabled: false
  - name: eth1
    type: ethernet
    state: up
    mac-address: 34:8A:B1:4B:16:E4
    ipv4:
      enabled: false
    ipv6:
      enabled: false
EOF
```

Nesse exemplo, definimos um estado desejado de duas interfaces Ethernet (eth0 e eth1) que não habilitam endereçamento IP, além de um vínculo com política round robin e o respectivo endereço que será usado para encaminhar o tráfego de rede.

Por fim, vamos criar o terceiro e último arquivo de estado desejado, que usará uma ponte de rede e será chamado de `node3.suse.com`:

```
cat << EOF > $CONFIG_DIR/network/node3.suse.com.yaml
routes:
  config:
```

```
    - destination: 0.0.0.0/0
      metric: 100
      next-hop-address: 192.168.122.1
      next-hop-interface: linux-br0
      table-id: 254
    - destination: 192.168.122.0/24
      metric: 100
      next-hop-address:
      next-hop-interface: linux-br0
      table-id: 254
dns-resolver:
  config:
    server:
      - 192.168.122.1
      - 8.8.8.8
interfaces:
  - name: eth0
    type: ethernet
    state: up
    mac-address: 34:8A:B1:4B:16:E5
    ipv4:
      enabled: false
    ipv6:
      enabled: false
  - name: linux-br0
    type: linux-bridge
    state: up
    ipv4:
      address:
        - ip: 192.168.122.70
          prefix-length: 24
      dhcp: false
      enabled: true
    bridge:
      options:
        group-forward-mask: 0
        mac-ageing-time: 300
        multicast-snooping: true
        stp:
          enabled: true
          forward-delay: 15
          hello-time: 2
          max-age: 20
          priority: 32768
      port:
        - name: eth0
          stp-hairpin-mode: false
```

```
          stp-path-cost: 100
          stp-priority: 32
EOF
```

Neste momento, o diretório de configuração deve ter a seguinte aparência:

```
├── definition.yaml
├── network/
│   ├── node1.suse.com.yaml
│   ├── node2.suse.com.yaml
│   └── node3.suse.com.yaml
└── base-images/
    └── SL-Micro.x86_64-6.1-Base-GM.raw
```

Nota

Os nomes dos arquivos no diretório `network/` são intencionais e correspondem aos nomes de host que serão definidos durante o processo de provisionamento.

12.5.6 Criando a imagem de sistema operacional

Agora que todas as configurações necessárias foram definidas, podemos criar a imagem executando simplesmente:

```
podman run --rm -it -v $CONFIG_DIR:/eib registry.suse.com/edge/3.3/edge-image-
builder:1.2.1 build --definition-file definition.yaml
```

A saída deve ter uma aparência semelhante a esta:

```
Generating image customization components...
Identifier ................... [SUCCESS]
Custom Files ................. [SKIPPED]
Time ......................... [SKIPPED]
Network ...................... [SUCCESS]
Groups ....................... [SKIPPED]
Users ........................ [SUCCESS]
Proxy ........................ [SKIPPED]
Rpm .......................... [SKIPPED]
Systemd ...................... [SKIPPED]
Elemental .................... [SKIPPED]
Suma ......................... [SKIPPED]
```

```
Embedded Artifact Registry ... [SKIPPED]
Keymap ..................... [SUCCESS]
Kubernetes ................. [SKIPPED]
Certificates ............... [SKIPPED]
Building RAW image...
Kernel Params .............. [SKIPPED]
Image build complete!
```

O trecho acima nos informa que o componente `Network` foi configurado com sucesso, e podemos prosseguir com o provisionamento dos nós de borda.

Nota

É possível inspecionar um arquivo de registro (`network-config.log`) e os respectivos arquivos de conexão do NetworkManager no diretório `_build` resultante abaixo do diretório com a marcação de data e hora da execução da imagem.

12.5.7 Provisionando os nós de borda

Vamos copiar a imagem RAW resultante:

```
mkdir edge-nodes && cd edge-nodes
for i in {1..4}; do cp $CONFIG_DIR/modified-image.raw node$i.raw; done
```

Veja que copiamos a imagem criada quatro vezes, mas especificamos apenas as configurações de rede para três nós. O motivo para isso é que também queremos demonstrar o que acontece quando provisionamos um nó que não corresponde a nenhuma das configurações desejadas.

Nota

Este guia usará a virtualização nos exemplos de provisionamento de nós. Assegure que as extensões necessárias estejam habilitadas no BIOS (consulte aqui (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-virt-support.html#sec-kvm-requires-hardware) para obter detalhes).

Vamos usar `virt-install` para criar máquinas virtuais por meio dos discos brutos copiados. Cada máquina virtual usará 10 GB de RAM e 6 vCPUs.

12.5.7.1 Provisionando o primeiro nó

Vamos criar a máquina virtual:

```
virt-install --name node1 --ram 10000 --vcpus 6 --disk path=node1.raw,format=raw --osinfo
 detect=on,name=sle-unknown --graphics none --console pty,target_type=serial --network
 default,mac=34:8A:B1:4B:16:E1 --network default,mac=34:8A:B1:4B:16:E2 --virt-type kvm --
import
```

Nota

É importante criar as interfaces de rede com os mesmos endereços MAC daqueles no estado desejado descrito acima.

Após o término da operação, veremos algo semelhante ao seguinte:

```
Starting install...
Creating domain...

Running text console command: virsh --connect qemu:///system console node1
Connected to domain 'node1'
Escape character is ^] (Ctrl + ])


Welcome to SUSE Linux Micro 6.0 (x86_64) - Kernel 6.4.0-18-default (tty1).

SSH host key: SHA256:XN/R5Tw43reG+QsOw480LxCnhkc/1uqMdwlI6KUBY70 (RSA)
SSH host key: SHA256:/96yGrPGKlhn04f1rb9cXv/2WJt4TtrIN5yEcN66r3s (DSA)
SSH host key: SHA256:Dy/YjBQ7LwjZGaaVcMhTWZNSOstxXBsPsvgJTJq5t00 (ECDSA)
SSH host key: SHA256:TNGqY1LRddpxD/jn/8dkT/9YmVl9hiwulqmayP+wOWQ (ED25519)
eth0: 192.168.122.50
eth1:


Configured with the Edge Image Builder
Activate the web console with: systemctl enable --now cockpit.socket

node1 login:
```

Agora podemos fazer login com o par de credenciais `root:eib` e também usar SSH para entrar no host, se for a preferência no lugar de `virsh console` apresentado aqui.

Após o login, vamos confirmar se todas as configurações estão corretas.

Verifique se o nome de host foi devidamente definido:

```
node1:~ # hostnamectl
```

```
 Static hostname: node1.suse.com
 ...
```

Verifique se o roteamento foi devidamente configurado:

```
node1:~ # ip r
default via 192.168.122.1 dev eth0 proto static metric 100
192.168.122.0/24 dev eth0 proto static scope link metric 100
192.168.122.0/24 dev eth0 proto kernel scope link src 192.168.122.50 metric 100
```

Verifique se a conexão com a Internet está disponível:

```
node1:~ # ping google.com
PING google.com (142.250.72.78) 56(84) bytes of data.
64 bytes from den16s09-in-f14.1e100.net (142.250.72.78): icmp_seq=1 ttl=56 time=13.2 ms
64 bytes from den16s09-in-f14.1e100.net (142.250.72.78): icmp_seq=2 ttl=56 time=13.4 ms
^C
--- google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 13.248/13.304/13.361/0.056 ms
```

Verifique se exatamente duas interfaces Ethernet foram configuradas e apenas uma delas está ativa:

```
node1:~ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
 default qlen 1000
    link/ether 34:8a:b1:4b:16:e1 brd ff:ff:ff:ff:ff:ff
    altname enp0s2
    altname ens2
    inet 192.168.122.50/24 brd 192.168.122.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
 default qlen 1000
    link/ether 34:8a:b1:4b:16:e2 brd ff:ff:ff:ff:ff:ff
    altname enp0s3
    altname ens3

node1:~ # nmcli -f NAME,UUID,TYPE,DEVICE,FILENAME con show
NAME  UUID                                  TYPE      DEVICE  FILENAME
eth0  dfd202f5-562f-5f07-8f2a-a7717756fb70  ethernet  eth0    /etc/NetworkManager/system-
connections/eth0.nmconnection
```

```
eth1  7e211aea-3d14-59cf-a4fa-be91dac5dbba  ethernet  --      /etc/NetworkManager/system-
connections/eth1.nmconnection
```

Veja que a segunda interface é `eth1` em vez do `eth3` predefinido em nosso estado de rede desejado. O motivo é que o NetworkManager Configurator (*nmc*) é capaz de detectar que o sistema operacional forneceu um nome diferente para o NIC com o endereço MAC `34:8a:b1:4b:16:e2` e ele ajusta suas configurações de maneira apropriada.

Verifique se foi isso mesmo que aconteceu analisando a fase Combustion do provisionamento:

```
node1:~ # journalctl -u combustion | grep nmc
Apr 23 09:20:19 localhost.localdomain combustion[1360]: [2024-04-23T09:20:19Z INFO
 nmc::apply_conf] Identified host: node1.suse.com
Apr 23 09:20:19 localhost.localdomain combustion[1360]: [2024-04-23T09:20:19Z INFO
 nmc::apply_conf] Set hostname: node1.suse.com
Apr 23 09:20:19 localhost.localdomain combustion[1360]: [2024-04-23T09:20:19Z INFO
 nmc::apply_conf] Processing interface 'eth0'...
Apr 23 09:20:19 localhost.localdomain combustion[1360]: [2024-04-23T09:20:19Z INFO
 nmc::apply_conf] Processing interface 'eth3'...
Apr 23 09:20:19 localhost.localdomain combustion[1360]: [2024-04-23T09:20:19Z INFO
 nmc::apply_conf] Using interface name 'eth1' instead of the preconfigured 'eth3'
Apr 23 09:20:19 localhost.localdomain combustion[1360]: [2024-04-23T09:20:19Z INFO  nmc]
 Successfully applied config
```

Vamos agora provisionar o restante dos nós, mas vamos mostrar apenas as diferenças na configuração final. Aplique qualquer uma ou todas as verificações acima a todos os nós que você vai provisionar.

12.5.7.2 Provisionando o segundo nó

Vamos criar a máquina virtual:

```
virt-install --name node2 --ram 10000 --vcpus 6 --disk path=node2.raw,format=raw --osinfo
 detect=on,name=sle-unknown --graphics none --console pty,target_type=serial --network
 default,mac=34:8A:B1:4B:16:E3 --network default,mac=34:8A:B1:4B:16:E4 --virt-type kvm --
import
```

Com a máquina virtual em funcionamento, podemos confirmar se esse nó usa as interfaces vinculadas:

```
node2:~ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
```

```
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master bond99
 state UP group default qlen 1000
    link/ether 34:8a:b1:4b:16:e3 brd ff:ff:ff:ff:ff:ff
    altname enp0s2
    altname ens2
3: eth1: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master bond99
 state UP group default qlen 1000
    link/ether 34:8a:b1:4b:16:e3 brd ff:ff:ff:ff:ff:ff permaddr 34:8a:b1:4b:16:e4
    altname enp0s3
    altname ens3
4: bond99: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
 default qlen 1000
    link/ether 34:8a:b1:4b:16:e3 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.60/24 brd 192.168.122.255 scope global noprefixroute bond99
       valid_lft forever preferred_lft forever
```

Confirme se o roteamento usa o vínculo:

```
node2:~ # ip r
default via 192.168.122.1 dev bond99 proto static metric 100
192.168.122.0/24 dev bond99 proto static scope link metric 100
192.168.122.0/24 dev bond99 proto kernel scope link src 192.168.122.60 metric 300
```

Garanta que os arquivos de conexão estáticos sejam devidamente utilizados:

```
node2:~ # nmcli -f NAME,UUID,TYPE,DEVICE,FILENAME con show
NAME    UUID                                  TYPE      DEVICE  FILENAME
bond99  4a920503-4862-5505-80fd-4738d07f44c6  bond      bond99  /etc/NetworkManager/
system-connections/bond99.nmconnection
eth0    dfd202f5-562f-5f07-8f2a-a7717756fb70  ethernet  eth0    /etc/NetworkManager/
system-connections/eth0.nmconnection
eth1    0523c0a1-5f5e-5603-bcf2-68155d5d322e  ethernet  eth1    /etc/NetworkManager/
system-connections/eth1.nmconnection
```

12.5.7.3 Provisionando o terceiro nó

Vamos criar a máquina virtual:

```
virt-install --name node3 --ram 10000 --vcpus 6 --disk path=node3.raw,format=raw --osinfo
 detect=on,name=sle-unknown --graphics none --console pty,target_type=serial --network
 default,mac=34:8A:B1:4B:16:E5 --virt-type kvm --import
```

Com a máquina virtual em funcionamento, podemos confirmar se esse nó usa uma ponte de rede:

```
node3:~ # ip a
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master linux-br0
 state UP group default qlen 1000
    link/ether 34:8a:b1:4b:16:e5 brd ff:ff:ff:ff:ff:ff
    altname enp0s2
    altname ens2
3: linux-br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
 default qlen 1000
    link/ether 34:8a:b1:4b:16:e5 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.70/24 brd 192.168.122.255 scope global noprefixroute linux-br0
       valid_lft forever preferred_lft forever
```

Confirme se o roteamento usa a ponte:

```
node3:~ # ip r
default via 192.168.122.1 dev linux-br0 proto static metric 100
192.168.122.0/24 dev linux-br0 proto static scope link metric 100
192.168.122.0/24 dev linux-br0 proto kernel scope link src 192.168.122.70 metric 425
```

Garanta que os arquivos de conexão estáticos sejam devidamente utilizados:

```
node3:~ # nmcli -f NAME,UUID,TYPE,DEVICE,FILENAME con show
NAME       UUID                                  TYPE      DEVICE     FILENAME
linux-br0  1f8f1469-ed20-5f2c-bacb-a6767bee9bc0  bridge    linux-br0  /etc/
NetworkManager/system-connections/linux-br0.nmconnection
eth0       dfd202f5-562f-5f07-8f2a-a7717756fb70  ethernet  eth0       /etc/
NetworkManager/system-connections/eth0.nmconnection
```

12.5.7.4 Provisionando o quarto nó

Por fim, vamos provisionar um nó que não corresponderá a nenhuma das configurações predefinidas por um endereço MAC. Nesse caso, vamos usar DHCP como padrão para configurar as interfaces de rede.

Vamos criar a máquina virtual:

```
virt-install --name node4 --ram 10000 --vcpus 6 --disk path=node4.raw,format=raw --osinfo
 detect=on,name=sle-unknown --graphics none --console pty,target_type=serial --network
 default --virt-type kvm --import
```

Com a máquina virtual em funcionamento, podemos confirmar se esse nó usa um endereço IP aleatório para a interface de rede:

```
localhost:~ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
 default qlen 1000
    link/ether 52:54:00:56:63:71 brd ff:ff:ff:ff:ff:ff
    altname enp0s2
    altname ens2
    inet 192.168.122.86/24 brd 192.168.122.255 scope global dynamic noprefixroute eth0
       valid_lft 3542sec preferred_lft 3542sec
    inet6 fe80::5054:ff:fe56:6371/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Verifique se o NMC não consegue aplicar configurações estáticas a esse nó:

```
localhost:~ # journalctl -u combustion | grep nmc
Apr 23 12:15:45 localhost.localdomain combustion[1357]: [2024-04-23T12:15:45Z ERROR nmc]
 Applying config failed: None of the preconfigured hosts match local NICs
```

Verifique se a interface Ethernet foi configurada por DHCP:

```
localhost:~ # journalctl | grep eth0
Apr 23 12:15:29 localhost.localdomain NetworkManager[704]: <info>  [1713874529.7801]
 manager: (eth0): new Ethernet device (/org/freedesktop/NetworkManager/Devices/2)
Apr 23 12:15:29 localhost.localdomain NetworkManager[704]: <info>  [1713874529.7802]
 device (eth0): state change: unmanaged -> unavailable (reason 'managed', sys-iface-
state: 'external')
Apr 23 12:15:29 localhost.localdomain NetworkManager[704]: <info>  [1713874529.7929]
 device (eth0): carrier: link connected
Apr 23 12:15:29 localhost.localdomain NetworkManager[704]: <info>  [1713874529.7931]
 device (eth0): state change: unavailable -> disconnected (reason 'carrier-changed', sys-
iface-state: 'managed')
Apr 23 12:15:29 localhost.localdomain NetworkManager[704]: <info>
 [1713874529.7944] device (eth0): Activation: starting connection 'Wired
 Connection' (300ed658-08d4-4281-9f8c-d1b8882d29b9)
Apr 23 12:15:29 localhost.localdomain NetworkManager[704]: <info>  [1713874529.7945]
 device (eth0): state change: disconnected -> prepare (reason 'none', sys-iface-state:
 'managed')
```

```
Apr 23 12:15:29 localhost.localdomain NetworkManager[704]: <info>  [1713874529.7947]
 device (eth0): state change: prepare -> config (reason 'none', sys-iface-state:
 'managed')
Apr 23 12:15:29 localhost.localdomain NetworkManager[704]: <info>  [1713874529.7953]
 device (eth0): state change: config -> ip-config (reason 'none', sys-iface-state:
 'managed')
Apr 23 12:15:29 localhost.localdomain NetworkManager[704]: <info>  [1713874529.7964]
 dhcp4 (eth0): activation: beginning transaction (timeout in 90 seconds)
Apr 23 12:15:33 localhost.localdomain NetworkManager[704]: <info>  [1713874533.1272]
 dhcp4 (eth0): state changed new lease, address=192.168.122.86

localhost:~ # nmcli -f NAME,UUID,TYPE,DEVICE,FILENAME con show
NAME              UUID                                  TYPE      DEVICE  FILENAME
Wired Connection  300ed658-08d4-4281-9f8c-d1b8882d29b9  ethernet  eth0    /var/run/
NetworkManager/system-connections/default_connection.nmconnection
```

12.5.8 Configurações unificadas de nós

Há situações em que não é possível usar endereços MAC conhecidos. Nesses casos, podemos recorrer à *configuração unificada*, que permite especificar as configurações em um arquivo `_all.yaml` que será aplicado a todos os nós provisionados.

Vamos criar e provisionar um nó de borda usando uma estrutura de configuração diferente. Siga todas as etapas, desde a *Seção 12.5.3, "Criando o diretório de configuração de imagem"* até a *Seção 12.5.5, "Definindo as configurações de rede"*.

Neste exemplo, definimos um estado desejado de duas interfaces Ethernet (eth0 e eth1): uma usando DHCP e outra que recebeu um endereço IP estático.

```
mkdir -p $CONFIG_DIR/network

cat <<- EOF > $CONFIG_DIR/network/_all.yaml
interfaces:
- name: eth0
  type: ethernet
  state: up
  ipv4:
    dhcp: true
    enabled: true
  ipv6:
    enabled: false
- name: eth1
  type: ethernet
  state: up
  ipv4:
```

```
    address:
    - ip: 10.0.0.1
      prefix-length: 24
    enabled: true
    dhcp: false
  ipv6:
    enabled: false
EOF
```

Vamos criar a imagem:

```
podman run --rm -it -v $CONFIG_DIR:/eib registry.suse.com/edge/3.3/edge-image-
builder:1.2.1 build --definition-file definition.yaml
```

Depois que a imagem for criada com sucesso, vamos criar uma máquina virtual com base nela:

```
virt-install --name node1 --ram 10000 --vcpus 6 --disk path=$CONFIG_DIR/modified-
image.raw,format=raw --osinfo detect=on,name=sle-unknown --graphics none --console
 pty,target_type=serial --network default --network default --virt-type kvm --import
```

O processo de provisionamento pode levar alguns minutos. Depois que ele for concluído, faça login no sistema com as credenciais fornecidas.

Verifique se o roteamento foi devidamente configurado:

```
localhost:~ # ip r
default via 192.168.122.1 dev eth0 proto dhcp src 192.168.122.100 metric 100
10.0.0.0/24 dev eth1 proto kernel scope link src 10.0.0.1 metric 101
192.168.122.0/24 dev eth0 proto kernel scope link src 192.168.122.100 metric 100
```

Verifique se a conexão com a Internet está disponível:

```
localhost:~ # ping google.com
PING google.com (142.250.72.46) 56(84) bytes of data.
64 bytes from den16s08-in-f14.1e100.net (142.250.72.46): icmp_seq=1 ttl=56 time=14.3 ms
64 bytes from den16s08-in-f14.1e100.net (142.250.72.46): icmp_seq=2 ttl=56 time=14.2 ms
^C
--- google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 14.196/14.260/14.324/0.064 ms
```

Verifique se as interfaces Ethernet estão configuradas e ativas:

```
localhost:~ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
```

```
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
 default qlen 1000
    link/ether 52:54:00:26:44:7a brd ff:ff:ff:ff:ff:ff
    altname enp1s0
    inet 192.168.122.100/24 brd 192.168.122.255 scope global dynamic noprefixroute eth0
       valid_lft 3505sec preferred_lft 3505sec
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
 default qlen 1000
    link/ether 52:54:00:ec:57:9e brd ff:ff:ff:ff:ff:ff
    altname enp7s0
    inet 10.0.0.1/24 brd 10.0.0.255 scope global noprefixroute eth1
       valid_lft forever preferred_lft forever

localhost:~ # nmcli -f NAME,UUID,TYPE,DEVICE,FILENAME con show
NAME  UUID                                  TYPE      DEVICE  FILENAME
eth0  dfd202f5-562f-5f07-8f2a-a7717756fb70  ethernet  eth0    /etc/NetworkManager/system-
connections/eth0.nmconnection
eth1  0523c0a1-5f5e-5603-bcf2-68155d5d322e  ethernet  eth1    /etc/NetworkManager/system-
connections/eth1.nmconnection

localhost:~ # cat /etc/NetworkManager/system-connections/eth0.nmconnection
[connection]
autoconnect=true
autoconnect-slaves=-1
id=eth0
interface-name=eth0
type=802-3-ethernet
uuid=dfd202f5-562f-5f07-8f2a-a7717756fb70

[ipv4]
dhcp-client-id=mac
dhcp-send-hostname=true
dhcp-timeout=2147483647
ignore-auto-dns=false
ignore-auto-routes=false
method=auto
never-default=false

[ipv6]
addr-gen-mode=0
dhcp-timeout=2147483647
method=disabled

localhost:~ # cat /etc/NetworkManager/system-connections/eth1.nmconnection
[connection]
autoconnect=true
```

```
autoconnect-slaves=-1
id=eth1
interface-name=eth1
type=802-3-ethernet
uuid=0523c0a1-5f5e-5603-bcf2-68155d5d322e

[ipv4]
address0=10.0.0.1/24
dhcp-timeout=2147483647
method=manual

[ipv6]
addr-gen-mode=0
dhcp-timeout=2147483647
method=disabled
```

12.5.9 Configurações de rede personalizadas

Já abordamos a configuração de rede padrão do Edge Image Builder que usa o NetworkManager Configurator. No entanto, há também a opção de modificá-la por meio de um script personalizado. Essa opção é muito flexível e independente de endereço MAC, mas tem a limitação de que seu uso não é tão prático para inicializar vários nós com uma única imagem.

Nota

A recomendação é usar a configuração de rede padrão por meio dos arquivos que descrevem o estado da rede desejado no diretório `/network`. Crie scripts personalizados apenas quando esse comportamento não é cabível no seu caso de uso.

Vamos criar e provisionar um nó de borda usando uma estrutura de configuração diferente. Siga todas as etapas, desde a *Seção 12.5.3, "Criando o diretório de configuração de imagem"* até a *Seção 12.5.5, "Definindo as configurações de rede"*.

Neste exemplo, vamos criar um script personalizado que aplica a configuração estática da interface `eth0` a todos os nós provisionados, além de remover e desabilitar as conexões com fio criadas automaticamente pelo NetworkManager. Isso é vantajoso nos casos em que você quer garantir que cada nó no cluster tenha uma configuração de rede idêntica e, sendo assim, você não precisa se preocupar com o endereço MAC de cada nó antes da criação da imagem.

Para começar, vamos armazenar o arquivo de conexão no diretório `/custom/files`:

```
mkdir -p $CONFIG_DIR/custom/files

cat << EOF > $CONFIG_DIR/custom/files/eth0.nmconnection
[connection]
autoconnect=true
autoconnect-slaves=-1
autoconnect-retries=1
id=eth0
interface-name=eth0
type=802-3-ethernet
uuid=dfd202f5-562f-5f07-8f2a-a7717756fb70
wait-device-timeout=60000

[ipv4]
dhcp-timeout=2147483647
method=auto

[ipv6]
addr-gen-mode=eui64
dhcp-timeout=2147483647
method=disabled
EOF
```

Com a configuração estática criada, vamos também criar nosso script de rede personalizado:

```
mkdir -p $CONFIG_DIR/network

cat << EOF > $CONFIG_DIR/network/configure-network.sh
#!/bin/bash
set -eux

# Remove and disable wired connections
mkdir -p /etc/NetworkManager/conf.d/
printf "[main]\nno-auto-default=*\n" > /etc/NetworkManager/conf.d/no-auto-default.conf
rm -f /var/run/NetworkManager/system-connections/* || true

# Copy pre-configured network configuration files into NetworkManager
mkdir -p /etc/NetworkManager/system-connections/
cp eth0.nmconnection /etc/NetworkManager/system-connections/
chmod 600 /etc/NetworkManager/system-connections/*.nmconnection
EOF

chmod a+x $CONFIG_DIR/network/configure-network.sh
```

Nota

Por padrão, o binário nmc ainda será incluído, para ser usado no script `configure-network.sh` se necessário.

Atenção

O script personalizado deve sempre ser inserido em `/network/configure-network.sh` no diretório de configuração. Se estiver presente, todos os outros arquivos serão ignorados. NÃO é possível configurar uma rede trabalhando com as configurações estáticas no formato YAML e um script personalizado ao mesmo tempo.

Neste momento, o diretório de configuração deve ter a seguinte aparência:

```
├── definition.yaml
├── custom/
│   └── files/
│       └── eth0.nmconnection
├── network/
│   └── configure-network.sh
└── base-images/
    └── SL-Micro.x86_64-6.1-Base-GM.raw
```

Vamos criar a imagem:

```
podman run --rm -it -v $CONFIG_DIR:/eib registry.suse.com/edge/3.3/edge-image-
builder:1.2.1 build --definition-file definition.yaml
```

Depois que a imagem for criada com sucesso, vamos criar uma máquina virtual com base nela:

```
virt-install --name node1 --ram 10000 --vcpus 6 --disk path=$CONFIG_DIR/modified-
image.raw,format=raw --osinfo detect=on,name=sle-unknown --graphics none --console
 pty,target_type=serial --network default --virt-type kvm --import
```

O processo de provisionamento pode levar alguns minutos. Depois que ele for concluído, faça login no sistema com as credenciais fornecidas.

Verifique se o roteamento foi devidamente configurado:

```
localhost:~ # ip r
default via 192.168.122.1 dev eth0 proto dhcp src 192.168.122.185 metric 100
192.168.122.0/24 dev eth0 proto kernel scope link src 192.168.122.185 metric 100
```

Verifique se a conexão com a Internet está disponível:

```
localhost:~ # ping google.com
PING google.com (142.250.72.78) 56(84) bytes of data.
64 bytes from den16s09-in-f14.1e100.net (142.250.72.78): icmp_seq=1 ttl=56 time=13.6 ms
64 bytes from den16s09-in-f14.1e100.net (142.250.72.78): icmp_seq=2 ttl=56 time=13.6 ms
^C
--- google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 13.592/13.599/13.606/0.007 ms
```

Verifique se uma interface Ethernet foi configurada estaticamente usando nosso arquivo de conexão e se ela está ativa:

```
localhost:~ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
 default qlen 1000
    link/ether 52:54:00:31:d0:1b brd ff:ff:ff:ff:ff:ff
    altname enp0s2
    altname ens2
    inet 192.168.122.185/24 brd 192.168.122.255 scope global dynamic noprefixroute eth0

localhost:~ # nmcli -f NAME,UUID,TYPE,DEVICE,FILENAME con show
NAME  UUID                                  TYPE      DEVICE  FILENAME
eth0  dfd202f5-562f-5f07-8f2a-a7717756fb70  ethernet  eth0    /etc/NetworkManager/system-
connections/eth0.nmconnection

localhost:~ # cat  /etc/NetworkManager/system-connections/eth0.nmconnection
[connection]
autoconnect=true
autoconnect-slaves=-1
autoconnect-retries=1
id=eth0
interface-name=eth0
type=802-3-ethernet
uuid=dfd202f5-562f-5f07-8f2a-a7717756fb70
wait-device-timeout=60000

[ipv4]
dhcp-timeout=2147483647
```

```
method=auto

[ipv6]
addr-gen-mode=eui64
dhcp-timeout=2147483647
method=disabled
```

13 Elemental

Elemental é uma pilha de software que possibilita o gerenciamento de sistema operacional nativo de nuvem e totalmente centralizado com o Kubernetes. A pilha Elemental consiste em vários componentes que residem no próprio Rancher ou em nós de borda. Os componentes principais são:

- **elemental-operator** : o operador principal que reside no Rancher e processa as solicitações de registro dos clientes.
- **elemental-register**: o cliente executado em nós de borda que possibilita o registro por meio do `elemental-operator`.
- **elemental-system-agent**: um agente que reside em nós de borda. Sua configuração é alimentada pelo `elemental-register` e recebe um `plan` para configurar o `rancher-system-agent`.
- **rancher-system-agent**: após o registro completo do nó de borda, esse componente assume o comando depois do `elemental-system-agent` e aguarda outros `plans` do Rancher Manager (por exemplo, instalação do Kubernetes).

Consulte a documentação upstream do Elemental (https://elemental.docs.rancher.com/) para obter todas as informações sobre o Elemental e seu relacionamento com o Rancher.

13.1 Como o SUSE Edge usa o Elemental?

Usamos partes do Elemental para gerenciar dispositivos remotos quando o Metal[3] não é viável (por exemplo, não existe BMC ou o dispositivo está protegido por um gateway NAT). Essa ferramenta permite que o operador inicialize os dispositivos em laboratório antes de saber quando ou para onde serão enviados. Especificamente, usamos os componentes `elemental-register` e `elemental-system-agent` para permitir a integração de hosts do SUSE Linux Micro no Rancher para casos de uso de provisionamento de rede "phone home". Quando o Edge Image Builder (EIB) é usado para criar imagens de implantação, o registro automático pelo Rancher via Elemental pode ser feito especificando a configuração de registro no diretório de configuração do EIB.

Nota

No SUSE Edge 3.3.1, **não** aproveitamos os aspectos de gerenciamento de sistema operacional do Elemental e, portanto, não é possível gerenciar a aplicação de patches de seu sistema operacional usando o Rancher. Em vez de usar as ferramentas do Elemental para criar imagens de implantação, o SUSE Edge usa as ferramentas do Edge Image Builder, que consome a configuração de registro.

13.2 Melhores práticas

13.2.1 Mídia de instalação

A maneira recomendada do SUSE Edge para criar imagens de implantação que possam usar o Elemental para registro no Rancher na área de implantação de "provisionamento de rede 'phone home'" é seguir as instruções detalhadas no início rápido sobre integração de host remoto com o Elemental (*Capítulo 2, Integração remota de host com o Elemental*).

13.2.2 Rótulos

O Elemental monitora o inventário com a CRD `MachineInventory` e permite selecionar um inventário, por exemplo, para selecionar máquinas nas quais implantar clusters Kubernetes com base em rótulos. Dessa forma, os usuários podem predefinir grande parte (se não tudo) de suas necessidades de infraestrutura antes mesmo da compra do hardware. Além disso, como os nós podem adicionar/remover rótulos de seu respectivo objeto de inventário (executando novamente o `elemental-register` com o sinalizador adicional `--label "FOO=BAR"`), podemos escrever scripts para descobrir e permitir que o Rancher saiba onde um nó é inicializado.

13.3 Problemas conhecidos

- Atualmente, a IU do Elemental não sabe como criar uma mídia de instalação ou atualizar sistemas operacionais não "Elemental Teal". Isso deve ser resolvido em versões futuras.

14 Akri

Akri é um projeto Sandbox da CNCF que visa a descobrir dispositivos folha para apresentá-los como recursos nativos do Kubernetes. Ele também permite programar um pod ou job para cada dispositivo descoberto. Os dispositivos podem ser de nó local ou de rede e usar uma ampla variedade de protocolos.

A documentação upstream do Akri está disponível em: https://docs.akri.sh

14.1 Como o SUSE Edge usa o Akri?

Atenção

Atualmente, o Akri está em prévia de tecnologia na pilha do SUSE Edge.

O Akri está disponível como parte da pilha do Edge sempre que é preciso descobrir e programar uma carga de trabalho com base nos dispositivos folha.

14.2 Instalando o Akri

O Akri está disponível como gráfico Helm no repositório Helm do Edge. A maneira recomendada de configurar o Akri é usar o gráfico Helm especificado para implantar os diversos componentes (agente, controlador, manipuladores de descoberta) e usar o mecanismo de implantação de sua preferência para implantar as CRDs de configuração do Akri.

14.3 Configurando o Akri

O Akri é configurado usando um objeto `akri.sh/Configuration`, que obtém todas as informações de como descobrir os dispositivos e do que fazer quando um correspondente é descoberto.

Veja abaixo um exemplo de configuração decomposto com a explicação de todos os campos:

```
apiVersion: akri.sh/v0
```

```
kind: Configuration
metadata:
  name: sample-configuration
spec:
```

Esta parte descreve a configuração do manipulador de descoberta. Você deve especificar o nome dele (os manipuladores disponíveis como parte do gráfico do Akri são `udev`, `opcua` e `onvif`). O `discoveryDetails` é específico do manipulador. Consulte a documentação do manipulador para saber como configurá-lo.

```
  discoveryHandler:
    name: debugEcho
    discoveryDetails: |+
      descriptions:
        - "foo"
        - "bar"
```

Esta seção define a carga de trabalho que será implantada para cada dispositivo descoberto. O exemplo mostra uma versão mínima da configuração do `Pod` em `brokerPodSpec`, todos os campos comuns da especificação de um pod podem ser usados aqui. Ele também mostra a sintaxe específica do Akri para solicitar o dispositivo na seção `resources`.

Se preferir, use um job no lugar do pod, com a chave `brokerJobSpec`, e insira a parte da especificação de um job para ele.

```
  brokerSpec:
    brokerPodSpec:
      containers:
      - name: broker-container
        image: rancher/hello-world
        resources:
          requests:
            "{{PLACEHOLDER}}" : "1"
          limits:
            "{{PLACEHOLDER}}" : "1"
```

Estas duas seções mostram como configurar o Akri para implantar um serviço por agente (`instanceService`) ou apontar para todos os agentes (`configurationService`). Eles contêm todos os elementos que pertencem a um serviço comum.

```
  instanceServiceSpec:
    type: ClusterIp
    ports:
    - name: http
```

```
    port: 80
    protocol: tcp
    targetPort: 80
configurationServiceSpec:
  type: ClusterIp
  ports:
  - name: https
    port: 443
    protocol: tcp
    targetPort: 443
```

O campo `brokerProperties` é um armazenamento de chave/valor que será exposto como variáveis de ambiente adicionais para um pod que solicita um dispositivo descoberto.

A capacidade é o número permitido de usuários simultâneos de um dispositivo descoberto.

```
brokerProperties:
  key: value
capacity: 1
```

14.4 Escrevendo e implantando manipuladores de descoberta adicionais

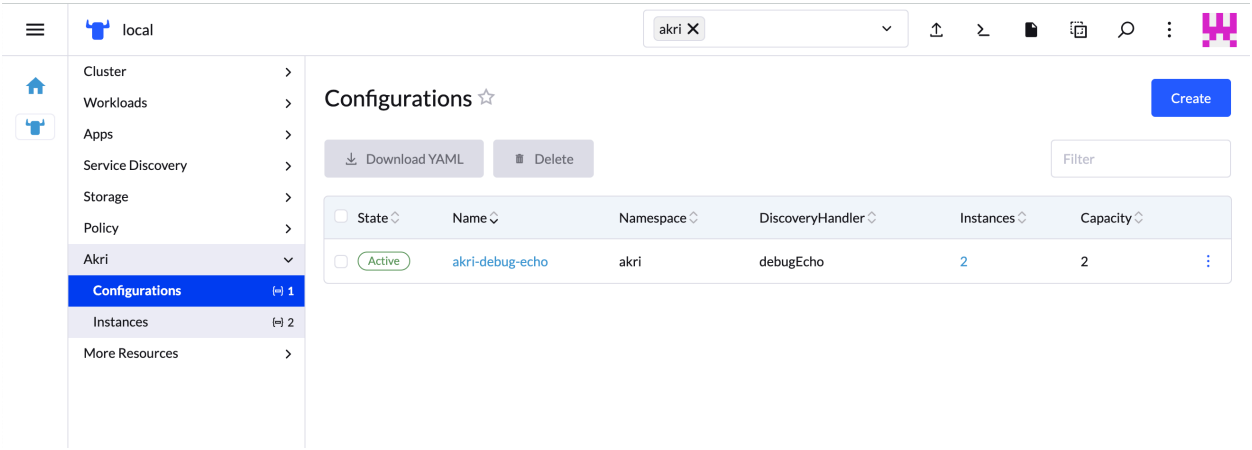
Se o protocolo usado por seu dispositivo não for coberto por um manipulador de descoberta existente, você pode escrever um próprio seguindo o guia de desenvolvimento de manipulador (https://docs.akri.sh/development/handler-development).

14.5 Extensão Akri do Rancher Dashboard

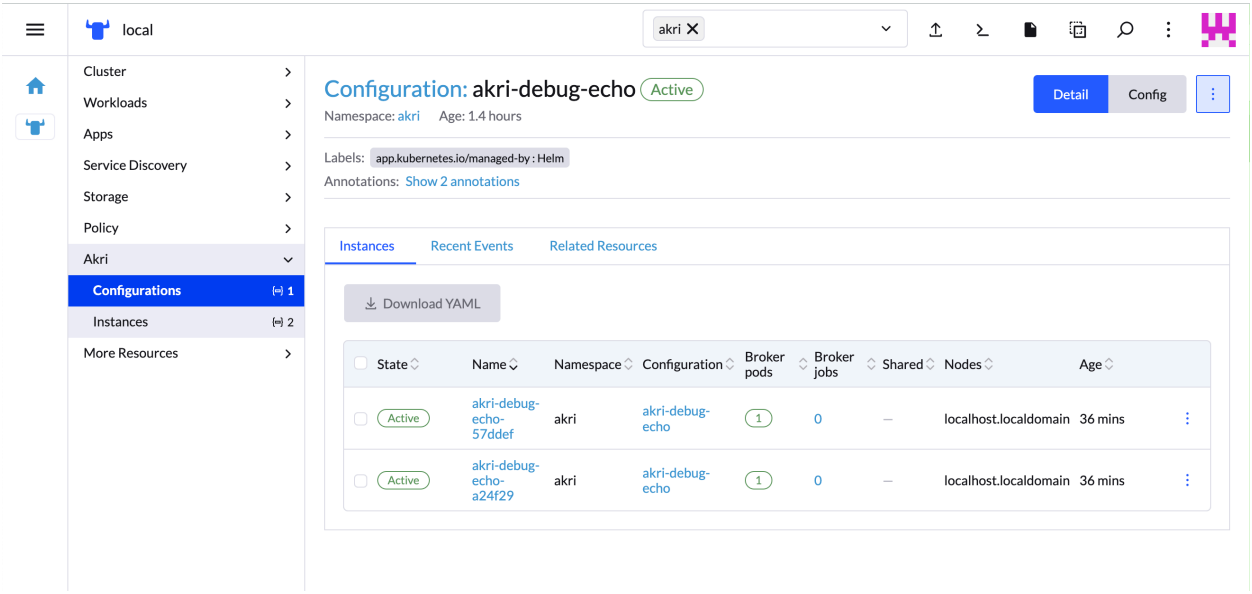
Com a extensão de dashboard Akri, você pode usar a interface de usuário do Rancher Dashboard para gerenciar e monitorar dispositivos folha e executar cargas de trabalho após a descoberta desses dispositivos.

Consulte o *Capítulo 6, Extensões do Rancher Dashboard* para ver a orientação de instalação.

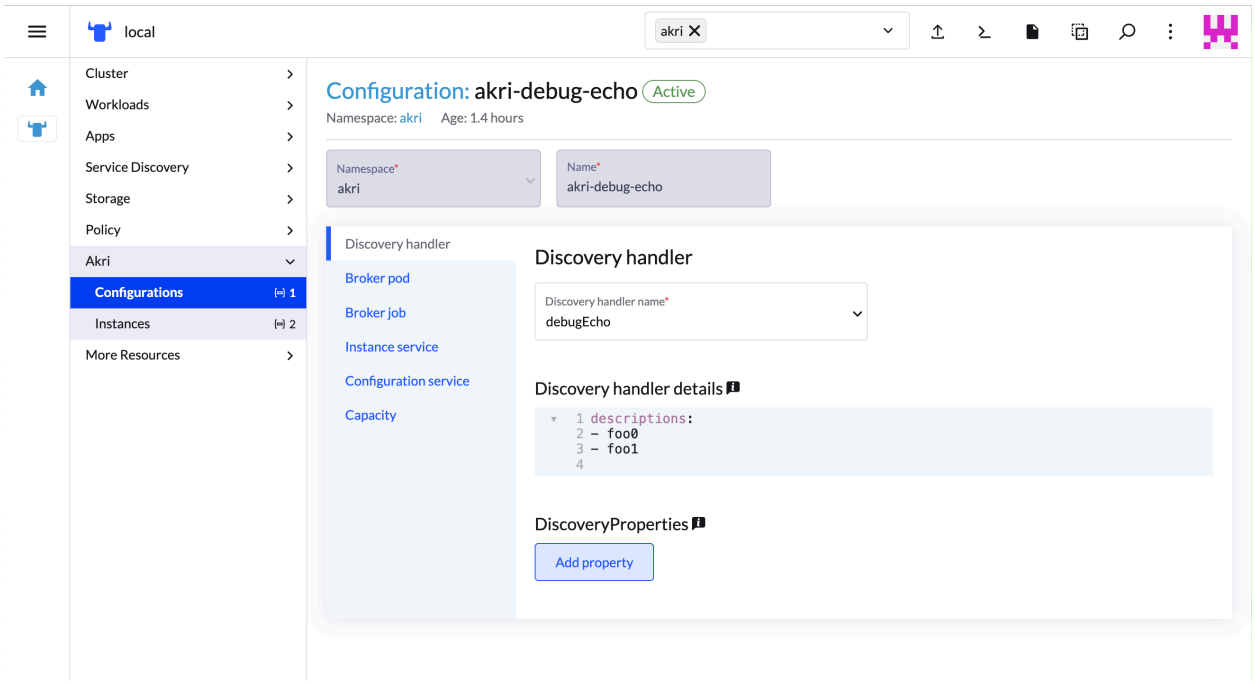
Depois que a extensão for instalada, navegue até um cluster habilitado pelo Akri usando o explorador de clusters. No grupo de navegação **Akri**, você verá as seções **Configurations** (Configurações) e **Instances** (Instâncias).



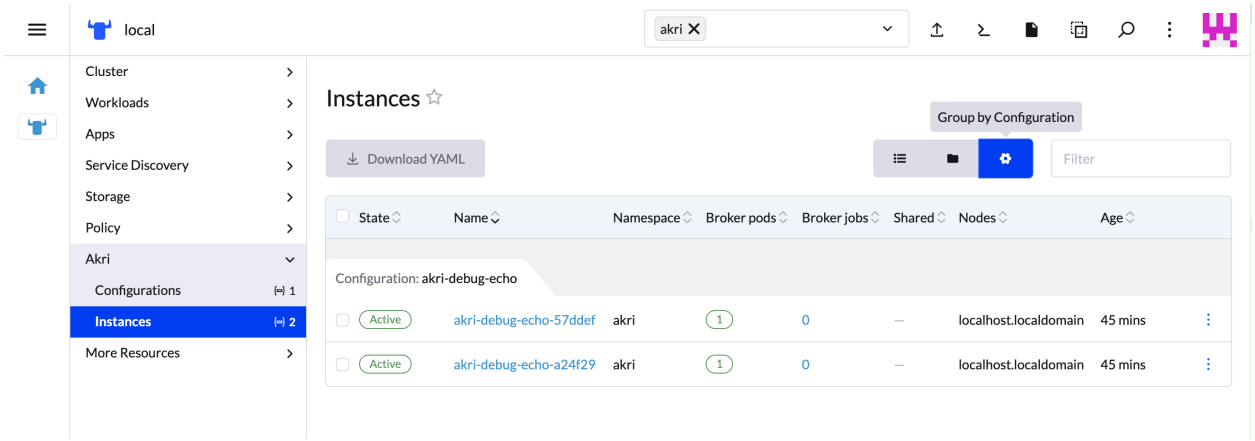
A lista de configurações apresenta informações sobre o `manipulador de descoberta de configuração` e o número de instâncias. Clique no nome para abrir a página de detalhes da configuração.



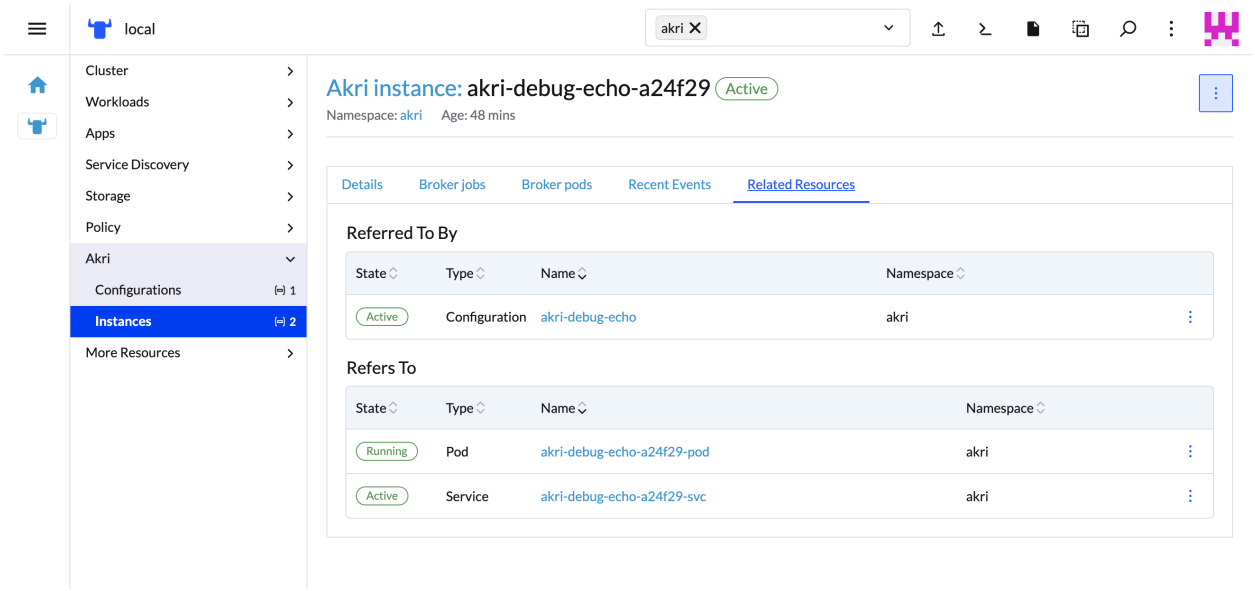
Você também pode editar ou criar uma nova **configuração**. A extensão permite selecionar o manipulador de descoberta, configurar o pod ou job do agente, personalizar configurações e serviços de instâncias e definir a capacidade da configuração.



Os dispositivos descobertos são relacionados na lista **Instances** (Instâncias).



Clique no nome da **instância** para abrir uma página de detalhes com as cargas de trabalho e o serviço da instância.


local
akri
Cluster
Workloads
Apps
Service Discovery
Storage
Policy
Akri
Configurations 1
Instances 2
More Resources
Akri instance: akri-debug-echo-a24f29 Active
Namespace: akri Age: 48 mins
Details
Broker jobs
Broker pods
Recent Events
Related Resources
Referred To By
State Type Name Namespace
Active Configuration akri-debug-echo akri
Refers To
State Type Name Namespace
Running Pod akri-debug-echo-a24f29-pod akri
Active Service akri-debug-echo-a24f29-svc akri

15 K3s

K3s (https://k3s.io/) é uma distribuição Kubernetes certificada e altamente disponível desenvolvida para cargas de trabalho de produção em locais autônomos, remotos e com restrição de recursos ou dentro de aplicações IoT.

Ele é empacotado como um binário único e pequeno, portanto, as instalações e atualizações são rápidas e fáceis.

15.1 Como o SUSE Edge usa o K3s

É possível usar o K3s como distribuição Kubernetes de base para a pilha do SUSE Edge. Ele foi projetado para ser instalado em um sistema operacional SUSE Linux Micro.

O uso do K3s como distribuição Kubernetes da pilha do SUSE Edge é recomendado apenas quando o etcd como back end não atende a suas restrições. Se o etcd como back end for possível, será melhor usar o RKE2 (*Capítulo 16, RKE2*).

15.2 Melhores práticas

15.2.1 Instalação

A maneira recomendada de instalar o K3s como parte da pilha do SUSE Edge é usar o Edge Image Builder (EIB). Consulte a documentação dele (*Capítulo 11, Edge Image Builder*) para obter mais detalhes de como configurá-lo para implantar o K3s.

Ele oferece suporte automático para configuração de alta disponibilidade e do Elemental.

15.2.2 Fleet para fluxo de trabalho do GitOps

A pilha do SUSE Edge usa o Fleet como ferramenta GitOps preferencial. Para obter mais informações sobre instalação e uso, consulte a seção do Fleet (*Capítulo 8, Fleet*) nesta documentação.

15.2.3 Gerenciamento de armazenamento

O K3s vem com armazenamento de caminho local pré-configurado, que é adequado para clusters de nó único. Para clusters que abrangem vários nós, recomendamos usar o SUSE Storage (*Capítulo 17, SUSE Storage*).

15.2.4 Balanceamento de carga e alta disponibilidade

Se você instalou o K3s com o EIB, esta parte já foi coberta pela documentação do EIB na seção de alta disponibilidade.

Do contrário, será necessário instalar e configurar o MetalLB de acordo com a nossa documentação do MetalLB (*Capítulo 25, MetalLB no K3s (usando o modo de camada 2)*).

16 RKE2

Consulte a documentação oficial do RKE2 (https://docs.rke2.io/).

O RKE2 é uma distribuição Kubernetes totalmente compatível dedicada à segurança e à conformidade das seguintes maneiras:

- Oferecendo padrões e opções de configuração para que os clusters sejam aprovados no CIS Kubernetes Benchmark v1.6 ou v1.23 com o mínimo de intervenção do operador
- Assegurando a conformidade com o FIPS 140-2
- Verificando regularmente se há CVEs nos componentes por meio do trivy (https://trivy.dev) no pipeline de compilação do RKE2

O RKE2 inicia os componentes do plano de controle como pods estáticos, gerenciados pelo kubelet. O runtime de contêiner incorporado é containerd.

Nota: O RKE2 também é conhecido como RKE Government para expressar outro caso de uso e setor a que ele atende.

16.1 RKE2 ou K3s

O K3s é uma distribuição Kubernetes leve e totalmente compatível, com foco em borda, IoT e ARM, otimizado para facilidade de uso e ambientes com restrição de recursos.

O RKE2 combina o melhor dos dois mundos: a versão 1.x do RKE (daqui em diante chamada de RKE1) e o K3s.

Do K3s, ele herda a usabilidade, a facilidade de operação e o modelo de implantação.

Do RKE1, ele herda o alinhamento próximo com o Kubernetes upstream. Em alguns locais, o K3s diverge do Kubernetes upstream para otimizar as implantações de borda, mas o RKE1 e o RKE2 mantêm um estreito alinhamento com o upstream.

16.2 Como o SUSE Edge usa o RKE2?

O RKE2 é um componente fundamental da pilha do SUSE Edge. Ele coexiste com o SUSE Linux Micro (*Capítulo 9, SUSE Linux Micro*), oferecendo a interface padrão do Kubernetes necessária para implantar cargas de trabalho do Edge.

16.3 Melhores práticas

16.3.1 Instalação

A maneira recomendada de instalar o RKE2 como parte da pilha do SUSE Edge é usar o Edge Image Builder (EIB). Consulte a documentação do EIB (*Capítulo 11, Edge Image Builder*) para obter mais detalhes de como configurá-lo para implantar o RKE2.

O EIB é flexível o suficiente para dar suporte aos parâmetros exigidos pelo RKE2, como especificar a versão do RKE2, a configuração de servidores (https://docs.rke2.io/reference/server_config) ou de agentes (https://docs.rke2.io/reference/linux_agent_config), englobando todos os casos de uso do Edge.

Para outros casos de uso envolvendo o Metal^3, o RKE2 também é usado e instalado. Nesses casos específicos, o provedor Cluster API RKE2 (https://github.com/rancher-sandbox/cluster-api-provider-rke2) implanta automaticamente o RKE2 nos clusters que são provisionados com o Metal^3 usando a pilha do Edge.

Nesses casos, a configuração do RKE2 deve ser aplicada às diversas CRDs envolvidas. Um exemplo de como fornecer uma CNI diferente usando a CRD `RKE2ControlPlane` tem esta aparência:

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
metadata:
  name: single-node-cluster
  namespace: default
spec:
  serverConfig:
    cni: calico
    cniMultusEnable: true
...
```

Para obter mais informações sobre os casos de uso do Metal^3, consulte o *Capítulo 10, Metal^3*.

16.3.2 Alta disponibilidade

Para implantações de alta disponibilidade, o EIB implanta e configura automaticamente o MetalLB (*Capítulo 19, MetalLB*) e o Endpoint Copier Operator (*Capítulo 20, Endpoint Copier Operator*) para expor o endpoint da API do RKE2 externamente.

16.3.3 Rede

A pilha do SUSE Edge suporta o Cilium (https://docs.cilium.io/en/stable/) e o Calico (https://docs.tigera.io/calico/latest/about/), com o Cilium como CNI padrão. O metaplug-in Multus (https://github.com/k8snetworkplumbingwg/multus-cni) também pode ser usado quando os pods exigem várias interfaces de rede. O RKE2 independente oferece suporte a uma ampla gama de opções de CNI (https://docs.rke2.io/install/network_options).

16.3.4 Armazenamento

O RKE2 não inclui nenhum tipo de classe ou operador de armazenamento persistente. Para clusters com vários nós, é recomendado usar o SUSE Storage (*Capítulo 17, SUSE Storage*).

17 SUSE Storage

O SUSE Storage é um sistema de armazenamento em blocos distribuído, leve, confiável e amigável projetado para Kubernetes. Trata-se de um produto com base no Longhorn, um projeto de código-fonte aberto inicialmente desenvolvido pela Rancher Labs e que agora é incubado pela CNCF.

17.1 Pré-requisitos

Se você está seguindo este guia, já deve ter os seguintes itens disponíveis:

- No mínimo, um host com o SUSE Linux Micro 6.1 instalado, que pode ser físico ou virtual
- Um cluster Kubernetes instalado, K3s ou RKE2
- Helm

17.2 Instalação manual do SUSE Storage

17.2.1 Instalando o Open-iSCSI

Um requisito essencial da implantação e do uso do SUSE Storage é a instalação do pacote `open-iscsi` e a execução do daemon `iscsid` em todos os nós Kubernetes. Isso é necessário porque o Longhorn conta com o `iscsiadm` no host para fornecer volumes persistentes ao Kubernetes.

Vamos instalá-lo:

```
transactional-update pkg install open-iscsi
```

Como o SUSE Linux Micro é um sistema operacional imutável, é importante notar que o pacote será instalado apenas em um novo instantâneo após a conclusão da operação. Para carregá-lo e para que o daemon `iscsid` seja executado, é necessário reinicializar no novo instantâneo que acabamos de criar. Execute o comando de reinicialização quando você estiver pronto:

```
reboot
```

Dica

Para obter mais ajuda na instalação do open-iscsi, consulte a documentação oficial do Longhorn (https://longhorn.io/docs/1.8.1/deploy/install/#installing-open-iscsi).

17.2.2 Instalando o SUSE Storage

Há várias maneiras de instalar o SUSE Storage em clusters Kubernetes. Este guia adota a instalação pelo Helm, mas você poderá seguir a documentação oficial (https://longhorn.io/docs/1.8.1/deploy/install/) caso queira outra abordagem.

1. Adicione o repositório de gráficos Helm do Rancher:

```
helm repo add rancher-charts https://charts.rancher.io/
```

2. Busque os gráficos mais recentes no repositório:

```
helm repo update
```

3. Instale o SUSE Storage no namespace `longhorn-system`:

```
helm install longhorn-crd rancher-charts/longhorn-crd --namespace longhorn-system --
create-namespace --version 106.2.0+up1.8.1
helm install longhorn rancher-charts/longhorn --namespace longhorn-system --version
 106.2.0+up1.8.1
```

4. Confirme se a implantação foi bem-sucedida:

```
kubectl -n longhorn-system get pods
```

```
localhost:~ # kubectl -n longhorn-system get pod
NAMESPACE         NAME                                                READY   STATUS
      RESTARTS        AGE
longhorn-system   longhorn-ui-5fc9fb76db-z5dc9                        1/1
 Running     0               90s
longhorn-system   longhorn-ui-5fc9fb76db-dcb65                        1/1
 Running     0               90s
longhorn-system   longhorn-manager-wts2v                              1/1
 Running     1 (77s ago)     90s
longhorn-system   longhorn-driver-deployer-5d4f79ddd-fxgcs            1/1
 Running     0               90s
longhorn-system   instance-manager-a9bf65a7808a1acd6616bcd4c03d925b   1/1
 Running     0               70s
```

```
longhorn-system   engine-image-ei-acb7590c-htqmp                    1/1
 Running     0              70s
longhorn-system   csi-attacher-5c4bfdcf59-j8xww                     1/1
 Running     0              50s
longhorn-system   csi-provisioner-667796df57-l69vh                  1/1
 Running     0              50s
longhorn-system   csi-attacher-5c4bfdcf59-xgd5z                     1/1
 Running     0              50s
longhorn-system   csi-provisioner-667796df57-dqkfr                  1/1
 Running     0              50s
longhorn-system   csi-attacher-5c4bfdcf59-wckt8                     1/1
 Running     0              50s
longhorn-system   csi-resizer-694f8f5f64-7n2kq                      1/1
 Running     0              50s
longhorn-system   csi-snapshotter-959b69d4b-rp4gk                   1/1
 Running     0              50s
longhorn-system   csi-resizer-694f8f5f64-r6ljc                      1/1
 Running     0              50s
longhorn-system   csi-resizer-694f8f5f64-k7429                      1/1
 Running     0              50s
longhorn-system   csi-snapshotter-959b69d4b-5k8pg                   1/1
 Running     0              50s
longhorn-system   csi-provisioner-667796df57-n5w9s                  1/1
 Running     0              50s
longhorn-system   csi-snapshotter-959b69d4b-x7b7t                   1/1
 Running     0              50s
longhorn-system   longhorn-csi-plugin-bsc8c                         3/3
 Running     0              50s
```

17.3 Criando volumes do SUSE Storage

O SUSE Storage usa os recursos do Kubernetes chamados `StorageClass` para provisionar automaticamente os objetos `PersistentVolume` aos pods. Pense no `StorageClass` como um método para os administradores descreverem *classes* ou *perfis* do armazenamento que eles oferecem.

Vamos criar um `StorageClass` com algumas opções padrão:

```
kubectl apply -f - <<EOF
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: longhorn-example
provisioner: driver.longhorn.io
allowVolumeExpansion: true
```

```
parameters:
  numberOfReplicas: "3"
  staleReplicaTimeout: "2880" # 48 hours in minutes
  fromBackup: ""
  fsType: "ext4"
EOF
```

Agora que temos um `StorageClass`, precisamos que um `PersistentVolumeClaim` faça referência a ele. Um `PersistentVolumeClaim` (PVC) é uma solicitação de armazenamento feita pelo usuário. Os PVCs consomem os recursos `PersistentVolume`. As declarações podem solicitar tamanhos e modos de acesso específicos (por exemplo, é possível montá-las como leitura/gravação uma vez ou como somente leitura várias vezes).

Vamos criar um `PersistentVolumeClaim`:

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: longhorn-volv-pvc
  namespace: longhorn-system
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: longhorn-example
  resources:
    requests:
      storage: 2Gi
EOF
```

Pronto! Com a criação do `PersistentVolumeClaim`, podemos prosseguir e anexá-lo a um `Pod`. Quando o `Pod` é implantado, o Kubernetes cria o volume do Longhorn e o vincula ao `Pod` quando há armazenamento disponível.

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
  namespace: longhorn-system
spec:
  containers:
  - name: volume-test
    image: nginx:stable-alpine
    imagePullPolicy: IfNotPresent
    volumeMounts:
```

```
    - name: volv
      mountPath: /data
    ports:
    - containerPort: 80
  volumes:
  - name: volv
    persistentVolumeClaim:
      claimName: longhorn-volv-pvc
EOF
```

Dica

O conceito de armazenamento no Kubernetes é um tópico complexo, porém importante. Mencionamos rapidamente alguns do recursos mais comuns do Kubernetes, mas sugerimos que você se familiarize com a documentação de terminologia (https://longhorn.io/docs/1.8.1/terminology/) que o Longhorn oferece.

Neste exemplo, o resultado deve ter esta aparência:

```
localhost:~ # kubectl get storageclass
NAME                 PROVISIONER          RECLAIMPOLICY   VOLUMEBINDINGMODE
 ALLOWVOLUMEEXPANSION   AGE
longhorn (default)   driver.longhorn.io   Delete          Immediate           true
            12m
longhorn-example     driver.longhorn.io   Delete          Immediate           true
            24s

localhost:~ # kubectl get pvc -n longhorn-system
NAME                STATUS   VOLUME                                     CAPACITY   ACCESS
 MODES   STORAGECLASS       AGE
longhorn-volv-pvc   Bound    pvc-f663a92e-ac32-49ae-b8e5-8a6cc29a7d1e   2Gi        RWO
         longhorn-example   54s

localhost:~ # kubectl get pods -n longhorn-system
NAME                                                READY   STATUS    RESTARTS      AGE
csi-attacher-5c4bfdcf59-qmjtz                       1/1     Running   0             14m
csi-attacher-5c4bfdcf59-s7n65                       1/1     Running   0             14m
csi-attacher-5c4bfdcf59-w9xgs                       1/1     Running   0             14m
csi-provisioner-667796df57-fmz2d                    1/1     Running   0             14m
csi-provisioner-667796df57-p7rjr                    1/1     Running   0             14m
csi-provisioner-667796df57-w9fdq                    1/1     Running   0             14m
csi-resizer-694f8f5f64-2rb8v                        1/1     Running   0             14m
csi-resizer-694f8f5f64-z9v9x                        1/1     Running   0             14m
csi-resizer-694f8f5f64-zlncz                        1/1     Running   0             14m
csi-snapshotter-959b69d4b-5dpvj                     1/1     Running   0             14m
```

```
csi-snapshotter-959b69d4b-lwwkv                    1/1     Running   0              14m
csi-snapshotter-959b69d4b-tzhwc                    1/1     Running   0              14m
engine-image-ei-5cefaf2b-hvdv5                     1/1     Running   0              14m
instance-manager-0ee452a2e9583753e35ad00602250c5b  1/1     Running   0              14m
longhorn-csi-plugin-gd2jx                          3/3     Running   0              14m
longhorn-driver-deployer-9f4fc86-j6h2b             1/1     Running   0              15m
longhorn-manager-z4lnl                             1/1     Running   0              15m
longhorn-ui-5f4b7bbf69-bln7h                       1/1     Running   3 (14m ago)    15m
longhorn-ui-5f4b7bbf69-lh97n                       1/1     Running   3 (14m ago)    15m
volume-test                                        1/1     Running   0              26s
```

17.4 Acessando a IU

Se você instalou o Longhorn com o kubectl ou o Helm, precisa configurar um controle de entrada para permitir o tráfego externo no cluster. A autenticação não está habilitada por padrão. Se o app de catálogo do Rancher foi usado, o Rancher criou automaticamente um controlador de entrada com controle de acesso (rancher-proxy).

1. Obtenha o endereço IP do serviço externo do Longhorn:

   ```
   kubectl -n longhorn-system get svc
   ```

2. Depois que você recuperar o endereço IP do `longhorn-frontend`, poderá começar a usar a IU navegando até ela pelo navegador.

17.5 Instalando com o Edge Image Builder

O SUSE Edge usa o *Capítulo 11, Edge Image Builder* para personalizar as imagens base do sistema operacional SUSE Linux Micro. Vamos demonstrar como fazer isso para provisionar um cluster RKE2 junto com o Longhorn.

Vamos criar o arquivo de definição:

```
export CONFIG_DIR=$HOME/eib
mkdir -p $CONFIG_DIR

cat << EOF > $CONFIG_DIR/iso-definition.yaml
apiVersion: 1.2
image:
  imageType: iso
```

```
  baseImage: SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso
  arch: x86_64
  outputImageName: eib-image.iso
kubernetes:
  version: v1.32.4+rke2r1
  helm:
    charts:
      - name: longhorn
        version: 106.2.0+up1.8.1
        repositoryName: longhorn
        targetNamespace: longhorn-system
        createNamespace: true
        installationNamespace: kube-system
      - name: longhorn-crd
        version: 106.2.0+up1.8.1
        repositoryName: longhorn
        targetNamespace: longhorn-system
        createNamespace: true
        installationNamespace: kube-system
    repositories:
      - name: longhorn
        url: https://charts.rancher.io
operatingSystem:
  packages:
    sccRegistrationCode: <reg-code>
    packageList:
      - open-iscsi
  users:
  - username: root
    encryptedPassword: \$6\$jHugJNNd3HElGsUZ\
$eodjVe4te5ps44SVcWshdfWizrP.xAyd71CVEXazBJ/.v799/WRCBXxfYmunlBO2yp1hm/zb4r8EmnrrNCF.P/
EOF
```

Nota

A personalização de qualquer um dos valores do gráfico Helm é possível com um arquivo separado fornecido em `helm.charts[].valuesFile`. Consulte a documentação upstream (https://github.com/suse-edge/edge-image-builder/blob/release-1.2/docs/building-images.md#kubernetes) para obter detalhes.

Vamos criar a imagem:

```
podman run --rm --privileged -it -v $CONFIG_DIR:/eib registry.suse.com/edge/3.3/edge-
image-builder:1.2.1 build --definition-file $CONFIG_DIR/iso-definition.yaml
```

Após a criação da imagem, você poderá usá-la para instalar o sistema operacional em um host físico ou virtual. Depois que o provisionamento é concluído, é possível fazer login no sistema usando o par de credenciais `root:eib`.

Verifique se o Longhorn foi implantado com sucesso:

```
localhost:~ # /var/lib/rancher/rke2/bin/kubectl --kubeconfig /etc/rancher/rke2/rke2.yaml
 -n longhorn-system get pods
NAME                                                READY   STATUS    RESTARTS        AGE
csi-attacher-5c4bfdcf59-qmjtz                       1/1     Running   0
 103s
csi-attacher-5c4bfdcf59-s7n65                       1/1     Running   0
 103s
csi-attacher-5c4bfdcf59-w9xgs                       1/1     Running   0
 103s
csi-provisioner-667796df57-fmz2d                    1/1     Running   0
 103s
csi-provisioner-667796df57-p7rjr                    1/1     Running   0
 103s
csi-provisioner-667796df57-w9fdq                    1/1     Running   0
 103s
csi-resizer-694f8f5f64-2rb8v                        1/1     Running   0
 103s
csi-resizer-694f8f5f64-z9v9x                        1/1     Running   0
 103s
csi-resizer-694f8f5f64-zlncz                        1/1     Running   0
 103s
csi-snapshotter-959b69d4b-5dpvj                     1/1     Running   0
 103s
csi-snapshotter-959b69d4b-lwwkv                     1/1     Running   0
 103s
csi-snapshotter-959b69d4b-tzhwc                     1/1     Running   0
 103s
engine-image-ei-5cefaf2b-hvdv5                      1/1     Running   0
 109s
instance-manager-0ee452a2e9583753e35ad00602250c5b   1/1     Running   0
 109s
longhorn-csi-plugin-gd2jx                           3/3     Running   0
 103s
longhorn-driver-deployer-9f4fc86-j6h2b              1/1     Running   0
 2m28s
longhorn-manager-z4lnl                              1/1     Running   0
 2m28s
longhorn-ui-5f4b7bbf69-bln7h                        1/1     Running   3 (2m7s ago)
 2m28s
```

```
longhorn-ui-5f4b7bbf69-lh97n                          1/1     Running   3 (2m10s ago)
 2m28s
```

Nota

Essa instalação não funciona em ambientes totalmente air-gapped. Para esses casos, consulte a *Seção 27.8, “Instalação do SUSE Storage”*.

18 SUSE Security

O SUSE Security é uma solução de segurança para Kubernetes que proporciona segurança de rede L7, de runtime e da cadeia de suprimento, e verificações de conformidade, em um pacote consistente.

O SUSE Security é um produto implantado como uma plataforma de vários contêineres, cada um se comunicando por diversas portas e interfaces. Internamente, ele usa o NeuVector como componente de segurança do contêiner subjacente. Os seguintes contêineres compõem a plataforma SUSE Security:

- Gerenciador. Um contêiner sem estado com console baseado na web. Normalmente, apenas um é necessário e pode ser executado em qualquer local. Uma falha no gerenciador não afeta as operações do controlador ou do executor. No entanto, determinadas notificações (eventos) e dados de conexão recentes são armazenados em cache na memória pelo gerenciador, portanto, a visualização deles pode ser afetada.

- Controlador. O "plano de controle" do SUSE Security deve ser implantado em uma configuração de alta disponibilidade, assim a configuração não se perde em caso de falha no nó. Ele pode ser executado em qualquer local, embora os clientes geralmente prefiram colocá-los em nós de "gerenciamento", mestre ou de infraestrutura devido à sua criticidade.

- Executor. Esse contêiner é implantado como um DaemonSet para que haja um executor em cada nó que será protegido. Em geral, a implantação é feita em cada nó do worker, mas é possível permitir a programação para que os nós mestre e de infraestrutura também sejam implantados lá. Nota: Se o executor não estiver em um nó do cluster e as conexões vierem de um pod nesse nó, o SUSE Security os identificará como cargas de trabalho "não gerenciadas".

- Verificador. Faz a verificação de vulnerabilidades usando o banco de dados CVE incorporado, conforme direcionado pelo controlador. É possível implantar vários verificadores para aumentar a capacidade de verificação. Os verificadores podem ser executados em qualquer local, mas costumam executar em nós onde os controladores são executados. Veja a seguir as considerações de dimensionamento dos nós do verificador. Também é possível invocar um verificador de forma independente, quando usado para

verificação de fase de compilação, por exemplo, em um pipeline que aciona a verificação, recupera os resultados e interrompe o verificador. O verificador inclui o banco de dados CVE mais recente, portanto, ele deve ser atualizado diariamente.

- Atualizador. Aciona uma atualização do verificador pelo cron job do Kubernetes quando uma atualização do banco de dados CVE é desejada. Faça essa configuração em seu ambiente.

Você encontra a documentação mais detalhada sobre a integração e as melhores práticas do SUSE Security aqui (https://open-docs.neuvector.com/).

18.1 Como o SUSE Edge usa o SUSE Security?

O SUSE Edge oferece uma configuração mais simples do SUSE Security como ponto de partida para implantações de borda.

18.2 Observações importantes

- O contêiner `Scanner` deve ter memória suficiente para extrair a imagem que será verificada na memória e expandi-la. Para verificar imagens com mais de 1 GB, aumente a memória do verificador um pouco acima do maior tamanho esperado da imagem.
- O modo de proteção espera grandes volumes de conexões de rede. O `Enforcer` requer CPU e memória no modo de proteção (bloqueio de firewall integrado) para armazenar e inspecionar as conexões e a possível carga (DLP). Aumentar a memória e dedicar um núcleo de CPU ao `Enforcer` garantem a capacidade de filtragem de pacotes adequada.

18.3 Instalando com o Edge Image Builder

O SUSE Edge usa o *Capítulo 11, Edge Image Builder* para personalizar as imagens base do sistema operacional SUSE Linux Micro. Siga a *Seção 27.7, "Instalação do SUSE Security"* para uma instalação air-gapped do SUSE Security em clusters Kubernetes provisionados pelo EIB.

19 MetalLB

Consulte a documentação oficial do MetalLB (https://metallb.universe.tf/).

> O MetalLB é uma implementação de balanceador de carga para clusters Kubernetes bare metal, usando protocolos de roteamento padrão.
>
> Em ambientes bare metal, a configuração de balanceadores de carga de rede é visivelmente mais complexa do que em ambientes de nuvem. Ao contrário das chamadas de API simples nas configurações de nuvem, o bare metal requer aplicações de rede dedicadas ou uma combinação de balanceadores de carga e configurações de IP virtual (VIP, Virtual IP) para gerenciar a alta disponibilidade (HA, High Availability) ou resolver o ponto único de falha (SPOF, Single Point of Failure) inerentes em um balanceador de carga de nó único. Essas configurações não são automatizadas com facilidade, criando desafios para as implantações do Kubernetes em que a escala dos componentes aumenta e reduz de forma dinâmica.
>
> O MetalLB resolve esses desafios usando o modelo do Kubernetes para criar serviços do tipo LoadBalancer como se operassem em um ambiente de nuvem, mesmo nas configurações bare metal.
>
> Há duas abordagens diferentes, pelo modo L2 (https://metallb.universe.tf/concepts/layer2/) (usando *truques* de ARP) ou pelo BGP (https://metallb.universe.tf/concepts/bgp/). Basicamente, o L2 não precisa de equipamento de rede especial, mas o BGP costuma ser melhor. Isso depende do caso de uso.

19.1 Como o SUSE Edge usa o MetalLB?

O SUSE Edge usa o MetalLB de duas maneiras principais:

- Como solução de balanceador de carga: o MetalLB atua como solução de balanceador de carga para máquinas bare metal.
- Em uma configuração do K3s/RKE2 de alta disponibilidade: o MetalLB permite o balanceamento de carga da API do Kubernetes usando um endereço IP virtual.

Nota

Para expor a API, o Endpoint Copier Operator (*Capítulo 20, Endpoint Copier Operator*) é usado para sincronização dos endpoints da API K8s do serviço `kubernetes` com um serviço LoadBalancer `kubernetes-vip`.

19.2 Melhores práticas

A instalação do MetalLB no modo L2 está descrita no *Capítulo 25, MetalLB no K3s (usando o modo de camada 2)*.

Um guia para instalação do MetalLB na frente do `kube-api-server` para obter uma topologia de alta disponibilidade está disponível no *Capítulo 26, MetalLB na frente do servidor da API Kubernetes*.

19.3 Problemas conhecidos

- O K3s vem com a própria solução de balanceador de carga chamada `Klipper`. Para usar o MetalLB, é necessário desabilitar o `Klipper`. Para fazer isso, inicie o servidor K3s com a opção `--disable servicelb`, conforme descrito na documentação do K3s (https://docs.k3s.io/networking).

20 Endpoint Copier Operator

Endpoint Copier Operator (https://github.com/suse-edge/endpoint-copier-operator) é um operador do Kubernetes que tem como finalidade criar uma cópia de um serviço e endpoint do Kubernetes e mantê-los sincronizados.

20.1 Como o SUSE Edge usa o Endpoint Copier Operator?

No SUSE Edge, o Endpoint Copier Operator desempenha um papel crucial na configuração de alta disponibilidade (HA, High Availability) para clusters K3s/RKE2. Para que isso seja possível, crie um serviço `kubernetes-vip` do tipo `LoadBalancer`, garantindo que seu endpoint esteja sempre sincronizado com o endpoint do kubernetes. O MetalLB (*Capítulo 19, MetalLB*) é usado para gerenciar o serviço `kubernetes-vip`, já que o endereço IP exposto é usado em outros nós para ingressar no cluster.

20.2 Melhores práticas

Você encontra a documentação completa sobre como usar o Endpoint Copier Operator aqui (https://github.com/suse-edge/endpoint-copier-operator/blob/main/README.md).

Consulte também nosso guia (*Capítulo 25, MetalLB no K3s (usando o modo de camada 2)*) para obter uma configuração de alta disponibilidade do K3s/RKE2 por meio do Endpoint Copier Operator e do MetalLB.

20.3 Problemas conhecidos

No momento, o funcionamento do Endpoint Copier Operator está limitado apenas a um serviço/endpoint. Há planos para estender o suporte a vários serviços/endpoints no futuro.

21 Edge Virtualization

Esta seção descreve como usar o Edge Virtualization para executar máquinas virtuais em nós de borda. O Edge Virtualization foi projetado para casos de uso de virtualização leve, em que se espera o uso de um fluxo de trabalho comum para implantação e gerenciamento de aplicativos tanto virtualizados quanto conteinerizados.

O SUSE Edge Virtualization oferece suporte a dois métodos de execução de máquinas virtuais:

1. Implantação manual de máquinas virtuais pelo libvirt + qemu-kvm no nível do host (não há envolvimento do Kubernetes)
2. Implantação do operador KubeVirt para gerenciamento de máquinas virtuais com base no Kubernetes

As duas opções são válidas, mas apenas a segunda está descrita abaixo. Para usar os mecanismos de virtualização padrão e prontos para uso incluídos no SUSE Linux Micro, há um guia completo aqui (https://documentation.suse.com/sles/15-SP6/html/SLES-all/chap-virtualization-introduction.html) e, apesar de ter sido escrito inicialmente para o SUSE Linux Enterprise Server, os conceitos são quase idênticos.

A princípio, este guia explica como implantar os componentes de virtualização adicionais em um sistema que já foi pré-implantado, mas acompanha uma seção que descreve como incorporar essa configuração à implantação inicial usando o Edge Image Builder. Se você não quer ler os conceitos básicos e definir as configurações manualmente, pule direto para essa seção.

21.1 Visão geral do KubeVirt

O KubeVirt permite o gerenciamento de máquinas virtuais com o Kubernetes junto ao restante de suas cargas de trabalho conteinerizadas. Para fazer isso, ele executa a parte referente ao espaço do usuário da pilha de virtualização do Linux em um contêiner. Isso minimiza os requisitos no sistema host, o que facilita a configuração e o gerenciamento.

Os detalhes sobre a arquitetura do KubeVirt estão disponíveis na documentação upstream (https://kubevirt.io/user-guide/architecture/).

21.2 Pré-requisitos

Se você está seguindo este guia, já deve ter os seguintes itens disponíveis:

- No mínimo, um host físico com o SUSE Linux Micro 6.1 instalado e com as extensões de virtualização habilitadas no BIOS (consulte aqui (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-virt-support.html#sec-kvm-requires-hardware) para obter detalhes).
- Em todos os nós, um cluster Kubernetes do K3s/RKE2 já implantado com um `kubeconfig` apropriado que permite o acesso de superusuário ao cluster.
- Acesso ao usuário root. Nestas instruções, consideramos você como usuário root, *sem* escalar seus privilégios por meio do `sudo`.
- Você tem o Helm (https://helm.sh/docs/intro/install/) disponível localmente com uma conexão de rede adequada para poder enviar as configurações ao cluster Kubernetes e fazer download das imagens necessárias.

21.3 Instalação manual do Edge Virtualization

Este guia não orientará na implantação do Kubernetes, mas ele considera que você já instalou a versão apropriada ao SUSE Edge do K3s (https://k3s.io/) ou do RKE2 (https://docs.rke2.io/install/quickstart) e configurou o kubeconfig de acordo para que os comandos `kubectl` padrão sejam executados como superusuário. Pressupomos que seu nó constitua um cluster de nó único, apesar de não haver diferenças significativas esperadas nas implantações de vários nós.

O SUSE Edge Virtualization é implantado por meio de três gráficos Helm separados, especificamente:

- **KubeVirt**: os componentes de virtualização principais, ou seja, as CRDs do Kubernetes, os operadores e outros componentes necessários para que o Kubernetes possa implantar e gerenciar máquinas virtuais.
- **Extensão de dashboard KubeVirt**: uma extensão de IU opcional do Rancher que permite o gerenciamento básico de máquinas virtuais, por exemplo, iniciar/parar máquinas virtuais e acessar o console.
- **Containerized Data Importer (CDI)**: um componente adicional que permite a integração de armazenamento persistente para KubeVirt, oferecendo recursos para as máquinas virtuais usarem os back ends de armazenamento do Kubernetes existentes para dados, mas também permitindo que os usuários importem ou clonem volumes de dados em máquinas virtuais.

Cada um desses gráficos Helm é versionado de acordo com a versão do SUSE Edge que você usa. Para uso em produção/com suporte, aplique os artefatos disponíveis no SUSE Registry.

Primeiro, garanta que o acesso ao `kubectl` esteja funcionando:

```
$ kubectl get nodes
```

Esse comando deve retornar algo semelhante ao seguinte:

```
NAME                   STATUS   ROLES                       AGE     VERSION
node1.edge.rdo.wales   Ready    control-plane,etcd,master   4h20m   v1.30.5+rke2r1
node2.edge.rdo.wales   Ready    control-plane,etcd,master   4h15m   v1.30.5+rke2r1
node3.edge.rdo.wales   Ready    control-plane,etcd,master   4h15m   v1.30.5+rke2r1
```

Agora você pode instalar os gráficos Helm **KubeVirt** e **Containerized Data Importer (CDI)**:

```
$ helm install kubevirt oci://registry.suse.com/edge/charts/kubevirt --namespace
 kubevirt-system --create-namespace
$ helm install cdi oci://registry.suse.com/edge/charts/cdi --namespace cdi-system --
create-namespace
```

Em alguns minutos, os componentes KubeVirt e CDI estarão implantados. Para validar isso, verifique todos os recursos implantados no namespace `kubevirt-system` e `cdi-system`.

Verifique os recursos do KubeVirt:

```
$ kubectl get all -n kubevirt-system
```

Esse comando deve retornar algo semelhante ao seguinte:

```
NAME                                    READY   STATUS    RESTARTS      AGE
pod/virt-operator-5fbcf48d58-p7xpm      1/1     Running   0             2m24s
pod/virt-operator-5fbcf48d58-wnf6s      1/1     Running   0             2m24s
pod/virt-handler-t594x                  1/1     Running   0             93s
pod/virt-controller-5f84c69884-cwjvd    1/1     Running   1 (64s ago)   93s
pod/virt-controller-5f84c69884-xxw6q    1/1     Running   1 (64s ago)   93s
pod/virt-api-7dfc54cf95-v8kcl           1/1     Running   1 (59s ago)   118s

NAME                                  TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
 AGE
service/kubevirt-prometheus-metrics   ClusterIP   None            <none>        443/TCP
 2m1s
service/virt-api                      ClusterIP   10.43.56.140    <none>        443/TCP
 2m1s
service/kubevirt-operator-webhook     ClusterIP   10.43.201.121   <none>        443/TCP
 2m1s
service/virt-exportproxy              ClusterIP   10.43.83.23     <none>        443/TCP
 2m1s

NAME                           DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
 SELECTOR            AGE
daemonset.apps/virt-handler   1         1         1       1            1
 kubernetes.io/os=linux   93s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/virt-operator       2/2     2            2           2m24s
deployment.apps/virt-controller     2/2     2            2           93s
deployment.apps/virt-api            1/1     1            1           118s

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/virt-operator-5fbcf48d58      2         2         2       2m24s
replicaset.apps/virt-controller-5f84c69884    2         2         2       93s
replicaset.apps/virt-api-7dfc54cf95           1         1         1       118s

NAME                              AGE     PHASE
kubevirt.kubevirt.io/kubevirt     2m24s   Deployed
```

Verifique os recursos do CDI:

```
$ kubectl get all -n cdi-system
```

Esse comando deve retornar algo semelhante ao seguinte:

```
NAME                                   READY   STATUS    RESTARTS   AGE
pod/cdi-operator-55c74f4b86-692xb      1/1     Running   0          2m24s
```

```
pod/cdi-apiserver-db465b888-62lvr      1/1     Running   0          2m21s
pod/cdi-deployment-56c7d74995-mgkfn    1/1     Running   0          2m21s
pod/cdi-uploadproxy-7d7b94b968-6kxc2   1/1     Running   0          2m22s

NAME                              TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
service/cdi-uploadproxy           ClusterIP   10.43.117.7    <none>        443/TCP
 2m22s
service/cdi-api                   ClusterIP   10.43.20.101   <none>        443/TCP
 2m22s
service/cdi-prometheus-metrics    ClusterIP   10.43.39.153   <none>        8080/TCP
 2m21s

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/cdi-operator       1/1     1            1           2m24s
deployment.apps/cdi-apiserver      1/1     1            1           2m22s
deployment.apps/cdi-deployment     1/1     1            1           2m21s
deployment.apps/cdi-uploadproxy    1/1     1            1           2m22s

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/cdi-operator-55c74f4b86       1         1         1       2m24s
replicaset.apps/cdi-apiserver-db465b888       1         1         1       2m21s
replicaset.apps/cdi-deployment-56c7d74995     1         1         1       2m21s
replicaset.apps/cdi-uploadproxy-7d7b94b968    1         1         1       2m22s
```

Para verificar se as definições de recursos personalizados (CRDs, Custom Resource Definitions) `VirtualMachine` foram implantadas, faça a validação com:

```
$ kubectl explain virtualmachine
```

Esse comando deve retornar a definição do objeto `VirtualMachine`, com o seguinte conteúdo:

```
GROUP:      kubevirt.io
KIND:       VirtualMachine
VERSION:    v1

DESCRIPTION:
    VirtualMachine handles the VirtualMachines that are not running or are in a
    stopped state The VirtualMachine contains the template to create the
    VirtualMachineInstance. It also mirrors the running state of the created
    VirtualMachineInstance in its status.
(snip)
```

21.4 Implantando máquinas virtuais

Agora que KubeVirt e CDI estão implantados, vamos definir uma máquina virtual simples com base no openSUSE Tumbleweed (https://get.opensuse.org/tumbleweed/). Essa máquina virtual tem as configurações mais simples, usando a configuração de "rede de pod" padrão idêntica a qualquer outro pod. Ela também utiliza um armazenamento não persistente, o que garante que seja efêmero como em qualquer contêiner que não tenha um PVC (https://kubernetes.io/docs/concepts/storage/persistent-volumes/).

```
$ kubectl apply -f - <<EOF
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: tumbleweed
  namespace: default
spec:
  runStrategy: Always
  template:
    spec:
      domain:
        devices: {}
        machine:
          type: q35
        memory:
          guest: 2Gi
        resources: {}
      volumes:
      - containerDisk:
          image: registry.opensuse.org/home/roxenham/tumbleweed-container-disk/
containerfile/cloud-image:latest
        name: tumbleweed-containerdisk-0
      - cloudInitNoCloud:
          userDataBase64:
 I2Nsb3VkLWNvbmZpZwpkaXNhYmxlX3Jvb3Q6IGZhbHNlCnNzaF9wd2F1dGg6IFRydWUKdXNlcnM6CiAgLSBkZWZhdWx0CiAgLSBuYW
        name: cloudinitdisk
EOF
```

A saída deve mostrar que uma `VirtualMachine` foi criada:

```
virtualmachine.kubevirt.io/tumbleweed created
```

Essa definição da `VirtualMachine` é mínima e especifica pouco da configuração. Ela simplesmente descreve que se trata de uma máquina do tipo "q35 (https://wiki.qemu.org/Features/Q35)" com 2 GB de memória, que usa uma imagem de disco baseada em um `containerDisk`

efêmero (ou seja, uma imagem de disco armazenada na imagem de contêiner de um repositório de imagens remoto), e especifica um disco cloudInit codificado com base64, que usamos apenas para criação de usuários e imposição de senha no momento da inicialização (usamos `base64 -d` para decodificá-lo).

Nota

Essa imagem da máquina virtual é somente para teste. Ela não tem suporte oficial e serve apenas como exemplo para documentação.

A inicialização da máquina leva alguns minutos porque ela precisa fazer download da imagem de disco do openSUSE Tumbleweed. Depois disso, você poderá consultar as informações para ver mais detalhes sobre a máquina virtual:

```
$ kubectl get vmi
```

Esse comando deve mostrar o nó em que a máquina virtual foi iniciada e o endereço IP dela. Como ela usa a rede de pod, lembre-se de que o endereço IP relatado será exatamente igual ao de qualquer outro pod e, dessa forma, roteável:

```
NAME          AGE     PHASE     IP           NODENAME               READY
tumbleweed    4m24s   Running   10.42.2.98   node3.edge.rdo.wales   True
```

Ao executar esses comandos nos próprios nós do cluster Kubernetes, com uma CNI para rotear o tráfego diretamente aos pods (por exemplo, Cilium), você pode executar `ssh` diretamente na máquina. Substitua o seguinte endereço IP pelo que foi atribuído à sua máquina virtual:

```
$ ssh suse@10.42.2.98
(password is "suse")
```

Depois que você estiver nessa máquina virtual, poderá explorá-la, mas lembre-se de que ela tem recursos limitados e apenas 1 GB de espaço em disco. Quando concluir, use `Ctrl-D` ou `exit` para se desconectar da sessão SSH.

O processo da máquina virtual ainda é agrupado em um pod padrão do Kubernetes. A CRD `VirtualMachine` é uma representação da máquina virtual desejada, mas o processo em que a máquina virtual foi realmente iniciada é realizado pelo pod `virt-launcher`, um pod padrão do Kubernetes, assim como com qualquer outro aplicativo. Para cada máquina virtual iniciada, você vê que existe um pod `virt-launcher`:

```
$ kubectl get pods
```

Em seguida, o pod `virt-launcher` deve aparecer para a máquina Tumbleweed que definimos:

```
NAME                              READY   STATUS    RESTARTS   AGE
virt-launcher-tumbleweed-8gcn4   3/3     Running   0          10m
```

Se você observar o pod `virt-launcher`, verá que ele está executando os processos `libvirt` e `qemu-kvm`. É possível entrar e analisar o conteúdo do pod, levando em conta que você precisa adaptar o seguinte comando com o nome do seu pod:

```
$ kubectl exec -it virt-launcher-tumbleweed-8gcn4 -- bash
```

Depois que você estiver no pod, execute os comandos `virsh` enquanto observa os processos. Você verá o binário `qemu-system-x86_64` em execução, junto com alguns processos para monitorar a máquina virtual, e também o local da imagem de disco e como a rede está conectada (como um dispositivo TAP):

```
qemu@tumbleweed:/> ps ax
  PID TTY      STAT   TIME COMMAND
    1 ?        Ssl    0:00 /usr/bin/virt-launcher-monitor --qemu-timeout 269s --name
 tumbleweed --uid b9655c11-38f7-4fa8-8f5d-bfe987dab42c --namespace default --kubevirt-
share-dir /var/run/kubevirt --ephemeral-disk-dir /var/run/kubevirt-ephemeral-disks --
container-disk-dir /var/run/kube
   12 ?        Sl     0:01 /usr/bin/virt-launcher --qemu-timeout 269s --name tumbleweed
 --uid b9655c11-38f7-4fa8-8f5d-bfe987dab42c --namespace default --kubevirt-share-dir /
var/run/kubevirt --ephemeral-disk-dir /var/run/kubevirt-ephemeral-disks --container-disk-
dir /var/run/kubevirt/con
   24 ?        Sl     0:00 /usr/sbin/virtlogd -f /etc/libvirt/virtlogd.conf
   25 ?        Sl     0:01 /usr/sbin/virtqemud -f /var/run/libvirt/virtqemud.conf
   83 ?        Sl     0:31 /usr/bin/qemu-system-x86_64 -name
 guest=default_tumbleweed,debug-threads=on -S -object {"qom-
type":"secret","id":"masterKey0","format":"raw","file":"/var/run/kubevirt-private/
libvirt/qemu/lib/domain-1-default_tumbleweed/master-key.aes"} -machine pc-q35-7.1,usb
  286 pts/0    Ss     0:00 bash
  320 pts/0    R+     0:00 ps ax

qemu@tumbleweed:/> virsh list --all
 Id   Name                 State
-----------------------------------
 1    default_tumbleweed   running

qemu@tumbleweed:/> virsh domblklist 1
 Target   Source
-------------------------------------------------------------------------------------------
 sda      /var/run/kubevirt-ephemeral-disks/disk-data/tumbleweed-containerdisk-0/
disk.qcow2
```

```
 sdb      /var/run/kubevirt-ephemeral-disks/cloud-init-data/default/tumbleweed/
noCloud.iso

qemu@tumbleweed:/> virsh domiflist 1
 Interface   Type       Source   Model                     MAC
-------------------------------------------------------------------------------
 tap0        ethernet   -        virtio-non-transitional   e6:e9:1a:05:c0:92

qemu@tumbleweed:/> exit
exit
```

Por fim, vamos excluir a máquina virtual para limpeza:

```
$ kubectl delete vm/tumbleweed
virtualmachine.kubevirt.io "tumbleweed" deleted
```

21.5 Usando o virtctl

Junto com a ferramenta CLI padrão do Kubernetes, ou seja, `kubectl`, o KubeVirt integra um utilitário CLI complementar que permite estabelecer interface com seu cluster para suprir algumas lacunas entre o cenário da virtualização e o cenário para o qual o Kubernetes foi projetado. Por exemplo, a ferramenta `virtctl` permite gerenciar o ciclo de vida das máquinas virtuais (início, parada, reinicialização etc.), concedendo acesso aos consoles virtuais, fazendo upload das imagens de máquina virtual e estabelecendo interface com as estruturas do Kubernetes, como serviços, sem usar diretamente a API ou as CRDs.

Vamos fazer download da versão estável mais recente da ferramenta `virtctl`:

```
$ export VERSION=v1.4.0
$ wget https://github.com/kubevirt/kubevirt/releases/download/$VERSION/virtctl-$VERSION-
linux-amd64
```

Se você usa uma arquitetura diferente ou uma máquina não Linux, pode encontrar outras versões aqui (https://github.com/kubevirt/kubevirt/releases). Antes de continuar, você precisa torná-la executável, o que pode ser útil para movê-la até um local em seu `$PATH`:

```
$ mv virtctl-$VERSION-linux-amd64 /usr/local/bin/virtctl
$ chmod a+x /usr/local/bin/virtctl
```

Na sequência, use a ferramenta de linha de comando `virtctl` para criar máquinas virtuais. Vamos replicar nossa máquina virtual anterior, observando que estamos fazendo pipe da saída diretamente em `kubectl apply`:

```
$ virtctl create vm --name virtctl-example --memory=1Gi \
```

```
    --volume-containerdisk=src:registry.opensuse.org/home/roxenham/tumbleweed-container-
disk/containerfile/cloud-image:latest \
    --cloud-init-user-data
 "I2Nsb3VkLWNvbmZpZwpkaXNhYmxlX3Jvb3Q6IGZhbHNlCnNzaF9wd2F1dGg6IFRydWUKdXNlcnM6CiAgLSBkZWZhdWx0CiAgLSBuY
 | kubectl apply -f -
```

Isso deve mostrar a máquina virtual em execução (desta vez, ela deve ser iniciada de forma bem mais rápida, já que a imagem de contêiner estará armazenada em cache):

```
$ kubectl get vmi
NAME              AGE   PHASE     IP           NODENAME               READY
virtctl-example   52s   Running   10.42.2.29   node3.edge.rdo.wales   True
```

Vamos usar `virtctl` para conexão direta com a máquina virtual:

```
$ virtctl ssh suse@virtctl-example
(password is "suse" - Ctrl-D to exit)
```

Há vários outros comandos que o `virtctl` pode usar. Por exemplo, o `virtctl console` concede a você acesso ao console serial quando a rede não está funcionando, e você pode usar o `virtctl guestosinfo` para obter as informações completas do sistema operacional, desde que o convidado tenha o `qemu-guest-agent` instalado e em execução.

Por fim, vamos pausar e retomar a máquina virtual:

```
$ virtctl pause vm virtctl-example
VMI virtctl-example was scheduled to pause
```

Observe que o objeto `VirtualMachine` aparece como **Paused** (Pausado), e o objeto `VirtualMachineInstance` como **Running** (Em execução), mas **READY = False** (PRONTO = Falso):

```
$ kubectl get vm
NAME              AGE     STATUS   READY
virtctl-example   8m14s   Paused   False

$ kubectl get vmi
NAME              AGE     PHASE     IP           NODENAME               READY
virtctl-example   8m15s   Running   10.42.2.29   node3.edge.rdo.wales   False
```

Veja também que não é mais possível se conectar à máquina virtual:

```
$ virtctl ssh suse@virtctl-example
can't access VMI virtctl-example: Operation cannot be fulfilled on
 virtualmachineinstance.kubevirt.io "virtctl-example": VMI is paused
```

Vamos retomar a máquina virtual e tentar novamente:

```
$ virtctl unpause vm virtctl-example
VMI virtctl-example was scheduled to unpause
```

Agora devemos conseguir reestabelecer a conexão:

```
$ virtctl ssh suse@virtctl-example
suse@vmi/virtctl-example.default's password:
suse@virtctl-example:~> exit
logout
```

Por fim, vamos remover a máquina virtual:

```
$ kubectl delete vm/virtctl-example
virtualmachine.kubevirt.io "virtctl-example" deleted
```

21.6 Rede de entrada simples

Nesta seção, mostramos como é possível expor máquinas virtuais como serviços padrão do Kubernetes e disponibilizá-los pelo serviço de entrada do Kubernetes, por exemplo, NGINX com RKE2 (https://docs.rke2.io/networking/networking_services#nginx-ingress-controller) ou Traefik com K3s (https://docs.k3s.io/networking/networking-services#traefik-ingress-controller). Este documento pressupõe que esses componentes já foram devidamente configurados e que você tem o ponteiro de DNS, por exemplo, usando um curinga, para apontar para os nós do servidor Kubernetes ou IP virtual de entrada para uma resolução de entrada correta.

Nota

A partir do SUSE Edge 3.1, se você usa o K3s em uma configuração de nó de vários servidores, talvez tenha que configurar um VIP com base no MetalLB para entrada. Isso não é necessário no RKE2.

No ambiente de exemplo, outra máquina virtual openSUSE Tumbleweed foi implantada, o cloud-init foi usado para instalar o NGINX como servidor web simples no momento da inicialização e uma mensagem simples foi configurada para ser exibida para verificar se ela funciona conforme o esperado quando uma chamada é feita. Para ver como isso é feito, basta executar `base64 -d` na seção cloud-init na saída a seguir.

Agora vamos criar a máquina virtual:

```
$ kubectl apply -f - <<EOF
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: ingress-example
  namespace: default
spec:
  runStrategy: Always
  template:
    metadata:
      labels:
        app: nginx
    spec:
      domain:
        devices: {}
        machine:
          type: q35
        memory:
          guest: 2Gi
        resources: {}
      volumes:
      - containerDisk:
          image: registry.opensuse.org/home/roxenham/tumbleweed-container-disk/
containerfile/cloud-image:latest
        name: tumbleweed-containerdisk-0
      - cloudInitNoCloud:
          userDataBase64:
 I2Nsb3VkLWNvbmZpZwpkaXNhYmxlX3Jvb3Q6IGZhbHNlCnNzaF9wd2F1dGg6IFRydWUKdXNlcnM6CiAgLSBkZWZhdWx0CiAgLSBuYW
        name: cloudinitdisk
EOF
```

Quando a máquina virtual é iniciada com sucesso, podemos usar o comando `virtctl` para expor `VirtualMachineInstance` com a porta externa `8080` e a porta de destino `80` (na qual o NGINX escuta por padrão). Usamos o comando `virtctl` aqui porque ele reconhece o mapeamento entre o objeto e o pod da máquina virtual. Isso cria um novo serviço:

```
$ virtctl expose vmi ingress-example --port=8080 --target-port=80 --name=ingress-example
Service ingress-example successfully exposed for vmi ingress-example
```

Desse modo, o serviço apropriado será criado automaticamente:

```
$ kubectl get svc/ingress-example
NAME              TYPE           CLUSTER-IP      EXTERNAL-IP       PORT(S)
          AGE
```

```
ingress-example   ClusterIP      10.43.217.19    <none>           8080/TCP
          9s
```

Se depois você usar `kubectl create ingress`, poderemos criar a seguir um objeto de entrada que aponte para esse serviço. Adapte o URL aqui (conhecido como "host" no objeto ingress (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_create/kubectl_create_ingress/)) de acordo com sua configuração do DNS e garanta que ele aponte para a porta `8080`:

```
$ kubectl create ingress ingress-example --rule=ingress-example.suse.local/=ingress-
example:8080
```

Com a configuração correta do DNS, você pode executar curl no URL imediatamente:

```
$ curl ingress-example.suse.local
It works!
```

Vamos fazer a limpeza removendo a máquina virtual e os respectivos recursos de entrada e serviço:

```
$ kubectl delete vm/ingress-example svc/ingress-example ingress/ingress-example
virtualmachine.kubevirt.io "ingress-example" deleted
service "ingress-example" deleted
ingress.networking.k8s.io "ingress-example" deleted
```

21.7 Usando a extensão de IU do Rancher

O SUSE Edge Virtualization oferece uma extensão de IU para o Rancher Manager, que permite o gerenciamento básico de máquinas virtuais usando a interface de usuário do Rancher Dashboard.

21.7.1 Instalação

Consulte as extensões do Rancher Dashboard (*Capítulo 6, Extensões do Rancher Dashboard*) para obter orientação de instalação.

21.7.2 Usando a extensão KubeVirt do Rancher Dashboard

A extensão introduz uma nova seção **KubeVirt** no explorador de clusters, que é adicionada a qualquer cluster gerenciado que tenha o KubeVirt instalado.

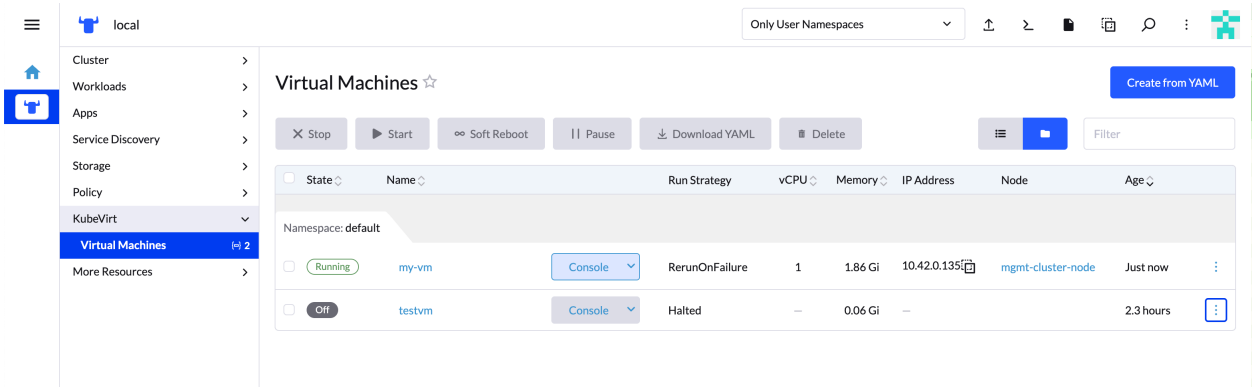
A extensão permite interagir diretamente com os recursos de máquina virtual do KubeVirt para gerenciar o ciclo de vida das máquinas virtuais.

21.7.2.1 Criando uma máquina virtual

1. Navegue até o **explorador de clustersr** clicando no cluster gerenciado habilitado para KubeVirt na navegação esquerda.

2. Navegue até a página **KubeVirt** > **Virtual Machines** (Máquinas virtuais) e clique em `Create from YAML` (Criar do YAML) na parte superior direita da tela.

3. Preencha ou cole uma definição da máquina virtual e clique em `Create` (Criar). Use a definição da máquina virtual da seção Implantando máquinas virtuais como modelo.



21.7.2.2 Ações da máquina virtual

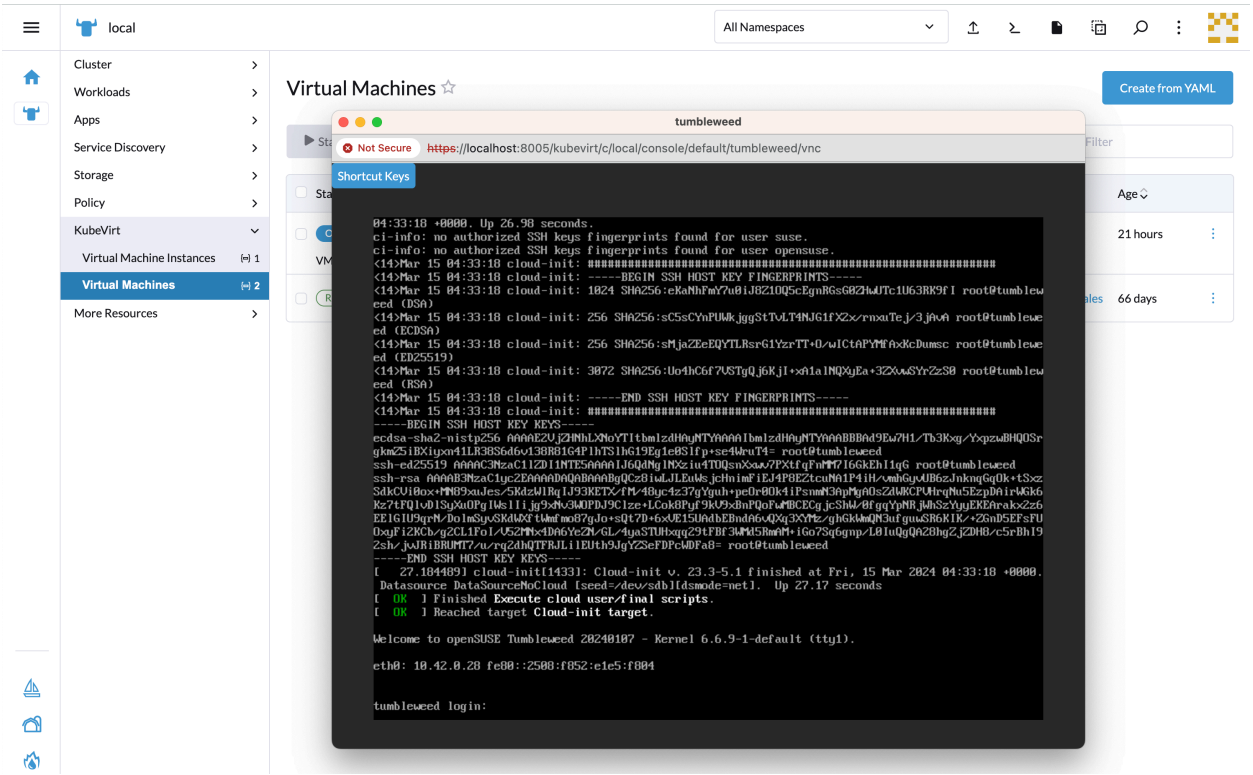
É possível usar o menu de ações acessado pela lista suspensa # à direita de cada máquina virtual para executar as ações iniciar, parar, pausar ou fazer reboot suave. Se preferir, use as ações em grupo na parte superior da lista selecionando as máquinas virtuais nas quais executar a ação.

A execução das ações pode afetar a estratégia de execução da máquina virtual. Consulte a tabela na documentação do KubeVirt (https://kubevirt.io/user-guide/compute/run_strategies/#virtctl) para obter mais detalhes.

21.7.2.3 Acessando o console da máquina virtual

A lista de "máquinas virtuais" oferece a lista suspensa `Console`, que permite conectar-se à máquina usando **VNC ou console serial**. Essa ação está disponível apenas em máquinas que estão em execução.

Em alguns casos, leva um curto tempo para que o console fique disponível em uma máquina virtual que acabou de ser iniciada.



21.8 Instalando com o Edge Image Builder

O SUSE Edge usa o *Capítulo 11, Edge Image Builder* para personalizar as imagens base do sistema operacional SUSE Linux Micro. Siga a *Seção 27.9, "Instalação do KubeVirt e CDI"* para saber sobre a instalação air-gapped tanto do KubeVirt quanto da CDI em clusters Kubernetes provisionados pelo EIB.

22 System Upgrade Controller

Consulte a documentação do System Upgrade Controller (https://github.com/rancher/system-upgrade-controller).

> O System Upgrade Controller (SUC) tem como objetivo oferecer um controlador de upgrade nativo do Kubernetes genérico (para nós). Ele introduz uma nova CRD, o plano, para definir todas as suas políticas/requisitos de upgrade. O plano é uma intenção evidente de mudar nós no cluster.

22.1 Como o SUSE Edge usa o System Upgrade Controller?

O SUSE Edge usa o `SUC` para facilitar diversas operações de "dia 2" relacionadas a upgrades de versão de sistema operacional e Kubernetes em clusters de gerenciamento e downstream.

As operações de "dia 2" são definidas em `planos do SUC`. Com base nesses planos, o `SUC` implanta as cargas de trabalho em cada nó para executar a respectiva operação de "dia 2".

O `SUC` também é usado no *Capítulo 23, Controller de upgrade*. Para saber mais sobre as principais diferenças entre o SUC e o Controller de upgrade, consulte a *Seção 23.2, "Comparação entre Controller de upgrade e System Upgrade Controller"*.

22.2 Instalando o System Upgrade Controller

Importante

A partir do Rancher v2.10.0 (https://github.com/rancher/rancher/releases/tag/v2.10.0), o `System Upgrade Controller` é instalado automaticamente.

Siga as etapas abaixo **apenas** se o seu ambiente **não** é gerenciado pelo Rancher, ou se a versão do Rancher é inferior a `v2.10.0`.

Recomendamos a instalação do SUC pelo Fleet (*Capítulo 8, Fleet*), localizado no repositório suse-edge/fleet-examples (https://github.com/suse-edge/fleet-examples).

Nota

Os recursos disponíveis no repositório `suse-edge/fleet-examples` sempre **devem** ser usados de uma versão válida de fleet-examples (https://github.com/suse-edge/fleet-examples/releases). Para determinar a versão que você precisa usar, consulte as Notas de lançamento (*Seção 52.1, "Resumo"*).

Se não for possível usar o Fleet para instalação do SUC, instale-o por meio do repositório de gráficos Helm do Rancher ou incorpore o gráfico Helm do Rancher ao seu próprio fluxo de trabalho do GitOps de terceiros.

Esta seção aborda o seguinte:

- Instalação do Fleet (*Seção 22.2.1, "Instalação do System Upgrade Controller pelo Fleet"*)
- Instalação do Helm (*Seção 22.2.2, "Instalação do System Upgrade Controller com o Helm"*)

22.2.1 Instalação do System Upgrade Controller pelo Fleet

Com o Fleet, há dois recursos possíveis para serem usados na implantação do SUC:

- GitRepo (https://fleet.rancher.io/ref-gitrepo): para casos de uso em que um servidor Git externo/local está disponível. Para obter instruções de instalação, consulte Instalação do System Upgrade Controller – GitRepo (*Seção 22.2.1.1, "Instalação do System Upgrade Controller – GitRepo"*).
- Bundle (https://fleet.rancher.io/bundle-add): para casos de uso air-gapped que não oferecem suporte à opção de servidor Git local. Para obter instruções de instalação, consulte Instalação do System Upgrade Controller – Bundle (*Seção 22.2.1.2, "Instalação do System Upgrade Controller – Bundle"*).

22.2.1.1 Instalação do System Upgrade Controller – GitRepo

Nota

Também é possível realizar esse processo pela IU do Rancher, se disponível. Para obter mais informações, consulte Accessing Fleet in the Rancher UI (https://ranchermanager.docs.rancher.com/v2.11/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui) (Acessando o Fleet na IU do Rancher).

Em seu cluster de **gerenciamento**:

1. Determine os clusters em que deseja implantar o SUC. Para fazer isso, implante um recurso `GitRepo` do SUC no espaço de trabalho correto do Fleet em seu cluster de **gerenciamento**. Por padrão, o Fleet tem dois espaços de trabalho:

   - `fleet-local`: para recursos que precisam ser implantados no cluster de **gerenciamento**.
   - `fleet-default`: para recursos que precisam ser implantados em clusters **downstream**.
     Para obter mais informações sobre espaços de trabalho do Fleet, consulte a documentação upstream (https://fleet.rancher.io/namespaces#gitrepos-bundles-clusters-clustergroups).

2. Implante o recurso `GitRepo` :

   - Para implantar o SUC no cluster de gerenciamento:

```
kubectl apply -n fleet-local -f - <<EOF
apiVersion: fleet.cattle.io/v1alpha1
kind: GitRepo
metadata:
  name: system-upgrade-controller
spec:
  revision: release-3.3.0
  paths:
  - fleets/day2/system-upgrade-controller
  repo: https://github.com/suse-edge/fleet-examples.git
```

```
EOF
```

- Para implantar o SUC em clusters downstream:

Nota

Antes de implantar o recurso a seguir, você **deve** especificar uma configuração válida de `targets`, para que o Fleet saiba em quais clusters downstream implantar seu recurso. Para obter informações de como mapear para clusters downstream, consulte Mapping to Downstream Clusters (https://fleet.rancher.io/gitrepo-targets) (Mapeando para clusters downstream).

```
kubectl apply -n fleet-default -f - <<EOF
apiVersion: fleet.cattle.io/v1alpha1
kind: GitRepo
metadata:
  name: system-upgrade-controller
spec:
  revision: release-3.3.0
  paths:
  - fleets/day2/system-upgrade-controller
  repo: https://github.com/suse-edge/fleet-examples.git
  targets:
  - clusterSelector: CHANGEME
  # Example matching all clusters:
  # targets:
  # - clusterSelector: {}
EOF
```

3. Valide que o recurso `GitRepo` foi implantado:

```
# Namespace will vary based on where you want to deploy SUC
kubectl get gitrepo system-upgrade-controller -n <fleet-local/fleet-default>

NAME                        REPO                                                COMMIT
          BUNDLEDEPLOYMENTS-READY   STATUS
system-upgrade-controller   https://github.com/suse-edge/fleet-examples.git
 release-3.3.0   1/1
```

4. Valide a implantação do System Upgrade Controller:

```
kubectl get deployment system-upgrade-controller -n cattle-system
NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
```

```
system-upgrade-controller   1/1     1            1           2m20s
```

22.2.1.2 Instalação do System Upgrade Controller – Bundle

Esta seção ilustra como criar e implantar um recurso `Bundle` de uma configuração padrão do Fleet usando fleet-cli (https://fleet.rancher.io/cli/fleet-cli/fleet).

1. Em uma máquina com acesso à rede, faça download de `fleet-cli`:

Nota

Garanta que a versão do fleet-cli do download seja a mesma do Fleet que foi implantado em seu cluster.

   - Para usuários do Mac, existe um Homebrew Formulae fleet-cli (https://formulae.brew.sh/formula/fleet-cli).
   - Para usuários do Linux e Windows, os binários estão presentes como **ativos** para cada versão (https://github.com/rancher/fleet/releases) do Fleet.
     - Linux AMD:

```
curl -L -o fleet-cli https://github.com/rancher/fleet/releases/download/
v0.12.2/fleet-linux-amd64
```

     - Linux ARM:

```
curl -L -o fleet-cli https://github.com/rancher/fleet/releases/download/
v0.12.2/fleet-linux-arm64
```

2. Torne o `fleet-cli` executável:

```
chmod +x fleet-cli
```

3. Clone a versão (https://github.com/suse-edge/fleet-examples/releases) do `suse-edge/fleet-examples` que deseja usar:

```
git clone -b release-3.3.0 https://github.com/suse-edge/fleet-examples.git
```

4. Navegue até a instância do Fleet do SUC, localizada no repositório `fleet-examples`:

```
cd fleet-examples/fleets/day2/system-upgrade-controller
```

5. Determine os clusters em que deseja implantar o SUC. Para fazer isso, implante o bundle do SUC no espaço de trabalho correto do Fleet em seu cluster de gerenciamento. Por padrão, o Fleet tem dois espaços de trabalho:

   - `fleet-local`: para recursos que precisam ser implantados no cluster de **gerenciamento**.
   - `fleet-default`: para recursos que precisam ser implantados em clusters **downstream**.
     Para obter mais informações sobre espaços de trabalho do Fleet, consulte a documentação upstream (https://fleet.rancher.io/namespaces#gitrepos-bundles-clusters-clustergroups).

6. Se você pretende implantar o SUC apenas em clusters downstream, crie um arquivo `targets.yaml` correspondente aos clusters específicos:

```
cat > targets.yaml <<EOF
targets:
- clusterSelector: CHANGEME
EOF
```

   Para obter informações sobre como mapear para clusters downstream, consulte Mapping to Downstream Clusters (https://fleet.rancher.io/gitrepo-targets) (Mapeando para clusters downstream).

7. Avance para a criação do bundle :

Nota

Certifique-se de que você **não** fez download de fleet-cli no diretório `fleet-examples/fleets/day2/system-upgrade-controller`; do contrário, ele será empacotado com o bundle, o que não é aconselhável.

- Para implantar o SUC no cluster de gerenciamento, execute:

```
fleet-cli apply --compress -n fleet-local -o - system-upgrade-controller . >
 system-upgrade-controller-bundle.yaml
```

- Para implantar o SUC em clusters downstream, execute:

```
fleet-cli apply --compress --targets-file=targets.yaml -n fleet-default -o -
 system-upgrade-controller . > system-upgrade-controller-bundle.yaml
```

Para obter mais informações sobre esse processo, consulte Convert a Helm Chart into a Bundle (https://fleet.rancher.io/bundle-add#convert-a-helm-chart-into-a-bundle) (Converter um gráfico Helm em bundle).
Para obter mais informações sobre o comando `fleet-cli apply`, consulte fleet apply (https://fleet.rancher.io/cli/fleet-cli/fleet_apply).

8. Transfira o bundle `system-upgrade-controller-bundle.yaml` para sua máquina do cluster de gerenciamento:

```
scp system-upgrade-controller-bundle.yaml <machine-address>:<filesystem-path>
```

9. No cluster de gerenciamento, implante o bundle `system-upgrade-controller-bundle.yaml`:

```
kubectl apply -f system-upgrade-controller-bundle.yaml
```

10. No cluster de gerenciamento, valide se o bundle foi implantado:

```
# Namespace will vary based on where you want to deploy SUC
kubectl get bundle system-upgrade-controller -n <fleet-local/fleet-default>

NAME                        BUNDLEDEPLOYMENTS-READY   STATUS
system-upgrade-controller   1/1
```

11. De acordo com o espaço de trabalho do Fleet em que você implantou o bundle, navegue até o cluster e valide a implantação do SUC:

Nota

O SUC é sempre implantado no namespace **cattle-system**.

```
kubectl get deployment system-upgrade-controller -n cattle-system
NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
system-upgrade-controller   1/1     1            1           111s
```

22.2.2 Instalação do System Upgrade Controller com o Helm

1. Adicione o repositório de gráficos do Rancher:

```
helm repo add rancher-charts https://charts.rancher.io/
```

2. Implante o gráfico do SUC:

```
helm install system-upgrade-controller rancher-charts/system-upgrade-controller --
version 106.0.0 --set global.cattle.psp.enabled=false -n cattle-system --create-
namespace
```

   Isto instalará a versão 0.15.2 do SUC, que é necessária para a plataforma Edge 3.3.1.

3. Valide a implantação do SUC:

```
kubectl get deployment system-upgrade-controller -n cattle-system
NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
system-upgrade-controller   1/1     1            1           37s
```

22.3 Monitorando os planos do System Upgrade Controller

É possível ver os planos do SUC destas maneiras:

- Pela IU do Rancher (*Seção 22.3.1, "Monitorando os planos do System Upgrade Controller – IU do Rancher"*).
- Por monitoramento manual (*Seção 22.3.2, "Monitorando os planos do System Upgrade Controller – Manual"*) dentro do cluster.

Importante

Os pods implantados para os planos do SUC se mantêm ativos por **15** minutos após a execução bem-sucedida e, depois disso, são removidos pelo job correspondente que os criou. Para ter acesso aos registros do pod após esse período, habilite o registro em seu cluster. Para obter informações sobre como fazer isso no Rancher, consulte Rancher Integration with Logging Services (https://ranchermanager.docs.rancher.com/v2.11/integrations-in-rancher/logging) (Integração do Rancher a serviços de registro).

22.3.1 Monitorando os planos do System Upgrade Controller – IU do Rancher

Para consultar os registros do pod referentes ao plano do SUC específico:

1. No canto superior esquerdo, # → <**nome-do-seu-cluster**>
2. Selecione Workloads → Pods (Cargas de trabalho → Pods).
3. Selecione o menu suspenso `Only User Namespaces` (Somente namespaces de usuário) e adicione o namespace `cattle-system`.
4. Na barra de filtro Pod, escreva o nome do pod do plano do SUC, que terá este formato de gabarito: `apply-<nome_do_plano>-on-<nome_do_nó>`

Nota

Pode haver pods tanto `Completed` (Concluídos) quanto `Unknown` (Desconhecidos) para um plano do SUC específico. Isso é esperado e acontece por causa da natureza de alguns upgrades.

5. Selecione o pod dos quais deseja revisar os registros e navegue até # → **View Logs** (Ver registros).

22.3.2 Monitorando os planos do System Upgrade Controller – Manual

Nota

As etapas abaixo pressupõem que o `kubectl` foi configurado para conectar-se ao cluster em que os **planos do SUC** foram implantados.

1. Liste os planos do **SUC** implantados:

```
kubectl get plans -n cattle-system
```

2. Obtenha o pod para o plano do **SUC**:

```
kubectl get pods -l upgrade.cattle.io/plan=<plan_name> -n cattle-system
```

Nota

Pode haver pods tanto `Completed` (Concluídos) quanto `Unknown` (Desconhecidos) para um plano do SUC específico. Isso é esperado e acontece por causa da natureza de alguns upgrades.

3. Obtenha os registros do pod:

```
kubectl logs <pod_name> -n cattle-system
```

23 Controller de upgrade

Um controlador do Kubernetes que faz upgrades dos seguintes componentes da plataforma SUSE Edge:

- Sistema operacional (SUSE Linux Micro)
- Kubernetes (K3s e RKE2)
- Componentes adicionais (Rancher, Elemental, SUSE Security etc.)

O Controller de upgrade (https://github.com/suse-edge/upgrade-controller) otimiza o processo de upgrade dos componentes mencionados acima, pois encapsula suas complexidades em um único recurso `voltado para o usuário` que funciona como um **acionador** de upgrade. Os usuários precisam apenas configurar esse recurso, e o `Controller de upgrade` cuida do restante.

Nota

Atualmente, o `Controller de upgrade` oferece suporte a upgrades da plataforma SUSE Edge apenas para clusters de **gerenciamento não air-gapped**. Consulte a *Seção 23.7, "Limitações conhecidas"* para obter mais informações.

23.1 Como o SUSE Edge usa o Controller de upgrade?

O **Controller de upgrade** é essencial para automação das operações de "dia 2" (que antes eram manuais) necessárias para fazer upgrade dos clusters de gerenciamento de uma versão de lançamento do SUSE Edge para a seguinte.

Para atingir essa automação, o Controller de upgrade usa ferramentas como o System Upgrade Controller (*Capítulo 22, System Upgrade Controller*) e o Helm Controller (https://github.com/k3s-io/helm-controller/).

Para obter mais detalhes de como o Controller de upgrade funciona, consulte a *Seção 23.4, "Como funciona o Controller de upgrade?"*.

Para saber as limitações conhecidas do Controller de upgrade, consulte a *Seção 23.7, "Limitações conhecidas"*.

Para obter informações sobre a diferença entre o Controller de upgrade e o System Upgrade Controller, consulte a *Seção 23.2, "Comparação entre Controller de upgrade e System Upgrade Controller"*.

23.2 Comparação entre Controller de upgrade e System Upgrade Controller

O System Upgrade Controller (SUC) (*Capítulo 22, System Upgrade Controller*) é uma ferramenta de uso geral que propaga as instruções de upgrade para os nós específicos do Kubernetes.

Apesar de oferecer suporte a algumas operações de "dia 2" para a plataforma SUSE Edge, ele **não** abrange todas elas. Mesmo para as operações com suporte, os usuários ainda precisam configurar, manter e implantar manualmente vários `planos do SUC`, um processo propenso a erros que pode provocar problemas inesperados.

Isso levou à necessidade de uma ferramenta para **automatizar** e **abstrair** a complexidade de gerenciar várias operações de "dia 2" na plataforma SUSE Edge. E, assim, o `Controller de upgrade` foi desenvolvido. Ele simplifica o processo de upgrade com a introdução de um único `recurso voltado para o usuário` que conduz o upgrade. Os usuários precisam gerenciar apenas esse recurso, e o `Controller de upgrade` cuida do restante.

23.3 Instalando o Controller de upgrade

23.3.1 Pré-requisitos

- Helm (https://helm.sh/docs/intro/install/)
- cert-manager (https://cert-manager.io/v1.15-docs/installation/helm/#installing-with-helm)
- System Upgrade Controller (*Seção 22.2, "Instalando o System Upgrade Controller"*)
- Um cluster Kubernetes; seja K3s ou RKE2

23.3.2 Etapas

1. Instale o gráfico Helm do Controller de upgrade no cluster de gerenciamento:

```
helm install upgrade-controller oci://registry.suse.com/edge/charts/upgrade-
controller --version 303.0.1+up0.1.1 --create-namespace --namespace upgrade-
controller-system
```

2. Valide a implantação do Controller de upgrade:

```
kubectl get deployment -n upgrade-controller-system
```

3. Valide o pod do Controller de upgrade:

```
kubectl get pods -n upgrade-controller-system
```

4. Valide os registros do pod do Controller de upgrade:

```
kubectl logs <pod_name> -n upgrade-controller-system
```

23.4 Como funciona o Controller de upgrade?

Para fazer upgrade de uma versão do Edge, o Controller de upgrade lança dois novos recursos personalizados (https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/) do Kubernetes:

- UpgradePlan (*Seção 23.5.1, "UpgradePlan"*): criado pelo usuário; armazena as configurações referentes ao upgrade de uma versão do Edge.
- ReleaseManifest (*Seção 23.5.2, "ReleaseManifest"*): criado pelo Controller de upgrade; armazena as versões de componente específicas a uma determinada versão de lançamento do Edge. **Os usuários não devem editar esse arquivo.**

O Controller de upgrade cria um recurso `ReleaseManifest` que armazena os dados do componente referentes à versão de lançamento do Edge especificada pelo usuário na propriedade `releaseVersion` do recurso `UpgradePlan`.

Usando os dados do componente do `ReleaseManifest`, o Controller de upgrade faz upgrade do componente da versão do Edge na seguinte ordem:

1. Sistema operacional (SO) (*Seção 23.4.1, "Upgrade do sistema operacional"*).
2. Kubernetes (*Seção 23.4.2, "Upgrade do Kubernetes"*).
3. Componentes adicionais (*Seção 23.4.3, "Upgrades de componentes adicionais"*).

Nota

Durante o processo de upgrade, o Controller de upgrade envia continuamente as informações de upgrade para o `UpgradePlan` criado. Para obter mais informações sobre como acompanhar o processo de upgrade, consulte Acompanhando o processo de upgrade (*Seção 23.6, "Acompanhando o processo de upgrade"*).

23.4.1 Upgrade do sistema operacional

Para fazer upgrade do sistema operacional, o Controller de upgrade cria planos do SUC (*Capítulo 22, System Upgrade Controller*) com o seguinte gabarito de nomenclatura:

- Para planos do SUC relacionados a upgrades de sistema operacional de nó do plano de controle: `control-plane-<nome-do-so>-<versão-do-so>-<sufixo>`.
- Para planos do SUC relacionados a upgrades de sistema operacional de nó do worker: `workers-<nome-do-so>-<versão-do-so>-<sufixo>`.

Com base nesses planos, o SUC cria as cargas de trabalho em cada nó do cluster que faz o upgrade real do sistema operacional.

Dependendo do `ReleaseManifest`, o upgrade de sistema operacional pode incluir:

- Atualizações somente de pacotes: para casos de uso em que a versão do sistema operacional não é alterada entre os lançamentos do Edge.
- Migração completa do sistema operacional: para casos de uso em que a versão do sistema operacional é alterada entre os lançamentos do Edge.

O upgrade é feito **um** nó de cada vez, começando primeiro pelos nós do plano de controle. Apenas quando o upgrade do nó do plano de controle é concluído que o upgrade dos nós do worker é iniciado.

Nota

O Controller de upgrade configura os planos do SUC de sistema operacional para fazer uma drenagem (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_drain/) dos nós do cluster, se o cluster tiver mais de um **um** nó do tipo especificado.

Para os clusters em que os nós do plano de controle são **mais de** um, e existe **apenas um** nó do worker, a drenagem será realizada somente para os nós do plano de controle e vice-versa.

Para obter informações sobre como desabilitar completamente as drenagens de nós, consulte a seção UpgradePlan (*Seção 23.5.1, "UpgradePlan"*).

23.4.2 Upgrade do Kubernetes

Para fazer upgrade da distribuição Kubernetes de um cluster, o Controller de upgrade cria planos do SUC (*Capítulo 22, System Upgrade Controller*) com o seguinte gabarito de nomenclatura:

- Para os planos do SUC relacionados a ugrades do Kubernetes em nós do plano de controle: `control-plane-<versão-do-k8s>-<sufixo>`.
- Para os planos do SUC relacionados a upgrades do Kubernetes em nós do worker: `workers-<versão-do-k8s>-<sufixo>`.

Com base nesses planos, o SUC cria as cargas de trabalho em cada nó do cluster que faz o upgrade real do Kubernetes.

O upgrade do Kubernetes é feito **um** nó de cada vez, começando primeiro pelos nós do plano de controle. Apenas quando o upgrade do nó do plano de controle é concluído que o upgrade dos nós do worker é iniciado.

Nota

O Controller de upgrade configura os planos do SUC do Kubernetes para fazer uma drenagem (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_drain/) dos nós do cluster, se o cluster tiver mais de um **um** nó do tipo especificado.

Para os clusters em que os nós do plano de controle são **mais de** um, e existe **apenas um** nó do worker, a drenagem será realizada somente para os nós do plano de controle e vice-versa.

Para obter informações sobre como desabilitar completamente as drenagens de nós, consulte a *Seção 23.5.1, "UpgradePlan"*.

23.4.3 Upgrades de componentes adicionais

Atualmente, todos os componentes adicionais são instalados por gráficos Helm. Para ver a lista completa dos componentes de uma versão específica, consulte as Notas de lançamento (*Seção 52.1, "Resumo"*).

Para gráficos Helm implantados pelo EIB (*Capítulo 11, Edge Image Builder*), o Controller de upgrade atualiza o HelmChart CR (https://docs.rke2.io/helm#using-the-helm-crd) existente de cada componente.

Para gráficos Helm implantados fora do EIB, o Controller de upgrade cria um recurso `HelmChart` para cada componente.

Após a criação/atualização do recurso `HelmChart`, o Controller de upgrade se baseia no helm-controller (https://github.com/k3s-io/helm-controller/) para detectar essa alteração e prosseguir com o upgrade real do componente.

O upgrade dos gráficos será feito em sequência de acordo com a ordem apresentada no `ReleaseManifest`. É possível também especificar mais valores no `UpgradePlan`. Se uma versão de gráfico permanecer inalterada no novo lançamento do SUSE Edge, o upgrade dele não será feito. Para obter mais informações sobre isso, consulte a *Seção 23.5.1, "UpgradePlan"*.

23.5 Extensões de API Kubernetes

Extensões para a API Kubernetes lançadas pelo Controller de upgrade.

23.5.1 UpgradePlan

O Controller de upgrade lança um novo recurso personalizado (https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/) do Kubernetes chamado `UpgradePlan`.

O `UpgradePlan` atua como um mecanismo de instrução para o Controller de upgrade e oferece suporte às seguintes configurações:

- `releaseVersion`: a versão de lançamento do Edge para a qual o upgrade do cluster deve ser feito. Ela deve seguir o controle de versão semântico (https://semver.org) e ser recuperada das Notas de lançamento (*Seção 52.1, "Resumo"*).
- `disableDrain` (**opcional**): indica se o Controller de upgrade deve ou não desabilitar as drenagens (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_drain/) de nós. Útil quando você tem cargas de trabalho com orçamentos de interrupção (https://kubernetes.io/docs/tasks/run-application/configure-pdb/).
  - Exemplo de desabilitação de drenagem de nó do plano de controle:

```
spec:
  disableDrain:
    controlPlane: true
```

  - Exemplo de desabilitação de drenagem de nó do plano de controle e do worker:

```
spec:
  disableDrain:
    controlPlane: true
    worker: true
```

- `helm` (**opcional**): especifica valores adicionais para componentes instalados pelo Helm.

Atenção

Somente é aconselhável usar esse campo para valores que sejam críticos aos upgrades. As atualizações nos valores de gráficos padrão deverão ser feitas depois que o upgrade dos respectivos gráficos for feito para a versão seguinte.

  - Exemplo:

```
spec:
  helm:
  - chart: foo
    values:
```

```
      bar: baz
```

23.5.2 ReleaseManifest

O Controller de upgrade lança um novo recurso personalizado (https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/) do Kubernetes chamado `ReleaseManifest`.

O recurso `ReleaseManifest` é criado pelo Controller de upgrade e armazena os dados de componente referentes a **uma** versão de lançamento específica do Edge. Isso significa que o upgrade de cada versão de lançamento do Edge será representado por um recurso `ReleaseManifest` diferente.

Atenção

O Controller de upgrade sempre deve criar o ReleaseManifest.

Não é aconselhável criar manualmente ou editar os recursos `ReleaseManifest` . O usuários que decidirem por esse método devem fazê-lo **por sua conta e risco**.

O ReleaseManifest inclui os seguintes dados de componente, mas sem limitação:

- Dados de sistema operacional: versão, arquiteturas suportadas, dados adicionais de upgrade etc.
- Dados de distribuição Kubernetes: versões suportadas do RKE2 (https://docs.rke2.io)/K3s (https://k3s.io)
- Dados de componentes adicionais: dados de gráficos Helm do SUSE (local, versão, nome etc.)

Para ver um exemplo da aparência de um ReleaseManifest, consulte a documentação upstream (https://github.com/suse-edge/upgrade-controller/blob/main/config/samples/lifecycle_v1alpha1_releasemanifest.yaml). *Tenha em mente que se trata apenas de um exemplo, sem a intenção de criar um recurso* `ReleaseManifest` *válido.*

23.6 Acompanhando o processo de upgrade

Esta seção apresenta como acompanhar e depurar o processo de upgrade que o Controller de upgrade inicia após a criação do recurso `UpgradePlan`.

23.6.1 Geral

Acesse as informações gerais sobre o estado do processo de upgrade em Condições de status do UpgradePlan.

Veja o status do recurso UpgradePlan da seguinte maneira:

```
kubectl get upgradeplan <upgradeplan_name> -n upgrade-controller-system -o yaml
```

Exemplo de execução do UpgradePlan:

```
apiVersion: lifecycle.suse.com/v1alpha1
kind: UpgradePlan
metadata:
  name: upgrade-plan-mgmt
  namespace: upgrade-controller-system
spec:
  releaseVersion: 3.3.1
status:
  conditions:
  - lastTransitionTime: "2024-10-01T06:26:27Z"
    message: Control plane nodes are being upgraded
    reason: InProgress
    status: "False"
    type: OSUpgraded
  - lastTransitionTime: "2024-10-01T06:26:27Z"
    message: Kubernetes upgrade is not yet started
    reason: Pending
    status: Unknown
    type: KubernetesUpgraded
  - lastTransitionTime: "2024-10-01T06:26:27Z"
    message: Rancher upgrade is not yet started
    reason: Pending
    status: Unknown
    type: RancherUpgraded
  - lastTransitionTime: "2024-10-01T06:26:27Z"
```

```
  - lastTransitionTime: "2024-10-01T06:26:27Z"
    message: MetalLB upgrade is not yet started
    reason: Pending
    status: Unknown
    type: MetalLBUpgraded
  - lastTransitionTime: "2024-10-01T06:26:27Z"
    message: CDI upgrade is not yet started
    reason: Pending
    status: Unknown
    type: CDIUpgraded
  - lastTransitionTime: "2024-10-01T06:26:27Z"
    message: KubeVirt upgrade is not yet started
    reason: Pending
    status: Unknown
    type: KubeVirtUpgraded
  - lastTransitionTime: "2024-10-01T06:26:27Z"
    message: NeuVector upgrade is not yet started
    reason: Pending
    status: Unknown
    type: NeuVectorUpgraded
  - lastTransitionTime: "2024-10-01T06:26:27Z"
    message: EndpointCopierOperator upgrade is not yet started
    reason: Pending
    status: Unknown
    type: EndpointCopierOperatorUpgraded
  - lastTransitionTime: "2024-10-01T06:26:27Z"
    message: Elemental upgrade is not yet started
    reason: Pending
    status: Unknown
    type: ElementalUpgraded
  - lastTransitionTime: "2024-10-01T06:26:27Z"
    message: SRIOV upgrade is not yet started
    reason: Pending
    status: Unknown
    type: SRIOVUpgraded
  - lastTransitionTime: "2024-10-01T06:26:27Z"
```

```
    reason: Pending
```

```
    status: Unknown
    type: RancherTurtlesUpgraded
  observedGeneration: 1
  sucNameSuffix: 90315a2b6d
```

Aqui você vê todos os componentes para os quais o Controller de upgrade tentará programar um upgrade. Cada condição segue o gabarito abaixo:

- `lastTransitionTime`: a hora em que a condição do componente mudou de um status para outro pela última vez.
- `message`: mensagem que indica o estado atual do upgrade da condição do componente específico.
- `reason`: o estado atual do upgrade da condição do componente específico. Os possíveis `motivos` incluem:
  - `Succeeded`: o upgrade do componente específico foi bem-sucedido.
  - `Failed`: falha no upgrade do componente específico.
  - `InProgress`: o upgrade do componente específico está em andamento.
  - `Pending`: o upgrade do componente específico ainda não foi programado.
  - `Skipped`: o componente específico não foi encontrado no cluster, portanto, o upgrade dele será ignorado.
  - `Error`: o componente específico encontrou um erro temporário.
- `status`: o status do `type` (tipo) da condição atual, que pode ser `True` (Verdadeiro), `False` (Falso) ou `Unknown` (Desconhecido).
- `type`: indicador do componente que está passando por upgrade.

O Controller de upgrade cria planos do SUC para as condições de componente do tipo `OSUpgraded` e `KubernetesUpgraded`. Para acompanhar em mais detalhes os planos do SUC criados para esses componentes, consulte a *Seção 22.3, "Monitorando os planos do System Upgrade Controller"*.

É possível acompanhar em mais detalhes todos os outros tipos de condição de componente visualizando os recursos criados para eles pelo helm-controller (https://github.com/k3s-io/helm-controller/). Para obter mais informações, consulte a *Seção 23.6.2, "Helm Controller"*.

É possível marcar um UpgradePlan programado pelo Controller de upgrade como `successful` (bem-sucedido) se:

1. Não há condições de componente `Pending` ou `InProgress`.
2. A propriedade `lastSuccessfulReleaseVersion` aponta para a `releaseVersion`, que é especificada na configuração do UpgradePlan. *O Controller de upgrade adiciona essa propriedade ao status do UpgradePlan após o processo de upgrade bem-sucedido.*

Exemplo de `UpgradePlan` bem-sucedido:

```
apiVersion: lifecycle.suse.com/v1alpha1
kind: UpgradePlan
metadata:
  name: upgrade-plan-mgmt
  namespace: upgrade-controller-system
spec:
  releaseVersion: 3.3.1
status:
  conditions:
  - lastTransitionTime: "2024-10-01T06:26:48Z"
    message: All cluster nodes are upgraded
    reason: Succeeded
    status: "True"
    type: OSUpgraded
  - lastTransitionTime: "2024-10-01T06:26:59Z"
    message: All cluster nodes are upgraded
    reason: Succeeded
    status: "True"
    type: KubernetesUpgraded
  - lastTransitionTime: "2024-10-01T06:27:13Z"
    message: Chart rancher upgrade succeeded
    reason: Succeeded
    status: "True"
    type: RancherUpgraded
  - lastTransitionTime: "2024-10-01T06:27:13Z"
    message: Chart longhorn is not installed
    reason: Skipped
    status: "False"
```

```
    message: Chart cdi is not installed
    reason: Skipped
    status: "False"
    type: CDIUpgraded
  - lastTransitionTime: "2024-10-01T06:27:13Z"
    message: Chart kubevirt is not installed
    reason: Skipped
    status: "False"
    type: KubeVirtUpgraded
  - lastTransitionTime: "2024-10-01T06:27:13Z"
    message: Chart neuvector-crd is not installed
    reason: Skipped
    status: "False"
    type: NeuVectorUpgraded
  - lastTransitionTime: "2024-10-01T06:27:14Z"
    message: Specified version of chart endpoint-copier-operator is already installed
    reason: Skipped
    status: "False"
    type: EndpointCopierOperatorUpgraded
  - lastTransitionTime: "2024-10-01T06:27:14Z"
    message: Chart elemental-operator upgrade succeeded
    reason: Succeeded
    status: "True"
    type: ElementalUpgraded
  - lastTransitionTime: "2024-10-01T06:27:15Z"
    message: Chart sriov-crd is not installed
    reason: Skipped
    status: "False"
    type: SRIOVUpgraded
  - lastTransitionTime: "2024-10-01T06:27:16Z"
    message: Chart akri is not installed
    reason: Skipped
    status: "False"
    type: AkriUpgraded
  - lastTransitionTime: "2024-10-01T06:27:19Z"
    message: Chart metal3 is not installed
```

23.6.2 Helm Controller

Esta seção explica como acompanhar os recursos criados pelo helm-controller (https://github.com/k3s-io/helm-controller/).

Nota

As etapas a seguir consideram que o `kubectl` foi configurado para conectar-se ao cluster no qual o Controller de upgrade foi implantado.

1. Localize o recurso `HelmChart` para o componente específico:

```
kubectl get helmcharts -n kube-system
```

2. Usando o nome do recurso `HelmChart`, localize o pod de upgrade que foi criado pelo `helm-controller`:

```
kubectl get pods -l helmcharts.helm.cattle.io/chart=<helmchart_name> -n kube-system

# Example for Rancher
kubectl get pods -l helmcharts.helm.cattle.io/chart=rancher -n kube-system
NAME                          READY   STATUS      RESTARTS   AGE
helm-install-rancher-tv9wn    0/1     Completed   0          16m
```

3. Exiba os registros do pod específico do componente:

```
kubectl logs <pod_name> -n kube-system
```

23.7 Limitações conhecidas

- Os upgrades de clusters downstream ainda não são gerenciados pelo Controller de upgrade. Para obter mais informações sobre como fazer upgrade de clusters downstream, consulte o *Capítulo 36, Clusters downstream*.
- O Controller de upgrade espera que os gráficos Helm adicionais do SUSE Edge implantados pelo EIB (*Capítulo 11, Edge Image Builder*) tenham o HelmChart CR (https://docs.rke2.io/helm#using-the-helm-crd) implantado no namespace `kube-system`. Para isso, configure a propriedade `installationNamespace` no arquivo de definição do EIB. Para obter mais informações, consulte a documentação upstream (https://github.com/suse-edge/edge-image-builder/blob/main/docs/building-images.md#kubernetes).

- Atualmente, o Controller de upgrade não tem como determinar a versão de lançamento do Edge que está em execução no cluster de gerenciamento. Insira uma versão de lançamento do Edge superior àquela que está em execução no cluster.
- Atualmente, o Controller de upgrade oferece suporte apenas a upgrades de ambientes **não air-gapped**. Os upgrades **air-gapped** ainda não são possíveis.

24 SUSE Multi-Linux Manager

O SUSE Multi-Linux Manager está incluído no SUSE Edge para oferecer a automação e o controle que mantêm o sistema operacional SUSE Linux Micro subjacente sempre atualizado em todos os nós da implantação de borda.

Para obter mais informações, consulte o *Capítulo 4, SUSE Multi-Linux Manager* e a documentação do SUSE Multi-Linux Manager (https://documentation.suse.com/suma/5.0/en/suse-manager/index.html).

III Guias de procedimentos

Guias de procedimentos e melhores práticas

25 MetalLB no K3s (usando o modo de camada 2)

O MetalLB é uma implementação de balanceador de carga para clusters Kubernetes bare metal, usando protocolos de roteamento padrão.

Neste guia, demonstramos como implantar o MetalLB no modo de camada 2 (L2).

25.1 Por que usar este método

O MetalLB é uma opção atrativa para balanceamento de carga em clusters Kubernetes bare metal por vários motivos:

1. Integração nativa com o Kubernetes: o MetalLB se integra perfeitamente ao Kubernetes, o que facilita a implantação e o gerenciamento usando as ferramentas e práticas conhecidas do Kubernetes.

2. Compatibilidade com bare metal: ao contrário dos balanceadores de carga com base na nuvem, o MetalLB foi especificamente projetado para implantações no local, em que os balanceadores de carga tradicionais podem não estar disponíveis ou ser viáveis.

3. Compatível com vários protocolos: o MetalLB é compatível com os modos de camada 2 e BGP (Border Gateway Protocol), proporcionando flexibilidade de acordo com as diferentes arquiteturas e requisitos de rede.

4. Alta disponibilidade: ao distribuir as responsabilidades de balanceamento de carga por vários nós, o MetalLB garante alta disponibilidade e confiabilidade aos seus serviços.

5. Escalabilidade: o MetalLB processa implantações de grande escala, ajustando a escala no cluster Kubernetes para atender à crescente demanda.

No modo de camada 2, um nó assume a responsabilidade de anunciar um serviço para a rede local. Da perspectiva da rede, simplesmente parece que a máquina tem vários endereços IP atribuídos à interface de rede.

A maior vantagem do modo de camada 2 é a sua universalidade: ele funciona em qualquer rede Ethernet, sem a necessidade de hardware especial nem de roteadores sofisticados.

25.2 MetalLB no K3s (usando L2)

Neste início rápido, o modo L2 será usado. Isso significa que não precisamos de nenhum equipamento de rede especial, mas de três IPs livres no intervalo de rede.

25.3 Pré-requisitos

- Um cluster K3s no qual o MetalLB será implantado.

Atenção

O K3S vem com o próprio balanceador de carga de serviço chamado Klipper. Você precisa desabilitá-lo para executar o MetalLB (https://metallb.universe.tf/configuration/k3s/). Para desabilitar o Klipper, o K3s precisa ser instalado com o sinalizador `--disable=servicelb`.

- Helm
- Três endereços IP livres dentro do intervalo de rede. Neste exemplo, `192.168.122.10-192.168.122.12`

Importante

Garanta que esses endereços IP não estejam atribuídos. Em um ambiente DHCP, esses endereços não devem fazer parte do pool DHCP para evitar atribuições duplas.

25.4 Implantação

Vamos usar o gráfico Helm do MetalLB publicado como parte da solução SUSE Edge:

```
helm install \
  metallb oci://registry.suse.com/edge/charts/metallb \
  --namespace metallb-system \
  --create-namespace

while ! kubectl wait --for condition=ready -n metallb-system $(kubectl get\
 pods -n metallb-system -l app.kubernetes.io/component=controller -o name)\
 --timeout=10s; do
```

```
 sleep 2
done
```

25.5 Configuração

Neste ponto, a instalação foi concluída. Agora é hora de configurar (https://metallb.universe.tf/configuration/) usando nossos valores de exemplo:

```
cat <<-EOF | kubectl apply -f -
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: ip-pool
  namespace: metallb-system
spec:
  addresses:
  - 192.168.122.10/32
  - 192.168.122.11/32
  - 192.168.122.12/32
EOF
```

```
cat <<-EOF | kubectl apply -f -
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: ip-pool-l2-adv
  namespace: metallb-system
spec:
  ipAddressPools:
  - ip-pool
EOF
```

Agora ele está pronto para uso. É possível personalizar vários componentes no modo L2, por exemplo:

- Serviços IPv6 e de pilha dupla (https://metallb.universe.tf/usage/#ipv6-and-dual-stack-services)
- Controlar a alocação de endereços automática (https://metallb.universe.tf/configuration/_advanced_ipaddresspool_configuration/#controlling-automatic-address-allocation)
- Reduzir o escopo da alocação de endereços para namespaces e serviços específicos (https://metallb.universe.tf/configuration/_advanced_ipaddresspool_configuration/#reduce-scope-of-address-allocation-to-specific-namespace-and-service)

- Limitando o conjunto de nós dos quais o serviço pode ser anunciado (https://metallb.universe.tf/configuration/_advanced_l2_configuration/#limiting-the-set-of-nodes-where-the-service-can-be-announced-from)
- Especificar as interfaces de rede das quais o IP do LB pode ser anunciado (https://metallb.universe.tf/configuration/_advanced_l2_configuration/#specify-network-interfaces-that-lb-ip-can-be-announced-from)

E muito mais para BGP (https://metallb.universe.tf/configuration/_advanced_bgp_configuration/).

25.5.1 Traefik e MetalLB

O Traefik é implantado por padrão com o K3s (ele pode ser desabilitado (https://docs.k3s.io/networking#traefik-ingress-controller) com `--disable=traefik`) e exposto como `LoadBalancer` (para uso com o Klipper). No entanto, como o Klipper precisa ser desabilitado, o serviço Traefik para entrada ainda é um tipo de `LoadBalancer`. Sendo assim, no momento da implantação do MetalLB, o primeiro IP será automaticamente atribuído à entrada do Traefik.

```
# Before deploying MetalLB
kubectl get svc -n kube-system traefik
NAME      TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)                      AGE
traefik   LoadBalancer   10.43.44.113   <pending>     80:31093/TCP,443:32095/TCP   28s
# After deploying MetalLB
kubectl get svc -n kube-system traefik
NAME      TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)                      AGE
traefik   LoadBalancer   10.43.44.113   192.168.122.10   80:31093/TCP,443:32095/TCP
 3m10s
```

Isso será aplicado mais adiante (*Seção 25.6.1, "Ingress com MetalLB"*) no processo.

25.6 Uso

Vamos criar uma implantação de exemplo:

```
cat <<- EOF | kubectl apply -f -
---
apiVersion: v1
kind: Namespace
metadata:
  name: hello-kubernetes
```

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: hello-kubernetes
  namespace: hello-kubernetes
  labels:
    app.kubernetes.io/name: hello-kubernetes
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-kubernetes
  namespace: hello-kubernetes
  labels:
    app.kubernetes.io/name: hello-kubernetes
spec:
  replicas: 2
  selector:
    matchLabels:
      app.kubernetes.io/name: hello-kubernetes
  template:
    metadata:
      labels:
        app.kubernetes.io/name: hello-kubernetes
    spec:
      serviceAccountName: hello-kubernetes
      containers:
        - name: hello-kubernetes
          image: "paulbouwer/hello-kubernetes:1.10"
          imagePullPolicy: IfNotPresent
          ports:
            - name: http
              containerPort: 8080
              protocol: TCP
          livenessProbe:
            httpGet:
              path: /
              port: http
          readinessProbe:
            httpGet:
              path: /
              port: http
          env:
          - name: HANDLER_PATH_PREFIX
            value: ""
          - name: RENDER_PATH_PREFIX
```

```
            value: ""
          - name: KUBERNETES_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
          - name: KUBERNETES_POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: KUBERNETES_NODE_NAME
            valueFrom:
              fieldRef:
                fieldPath: spec.nodeName
          - name: CONTAINER_IMAGE
            value: "paulbouwer/hello-kubernetes:1.10"
EOF
```

E, por fim, o serviço:

```
cat <<- EOF | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
  name: hello-kubernetes
  namespace: hello-kubernetes
  labels:
    app.kubernetes.io/name: hello-kubernetes
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: http
      protocol: TCP
      name: http
  selector:
    app.kubernetes.io/name: hello-kubernetes
EOF
```

Vamos vê-lo em ação:

```
kubectl get svc -n hello-kubernetes
NAME               TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)        AGE
hello-kubernetes   LoadBalancer   10.43.127.75   192.168.122.11   80:31461/TCP   8s

curl http://192.168.122.11
<!DOCTYPE html>
<html>
<head>
```

```
    <title>Hello Kubernetes!</title>
    <link rel="stylesheet" type="text/css" href="/css/main.css">
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Ubuntu:300" >
</head>
<body>

  <div class="main">
    <img src="/images/kubernetes.png"/>
    <div class="content">
      <div id="message">
  Hello world!
</div>
<div id="info">
  <table>
    <tr>
      <th>namespace:</th>
      <td>hello-kubernetes</td>
    </tr>
    <tr>
      <th>pod:</th>
      <td>hello-kubernetes-7c8575c848-2c6ps</td>
    </tr>
    <tr>
      <th>node:</th>
      <td>allinone (Linux 5.14.21-150400.24.46-default)</td>
    </tr>
  </table>
</div>
<div id="footer">
  paulbouwer/hello-kubernetes:1.10 (linux/amd64)
</div>
    </div>
  </div>

</body>
</html>
```

25.6.1 Ingress com MetalLB

Como o Traefik já atua como controlador de entrada, podemos expor o tráfego HTTP/HTTPS por um objeto `Ingress`, por exemplo:

```
IP=$(kubectl get svc -n kube-system traefik -o
 jsonpath="{.status.loadBalancer.ingress[0].ip}")
cat <<- EOF | kubectl apply -f -
```

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hello-kubernetes-ingress
  namespace: hello-kubernetes
spec:
  rules:
  - host: hellok3s.${IP}.sslip.io
    http:
      paths:
        - path: "/"
          pathType: Prefix
          backend:
            service:
              name: hello-kubernetes
              port:
                name: http
EOF
```

E depois:

```
curl http://hellok3s.${IP}.sslip.io
<!DOCTYPE html>
<html>
<head>
    <title>Hello Kubernetes!</title>
    <link rel="stylesheet" type="text/css" href="/css/main.css">
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Ubuntu:300" >
</head>
<body>

  <div class="main">
    <img src="/images/kubernetes.png"/>
    <div class="content">
      <div id="message">
  Hello world!
</div>
<div id="info">
  <table>
    <tr>
      <th>namespace:</th>
      <td>hello-kubernetes</td>
    </tr>
    <tr>
      <th>pod:</th>
      <td>hello-kubernetes-7c8575c848-fvqm2</td>
    </tr>
    <tr>
```

```
      <th>node:</th>
      <td>allinone (Linux 5.14.21-150400.24.46-default)</td>
    </tr>
  </table>
</div>
<div id="footer">
  paulbouwer/hello-kubernetes:1.10 (linux/amd64)
</div>
    </div>
  </div>

</body>
</html>
```

Verifique se o MetalLB funciona corretamente:

```
% arping hellok3s.${IP}.sslip.io

ARPING 192.168.64.210
60 bytes from 92:12:36:00:d3:58 (192.168.64.210): index=0 time=1.169 msec
60 bytes from 92:12:36:00:d3:58 (192.168.64.210): index=1 time=2.992 msec
60 bytes from 92:12:36:00:d3:58 (192.168.64.210): index=2 time=2.884 msec
```

No exemplo acima, o tráfego flui da seguinte maneira:

1. `hellok3s.${IP}.sslip.io` é resolvido para o IP atual.
2. Em seguida, o tráfego é processado pelo pod `metallb-speaker`.
3. `metallb-speaker` redireciona o tráfego para o controlador `traefik`.
4. Por fim, o Traefik encaminha a solicitação para o serviço `hello-kubernetes`.

26 MetalLB na frente do servidor da API Kubernetes

Este guia demonstra como usar um serviço do MetalLB para expor a API RKE2/K3s externamente em um cluster de alta disponibilidade com três nós do plano de controle. Para isso, um serviço do Kubernetes do tipo `LoadBalancer` e endpoints serão manualmente criados. Os endpoints armazenam os IPs de todos os nós do plano de controle disponíveis no cluster. Para que o endpoint sempre se mantenha sincronizado com os eventos que ocorrem no cluster (adicionar/remover um nó ou quando um nó fica offline), o Endpoint Copier Operator (*Capítulo 20, Endpoint Copier Operator*) é implantado. O operador monitora os eventos ocorridos no endpoint padrão `kubernetes` e atualiza aquele gerenciado automaticamente para mantê-los sincronizados. Como o serviço gerenciado é do tipo `LoadBalancer`, o MetalLB o atribui a um `ExternalIP` estático. Esse `ExternalIP` será usado para comunicação com o API Server.

26.1 Pré-requisitos

- Três hosts para implantar o RKE2/K3s em cima.
  - Certifique-se de que os hosts tenham nomes diferentes.
  - Para teste, eles podem ser máquinas virtuais.
- Pelo menos 2 IPs disponíveis na rede (um para o Traefik/Nginx e outro para o serviço gerenciado).
- Helm

26.2 Instalando o RKE2/K3s

Nota

Se você não deseja usar um cluster novo, mas um existente, ignore esta etapa e avance para a próxima.

Primeiramente, é necessário reservar um IP livre para usar mais adiante como `ExternalIP` do serviço gerenciado.

Acesse o primeiro host por SSH e instale a distribuição desejada no modo de cluster.

Para RKE2:

```
# Export the free IP mentioned above
export VIP_SERVICE_IP=<ip>

curl -sfL https://get.rke2.io | INSTALL_RKE2_EXEC="server \
 --write-kubeconfig-mode=644 --tls-san=${VIP_SERVICE_IP} \
 --tls-san=https://${VIP_SERVICE_IP}.sslip.io" sh -

systemctl enable rke2-server.service
systemctl start rke2-server.service

# Fetch the cluster token:
RKE2_TOKEN=$(tr -d '\n' < /var/lib/rancher/rke2/server/node-token)
```

Para K3s:

```
# Export the free IP mentioned above
export VIP_SERVICE_IP=<ip>
export INSTALL_K3S_SKIP_START=false

curl -sfL https://get.k3s.io | INSTALL_K3S_EXEC="server --cluster-init \
 --disable=servicelb --write-kubeconfig-mode=644 --tls-san=${VIP_SERVICE_IP} \
 --tls-san=https://${VIP_SERVICE_IP}.sslip.io" K3S_TOKEN=foobar sh -
```

Nota

Verifique se o sinalizador `--disable=servicelb` foi inserido no comando `k3s server`.

Importante

A partir de agora, os comandos devem ser executados na máquina local.

Para acessar o servidor da API de um ambiente externo, o IP da VM do RKE2/K3s será usado.

```
# Replace <node-ip> with the actual IP of the machine
export NODE_IP=<node-ip>
export KUBE_DISTRIBUTION=<k3s/rke2>

scp ${NODE_IP}:/etc/rancher/${KUBE_DISTRIBUTION}/${KUBE_DISTRIBUTION}.yaml ~/.kube/config
 && sed \
```

```
-i '' "s/127.0.0.1/${NODE_IP}/g" ~/.kube/config && chmod 600 ~/.kube/config
```

26.3 Configurando um cluster existente

Nota

Esta etapa é válida apenas se você pretende usar um cluster RKE2/K3s existente.

Modifique os sinalizadores `tls-san` para usar um cluster existente. Além disso, o LB `servicelb` deve ser desabilitado no K3s.

Para alterar os sinalizadores dos servidores RKE2 ou K3s, você precisa modificar o arquivo `/etc/systemd/system/rke2.service` ou `/etc/systemd/system/k3s.service` em todas as VMs no cluster, dependendo da distribuição.

Insira os sinalizadores em `ExecStart`. Por exemplo:

Para RKE2:

```
# Replace the <vip-service-ip> with the actual ip
ExecStart=/usr/local/bin/rke2 \
    server \
        '--write-kubeconfig-mode=644' \
        '--tls-san=<vip-service-ip>' \
        '--tls-san=https://<vip-service-ip>.sslip.io' \
```

Para K3s:

```
# Replace the <vip-service-ip> with the actual ip
ExecStart=/usr/local/bin/k3s \
    server \
        '--cluster-init' \
        '--write-kubeconfig-mode=644' \
        '--disable=servicelb' \
        '--tls-san=<vip-service-ip>' \
        '--tls-san=https://<vip-service-ip>.sslip.io' \
```

Na sequência, os seguintes comandos devem ser executados para carregar as novas configurações:

```
systemctl daemon-reload
systemctl restart ${KUBE_DISTRIBUTION}
```

26.4 Instalando o MetalLB

Para implantar o `MetalLB`, use o guia do MetalLB no K3s (*Capítulo 25, MetalLB no K3s (usando o modo de camada 2)*).

NOTA: Assegure que os endereços IP IPAddressPool do `ip-pool` não sobreponham os endereços IP já selecionados para o serviço `LoadBalancer`.

Crie um `IpAddressPool` separado para usar apenas para o serviço gerenciado.

```
# Export the VIP_SERVICE_IP on the local machine
# Replace with the actual IP
export VIP_SERVICE_IP=<ip>

cat <<-EOF | kubectl apply -f -
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: kubernetes-vip-ip-pool
  namespace: metallb-system
spec:
  addresses:
  - ${VIP_SERVICE_IP}/32
  serviceAllocation:
    priority: 100
    namespaces:
      - default
EOF
```

```
cat <<-EOF | kubectl apply -f -
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: ip-pool-l2-adv
  namespace: metallb-system
spec:
  ipAddressPools:
  - ip-pool
  - kubernetes-vip-ip-pool
EOF
```

26.5 Instalando o Endpoint Copier Operator

```
helm install \
```

```
endpoint-copier-operator oci://registry.suse.com/edge/charts/endpoint-copier-operator \
--namespace endpoint-copier-operator \
--create-namespace
```

O comando acima implanta o operador `endpoint-copier-operator` com duas réplicas: uma será a líder e a outra assumirá a função de líder se necessário.

Agora o serviço `kubernetes-vip`, que será reconciliado pelo operador, deve ser implantado, e um endpoint com as portas e o IP configurados será criado.

Para RKE2:

```
cat <<-EOF | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
  name: kubernetes-vip
  namespace: default
spec:
  ports:
  - name: rke2-api
    port: 9345
    protocol: TCP
    targetPort: 9345
  - name: k8s-api
    port: 6443
    protocol: TCP
    targetPort: 6443
  type: LoadBalancer
EOF
```

Para K3s:

```
cat <<-EOF | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
  name: kubernetes-vip
  namespace: default
spec:
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: https
    port: 6443
    protocol: TCP
```

```
    targetPort: 6443
  sessionAffinity: None
  type: LoadBalancer
EOF
```

Verifique se o serviço `kubernetes-vip` tem o endereço IP correto:

```
kubectl get service kubernetes-vip -n default \
 -o=jsonpath='{.status.loadBalancer.ingress[0].ip}'
```

Garanta que os recursos dos endpoints `kubernetes-vip` e `kubernetes` no namespace `default` apontem para os mesmos IPs.

```
kubectl get endpoints kubernetes kubernetes-vip
```

Se tudo estiver correto, a última coisa a se fazer é usar o `VIP_SERVICE_IP` no `Kubeconfig`.

```
sed -i '' "s/${NODE_IP}/${VIP_SERVICE_IP}/g" ~/.kube/config
```

A partir de agora, o `kubectl` sempre vai passar pelo serviço `kubernetes-vip`.

26.6 Adicionando nós do plano de controle

Para monitorar o processo inteiro, é possível abrir mais duas guias do terminal.

Primeiro terminal:

```
watch kubectl get nodes
```

Segundo terminal:

```
watch kubectl get endpoints
```

Agora execute os comandos a seguir no segundo e no terceiro nó.

Para RKE2:

```
# Export the VIP_SERVICE_IP in the VM
# Replace with the actual IP
export VIP_SERVICE_IP=<ip>

curl -sfL https://get.rke2.io | INSTALL_RKE2_TYPE="server" sh -
systemctl enable rke2-server.service
```

```
mkdir -p /etc/rancher/rke2/
cat <<EOF > /etc/rancher/rke2/config.yaml
server: https://${VIP_SERVICE_IP}:9345
token: ${RKE2_TOKEN}
EOF

systemctl start rke2-server.service
```

Para K3s:

```
# Export the VIP_SERVICE_IP in the VM
# Replace with the actual IP
export VIP_SERVICE_IP=<ip>
export INSTALL_K3S_SKIP_START=false

curl -sfL https://get.k3s.io | INSTALL_K3S_EXEC="server \
 --server https://${VIP_SERVICE_IP}:6443 --disable=servicelb \
 --write-kubeconfig-mode=644" K3S_TOKEN=foobar sh -
```

27 Implantações air-gapped com o Edge Image Builder

27.1 Introdução

Este guia mostra como implantar vários componentes do SUSE Edge totalmente air-gapped no SUSE Linux Micro 6.1 usando o Edge Image Builder (EIB) (*Capítulo 11, Edge Image Builder*). Dessa forma, você pode inicializar em uma imagem personalizada e pronta para uso (CRB) criada pelo EIB e ter os componentes especificados implantados em um cluster RKE2 ou K3s, sem conexão com a Internet nem qualquer etapa manual. Essa configuração é extremamente prática para clientes que desejam fazer bake prévio de todos os artefatos necessários para implantação na imagem do sistema operacional, assim eles ficam disponíveis logo após a inicialização.

Vamos abordar a instalação air-gapped do:

- *Capítulo 5, Rancher*
- *Capítulo 18, SUSE Security*
- *Capítulo 17, SUSE Storage*
- *Capítulo 21, Edge Virtualization*

Atenção

O EIB analisa e faz pré-download de todas as imagens referenciadas nos gráficos Helm e nos manifestos do Kubernetes fornecidos. No entanto, alguns deles podem tentar obter as imagens do contêiner e criar recursos do Kubernetes com base nas imagens em runtime. Nesses casos, precisamos especificar manualmente as imagens necessárias no arquivo de definição para configurar um ambiente completamente air-gapped.

27.2 Pré-requisitos

Se você está seguindo este guia, consideramos que já esteja familiarizado com o EIB (*Capítulo 11, Edge Image Builder*). Do contrário, consulte o Guia de Início Rápido (*Capítulo 3, Clusters independentes com o Edge Image Builder*) para entender melhor os conceitos apresentados na prática a seguir.

27.3 Configuração da rede libvirt

Nota

Para demonstrar a implantação air-gapped, este guia usa a rede air-gapped simulada `libvirt` e a configuração a seguir é adaptada a ela. Em suas próprias implantações, você pode precisar modificar a configuração `host1.local.yaml` que será introduzida na etapa seguinte.

Se você prefere usar a mesma configuração de rede `libvirt`, siga adiante. Do contrário, pule para a *Seção 27.4, "Configuração do diretório base"*.

Vamos criar uma configuração de rede isolada com o intervalo de endereços IP `192.168.100.2/24` para DHCP:

```
cat << EOF > isolatednetwork.xml
<network>
  <name>isolatednetwork</name>
  <bridge name='virbr1' stp='on' delay='0'/>
  <ip address='192.168.100.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.100.2' end='192.168.100.254'/>
    </dhcp>
  </ip>
</network>
EOF
```

Agora resta apenas criar e iniciar a rede:

```
virsh net-define isolatednetwork.xml
virsh net-start isolatednetwork
```

27.4 Configuração do diretório base

A configuração do diretório base é igual em todos os diversos componentes, portanto, vamos defini-la aqui.

Vamos primeiro criar os subdiretórios necessários:

```
export CONFIG_DIR=$HOME/config
mkdir -p $CONFIG_DIR/base-images
mkdir -p $CONFIG_DIR/network
```

```
mkdir -p $CONFIG_DIR/kubernetes/helm/values
```

Adicione qualquer imagem base que você queira usar ao diretório `base-images`. O foco deste guia é a autoinstalação da ISO disponível aqui (https://www.suse.com/download/sle-micro/).

Vamos copiar a imagem baixada:

```
cp SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso $CONFIG_DIR/base-images/
slemicro.iso
```

Nota

O EIB nunca modifica a entrada da imagem base.

Vamos criar um arquivo com a configuração de rede desejada:

```
cat << EOF > $CONFIG_DIR/network/host1.local.yaml
routes:
  config:
  - destination: 0.0.0.0/0
    metric: 100
    next-hop-address: 192.168.100.1
    next-hop-interface: eth0
    table-id: 254
  - destination: 192.168.100.0/24
    metric: 100
    next-hop-address:
    next-hop-interface: eth0
    table-id: 254
dns-resolver:
  config:
    server:
    - 192.168.100.1
    - 8.8.8.8
interfaces:
- name: eth0
  type: ethernet
  state: up
  mac-address: 34:8A:B1:4B:16:E7
  ipv4:
    address:
    - ip: 192.168.100.50
      prefix-length: 24
    dhcp: false
    enabled: true
  ipv6:
```

```
    enabled: false
EOF
```

Esta configuração garante que os seguintes elementos estejam presentes nos sistemas provisionados (usando o endereço MAC especificado):

- uma interface Ethernet com endereço IP estático
- roteamento
- DNS
- nome de host (`host1.local`)

A estrutura do arquivo resultante deve ter a seguinte aparência:

```
├── kubernetes/
│   └── helm/
│       └── values/
├── base-images/
│   └── slemicro.iso
└── network/
    └── host1.local.yaml
```

27.5 Arquivo de definição de base

O Edge Image Builder usa *arquivos de definição* para modificar as imagens do SUSE Linux Micro. Esses arquivos contêm a maioria das opções configuráveis. Muitas dessas opções se repetem nas diferentes seções de componentes, portanto, vamos listá-las e explicá-las aqui.

Dica

A lista completa das opções de personalização no arquivo de definição está disponível na documentação upstream (https://github.com/suse-edge/edge-image-builder/blob/release-1.1/docs/building-images.md#image-definition-file)

Vamos analisar os seguintes campos que estarão presentes em todos os arquivos de definição:

```
apiVersion: 1.2
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
```

```
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword: $6$jHugJNNd3HElGsUZ
$eodjVe4te5ps44SVcWshdfWizrP.xAyd71CVEXazBJ/.v799/WRCBXxfYmunlBO2yp1hm/zb4r8EmnrrNCF.P/
kubernetes:
  version: v1.32.4+rke2r1
embeddedArtifactRegistry:
  images:
    - ...
```

A seção `image` é obrigatória e especifica a imagem de entrada, sua arquitetura e tipo, além do nome da imagem de saída.

A seção `operatingSystem` é opcional e contém uma configuração para permitir o login nos sistemas provisionados com o nome de usuário/senha `root/eib`.

A seção `kubernetes` é opcional e define o tipo e a versão do Kubernetes. Vamos usar a distribuição RKE2. Em vez disso, se você preferir o K3s, use `kubernetes.version: v1.32.4+k3s1`. Exceto se claramente configurado no campo `kubernetes.nodes`, todos os clusters que inicializamos neste guia são de nó único.

A seção `embeddedArtifactRegistry` incluirá todas as imagens que apenas são referenciadas e extraídas em runtime para o componente específico.

27.6 Instalação do Rancher

Nota

A exibição da implantação do Rancher (*Capítulo 5, Rancher*) será muito reduzida para fins de demonstração. Em sua implantação real, podem ser necessários artefatos adicionais, dependendo da sua configuração.

Os ativo da versão do Rancher 2.11.2 (https://github.com/rancher/rancher/releases/tag/v2.11.2) contêm um arquivo `rancher-images.txt` que lista todas as imagens necessárias para uma instalação air-gapped.

Há mais de 600 imagens do contêiner no total, o que significa que a imagem CRB resultante teria cerca de 30 GB. No caso da nossa instalação do Rancher, vamos reduzir a lista para a menor configuração de trabalho. A partir disso, você pode readicionar qualquer imagem que possa precisar em sua implantação.

Vamos criar o arquivo de definição e incluir a lista de imagens reduzida:

```
apiVersion: 1.2
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword: $6$jHugJNNd3HElGsUZ
$eodjVe4te5ps44SVcWshdfWizrP.xAyd71CVEXazBJ/.v799/WRCBXxfYmunlBO2yp1hm/zb4r8EmnrrNCF.P/
kubernetes:
  version: v1.32.4+rke2r1
  manifests:
    urls:
    - https://github.com/cert-manager/cert-manager/releases/download/v1.15.3/cert-
manager.crds.yaml
  helm:
    charts:
      - name: rancher
        version: 2.11.2
        repositoryName: rancher-prime
        valuesFile: rancher-values.yaml
        targetNamespace: cattle-system
        createNamespace: true
        installationNamespace: kube-system
      - name: cert-manager
        installationNamespace: kube-system
        createNamespace: true
        repositoryName: jetstack
        targetNamespace: cert-manager
        version: 1.15.3
    repositories:
      - name: jetstack
        url: https://charts.jetstack.io
      - name: rancher-prime
        url: https://charts.rancher.com/server-charts/prime
embeddedArtifactRegistry:
  images:
    - name: registry.rancher.com/rancher/backup-restore-operator:v7.0.1
    - name: registry.rancher.com/rancher/calico-cni:v3.29.0-rancher1
    - name: registry.rancher.com/rancher/cis-operator:v1.4.0
    - name: registry.rancher.com/rancher/flannel-cni:v1.4.1-rancher1
    - name: registry.rancher.com/rancher/fleet-agent:v0.12.2
    - name: registry.rancher.com/rancher/fleet:v0.12.2
```

```
    - name: registry.rancher.com/rancher/hardened-addon-resizer:1.8.22-build20250110
    - name: registry.rancher.com/rancher/hardened-calico:v3.29.2-build20250306
    - name: registry.rancher.com/rancher/hardened-cluster-autoscaler:v1.9.0-build20241126
    - name: registry.rancher.com/rancher/hardened-cni-plugins:v1.6.2-build20250306
    - name: registry.rancher.com/rancher/hardened-coredns:v1.12.0-build20241126
    - name: registry.rancher.com/rancher/hardened-dns-node-cache:1.24.0-build20241211
    - name: registry.rancher.com/rancher/hardened-etcd:v3.5.19-k3s1-build20250306
    - name: registry.rancher.com/rancher/hardened-flannel:v0.26.5-build20250306
    - name: registry.rancher.com/rancher/hardened-k8s-metrics-server:v0.7.2-build20250110
    - name: registry.rancher.com/rancher/hardened-kubernetes:v1.32.3-rke2r1-build20250312
    - name: registry.rancher.com/rancher/hardened-multus-cni:v4.1.4-build20250108
    - name: registry.rancher.com/rancher/hardened-whereabouts:v0.8.0-build20250131
    - name: registry.rancher.com/rancher/k3s-upgrade:v1.32.3-k3s1
    - name: registry.rancher.com/rancher/klipper-helm:v0.9.4-build20250113
    - name: registry.rancher.com/rancher/klipper-lb:v0.4.13
    - name: registry.rancher.com/rancher/kube-api-auth:v0.2.4
    - name: registry.rancher.com/rancher/kubectl:v1.32.2
    - name: registry.rancher.com/rancher/kuberlr-kubectl:v4.0.2
    - name: registry.rancher.com/rancher/local-path-provisioner:v0.0.31
    - name: registry.rancher.com/rancher/machine:v0.15.0-rancher125
    - name: registry.rancher.com/rancher/mirrored-cluster-api-controller:v1.9.5
    - name: registry.rancher.com/rancher/nginx-ingress-controller:v1.12.1-hardened1
    - name: registry.rancher.com/rancher/prom-prometheus:v2.55.1
    - name: registry.rancher.com/rancher/prometheus-federator:v3.0.1
    - name: registry.rancher.com/rancher/pushprox-client:v0.1.4-rancher2-client
    - name: registry.rancher.com/rancher/pushprox-proxy:v0.1.4-rancher2-proxy
    - name: registry.rancher.com/rancher/rancher-agent:v2.11.1
    - name: registry.rancher.com/rancher/rancher-csp-adapter:v6.0.0
    - name: registry.rancher.com/rancher/rancher-webhook:v0.7.1
    - name: registry.rancher.com/rancher/rancher:v2.11.1
    - name: registry.rancher.com/rancher/remotedialer-proxy:v0.4.4
    - name: registry.rancher.com/rancher/rke-tools:v0.1.111
    - name: registry.rancher.com/rancher/rke2-cloud-provider:v1.32.0-
rc3.0.20241220224140-68fbd1a6b543-build20250101
    - name: registry.rancher.com/rancher/rke2-runtime:v1.32.3-rke2r1
    - name: registry.rancher.com/rancher/rke2-upgrade:v1.32.3-rke2r1
    - name: registry.rancher.com/rancher/security-scan:v0.6.0
    - name: registry.rancher.com/rancher/shell:v0.4.0
    - name: registry.rancher.com/rancher/system-agent-installer-k3s:v1.32.3-k3s1
    - name: registry.rancher.com/rancher/system-agent-installer-rke2:v1.32.3-rke2r1
    - name: registry.rancher.com/rancher/system-agent:v0.3.12-suc
    - name: registry.rancher.com/rancher/system-upgrade-controller:v0.15.2
    - name: registry.rancher.com/rancher/ui-plugin-catalog:4.0.1
    - name: registry.rancher.com/rancher/kubectl:v1.20.2
    - name: registry.rancher.com/rancher/shell:v0.1.24
    - name: registry.rancher.com/rancher/mirrored-ingress-nginx-kube-webhook-
certgen:v1.5.0
```

```
    - name: registry.rancher.com/rancher/mirrored-ingress-nginx-kube-webhook-
certgen:v1.5.2
```

Em comparação com a lista completa de mais de 600 imagens do contêiner, essa versão reduzida contém apenas cerca de 60, o que faz com que a nova imagem CRB tenha somente 7 GB.

Vamos criar também um arquivo de valores do Helm para o Rancher:

```
cat << EOF > $CONFIG_DIR/kubernetes/helm/values/rancher-values.yaml
hostname: 192.168.100.50.sslip.io
replicas: 1
bootstrapPassword: "adminadminadmin"
systemDefaultRegistry: registry.rancher.com
useBundledSystemChart: true
EOF
```

Atenção

A definição de `systemDefaultRegistry` como `registry.rancher.com` permite que o Rancher procure automaticamente as imagens no registro de artefatos incorporado que foi iniciado na imagem CRB no momento da inicialização. A omissão desse campo pode impedir que as imagens do contêiner sejam encontradas no nó.

Vamos criar a imagem:

```
podman run --rm -it --privileged -v $CONFIG_DIR:/eib \
registry.suse.com/edge/3.3/edge-image-builder:1.2.1 \
build --definition-file eib-iso-definition.yaml
```

A saída deve ter uma aparência semelhante a esta:

```
Downloading file: dl-manifest-1.yaml 100% |
████████████████████████████████████████████████████████████████████| (583/583
 kB, 12 MB/s)
Pulling selected Helm charts... 100% |
████████████████████████████████████████████████████████████████████████████|
 (2/2, 3 it/s)
Generating image customization components...
Identifier ................... [SUCCESS]
Custom Files ................. [SKIPPED]
Time ......................... [SKIPPED]
Network ...................... [SUCCESS]
Groups ....................... [SKIPPED]
Users ........................ [SUCCESS]
Proxy ........................ [SKIPPED]
Rpm .......................... [SKIPPED]
```

```
Os Files ..................... [SKIPPED]
Systemd ...................... [SKIPPED]
Fips ......................... [SKIPPED]
Elemental .................... [SKIPPED]
Suma ......................... [SKIPPED]
Populating Embedded Artifact Registry... 100% |
████████████████████████████████████████████████████████████████████████████| (56/56, 8
 it/min)
Embedded Artifact Registry ... [SUCCESS]
Keymap ....................... [SUCCESS]
Configuring Kubernetes component...
The Kubernetes CNI is not explicitly set, defaulting to 'cilium'.
Downloading file: rke2_installer.sh
Downloading file: rke2-images-core.linux-amd64.tar.zst 100% |
████████████████████████████████████████████████████████| (644/644 MB, 29 MB/s)
Downloading file: rke2-images-cilium.linux-amd64.tar.zst 100% |
████████████████████████████████████████████████████| (400/400 MB, 29 MB/s)
Downloading file: rke2.linux-amd64.tar.gz 100% |
████████████████████████████████████████████████████████████████████████| (36/36 MB,
 30 MB/s)
Downloading file: sha256sum-amd64.txt 100% |
████████████████████████████████████████████████████████████████████████████| (4.3/4.3
 kB, 29 MB/s)
Kubernetes ................... [SUCCESS]
Certificates ................. [SKIPPED]
Cleanup ...................... [SKIPPED]
Building ISO image...
Kernel Params ................ [SKIPPED]
Build complete, the image can be found at: eib-image.iso
```

Após o provisionamento do nó usando a imagem criada, poderemos verificar a instalação do Rancher:

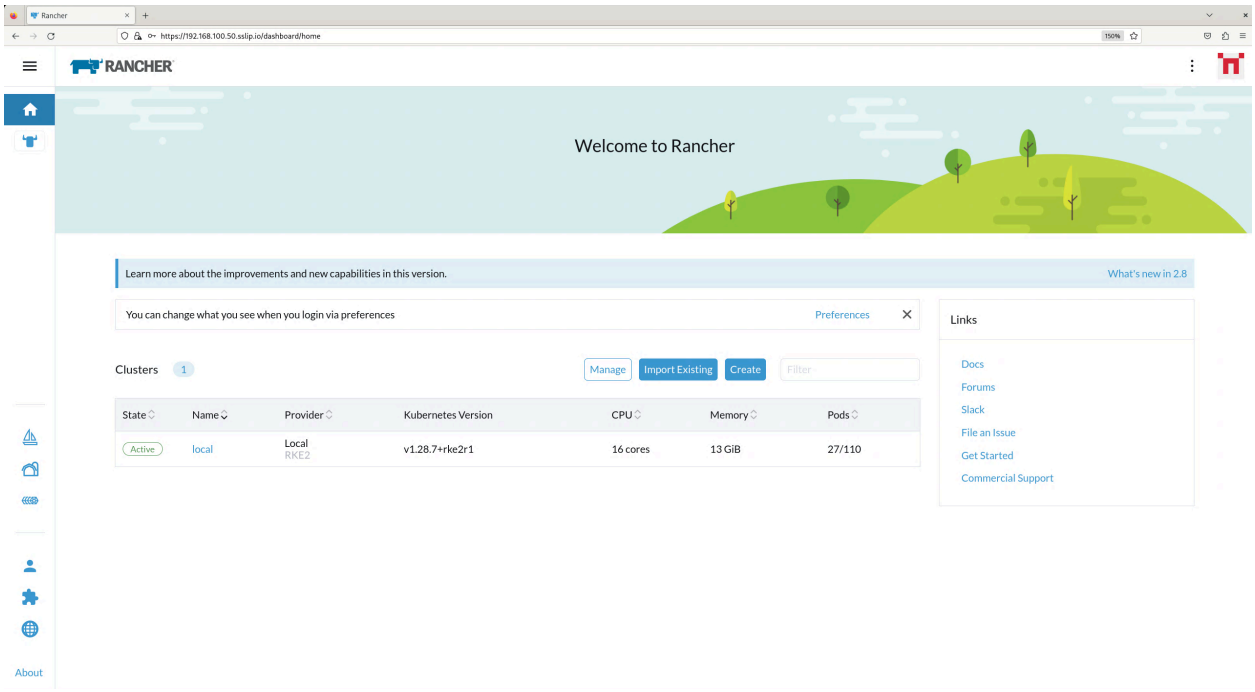
```
/var/lib/rancher/rke2/bin/kubectl get all -n cattle-system --kubeconfig /etc/rancher/
rke2/rke2.yaml
```

A saída deve mostrar que tudo foi implantado com sucesso, com uma aparência similar a esta:

```
NAME                                    READY   STATUS      RESTARTS   AGE
pod/helm-operation-6l6ld                0/2     Completed   0          107s
pod/helm-operation-8tk2v                0/2     Completed   0          2m2s
pod/helm-operation-blnrr                0/2     Completed   0          2m49s
pod/helm-operation-hdcmt                0/2     Completed   0          3m19s
pod/helm-operation-m74c7                0/2     Completed   0          97s
pod/helm-operation-qzzr4                0/2     Completed   0          2m30s
pod/helm-operation-s9jh5                0/2     Completed   0          3m
pod/helm-operation-tq7ts                0/2     Completed   0          2m41s
pod/rancher-99d599967-ftjkk             1/1     Running     0          4m15s
```

```
pod/rancher-webhook-79798674c5-6w28t              1/1     Running     0          2m27s
pod/system-upgrade-controller-56696956b-trq5c     1/1     Running     0          104s

NAME                          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)           AGE
service/rancher               ClusterIP   10.43.255.80   <none>        80/TCP,443/TCP    4m15s
service/rancher-webhook       ClusterIP   10.43.7.238    <none>        443/TCP           2m27s

NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/rancher                    1/1     1            1           4m15s
deployment.apps/rancher-webhook            1/1     1            1           2m27s
deployment.apps/system-upgrade-controller  1/1     1            1           104s

NAME                                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/rancher-99d599967                    1         1         1       4m15s
replicaset.apps/rancher-webhook-79798674c5           1         1         1       2m27s
replicaset.apps/system-upgrade-controller-56696956b  1         1         1       104s
```

Vamos acessar `https://192.168.100.50.sslip.io` e fazer login com a senha `adminadminadmin` que já definimos, o que nos leva ao dashboard do Rancher:



27.7 Instalação do SUSE Security

Ao contrário da instalação do Rancher, a do SUSE Security não requer nenhum processamento adicional no EIB. O EIB isola automaticamente as imagens necessárias ao seu componente subjacente NeuVector.

Vamos criar o arquivo de definição:

```
apiVersion: 1.2
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword: $6$jHugJNNd3HElGsUZ
$eodjVe4te5ps44SVcWshdfWizrP.xAyd71CVEXazBJ/.v799/WRCBXxfYmunlBO2yp1hm/zb4r8EmnrrNCF.P/
kubernetes:
  version: v1.32.4+rke2r1
  helm:
    charts:
      - name: neuvector-crd
        version: 106.0.1+up2.8.6
        repositoryName: rancher-charts
        targetNamespace: neuvector
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: neuvector-values.yaml
      - name: neuvector
        version: 106.0.1+up2.8.6
        repositoryName: rancher-charts
        targetNamespace: neuvector
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: neuvector-values.yaml
    repositories:
      - name: rancher-charts
        url: https://charts.rancher.io/
```

Vamos criar também um arquivo de valores do Helm para o NeuVector:

```
cat << EOF > $CONFIG_DIR/kubernetes/helm/values/neuvector-values.yaml
controller:
  replicas: 1
manager:
  enabled: false
cve:
  scanner:
    enabled: false
    replicas: 1
k3s:
```

```
  enabled: true
crdwebhook:
  enabled: false
EOF
```

Vamos criar a imagem:

```
podman run --rm -it --privileged -v $CONFIG_DIR:/eib \
registry.suse.com/edge/3.3/edge-image-builder:1.2.1 \
build --definition-file eib-iso-definition.yaml
```

A saída deve ter uma aparência semelhante a esta:

```
Pulling selected Helm charts... 100% |
████████████████████████████████████████████████████████████████████████████████|
 (2/2, 4 it/s)
Generating image customization components...
Identifier ................... [SUCCESS]
Custom Files ................. [SKIPPED]
Time ......................... [SKIPPED]
Network ...................... [SUCCESS]
Groups ....................... [SKIPPED]
Users ........................ [SUCCESS]
Proxy ........................ [SKIPPED]
Rpm .......................... [SKIPPED]
Os Files ..................... [SKIPPED]
Systemd ...................... [SKIPPED]
Fips ......................... [SKIPPED]
Elemental .................... [SKIPPED]
Suma ......................... [SKIPPED]
Populating Embedded Artifact Registry... 100% |
████████████████████████████████████████████████████████████████| (5/5, 13
 it/min)
Embedded Artifact Registry ... [SUCCESS]
Keymap ....................... [SUCCESS]
Configuring Kubernetes component...
The Kubernetes CNI is not explicitly set, defaulting to 'cilium'.
Downloading file: rke2_installer.sh
Kubernetes ................... [SUCCESS]
Certificates ................. [SKIPPED]
Cleanup ...................... [SKIPPED]
Building ISO image...
Kernel Params ................ [SKIPPED]
Build complete, the image can be found at: eib-image.iso
```

Após o provisionamento do nó usando a imagem criada, poderemos verificar a instalação do SUSE Security:

```
/var/lib/rancher/rke2/bin/kubectl get all -n neuvector --kubeconfig /etc/rancher/rke2/
rke2.yaml
```

A saída deve mostrar que tudo foi implantado com sucesso, com uma aparência similar a esta:

```
NAME                                          READY   STATUS      RESTARTS   AGE
pod/neuvector-cert-upgrader-job-bxbnz         0/1     Completed   0          3m39s
pod/neuvector-controller-pod-7d854bfdc7-nhxjf   1/1     Running     0          3m44s
pod/neuvector-enforcer-pod-ct8jm              1/1     Running     0          3m44s

NAME                                     TYPE        CLUSTER-IP      EXTERNAL-IP
 PORT(S)                          AGE
service/neuvector-svc-admission-webhook   ClusterIP   10.43.234.241   <none>        443/
TCP                          3m44s
service/neuvector-svc-controller          ClusterIP   None            <none>
 18300/TCP,18301/TCP,18301/UDP   3m44s
service/neuvector-svc-crd-webhook         ClusterIP   10.43.50.190    <none>        443/
TCP                          3m44s

NAME                                   DESIRED   CURRENT   READY   UP-TO-DATE
 AVAILABLE   NODE SELECTOR   AGE
daemonset.apps/neuvector-enforcer-pod   1         1         1       1            1
    <none>          3m44s

NAME                                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/neuvector-controller-pod   1/1     1            1           3m44s

NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/neuvector-controller-pod-7d854bfdc7   1         1         1       3m44s

NAME                                      SCHEDULE    TIMEZONE   SUSPEND   ACTIVE
 LAST SCHEDULE   AGE
cronjob.batch/neuvector-cert-upgrader-pod   0 0 1 1 *   <none>     True      0
 <none>          3m44s
cronjob.batch/neuvector-updater-pod         0 0 * * *   <none>     False     0
 <none>          3m44s

NAME                                   STATUS     COMPLETIONS   DURATION   AGE
job.batch/neuvector-cert-upgrader-job   Complete   1/1           7s         3m39s
```

27.8 Instalação do SUSE Storage

A documentação oficial (https://longhorn.io/docs/1.8.1/deploy/install/airgap/) do Longhorn contém o arquivo `longhorn-images.txt` que lista todas as imagens necessárias a uma instalação air-gapped. Vamos incluir as contrapartes espelhadas do registro do contêiner do Rancher em nosso arquivo de definição. Vamos criá-lo:

```
apiVersion: 1.2
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword: $6$jHugJNNd3HElGsUZ
$eodjVe4te5ps44SVcWshdfWizrP.xAyd71CVEXazBJ/.v799/WRCBXxfYmunlBO2yp1hm/zb4r8EmnrrNCF.P/
  packages:
    sccRegistrationCode: [reg-code]
    packageList:
      - open-iscsi
kubernetes:
  version: v1.32.4+rke2r1
  helm:
    charts:
      - name: longhorn
        repositoryName: longhorn
        targetNamespace: longhorn-system
        createNamespace: true
        version: 106.2.0+up1.8.1
      - name: longhorn-crd
        repositoryName: longhorn
        targetNamespace: longhorn-system
        createNamespace: true
        installationNamespace: kube-system
        version: 106.2.0+up1.8.1
    repositories:
      - name: longhorn
        url: https://charts.rancher.io
embeddedArtifactRegistry:
  images:
    - name: registry.suse.com/rancher/mirrored-longhornio-csi-attacher:v4.8.1
    - name: registry.suse.com/rancher/mirrored-longhornio-csi-provisioner:v5.2.0
    - name: registry.suse.com/rancher/mirrored-longhornio-csi-resizer:v1.13.2
    - name: registry.suse.com/rancher/mirrored-longhornio-csi-snapshotter:v8.2.0
```

```
    - name: registry.suse.com/rancher/mirrored-longhornio-csi-node-driver-
registrar:v2.13.0
    - name: registry.suse.com/rancher/mirrored-longhornio-livenessprobe:v2.15.0
    - name: registry.suse.com/rancher/mirrored-longhornio-backing-image-manager:v1.8.1
    - name: registry.suse.com/rancher/mirrored-longhornio-longhorn-engine:v1.8.1
    - name: registry.suse.com/rancher/mirrored-longhornio-longhorn-instance-
manager:v1.8.1
    - name: registry.suse.com/rancher/mirrored-longhornio-longhorn-manager:v1.8.1
    - name: registry.suse.com/rancher/mirrored-longhornio-longhorn-share-manager:v1.8.1
    - name: registry.suse.com/rancher/mirrored-longhornio-longhorn-ui:v1.8.1
    - name: registry.suse.com/rancher/mirrored-longhornio-support-bundle-kit:v0.0.52
    - name: registry.suse.com/rancher/mirrored-longhornio-longhorn-cli:v1.8.1
```

Nota

Veja que o arquivo de definição lista o pacote `open-iscsi`. Ele é necessário porque o Longhorn precisa que o daemon `iscsiadm` seja executado em nós diferentes para fornecer volumes persistentes ao Kubernetes.

Vamos criar a imagem:

```
podman run --rm -it --privileged -v $CONFIG_DIR:/eib \
registry.suse.com/edge/3.3/edge-image-builder:1.2.1 \
build --definition-file eib-iso-definition.yaml
```

A saída deve ter uma aparência semelhante a esta:

```
Setting up Podman API listener...
Pulling selected Helm charts... 100% |
 (2/2, 3 it/s)
Generating image customization components...
Identifier ................... [SUCCESS]
Custom Files ................. [SKIPPED]
Time ......................... [SKIPPED]
Network ...................... [SUCCESS]
Groups ....................... [SKIPPED]
Users ........................ [SUCCESS]
Proxy ........................ [SKIPPED]
Resolving package dependencies...
Rpm .......................... [SUCCESS]
Os Files ..................... [SKIPPED]
Systemd ...................... [SKIPPED]
Fips ......................... [SKIPPED]
Elemental .................... [SKIPPED]
Suma ......................... [SKIPPED]
```

```
Populating Embedded Artifact Registry... 100% |
 (15/15, 20956 it/s)
Embedded Artifact Registry ... [SUCCESS]
Keymap ..................... [SUCCESS]
Configuring Kubernetes component...
The Kubernetes CNI is not explicitly set, defaulting to 'cilium'.
Downloading file: rke2_installer.sh
Downloading file: rke2-images-core.linux-amd64.tar.zst 100% (782/782 MB, 108 MB/s)
Downloading file: rke2-images-cilium.linux-amd64.tar.zst 100% (367/367 MB, 104 MB/s)
Downloading file: rke2.linux-amd64.tar.gz 100% (34/34 MB, 108 MB/s)
Downloading file: sha256sum-amd64.txt 100% (3.9/3.9 kB, 7.5 MB/s)
Kubernetes ................. [SUCCESS]
Certificates ............... [SKIPPED]
Cleanup .................... [SKIPPED]
Building ISO image...
Kernel Params .............. [SKIPPED]
Build complete, the image can be found at: eib-image.iso
```

Após o provisionamento do nó usando a imagem criada, poderemos verificar a instalação do Longhorn:

```
/var/lib/rancher/rke2/bin/kubectl get all -n longhorn-system --kubeconfig /etc/rancher/
rke2/rke2.yaml
```

A saída deve mostrar que tudo foi implantado com sucesso, com uma aparência similar a esta:

```
NAME                                                READY   STATUS    RESTARTS   AGE
pod/csi-attacher-787fd9c6c8-sf42d                   1/1     Running   0
 2m28s
pod/csi-attacher-787fd9c6c8-tb82p                   1/1     Running   0
 2m28s
pod/csi-attacher-787fd9c6c8-zhc6s                   1/1     Running   0
 2m28s
pod/csi-provisioner-74486b95c6-b2v9s                1/1     Running   0
 2m28s
pod/csi-provisioner-74486b95c6-hwllt                1/1     Running   0
 2m28s
pod/csi-provisioner-74486b95c6-mlrpk                1/1     Running   0
 2m28s
pod/csi-resizer-859d4557fd-t54zk                    1/1     Running   0
 2m28s
pod/csi-resizer-859d4557fd-vdt5d                    1/1     Running   0
 2m28s
pod/csi-resizer-859d4557fd-x9kh4                    1/1     Running   0
 2m28s
pod/csi-snapshotter-6f69c6c8cc-r62gr                1/1     Running   0
 2m28s
```

```
pod/csi-snapshotter-6f69c6c8cc-vrwjn                    1/1     Running   0
 2m28s
pod/csi-snapshotter-6f69c6c8cc-z65nb                    1/1     Running   0
 2m28s
pod/engine-image-ei-4623b511-9vhkb                      1/1     Running   0
 3m13s
pod/instance-manager-6f95fd57d4a4cd0459e469d75a300552   1/1     Running   0
 2m43s
pod/longhorn-csi-plugin-gx98x                           3/3     Running   0
 2m28s
pod/longhorn-driver-deployer-55f9c88499-fbm6q           1/1     Running   0
 3m28s
pod/longhorn-manager-dpdp7                              2/2     Running   0
 3m28s
pod/longhorn-ui-59c85fcf94-gg5hq                        1/1     Running   0
 3m28s
pod/longhorn-ui-59c85fcf94-s49jc                        1/1     Running   0
 3m28s

NAME                                TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)
 AGE
service/longhorn-admission-webhook    ClusterIP   10.43.77.89    <none>        9502/TCP
 3m28s
service/longhorn-backend              ClusterIP   10.43.56.17    <none>        9500/TCP
 3m28s
service/longhorn-conversion-webhook   ClusterIP   10.43.54.73    <none>        9501/TCP
 3m28s
service/longhorn-frontend             ClusterIP   10.43.22.82    <none>        80/TCP
 3m28s
service/longhorn-recovery-backend     ClusterIP   10.43.45.143   <none>        9503/TCP
 3m28s

NAME                                      DESIRED   CURRENT   READY   UP-TO-DATE
 AVAILABLE   NODE SELECTOR   AGE
daemonset.apps/engine-image-ei-4623b511   1         1         1       1            1
      <none>          3m13s
daemonset.apps/longhorn-csi-plugin        1         1         1       1            1
      <none>          2m28s
daemonset.apps/longhorn-manager           1         1         1       1            1
      <none>          3m28s

NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/csi-attacher               3/3     3            3           2m28s
deployment.apps/csi-provisioner            3/3     3            3           2m28s
deployment.apps/csi-resizer                3/3     3            3           2m28s
deployment.apps/csi-snapshotter            3/3     3            3           2m28s
deployment.apps/longhorn-driver-deployer   1/1     1            1           3m28s
```

```
deployment.apps/longhorn-ui                2/2     2            2           3m28s

NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/csi-attacher-787fd9c6c8             3         3         3       2m28s
replicaset.apps/csi-provisioner-74486b95c6          3         3         3       2m28s
replicaset.apps/csi-resizer-859d4557fd              3         3         3       2m28s
replicaset.apps/csi-snapshotter-6f69c6c8cc          3         3         3       2m28s
replicaset.apps/longhorn-driver-deployer-55f9c88499 1         1         1       3m28s
replicaset.apps/longhorn-ui-59c85fcf94              2         2         2       3m28s
```

27.9 Instalação do KubeVirt e CDI

Os gráficos Helm para KubeVirt e CDI instalam apenas os respectivos operadores. Cabe aos operadores implantar o restante dos sistemas, ou seja, precisamos incluir todas as imagens do contêiner necessárias em nosso arquivo de definição. Vamos criá-lo:

```
apiVersion: 1.2
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword: $6$jHugJNNd3HElGsUZ
$eodjVe4te5ps44SVcWshdfWizrP.xAyd71CVEXazBJ/.v799/WRCBXxfYmunlBO2yp1hm/zb4r8EmnrrNCF.P/
kubernetes:
  version: v1.32.4+rke2r1
  helm:
    charts:
      - name: kubevirt
        repositoryName: suse-edge
        version: 303.0.0+up0.5.0
        targetNamespace: kubevirt-system
        createNamespace: true
        installationNamespace: kube-system
      - name: cdi
        repositoryName: suse-edge
        version: 303.0.0+up0.5.0
        targetNamespace: cdi-system
        createNamespace: true
        installationNamespace: kube-system
    repositories:
```

```
      - name: suse-edge
        url: oci://registry.suse.com/edge/charts
embeddedArtifactRegistry:
  images:
    - name: registry.suse.com/suse/sles/15.6/cdi-uploadproxy:1.60.1-150600.3.9.1
    - name: registry.suse.com/suse/sles/15.6/cdi-uploadserver:1.60.1-150600.3.9.1
    - name: registry.suse.com/suse/sles/15.6/cdi-apiserver:1.60.1-150600.3.9.1
    - name: registry.suse.com/suse/sles/15.6/cdi-controller:1.60.1-150600.3.9.1
    - name: registry.suse.com/suse/sles/15.6/cdi-importer:1.60.1-150600.3.9.1
    - name: registry.suse.com/suse/sles/15.6/cdi-cloner:1.60.1-150600.3.9.1
    - name: registry.suse.com/suse/sles/15.6/virt-api:1.3.1-150600.5.9.1
    - name: registry.suse.com/suse/sles/15.6/virt-controller:1.3.1-150600.5.9.1
    - name: registry.suse.com/suse/sles/15.6/virt-launcher:1.3.1-150600.5.9.1
    - name: registry.suse.com/suse/sles/15.6/virt-handler:1.3.1-150600.5.9.1
    - name: registry.suse.com/suse/sles/15.6/virt-exportproxy:1.3.1-150600.5.9.1
    - name: registry.suse.com/suse/sles/15.6/virt-exportserver:1.3.1-150600.5.9.1
```

Vamos criar a imagem:

```
podman run --rm -it --privileged -v $CONFIG_DIR:/eib \
registry.suse.com/edge/3.3/edge-image-builder:1.2.1 \
build --definition-file eib-iso-definition.yaml
```

A saída deve ter uma aparência semelhante a esta:

```
Pulling selected Helm charts... 100% |
 (2/2, 48 it/min)
Generating image customization components...
Identifier ................... [SUCCESS]
Custom Files ................. [SKIPPED]
Time ......................... [SKIPPED]
Network ...................... [SUCCESS]
Groups ....................... [SKIPPED]
Users ........................ [SUCCESS]
Proxy ........................ [SKIPPED]
Rpm .......................... [SKIPPED]
Os Files ..................... [SKIPPED]
Systemd ...................... [SKIPPED]
Fips ......................... [SKIPPED]
Elemental .................... [SKIPPED]
Suma ......................... [SKIPPED]
Populating Embedded Artifact Registry... 100% |
 (15/15, 4 it/min)
Embedded Artifact Registry ... [SUCCESS]
Keymap ....................... [SUCCESS]
```

```
Configuring Kubernetes component...
The Kubernetes CNI is not explicitly set, defaulting to 'cilium'.
Downloading file: rke2_installer.sh
Kubernetes ................... [SUCCESS]
Certificates ................. [SKIPPED]
Cleanup ...................... [SKIPPED]
Building ISO image...
Kernel Params ................ [SKIPPED]
Build complete, the image can be found at: eib-image.iso
```

Após o provisionamento do nó usando a imagem criada, poderemos verificar a instalação do KubeVirt e do CDI.

Verifique o KubeVirt:

```
/var/lib/rancher/rke2/bin/kubectl get all -n kubevirt-system --kubeconfig /etc/rancher/
rke2/rke2.yaml
```

A saída deve mostrar que tudo foi implantado com sucesso, com uma aparência similar a esta:

```
NAME                                  READY   STATUS    RESTARTS   AGE
pod/virt-api-59cb997648-mmt67         1/1     Running   0          2m34s
pod/virt-controller-69786b785-7cc96   1/1     Running   0          2m8s
pod/virt-controller-69786b785-wq2dz   1/1     Running   0          2m8s
pod/virt-handler-2l4dm                1/1     Running   0          2m8s
pod/virt-operator-7c444cff46-nps4l    1/1     Running   0          3m1s
pod/virt-operator-7c444cff46-r25xq    1/1     Running   0          3m1s

NAME                                  TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
 AGE
service/kubevirt-operator-webhook     ClusterIP   10.43.167.109   <none>        443/TCP
 2m36s
service/kubevirt-prometheus-metrics   ClusterIP   None            <none>        443/TCP
 2m36s
service/virt-api                      ClusterIP   10.43.18.202    <none>        443/TCP
 2m36s
service/virt-exportproxy              ClusterIP   10.43.142.188   <none>        443/TCP
 2m36s

NAME                           DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
 SELECTOR            AGE
daemonset.apps/virt-handler   1         1         1       1            1
 kubernetes.io/os=linux   2m8s

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/virt-api          1/1     1            1           2m34s
deployment.apps/virt-controller   2/2     2            2           2m8s
```

```
deployment.apps/virt-operator     2/2     2            2           3m1s

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/virt-api-59cb997648        1         1         1       2m34s
replicaset.apps/virt-controller-69786b785  2         2         2       2m8s
replicaset.apps/virt-operator-7c444cff46   2         2         2       3m1s

NAME                             AGE    PHASE
kubevirt.kubevirt.io/kubevirt    3m1s   Deployed
```

Verifique o CDI:

```
/var/lib/rancher/rke2/bin/kubectl get all -n cdi-system --kubeconfig /etc/rancher/rke2/
rke2.yaml
```

A saída deve mostrar que tudo foi implantado com sucesso, com uma aparência similar a esta:

```
NAME                                    READY   STATUS    RESTARTS   AGE
pod/cdi-apiserver-5598c9bf47-pqfxw      1/1     Running   0          3m44s
pod/cdi-deployment-7cbc5db7f8-g46z7     1/1     Running   0          3m44s
pod/cdi-operator-777c865745-2qcnj       1/1     Running   0          3m48s
pod/cdi-uploadproxy-646f4cd7f7-fzkv7    1/1     Running   0          3m44s

NAME                              TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
service/cdi-api                   ClusterIP   10.43.2.224    <none>        443/TCP
 3m44s
service/cdi-prometheus-metrics    ClusterIP   10.43.237.13   <none>        8080/TCP
 3m44s
service/cdi-uploadproxy           ClusterIP   10.43.114.91   <none>        443/TCP
 3m44s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/cdi-apiserver       1/1     1            1           3m44s
deployment.apps/cdi-deployment      1/1     1            1           3m44s
deployment.apps/cdi-operator        1/1     1            1           3m48s
deployment.apps/cdi-uploadproxy     1/1     1            1           3m44s

NAME                                         DESIRED   CURRENT   READY   AGE
replicaset.apps/cdi-apiserver-5598c9bf47     1         1         1       3m44s
replicaset.apps/cdi-deployment-7cbc5db7f8    1         1         1       3m44s
replicaset.apps/cdi-operator-777c865745      1         1         1       3m48s
replicaset.apps/cdi-uploadproxy-646f4cd7f7   1         1         1       3m44s
```

27.10 Solução de problemas

Se você tiver problemas ao criar as imagens ou se quiser testar e depurar o processo em mais detalhes, consulte a documentação upstream (https://github.com/suse-edge/edge-image-builder/tree/release-1.1/docs).

28 Criando imagens atualizadas do SUSE Linux Micro com o Kiwi

Esta seção explica como gerar imagens atualizadas do SUSE Linux Micro para usar com o Edge Image Builder, usando Cluster API (CAPI) + Metal^3, ou como gravar a imagem do disco diretamente em um dispositivo em blocos. Esse processo é útil quando há necessidade de incluir os patches mais recentes nas imagens de inicialização do sistema inicial (para minimizar a transferência de patches após a instalação) ou quando a CAPI é usada, em que é preferível reinstalar o sistema operacional com uma nova imagem em vez de fazer upgrade dos hosts no local.

Esse processo usa o Kiwi (https://osinside.github.io/kiwi/) para a criação da imagem. O SUSE Edge vem com uma versão conteinerizada que simplifica o processo geral com um utilitário auxiliar integrado, o que permite especificar o **perfil** de destino obrigatório. O perfil define o tipo de imagem de saída necessário, sendo os mais comuns relacionados a seguir:

- "**Base**": uma imagem de disco do SUSE Linux Micro com um conjunto de pacotes reduzido (inclui o podman).
- "**Base-SelfInstall**": uma imagem SelfInstall conforme a imagem "Base" acima.
- "**Base-RT**": igual à imagem "Base" acima, mas usa o kernel em tempo real (rt) no lugar.
- "**Base-RT-SelfInstall**": uma imagem SelfInstall conforme a "Base-RT" acima.
- "**Default**": uma imagem de disco do SUSE Linux Micro conforme a "Base" acima, mas com algumas outras ferramentas, como pilha de virtualização, Cockpit e salt-minion.
- "**Default-SelfInstall**": uma imagem SelfInstall conforme a imagem "Default" acima.

Consulte a documentação do SUSE Linux Micro 6.1 (https://documentation.suse.com/sle-micro/6.1/html/Micro-deployment-images/index.html#alp-images-installer-type) para obter mais detalhes.

Esse processo funciona nas duas arquiteturas AMD64/Intel 64 e AArch64, embora nem todos os perfis de imagem estejam disponíveis para ambas, por exemplo, no SUSE Edge 3.3, em que o SUSE Linux Micro 6.1 é usado, um perfil com kernel em tempo real (ou seja, "Base-RT" ou "Base-RT-SelfInstall") não está disponível para AArch64 no momento.

Nota

É necessário usar um host de build com a mesma arquitetura das imagens que estão sendo criadas. Em outras palavras, para criar uma imagem AArch64, é necessário usar um host de build AArch64, e vice-versa para AMD64/Intel 64. Não há suporte para builds cruzados no momento.

28.1 Pré-requisitos

O construtor de imagens Kiwi requer o seguinte:

- Um host SUSE Linux Micro 6.1 ("sistema de build") com a mesma arquitetura da imagem que está sendo criada.
- O sistema de build já deve ter sido registrado pelo `SUSEConnect` (o registro é usado para obter os pacotes mais recentes dos repositórios SUSE).
- Uma conexão de Internet para obter os pacotes necessários. Se conectado por proxy, o host de build precisa ser pré-configurado.
- É necessário desabilitar o SELinux no host de build (já que ocorre a rotulagem do SELinux no contêiner e isso pode entrar em conflito com a política do host).
- Pelo menos 10 GB de espaço livre no disco para acomodar a imagem do contêiner, a raiz do build e uma ou mais imagens de saída resultantes.

28.2 Introdução

Devido a algumas limitações, é necessário desabilitar o SELinux. Conecte-se ao host de build da imagem do SUSE Linux Micro 6.1 e confirme se o SELinux está desabilitado:

```
# setenforce 0
```

Crie um diretório de saída para compartilhar com o contêiner de build do Kiwi no qual salvar as imagens resultantes:

```
# mkdir ~/output
```

Extraia a imagem mais recente do construtor Kiwi do SUSE Registry:

```
# podman pull registry.suse.com/edge/3.3/kiwi-builder:10.2.12.0
```

```
(...)
```

28.3 Criando a imagem padrão

Este é o comportamento padrão do contêiner de imagens do Kiwi quando nenhum argumento é inserido durante a execução da imagem do contêiner. O seguinte comando executa o `podman` com dois diretórios mapeados para o contêiner:

- O diretório do repositório de pacotes `/etc/zypp/repos.d` do SUSE Linux Micro do host subjacente.
- O diretório de saída `~/output` criado acima.

O contêiner de imagens do Kiwi requer a execução do script auxiliar `build-image` como:

```
# podman run --privileged -v /etc/zypp/repos.d:/micro-sdk/repos/ -v ~/output:/tmp/output
 \
    -it registry.suse.com/edge/3.3/kiwi-builder:10.2.12.0 build-image
(...)
```

Nota

Se você está executando o script pela primeira vez, é esperado que ocorra uma **falha** nele logo depois de ser iniciado com o erro: "**ERROR: Early loop device test failed, please retry the container run.**" (ERRO: Falha no teste do dispositivo no ciclo inicial. Tente executar o contêiner novamente.). Trata-se de um sintoma em que os dispositivos que são criados em loop no sistema host subjacente não ficam imediatamente visíveis dentro da imagem do contêiner. Você apenas precisa executar o comando novamente, e ele deverá prosseguir sem problemas.

Após alguns minutos, as imagens estarão disponíveis no diretório de saída local:

```
(...)
INFO: Image build successful, generated images are available in the 'output' directory.

# ls -1 output/
SLE-Micro.x86_64-6.1.changes
SLE-Micro.x86_64-6.1.packages
SLE-Micro.x86_64-6.1.raw
SLE-Micro.x86_64-6.1.verified
```

```
build
kiwi.result
kiwi.result.json
```

28.4 Criando imagens com outros perfis

Para criar perfis de imagens diferentes, é usada a opção de comando "**-p**" no script auxiliar da imagem do contêiner do Kiwi. Por exemplo, para criar a imagem ISO "**Default-SelfInstall**":

```
# podman run --privileged -v /etc/zypp/repos.d:/micro-sdk/repos/ -v ~/output:/tmp/output
 \
    -it registry.suse.com/edge/3.3/kiwi-builder:10.2.12.0 build-image -p Default-
SelfInstall
(...)
```

Nota

Para evitar perda de dados, o Kiwi recusará a execução se houver imagens no diretório `output`. É necessário remover o conteúdo do diretório de saída antes de prosseguir com o `rm -f output/*`.

Uma alternativa é criar a imagem ISO SelfInstall com o kernel RealTime ("**kernel-rt**"):

```
# podman run --privileged -v /etc/zypp/repos.d:/micro-sdk/repos/ -v ~/output:/tmp/output
 \
    -it registry.suse.com/edge/3.3/kiwi-builder:10.2.12.0 build-image -p Base-RT-
SelfInstall
(...)
```

28.5 Criando imagens com tamanho de setor grande

Alguns modelos de hardware exigem imagens com tamanho de setor grande, ou seja, **4096 bytes** em vez do padrão de 512 bytes. O construtor Kiwi conteinerizado permite gerar imagens com tamanho de bloco grande especificando o parâmetro "**-b**". Por exemplo, para criar a imagem "**Default-SelfInstall**" com um tamanho de setor grande:

```
# podman run --privileged -v /etc/zypp/repos.d:/micro-sdk/repos/ -v ~/output:/tmp/output
 \
```

```
    -it registry.suse.com/edge/3.3/kiwi-builder:10.2.12.0 build-image -p Default-
SelfInstall -b
(...)
```

28.6 Usando um arquivo de definição de imagem personalizado do Kiwi

Para casos de uso avançados, é possível usar um arquivo de definição de imagem personalizado do Kiwi (`SL-Micro.kiwi`) junto com os scripts necessários após a criação. Para isso, substitua as definições padrão predefinidas pela equipe do SUSE Edge.

Crie e mapeie um novo diretório para a imagem do contêiner em que o script auxiliar faz a busca (`/micro-sdk/defs`):

```
# mkdir ~/mydefs/
# cp /path/to/SL-Micro.kiwi ~/mydefs/
# cp /path/to/config.sh ~/mydefs/
# podman run --privileged -v /etc/zypp/repos.d:/micro-sdk/repos/ -v ~/output:/tmp/output
 -v ~/mydefs/:/micro-sdk/defs/ \
    -it registry.suse.com/edge/3.3/kiwi-builder:10.2.12.0 build-image
(...)
```

Atenção

Isso é necessário apenas em casos de uso avançados e pode causar problemas de suporte. Contate seu representante SUSE para receber mais conselhos e orientações.

Para acessar os arquivos de definição de imagem padrão do Kiwi incluídos no contêiner, use os seguintes comandos:

```
$ podman create --name kiwi-builder registry.suse.com/edge/3.3/kiwi-builder:10.2.12.0
$ podman cp kiwi-builder:/micro-sdk/defs/SL-Micro.kiwi .
$ podman cp kiwi-builder:/micro-sdk/defs/SL-Micro.kiwi.4096 .
$ podman rm kiwi-builder
$ ls ./SL-Micro.*
(...)
```

29 Usando clusterclass para implantar clusters downstream

29.1 Introdução

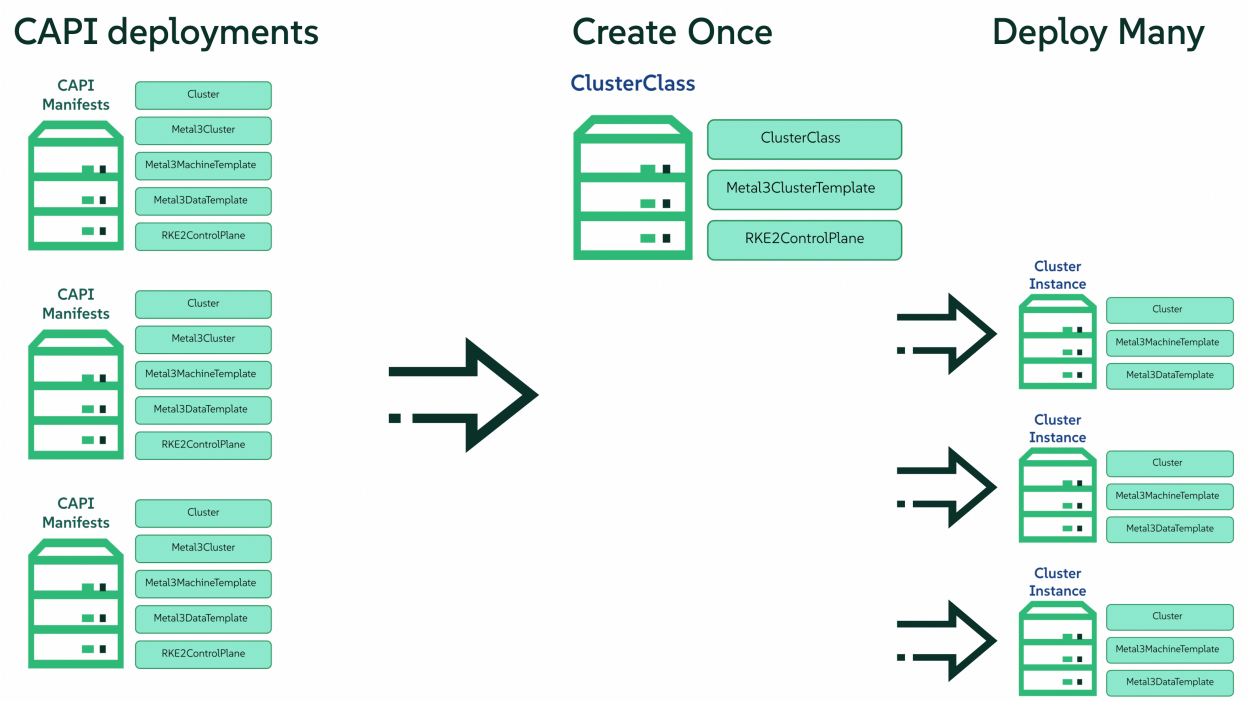
O provisionamento de clusters Kubernetes é uma tarefa complexa que demanda profunda experiência na configuração de componentes de cluster. À medida que as configurações se tornam cada vez mais complexas, ou as demandas de diversos provedores geram inúmeras definições de recursos específicas do provedor, a criação de clusters pode ser um processo assustador. Felizmente, a Kubernetes Cluster API (CAPI) oferece uma abordagem declarativa e mais refinada que é aprimorada pelo ClusterClass. Esse recurso apresenta um modelo orientado por gabarito que permite definir uma classe de cluster reutilizável que encapsula a complexidade e promove a consistência.

29.2 O que é ClusterClass?

O projeto CAPI lançou o recurso ClusterClass como uma mudança de paradigma no gerenciamento do ciclo de vida de clusters Kubernetes por meio da adoção de uma metodologia baseada em gabarito para criação de instâncias de cluster. Em vez de definir recursos separadamente para cada cluster, os usuários definem um ClusterClass, que serve como um diagrama de referência abrangente e reutilizável. Essa representação abstrata encapsula o estado e a configuração desejados de um cluster Kubernetes, o que permite a criação rápida e consistente de vários clusters que seguem as especificações definidas. Essa abstração reduz a sobrecarga de configuração, resultando em manifestos de implantação mais gerenciáveis. Isso significa que os componentes principais de um cluster de carga de trabalho são definidos no nível da classe, permitindo que os usuários recorram a esses gabaritos como variantes de cluster Kubernetes que podem ser reutilizadas uma ou várias vezes para provisionamento de clusters. A implementação do ClusterClass oferece diversas vantagens importantes que abordam os desafios inerentes ao gerenciamento tradicional da CAPI em escala:

- Redução significativa na complexidade e detalhamento do YAML
- Processos otimizados de manutenção e atualização
- Consistência e padronização aprimoradas nas implantações

- Escalabilidade e recursos de automação aprimorados
- Gerenciamento declarativo e controle de versão robusto



29.3 Exemplo de arquivo de provisionamento da CAPI atual

A implantação de um cluster Kubernetes por meio do provedor Cluster API (CAPI) e RKE2 requer a definição de vários recursos personalizados. Esses recursos definem o estado desejado do cluster e sua infraestrutura subjacente, o que permite à CAPI orquestrar o provisionamento e o ciclo de vida de gerenciamento. O trecho do código a seguir ilustra os tipos de recursos que devem ser configurados:

- **Cluster**: esse recurso encapsula as configurações de alto nível, incluindo a topologia de rede que controla a comunicação entre os nós e a descoberta de serviços. Ele também estabelece vínculos essenciais com a especificação do plano de controle e o recurso do provedor de infraestrutura designado e, desse modo, informa a CAPI sobre a arquitetura desejada do cluster e a infraestrutura subjacente na qual ele deverá ser provisionado.
- **Metal3Cluster**: esse recurso define os atributos no nível da infraestrutura exclusivos do Metal3, por exemplo, o endpoint externo pelo qual o servidor da API Kubernetes estará acessível.
- **RKE2ControlPlane**: esse recurso define as características e o comportamento dos nós do plano de controle do cluster. Nessa especificação, os parâmetros, como o número desejado de réplicas do plano de controle (essencial para garantir alta disponibilidade e tolerância a falhas), a versão da distribuição Kubernetes específica (alinhada à versão do RKE2 selecionada) e a estratégia de distribuição das atualizações para os componentes do plano de controle são configurados. Além disso, esse recurso determina a interface de rede de contêiner (CNI, Container Network Interface) que será usada dentro do cluster e facilita a injeção das configurações específicas do agente, em geral aproveitando o Ignition para provisionamento contínuo e automatizado dos agentes do RKE2 nos nós do plano de controle.
- **Metal3MachineTemplate**: esse recurso funciona como um diagrama de referência para criação das instâncias de computação individuais que formam os nós do worker do cluster Kubernetes definindo a imagem que será usada.
- **Metal3DataTemplate**: como complemento ao Metal3MachineTemplate, o recurso Metal3DataTemplate permite especificar metadados adicionais para instâncias de máquina recém-provisionadas.

```
---
```

```yaml
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: emea-spa-cluster-3
  namespace: emea-spa
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - 192.168.0.0/18
    services:
      cidrBlocks:
        - 10.96.0.0/12
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: RKE2ControlPlane
    name: emea-spa-cluster-3
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3Cluster
    name: emea-spa-cluster-3
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3Cluster
metadata:
  name: emea-spa-cluster-3
  namespace: emea-spa
spec:
  controlPlaneEndpoint:
    host: 192.168.122.203
    port: 6443
  noCloudProvider: true
---
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
metadata:
  name: emea-spa-cluster-3
  namespace: emea-spa
spec:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3MachineTemplate
    name: emea-spa-cluster-3
  replicas: 1
  version: v1.32.4+rke2r1
  rolloutStrategy:
    type: "RollingUpdate"
```

```
    rollingUpdate:
      maxSurge: 1
  registrationMethod: "control-plane-endpoint"
  registrationAddress: 192.168.122.203
  serverConfig:
    cni: cilium
    cniMultusEnable: true
    tlsSan:
      - 192.168.122.203
      - https://192.168.122.203.sslip.io
  agentConfig:
    format: ignition
    additionalUserData:
      config: |
        variant: fcos
        version: 1.4.0
        storage:
          files:
            - path: /var/lib/rancher/rke2/server/manifests/endpoint-copier-operator.yaml
              overwrite: true
              contents:
                inline: |
                  apiVersion: helm.cattle.io/v1
                  kind: HelmChart
                  metadata:
                    name: endpoint-copier-operator
                    namespace: kube-system
                  spec:
                    chart: oci://registry.suse.com/edge/charts/endpoint-copier-operator
                    targetNamespace: endpoint-copier-operator
                    version: 303.0.0+up0.2.1
                    createNamespace: true
            - path: /var/lib/rancher/rke2/server/manifests/metallb.yaml
              overwrite: true
              contents:
                inline: |
                  apiVersion: helm.cattle.io/v1
                  kind: HelmChart
                  metadata:
                    name: metallb
                    namespace: kube-system
                  spec:
                    chart: oci://registry.suse.com/edge/charts/metallb
                    targetNamespace: metallb-system
                    version: 303.0.0+up0.14.9
                    createNamespace: true
```

```
            - path: /var/lib/rancher/rke2/server/manifests/metallb-cr.yaml
              overwrite: true
              contents:
                inline: |
                  apiVersion: metallb.io/v1beta1
                  kind: IPAddressPool
                  metadata:
                    name: kubernetes-vip-ip-pool
                    namespace: metallb-system
                  spec:
                    addresses:
                      - 192.168.122.203/32
                    serviceAllocation:
                      priority: 100
                      namespaces:
                        - default
                      serviceSelectors:
                        - matchExpressions:
                          - {key: "serviceType", operator: In, values: [kubernetes-vip]}
                  ---
                  apiVersion: metallb.io/v1beta1
                  kind: L2Advertisement
                  metadata:
                    name: ip-pool-l2-adv
                    namespace: metallb-system
                  spec:
                    ipAddressPools:
                      - kubernetes-vip-ip-pool
            - path: /var/lib/rancher/rke2/server/manifests/endpoint-svc.yaml
              overwrite: true
              contents:
                inline: |
                  apiVersion: v1
                  kind: Service
                  metadata:
                    name: kubernetes-vip
                    namespace: default
                    labels:
                      serviceType: kubernetes-vip
                  spec:
                    ports:
                    - name: rke2-api
                      port: 9345
                      protocol: TCP
                      targetPort: 9345
                    - name: k8s-api
                      port: 6443
```

```
                    protocol: TCP
                    targetPort: 6443
                  type: LoadBalancer
        systemd:
          units:
            - name: rke2-preinstall.service
              enabled: true
              contents: |
                [Unit]
                Description=rke2-preinstall
                Wants=network-online.target
                Before=rke2-install.service
                ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
                [Service]
                Type=oneshot
                User=root
                ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
                ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/$(jq -r .uuid /mnt/
openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
                ExecStart=/bin/sh -c "echo \"node-name: $(jq -r .name /mnt/openstack/
latest/meta_data.json)\" >> /etc/rancher/rke2/config.yaml"
                ExecStartPost=/bin/sh -c "umount /mnt"
                [Install]
                WantedBy=multi-user.target
    kubelet:
      extraArgs:
        - provider-id=metal3://BAREMETALHOST_UUID
    nodeName: "localhost.localdomain"
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3MachineTemplate
metadata:
  name: emea-spa-cluster-3
  namespace: emea-spa
spec:
  nodeReuse: True
  template:
    spec:
      automatedCleaningMode: metadata
      dataTemplate:
        name: emea-spa-cluster-3
      hostSelector:
        matchLabels:
          cluster-role: control-plane
          deploy-region: emea-spa
          node: group-3
      image:
```

```
        checksum: http://fileserver.local:8080/eibimage-downstream-cluster.raw.sha256
        checksumType: sha256
        format: raw
        url: http://fileserver.local:8080/eibimage-downstream-cluster.raw
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3DataTemplate
metadata:
  name: emea-spa-cluster-3
  namespace: emea-spa
spec:
  clusterName: emea-spa-cluster-3
  metaData:
    objectNames:
      - key: name
        object: machine
      - key: local-hostname
        object: machine
      - key: local_hostname
        object: machine
```

29.4 Transformando o arquivo de provisionamento da CAPI em ClusterClass

29.4.1 Definição do ClusterClass

O código a seguir define o recurso ClusterClass, um gabarito declarativo para implantação consistente de um tipo específico de cluster Kubernetes. Essa especificação inclui as configurações comuns de infraestrutura e de plano de controle, o que permite o provisionamento eficiente e o gerenciamento do ciclo de vida uniforme de toda a frota de clusters. Há algumas variáveis no exemplo abaixo do clusterclass que serão substituídas pelos valores reais durante o processo de criação de instância do cluster. As seguintes variáveis são usadas no exemplo:

- `controlPlaneMachineTemplate`: esse é o nome para definir a referência de gabarito da máquina do ControlPlane que será usada
- `controlPlaneEndpointHost`: esse é o nome de host ou endereço IP do endpoint do plano de controle
- `tlsSan`: esse é o nome alternativo da entidade TLS para o endpoint do plano de controle

O arquivo de definição de clusterclass é definido com base nos três recursos a seguir:

- **ClusterClass**: esse recurso encapsula a definição da classe do cluster inteira, incluindo os gabaritos de plano de controle e de infraestrutura. Além disso, ele inclui a lista das variáveis que serão substituídas durante o processo de criação de instâncias.
- **RKE2ControlPlaneTemplate**: esse recurso define o gabarito do plano de controle, especificando a configuração desejada para os nós do plano de controle. Ele inclui os parâmetros, como número de réplicas, versão do Kubernetes e CNI, que serão usados. Alguns parâmetros também serão substituídos pelos valores corretos durante o processo de criação de instâncias.
- **Metal3ClusterTemplate**: esse recurso define o gabarito de infraestrutura, especificando a configuração desejada da infraestrutura subjacente. Ele inclui parâmetros, como o endpoint do plano de controle e o sinalizador noCloudProvider. Alguns parâmetros também serão substituídos pelos valores corretos durante o processo de criação de instâncias.

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlaneTemplate
metadata:
  name: example-controlplane-type2
  namespace: emea-spa
spec:
  template:
    spec:
      infrastructureRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
        kind: Metal3MachineTemplate
        name: example-controlplane    # This will be replaced by the patch applied in
 each cluster instances
        namespace: emea-spa
      replicas: 1
      version: v1.32.4+rke2r1
      rolloutStrategy:
        type: "RollingUpdate"
        rollingUpdate:
          maxSurge: 1
      registrationMethod: "control-plane-endpoint"
      registrationAddress: "default"  # This will be replaced by the patch applied in
 each cluster instances
      serverConfig:
        cni: cilium
        cniMultusEnable: true
```

```
        tlsSan:
          - "default"  # This will be replaced by the patch applied in each cluster
 instances
      agentConfig:
        format: ignition
        additionalUserData:
          config: |
            default
        kubelet:
          extraArgs:
            - provider-id=metal3://BAREMETALHOST_UUID
        nodeName: "localhost.localdomain"
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3ClusterTemplate
metadata:
  name: example-cluster-template-type2
  namespace: emea-spa
spec:
  template:
    spec:
      controlPlaneEndpoint:
        host: "default"  # This will be replaced by the patch applied in each cluster
 instances
        port: 6443
      noCloudProvider: true
---
apiVersion: cluster.x-k8s.io/v1beta1
kind: ClusterClass
metadata:
  name: example-clusterclass-type2
  namespace: emea-spa
spec:
  variables:
    - name: controlPlaneMachineTemplate
      required: true
      schema:
        openAPIV3Schema:
          type: string
    - name: controlPlaneEndpointHost
      required: true
      schema:
        openAPIV3Schema:
          type: string
    - name: tlsSan
      required: true
      schema:
```

```
      openAPIV3Schema:
        type: array
        items:
          type: string
infrastructure:
  ref:
    kind: Metal3ClusterTemplate
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    name: example-cluster-template-type2
controlPlane:
  ref:
    kind: RKE2ControlPlaneTemplate
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    name: example-controlplane-type2
patches:
  - name: setControlPlaneMachineTemplate
    definitions:
      - selector:
          apiVersion: controlplane.cluster.x-k8s.io/v1beta1
          kind: RKE2ControlPlaneTemplate
          matchResources:
            controlPlane: true
        jsonPatches:
          - op: replace
            path: "/spec/template/spec/infrastructureRef/name"
            valueFrom:
              variable: controlPlaneMachineTemplate
  - name: setControlPlaneEndpoint
    definitions:
      - selector:
          apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
          kind: Metal3ClusterTemplate
          matchResources:
            infrastructureCluster: true  # Added to select InfraCluster
        jsonPatches:
          - op: replace
            path: "/spec/template/spec/controlPlaneEndpoint/host"
            valueFrom:
              variable: controlPlaneEndpointHost
  - name: setRegistrationAddress
    definitions:
      - selector:
          apiVersion: controlplane.cluster.x-k8s.io/v1beta1
          kind: RKE2ControlPlaneTemplate
          matchResources:
            controlPlane: true  # Added to select ControlPlane
        jsonPatches:
```

```
            - op: replace
              path: "/spec/template/spec/registrationAddress"
              valueFrom:
                variable: controlPlaneEndpointHost
    - name: setTlsSan
      definitions:
        - selector:
            apiVersion: controlplane.cluster.x-k8s.io/v1beta1
            kind: RKE2ControlPlaneTemplate
            matchResources:
              controlPlane: true  # Added to select ControlPlane
          jsonPatches:
            - op: replace
              path: "/spec/template/spec/serverConfig/tlsSan"
              valueFrom:
                variable: tlsSan
    - name: updateAdditionalUserData
      definitions:
        - selector:
            apiVersion: controlplane.cluster.x-k8s.io/v1beta1
            kind: RKE2ControlPlaneTemplate
            matchResources:
              controlPlane: true
          jsonPatches:
            - op: replace
              path: "/spec/template/spec/agentConfig/additionalUserData"
              valueFrom:
                template: |
                  config: |
                    variant: fcos
                    version: 1.4.0
                    storage:
                      files:
                        - path: /var/lib/rancher/rke2/server/manifests/endpoint-copier-
operator.yaml
                          overwrite: true
                          contents:
                            inline: |
                              apiVersion: helm.cattle.io/v1
                              kind: HelmChart
                              metadata:
                                name: endpoint-copier-operator
                                namespace: kube-system
                              spec:
                                chart: oci://registry.suse.com/edge/charts/endpoint-
copier-operator
                                targetNamespace: endpoint-copier-operator
```

```
                              version: 303.0.0+up0.2.1
                              createNamespace: true
                        - path: /var/lib/rancher/rke2/server/manifests/metallb.yaml
                          overwrite: true
                          contents:
                            inline: |
                              apiVersion: helm.cattle.io/v1
                              kind: HelmChart
                              metadata:
                                name: metallb
                                namespace: kube-system
                              spec:
                                chart: oci://registry.suse.com/edge/charts/metallb
                                targetNamespace: metallb-system
                                version: 303.0.0+up0.14.9
                                createNamespace: true
                        - path: /var/lib/rancher/rke2/server/manifests/metallb-cr.yaml
                          overwrite: true
                          contents:
                            inline: |
                              apiVersion: metallb.io/v1beta1
                              kind: IPAddressPool
                              metadata:
                                name: kubernetes-vip-ip-pool
                                namespace: metallb-system
                              spec:
                                addresses:
                                  - {{ .controlPlaneEndpointHost }}/32
                                serviceAllocation:
                                  priority: 100
                                  namespaces:
                                    - default
                                  serviceSelectors:
                                    - matchExpressions:
                                      - {key: "serviceType", operator: In, values:
[kubernetes-vip]}
                              ---
                              apiVersion: metallb.io/v1beta1
                              kind: L2Advertisement
                              metadata:
                                name: ip-pool-l2-adv
                                namespace: metallb-system
                              spec:
                                ipAddressPools:
                                  - kubernetes-vip-ip-pool
                        - path: /var/lib/rancher/rke2/server/manifests/endpoint-svc.yaml
                          overwrite: true
```

```
                        contents:
                          inline: |
                            apiVersion: v1
                            kind: Service
                            metadata:
                              name: kubernetes-vip
                              namespace: default
                              labels:
                                serviceType: kubernetes-vip
                            spec:
                              ports:
                              - name: rke2-api
                                port: 9345
                                protocol: TCP
                                targetPort: 9345
                              - name: k8s-api
                                port: 6443
                                protocol: TCP
                                targetPort: 6443
                              type: LoadBalancer
                    systemd:
                      units:
                        - name: rke2-preinstall.service
                          enabled: true
                          contents: |
                            [Unit]
                            Description=rke2-preinstall
                            Wants=network-online.target
                            Before=rke2-install.service
                            ConditionPathExists=!/run/cluster-api/bootstrap-
success.complete
                            [Service]
                            Type=oneshot
                            User=root
                            ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
                            ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/$(jq -
r .uuid /mnt/openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
                            ExecStart=/bin/sh -c "echo \"node-name: $(jq -r .name /mnt/
openstack/latest/meta_data.json)\" >> /etc/rancher/rke2/config.yaml"
                            ExecStartPost=/bin/sh -c "umount /mnt"
                            [Install]
                            WantedBy=multi-user.target
```

29.4.2 Definição da instância do cluster

No contexto do ClusterClass, uma instância de cluster se refere à criação de instância em execução específica de um cluster que foi criado com base no ClusterClass definido. Isso representa a implantação concreta com suas configurações, recursos e estado operacional exclusivos, derivados diretamente do diagrama de referência especificado no ClusterClass. Isso inclui o conjunto específico de máquinas, as configurações de rede e os componentes associados do Kubernetes que estão ativamente em execução. Saber o que é uma instância de cluster é essencial para gerenciar o ciclo de vida, fazer upgrades, executar operações de ajuste de escala e monitorar um determinado cluster implantado que foi provisionado usando a estrutura do ClusterClass.

Para definir uma instância de cluster, precisamos definir os seguintes recursos:

- Cluster
- Metal3MachineTemplate
- Metal3DataTemplate

As variáveis definidas no gabarito (arquivo de definição de clusterclass) serão substituídas pelos valores finais para esta criação de instância do cluster:

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: emea-spa-cluster-3
  namespace: emea-spa
spec:
  topology:
    class: example-clusterclass-type2  # Correct way to reference ClusterClass
    version: v1.32.4+rke2r1
    controlPlane:
      replicas: 1
    variables:                         # Variables to be replaced for this cluster
 instance
      - name: controlPlaneMachineTemplate
        value: emea-spa-cluster-3-machinetemplate
      - name: controlPlaneEndpointHost
        value: 192.168.122.203
      - name: tlsSan
        value:
          - 192.168.122.203
          - https://192.168.122.203.sslip.io
---
```

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3MachineTemplate
metadata:
  name: emea-spa-cluster-3-machinetemplate
  namespace: emea-spa
spec:
  nodeReuse: True
  template:
    spec:
      automatedCleaningMode: metadata
      dataTemplate:
        name: emea-spa-cluster-3
      hostSelector:
        matchLabels:
          cluster-role: control-plane
          deploy-region: emea-spa
          cluster-type: type2
      image:
        checksum: http://fileserver.local:8080/eibimage-downstream-cluster.raw.sha256
        checksumType: sha256
        format: raw
        url: http://fileserver.local:8080/eibimage-downstream-cluster.raw
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3DataTemplate
metadata:
  name: emea-spa-cluster-3
  namespace: emea-spa
spec:
  clusterName: emea-spa-cluster-3
  metaData:
    objectNames:
      - key: name
        object: machine
      - key: local-hostname
        object: machine
```

Essa abordagem permite um processo mais ágil, implantando um cluster com apenas três recursos assim que você define o clusterclass.

IV Dicas e truques

Dicas e truques para componentes do Edge

30 Edge Image Builder

30.1 Comuns

- Se você está em um ambiente não Linux e seguindo estas instruções para criar uma imagem, provavelmente está executando o `Podman` em uma máquina virtual. Por padrão, essa máquina virtual está configurada para ter uma pequena quantidade de recursos do sistema alocada para ela e pode provocar instabilidade no `Edge Image Builder` durante operações de uso intensivo de recursos, como o processo de resolução de RPM. Será necessário ajustar os recursos da máquina do podman usando o Podman Desktop (ícone de engrenagem de configurações → ícone de edição da máquina do podman) ou diretamente pelo comando (https://docs.podman.io/en/stable/markdown/podman-machine-set.1.html) `podman-machine-set`.
- Neste momento, o `Edge Image Builder` não pode criar imagens em uma configuração de arquitetura cruzada, ou seja, você deve executá-lo em:
  - Sistemas AArch64 (como Apple Silicon) para criar imagens `aarch64` do SL Micro
  - Sistemas AMD64/Intel 64 para criar imagens `x86_64` do SL Micro

30.2 Kubernetes

- A criação de clusters Kubernetes de vários nós requer o ajuste da seção `kubernetes` no arquivo de definição para:
  - listar todos os nós de servidor e de agente em `kubernetes.nodes`
  - definir um endereço IP virtual que será usado para que todos os nós não inicializadores ingressem no cluster em `kubernetes.network.apiVIP`
  - (opcional) definir um host de API para especificar o endereço do domínio para acessar o cluster em `kubernetes.network.apiHost`. Para saber mais sobre essa configuração, consulte os documentos da seção do Kubernetes (https://github.com/suse-edge/edge-image-builder/blob/main/docs/building-images.md#kubernetes)
- O `Edge Image Builder` se baseia nos nomes de host dos diferentes nós para determinar o tipo de Kubernetes deles (`server` ou `agent`). Essa configuração é gerenciada no arquivo de definição, mas para a configuração geral de rede das máquinas, podemos usar a configuração de DHCP, conforme descrito no *Capítulo 12, Rede de borda*.

31 Elemental

31.1 Comuns

31.1.1 Expor o serviço do Rancher

Ao usar o RKE2 ou K3s, precisamos expor os serviços (neste contexto, do Rancher) do cluster de gerenciamento já que eles não são expostos por padrão. No RKE2, há um controlador de entrada NGINX, enquanto o k3s usa o Traefik. O fluxo de trabalho atual sugere o uso do MetalLB para anunciar um serviço (anúncio por L2 ou BGP) e do respectivo controlador de entrada para criar uma entrada usando `HelmChartConfig`, já que a criação de um novo objeto de entrada substitui a configuração existente.

1. Instalar o Rancher Prime (pelo Helm) e configurar os valores necessários

```
hostname: rancher-192.168.64.101.sslip.io
replicas: 1
bootstrapPassword: Admin
global.cattle.psp.enabled: "false"
```

Dica

Consulte a documentação de instalação do Rancher (https://ranchermanager.docs.rancher.com/v2.11/getting-started/installation-and-upgrade/install-upgrade-on-a-kubernetes-cluster) para obter mais detalhes.

2. Criar um serviço LoadBalancer para expor o Rancher

```
kubectl apply -f - <<EOF
apiVersion: helm.cattle.io/v1
kind: HelmChartConfig
metadata:
  name: rke2-ingress-nginx
  namespace: kube-system
spec:
  valuesContent: |-
    controller:
      config:
```

```
        use-forwarded-headers: "true"
        enable-real-ip: "true"
      publishService:
        enabled: true
      service:
        enabled: true
        type: LoadBalancer
        externalTrafficPolicy: Local
EOF
```

3. Criar um pool de endereços IP para o serviço usando o endereço IP que já configuramos nos valores do Helm

```
kubectl apply -f - <<EOF
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: ingress-ippool
  namespace: metallb-system
spec:
  addresses:
  - 192.168.64.101/32
  serviceAllocation:
    priority: 100
    serviceSelectors:
    - matchExpressions:
      - {key: app.kubernetes.io/name, operator: In, values: [rke2-ingress-nginx]}
EOF
```

4. Criar um anúncio por L2 para o pool de endereços IP

```
kubectl apply -f - <<EOF
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: ingress-l2-adv
  namespace: metallb-system
spec:
  ipAddressPools:
  - ingress-ippool
EOF
```

5. Garantir que o Elemental seja devidamente instalado

a. Instale o operador Elemental e a respectiva IU nos nós de gerenciamento.

b. Adicione a configuração do Elemental ao nó downstream junto com um código de registro, pois isso solicitará que o Edge Image Builder inclua a opção de registro remoto na máquina.

Dica

Consulte a *Seção 2.5, "Instalar o Elemental"* e a *Seção 2.6, "Configurar o Elemental"* para obter informações e exemplos adicionais.

31.2 Específico do hardware

31.2.1 Trusted Platform Module

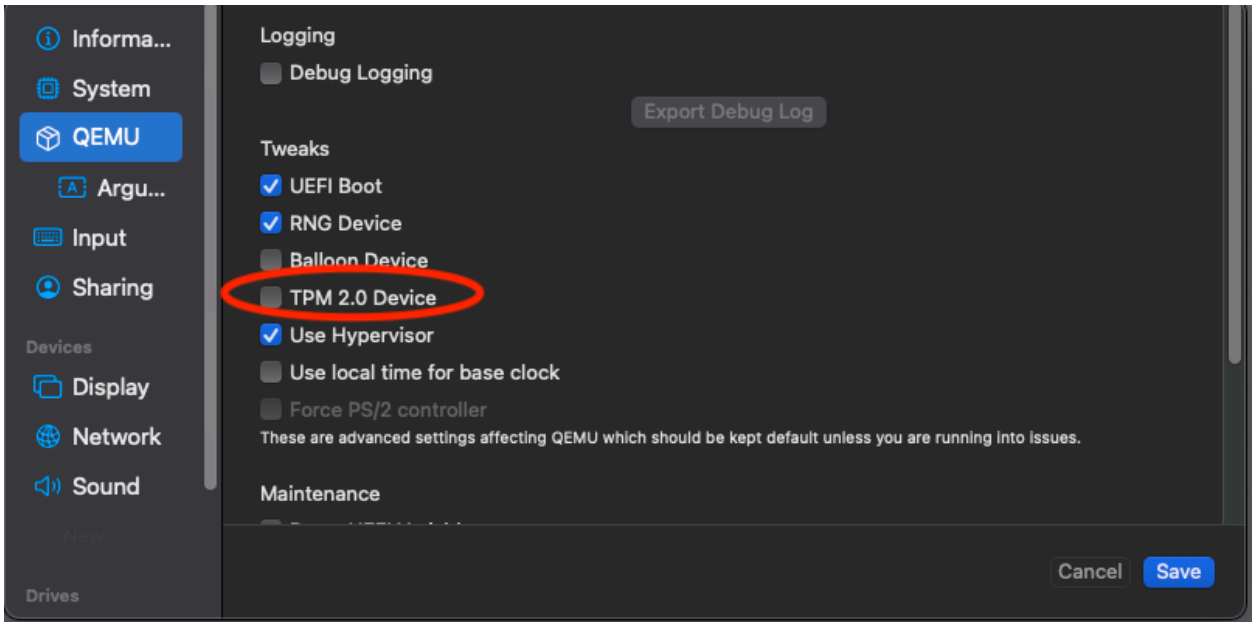
É necessário gerenciar de maneira apropriada a configuração do Trusted Platform Module (https://elemental.docs.rancher.com/tpm/) (TPM). Se isso não for feito, haverá erros semelhantes a estes:

```
Nov 25 18:17:06 eled elemental-register[4038]: Error: registering machine: cannot
 generate authentication token: opening tpm for getting attestation data: TPM device not
 available
```

Para mitigá-los, siga uma das abordagens abaixo:

- Habilitar o TPM nas configurações da máquina virtual

Exemplo com UTM no MacOS



- Emular o TPM usando um valor negativo para a semente do TPM no recurso `MachineRegistration`

```
apiVersion: elemental.cattle.io/v1beta1
kind: MachineRegistration
metadata:
  name: ...
  namespace: ...
spec:
    ...
    elemental:
      ...
      registration:
        emulate-tpm: true
        emulated-tpm-seed: -1
```

- Desabilitar o TPM no recurso `MachineRegistration`

```
apiVersion: elemental.cattle.io/v1beta1
kind: MachineRegistration
metadata:
  name: ...
  namespace: ...
spec:
    ...
    elemental:
      ...
      registration:
```

```
        emulate-tpm: false
```

V Integração de terceiros

Como integrar ferramentas de terceiros

32 NATS

NATS (https://nats.io/) é uma tecnologia conectiva projetada para o mundo cada vez mais hiperconectado. Trata-se de uma tecnologia única que permite que os aplicativos se comuniquem de forma segura com qualquer combinação de fornecedores de nuvem, ambientes locais e dispositivos de borda, web e móveis. O NATS consiste em uma família de produtos de código-fonte aberto estreitamente integrados, mas que podem ser implantados com facilidade e de modo independente. O NATS é usado mundialmente por milhares de empresas, atendendo a casos de uso como microsserviços, computação de borda, dispositivos móveis e IoT, e pode ser usado para expandir ou substituir o serviço de mensagens tradicional.

32.1 Arquitetura

O NATS é uma infraestrutura que permite a troca de dados entre aplicativos na forma de mensagens.

32.1.1 Aplicativos clientes do NATS

É possível usar as bibliotecas de clientes do NATS para permitir que os aplicativos publiquem, assinem, solicitem e respondam entre instâncias diferentes. Esses aplicativos costumam ser chamados de `aplicativos clientes`.

32.1.2 Infraestrutura de serviço do NATS

Os serviços do NATS são fornecidos por um ou mais processos do servidor NATS, que são configurados para interconexão uns com os outros e oferecem a infraestrutura de serviço do NATS, que pode ajustar a escala de um único processo do servidor NATS executado em um dispositivo de endpoint para um supercluster público global com muitos clusters, envolvendo todos os principais provedores de nuvem de todas as regiões do mundo.

32.1.3 Design de mensagens simples

O NATS facilita a comunicação dos aplicativos ao enviar e receber mensagens. Essas mensagens são endereçadas e identificadas por strings de assunto e não dependem do local da rede. Os dados são codificados e estruturados como uma mensagem e enviados por um publicador. A mensagem é recebida, decodificada e processada por um ou mais subscritores.

32.1.4 NATS JetStream

O NATS tem um sistema de persistência distribuída incorporado chamado JetStream. O JetStream foi criado para solucionar os problemas de transmissão identificados nas tecnologias atuais: complexidade, fragilidade e falta de escalabilidade. O JetStream também soluciona o problema de acoplamento entre publicador e subscritor (os subscritores precisam estar em funcionamento para receber a mensagem quando ela for publicada). Há mais informações sobre o NATS JetStream aqui (https://docs.nats.io/nats-concepts/jetstream).

32.2 Instalação

32.2.1 Instalando o NATS no K3s

O NATS foi projetado para várias arquiteturas para ser facilmente instalado no K3s (*Capítulo 15, K3s*).

Vamos criar um arquivo de valores para substituir os valores padrão do NATS.

```
cat > values.yaml <<EOF
cluster:
  # Enable the HA setup of the NATS
  enabled: true
  replicas: 3

nats:
  jetstream:
    # Enable JetStream
    enabled: true

    memStorage:
      enabled: true
```

```
      size: 2Gi

    fileStorage:
      enabled: true
      size: 1Gi
      storageDirectory: /data/
EOF
```

Agora vamos instalar o NATS pelo Helm:

```
helm repo add nats https://nats-io.github.io/k8s/helm/charts/
helm install nats nats/nats --namespace nats --values values.yaml \
 --create-namespace
```

Com o arquivo `values.yaml` acima, os seguintes componentes estarão no namespace `nats`:

1. Versão de alta disponibilidade do NATS Statefulset com três contêineres: servidor NATS, recarregador de configurações e sidecars de métricas.
2. Contêiner NATS box, que vem com um conjunto de utilitários do `NATS` que são usados para verificar a configuração.
3. O JetStream também aproveita o back end de chave-valor incluído nos `PVCs` vinculados aos pods.

32.2.1.1 Testando a configuração

```
kubectl exec -n nats -it deployment/nats-box -- /bin/sh -l
```

1. Crie uma assinatura para a entidade de teste:

   ```
   nats sub test &
   ```

2. Envie uma mensagem para a entidade de teste:

   ```
   nats pub test hi
   ```

32.2.1.2 Limpando

```
helm -n nats uninstall nats
rm values.yaml
```

32.2.2 NATS como back end para K3s

Um componente que o K3s usa é o KINE (https://github.com/k3s-io/kine), um shim que permite a substituição do etcd por back ends de armazenamento alternativos originalmente destinados a bancos de dados relacionais. Como o JetStream fornece uma API de chave-valor, é possível usar o NATS como back end para o cluster K3s.

Já existe uma PR mesclada que simplifica o NATS incorporado no K3s, mas a alteração ainda não está incluída (https://github.com/k3s-io/k3s/issues/7410#issue-1692989394) nas versões do K3s.

Por esse motivo, o binário do K3s deve ser criado manualmente.

32.2.2.1 Criando o K3s

```
git clone --depth 1 https://github.com/k3s-io/k3s.git && cd k3s
```

O seguinte comando adiciona o `nats` às tags de build para habilitar o recurso NATS incorporado no K3s:

```
sed -i '' 's/TAGS="ctrd/TAGS="nats ctrd/g' scripts/build
make local
```

Substitua <node-ip> pelo IP real do nó onde o K3s será iniciado:

```
export NODE_IP=<node-ip>
sudo scp dist/artifacts/k3s-arm64 ${NODE_IP}:/usr/local/bin/k3s
```

Nota

A criação local do K3s requer o plug-in buildx Docker CLI. Ele pode ser manualmente instalado (https://github.com/docker/buildx#manual-download) em caso de falha no `$ make local`.

32.2.2.2 Instalando a CLI do NATS

```
TMPDIR=$(mktemp -d)
nats_version="nats-0.0.35-linux-arm64"
curl -o "${TMPDIR}/nats.zip" -sfL https://github.com/nats-io/natscli/releases/download/
v0.0.35/${nats_version}.zip
unzip "${TMPDIR}/nats.zip" -d "${TMPDIR}"
```

```
sudo scp ${TMPDIR}/${nats_version}/nats ${NODE_IP}:/usr/local/bin/nats
rm -rf ${TMPDIR}
```

32.2.2.3 Executando o NATS como back end do K3s

Vamos acessar o nó por `ssh` e executar o K3s com o sinalizador `--datastore-endpoint` apontando para `nats`.

Nota

O comando a seguir inicia o K3s como um processo em primeiro plano, assim é possível acompanhar os registros facilmente para ver se há problemas. Para não bloquear o terminal atual, adicione o sinalizador `&` antes do comando para iniciá-lo como um processo em segundo plano.

```
k3s server  --datastore-endpoint=nats://
```

Nota

Para tornar o servidor K3s com o NATS como back end permanente em sua VM `slemicro`, execute o script abaixo para criar um serviço `systemd` com as configurações necessárias.

```
export INSTALL_K3S_SKIP_START=false
export INSTALL_K3S_SKIP_DOWNLOAD=true

curl -sfL https://get.k3s.io | INSTALL_K3S_EXEC="server \
 --datastore-endpoint=nats://"  sh -
```

32.2.2.4 Solução de problemas

É possível executar os seguintes comandos no nó para verificar se tudo funciona corretamente com o fluxo:

```
nats str report -a
nats str view -a
```

33 GPUs NVIDIA no SUSE Linux Micro

33.1 Introdução

Este guia demonstra como implementar o suporte à GPU NVIDIA no nível do host usando os drivers de código-fonte aberto (https://github.com/NVIDIA/open-gpu-kernel-modules) predefinidos no SUSE Linux Micro 6.1. Esses drivers são integrados ao sistema operacional, e não dinamicamente carregados pelo NVIDIA GPU Operator (https://github.com/NVIDIA/gpu-operator). Essa configuração é altamente aconselhável para clientes que desejam fazer bake prévio de todos os artefatos necessários para implantação na imagem, e quando não é um requisito a seleção dinâmica da versão do driver, ou seja, o usuário ter que selecionar a versão do driver pelo Kubernetes. Inicialmente, este guia explica como implantar componentes adicionais em um sistema que já tenha sido pré-implantado, mas prossegue com uma seção que descreve como incorporar essa configuração à implantação inicial por meio do Edge Image Builder. Se você não quer passar pelas etapas básicas e configurar tudo manualmente, avance direto para essa seção.

É importante destacar que o suporte a esses drivers é oferecido pela SUSE e NVIDIA em estreita colaboração, sendo que os drivers são criados e distribuídos pela SUSE como parte dos repositórios de pacotes. No entanto, se você tiver qualquer problema ou dúvida sobre a combinação dos drivers usada, contate os gerentes de conta da SUSE ou NVIDIA para receber mais ajuda. Se você pretende usar o NVIDIA AI Enterprise (https://www.nvidia.com/en-gb/data-center/products/ai-enterprise/) (NVAIE), certifique-se de que tenha uma GPU certificada para NVAIE (https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/platform-support.html#supported-nvidia-gpus-and-systems), o que *pode* exigir o uso de drivers proprietários da NVIDIA. Se estiver em dúvida, fale com seu representante NVIDIA.

Este guia *não* aborda mais informações sobre a integração do NVIDIA GPU Operator. A integração do NVIDIA GPU Operator para Kubernetes não é explicada aqui, mas você ainda pode seguir a maioria das etapas neste guia para configurar o sistema operacional subjacente e simplesmente permitir que o GPU Operator use os drivers *pré-instalados* por meio do sinalizador `driver.enabled=false` no gráfico Helm do NVIDIA GPU Operator, que apenas vai selecionar os drivers instalados no host. A NVIDIA apresenta instruções mais detalhadas aqui (https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/install-gpu-operator.html#chart-customization-options).

33.2 Pré-requisitos

Se você está seguindo este guia, já deve ter os seguintes itens disponíveis:

- No mínimo, um host com o SUSE Linux Micro 6.1 instalado, que pode ser físico ou virtual.
- Seus hosts são anexados a uma assinatura porque isso é exigido para acesso ao pacote. Há uma avaliação disponível aqui (https://www.suse.com/download/sle-micro/).
- Uma GPU NVIDIA compatível (https://github.com/NVIDIA/open-gpu-kernel-modules#compatible-gpus) instalada (ou *completamente* transferida para a máquina virtual na qual o SUSE Linux Micro está em execução).
- Acesso ao usuário root. Nestas instruções, consideramos você como usuário root, *sem* escalar seus privilégios por meio do `sudo`.

33.3 Instalação manual

Nesta seção, você vai instalar os drivers da NVIDIA diretamente no sistema operacional SUSE Linux Micro, já que o driver de código aberto da NVIDIA agora faz parte dos repositórios de pacotes principais do SUSE Linux Micro, o que o torna tão fácil de instalar quanto os pacotes RPM obrigatórios. Não há necessidade de compilação nem download de pacotes executáveis. Veja a seguir como implantar a geração "G06" do driver, que oferece suporte às GPUs mais recentes (consulte aqui (https://en.opensuse.org/SDB:NVIDIA_drivers#Install) para obter mais informações). Sendo assim, selecione a geração do driver apropriada à GPU NVIDIA do seu sistema. Para GPUs modernas, o driver "G06" é a opção mais comum.

Antes de começar, é importante reconhecer que, além do driver de código aberto da NVIDIA que a SUSE distribui como parte do SUSE Linux Micro, você precisa de componentes NVIDIA adicionais para sua configuração. Eles podem ser bibliotecas OpenGL, kits de ferramentas CUDA, utilitários de linha de comando, como `nvidia-smi`, e componentes de integração de contêiner, como `nvidia-container-toolkit`. Muitos desses componentes não são distribuídos pela SUSE porque são softwares proprietários da NVIDIA, ou porque não faz sentido distribuí-los no lugar da NVIDIA. Portanto, como parte das instruções, vamos configurar repositórios adicionais para termos acesso a esses componentes e percorrer alguns exemplos de como usar essas ferramentas para obter um sistema totalmente funcional. É importante diferenciar entre os repositórios SUSE e NVIDIA, já que pode haver incompatibilidade entre as versões dos pacotes que a NVIDIA

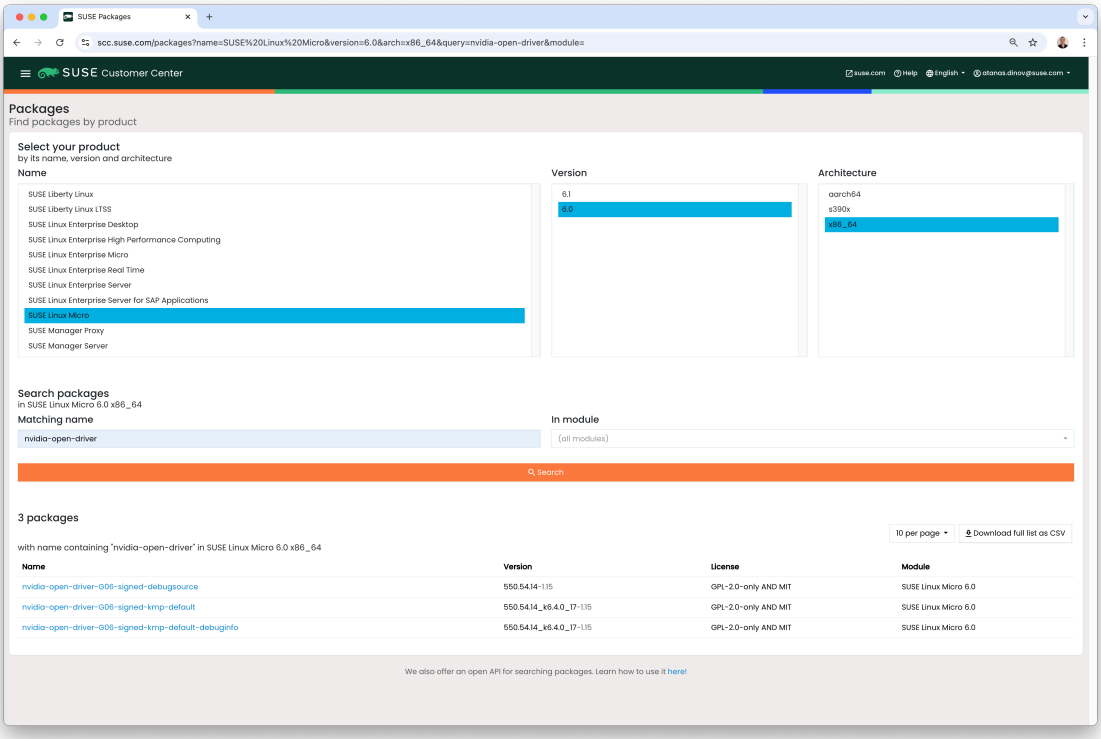
disponibiliza e as que a SUSE compilou. Geralmente, isso acontece quando a SUSE cria uma nova versão do driver de código aberto disponível e leva alguns dias até que os pacotes equivalentes sejam disponibilizados nos repositórios NVIDIA para compatibilidade.

A recomendação é garantir que você selecione uma versão do driver compatível com a GPU e que atenda aos possíveis requisitos de CUDA verificando o seguinte:

- As Notas de lançamento de CUDA (https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/)
- A versão do driver que você pretende implantar tem uma versão correspondente no repositório NVIDIA (https://download.nvidia.com/suse/sle15sp6/x86_64/) e é certeza de que há versões de pacote equivalentes para os componentes de suporte disponíveis

Dica

Para encontrar as versões de driver de código aberto da NVIDIA, execute `zypper se -s nvidia-open-driver` na máquina de destino *ou* pesquise no SUSE Customer Center por "nvidia-open-driver" no SUSE Linux Micro 6.1 para AMD64/Intel 64 (https://scc.suse.com/packages?name=SUSE%20Linux%20Micro&version=6.1&arch=x86_64).



Depois que você confirmar a disponibilidade de uma versão equivalente nos repositórios NVIDIA, estará pronto para instalar os pacotes no sistema operacional host. Para isso, precisamos abrir uma sessão `transactional-update`, que cria um novo instantâneo de leitura/gravação do sistema operacional subjacente para que possamos fazer alterações na plataforma imutável (para obter mais instruções sobre `transactional-update`, consulte aqui (https://documentation.suse.com/sle-micro/6.1/html/Micro-transactional-updates/transactional-updates.html)):

```
transactional-update shell
```

Quando estiver no shell `transactional-update`, adicione outro repositório de pacotes da NVIDIA para que possamos obter os utilitários adicionais, por exemplo, `nvidia-smi`:

```
zypper ar https://download.nvidia.com/suse/sle15sp6/ nvidia-suse-main
zypper --gpg-auto-import-keys refresh
```

Depois disso, você poderá instalar o driver e `nvidia-compute-utils` para os utilitários adicionais. Se os utilitários não forem necessários, você poderá omiti-los. No entanto, é importante instalá-los neste estágio para fins de teste:

```
zypper install -y --auto-agree-with-licenses nvidia-open-driver-G06-signed-kmp nvidia-
compute-utils-G06
```

Nota

Se houver falha na instalação, poderá ser indicativo de uma incompatibilidade de dependência entre a versão do driver selecionada e a que a NVIDIA distribui em seus repositórios. Consulte a seção anterior para verificar se as suas versões são compatíveis. Tente instalar outra versão do driver. Por exemplo, se os repositórios NVIDIA tiverem uma versão mais antiga, tente especificar `nvidia-open-driver-G06-signed-kmp=550.54.14` em seu comando de instalação para especificar uma versão equivalente.

Em seguida, se você *não* usa uma GPU compatível (lembrando que a lista está disponível aqui (https://github.com/NVIDIA/open-gpu-kernel-modules#compatible-gpus)), pode conferir se o driver funciona habilitando o suporte no nível do módulo, mas a sua milhagem poderá variar. Pule esta etapa se você usa uma GPU *compatível*:

```
sed -i '/NVreg_OpenRmEnableUnsupportedGpus/s/^#//g' /etc/modprobe.d/50-nvidia-
default.conf
```

Agora que você instalou os pacotes, saia da sessão `transactional-update`:

```
exit
```

Nota

Antes de prosseguir, garanta que já tenha saído da sessão `transactional-update`.

Agora que você instalou os drivers, faça a reinicialização. Como o SUSE Linux Micro é um sistema operacional imutável, ele precisa ser reinicializado no novo instantâneo criado na etapa anterior. Os drivers são instalados apenas nesse novo instantâneo, já que não é possível carregá-los sem a reinicialização nele, o que é feito automaticamente. Execute o comando de reinicialização quando estiver pronto:

```
reboot
```

Após a reinicialização bem-sucedida do sistema, faça login novamente e use a ferramenta `nvidia-smi` para verificar se o driver foi carregado com sucesso e pode acessar e enumerar as GPUs:

```
nvidia-smi
```

A saída desse comando deve ser similar à mostrada abaixo, levando em consideração que temos duas GPUs no exemplo a seguir:

```
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 545.29.06              Driver Version: 545.29.06    CUDA Version: 12.3     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA A100-PCIE-40GB          Off | 00000000:17:00.0 Off |                    0 |
| N/A   29C    P0              35W / 250W |      4MiB / 40960MiB |      0%      Default |
|                                         |                      |             Disabled |
+-----------------------------------------+----------------------+----------------------+
|   1  NVIDIA A100-PCIE-40GB          Off | 00000000:CA:00.0 Off |                    0 |
| N/A   30C    P0              33W / 250W |      4MiB / 40960MiB |      0%      Default |
|                                         |                      |             Disabled |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
```

```
|        ID   ID                                                                    Usage      |
|=============================================================================================|
|  No running processes found                                                                 |
+---------------------------------------------------------------------------------------------+
```

E assim concluímos o processo de instalação e verificação dos drivers da NVIDIA no sistema SUSE Linux Micro.

33.4 Validação adicional da instalação manual

Neste estágio, a única coisa que podemos verificar é que, no nível do host, é possível acessar o dispositivo NVIDIA e os drivers são carregados com sucesso. No entanto, para garantir que tudo esteja funcionando, há um teste simples para comprovar que a GPU pode receber as instruções de um aplicativo no espaço do usuário, de preferência por meio de um contêiner, e pela biblioteca CUDA, que é o mais comum para uma carga de trabalho real. Para isso, é possível fazer outra modificação no sistema operacional host instalando o `nvidia-container-toolkit` (NVIDIA Container Toolkit (https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html#installing-with-zypper)). Em primeiro lugar, abra outro shell `transactional-update`, lembrando que talvez isso já tenha sido feito em uma transação única na etapa anterior, e veja como fazer isso de maneira totalmente automatizada em uma seção posterior:

```
transactional-update shell
```

Em seguida, instale o pacote `nvidia-container-toolkit` do repositório NVIDIA Container Toolkit:

- O `nvidia-container-toolkit.repo` a seguir contém um repositório estável (`nvidia-container-toolkit`) e outro experimental (`nvidia-container-toolkit-experimental`). O repositório estável é recomendado para uso em produção. O repositório experimental está desabilitado por padrão.

```
zypper ar https://nvidia.github.io/libnvidia-container/stable/rpm/nvidia-container-
toolkit.repo
zypper --gpg-auto-import-keys install -y nvidia-container-toolkit
```

Quando estiver pronto, saia do shell `transactional-update`:

```
exit
```

…e reinicie a máquina em um novo instantâneo:

```
reboot
```

Nota

Como antes, você precisa garantir que saiu do `transactional-shell` e reinicializou a máquina para que as alterações tenham efeito.

Depois que a máquina for reinicializada, verifique se o sistema pode enumerar corretamente os dispositivos usando o NVIDIA Container Toolkit. A saída deve ser detalhada, com mensagens INFO e WARN, mas sem mensagens ERROR:

```
nvidia-ctk cdi generate --output=/etc/cdi/nvidia.yaml
```

Esse procedimento garante que os contêineres iniciados na máquina possam usar os dispositivos de GPU NVIDIA que foram descobertos. Quando estiver pronto, você poderá executar um contêiner baseado em podman. Usar o `podman` é uma boa forma de validar o acesso ao dispositivo NVIDIA de dentro de um contêiner, o que traz segurança para fazer o mesmo com o Kubernetes no futuro. Conceda ao `podman` acesso para os dispositivos NVIDIA identificados que foram processados pelo comando anterior, com base na SLE BCI (https://registry.suse.com/repositories/bci-bci-base-15sp6), e simplesmente execute o comando Bash:

```
podman run --rm --device nvidia.com/gpu=all --security-opt=label=disable -it
 registry.suse.com/bci/bci-base:latest bash
```

Agora você executará os comandos de um contêiner podman temporário. Ele não tem acesso ao seu sistema subjacente e é efêmero, portanto, o que quer que se faça nele não vai persistir, e você não poderá danificar nada no host subjacente. Como estamos em um contêiner, podemos instalar as bibliotecas CUDA necessárias; mais uma vez, verificando a versão correta do CUDA para seu driver aqui (https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/), embora a saída anterior de `nvidia-smi` deva mostrar a versão necessária do CUDA. No exemplo a seguir, vamos instalar o *CUDA 12.3* e extrair muitos exemplos, demonstrações e kits de desenvolvimento para você validar o GPU na íntegra:

```
zypper ar https://developer.download.nvidia.com/compute/cuda/repos/sles15/x86_64/ cuda-
suse
zypper in -y cuda-libraries-devel-12-3 cuda-minimal-build-12-3 cuda-demo-suite-12-3
```

Após a devida instalação, não saia do contêiner. Executaremos o exemplo `deviceQuery` de CUDA, que valida de maneira completa o acesso da GPU por CUDA, e de dentro do próprio contêiner:

```
/usr/local/cuda-12/extras/demo_suite/deviceQuery
```

Se tudo correr bem, você verá uma saída como esta mostrada a seguir, com destaque para a mensagem `Result = PASS` no fim do comando e para a identificação correta das duas GPUs feita pelo sistema, considerando que seu ambiente pode ter apenas uma:

```
/usr/local/cuda-12/extras/demo_suite/deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 2 CUDA Capable device(s)

Device 0: "NVIDIA A100-PCIE-40GB"
  CUDA Driver Version / Runtime Version          12.2 / 12.1
  CUDA Capability Major/Minor version number:    8.0
  Total amount of global memory:                 40339 MBytes (42298834944 bytes)
  (108) Multiprocessors, ( 64) CUDA Cores/MP:     6912 CUDA Cores
  GPU Max Clock rate:                            1410 MHz (1.41 GHz)
  Memory Clock rate:                             1215 Mhz
  Memory Bus Width:                              5120-bit
  L2 Cache Size:                                 41943040 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536),
 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 3 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Enabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Compute Preemption:            Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 23 / 0
  Compute Mode:
```

```
     < Default (multiple host threads can use ::cudaSetDevice() with device
 simultaneously) >

Device 1: <snip to reduce output for multiple devices>
     < Default (multiple host threads can use ::cudaSetDevice() with device
 simultaneously) >
> Peer access from NVIDIA A100-PCIE-40GB (GPU0) -> NVIDIA A100-PCIE-40GB (GPU1) : Yes
> Peer access from NVIDIA A100-PCIE-40GB (GPU1) -> NVIDIA A100-PCIE-40GB (GPU0) : Yes

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 12.3, CUDA Runtime Version =
 12.3, NumDevs = 2, Device0 = NVIDIA A100-PCIE-40GB, Device1 = NVIDIA A100-PCIE-40GB
Result = PASS
```

A partir deste ponto, você pode continuar executando qualquer outra carga de trabalho CUDA. Use os compiladores e qualquer outro elemento do ecossistema CUDA para realizar mais testes. Ao concluir, saia do contêiner, lembrando que tudo o que você instalou nele é efêmero (será perdido!) e não afeta o sistema operacional subjacente:

```
exit
```

33.5 Implementação com o Kubernetes

Agora que comprovamos a instalação e o uso do driver de código aberto da NVIDIA no SUSE Linux Micro, vamos explorar a configuração do Kubernetes na mesma máquina. Este guia não orienta na implantação do Kubernetes, mas ele considera que você já tenha o K3s (https://k3s.io/) ou o RKE2 (https://docs.rke2.io/install/quickstart) instalado e que o kubeconfig esteja devidamente configurado, de modo que os comandos `kubectl` padrão possam ser executados como superusuário. Supomos que o seu nó forme um cluster de nó único, embora as etapas principais sejam similares nos clusters de vários nós. Primeiramente, garanta que o acesso ao `kubectl` esteja funcionando:

```
kubectl get nodes
```

Esse comando deve retornar algo semelhante ao seguinte:

```
NAME       STATUS   ROLES                       AGE   VERSION
node0001   Ready    control-plane,etcd,master   13d   v1.32.4+rke2r1
```

Você deve descobrir que a instalação do k3s/rke2 detectou o NVIDIA Container Toolkit no host e configurou automaticamente a integração do runtime NVIDIA ao `containerd` (a interface de tempo de execução do contêiner que o k3s/rke2 usa). Para confirmar isso, verifique o arquivo `config.toml` de containerd:

```
tail -n8 /var/lib/rancher/rke2/agent/etc/containerd/config.toml
```

Isso deve retornar algo semelhante ao que está mostrado a seguir. O local equivalente do K3s é `/var/lib/rancher/k3s/agent/etc/containerd/config.toml`:

```
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes."nvidia"]
  runtime_type = "io.containerd.runc.v2"
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes."nvidia".options]
  BinaryName = "/usr/bin/nvidia-container-runtime"
```

Nota

Se estas entradas não estiverem presentes, talvez a detecção tenha falhado. O motivo pode ser a máquina ou os serviços do Kubernetes que não foram reiniciados. Se necessário, adicione-os manualmente conforme explicado acima.

Depois disso, precisamos configurar o `RuntimeClass` NVIDIA como runtime do Kubernetes adicional ao padrão, garantindo que as solicitações dos usuários para pods que precisam de acesso à GPU possam usar o NVIDIA Container Toolkit para fazer isso, por meio do comando `nvidia-container-runtime`, conforme definido na configuração de `containerd`:

```
kubectl apply -f - <<EOF
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: nvidia
handler: nvidia
EOF
```

A próxima etapa é configurar o plug-in de dispositivo NVIDIA (https://github.com/NVIDIA/k8s-device-plugin), que configura o Kubernetes para aproveitar as GPUs NVIDIA como recursos no cluster que pode ser usado, trabalhando em conjunto com o NVIDIA Container Toolkit. Inicialmente, essa ferramenta detecta todos os recursos no host subjacente, incluindo GPUs, drivers e outros (como GL), e permite solicitar recursos para a GPU e consumi-los como parte dos seus aplicativos.

Primeiro, você precisa adicionar e atualizar o repositório Helm para o plug-in de dispositivo NVIDIA:

```
helm repo add nvdp https://nvidia.github.io/k8s-device-plugin
helm repo update
```

Agora você pode instalar o plug-in de dispositivo NVIDIA:

```
helm upgrade -i nvdp nvdp/nvidia-device-plugin --namespace nvidia-device-plugin --create-
namespace --version 0.14.5 --set runtimeClassName=nvidia
```

Após alguns minutos, você verá um novo pod em execução que concluirá a detecção em seus nós disponíveis e os marcará com o número de GPUs que foram detectadas:

```
kubectl get pods -n nvidia-device-plugin
NAME                              READY   STATUS    RESTARTS      AGE
nvdp-nvidia-device-plugin-jp697   1/1     Running   2 (12h ago)   6d3h

kubectl get node node0001 -o json | jq .status.capacity
{
  "cpu": "128",
  "ephemeral-storage": "466889732Ki",
  "hugepages-1Gi": "0",
  "hugepages-2Mi": "0",
  "memory": "32545636Ki",
  "nvidia.com/gpu": "1",                      <----
  "pods": "110"
}
```

Agora você está pronto para criar um pod NVIDIA que tenta usar essa GPU. Vamos tentar com o contêiner de benchmark CUDA:

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: nbody-gpu-benchmark
  namespace: default
spec:
  restartPolicy: OnFailure
  runtimeClassName: nvidia
  containers:
  - name: cuda-container
    image: nvcr.io/nvidia/k8s/cuda-sample:nbody
    args: ["nbody", "-gpu", "-benchmark"]
```

```
    resources:
      limits:
        nvidia.com/gpu: 1
    env:
    - name: NVIDIA_VISIBLE_DEVICES
      value: all
    - name: NVIDIA_DRIVER_CAPABILITIES
      value: all
EOF
```

Se tudo correu bem, você pode consultar os registros para ver as informações de benchmark:

```
kubectl logs nbody-gpu-benchmark
Run "nbody -benchmark [-numbodies=<numBodies>]" to measure performance.
 -fullscreen       (run n-body simulation in fullscreen mode)
 -fp64             (use double precision floating point values for simulation)
 -hostmem          (stores simulation data in host memory)
 -benchmark        (run benchmark to measure performance)
 -numbodies=<N>    (number of bodies (>= 1) to run in simulation)
 -device=<d>       (where d=0,1,2.... for the CUDA device to use)
 -numdevices=<i>   (where i=(number of CUDA devices > 0) to use for simulation)
 -compare          (compares simulation results running once on the default GPU and once
 on the CPU)
 -cpu              (run n-body simulation on the CPU)
 -tipsy=<file.bin> (load a tipsy model file for simulation)

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when
 GPU Boost is enabled.

> Windowed mode
> Simulation data stored in video memory
> Single precision floating point simulation
> 1 Devices used for simulation
GPU Device 0: "Turing" with compute capability 7.5

> Compute 7.5 CUDA device: [Tesla T4]
40960 bodies, total time for 10 iterations: 101.677 ms
= 165.005 billion interactions per second
= 3300.103 single-precision GFLOP/s at 20 flops per interaction
```

Por fim, se os seus aplicativos exigem OpenGL, instale as bibliotecas OpenGL da NVIDIA necessárias no nível do host; e o plug-in de dispositivo NVIDIA e o NVIDIA Container Toolkit podem torná-las disponíveis aos contêineres. Para isso, instale o pacote da seguinte maneira:

```
transactional-update pkg install nvidia-gl-G06
```

Nota

Você precisa reinicializar para disponibilizar esse pacote aos seus aplicativos. O plug-in de dispositivo NVIDIA deve fazer essa nova detecção automaticamente por meio do NVIDIA Container Toolkit.

33.6 Reunindo tudo com o Edge Image Builder

Você demonstrou a total funcionalidade dos aplicativos e das GPUs no SUSE Linux Micro e agora quer usar o *Capítulo 11, Edge Image Builder* para disponibilizar tudo em uma imagem de disco ISO ou RAW implantável/consumível. Este guia não explica como usar o Edge Image Builder, mas aborda as configurações necessárias para criar essa imagem. Veja a seguir um exemplo de definição de imagem, junto com os arquivos de configuração necessários do Kubernetes, para garantir que todos os componentes obrigatórios sejam implantados imediatamente. Esta é a estrutura de diretórios do Edge Image Builder para o exemplo apresentado abaixo:

```
.
├── base-images
│   └── SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso
├── eib-config-iso.yaml
├── kubernetes
│   ├── config
│   │   └── server.yaml
│   ├── helm
│   │   └── values
│   │       └── nvidia-device-plugin.yaml
│   └── manifests
│       └── nvidia-runtime-class.yaml
└── rpms
    └── gpg-keys
        └── nvidia-container-toolkit.key
```

Vamos explorar esses arquivos. Primeiro, esta é uma definição da imagem de amostra para um cluster de nó único com o K3s e que também implanta os utilitários e os pacotes OpenGL (`eib-config-iso.yaml`):

```
apiVersion: 1.2
image:
  arch: x86_64
  imageType: iso
  baseImage: SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso
```

```
  outputImageName: deployimage.iso
operatingSystem:
  time:
    timezone: Europe/London
    ntp:
      pools:
        - 2.suse.pool.ntp.org
  isoConfiguration:
    installDevice: /dev/sda
  users:
    - username: root
      encryptedPassword: $6$XcQN1xkuQKjWEtQG
$WbhV80rbveDLJDz1c93K5Ga9JDjt3mF.ZUnhYtsS7uE52FR8mmT8Cnii/JPeFk9jzQO6eapESYZesZHO9EslD1
  packages:
    packageList:
      - nvidia-open-driver-G06-signed-kmp-default
      - nvidia-compute-utils-G06
      - nvidia-gl-G06
      - nvidia-container-toolkit
    additionalRepos:
      - url: https://download.nvidia.com/suse/sle15sp6/
      - url: https://nvidia.github.io/libnvidia-container/stable/rpm/x86_64
    sccRegistrationCode: [snip]
kubernetes:
  version: v1.32.4+k3s1
  helm:
    charts:
      - name: nvidia-device-plugin
        version: v0.14.5
        installationNamespace: kube-system
        targetNamespace: nvidia-device-plugin
        createNamespace: true
        valuesFile: nvidia-device-plugin.yaml
        repositoryName: nvidia
    repositories:
      - name: nvidia
        url: https://nvidia.github.io/k8s-device-plugin
```

Nota

Este é apenas um exemplo. Você pode precisar personalizá-lo de acordo com os seus requisitos e expectativas. Além disso, se você usa o SUSE Linux Micro, precisa inserir seu próprio `sccRegistrationCode` para resolver as dependências de pacotes e obter os drivers da NVIDIA.

Além disso, precisamos adicionar outros componentes para que sejam carregados pelo Kubernetes no momento da inicialização. O diretório do EIB precisa primeiro de um diretório `kubernetes`, com os subdiretórios para configuração, os valores dos gráficos Helm e todos os manifestos adicionais necessários:

```
mkdir -p kubernetes/config kubernetes/helm/values kubernetes/manifests
```

Agora vamos definir a configuração (opcional) do Kubernetes escolhendo uma CNI (que assume o Cilium como padrão se nada for selecionado) e habilitando o SELinux:

```
cat << EOF > kubernetes/config/server.yaml
cni: cilium
selinux: true
EOF
```

Agora garanta que o RuntimeClass NVIDIA tenha sido criado no cluster Kubernetes:

```
cat << EOF > kubernetes/manifests/nvidia-runtime-class.yaml
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: nvidia
handler: nvidia
EOF
```

Usamos o Helm Controller incorporado para implantar o plug-in de dispositivo NVIDIA pelo próprio Kubernetes. Vamos inserir uma classe de runtime no arquivo de valores do gráfico:

```
cat << EOF > kubernetes/helm/values/nvidia-device-plugin.yaml
runtimeClassName: nvidia
EOF
```

Precisamos obter a chave pública do RPM do NVIDIA Container Toolkit antes de continuar:

```
mkdir -p rpms/gpg-keys
curl -o rpms/gpg-keys/nvidia-container-toolkit.key https://nvidia.github.io/libnvidia-
container/gpgkey
```

Todos os artefatos necessários, incluindo o binário do Kubernetes, as imagens do contêiner, os gráficos Helm (e todas as imagens referenciadas), serão automaticamente isolados. Isso significa que, por padrão, o sistema não deve exigir conectividade de Internet no momento da implantação. Agora você precisa apenas obter a ISO do SUSE Linux Micro da página de

downloads da SUSE (https://www.suse.com/download/sle-micro/) (e salvá-la no diretório `base-images`) e chamar a ferramenta Edge Image Builder para gerar a ISO para você. Para concluir o exemplo, este é o comando que foi usado para criar a imagem:

```
podman run --rm --privileged -it -v /path/to/eib-files/:/eib \
registry.suse.com/edge/3.3/edge-image-builder:1.2.1 \
build --definition-file eib-config-iso.yaml
```

Para obter mais instruções, consulte a documentação (https://github.com/suse-edge/edge-image-builder/blob/release-1.2/docs/building-images.md) do Edge Image Builder.

33.7 Resolvendo problemas

33.7.1 nvidia-smi não encontra a GPU

Verifique as mensagens do kernel usando o `dmesg`. Se isso indicar que não é possível alocar o `NvKMSKapDevice`, adote a solução alternativa para a GPU sem suporte:

```
sed -i '/NVreg_OpenRmEnableUnsupportedGpus/s/^#//g' /etc/modprobe.d/50-nvidia-
default.conf
```

> *NOTA*: Será necessário recarregar, ou reinicializar, o módulo do kernel se você alterar a configuração dele na etapa anterior para que as alterações entrem em vigor.

VI Operações de dia 2

Esta seção explica como os administradores podem realizar as tarefas de operações de "dia 2" nos clusters de gerenciamento e downstream.

34 Migração do Edge 3.3

Esta seção explica como migrar os clusters de `gerenciamento` e `downstream` do `Edge 3.2` para o `Edge 3.3.0`.

Importante

Faça sempre as migrações de cluster da versão `Z-stream mais recente` do `Edge 3.2`.

Migre sempre para a versão `Edge 3.3.0`. Para os upgrades pós-migração subsequentes, consulte as seções referentes ao cluster de gerenciamento (*Capítulo 35, Cluster de gerenciamento*) e downstream (*Capítulo 36, Clusters downstream*).

34.1 Cluster de gerenciamento

Esta seção aborda os seguintes tópicos:

Seção 34.1.1, "Pré-requisitos": etapas de pré-requisito para concluir antes de iniciar a migração.

Seção 34.1.2, "Controller de upgrade": como migrar um cluster de `gerenciamento` usando o *Capítulo 23, Controller de upgrade*.

Seção 34.1.3, "Fleet": como migrar um cluster de `gerenciamento` usando o *Capítulo 8, Fleet*.

34.1.1 Pré-requisitos

34.1.1.1 Fazer upgrade das CRDs do Bare Metal Operator

Nota

Aplica-se apenas a clusters que exigem upgrade do gráfico do *Capítulo 10, Metal*[3].

O gráfico Helm do `Metal3` inclui as CRDs do Bare Metal Operator (BMO) (https://book.metal3.io/bmo/introduction.html) aproveitando o diretório CRD (https://helm.sh/docs/chart_best_practices/custom_resource_definitions/#method-1-let-helm-do-it-for-you) do Helm.

Entretanto, essa abordagem tem algumas limitações, especificamente a incapacidade de fazer upgrade das CRDs nesse diretório usando o Helm. Para obter mais informações, consulte a documentação do Helm (https://helm.sh/docs/chart_best_practices/custom_resource_definitions/#some-caveats-and-explanations).

Como resultado, antes de fazer upgrade do Metal[3] para uma versão compatível do `Edge 3.3.0`, os usuários devem fazer upgrade manual das CRDs subjacentes do BMO.

Em uma máquina com o `Helm` instalado e o `kubectl` configurado para apontar para o cluster de `gerenciamento`:

1. Aplique manualmente as CRDs do BMO:

```
helm show crds oci://registry.suse.com/edge/charts/metal3 --version 303.0.7+up0.11.5
 | kubectl apply -f -
```

34.1.2 Controller de upgrade

Importante

O `Controller de upgrade` oferece suporte a migrações de versão do Edge somente para clusters de **gerenciamento não air-gapped**.

Os seguintes tópicos são abordados como parte desta seção:

Seção 34.1.2.1, "Pré-requisitos": pré-requisitos específicos para o `Controller de upgrade`.

Seção 34.1.2.2, "Etapas de migração": etapas para migrar o cluster de `gerenciamento` para uma nova versão do Edge usando o `Controller de upgrade`.

34.1.2.1 Pré-requisitos

34.1.2.1.1 Controller de upgrade do Edge 3.3

Antes de usar o `Controller de upgrade`, verifique se ele executa uma versão com capacidade de migração para o lançamento desejado do Edge.

Para fazer isso:

1. Se você já tem o `Controller de upgrade` implantado de uma versão anterior do Edge, faça upgrade do respectivo gráfico:

   ```
   helm upgrade upgrade-controller -n upgrade-controller-system oci://
   registry.suse.com/edge/charts/upgrade-controller --version 303.0.1+up0.1.1
   ```

2. Se você **não** tem o `Controller de upgrade` implantado, siga a *Seção 23.3, "Instalando o Controller de upgrade"*.

34.1.2.2 Etapas de migração

A execução da migração de um cluster de `gerenciamento` com o `Controller de upgrade` é basicamente similar à execução de um upgrade.

A única diferença é que o `UpgradePlan` **deve** especificar a versão de lançamento `3.3.0`:

```
apiVersion: lifecycle.suse.com/v1alpha1
kind: UpgradePlan
metadata:
  name: upgrade-plan-mgmt
  # Change to the namespace of your Upgrade Controller
  namespace: CHANGE_ME
spec:
  releaseVersion: 3.3.0
```

Para obter informações sobre como usar o `UpgradePlan` acima para fazer uma migração, consulte Processo de upgrade do Controller de upgrade (*Seção 35.1, "Controller de upgrade"*).

34.1.3 Fleet

Nota

Sempre que possível, siga as instruções da *Seção 34.1.2, "Controller de upgrade"* para a migração.

Consulte esta seção apenas para casos de uso não cobertos pelo `Controller de upgrade`.

A execução da migração de um cluster de `gerenciamento` com o `Fleet` é basicamente similar a de um upgrade.

As **principais** diferenças são:

1. As instâncias do Fleet **devem ser usadas** a partir da versão release-3.3.0 (https://github.com/suse-edge/fleet-examples/releases/tag/release-3.3.0) do repositório `suse-edge/fleet-examples`.
2. Os gráficos programados para upgrade **devem** ser atualizados para versões compatíveis com o `Edge 3.3.0`. Para ver uma lista de componentes do `Edge 3.3.0`, consulte a *Seção 52.4, "Versão 3.3.0"*.

Importante

Para garantir a migração bem-sucedida do `Edge 3.3.0`, é importante que os usuários cumpram os pontos descritos acima.

Considerando os pontos acima, os usuários podem seguir a documentação do Fleet sobre cluster de `gerenciamento` (*Seção 35.2, "Fleet"*) para obter um guia completo das etapas necessárias à migração.

34.2 Clusters downstream

Seção 34.2.1, "Fleet": como migrar um cluster `downstream` usando o *Capítulo 8, Fleet*.

34.2.1 Fleet

A execução da migração de um cluster `downstream` com o `Fleet` é basicamente similar a de um upgrade.

As **principais** diferenças são:

1. As instâncias do Fleet **devem ser usadas** a partir da versão release-3.3.0 (https://github.com/suse-edge/fleet-examples/releases/tag/release-3.3.0) do repositório `suse-edge/fleet-examples`.
2. Os gráficos programados para upgrade **devem** ser atualizados para versões compatíveis com o `Edge 3.3.0`. Para ver uma lista de componentes do `Edge 3.3.0`, consulte a *Seção 52.4, "Versão 3.3.0"*.

Importante

Para garantir a migração bem-sucedida do `Edge 3.3.0`, é importante que os usuários cumpram os pontos descritos acima.

Considerando os pontos acima, os usuários podem seguir a documentação do Fleet sobre cluster `downstream` (*Seção 36.1, "Fleet"*) para obter um guia completo das etapas necessárias à migração.

35 Cluster de gerenciamento

Atualmente, há duas maneiras de realizar as operações de "dia 2" no cluster de `gerenciamento`:

1. Usando o *Capítulo 23, Controller de upgrade*: *Seção 35.1, “Controller de upgrade”*
2. Usando o *Capítulo 8, Fleet*: *Seção 35.2, “Fleet”*

35.1 Controller de upgrade

Importante

O `Controller de upgrade` oferece suporte apenas a operações de `dia 2` em clusters de **gerenciamento não air-gapped**.

Esta seção explica como realizar as diversas operações de `dia 2` relacionadas ao upgrade do cluster de `gerenciamento` de uma versão da plataforma Edge para outra.

O Controller de upgrade (*Capítulo 23, Controller de upgrade*) automatiza as operações de `dia 2`, o que inclui:

- Upgrade de sistema operacional SUSE Linux Micro (*Capítulo 9, SUSE Linux Micro*)
- Upgrade de Kubernetes com *Capítulo 16, RKE2* ou *Capítulo 15, K3s*
- Upgrade de componentes adicionais do SUSE (SUSE Rancher Prime, SUSE Security etc.)

35.1.1 Pré-requisitos

Antes de fazer upgrade do cluster de `gerenciamento`, é necessário cumprir os seguintes pré-requisitos:

1. `Nós registrados do SCC`: assegure que os nós do cluster do seu sistema operacional sejam registrados com uma chave de assinatura compatível com a versão de sistema operacional especificada no lançamento do Edge (*Seção 52.1, "Resumo"*) para o qual você pretende fazer upgrade.

2. `Controller de upgrade`: garanta que o `Controller de upgrade` tenha sido implantado no cluster de `gerenciamento`. Para ver as etapas de instalação, consulte a *Seção 23.3, "Instalando o Controller de upgrade"*.

35.1.2 Fazer upgrade

1. Determine a versão de lançamento do Edge (*Seção 52.1, "Resumo"*) para a qual você quer fazer upgrade do seu cluster de `gerenciamento`.

2. No cluster de `gerenciamento`, implante um `UpgradePlan` que especifique a `versão de lançamento` desejada. É necessário implantar o `UpgradePlan` no namespace do `Controller de upgrade`.

```
kubectl apply -n <upgrade_controller_namespace> -f - <<EOF
apiVersion: lifecycle.suse.com/v1alpha1
kind: UpgradePlan
metadata:
  name: upgrade-plan-mgmt
spec:
  # Version retrieved from release notes
  releaseVersion: 3.X.Y
EOF
```

Nota

Em alguns casos de uso, talvez você queira fazer configurações adicionais no `UpgradePlan`. Para saber todas as configurações possíveis, consulte a *Seção 23.5.1, "UpgradePlan"*.

3. A implantação do `UpgradePlan` no namespace do `Controller de upgrade` inicia o `processo de upgrade`.

Nota

Para obter mais informações sobre o `processo de upgrade` real, consulte a *Seção 23.4, "Como funciona o Controller de upgrade?"*.

Para obter informações sobre como acompanhar o `processo de upgrade`, consulte a *Seção 23.6, "Acompanhando o processo de upgrade"*.

35.2 Fleet

Esta seção apresenta informações sobre como executar operações de "dia 2" usando o componente Fleet (*Capítulo 8, Fleet*).

Os seguintes tópicos são abordados como parte desta seção:

1. *Seção 35.2.1, "Componentes"*: componentes padrão usados para todas as operações de "dia 2".
2. *Seção 35.2.2, "Determinar seu caso de uso"*: apresenta uma visão geral dos recursos personalizados do Fleet que serão usados e como se adaptam aos diferentes casos de uso das operações de "dia 2".
3. *Seção 35.2.3, "Fluxo de trabalho de dia 2"*: fornece um guia de fluxo de trabalho para execução de operações de "dia 2" com o Fleet.
4. *Seção 35.2.4, "Upgrade de sistema operacional"*: descreve como fazer upgrades de sistema operacional com o Fleet.

5. *Seção 35.2.5, "Upgrade da versão do Kubernetes"*: descreve como fazer upgrades de versão do Kubernetes com o Fleet.

6. *Seção 35.2.6, "Upgrade do gráfico Helm"*: descreve como fazer upgrades de gráficos Helm com o Fleet.

35.2.1 Componentes

Veja a seguir uma descrição dos componentes padrão que você deve configurar no cluster de `gerenciamento` para realizar operações de "dia 2" com sucesso pelo Fleet.

35.2.1.1 Rancher

(**Opcional**) Responsável por gerenciar os `clusters downstream` e implantar o `System Upgrade Controller` no `cluster de gerenciamento`.

Para obter mais informações, consulte o *Capítulo 5, Rancher*.

35.2.1.2 System Upgrade Controller (SUC)

System Upgrade Controller é responsável por executar tarefas em nós especificados com base nos dados de configuração fornecidos por um recurso personalizado chamado `plano`.

O **SUC** é ativamente usado para fazer upgrade do sistema operacional e da distribuição Kubernetes.

Para obter mais informações sobre o componente **SUC** e como ele se adapta à pilha do Edge, consulte o *Capítulo 22, System Upgrade Controller*.

35.2.2 Determinar seu caso de uso

O Fleet usa dois tipos de recursos personalizados (https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/) para permitir o gerenciamento de recursos do Kubernetes e do Helm.

Veja a seguir as informações sobre a finalidade desses recursos e para quais casos de uso eles são mais adequados no contexto das operações de "dia 2".

35.2.2.1 GitRepo

O `GitRepo` é um recurso do Fleet (*Capítulo 8, Fleet*) que representa um repositório Git do qual o `Fleet` pode criar `Bundles`. Cada `Bundle` é criado com base nos caminhos de configuração definidos no recurso `GitRepo`. Para obter mais informações, consulte a documentação do GitRepo (https://fleet.rancher.io/gitrepo-add).

No contexto das operações de "dia 2", os recursos `GitRepo` costumam ser usados para implantar o `SUC` ou os `planos do SUC` em ambientes **não air-gapped** que usam a abordagem *GitOps do Fleet*.

Se preferir, use os recursos `GitRepo` para implantar o `SUC` ou os `planos do SUC` em ambientes **air-gapped**, **desde que você espelhe a configuração do seu repositório por um servidor Git local**.

35.2.2.2 Bundle

Os `Bundles` armazenam recursos **brutos** do Kubernetes que serão implantados no cluster de destino. Normalmente, eles são criados de um recurso `GitRepo`, mas há casos de uso em que podem ser implantados manualmente. Para obter mais informações, consulte a documentação do Bundle (https://fleet.rancher.io/bundle-add).

No contexto das operações de "dia 2", os recursos `Bundle` costumam ser usados para implantar o `SUC` ou `planos do SUC` em ambientes **air-gapped** que não usam nenhuma forma de procedimento do *GitOps local* (por exemplo, um **servidor git local**).

Se o seu caso de uso não possibilita um fluxo de trabalho *GitOps* (por exemplo, usando um repositório Git), a alternativa é usar os recursos `Bundle` para implantar o `SUC` ou os `planos do SUC` em ambientes **não air-gapped**.

35.2.3 Fluxo de trabalho de dia 2

Veja a seguir um fluxo de trabalho de "dia 2" que deve ser seguido para fazer upgrade de um cluster de gerenciamento para uma versão específica do Edge.

1. Upgrade de sistema operacional (*Seção 35.2.4, "Upgrade de sistema operacional"*)
2. Upgrade de versão do Kubernetes (*Seção 35.2.5, "Upgrade da versão do Kubernetes"*)
3. Upgrade de gráfico Helm (*Seção 35.2.6, "Upgrade do gráfico Helm"*)

35.2.4 Upgrade de sistema operacional

Esta seção descreve como fazer upgrade de um sistema operacional usando o *Capítulo 8, Fleet* e o *Capítulo 22, System Upgrade Controller*.

Os seguintes tópicos são abordados como parte desta seção:

1. *Seção 35.2.4.1, "Componentes"*: componentes adicionais usados pelo processo de upgrade.
2. *Seção 35.2.4.2, "Visão geral"*: visão geral do processo de upgrade.
3. *Seção 35.2.4.3, "Requisitos"*: requisitos do processo de upgrade.
4. *Seção 35.2.4.4, "Upgrade de sistema operacional: implantação do plano do SUC"*: informações de como implantar os `planos do SUC`, responsáveis por acionar o processo de upgrade.

35.2.4.1 Componentes

Esta seção aborda os componentes personalizados que o processo de `upgrade de sistema operacional` usa com os componentes de "dia 2" padrão (*Seção 35.2.1, "Componentes"*).

35.2.4.1.1 systemd.service

O upgrade de sistema operacional em um nó específico é executado pelo systemd.service (https://www.freedesktop.org/software/systemd/man/latest/systemd.service.html).

É criado um serviço diferente dependendo do tipo de upgrade que o sistema operacional requer de uma versão do Edge para outra:

- Para versões do Edge que requerem a mesma versão de sistema operacional (por exemplo, `6.0`), o `os-pkg-update.service` é criado. Ele usa o transactional-update (https://kubic.opensuse.org/documentation/man-pages/transactional-update.8.html) para fazer upgrade de um pacote normal (https://en.opensuse.org/SDB:Zypper_usage#Updating_packages).
- Para versões do Edge que requerem a migração da versão de sistema operacional (por exemplo, `6.0` → `6.1`), o `os-migration.service` é criado. Ele usa o transactional-update (https://kubic.opensuse.org/documentation/man-pages/transactional-update.8.html) para fazer o seguinte:
  a. O upgrade de pacote normal (https://en.opensuse.org/SDB:Zypper_usage#Updating_packages), que garante que todos os pacotes sejam atualizados para mitigar falhas na migração relacionadas a versões antigas dos pacotes.
  b. A migração de sistema operacional usando o comando `zypper migration`.

Os serviços mencionados acima são distribuídos para cada nó por meio de um `plano do SUC`, que deve estar localizado no cluster de gerenciamento que precisa do upgrade de sistema operacional.

35.2.4.2 Visão geral

O upgrade do sistema operacional para nós de cluster de gerenciamento é feito pelo `Fleet` e pelo `System Upgrade Controller (SUC)`.

O **Fleet** é usado para implantar e gerenciar os `planos do SUC` no cluster desejado.

Nota

Os `planos do SUC` são recursos personalizados (https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/) que descrevem as etapas que o `SUC` precisa seguir para execução de uma tarefa específica em um conjunto de nós. Para ver um exemplo de como é um `plano do SUC`, consulte o repositório upstream (https://github.com/rancher/system-upgrade-controller?tab=readme-ov-file#example-plans).

Os `planos do SUC de sistema operacional` são enviados a cada cluster por meio da implantação de um recurso GitRepo (https://fleet.rancher.io/gitrepo-add) ou Bundle (https://fleet.rancher.io/bundle-add) em um espaço de trabalho (https://fleet.rancher.io/namespaces#gitrepos-bundles-clusters-clustergroups) do Fleet específico. O Fleet recupera o `GitRepo/Bundle` implantado e implanta seu conteúdo (os `planos do SUC de sistema operacional`) no(s) cluster(s) desejado(s).

Nota

Os recursos `GitRepo/Bundle` sempre são implantados no `cluster de gerenciamento`. O uso do recurso `GitRepo` ou `Bundle` depende do seu caso de uso. Consulte a *Seção 35.2.2, "Determinar seu caso de uso"* para obter mais informações.

Os `planos do SUC de sistema operacional` descrevem o seguinte fluxo de trabalho:

1. Sempre use o comando cordon (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_cordon/) nos nós antes dos upgrades de sistema operacional.
2. Sempre faça upgrade dos nós do `plano de controle` antes dos nós do `worker`.
3. Sempre faça upgrade do cluster **um** nó de cada vez.

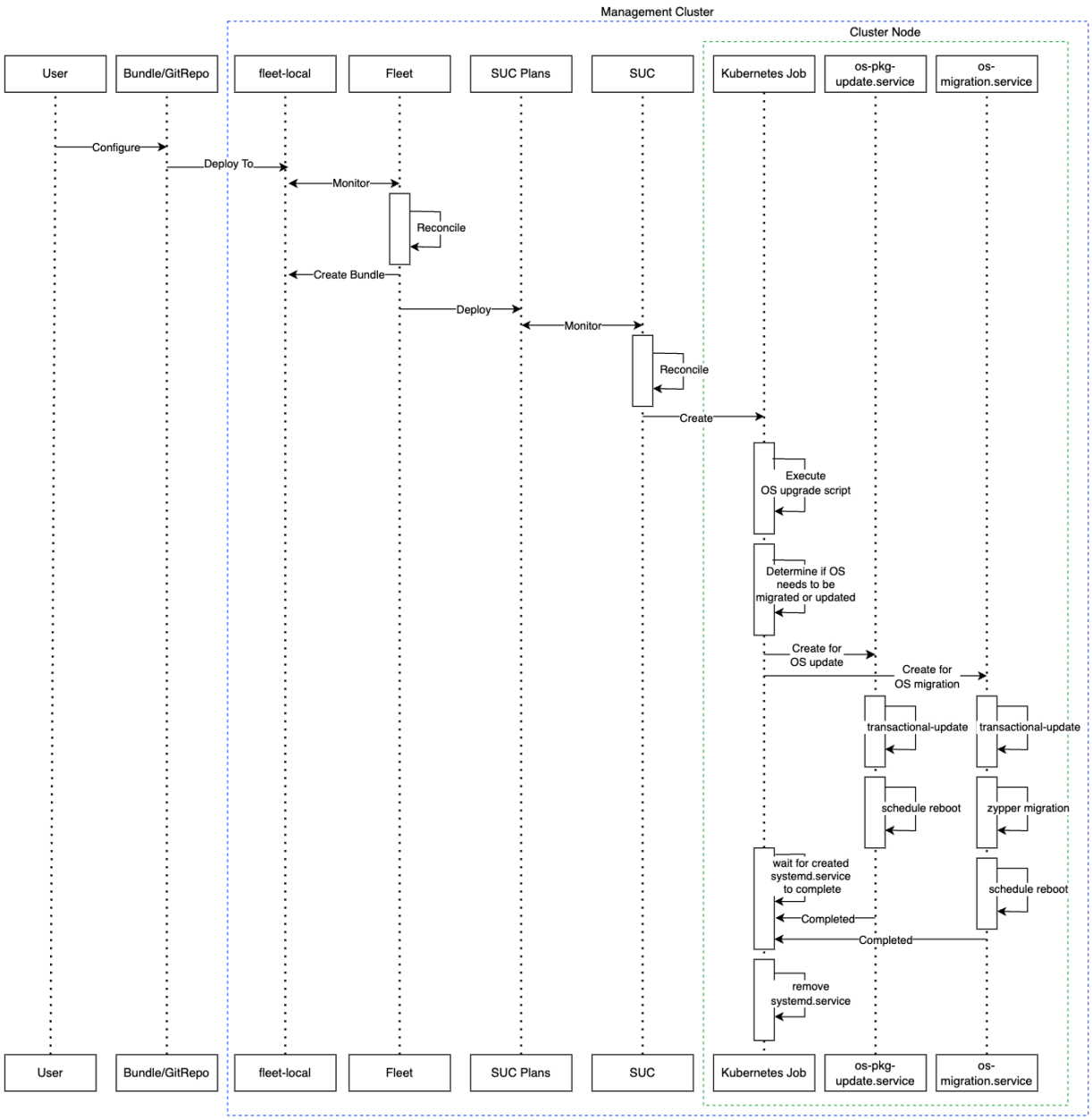
Depois que os `planos do SUC de sistema operacional` forem implantados, o fluxo de trabalho terá esta aparência:

1. O SUC reconcilia os `planos do SUC de sistema operacional` implantados e cria um `Kubernetes Job` em **cada nó**.
2. O `Kubernetes Job` cria um systemd.service (*Seção 35.2.4.1.1, "systemd.service"*) para upgrade de pacote ou migração de sistema operacional.
3. O `systemd.service` criado aciona o processo de upgrade de sistema operacional no nó específico.

Importante

Após o término do processo de upgrade do sistema operacional, o nó correspondente será `reinicializado` para aplicar as atualizações ao sistema.

Veja a seguir um diagrama da descrição acima:



35.2.4.3 Requisitos

Geral:

1. **Máquina registrada no SCC**: todos os nós do cluster de gerenciamento devem ser registrados no `https://scc.suse.com/` para que o respectivo `systemd.service` possa se conectar com sucesso ao repositório RPM desejado.

   > **Importante**
   >
   > Para lançamentos do Edge que exigem a migração da versão do sistema operacional (por exemplo, `6.0` → `6.1`), verifique se a chave do SCC oferece suporte à migração para a nova versão.

2. **Garantir que as tolerâncias do plano do SUC correspondam às do nó**: se os nós do cluster Kubernetes têm **taints** personalizados, adicione tolerâncias (https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/) a esses taints nos **planos do SUC**. Por padrão, os **planos do SUC** têm tolerâncias apenas para nós do **plano de controle**. As tolerâncias padrão incluem:

   - *CriticalAddonsOnly=true:NoExecute*
   - *node-role.kubernetes.io/control-plane:NoSchedule*
   - *node-role.kubernetes.io/etcd:NoExecute*

   > **Nota**
   >
   > As tolerâncias adicionais devem ser incluídas na seção `.spec.tolerations` de cada plano. Os **planos do SUC** relacionados a upgrade de sistema operacional estão disponíveis no repositório suse-edge/fleet-examples (https://github.com/suse-edge/fleet-examples) em `fleets/day2/system-upgrade-controller-plans/os-upgrade`. **Use os planos de uma tag de versão (https://github.com/suse-edge/fleet-examples/releases) válida do repositório.**
   >
   > Um exemplo de definição de tolerâncias personalizadas para o plano do SUC de **plano de controle** tem esta aparência:

   ```
   apiVersion: upgrade.cattle.io/v1
   kind: Plan
   ```

```
metadata:
  name: os-upgrade-control-plane
spec:
  ...
  tolerations:
  # default tolerations
  - key: "CriticalAddonsOnly"
    operator: "Equal"
    value: "true"
    effect: "NoExecute"
  - key: "node-role.kubernetes.io/control-plane"
    operator: "Equal"
    effect: "NoSchedule"
  - key: "node-role.kubernetes.io/etcd"
    operator: "Equal"
    effect: "NoExecute"
  # custom toleration
  - key: "foo"
    operator: "Equal"
    value: "bar"
    effect: "NoSchedule"
...
```

Air-gapped:

1. **Espelhar repositórios RPM do SUSE**: os repositórios RPM de sistema operacional devem ser espelhados localmente para que o `systemd.service` tenha acesso a eles. Para isso, use RMT (https://documentation.suse.com/sles/15-SP6/html/SLES-all/book-rmt.html) ou SUMA (https://documentation.suse.com/suma/5.0/en/suse-manager/index.html).

35.2.4.4 Upgrade de sistema operacional: implantação do plano do SUC

Importante

Em ambientes que já passaram por upgrade usando esse procedimento, os usuários devem garantir que **uma** das seguintes etapas seja concluída:

- `Remover os planos do SUC que já foram implantados relacionados a versões de lançamento mais antigas do Edge do cluster de gerenciamento`: para fazer isso, remova o cluster desejado da configuração de destino (https://fleet.rancher.io/gitrepo-targets#target-matching) do `GitRepo/Bundle`, ou remova o recurso `GitRepo/Bundle` completamente.
- `Reutilizar o recurso GitRepo/Bundle existente`: para fazer isso, aponte a revisão do recurso para uma nova tag que inclua as instâncias do Fleet corretas para a versão (https://github.com/suse-edge/fleet-examples/releases) desejada de `suse-edge/fleet-examples`.

Isso é feito para evitar conflitos entre os `planos do SUC` de versões de lançamento mais antigas do Edge.

Se os usuários tentarem fazer upgrade e já houver `planos do SUC` no cluster de gerenciamento, eles verão o seguinte erro no Fleet:

```
Not installed: Unable to continue with install: Plan <plan_name> in namespace
 <plan_namespace> exists and cannot be imported into the current release: invalid
 ownership metadata; annotation validation error..
```

Conforme mencionado na *Seção 35.2.4.2, "Visão geral"*, os upgrades de sistema operacional são feitos enviando os `planos do SUC` ao cluster desejado de uma destas maneiras:

- Recurso `GitRepo` do Fleet: *Seção 35.2.4.4.1, "Implantação do plano do SUC: recurso GitRepo"*.
- Recurso `Bundle` do Fleet: *Seção 35.2.4.4.2, "Implantação do plano do SUC – recurso Bundle"*.

Para determinar o recurso que você deve usar, consulte a *Seção 35.2.2, "Determinar seu caso de uso"*.

Para casos de uso de implantação dos `planos do SUC de sistema operacional` de uma ferramenta GitOps de terceiros, consulte a *Seção 35.2.4.4.3, "Implantação do plano do SUC: fluxo de trabalho do GitOps de terceiros"*.

35.2.4.4.1 Implantação do plano do SUC: recurso GitRepo

Um recurso **GitRepo**, que distribui os `planos do SUC de sistema operacional` necessários, pode ser implantado de uma das seguintes maneiras:

1. Pela `IU do Rancher`: *Seção 35.2.4.4.1.1, "Criação do GitRepo: IU do Rancher"* (quando o `Rancher` está disponível).
2. Pela implantação manual do recurso (*Seção 35.2.4.4.1.2, "Criação do GitRepo: manual"*) em seu `cluster de gerenciamento`.

Após a implantação, para monitorar o processo de upgrade do sistema operacional dos nós do cluster de destino, consulte a *Seção 22.3, "Monitorando os planos do System Upgrade Controller"*.

35.2.4.4.1.1 Criação do GitRepo: IU do Rancher

Para criar um recurso `GitRepo` pela IU do Rancher, siga a documentação (https://ranchermanager.docs.rancher.com/v2.11/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui) oficial dele.

A equipe do Edge mantém uma instância do Fleet (https://github.com/suse-edge/fleet-examples/tree/release-3.3.0/fleets/day2/system-upgrade-controller-plans/os-upgrade) pronta para uso. Dependendo do seu ambiente, ela pode ser usada diretamente ou como gabarito.

> **Importante**
>
> Use sempre essa instância do Fleet de uma tag de versão (https://github.com/suse-edge/fleet-examples/releases) válida do Edge.

Para casos de uso em que não há necessidade de incluir alterações personalizadas nos `planos do SUC` que a instância do Fleet envia, os usuários podem fazer referência à instância do Fleet de `os-upgrade` do repositório `suse-edge/fleet-examples`.

Nos casos em que as alterações personalizadas são necessárias (por exemplo, para adicionar tolerâncias personalizadas), os usuários devem fazer referência à instância do Fleet de `os-upgrade` de um repositório separado, para que possam adicioná-las aos planos do SUC conforme necessário.

Há um exemplo de como configurar o `GitRepo` para usar a instância do Fleet de um repositório `suse-edge/fleet-examples` disponível aqui (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/gitrepos/day2/os-upgrade-gitrepo.yaml).

35.2.4.4.1.2 Criação do GitRepo: manual

1. Obtenha o recurso **GitRepo**:

```
curl -o os-upgrade-gitrepo.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.3.0/gitrepos/day2/os-upgrade-gitrepo.yaml
```

2. Edite a configuração do **GitRepo**:

   - Remova a seção `spec.targets` (necessário apenas para clusters downstream).

```
# Example using sed
sed -i.bak '/^  targets:/,$d' os-upgrade-gitrepo.yaml && rm -f os-upgrade-gitrepo.yaml.bak

# Example using yq (v4+)
yq eval 'del(.spec.targets)' -i os-upgrade-gitrepo.yaml
```

   - Aponte o namespace do `GitRepo` para o namespace `fleet-local`, feito para implantar o recurso no cluster de gerenciamento.

```
# Example using sed
sed -i.bak 's/namespace: fleet-default/namespace: fleet-local/' os-upgrade-gitrepo.yaml && rm -f os-upgrade-gitrepo.yaml.bak

# Example using yq (v4+)
yq eval '.metadata.namespace = "fleet-local"' -i os-upgrade-gitrepo.yaml
```

3. Aplique o recurso **GitRepo** ao `cluster de gerenciamento`:

```
kubectl apply -f os-upgrade-gitrepo.yaml
```

4. Visualize o recurso **GitRepo** criado no namespace `fleet-local`:

```
kubectl get gitrepo os-upgrade -n fleet-local

# Example output
NAME            REPO                                                COMMIT
 BUNDLEDEPLOYMENTS-READY   STATUS
os-upgrade      https://github.com/suse-edge/fleet-examples.git   release-3.3.0  0/0
```

35.2.4.4.2 Implantação do plano do SUC – recurso Bundle

O recurso **Bundle**, que envia os `planos do SUC de sistema operacional` necessários, pode ser implantado de uma das seguintes maneiras:

1. Pela `IU do Rancher` : *Seção 35.2.4.4.2.1, "Criação do bundle – IU do Rancher"* (quando o `Rancher` está disponível).
2. Por meio da implantação manual do recurso (*Seção 35.2.4.4.2.2, "Criação do bundle – manual"*) no `cluster de gerenciamento`.

Após a implantação, para monitorar o processo de upgrade do sistema operacional dos nós do cluster de destino, consulte a *Seção 22.3, "Monitorando os planos do System Upgrade Controller"*.

35.2.4.4.2.1 Criação do bundle – IU do Rancher

A equipe do Edge mantém um bundle (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/bundles/day2/system-upgrade-controller-plans/os-upgrade/os-upgrade-bundle.yaml) pronto para uso que pode ser usado nas etapas a seguir.

Importante

Sempre use esse bundle de uma tag de versão (https://github.com/suse-edge/fleet-examples/releases) válida do Edge.

Para criar um bundle pela IU do Rancher:

1. No canto superior esquerdo, clique em # → **Continuous Delivery** (Entrega contínua).
2. Vá para **Advanced** > **Bundles** (Avançado > Bundles).
3. Selecione **Create from YAML** (Criar do YAML).
4. Nesse local, você pode criar o bundle de uma das seguintes maneiras:

Nota

Há casos de uso em que você precisa incluir alterações personalizadas nos `planos do SUC` que o bundle envia (por exemplo, para adicionar tolerâncias personalizadas). Inclua essas alterações no bundle que será gerado pelas etapas a seguir.

   a. Copie manualmente o conteúdo do bundle (https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.3.0/bundles/day2/system-upgrade-controller-plans/os-upgrade/os-upgrade-bundle.yaml) de `suse-edge/fleet-examples` para a página **Create from YAML** (Criar do YAML).

   b. Clone o repositório suse-edge/fleet-examples (https://github.com/suse-edge/fleet-examples) da tag da versão (https://github.com/suse-edge/fleet-examples/releases) desejada e selecione a opção **Read from File** (Ler arquivo) na página **Create from YAML** (Criar do YAML). Dessa página, navegue até o local do bundle (`bundles/day2/system-upgrade-controller-plans/os-upgrade`) e selecione o arquivo do bundle. Esse procedimento preencherá automaticamente a página **Create from YAML** (Criar do YAML) com o conteúdo do bundle.

5. Edite o bundle na IU do Rancher:

   - Altere o **namespace** do `Bundle` para apontar para o namespace `fleet-local`.

```
# Example
kind: Bundle
apiVersion: fleet.cattle.io/v1alpha1
metadata:
  name: os-upgrade
  namespace: fleet-local
```

```
...
```

- Altere os clusters de **destino** para que o `Bundle` aponte para o seu cluster `local` (gerenciamento):

```
spec:
  targets:
  - clusterName: local
```

Nota

Em alguns casos de uso, o cluster `local` pode ter um nome diferente.

Para recuperar o nome do cluster `local`, execute o comando abaixo:

```
kubectl get clusters.fleet.cattle.io -n fleet-local
```

6. Selecione **Create** (Criar).

35.2.4.4.2.2 Criação do bundle – manual

1. Obtenha o recurso **Bundle**:

```
curl -o os-upgrade-bundle.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.3.0/bundles/day2/system-upgrade-controller-plans/os-upgrade/os-upgrade-bundle.yaml
```

2. Edite a configuração do `Bundle`:

- Altere os clusters de **destino** para que o `Bundle` aponte para o seu cluster `local` (gerenciamento):

```
spec:
  targets:
  - clusterName: local
```

Nota

Em alguns casos de uso, o cluster `local` pode ter um nome diferente.

Para recuperar o nome do cluster `local`, execute o comando abaixo:

```
kubectl get clusters.fleet.cattle.io -n fleet-local
```

- Altere o **namespace** do `Bundle` para apontar para o namespace `fleet-local`.

```
# Example
kind: Bundle
apiVersion: fleet.cattle.io/v1alpha1
metadata:
  name: os-upgrade
  namespace: fleet-local
...
```

3. Aplique o recurso **Bundle** ao seu `cluster de gerenciamento`:

```
kubectl apply -f os-upgrade-bundle.yaml
```

4. Visualize o recurso **Bundle** criado no namespace `fleet-local`:

```
kubectl get bundles -n fleet-local
```

35.2.4.4.3 Implantação do plano do SUC: fluxo de trabalho do GitOps de terceiros

Em alguns casos de uso, talvez você queira incorporar os `planos do SUC de sistema operacional` ao fluxo de trabalho do GitOps de terceiros (por exemplo, o `Flux`).

Para obter os recursos de upgrade de sistema operacional que você precisa, primeiro determine a tag da versão (https://github.com/suse-edge/fleet-examples/releases) do Edge do repositório suse-edge/fleet-examples (https://github.com/suse-edge/fleet-examples) que deseja usar.

Depois disso, os recursos estarão disponíveis em `fleets/day2/system-upgrade-controller-plans/os-upgrade`, em que:

- `plan-control-plane.yaml` é um recurso do plano do SUC para nós do **plano de controle**.
- `plan-worker.yaml` é um recurso do plano do SUC para nós do **worker**.
- `secret.yaml` é um segredo que contém o script `upgrade.sh`, responsável por criar o systemd.service (*Seção 35.2.4.1.1, "systemd.service"*).
- `config-map.yaml` é um ConfigMap que armazena as configurações consumidas pelo script `upgrade.sh`.

Importante

Esses recursos do `plano` são interpretados pelo `System Upgrade Controller` e devem ser implantados em cada cluster downstream do qual você deseja fazer upgrade. Para obter informações sobre a implantação do SUC, consulte a *Seção 22.2, "Instalando o System Upgrade Controller"*.

Para entender melhor como usar o fluxo de trabalho do GitOps para implantar os **planos do SUC** para upgrade de sistema operacional, consulte a visão geral (*Seção 35.2.4.2, "Visão geral"*).

35.2.5 Upgrade da versão do Kubernetes

Esta seção descreve como fazer um upgrade do Kubernetes usando o *Capítulo 8, Fleet* e o *Capítulo 22, System Upgrade Controller*.

Os seguintes tópicos são abordados como parte desta seção:

1. *Seção 35.2.5.1, "Componentes"*: componentes adicionais usados pelo processo de upgrade.
2. *Seção 35.2.5.2, "Visão geral"*: visão geral do processo de upgrade.
3. *Seção 35.2.5.3, "Requisitos"*: requisitos do processo de upgrade.
4. *Seção 35.2.5.4, "Upgrade do K8s: implantação do plano do SUC"*: informações de como implantar os `planos do SUC`, responsáveis por acionar o processo de upgrade.

35.2.5.1 Componentes

Esta seção aborda os componentes personalizados que o processo de `upgrade do K8s` usa com os componentes de "dia 2" padrão (*Seção 35.2.1, "Componentes"*).

35.2.5.1.1 rke2-upgrade

Imagem do contêiner responsável por fazer upgrade da versão do RKE2 de um nó específico.

Incluída em um pod criado pelo **SUC** com base no **plano do SUC**. O plano deve estar localizado em cada **cluster** que precisa de upgrade do RKE2.

Para obter mais informações sobre como a imagem `rke2-upgrade` faz o upgrade, consulte a documentação upstream (https://github.com/rancher/rke2-upgrade/tree/master).

35.2.5.1.2 k3s-upgrade

Imagem do contêiner responsável por fazer upgrade da versão do K3s de um nó específico.

Incluída em um pod criado pelo **SUC** com base no **plano do SUC**. O plano deve estar localizado em cada **cluster** que precisa de upgrade do K3s.

Para obter mais informações sobre como a imagem `k3s-upgrade` faz o upgrade, consulte a documentação upstream (https://github.com/k3s-io/k3s-upgrade).

35.2.5.2 Visão geral

O upgrade de distribuições Kubernetes para nós de cluster de gerenciamento é feito pelo `Fleet` e pelo `System Upgrade Controller (SUC)`.

O `Fleet` é usado para implantar e gerenciar `planos do SUC` no cluster desejado.

Nota

Os `planos do SUC` são recursos personalizados (https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/) que descrevem as etapas que o **SUC** precisa seguir para execução de uma tarefa específica em um conjunto de nós. Para ver um exemplo de como é um `plano do SUC`, consulte o repositório upstream (https://github.com/rancher/system-upgrade-controller?tab=readme-ov-file#example-plans).

Os `planos do SUC do K8s` são enviados a cada cluster por meio da implantação de um recurso GitRepo (https://fleet.rancher.io/gitrepo-add) ou Bundle (https://fleet.rancher.io/bundle-add) em um espaço de trabalho (https://fleet.rancher.io/namespaces#gitrepos-bundles-clusters-clustergroups) do Fleet específico. O Fleet recupera o `GitRepo/Bundle` implantado e implanta seu conteúdo (os `planos do SUC do K8s`) no(s) cluster(s) desejado(s).

Nota

Os recursos `GitRepo/Bundle` sempre são implantados no `cluster de gerenciamento`. O uso do recurso `GitRepo` ou `Bundle` depende do seu caso de uso. Consulte a *Seção 35.2.2, "Determinar seu caso de uso"* para obter mais informações.

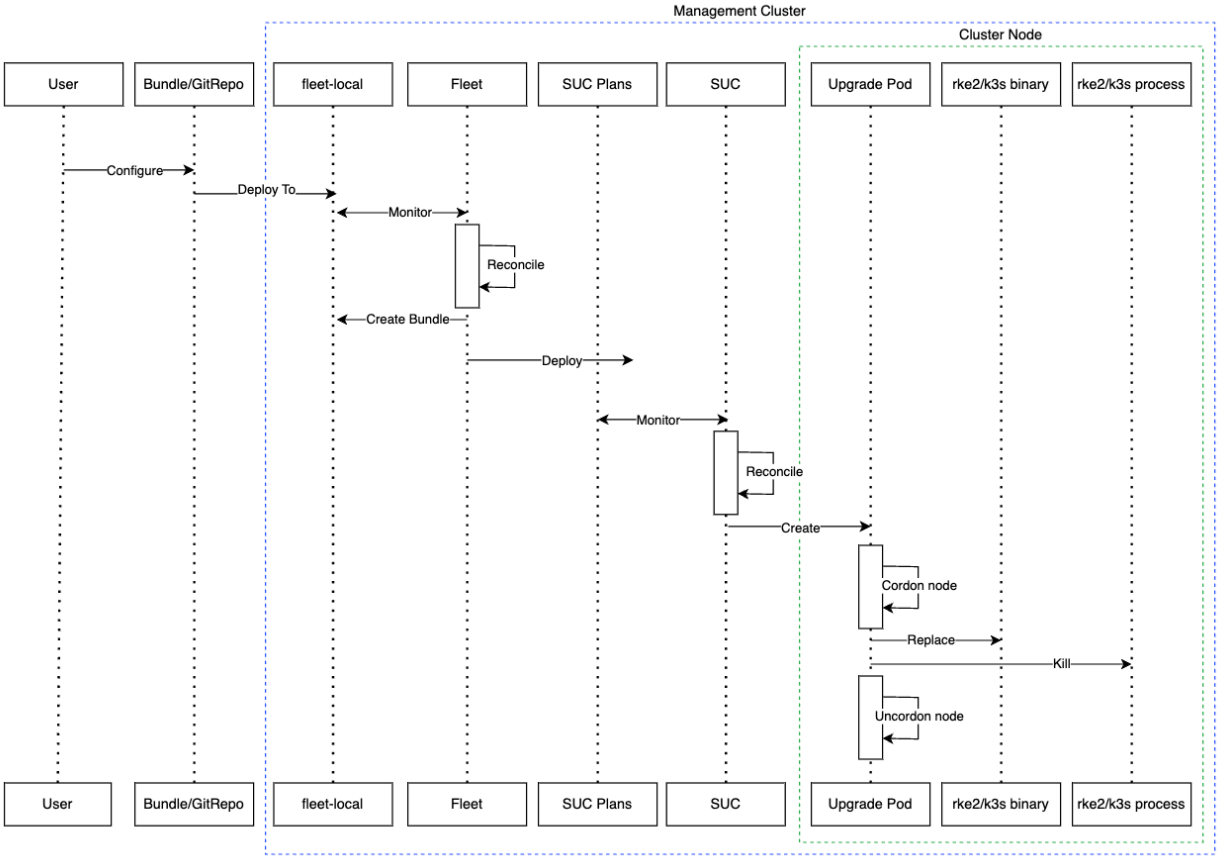
Os `planos do SUC do K8s` descrevem o seguinte fluxo de trabalho:

1. Sempre use o comando cordon (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_cordon/) nos nós antes dos upgrades do K8s.
2. Sempre faça upgrade dos nós do `plano de controle` antes dos nós do `worker`.
3. Sempre faça upgrade dos nós do `plano de controle` **um** de cada vez, e dos nós do `worker` **dois** nós de cada vez.

Depois que os `planos do SUC do K8s` forem implantados, o fluxo de trabalho terá esta aparência:

1. O SUC reconcilia os `planos do SUC do K8s` implantados e cria um `Kubernetes Job` em **cada nó**.
2. Dependendo da distribuição Kubernetes, o job cria um pod que executa a imagem de contêiner do rke2-upgrade (*Seção 35.2.5.1.1, "rke2-upgrade"*) ou do k3s-upgrade (*Seção 35.2.5.1.2, "k3s-upgrade"*).
3. O pod criado vai percorrer o seguinte fluxo de trabalho:
    a. Substitua o binário `rke2/k3s` existente no nó pelo da imagem `rke2-upgrade/k3s-upgrade`.
    b. Cancele o processo do `rke2/k3s` em execução.
4. O cancelamento do processo do `rke2/k3s` aciona a reinicialização, o que inicia um novo processo que executa o binário atualizado, resultando em uma versão atualizada da distribuição Kubernetes.

Veja a seguir um diagrama da descrição acima:



35.2.5.3 Requisitos

1. **Faça backup da distribuição Kubernetes:**

    a. Para **clusters RKE2**, consulte a documentação sobre backup e restauração do RKE2 (https://docs.rke2.io/datastore/backup_restore).

    b. Para **clusters K3s**, consulte a documentação sobre backup e restauração do K3s (https://docs.k3s.io/datastore/backup-restore).

2. **Garantir que as tolerâncias do plano do SUC correspondam às do nó**: se os nós do cluster Kubernetes têm **taints** personalizados, adicione tolerâncias (https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/) a esses taints nos **planos do SUC**. Por padrão, os **planos do SUC** têm tolerâncias apenas para nós do **plano de controle**. As tolerâncias padrão incluem:

    - *CriticalAddonsOnly = true:NoExecute*
    - *node-role.kubernetes.io/control-plane:NoSchedule*
    - *node-role.kubernetes.io/etcd:NoExecute*

    Nota

    As tolerâncias adicionais devem ser incluídas na seção `.spec.tolerations` de cada plano. Os **planos do SUC** relacionados ao upgrade de versões do Kubernetes estão disponíveis no repositório suse-edge/fleet-examples (https://github.com/suse-edge/fleet-examples) em:

    - Para o **RKE2**: `fleets/day2/system-upgrade-controller-plans/rke2-upgrade`
    - Para o **K3s**: `fleets/day2/system-upgrade-controller-plans/k3s-upgrade`

    Use os planos de uma tag de versão (https://github.com/suse-edge/fleet-examples/releases) válida do repositório.

    Um exemplo de definição de tolerâncias personalizadas para o plano do SUC de **plano de controle** do RKE2 tem esta aparência:

    ```
    apiVersion: upgrade.cattle.io/v1
    ```

```
kind: Plan
metadata:
  name: rke2-upgrade-control-plane
spec:
  ...
  tolerations:
  # default tolerations
  - key: "CriticalAddonsOnly"
    operator: "Equal"
    value: "true"
    effect: "NoExecute"
  - key: "node-role.kubernetes.io/control-plane"
    operator: "Equal"
    effect: "NoSchedule"
  - key: "node-role.kubernetes.io/etcd"
    operator: "Equal"
    effect: "NoExecute"
  # custom toleration
  - key: "foo"
    operator: "Equal"
    value: "bar"
    effect: "NoSchedule"
...
```

35.2.5.4 Upgrade do K8s: implantação do plano do SUC

Importante

Em ambientes que já passaram por upgrade usando esse procedimento, os usuários devem garantir que **uma** das seguintes etapas seja concluída:

- `Remover os planos do SUC que já foram implantados relacionados a versões de lançamento mais antigas do Edge do cluster de gerenciamento`: para fazer isso, remova o cluster desejado da configuração de destino (https://fleet.rancher.io/gitrepo-targets#target-matching) do `GitRepo/Bundle`, ou remova o recurso `GitRepo/Bundle` completamente.
- `Reutilizar o recurso GitRepo/Bundle existente`: para fazer isso, aponte a revisão do recurso para uma nova tag que inclua as instâncias do Fleet corretas para a versão (https://github.com/suse-edge/fleet-examples/releases) desejada de `suse-edge/fleet-examples`.

Isso é feito para evitar conflitos entre os `planos do SUC` de versões de lançamento mais antigas do Edge.

Se os usuários tentarem fazer upgrade e já houver `planos do SUC` no cluster de gerenciamento, eles verão o seguinte erro no Fleet:

```
Not installed: Unable to continue with install: Plan <plan_name> in namespace
 <plan_namespace> exists and cannot be imported into the current release: invalid
 ownership metadata; annotation validation error..
```

Conforme mencionado na *Seção 35.2.5.2, "Visão geral"*, os upgrades do Kubernetes são feitos enviando os `planos do SUC` para o cluster desejado de uma destas maneiras:

- Recurso GitRepo do Fleet (*Seção 35.2.5.4.1, "Implantação do plano do SUC: recurso GitRepo"*)
- Recurso Bundle do Fleet (*Seção 35.2.5.4.2, "Implantação do plano do SUC – recurso Bundle"*)

Para determinar o recurso que você deve usar, consulte a *Seção 35.2.2, "Determinar seu caso de uso"*.

Para casos de uso de implantação dos `planos do SUC do K8s` de uma ferramenta GitOps de terceiros, consulte a *Seção 35.2.5.4.3, "Implantação do plano do SUC: fluxo de trabalho do GitOps de terceiros"*.

35.2.5.4.1 Implantação do plano do SUC: recurso GitRepo

Um recurso **GitRepo**, que distribui os `planos do SUC do K8s` necessários, pode ser implantado de uma das seguintes maneiras:

1. Pela `IU do Rancher`: *Seção 35.2.5.4.1.1, "Criação do GitRepo: IU do Rancher"* (quando o `Rancher` está disponível).
2. Pela implantação manual do recurso (*Seção 35.2.5.4.1.2, "Criação do GitRepo: manual"*) no `cluster de gerenciamento`.

Após a implantação, para monitorar o processo de upgrade do Kubernetes dos nós do cluster de destino, consulte a *Seção 22.3, "Monitorando os planos do System Upgrade Controller"*.

35.2.5.4.1.1 Criação do GitRepo: IU do Rancher

Para criar um recurso `GitRepo` pela IU do Rancher, siga a documentação (https://ranchermanager.docs.rancher.com/v2.11/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui) oficial dele.

A equipe do Edge mantém instâncias do Fleet prontas para uso para distribuições Kubernetes tanto do rke2 (https://github.com/suse-edge/fleet-examples/tree/release-3.3.0/fleets/day2/system-upgrade-controller-plans/rke2-upgrade) quanto do k3s (https://github.com/suse-edge/fleet-examples/tree/release-3.3.0/fleets/day2/system-upgrade-controller-plans/k3s-upgrade). Dependendo do ambiente, a instância do Fleet pode ser usada diretamente ou como gabarito.

Importante

Sempre use as instâncias do Fleet de uma tag de versão (https://github.com/suse-edge/fleet-examples/releases) válida do Edge.

Para casos de uso em que não há necessidade de incluir alterações personalizadas nos `planos do SUC` que as instâncias do Fleet envia, os usuários podem fazer referência a essas instâncias do Fleet diretamente do repositório `suse-edge/fleet-examples`.

Nos casos em que as alterações personalizadas são necessárias (por exemplo, para adicionar tolerâncias personalizadas), os usuários devem fazer referência às instâncias do Fleet de um repositório separado, para que possam adicionas as alterações aos planos do SUC conforme necessário.

Exemplos de configuração do recurso `GitRepo` usando as instâncias do Fleet do repositório `suse-edge/fleet-examples`:

- RKE2 (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/gitrepos/day2/rke2-upgrade-gitrepo.yaml)
- K3s (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/gitrepos/day2/k3s-upgrade-gitrepo.yaml)

35.2.5.4.1.2 Criação do GitRepo: manual

1. Obtenha o recurso **GitRepo**:

    - Para clusters **RKE2**:

    ```
    curl -o rke2-upgrade-gitrepo.yaml https://raw.githubusercontent.com/suse-edge/
    fleet-examples/refs/tags/release-3.3.0/gitrepos/day2/rke2-upgrade-gitrepo.yaml
    ```

    - Para clusters **K3s**:

    ```
    curl -o k3s-upgrade-gitrepo.yaml https://raw.githubusercontent.com/suse-edge/
    fleet-examples/refs/tags/release-3.3.0/gitrepos/day2/k3s-upgrade-gitrepo.yaml
    ```

2. Edite a configuração do **GitRepo**:

    - Remova a seção `spec.targets` (necessário apenas para clusters downstream).

        - Para RKE2:

        ```
        # Example using sed
        sed -i.bak '/^  targets:/,$d' rke2-upgrade-gitrepo.yaml && rm -f rke2-
        upgrade-gitrepo.yaml.bak

        # Example using yq (v4+)
        yq eval 'del(.spec.targets)' -i rke2-upgrade-gitrepo.yaml
        ```

        - Para K3s:

        ```
        # Example using sed
        sed -i.bak '/^  targets:/,$d' k3s-upgrade-gitrepo.yaml && rm -f k3s-
        upgrade-gitrepo.yaml.bak

        # Example using yq (v4+)
        yq eval 'del(.spec.targets)' -i k3s-upgrade-gitrepo.yaml
        ```

    - Aponte o namespace do `GitRepo` para o namespace `fleet-local`, feito para implantar o recurso no cluster de gerenciamento.

        - Para RKE2:

        ```
        # Example using sed
        sed -i.bak 's/namespace: fleet-default/namespace: fleet-local/' rke2-
        upgrade-gitrepo.yaml && rm -f rke2-upgrade-gitrepo.yaml.bak
        ```

```
# Example using yq (v4+)
yq eval '.metadata.namespace = "fleet-local"' -i rke2-upgrade-
gitrepo.yaml
```

- Para K3s:

```
# Example using sed
sed -i.bak 's/namespace: fleet-default/namespace: fleet-local/' k3s-
upgrade-gitrepo.yaml && rm -f k3s-upgrade-gitrepo.yaml.bak

# Example using yq (v4+)
yq eval '.metadata.namespace = "fleet-local"' -i k3s-upgrade-gitrepo.yaml
```

3. Aplique os recursos **GitRepo** ao `cluster de gerenciamento`:

```
# RKE2
kubectl apply -f rke2-upgrade-gitrepo.yaml

# K3s
kubectl apply -f k3s-upgrade-gitrepo.yaml
```

4. Visualize o recurso **GitRepo** criado no namespace `fleet-local`:

```
# RKE2
kubectl get gitrepo rke2-upgrade -n fleet-local

# K3s
kubectl get gitrepo k3s-upgrade -n fleet-local

# Example output
NAME           REPO                                                COMMIT
 BUNDLEDEPLOYMENTS-READY   STATUS
k3s-upgrade    https://github.com/suse-edge/fleet-examples.git   fleet-local   0/0
rke2-upgrade   https://github.com/suse-edge/fleet-examples.git   fleet-local   0/0
```

35.2.5.4.2 Implantação do plano do SUC – recurso Bundle

O recurso **Bundle**, que envia os `planos do SUC de upgrade do Kubernetes` necessários, pode ser implantado de uma das seguintes maneiras:

1. Pela `IU do Rancher`: *Seção 35.2.5.4.2.1, "Criação do bundle – IU do Rancher"* (quando o `Rancher` está disponível).
2. Pela implantação manual do recurso (*Seção 35.2.5.4.2.2, "Criação do bundle – manual"*) no `cluster de gerenciamento`.

Após a implantação, para monitorar o processo de upgrade do Kubernetes dos nós do cluster de destino, consulte a *Seção 22.3, "Monitorando os planos do System Upgrade Controller"*.

35.2.5.4.2.1 Criação do bundle – IU do Rancher

A equipe do Edge mantém bundles prontos para uso para distribuições Kubernetes tanto do rke2 (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/bundles/day2/system-upgrade-controller-plans/rke2-upgrade/plan-bundle.yaml) quanto do k3s (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/bundles/day2/system-upgrade-controller-plans/k3s-upgrade/plan-bundle.yaml). Dependendo do ambiente, esses bundles podem ser usados diretamente ou como gabaritos.

Importante

Sempre use esse bundle de uma tag de versão (https://github.com/suse-edge/fleet-examples/releases) válida do Edge.

Para criar um bundle pela IU do Rancher:

1. No canto superior esquerdo, clique em # → **Continuous Delivery** (Entrega contínua).
2. Vá para **Advanced** > **Bundles** (Avançado > Bundles).
3. Selecione **Create from YAML** (Criar do YAML).
4. Nesse local, você pode criar o bundle de uma das seguintes maneiras:

Nota

Há casos de uso em que você precisa incluir alterações personalizadas nos `planos do SUC` que o bundle envia (por exemplo, para adicionar tolerâncias personalizadas). Inclua essas alterações no bundle que será gerado pelas etapas a seguir.

a. Copie manualmente o conteúdo do bundle referente ao RKE2 (https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.3.0/bundles/day2/system-upgrade-controller-plans/rke2-upgrade/plan-bundle.yaml) ou ao K3s (https://raw.githubusercontent.com/suse-edge/fleet-

examples/refs/tags/release-3.3.0/bundles/day2/system-upgrade-controller-plans/k3s-upgrade/plan-bundle.yaml) de `suse-edge/fleet-examples` para a página **Create from YAML** (Criar do YAML).

b. Clone o repositório suse-edge/fleet-examples (https://github.com/suse-edge/fleet-examples.git) da tag da versão (https://github.com/suse-edge/fleet-examples/releases) desejada e selecione a opção **Read from File** (Ler arquivo) na página **Create from YAML** (Criar do YAML). Dessa página, navegue até o bundle necessário (`bundles/day2/system-upgrade-controller-plans/rke2-upgrade/plan-bundle.yaml` para RKE2 e `bundles/day2/system-upgrade-controller-plans/k3s-upgrade/plan-bundle.yaml` para K3s). Esse procedimento preencherá automaticamente a página **Create from YAML** (Criar do YAML) com o conteúdo do bundle.

5. Edite o bundle na IU do Rancher:

   - Altere o **namespace** do `Bundle` para apontar para o namespace `fleet-local`.

```
# Example
kind: Bundle
apiVersion: fleet.cattle.io/v1alpha1
metadata:
  name: rke2-upgrade
  namespace: fleet-local
...
```

   - Altere os clusters de **destino** para que o `Bundle` aponte para o seu cluster `local` (gerenciamento):

```
spec:
  targets:
  - clusterName: local
```

Nota

Em alguns casos de uso, o cluster `local` pode ter um nome diferente.

Para recuperar o nome do cluster `local`, execute o comando abaixo:

```
kubectl get clusters.fleet.cattle.io -n fleet-local
```

6. Selecione **Create** (Criar).

35.2.5.4.2.2 Criação do bundle – manual

1. Obtenha os recursos **Bundle**:

   - Para clusters **RKE2**:

   ```
   curl -o rke2-plan-bundle.yaml https://raw.githubusercontent.com/suse-edge/
   fleet-examples/refs/tags/release-3.3.0/bundles/day2/system-upgrade-controller-
   plans/rke2-upgrade/plan-bundle.yaml
   ```

   - Para clusters **K3s**:

   ```
   curl -o k3s-plan-bundle.yaml https://raw.githubusercontent.com/suse-edge/fleet-
   examples/refs/tags/release-3.3.0/bundles/day2/system-upgrade-controller-plans/
   k3s-upgrade/plan-bundle.yaml
   ```

2. Edite a configuração do `Bundle`:

   - Altere os clusters de **destino** para que o `Bundle` aponte para o seu cluster `local` (gerenciamento):

   ```
   spec:
     targets:
     - clusterName: local
   ```

   Nota

   Em alguns casos de uso, o cluster `local` pode ter um nome diferente.

   Para recuperar o nome do cluster `local`, execute o comando abaixo:

   ```
   kubectl get clusters.fleet.cattle.io -n fleet-local
   ```

   - Altere o **namespace** do `Bundle` para apontar para o namespace `fleet-local`.

```
# Example
kind: Bundle
apiVersion: fleet.cattle.io/v1alpha1
metadata:
  name: rke2-upgrade
  namespace: fleet-local
...
```

3. Aplique os recursos **Bundle** ao `cluster de gerenciamento`:

```
# For RKE2
kubectl apply -f rke2-plan-bundle.yaml

# For K3s
kubectl apply -f k3s-plan-bundle.yaml
```

4. Visualize o recurso **Bundle** criado no namespace `fleet-local`:

```
# For RKE2
kubectl get bundles rke2-upgrade -n fleet-local

# For K3s
kubectl get bundles k3s-upgrade -n fleet-local

# Example output
NAME           BUNDLEDEPLOYMENTS-READY   STATUS
k3s-upgrade    0/0
rke2-upgrade   0/0
```

35.2.5.4.3 Implantação do plano do SUC: fluxo de trabalho do GitOps de terceiros

Em alguns casos de uso, talvez você queira incorporar os `planos do SUC de upgrade do Kubernetes` ao fluxo de trabalho do GitOps de terceiros (por exemplo, o `Flux`).

Para obter os recursos de upgrade do K8s upgrade que você precisa, primeiro determine a tag da versão (https://github.com/suse-edge/fleet-examples/releases) do Edge do repositório suse-edge/fleet-examples (https://github.com/suse-edge/fleet-examples.git) que deseja usar.

Depois disso, os recursos estarão em:

- Para ugrade do cluster RKE2:
  - Para os nós do `plano de controle`: `fleets/day2/system-upgrade-controller-plans/rke2-upgrade/plan-control-plane.yaml`
  - Para os nós de `worker`: `fleets/day2/system-upgrade-controller-plans/rke2-upgrade/plan-worker.yaml`
- Para upgrade do cluster K3s:
  - Para os nós do `plano de controle`: `fleets/day2/system-upgrade-controller-plans/k3s-upgrade/plan-control-plane.yaml`
  - Para os nós de `worker`: `fleets/day2/system-upgrade-controller-plans/k3s-upgrade/plan-worker.yaml`

Importante

Esses recursos do `plano` são interpretados pelo `System Upgrade Controller` e devem ser implantados em cada cluster downstream do qual você deseja fazer upgrade. Para obter informações sobre a implantação do SUC, consulte a *Seção 22.2, "Instalando o System Upgrade Controller"*.

Para entender melhor como usar o fluxo de trabalho do GitOps para implantar os **planos do SUC** para upgrade de versão do Kubernetes, consulte a visão geral (*Seção 35.2.5.2, "Visão geral"*) do procedimento de atualização com o `Fleet`.

35.2.6 Upgrade do gráfico Helm

Esta seção aborda as seguintes partes:

1. *Seção 35.2.6.1, "Preparação para ambientes air-gapped"*: armazena informações sobre como enviar gráficos e imagens OCI relacionados ao Edge para seu registro particular.
2. *Seção 35.2.6.2, "Procedimento de upgrade"*: armazena informações sobre diversos casos de uso de upgrade de gráficos Helm e o respectivo procedimento de upgrade.

35.2.6.1 Preparação para ambientes air-gapped

35.2.6.1.1 Garantir que você tenha acesso ao Fleet por gráfico Helm

Dependendo da compatibilidade do seu ambiente, você poderá seguir uma destas opções:

1. Hospede os recursos do Fleet do seu gráfico em um servidor Git local que possa ser acessado pelo `cluster de gerenciamento`.
2. Use a CLI do Fleet para converter um gráfico Helm em bundle (https://fleet.rancher.io/bundle-add#convert-a-helm-chart-into-a-bundle), que você pode usar diretamente e não precisa ser hospedado em nenhum lugar. É possível recuperar a CLI do Fleet da página referente à respectiva versão (https://github.com/rancher/fleet/releases/tag/v0.12.2). Para usuários do Mac, existe um Homebrew Formulae fleet-cli (https://formulae.brew.sh/formula/fleet-cli).

35.2.6.1.2 Encontrar os ativos necessários à sua versão de lançamento do Edge

1. Vá para a página de versões (https://github.com/suse-edge/fleet-examples/releases) do "dia 2" e localize a versão do Edge para a qual você quer fazer upgrade do gráfico e clique em **Assets** (Ativos).
2. Na seção "**Assets**" (Ativos), faça download dos seguintes arquivos:

Arquivo de versão	**Descrição**
edge-save-images.sh	Obtém as imagens especificadas no arquivo `edge-release-images.txt` e as compacta em um arquivo ".tar.gz".
edge-save-oci-artefacts.sh	Obtém as imagens de gráfico OCI relacionadas à versão especifica do Edge e as compacta em um arquivo ".tar.gz".
edge-load-images.sh	Carrega as imagens de um arquivo ".tar.gz", faz a remarcação e as envia a um registro particular.

edge-load-oci-artefacts.sh	Detecta o diretório que contém os pacotes ".tgz" de gráficos OCI do Edge e os carrega em um registro particular.
edge-release-helm-oci-artefacts.txt	Contém uma lista de imagens de gráfico OCI relacionadas a uma versão específica do Edge.
edge-release-images.txt	Contém uma lista de imagens relacionadas a uma versão específica do Edge.

35.2.6.1.3 Criar o arquivo de imagens da versão do Edge

Em uma máquina com acesso à Internet:

1. Torne o `edge-save-images.sh` executável:

```
chmod +x edge-save-images.sh
```

2. Gere o arquivo de imagens:

```
./edge-save-images.sh --source-registry registry.suse.com
```

3. Esse procedimento cria um arquivo pronto para carregamento chamado `edge-images.tar.gz`.

Nota

Se a opção `-i|--images` foi especificada, o nome do arquivo pode ser diferente.

4. Copie esse arquivo para sua máquina **air-gapped**:

```
scp edge-images.tar.gz <user>@<machine_ip>:/path
```

35.2.6.1.4 Criar o arquivo de imagens de gráfico OCI do Edge

Em uma máquina com acesso à Internet:

1. Torne o `edge-save-oci-artefacts.sh` executável:

```
chmod +x edge-save-oci-artefacts.sh
```

2. Gere o arquivo de imagens de gráfico OCI:

```
./edge-save-oci-artefacts.sh --source-registry registry.suse.com
```

3. Esse procedimento cria um arquivo chamado `oci-artefacts.tar.gz`.

Nota

Se a opção `-a|--archive` foi especificada, o nome do arquivo pode ser diferente.

4. Copie esse arquivo para sua máquina **air-gapped**:

```
scp oci-artefacts.tar.gz <user>@<machine_ip>:/path
```

35.2.6.1.5 Carregar as imagens da versão do Edge para sua máquina air-gapped

Na máquina air-gapped:

1. Faça login no seu registro particular (se necessário):

```
podman login <REGISTRY.YOURDOMAIN.COM:PORT>
```

2. Torne o `edge-load-images.sh` executável:

```
chmod +x edge-load-images.sh
```

3. Execute o script, especificando o arquivo já **copiado** `edge-images.tar.gz`:

```
./edge-load-images.sh --source-registry registry.suse.com --registry
 <REGISTRY.YOURDOMAIN.COM:PORT> --images edge-images.tar.gz
```

Nota

Desse modo, todas as imagens do `edge-images.tar.gz` serão carregadas, remarcadas e enviadas ao registro especificado na opção `--registry`.

35.2.6.1.6 Carregar as imagens de gráfico OCI do Edge em sua máquina air-gapped

Na máquina air-gapped:

1. Faça login no seu registro particular (se necessário):

```
podman login <REGISTRY.YOURDOMAIN.COM:PORT>
```

2. Torne o `edge-load-oci-artefacts.sh` executável:

```
chmod +x edge-load-oci-artefacts.sh
```

3. Descompacte o arquivo `oci-artefacts.tar.gz` copiado:

```
tar -xvf oci-artefacts.tar.gz
```

4. Isso cria um diretório com o gabarito de nomenclatura `edge-release-oci-tgz-<data>`.

5. Especifique esse diretório no script `edge-load-oci-artefacts.sh` para carregar as imagens de gráfico OCI do Edge em seu registro particular:

Nota

Esse script assume que a CLI `helm` foi pré-instalada em seu ambiente. Para obter instruções de instalação do Helm, consulte Installing Helm (https://helm.sh/docs/intro/install/) (Instalando o Helm).

```
./edge-load-oci-artefacts.sh --archive-directory edge-release-oci-tgz-<date> --
registry <REGISTRY.YOURDOMAIN.COM:PORT> --source-registry registry.suse.com
```

35.2.6.1.7 Configurar o registro particular na distribuição Kubernetes

Para RKE2, consulte Private Registry Configuration (https://docs.rke2.io/install/private_registry) (Configuração de registro particular)

Para K3s, consulte Private Registry Configuration (https://docs.k3s.io/installation/private-registry) (Configuração de registro particular)

35.2.6.2 Procedimento de upgrade

Esta seção tem como foco os seguintes casos de uso do procedimento de upgrade do Helm:

1. *Seção 35.2.6.2.1, "Eu tenho um novo cluster e desejo implantar e gerenciar um gráfico Helm do Edge"*
2. *Seção 35.2.6.2.2, "Quero fazer upgrade de um gráfico Helm gerenciado pelo Fleet"*
3. *Seção 35.2.6.2.3, "Quero fazer upgrade de um gráfico Helm implantado pelo EIB"*

Importante

Não é possível fazer upgrade confiável de gráficos Helm implantados manualmente. Sugerimos reimplantá-los usando o método da *Seção 35.2.6.2.1, "Eu tenho um novo cluster e desejo implantar e gerenciar um gráfico Helm do Edge"*.

35.2.6.2.1 Eu tenho um novo cluster e desejo implantar e gerenciar um gráfico Helm do Edge

Esta seção aborda o seguinte:

1. *Seção 35.2.6.2.1.1, "Preparar os recursos do Fleet para seu gráfico"*.
2. *Seção 35.2.6.2.1.2, "Implantar o Fleet para seu gráfico"*.
3. *Seção 35.2.6.2.1.3, "Gerenciar o gráfico Helm implantado"*.

35.2.6.2.1.1 Preparar os recursos do Fleet para seu gráfico

1. Adquira os recursos do Fleet do gráfico com base na tag da versão (https://github.com/suse-edge/fleet-examples/releases) do Edge que deseja usar.
2. Navegue até a instância do Fleet do gráfico Helm (`fleets/day2/chart-templates/<gráfico>`).
3. **Se você pretende usar um fluxo de trabalho do GitOps**, copie o diretório do Fleet do gráfico para o repositório Git do qual você vai executar o GitOps.

4. **Opcionalmente**, se o gráfico Helm exigir configurações em seus **valores**, edite a configuração `.helm.values` no arquivo `fleet.yaml` do diretório copiado.

5. **Opcionalmente**, pode haver casos de uso em que você tenha que adicionar outros recursos à instância do Fleet do seu gráfico para que se adapte melhor ao seu ambiente. Para obter informações de como aprimorar o diretório do Fleet, consulte Git Repository Contents (https://fleet.rancher.io/gitrepo-content) (Conteúdo do repositório Git).

Nota

Em alguns casos, o tempo limite padrão que o Fleet usa nas operações do Helm pode ser insuficiente e resultar no seguinte erro:

```
failed pre-install: context deadline exceeded
```

Nesses casos, adicione a propriedade timeoutSeconds (https://fleet.rancher.io/ref-crds#helmoptions) à configuração `helm` do arquivo `fleet.yaml`.

Esta é a aparência de um **exemplo** de gráfico Helm do `longhorn`:

- Estrutura de repositórios Git do usuário:

```
<user_repository_root>
├── longhorn
│   └── fleet.yaml
└── longhorn-crd
    └── fleet.yaml
```

- Conteúdo do `fleet.yaml` preenchido com os dados do `Longhorn` do usuário:

```
defaultNamespace: longhorn-system

helm:
  # timeoutSeconds: 10
  releaseName: "longhorn"
  chart: "longhorn"
  repo: "https://charts.rancher.io/"
  version: "106.2.0+up1.8.1"
  takeOwnership: true
  # custom chart value overrides
  values:
    # Example for user provided custom values content
    defaultSettings:
```

```
      deletingConfirmationFlag: true

# https://fleet.rancher.io/bundle-diffs
diff:
  comparePatches:
  - apiVersion: apiextensions.k8s.io/v1
    kind: CustomResourceDefinition
    name: engineimages.longhorn.io
    operations:
    - {"op":"remove", "path":"/status/conditions"}
    - {"op":"remove", "path":"/status/storedVersions"}
    - {"op":"remove", "path":"/status/acceptedNames"}
  - apiVersion: apiextensions.k8s.io/v1
    kind: CustomResourceDefinition
    name: nodes.longhorn.io
    operations:
    - {"op":"remove", "path":"/status/conditions"}
    - {"op":"remove", "path":"/status/storedVersions"}
    - {"op":"remove", "path":"/status/acceptedNames"}
  - apiVersion: apiextensions.k8s.io/v1
    kind: CustomResourceDefinition
    name: volumes.longhorn.io
    operations:
    - {"op":"remove", "path":"/status/conditions"}
    - {"op":"remove", "path":"/status/storedVersions"}
    - {"op":"remove", "path":"/status/acceptedNames"}
```

Nota

Estes são apenas valores de exemplo usados para ilustrar as configurações comuns no gráfico do `longhorn`. Eles **NÃO** devem ser considerados diretrizes de implantação para o gráfico do `longhorn`.

35.2.6.2.1.2 Implantar o Fleet para seu gráfico

É possível implantar o Fleet para seu gráfico usando o GitRepo (*Seção 35.2.6.2.1.2.1, "GitRepo"*) ou o bundle (*Seção 35.2.6.2.1.2.2, "Bundle"*).

Nota

Durante a implantação do Fleet, se você receber a mensagem `Modified` (Modificado), adicione uma entrada `comparePatches` correspondente à seção `diff` do Fleet. Para obter mais informações, consulte Generating Diffs to Ignore Modified GitRepos (https://fleet.rancher.io/bundle-diffs) (Gerando diffs para ignorar GitRepos modificados).

35.2.6.2.1.2.1 GitRepo

O recurso GitRepo (https://fleet.rancher.io/ref-gitrepo) do Fleet armazena as informações sobre como acessar os recursos do Fleet do seu gráfico e a quais clusters ele precisa aplicar esses recursos.

É possível implantar o recurso `GitRepo` pela IU do Rancher (https://ranchermanager.docs.rancher.com/v2.11/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui) ou manualmente pela implantação (https://fleet.rancher.io/tut-deployment) do recurso no `cluster de gerenciamento`.

Exemplo de recurso `GitRepo` do **Longhorn** para implantação **manual**:

```
apiVersion: fleet.cattle.io/v1alpha1
kind: GitRepo
metadata:
  name: longhorn-git-repo
  namespace: fleet-local
spec:
  # If using a tag
  # revision: user_repository_tag
  #
  # If using a branch
  # branch: user_repository_branch
  paths:
  # As seen in the 'Prepare your Fleet resources' example
  - longhorn
  - longhorn-crd
  repo: user_repository_url
```

35.2.6.2.1.2.2 Bundle

Os recursos Bundle (https://fleet.rancher.io/bundle-add) armazenam os recursos brutos do Kubernetes que o Fleet precisa implantar. Normalmente, a recomendação é usar a abordagem do `GitRepo`, mas para casos de uso em que o ambiente é air-gapped e não oferece suporte a um servidor Git local, os `Bundles` podem ajudar na propagação do Fleet do gráfico Helm aos clusters de destino.

É possível implantar um `Bundle` pela IU do Rancher (`Continuous Delivery → Advanced → Bundles → Create from YAML` (Entrega contínua → Avançado → Bundles → Criar do YAML)) ou pela implantação manual do recurso `Bundle` no namespace correto do Fleet. Para obter informações sobre os namespaces do Fleet, consulte a documentação (https://fleet.rancher.io/namespaces#gitrepos-bundles-clusters-clustergroups) upstream.

É possível criar `Bundles` para gráficos Helm do Edge usando a abordagem Converter um gráfico Helm em bundle (https://fleet.rancher.io/bundle-add#convert-a-helm-chart-into-a-bundle) do Fleet.

Veja a seguir um exemplo de como criar um recurso `Bundle` com base nos gabaritos de gráfico Helm longhorn (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/fleets/day2/chart-templates/longhorn/longhorn/fleet.yaml) e longhorn-crd (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/fleets/day2/chart-templates/longhorn/longhorn-crd/fleet.yaml) do Fleet e implantar manualmente esse bundle no `cluster de gerenciamento`.

Nota

Para ilustrar o fluxo de trabalho, o exemplo a seguir usa a estrutura de diretório suse-edge/fleet-examples (https://github.com/suse-edge/fleet-examples).

1. Navegue até o gabarito de gráfico longhorn (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/fleets/day2/chart-templates/longhorn/longhorn/fleet.yaml) do Fleet:

   ```
   cd fleets/day2/chart-templates/longhorn/longhorn
   ```

2. Crie um arquivo `targets.yaml` para instruir o Fleet sobre os clusters em que ele deve implantar o gráfico Helm:

   ```
   cat > targets.yaml <<EOF
   targets:
   # Match your local (management) cluster
   - clusterName: local
   ```

```
EOF
```

Nota

Em alguns casos de uso, o nome do seu cluster local pode ser diferente.

Para recuperar o nome do seu cluster local, execute este comando:

```
kubectl get clusters.fleet.cattle.io -n fleet-local
```

3. Converta o gráfico Helm `Longhorn` do Fleet em um recurso bundle usando fleet-cli (https://fleet.rancher.io/cli/fleet-cli/fleet).

Nota

É possível recuperar a CLI do Fleet da página **Assets** (Ativos) da respectiva versão (https://github.com/rancher/fleet/releases/tag/v0.12.2) (`fleet-linux-amd64`).

Para usuários do Mac, existe um Homebrew Formulae fleet-cli (https://formulae.brew.sh/formula/fleet-cli).

```
fleet apply --compress --targets-file=targets.yaml -n fleet-local -o - longhorn-
bundle > longhorn-bundle.yaml
```

4. Navegue até o gabarito de gráfico longhorn-crd (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/fleets/day2/chart-templates/longhorn/longhorn-crd/fleet.yaml) do Fleet:

```
cd fleets/day2/chart-templates/longhorn/longhorn-crd
```

5. Crie um arquivo `targets.yaml` para instruir o Fleet sobre os clusters em que ele deve implantar o gráfico Helm:

```
cat > targets.yaml <<EOF
targets:
# Match your local (management) cluster
- clusterName: local
EOF
```

6. Converta o gráfico Helm `Longhorn CRD` do Fleet em um recurso bundle usando fleet-cli (https://fleet.rancher.io/cli/fleet-cli/fleet).

```
fleet apply --compress --targets-file=targets.yaml -n fleet-local -o - longhorn-crd-
bundle > longhorn-crd-bundle.yaml
```

7. Implante os arquivos `longhorn-bundle.yaml` e `longhorn-crd-bundle.yaml` no `cluster de gerenciamento`:

```
kubectl apply -f longhorn-crd-bundle.yaml
kubectl apply -f longhorn-bundle.yaml
```

Siga estas etapas para garantir que o `SUSE Storage` seja implantado em todos os clusters de gerenciamento especificados.

35.2.6.2.1.3 Gerenciar o gráfico Helm implantado

Após a implantação com o Fleet, para fazer upgrade dos gráficos Helm, consulte a *Seção 35.2.6.2.2, "Quero fazer upgrade de um gráfico Helm gerenciado pelo Fleet"*.

35.2.6.2.2 Quero fazer upgrade de um gráfico Helm gerenciado pelo Fleet

1. Determine a versão para a qual você precisa fazer upgrade do seu gráfico para que ele fique compatível com o lançamento desejado do Edge. Você encontra a versão do gráfico Helm por lançamento do Edge nas Notas de lançamento (*Seção 52.1, "Resumo"*).
2. No repositório Git monitorado pelo Fleet, edite o arquivo `fleet.yaml` do gráfico Helm com a **versão** e o **repositório** corretos do gráfico conforme as Notas de lançamento (*Seção 52.1, "Resumo"*).
3. Depois de confirmar e enviar as alterações ao repositório, será acionado um upgrade do gráfico Helm desejado.

35.2.6.2.3 Quero fazer upgrade de um gráfico Helm implantado pelo EIB

O *Capítulo 11, Edge Image Builder* implanta gráficos Helm criando um recurso `HelmChart` e usando o `helm-controller` introduzido pelo recurso de integração do Helm RKE2 (https://docs.rke2.io/helm)/K3s (https://docs.k3s.io/helm).

Para garantir que o upgrade do gráfico Helm implantado pelo `EIB` seja feito com sucesso, os usuários precisam fazer upgrade dos respectivos recursos `HelmChart`.

Veja a seguir informações sobre:

- A visão geral (*Seção 35.2.6.2.3.1, "Visão geral"*) do processo de upgrade.
- As etapas de upgrade necessárias (*Seção 35.2.6.2.3.2, "Etapas de upgrade"*).
- Um exemplo (*Seção 35.2.6.2.3.3, "Exemplo"*) que demonstra o upgrade de um gráfico Longhorn (https://longhorn.io) usando o método explicado.
- Como usar o processo de upgrade com uma ferramenta GitOps diferente (*Seção 35.2.6.2.3.4, "Upgrade do gráfico Helm usando uma ferramenta GitOps de terceiros"*).

35.2.6.2.3.1 Visão geral

O upgrade dos gráficos Helm que são implantados pelo `EIB` é feito por uma instância do `Fleet` chamada eib-charts-upgrader (https://github.com/suse-edge/fleet-examples/tree/release-3.3.0/fleets/day2/eib-charts-upgrader).

Essa instância do `Fleet` processa os dados **fornecidos pelo usuário** para **atualizar** um conjunto específico de recursos HelmChart.

A atualização desses recursos aciona o helm-controller (https://github.com/k3s-io/helm-controller), que faz **upgrade** dos gráficos Helm associados aos recursos `HelmChart` modificados.

O usuário precisa apenas:

1. Obter (https://helm.sh/docs/helm/helm_pull/) localmente os arquivos para cada gráfico Helm que precisa de upgrade.
2. Especificar esses arquivos no script generate-chart-upgrade-data.sh (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/scripts/day2/generate-chart-upgrade-data.sh) `generate-chart-upgrade-data.sh`, que incluirá os dados dos arquivos na instância do Fleet `eib-charts-upgrader`.
3. Implantar a instância do Fleet `eib-charts-upgrader` no respectivo `cluster de gerenciamento`. Isso é feito por um recurso `GitRepo` ou `Bundle`.

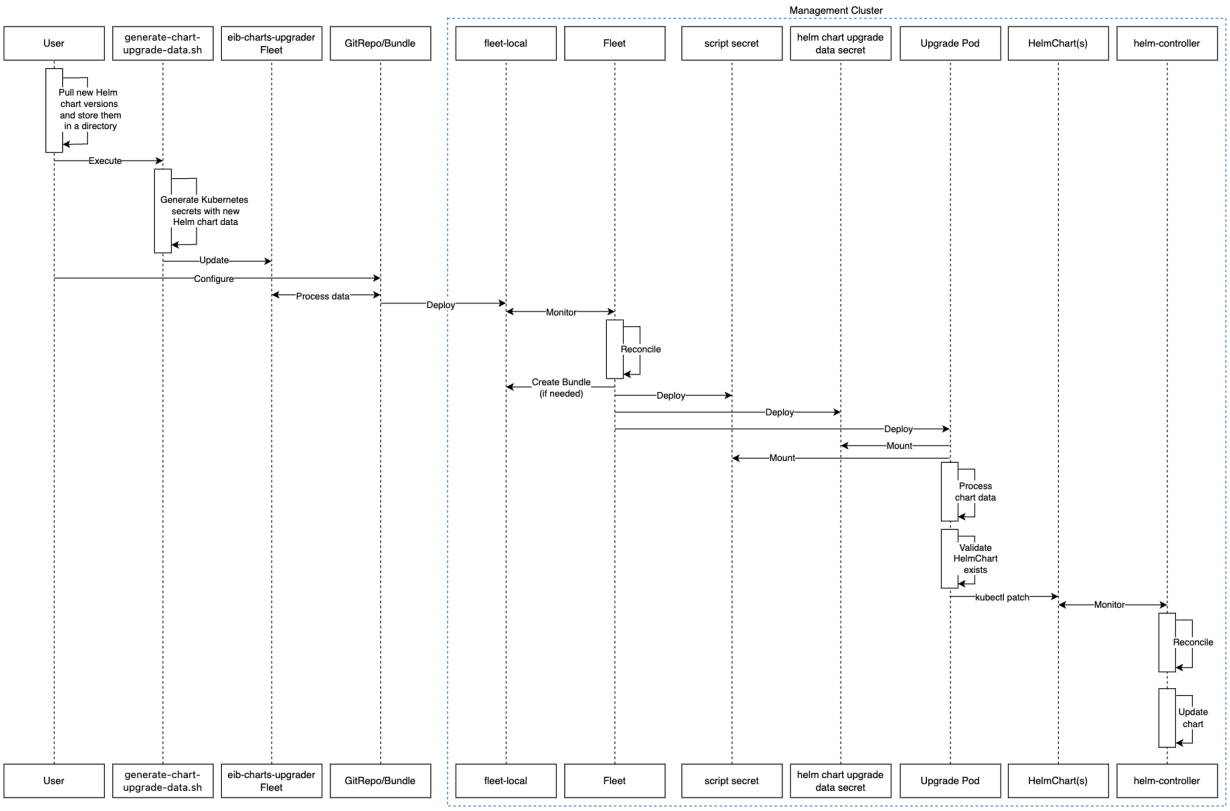
Após a implantação, o `eib-charts-upgrader`, com ajuda do Fleet, enviará os recursos para o cluster de gerenciamento desejado.

Esses recursos incluem:

1. Um conjunto de `segredos` com os dados do gráfico Helm **fornecidos pelo usuário**.
2. Um `Kubernetes Job` para implantar o `pod` que vai montar os `segredos` já mencionados e, com base neles, aplicar o patch (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_patch/) aos recursos HelmChart correspondentes.

Como já foi mencionado, isso acionará o `helm-controller`, que faz upgrade do gráfico Helm real.

Veja a seguir um diagrama da descrição acima:



35.2.6.2.3.2 Etapas de upgrade

1. Clone o repositório `suse-edge/fleet-examples` da tag (https://github.com/suse-edge/fleet-examples/releases/tag/release-3.3.0) da versão correta.
2. Crie um diretório para armazenar um ou mais arquivos dos gráficos Helm enviados.

```
mkdir archives
```

3. Dentro do diretório do arquivo recém-criado, extraia (https://helm.sh/docs/helm/helm_pull/) os arquivos dos gráficos Helm dos quais você deseja fazer upgrade:

```
cd archives
helm pull [chart URL | repo/chartname]

# Alternatively if you want to pull a specific version:
# helm pull [chart URL | repo/chartname] --version 0.0.0
```

4. Em **Assets** (Ativos) da tag da versão (https://github.com/suse-edge/fleet-examples/releases/tag/release-3.3.0) desejada, faça download do script `generate-chart-upgrade-data.sh`.

5. Execute o script `generate-chart-upgrade-data.sh`:

```
chmod +x ./generate-chart-upgrade-data.sh

./generate-chart-upgrade-data.sh --archive-dir /foo/bar/archives/ --fleet-path /foo/bar/fleet-examples/fleets/day2/eib-charts-upgrader
```

   Para cada arquivo de gráfico no diretório `--archive-dir`, o script gera um arquivo `YAML de segredo do Kubernetes` com os dados de upgrade do gráfico e o armazena no diretório `base/secrets` da instância do Fleet especificada por `--fleet-path`.
   O script `generate-chart-upgrade-data.sh` também aplica modificações adicionais à instância do Fleet para garantir que os arquivos `YAML de segredo do Kubernetes` gerados sejam corretamente usados pela carga de trabalho implantada pelo Fleet.

Importante

Os usuários não devem fazer alterações no conteúdo que é gerado pelo script `generate-chart-upgrade-data.sh`.

As etapas abaixo dependem do ambiente que está em execução:

1. Para um ambiente com suporte a GitOps (por exemplo, não air-gapped ou air-gapped, mas com suporte a servidor Git local):

   a. Copie a instância do Fleet `fleets/day2/eib-charts-upgrader` para o repositório que você vai usar com o GitOps.

Nota

Garanta que o Fleet inclua as alterações feitas pelo script `generate-chart-upgrade-data.sh`.

b. Configure o recurso `GitRepo` que será usado para enviar todos os recursos da instância do Fleet `eib-charts-upgrader`.

   i. Para configuração e implantação do `GitRepo` pela IU do Rancher, consulte Accessing Fleet in the Rancher UI (https://ranchermanager.docs.rancher.com/v2.11/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui) (Acessando o Fleet na IU do Racher).

   ii. Para configuração e implantação manuais do `GitRepo`, consulte Creating a Deployment (https://fleet.rancher.io/tut-deployment) (Criando uma implantação).

2. Para um ambiente sem suporte a GitOps (por exemplo, air-gapped que não permite o uso de servidor Git local):

   a. Faça download do binário `fleet-cli` na página da versão (https://github.com/rancher/fleet/releases/tag/v0.12.2) do `rancher/fleet` (`fleet-linux-amd64` para Linux). Para usuários do Mac, existe um Homebrew Formulae que pode ser usado: fleet-cli (https://formulae.brew.sh/formula/fleet-cli).

   b. Navegue até a instância do Fleet `eib-charts-upgrader`:

```
cd /foo/bar/fleet-examples/fleets/day2/eib-charts-upgrader
```

   c. Crie um arquivo `targets.yaml` para instruir o Fleet sobre onde implantar seus recursos:

```
cat > targets.yaml <<EOF
targets:
# To map the local(management) cluster
- clusterName: local
EOF
```

Nota

Em alguns casos de uso, o cluster `local` pode ter um nome diferente.

Para recuperar o nome do cluster `local`, execute o comando abaixo:

```
kubectl get clusters.fleet.cattle.io -n fleet-local
```

d. Use `fleet-cli` para converter o Fleet em um recurso `Bundle`:

```
fleet apply --compress --targets-file=targets.yaml -n fleet-local -o - eib-
charts-upgrade > bundle.yaml
```

Isso cria um bundle (`bundle.yaml`) para armazenar todos os recursos usados como gabaritos da instância do Fleet `eib-charts-upgrader`.
Para obter mais informações sobre o comando `fleet apply`, consulte fleet apply (https://fleet.rancher.io/cli/fleet-cli/fleet_apply).
Para obter mais informações sobre como converter instâncias do Fleet em bundles, consulte Convert a Helm Chart into a Bundle (https://fleet.rancher.io/bundle-add#convert-a-helm-chart-into-a-bundle) (Converter um gráfico Helm em bundle).

e. Implante o `Bundle`. É possível fazer isso de uma destas maneiras:

i. Pela IU do Rancher: navegue até **Continuous Delivery** → **Advanced** → **Bundles** → **Create from YAML** (Entrega contínua → Avançado → Bundles → Criar do YAML) e cole o conteúdo do `bundle.yaml` ou clique na opção `Read from File` (Ler arquivo) e especifique o arquivo.

ii. Manualmente: implante o arquivo `bundle.yaml` manualmente no `cluster de gerenciamento`.

A execução dessas etapas resulta na implantação bem-sucedida do recurso `GitRepo/Bundle`. O Fleet seleciona o recurso e seu conteúdo é implantado nos clusters de destino que o usuário especificou nas etapas anteriores. Para obter uma visão geral do processo, consulte a *Seção 35.2.6.2.3.1, "Visão geral"*.

Para obter informações sobre como acompanhar o processo de upgrade, consulte a *Seção 35.2.6.2.3.3, "Exemplo"*.

Importante

Após a verificação bem-sucedida da atualização do gráfico, remova o recurso `Bundle/GitRepo`.

Isso removerá os recursos de upgrade que não são mais necessários do cluster de `gerenciamento`, garantindo que não haja conflitos com versões futuras.

35.2.6.2.3.3 Exemplo

Nota

No exemplo a seguir, demonstramos como fazer upgrade de um gráfico Helm implantado pelo `EIB` de uma versão para outra em um cluster de `gerenciamento`. Observe que as versões usadas neste exemplo **não** são recomendações. Para recomendações de versão específicas a um lançamento do Edge, consulte as Notas de lançamento (*Seção 52.1, "Resumo"*).

Caso de uso:

- Um cluster de `gerenciamento` que executa uma versão mais antiga do Longhorn (https://longhorn.io).
- O cluster foi implantado pelo EIB usando o seguinte *trecho* de definição da imagem:

```
kubernetes:
  helm:
    charts:
    - name: longhorn-crd
      repositoryName: rancher-charts
      targetNamespace: longhorn-system
      createNamespace: true
      version: 104.2.0+up1.7.1
      installationNamespace: kube-system
    - name: longhorn
      repositoryName: rancher-charts
      targetNamespace: longhorn-system
      createNamespace: true
      version: 104.2.0+up1.7.1
      installationNamespace: kube-system
    repositories:
```

```
    - name: rancher-charts
      url: https://charts.rancher.io/
...
```

- É preciso fazer upgrade do `SUSE Storage` para uma versão compatível com o Edge 3.3.1, ou seja, para a versão `106.2.0+up1.8.1`.
- Presume-se que o `cluster de gerenciamento` seja **air-gapped**, sem suporte a servidor Git local e com uma instalação ativa do Rancher.

Siga as etapas de upgrade (*Seção 35.2.6.2.3.2, "Etapas de upgrade"*):

1. Clone o repositório `suse-edge/fleet-example` da tag `release-3.3.0`.

```
git clone -b release-3.3.0 https://github.com/suse-edge/fleet-examples.git
```

2. Crie um diretório para armazenar o arquivo de upgrade do `Longhorn`.

```
mkdir archives
```

3. Extraia a versão desejada do arquivo de gráficos do `Longhorn`:

```
# First add the Rancher Helm chart repository
helm repo add rancher-charts https://charts.rancher.io/

# Pull the Longhorn 1.8.1 CRD archive
helm pull rancher-charts/longhorn-crd --version 106.2.0+up1.8.1

# Pull the Longhorn 1.8.1 chart archive
helm pull rancher-charts/longhorn --version 106.2.0+up1.8.1
```

4. Fora do diretório `archives`, faça download do script `generate-chart-upgrade-data.sh` da tag (https://github.com/suse-edge/fleet-examples/releases/tag/release-3.3.0) da versão do `suse-edge/fleet-examples`.
5. A configuração do diretório deve ter uma aparência como esta:

```
.
├── archives
|   ├── longhorn-106.2.0+up1.8.1.tgz
|   └── longhorn-crd-106.2.0+up1.8.1.tgz
├── fleet-examples
...
|   ├── fleets
|   |   ├── day2
```

```
|   |   |   ├── ...
|   |   |   ├── eib-charts-upgrader
|   |   |   |   ├── base
|   |   |   |   |   ├── job.yaml
|   |   |   |   |   ├── kustomization.yaml
|   |   |   |   |   ├── patches
|   |   |   |   |   |   └── job-patch.yaml
|   |   |   |   |   ├── rbac
|   |   |   |   |   |   ├── cluster-role-binding.yaml
|   |   |   |   |   |   ├── cluster-role.yaml
|   |   |   |   |   |   ├── kustomization.yaml
|   |   |   |   |   |   └── sa.yaml
|   |   |   |   |   └── secrets
|   |   |   |   |       ├── eib-charts-upgrader-script.yaml
|   |   |   |   |       └── kustomization.yaml
|   |   |   |   ├── fleet.yaml
|   |   |   |   └── kustomization.yaml
|   |   |   └── ...
|   └── ...
└── generate-chart-upgrade-data.sh
```

6. Execute o script `generate-chart-upgrade-data.sh`:

```
# First make the script executable
chmod +x ./generate-chart-upgrade-data.sh

# Then execute the script
./generate-chart-upgrade-data.sh --archive-dir ./archives --fleet-path ./fleet-
examples/fleets/day2/eib-charts-upgrader
```

A estrutura de diretórios após a execução do script deve ser parecida com esta:

```
.
├── archives
|   ├── longhorn-106.2.0+up1.8.1.tgz
|   └── longhorn-crd-106.2.0+up1.8.1.tgz
├── fleet-examples
...
|   ├── fleets
|   |   ├── day2
|   |   |   ├── ...
|   |   |   ├── eib-charts-upgrader
|   |   |   |   ├── base
|   |   |   |   |   ├── job.yaml
|   |   |   |   |   ├── kustomization.yaml
|   |   |   |   |   ├── patches
|   |   |   |   |   |   └── job-patch.yaml
```

```
|   |   |   |   |   ├── rbac
|   |   |   |   |   |   ├── cluster-role-binding.yaml
|   |   |   |   |   |   ├── cluster-role.yaml
|   |   |   |   |   |   ├── kustomization.yaml
|   |   |   |   |   |   └── sa.yaml
|   |   |   |   |   └── secrets
|   |   |   |   |       ├── eib-charts-upgrader-script.yaml
|   |   |   |   |       ├── kustomization.yaml
|   |   |   |   |       ├── longhorn-VERSION.yaml - secret created by the generate-
chart-upgrade-data.sh script
|   |   |   |   |       └── longhorn-crd-VERSION.yaml - secret created by the
 generate-chart-upgrade-data.sh script
|   |   |   |   ├── fleet.yaml
|   |   |   |   └── kustomization.yaml
|   |   |   └── ...
|   └── ...
└── generate-chart-upgrade-data.sh
```

Os arquivos alterados no Git devem ter aparência similar a esta:

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
 modified:   fleets/day2/eib-charts-upgrader/base/patches/job-patch.yaml
 modified:   fleets/day2/eib-charts-upgrader/base/secrets/kustomization.yaml

Untracked files:
  (use "git add <file>..." to include in what will be committed)
 fleets/day2/eib-charts-upgrader/base/secrets/longhorn-VERSION.yaml
 fleets/day2/eib-charts-upgrader/base/secrets/longhorn-crd-VERSION.yaml
```

7. Crie um `Bundle` para a instância do Fleet `eib-charts-upgrader`:

   a. Primeiro, navegue até o Fleet:

   ```
   cd ./fleet-examples/fleets/day2/eib-charts-upgrader
   ```

   b. Em seguida, crie um arquivo `targets.yaml`:

   ```
   cat > targets.yaml <<EOF
   targets:
   - clusterName: local
   ```

```
EOF
```

c. Depois disso, use o binário `fleet-cli` para converter a instância do Fleet em um bundle:

```
fleet apply --compress --targets-file=targets.yaml -n fleet-local -o - eib-charts-upgrade > bundle.yaml
```

8. Implante o bundle usando a IU do Rancher:

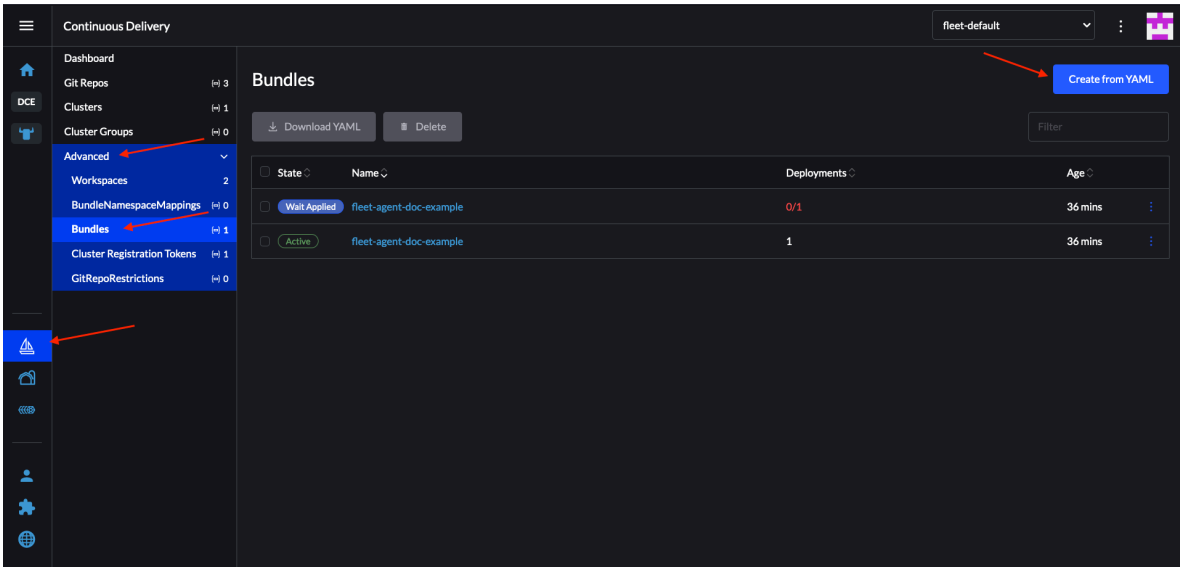


FIGURA 35.1: IMPLANTAR O BUNDLE USANDO A IU DO RANCHER

Agora selecione **Read from File** (Ler arquivo) e encontre o arquivo `bundle.yaml` no sistema.
Isso preencherá automaticamente o `Bundle` na IU do Rancher.
Selecione **Create** (Criar).

9. Após a implantação bem-sucedida, o bundle terá uma aparência similar a esta:

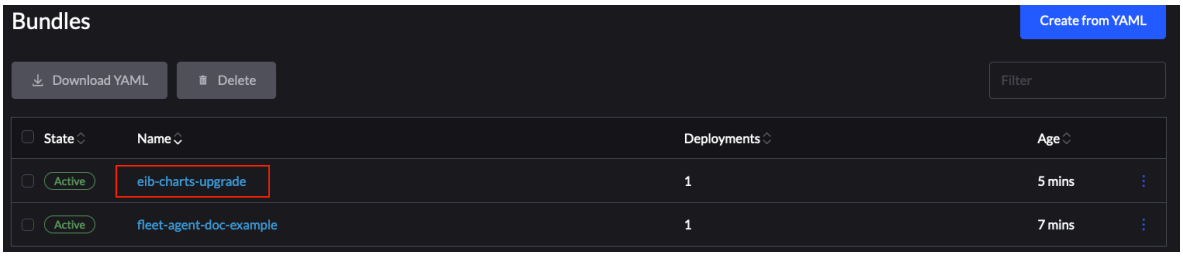
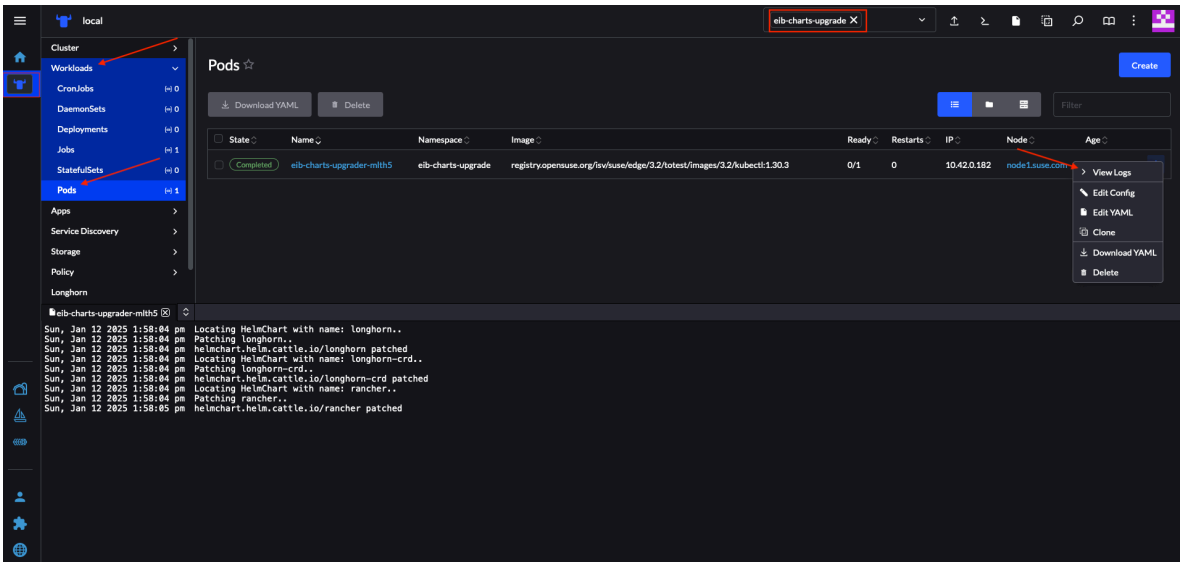


FIGURA 35.2: BUNDLE IMPLANTADO COM SUCESSO

Após a implantação bem-sucedida do `Bundle`, monitore o processo de upgrade da seguinte maneira:

1. Consulte os registros do `pod de upgrade`:



2. Agora consulte os registros do pod criado pelo helm-controller para o upgrade:

   a. O nome do pod seguirá o seguinte gabarito: `helm-install-longhorn-<sufixo-aleatório>`

   b. O pod estará no namespace em que o recurso `HelmChart` foi implantado. No nosso caso, o namespace é `kube-system`.

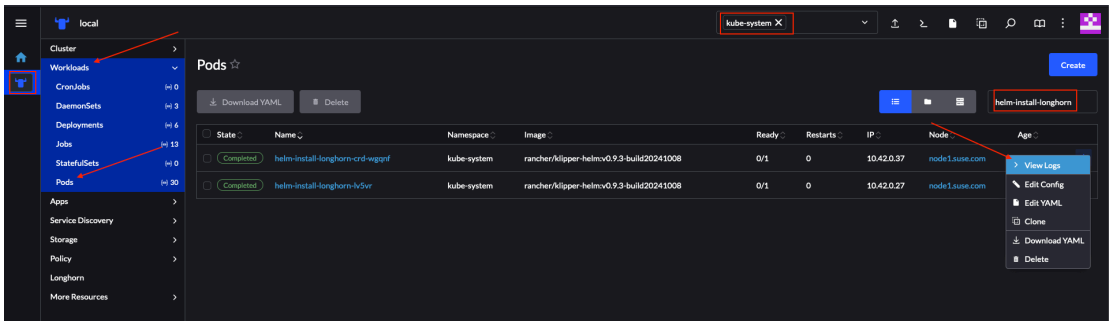


FIGURA 35.3: REGISTROS DO GRÁFICO DO LONGHORN ATUALIZADO COM SUCESSO

3. Verifique se a versão do `HelmChart` foi atualizada navegando até a seção `HelmCharts` do Rancher (`More Resources → HelmCharts` "Mais recursos → HelmCharts"). Selecione o namespace em que o gráfico foi implantado; que, neste exemplo, é `kube-system`.

4. Por fim, verifique se os pods do Longhorn estão em execução.

Depois de fazer as validações acima, é seguro pressupor que o upgrade do gráfico Helm do Longhorn foi feito para a versão `106.2.0+up1.8.1`.

35.2.6.2.3.4 Upgrade do gráfico Helm usando uma ferramenta GitOps de terceiros

Em alguns casos de uso, os usuários talvez queiram seguir esse procedimento de upgrade com um fluxo de trabalho do GitOps diferente do Fleet (por exemplo, `Flux`).

Para gerar os recursos necessários ao procedimento de upgrade, você pode usar o script `generate-chart-upgrade-data.sh` para preencher a instância do Fleet `eib-charts-upgrader` com os dados fornecidos pelo usuário. Para obter mais informações sobre como fazer isso, consulte a *Seção 35.2.6.2.3.2, "Etapas de upgrade"*.

Com a configuração completa, você pode usar o kustomize (https://kustomize.io) para gerar uma solução totalmente funcional para ser implantada no seu cluster:

```
cd /foo/bar/fleets/day2/eib-charts-upgrader

kustomize build .
```

Para incluir a solução no fluxo de trabalho do GitOps, remova o arquivo `fleet.yaml` e use o que sobrou como configuração válida do `Kustomize`. Lembre-se de executar primeiro o script `generate-chart-upgrade-data.sh` para que ele possa preencher a configuração do `Kustomize` com os dados dos gráficos Helm para o qual você quer fazer upgrade.

Para saber qual é a finalidade de uso desse fluxo de trabalho, consulte a *Seção 35.2.6.2.3.1, "Visão geral"* e a *Seção 35.2.6.2.3.2, "Etapas de upgrade"*.

36 Clusters downstream

Importante

As etapas a seguir não são válidas para clusters `downstream` gerenciados pelo SUSE Edge for Telco (*Capítulo 37, SUSE Edge for Telco*). Para obter orientação sobre o upgrade desses clusters, consulte a *Seção 43.2, "Upgrades de cluster downstream"*.

Esta seção apresenta as maneiras possíveis de executar operações de "dia 2" em diferentes partes do cluster `downstream`.

36.1 Fleet

Esta seção apresenta informações sobre como executar operações de "dia 2" usando o componente Fleet (*Capítulo 8, Fleet*).

Os seguintes tópicos são abordados como parte desta seção:

1. *Seção 36.1.1, "Componentes"*: componentes padrão usados para todas as operações de "dia 2".
2. *Seção 36.1.2, "Determinar seu caso de uso"*: apresenta uma visão geral dos recursos personalizados do Fleet que serão usados e como se adaptam aos diferentes casos de uso das operações de "dia 2".
3. *Seção 36.1.3, "Fluxo de trabalho de dia 2"*: fornece um guia de fluxo de trabalho para execução de operações de "dia 2" com o Fleet.
4. *Seção 36.1.4, "Upgrade de sistema operacional"*: descreve como fazer upgrades de sistema operacional com o Fleet.
5. *Seção 36.1.5, "Upgrade da versão do Kubernetes"*: descreve como fazer upgrades de versão do Kubernetes com o Fleet.
6. *Seção 36.1.6, "Upgrade do gráfico Helm"*: descreve como fazer upgrades de gráficos Helm com o Fleet.

36.1.1 Componentes

Veja a seguir uma descrição dos componentes padrão que você deve configurar no cluster `downstream` para realizar operações de "dia 2" com sucesso pelo Fleet.

36.1.1.1 System Upgrade Controller (SUC)

Nota

Deve ser implantado em cada cluster downstream.

System Upgrade Controller é responsável por executar tarefas em nós especificados com base nos dados de configuração fornecidos por um recurso personalizado chamado `plano`.

O **SUC** é ativamente usado para fazer upgrade do sistema operacional e da distribuição Kubernetes.

Para obter mais informações sobre o componente **SUC** e como ele se adapta à pilha do Edge, consulte o *Capítulo 22, System Upgrade Controller*.

Para obter informações sobre como implantar o **SUC**, determine primeiro o seu caso de uso (*Seção 36.1.2, "Determinar seu caso de uso"*) e depois consulte Instalação do System Upgrade Controller – GitRepo (*Seção 22.2.1.1, "Instalação do System Upgrade Controller – GitRepo"*) ou Instalação do System Upgrade Controller – Bundle (*Seção 22.2.1.2, "Instalação do System Upgrade Controller – Bundle"*).

36.1.2 Determinar seu caso de uso

O Fleet usa dois tipos de recursos personalizados (https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/) para permitir o gerenciamento de recursos do Kubernetes e do Helm.

Veja a seguir as informações sobre a finalidade desses recursos e para quais casos de uso eles são mais adequados no contexto das operações de "dia 2".

36.1.2.1 GitRepo

O `GitRepo` é um recurso do Fleet (*Capítulo 8, Fleet*) que representa um repositório Git do qual o `Fleet` pode criar `Bundles`. Cada `Bundle` é criado com base nos caminhos de configuração definidos no recurso `GitRepo`. Para obter mais informações, consulte a documentação do GitRepo (https://fleet.rancher.io/gitrepo-add).

No contexto das operações de "dia 2", os recursos `GitRepo` costumam ser usados para implantar o `SUC` ou os `planos do SUC` em ambientes **não air-gapped** que usam a abordagem *GitOps do Fleet*.

Se preferir, use os recursos `GitRepo` para implantar o `SUC` ou os `planos do SUC` em ambientes **air-gapped**, **desde que você espelhe a configuração do seu repositório por um servidor Git local**.

36.1.2.2 Bundle

Os `Bundles` armazenam recursos **brutos** do Kubernetes que serão implantados no cluster de destino. Normalmente, eles são criados de um recurso `GitRepo`, mas há casos de uso em que podem ser implantados manualmente. Para obter mais informações, consulte a documentação do Bundle (https://fleet.rancher.io/bundle-add).

No contexto das operações de "dia 2", os recursos `Bundle` costumam ser usados para implantar o `SUC` ou `planos do SUC` em ambientes **air-gapped** que não usam nenhuma forma de procedimento do *GitOps local* (por exemplo, um **servidor git local**).

Se o seu caso de uso não possibilita um fluxo de trabalho *GitOps* (por exemplo, usando um repositório Git), a alternativa é usar os recursos `Bundle` para implantar o `SUC` ou os `planos do SUC` em ambientes **não air-gapped**.

36.1.3 Fluxo de trabalho de dia 2

Veja abaixo o fluxo de trabalho de "dia 2" que deve ser seguido para fazer upgrading de um cluster downstream para uma versão específica do Edge.

1. Upgrade de sistema operacional (*Seção 36.1.4, "Upgrade de sistema operacional"*)
2. Upgrade de versão do Kubernetes (*Seção 36.1.5, "Upgrade da versão do Kubernetes"*)
3. Upgrade de gráfico Helm (*Seção 36.1.6, "Upgrade do gráfico Helm"*)

36.1.4 Upgrade de sistema operacional

Esta seção descreve como fazer upgrade de um sistema operacional usando o *Capítulo 8, Fleet* e o *Capítulo 22, System Upgrade Controller*.

Os seguintes tópicos são abordados como parte desta seção:

1. *Seção 36.1.4.1, "Componentes"*: componentes adicionais usados pelo processo de upgrade.
2. *Seção 36.1.4.2, "Visão geral"*: visão geral do processo de upgrade.
3. *Seção 36.1.4.3, "Requisitos"*: requisitos do processo de upgrade.
4. *Seção 36.1.4.4, "Upgrade de sistema operacional: implantação do plano do SUC"*: informações de como implantar os `planos do SUC`, responsáveis por acionar o processo de upgrade.

36.1.4.1 Componentes

Esta seção aborda os componentes personalizados que o processo de `upgrade de sistema operacional` usa com os componentes de "dia 2" padrão (*Seção 36.1.1, "Componentes"*).

36.1.4.1.1 systemd.service

O upgrade de sistema operacional em um nó específico é executado pelo systemd.service (https://www.freedesktop.org/software/systemd/man/latest/systemd.service.html).

É criado um serviço diferente dependendo do tipo de upgrade que o sistema operacional requer de uma versão do Edge para outra:

- Para versões do Edge que requerem a mesma versão de sistema operacional (por exemplo, `6.0`), o `os-pkg-update.service` é criado. Ele usa o transactional-update (https://kubic.opensuse.org/documentation/man-pages/transactional-update.8.html) para fazer upgrade de um pacote normal (https://en.opensuse.org/SDB:Zypper_usage#Updating_packages).
- Para versões do Edge que requerem a migração da versão de sistema operacional (por exemplo, `6.0` → `6.1`), o `os-migration.service` é criado. Ele usa o transactional-update (https://kubic.opensuse.org/documentation/man-pages/transactional-update.8.html) para fazer o seguinte:
  a. O upgrade de pacote normal (https://en.opensuse.org/SDB:Zypper_usage#Updating_packages), que garante que todos os pacotes sejam atualizados para mitigar falhas na migração relacionadas a versões antigas dos pacotes.
  b. A migração de sistema operacional usando o comando `zypper migration`.

Os serviços mencionados acima são distribuídos para cada nó por meio do `plano do SUC`, que deve estar localizado no cluster downstream que precisa do upgrade de sistema operacional.

36.1.4.2 Visão geral

O upgrade do sistema operacional para nós de cluster downstream é feito pelo `Fleet` e pelo `System Upgrade Controller (SUC)`.

O **Fleet** é usado para implantar e gerenciar os `planos do SUC` no cluster desejado.

Nota

Os `planos do SUC` são recursos personalizados (https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/) que descrevem as etapas que o `SUC` precisa seguir para execução de uma tarefa específica em um conjunto de nós. Para ver um exemplo de como é um `plano do SUC`, consulte o repositório upstream (https://github.com/rancher/system-upgrade-controller?tab=readme-ov-file#example-plans).

Os `planos do SUC de sistema operacional` são enviados a cada cluster por meio da implantação de um recurso GitRepo (https://fleet.rancher.io/gitrepo-add) ou Bundle (https://fleet.rancher.io/bundle-add) em um espaço de trabalho (https://fleet.rancher.io/namespaces#gitrepos-bundles-clusters-clustergroups) do Fleet específico. O Fleet recupera o `GitRepo/Bundle` implantado e implanta seu conteúdo (os `planos do SUC de sistema operacional`) no(s) cluster(s) desejado(s).

Nota

Os recursos `GitRepo/Bundle` sempre são implantados no `cluster de gerenciamento`. O uso do recurso `GitRepo` ou `Bundle` depende do seu caso de uso. Consulte a *Seção 36.1.2, "Determinar seu caso de uso"* para obter mais informações.

Os `planos do SUC de sistema operacional` descrevem o seguinte fluxo de trabalho:

1. Sempre use o comando cordon (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_cordon/) nos nós antes dos upgrades de sistema operacional.
2. Sempre faça upgrade dos nós do `plano de controle` antes dos nós do `worker`.
3. Sempre faça upgrade do cluster **um** nó de cada vez.

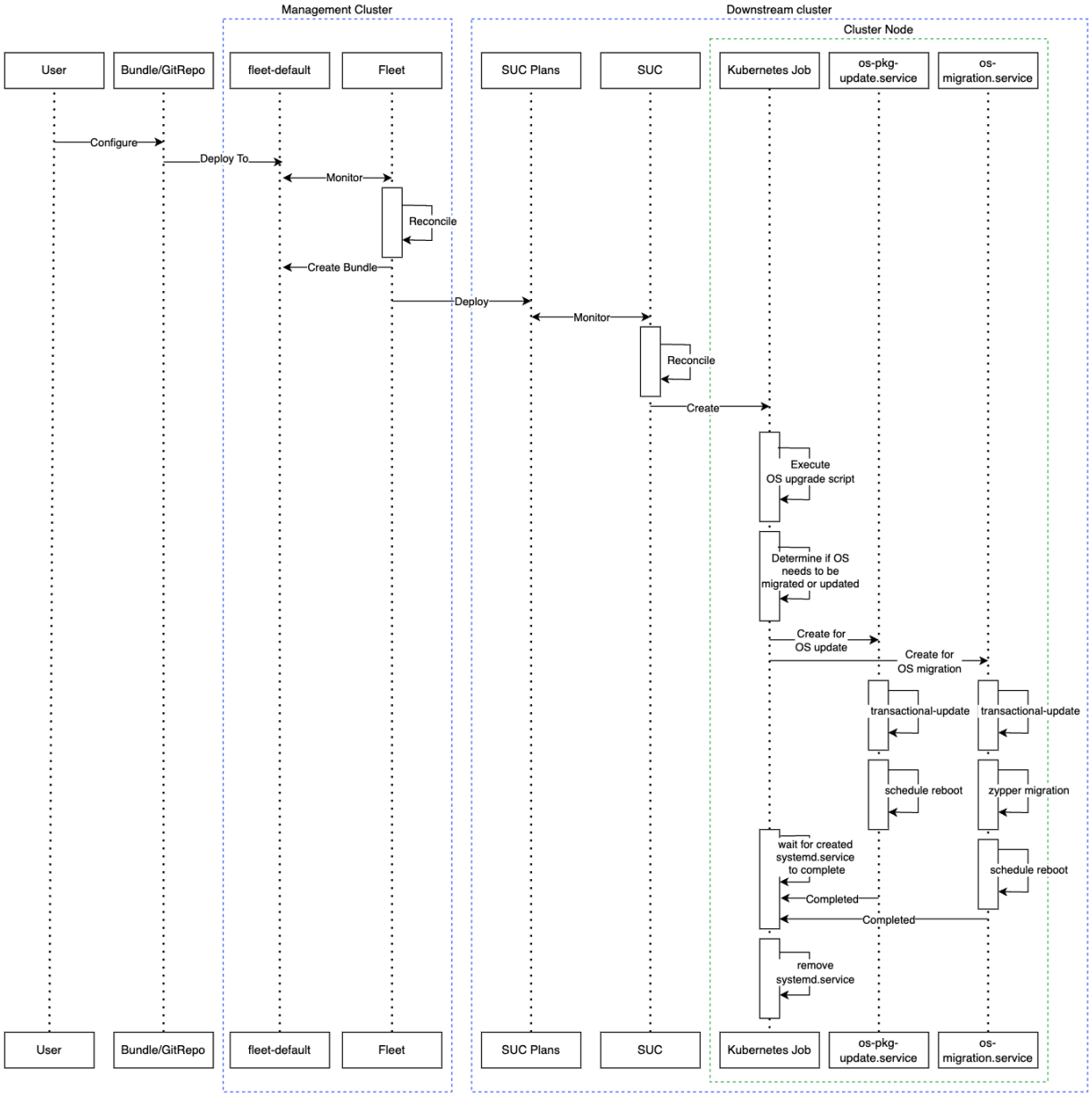
Depois que os `planos do SUC de sistema operacional` forem implantados, o fluxo de trabalho terá esta aparência:

1. O SUC reconcilia os `planos do SUC de sistema operacional` implantados e cria um `Kubernetes Job` em **cada nó**.
2. O `Kubernetes Job` cria um systemd.service (*Seção 36.1.4.1.1, "systemd.service"*) para upgrade de pacote ou migração de sistema operacional.
3. O `systemd.service` criado aciona o processo de upgrade de sistema operacional no nó específico.

Importante

Após o término do processo de upgrade do sistema operacional, o nó correspondente será `reinicializado` para aplicar as atualizações ao sistema.

Veja a seguir um diagrama da descrição acima:



36.1.4.3 Requisitos

Geral:

1. **Máquina registrada no SCC**: todos os nós do cluster downstream devem ser registrados no `https://scc.suse.com/` para que o respectivo `systemd.service` possa se conectar com sucesso ao repositório RPM desejado.

Importante

Para lançamentos do Edge que exigem a migração da versão do sistema operacional (por exemplo, `6.0` → `6.1`), verifique se a chave do SCC oferece suporte à migração para a nova versão.

2. **Garantir que as tolerâncias do plano do SUC correspondam às do nó**: se os nós do cluster Kubernetes têm **taints** personalizados, adicione tolerâncias (https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/) a esses taints nos **planos do SUC**. Por padrão, os **planos do SUC** têm tolerâncias apenas para nós do **plano de controle**. As tolerâncias padrão incluem:

   - *CriticalAddonsOnly=true:NoExecute*
   - *node-role.kubernetes.io/control-plane:NoSchedule*
   - *node-role.kubernetes.io/etcd:NoExecute*

Nota

As tolerâncias adicionais devem ser incluídas na seção `.spec.tolerations` de cada plano. Os **planos do SUC** relacionados a upgrade de sistema operacional estão disponíveis no repositório suse-edge/fleet-examples (https://github.com/suse-edge/fleet-examples) em `fleets/day2/system-upgrade-controller-plans/os-upgrade`. **Use os planos de uma tag de versão (https://github.com/suse-edge/fleet-examples/releases) válida do repositório.**

Um exemplo de definição de tolerâncias personalizadas para o plano do SUC de **plano de controle** tem esta aparência:

```
apiVersion: upgrade.cattle.io/v1
kind: Plan
metadata:
  name: os-upgrade-control-plane
spec:
  ...
  tolerations:
  # default tolerations
  - key: "CriticalAddonsOnly"
    operator: "Equal"
```

```
    value: "true"
    effect: "NoExecute"
  - key: "node-role.kubernetes.io/control-plane"
    operator: "Equal"
    effect: "NoSchedule"
  - key: "node-role.kubernetes.io/etcd"
    operator: "Equal"
    effect: "NoExecute"
  # custom toleration
  - key: "foo"
    operator: "Equal"
    value: "bar"
    effect: "NoSchedule"
...
```

Air-gapped:

1. **Espelhar repositórios RPM do SUSE**: os repositórios RPM de sistema operacional devem ser espelhados localmente para que o `systemd.service` tenha acesso a eles. Para isso, use RMT (https://documentation.suse.com/sles/15-SP6/html/SLES-all/book-rmt.html) ou SUMA (https://documentation.suse.com/suma/5.0/en/suse-manager/index.html).

36.1.4.4 Upgrade de sistema operacional: implantação do plano do SUC

Importante

Em ambientes que já passaram por upgrade usando esse procedimento, os usuários devem garantir que **uma** das seguintes etapas seja concluída:

- `Remover os planos do SUC que já foram implantados relacionados a versões de lançamento mais antigas do Edge do cluster downstream`: para fazer isso, remova o cluster desejado da configuração de destino (https://fleet.rancher.io/gitrepo-targets#target-matching) do `GitRepo/Bundle` existente ou remova o recurso `GitRepo/Bundle` completamente.
- `Reutilizar o recurso GitRepo/Bundle existente`: para fazer isso, aponte a revisão do recurso para uma nova tag que inclua as instâncias do Fleet corretas para a versão (https://github.com/suse-edge/fleet-examples/releases) desejada de `suse-edge/fleet-examples`.

Isso é feito para evitar conflitos entre os `planos do SUC` de versões de lançamento mais antigas do Edge.

Se os usuários tentarem fazer upgrade e já houver `planos do SUC` no cluster downstream, eles verão o seguinte erro no Fleet:

```
Not installed: Unable to continue with install: Plan <plan_name> in namespace
 <plan_namespace> exists and cannot be imported into the current release: invalid
 ownership metadata; annotation validation error..
```

Conforme mencionado na *Seção 36.1.4.2, "Visão geral"*, os upgrades de sistema operacional são feitos enviando os `planos do SUC` ao cluster desejado de uma destas maneiras:

- Recurso `GitRepo` do Fleet: *Seção 36.1.4.4.1, "Implantação do plano do SUC: recurso GitRepo"*.
- Recurso `Bundle` do Fleet: *Seção 36.1.4.4.2, "Implantação do plano do SUC – recurso Bundle"*.

Para determinar o recurso que você deve usar, consulte a *Seção 36.1.2, "Determinar seu caso de uso"*.

Para casos de uso de implantação dos `planos do SUC de sistema operacional` de uma ferramenta GitOps de terceiros, consulte a *Seção 36.1.4.4.3, "Implantação do plano do SUC: fluxo de trabalho do GitOps de terceiros"*.

36.1.4.4.1 Implantação do plano do SUC: recurso GitRepo

Um recurso **GitRepo**, que distribui os `planos do SUC de sistema operacional` necessários, pode ser implantado de uma das seguintes maneiras:

1. Pela `IU do Rancher`: *Seção 36.1.4.4.1.1, "Criação do GitRepo: IU do Rancher"* (quando o `Rancher` está disponível).
2. Pela implantação manual do recurso (*Seção 36.1.4.4.1.2, "Criação do GitRepo: manual"*) no `cluster de gerenciamento`.

Após a implantação, para monitorar o processo de upgrade do sistema operacional dos nós do cluster de destino, consulte a *Seção 22.3, "Monitorando os planos do System Upgrade Controller"*.

36.1.4.4.1.1 Criação do GitRepo: IU do Rancher

Para criar um recurso `GitRepo` pela IU do Rancher, siga a documentação (https://ranchermanager.docs.rancher.com/v2.11/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui) oficial dele.

A equipe do Edge mantém uma instância do Fleet (https://github.com/suse-edge/fleet-examples/tree/release-3.3.0/fleets/day2/system-upgrade-controller-plans/os-upgrade) pronta para uso. Dependendo do seu ambiente, ela pode ser usada diretamente ou como gabarito.

Importante

Use sempre essa instância do Fleet de uma tag de versão (https://github.com/suse-edge/fleet-examples/releases) válida do Edge.

Para casos de uso em que não há necessidade de incluir alterações personalizadas nos `planos do SUC` que a instância do Fleet envia, os usuários podem fazer referência à instância do Fleet de `os-upgrade` do repositório `suse-edge/fleet-examples`.

Nos casos em que as alterações personalizadas são necessárias (por exemplo, para adicionar tolerâncias personalizadas), os usuários devem fazer referência à instância do Fleet de `os-upgrade` de um repositório separado, para que possam adicioná-las aos planos do SUC conforme necessário.

Há um exemplo de como configurar o `GitRepo` para usar a instância do Fleet de um repositório `suse-edge/fleet-examples` disponível aqui (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/gitrepos/day2/os-upgrade-gitrepo.yaml).

36.1.4.4.1.2 Criação do GitRepo: manual

1. Obtenha o recurso **GitRepo**:

```
curl -o os-upgrade-gitrepo.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.3.0/gitrepos/day2/os-upgrade-gitrepo.yaml
```

2. Edite a configuração do **GitRepo** e especifique a lista de destinos desejados em `spec.targets`. Por padrão, os recursos `GitRepo` de `suse-edge/fleet-examples` **NÃO** são mapeados para clusters downstream.

- Para corresponder todos os clusters, altere o **destino** do `GitRepo` padrão para:

```
spec:
  targets:
  - clusterSelector: {}
```

- Se preferir uma seleção mais granular de clusters, consulte Mapping to Downstream Clusters (https://fleet.rancher.io/gitrepo-targets) (Mapeando para clusters downstream).

3. Aplique o recurso **GitRepo** ao `cluster de gerenciamento`:

```
kubectl apply -f os-upgrade-gitrepo.yaml
```

4. Visualize o recurso **GitRepo** criado no namespace `fleet-default`:

```
kubectl get gitrepo os-upgrade -n fleet-default

# Example output
NAME            REPO                                                COMMIT
 BUNDLEDEPLOYMENTS-READY   STATUS
os-upgrade      https://github.com/suse-edge/fleet-examples.git   release-3.3.0  0/0
```

36.1.4.4.2 Implantação do plano do SUC – recurso Bundle

O recurso **Bundle**, que envia os `planos do SUC de sistema operacional` necessários, pode ser implantado de uma das seguintes maneiras:

1. Pela `IU do Rancher`: *Seção 36.1.4.4.2.1, "Criação do bundle – IU do Rancher"* (quando o `Rancher` está disponível).

2. Pela implantação manual do recurso (*Seção 36.1.4.4.2.2, "Criação do bundle – manual"*) no `cluster de gerenciamento`.

Após a implantação, para monitorar o processo de upgrade do sistema operacional dos nós do cluster de destino, consulte a *Seção 22.3, "Monitorando os planos do System Upgrade Controller"*.

36.1.4.4.2.1 Criação do bundle – IU do Rancher

A equipe do Edge mantém um bundle (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/bundles/day2/system-upgrade-controller-plans/os-upgrade/os-upgrade-bundle.yaml) pronto para uso que pode ser usado nas etapas a seguir.

Importante

Sempre use esse bundle de uma tag de versão (https://github.com/suse-edge/fleet-examples/releases) válida do Edge.

Para criar um bundle pela IU do Rancher:

1. No canto superior esquerdo, clique em # → **Continuous Delivery** (Entrega contínua).
2. Vá para **Advanced** > **Bundles** (Avançado > Bundles).
3. Selecione **Create from YAML** (Criar do YAML).
4. Nesse local, você pode criar o bundle de uma das seguintes maneiras:

Nota

Há casos de uso em que você precisa incluir alterações personalizadas nos `planos do SUC` que o bundle envia (por exemplo, para adicionar tolerâncias personalizadas). Inclua essas alterações no bundle que será gerado pelas etapas a seguir.

a. Copie manualmente o conteúdo do bundle (https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.3.0/bundles/day2/system-upgrade-controller-plans/os-upgrade/os-upgrade-bundle.yaml) de `suse-edge/fleet-examples` para a página **Create from YAML** (Criar do YAML).

b. Clone o repositório suse-edge/fleet-examples (https://github.com/suse-edge/fleet-examples) da tag da versão (https://github.com/suse-edge/fleet-examples/releases) desejada e selecione a opção **Read from File** (Ler arquivo) na página **Create from YAML** (Criar do YAML). Dessa página, navegue até o local do bundle (`bundles/day2/system-upgrade-controller-plans/os-upgrade`) e selecione o arquivo do bundle. Esse procedimento preencherá automaticamente a página **Create from YAML** (Criar do YAML) com o conteúdo do bundle.

5. Altere os clusters de **destino** para o `Bundle`:

    - Para corresponder todos os clusters downstream, altere o bundle padrão `.spec.targets` para:

    ```
    spec:
      targets:
      - clusterSelector: {}
    ```

    - Para mapeamentos mais granulares de clusters downstream, consulte Mapping to Downstream Clusters (https://fleet.rancher.io/gitrepo-targets) (Mapeando para clusters downstream).

6. Selecione **Create** (Criar).

36.1.4.4.2.2 Criação do bundle – manual

1. Obtenha o recurso **Bundle**:

    ```
    curl -o os-upgrade-bundle.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.3.0/bundles/day2/system-upgrade-controller-plans/os-upgrade/os-upgrade-bundle.yaml
    ```

2. Edite as configurações de **destino** do `Bundle`. Em `spec.targets`, insira a lista de destinos desejada. Por padrão, os recursos `Bundle` de `suse-edge/fleet-examples` **NÃO** são mapeados para clusters downstream.

    - Para corresponder todos os clusters, altere o **destino** do `Bundle` padrão para:

    ```
    spec:
      targets:
      - clusterSelector: {}
    ```

    - Se preferir uma seleção mais granular de clusters, consulte Mapping to Downstream Clusters (https://fleet.rancher.io/gitrepo-targets) (Mapeando para clusters downstream).

3. Aplique o recurso **Bundle** ao seu `cluster de gerenciamento`:

```
kubectl apply -f os-upgrade-bundle.yaml
```

4. Visualize o recurso **Bundle** criado no namespace `fleet-default`:

```
kubectl get bundles -n fleet-default
```

36.1.4.4.3 Implantação do plano do SUC: fluxo de trabalho do GitOps de terceiros

Em alguns casos de uso, talvez você queira incorporar os `planos do SUC de sistema operacional` ao fluxo de trabalho do GitOps de terceiros (por exemplo, o `Flux`).

Para obter os recursos de upgrade de sistema operacional que você precisa, primeiro determine a tag da versão (https://github.com/suse-edge/fleet-examples/releases) do Edge do repositório suse-edge/fleet-examples (https://github.com/suse-edge/fleet-examples) que deseja usar.

Depois disso, os recursos estarão disponíveis em `fleets/day2/system-upgrade-controller-plans/os-upgrade`, em que:

- `plan-control-plane.yaml` é um recurso do plano do SUC para nós do **plano de controle**.
- `plan-worker.yaml` é um recurso do plano do SUC para nós do **worker**.
- `secret.yaml` é um segredo que contém o script `upgrade.sh`, responsável por criar o systemd.service (*Seção 36.1.4.1.1, "systemd.service"*).
- `config-map.yaml` é um ConfigMap que armazena as configurações consumidas pelo script `upgrade.sh`.

Importante

Esses recursos do `plano` são interpretados pelo `System Upgrade Controller` e devem ser implantados em cada cluster downstream do qual você deseja fazer upgrade. Para obter informações sobre a implantação do SUC, consulte a *Seção 22.2, "Instalando o System Upgrade Controller"*.

Para entender melhor como usar o fluxo de trabalho do GitOps para implantar os **planos do SUC** para upgrade de sistema operacional, consulte a visão geral (*Seção 36.1.4.2, "Visão geral"*).

36.1.5 Upgrade da versão do Kubernetes

Importante

Esta seção aborda os upgrades do Kubernetes para clusters downstream que **NÃO** foram criados por uma instância do Rancher (*Capítulo 5, Rancher*). Para obter informações sobre como fazer upgrade da versão do Kubernetes dos clusters criados pelo `Rancher`, consulte Upgrading and Rolling Back Kubernetes (https://ranchermanager.docs.rancher.com/v2.11/getting-started/installation-and-upgrade/upgrade-and-roll-back-kubernetes#upgrading-the-kubernetes-version) (Fazendo upgrade e rollback do Kubernetes).

Esta seção descreve como fazer um upgrade do Kubernetes usando o *Capítulo 8, Fleet* e o *Capítulo 22, System Upgrade Controller*.

Os seguintes tópicos são abordados como parte desta seção:

1. *Seção 36.1.5.1, "Componentes"*: componentes adicionais usados pelo processo de upgrade.
2. *Seção 36.1.5.2, "Visão geral"*: visão geral do processo de upgrade.
3. *Seção 36.1.5.3, "Requisitos"*: requisitos do processo de upgrade.
4. *Seção 36.1.5.4, "Upgrade do K8s: implantação do plano do SUC"*: informações de como implantar os `planos do SUC`, responsáveis por acionar o processo de upgrade.

36.1.5.1 Componentes

Esta seção aborda os componentes personalizados que o processo de `upgrade do K8s` usa com os componentes de "dia 2" padrão (*Seção 36.1.1, "Componentes"*).

36.1.5.1.1 rke2-upgrade

Imagem do contêiner responsável por fazer upgrade da versão do RKE2 de um nó específico.

Incluída em um pod criado pelo **SUC** com base no **plano do SUC**. O plano deve estar localizado em cada **cluster** que precisa de upgrade do RKE2.

Para obter mais informações sobre como a imagem `rke2-upgrade` faz o upgrade, consulte a documentação upstream (https://github.com/rancher/rke2-upgrade/tree/master).

36.1.5.1.2 k3s-upgrade

Imagem do contêiner responsável por fazer upgrade da versão do K3s de um nó específico.

Incluída em um pod criado pelo **SUC** com base no **plano do SUC**. O plano deve estar localizado em cada **cluster** que precisa de upgrade do K3s.

Para obter mais informações sobre como a imagem `k3s-upgrade` faz o upgrade, consulte a documentação upstream (https://github.com/k3s-io/k3s-upgrade).

36.1.5.2 Visão geral

O upgrade de distribuições Kubernetes para nós de cluster downstream é feito pelo `Fleet` e pelo `System Upgrade Controller (SUC)`.

O `Fleet` é usado para implantar e gerenciar `planos do SUC` no cluster desejado.

Nota

Os `planos do SUC` são recursos personalizados (https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/) que descrevem as etapas que o **SUC** precisa seguir para execução de uma tarefa específica em um conjunto de nós. Para ver um exemplo de como é um `plano do SUC`, consulte o repositório upstream (https://github.com/rancher/system-upgrade-controller?tab=readme-ov-file#example-plans).

Os `planos do SUC do K8s` são enviados a cada cluster por meio da implantação de um recurso GitRepo (https://fleet.rancher.io/gitrepo-add) ou Bundle (https://fleet.rancher.io/bundle-add) em um espaço de trabalho (https://fleet.rancher.io/namespaces#gitrepos-bundles-clusters-clustergroups) do Fleet específico. O Fleet recupera o `GitRepo/Bundle` implantado e implanta seu conteúdo (os `planos do SUC do K8s`) no(s) cluster(s) desejado(s).

Nota

Os recursos `GitRepo/Bundle` sempre são implantados no `cluster de gerenciamento`. O uso do recurso `GitRepo` ou `Bundle` depende do seu caso de uso. Consulte a *Seção 36.1.2, "Determinar seu caso de uso"* para obter mais informações.

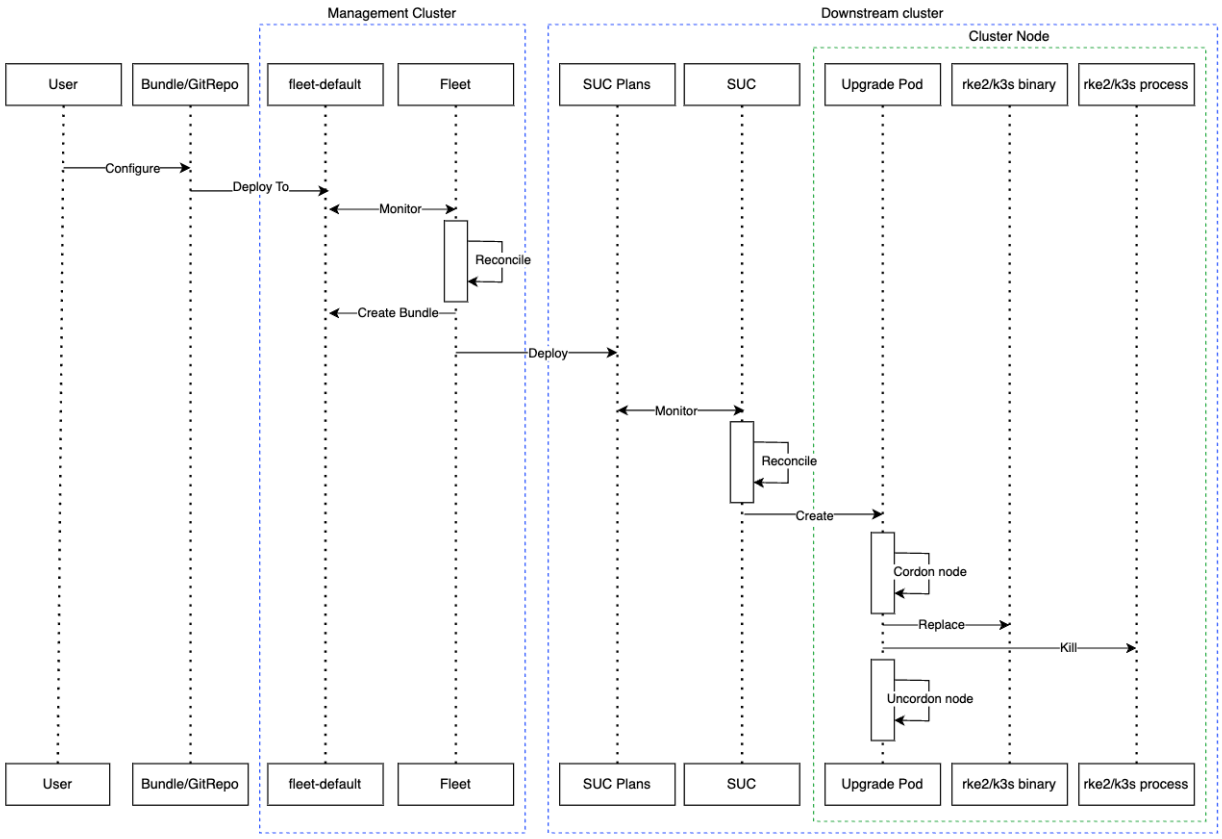
Os `planos do SUC do K8s` descrevem o seguinte fluxo de trabalho:

1. Sempre use o comando cordon (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_cordon/) nos nós antes dos upgrades do K8s.
2. Sempre faça upgrade dos nós do `plano de controle` antes dos nós do `worker`.
3. Sempre faça upgrade dos nós do `plano de controle` **um** de cada vez, e dos nós do `worker` **dois** nós de cada vez.

Depois que os `planos do SUC do K8s` forem implantados, o fluxo de trabalho terá esta aparência:

1. O SUC reconcilia os `planos do SUC do K8s` implantados e cria um `Kubernetes Job` em **cada nó**.
2. Dependendo da distribuição Kubernetes, o job cria um pod que executa a imagem de contêiner do rke2-upgrade (*Seção 36.1.5.1.1, "rke2-upgrade"*) ou do k3s-upgrade (*Seção 36.1.5.1.2, "k3s-upgrade"*).
3. O pod criado vai percorrer o seguinte fluxo de trabalho:
    a. Substitua o binário `rke2/k3s` existente no nó pelo da imagem `rke2-upgrade/k3s-upgrade`.
    b. Cancele o processo do `rke2/k3s` em execução.
4. O cancelamento do processo do `rke2/k3s` aciona a reinicialização, o que inicia um novo processo que executa o binário atualizado, resultando em uma versão atualizada da distribuição Kubernetes.

Veja a seguir um diagrama da descrição acima:



36.1.5.3 Requisitos

1. **Faça backup da distribuição Kubernetes:**

    a. Para **clusters RKE2**, consulte a documentação sobre backup e restauração do RKE2 (https://docs.rke2.io/datastore/backup_restore).

    b. Para **clusters K3s**, consulte a documentação sobre backup e restauração do K3s (https://docs.k3s.io/datastore/backup-restore).

2. **Garantir que as tolerâncias do plano do SUC correspondam às do nó**: se os nós do cluster Kubernetes têm **taints** personalizados, adicione tolerâncias (https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/) a esses taints nos **planos do SUC**. Por padrão, os **planos do SUC** têm tolerâncias apenas para nós do **plano de controle**. As tolerâncias padrão incluem:

    - *CriticalAddonsOnly = true:NoExecute*
    - *node-role.kubernetes.io/control-plane:NoSchedule*
    - *node-role.kubernetes.io/etcd:NoExecute*

    Nota

    As tolerâncias adicionais devem ser incluídas na seção `.spec.tolerations` de cada plano. Os **planos do SUC** relacionados ao upgrade de versões do Kubernetes estão disponíveis no repositório suse-edge/fleet-examples (https://github.com/suse-edge/fleet-examples) em:

    - Para o **RKE2**: `fleets/day2/system-upgrade-controller-plans/rke2-upgrade`
    - Para o **K3s**: `fleets/day2/system-upgrade-controller-plans/k3s-upgrade`

    Use os planos de uma tag de versão (https://github.com/suse-edge/fleet-examples/releases) válida do repositório.

    Um exemplo de definição de tolerâncias personalizadas para o plano do SUC de **plano de controle** do RKE2 tem esta aparência:

    ```
    apiVersion: upgrade.cattle.io/v1
    ```

```
kind: Plan
metadata:
  name: rke2-upgrade-control-plane
spec:
  ...
  tolerations:
  # default tolerations
  - key: "CriticalAddonsOnly"
    operator: "Equal"
    value: "true"
    effect: "NoExecute"
  - key: "node-role.kubernetes.io/control-plane"
    operator: "Equal"
    effect: "NoSchedule"
  - key: "node-role.kubernetes.io/etcd"
    operator: "Equal"
    effect: "NoExecute"
  # custom toleration
  - key: "foo"
    operator: "Equal"
    value: "bar"
    effect: "NoSchedule"
...
```

36.1.5.4 Upgrade do K8s: implantação do plano do SUC

Importante

Em ambientes que já passaram por upgrade usando esse procedimento, os usuários devem garantir que **uma** das seguintes etapas seja concluída:

- `Remover os planos do SUC que já foram implantados relacionados a versões de lançamento mais antigas do Edge do cluster downstream`: para fazer isso, remova o cluster desejado da configuração de destino (https://fleet.rancher.io/gitrepo-targets#target-matching) do `GitRepo/Bundle` existente ou remova o recurso `GitRepo/Bundle` completamente.
- `Reutilizar o recurso GitRepo/Bundle existente`: para fazer isso, aponte a revisão do recurso para uma nova tag que inclua as instâncias do Fleet corretas para a versão (https://github.com/suse-edge/fleet-examples/releases) desejada de `suse-edge/fleet-examples`.

Isso é feito para evitar conflitos entre os `planos do SUC` de versões de lançamento mais antigas do Edge.

Se os usuários tentarem fazer upgrade e já houver `planos do SUC` no cluster downstream, eles verão o seguinte erro no Fleet:

```
Not installed: Unable to continue with install: Plan <plan_name> in namespace
 <plan_namespace> exists and cannot be imported into the current release: invalid
 ownership metadata; annotation validation error..
```

Conforme mencionado na *Seção 36.1.5.2, "Visão geral"*, os upgrades do Kubernetes são feitos enviando os `planos do SUC` ao cluster desejado de uma destas maneiras:

- Recurso GitRepo do Fleet (*Seção 36.1.5.4.1, "Implantação do plano do SUC: recurso GitRepo"*)
- Recurso Bundle do Fleet (*Seção 36.1.5.4.2, "Implantação do plano do SUC – recurso Bundle"*)

Para determinar o recurso que você deve usar, consulte a *Seção 36.1.2, "Determinar seu caso de uso"*.

Para casos de uso de implantação dos `planos do SUC do K8s` de uma ferramenta GitOps de terceiros, consulte a *Seção 36.1.5.4.3, "Implantação do plano do SUC: fluxo de trabalho do GitOps de terceiros"*.

36.1.5.4.1 Implantação do plano do SUC: recurso GitRepo

Um recurso **GitRepo**, que distribui os `planos do SUC do K8s` necessários, pode ser implantado de uma das seguintes maneiras:

1. Pela `IU do Rancher`: *Seção 36.1.5.4.1.1, "Criação do GitRepo: IU do Rancher"* (quando o `Rancher` está disponível).
2. Pela implantação manual do recurso (*Seção 36.1.5.4.1.2, "Criação do GitRepo: manual"*) no `cluster de gerenciamento`.

Após a implantação, para monitorar o processo de upgrade do Kubernetes dos nós do cluster de destino, consulte a *Seção 22.3, "Monitorando os planos do System Upgrade Controller"*.

36.1.5.4.1.1 Criação do GitRepo: IU do Rancher

Para criar um recurso `GitRepo` pela IU do Rancher, siga a documentação (https://ranchermanager.docs.rancher.com/v2.11/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui) oficial dele.

A equipe do Edge mantém instâncias do Fleet prontas para uso para distribuições Kubernetes tanto do rke2 (https://github.com/suse-edge/fleet-examples/tree/release-3.3.0/fleets/day2/system-upgrade-controller-plans/rke2-upgrade) quanto do k3s (https://github.com/suse-edge/fleet-examples/tree/release-3.3.0/fleets/day2/system-upgrade-controller-plans/k3s-upgrade). Dependendo do ambiente, a instância do Fleet pode ser usada diretamente ou como gabarito.

Importante

Sempre use as instâncias do Fleet de uma tag de versão (https://github.com/suse-edge/fleet-examples/releases) válida do Edge.

Para casos de uso em que não há necessidade de incluir alterações personalizadas nos `planos do SUC` que as instâncias do Fleet envia, os usuários podem fazer referência a essas instâncias do Fleet diretamente do repositório `suse-edge/fleet-examples`.

Nos casos em que as alterações personalizadas são necessárias (por exemplo, para adicionar tolerâncias personalizadas), os usuários devem fazer referência às instâncias do Fleet de um repositório separado, para que possam adicionas as alterações aos planos do SUC conforme necessário.

Exemplos de configuração do recurso `GitRepo` usando as instâncias do Fleet do repositório `suse-edge/fleet-examples`:

- RKE2 (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/gitrepos/day2/rke2-upgrade-gitrepo.yaml)
- K3s (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/gitrepos/day2/k3s-upgrade-gitrepo.yaml)

36.1.5.4.1.2 Criação do GitRepo: manual

1. Obtenha o recurso **GitRepo**:

    - Para clusters **RKE2**:

    ```
    curl -o rke2-upgrade-gitrepo.yaml https://raw.githubusercontent.com/suse-edge/
    fleet-examples/refs/tags/release-3.3.0/gitrepos/day2/rke2-upgrade-gitrepo.yaml
    ```

    - Para clusters **K3s**:

    ```
    curl -o k3s-upgrade-gitrepo.yaml https://raw.githubusercontent.com/suse-edge/
    fleet-examples/refs/tags/release-3.3.0/gitrepos/day2/k3s-upgrade-gitrepo.yaml
    ```

2. Edite a configuração do **GitRepo** e especifique a lista de destinos desejados em `spec.targets`. Por padrão, os recursos `GitRepo` de `suse-edge/fleet-examples` **NÃO** são mapeados para clusters downstream.

    - Para corresponder todos os clusters, altere o **destino** do `GitRepo` padrão para:

    ```
    spec:
      targets:
      - clusterSelector: {}
    ```

    - Se preferir uma seleção mais granular de clusters, consulte Mapping to Downstream Clusters (https://fleet.rancher.io/gitrepo-targets) (Mapeando para clusters downstream).

3. Aplique os recursos **GitRepo** ao `cluster de gerenciamento`:

    ```
    # RKE2
    kubectl apply -f rke2-upgrade-gitrepo.yaml

    # K3s
    kubectl apply -f k3s-upgrade-gitrepo.yaml
    ```

4. Visualize o recurso **GitRepo** criado no namespace `fleet-default`:

    ```
    # RKE2
    kubectl get gitrepo rke2-upgrade -n fleet-default

    # K3s
    kubectl get gitrepo k3s-upgrade -n fleet-default

    # Example output
    ```

```
NAME           REPO                                                COMMIT
 BUNDLEDEPLOYMENTS-READY   STATUS
k3s-upgrade    https://github.com/suse-edge/fleet-examples.git   fleet-default   0/0
rke2-upgrade   https://github.com/suse-edge/fleet-examples.git   fleet-default   0/0
```

36.1.5.4.2 Implantação do plano do SUC – recurso Bundle

O recurso **Bundle**, que envia os `planos do SUC de upgrade do Kubernetes` necessários, pode ser implantado de uma das seguintes maneiras:

1. Pela `IU do Rancher`: *Seção 36.1.5.4.2.1, "Criação do bundle – IU do Rancher"* (quando o `Rancher` está disponível).
2. Pela implantação manual do recurso (*Seção 36.1.5.4.2.2, "Criação do bundle – manual"*) no `cluster de gerenciamento`.

Após a implantação, para monitorar o processo de upgrade do Kubernetes dos nós do cluster de destino, consulte a *Seção 22.3, "Monitorando os planos do System Upgrade Controller"*.

36.1.5.4.2.1 Criação do bundle – IU do Rancher

A equipe do Edge mantém bundles prontos para uso para distribuições Kubernetes tanto do rke2 (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/bundles/day2/system-upgrade-controller-plans/rke2-upgrade/plan-bundle.yaml) quanto do k3s (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/bundles/day2/system-upgrade-controller-plans/k3s-upgrade/plan-bundle.yaml). Dependendo do ambiente, esses bundles podem ser usados diretamente ou como gabaritos.

Importante

Sempre use esse bundle de uma tag de versão (https://github.com/suse-edge/fleet-examples/releases) válida do Edge.

Para criar um bundle pela IU do Rancher:

1. No canto superior esquerdo, clique em # → **Continuous Delivery** (Entrega contínua).
2. Vá para **Advanced** > **Bundles** (Avançado > Bundles).

3. Selecione **Create from YAML** (Criar do YAML).

4. Nesse local, você pode criar o bundle de uma das seguintes maneiras:

Nota

Há casos de uso em que você precisa incluir alterações personalizadas nos `planos do SUC` que o bundle envia (por exemplo, para adicionar tolerâncias personalizadas). Inclua essas alterações no bundle que será gerado pelas etapas a seguir.

   a. Copie manualmente o conteúdo do bundle referente ao RKE2 (https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.3.0/bundles/day2/system-upgrade-controller-plans/rke2-upgrade/plan-bundle.yaml) ou ao K3s (https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.3.0/bundles/day2/system-upgrade-controller-plans/k3s-upgrade/plan-bundle.yaml) de `suse-edge/fleet-examples` para a página **Create from YAML** (Criar do YAML).

   b. Clone o repositório suse-edge/fleet-examples (https://github.com/suse-edge/fleet-examples.git) da tag da versão (https://github.com/suse-edge/fleet-examples/releases) desejada e selecione a opção **Read from File** (Ler arquivo) na página **Create from YAML** (Criar do YAML). Dessa página, navegue até o bundle necessário (`bundles/day2/system-upgrade-controller-plans/rke2-upgrade/plan-bundle.yaml` para RKE2 e `bundles/day2/system-upgrade-controller-plans/k3s-upgrade/plan-bundle.yaml` para K3s). Esse procedimento preencherá automaticamente a página **Create from YAML** (Criar do YAML) com o conteúdo do bundle.

5. Altere os clusters de **destino** para o `Bundle`:

   - Para corresponder todos os clusters downstream, altere o bundle padrão `.spec.targets` para:

```
spec:
  targets:
```

```
  - clusterSelector: {}
```

- Para mapeamentos mais granulares de clusters downstream, consulte Mapping to Downstream Clusters (https://fleet.rancher.io/gitrepo-targets) (Mapeando para clusters downstream).

6. Selecione **Create** (Criar).

36.1.5.4.2.2 Criação do bundle – manual

1. Obtenha os recursos **Bundle**:

   - Para clusters **RKE2**:

   ```
   curl -o rke2-plan-bundle.yaml https://raw.githubusercontent.com/suse-edge/
   fleet-examples/refs/tags/release-3.3.0/bundles/day2/system-upgrade-controller-
   plans/rke2-upgrade/plan-bundle.yaml
   ```

   - Para clusters **K3s**:

   ```
   curl -o k3s-plan-bundle.yaml https://raw.githubusercontent.com/suse-edge/fleet-
   examples/refs/tags/release-3.3.0/bundles/day2/system-upgrade-controller-plans/
   k3s-upgrade/plan-bundle.yaml
   ```

2. Edite as configurações de **destino** do `Bundle`. Em `spec.targets`, insira a lista de destinos desejada. Por padrão, os recursos `Bundle` de `suse-edge/fleet-examples` **NÃO** são mapeados para clusters downstream.

   - Para corresponder todos os clusters, altere o **destino** do `Bundle` padrão para:

   ```
   spec:
     targets:
     - clusterSelector: {}
   ```

   - Se preferir uma seleção mais granular de clusters, consulte Mapping to Downstream Clusters (https://fleet.rancher.io/gitrepo-targets) (Mapeando para clusters downstream).

3. Aplique os recursos **Bundle** ao `cluster de gerenciamento`:

   ```
   # For RKE2
   kubectl apply -f rke2-plan-bundle.yaml
   ```

```
# For K3s
kubectl apply -f k3s-plan-bundle.yaml
```

4. Visualize o recurso **Bundle** criado no namespace `fleet-default`:

```
# For RKE2
kubectl get bundles rke2-upgrade -n fleet-default

# For K3s
kubectl get bundles k3s-upgrade -n fleet-default

# Example output
NAME           BUNDLEDEPLOYMENTS-READY   STATUS
k3s-upgrade    0/0
rke2-upgrade   0/0
```

36.1.5.4.3 Implantação do plano do SUC: fluxo de trabalho do GitOps de terceiros

Em alguns casos de uso, talvez você queira incorporar os `planos do SUC de upgrade do Kubernetes` ao fluxo de trabalho do GitOps de terceiros (por exemplo, o `Flux`).

Para obter os recursos de upgrade do K8s upgrade que você precisa, primeiro determine a tag da versão (https://github.com/suse-edge/fleet-examples/releases) do Edge do repositório suse-edge/fleet-examples (https://github.com/suse-edge/fleet-examples.git) que deseja usar.

Depois disso, os recursos estarão em:

- Para ugrade do cluster RKE2:
  - Para os nós do `plano de controle`: `fleets/day2/system-upgrade-controller-plans/rke2-upgrade/plan-control-plane.yaml`
  - Para os nós de `worker`: `fleets/day2/system-upgrade-controller-plans/rke2-upgrade/plan-worker.yaml`
- Para upgrade do cluster K3s:
  - Para os nós do `plano de controle`: `fleets/day2/system-upgrade-controller-plans/k3s-upgrade/plan-control-plane.yaml`
  - Para os nós de `worker`: `fleets/day2/system-upgrade-controller-plans/k3s-upgrade/plan-worker.yaml`

Importante

Esses recursos do `plano` são interpretados pelo `System Upgrade Controller` e devem ser implantados em cada cluster downstream do qual você deseja fazer upgrade. Para obter informações sobre a implantação do SUC, consulte a *Seção 22.2, "Instalando o System Upgrade Controller"*.

Para entender melhor como usar o fluxo de trabalho do GitOps para implantar os **planos do SUC** para upgrade de versão do Kubernetes, consulte a visão geral (*Seção 36.1.5.2, "Visão geral"*) do procedimento de atualização com o `Fleet`.

36.1.6 Upgrade do gráfico Helm

Esta seção aborda as seguintes partes:

1. *Seção 36.1.6.1, "Preparação para ambientes air-gapped"*: armazena informações sobre como enviar gráficos e imagens OCI relacionados ao Edge para seu registro particular.
2. *Seção 36.1.6.2, "Procedimento de upgrade"*: armazena informações sobre diversos casos de uso de upgrade de gráficos Helm e o respectivo procedimento de upgrade.

36.1.6.1 Preparação para ambientes air-gapped

36.1.6.1.1 Garantir que você tenha acesso ao Fleet por gráfico Helm

Dependendo da compatibilidade do seu ambiente, você poderá seguir uma destas opções:

1. Hospede os recursos do Fleet do seu gráfico em um servidor Git local que possa ser acessado pelo `cluster de gerenciamento`.
2. Use a CLI do Fleet para converter um gráfico Helm em bundle (https://fleet.rancher.io/bundle-add#convert-a-helm-chart-into-a-bundle), que você pode usar diretamente e não precisa ser hospedado em nenhum lugar. É possível recuperar a CLI do Fleet da página referente à respectiva versão (https://github.com/rancher/fleet/releases/tag/v0.12.2). Para usuários do Mac, existe um Homebrew Formulae fleet-cli (https://formulae.brew.sh/formula/fleet-cli).

36.1.6.1.2 Encontrar os ativos necessários à sua versão de lançamento do Edge

1. Vá para a página de versões (https://github.com/suse-edge/fleet-examples/releases) do "dia 2" e localize a versão do Edge para a qual você quer fazer upgrade do gráfico e clique em **Assets** (Ativos).

2. Na seção "**Assets**" (Ativos), faça download dos seguintes arquivos:

Arquivo de versão	Descrição
edge-save-images.sh	Obtém as imagens especificadas no arquivo `edge-release-images.txt` e as compacta em um arquivo ".tar.gz".
edge-save-oci-artefacts.sh	Obtém as imagens de gráfico OCI relacionadas à versão especifica do Edge e as compacta em um arquivo ".tar.gz".
edge-load-images.sh	Carrega as imagens de um arquivo ".tar.gz", faz a remarcação e as envia a um registro particular.
edge-load-oci-artefacts.sh	Detecta o diretório que contém os pacotes ".tgz" de gráficos OCI do Edge e os carrega em um registro particular.
edge-release-helm-oci-artefacts.txt	Contém uma lista de imagens de gráfico OCI relacionadas a uma versão específica do Edge.
edge-release-images.txt	Contém uma lista de imagens relacionadas a uma versão específica do Edge.

36.1.6.1.3 Criar o arquivo de imagens da versão do Edge

Em uma máquina com acesso à Internet:

1. Torne o `edge-save-images.sh` executável:

```
chmod +x edge-save-images.sh
```

2. Gere o arquivo de imagens:

```
./edge-save-images.sh --source-registry registry.suse.com
```

3. Esse procedimento cria um arquivo pronto para carregamento chamado `edge-images.tar.gz`.

Nota

Se a opção `-i|--images` foi especificada, o nome do arquivo pode ser diferente.

4. Copie esse arquivo para sua máquina **air-gapped**:

```
scp edge-images.tar.gz <user>@<machine_ip>:/path
```

36.1.6.1.4 Criar o arquivo de imagens de gráfico OCI do Edge

Em uma máquina com acesso à Internet:

1. Torne o `edge-save-oci-artefacts.sh` executável:

```
chmod +x edge-save-oci-artefacts.sh
```

2. Gere o arquivo de imagens de gráfico OCI:

```
./edge-save-oci-artefacts.sh --source-registry registry.suse.com
```

3. Esse procedimento cria um arquivo chamado `oci-artefacts.tar.gz`.

Nota

Se a opção `-a|--archive` foi especificada, o nome do arquivo pode ser diferente.

4. Copie esse arquivo para sua máquina **air-gapped**:

```
scp oci-artefacts.tar.gz <user>@<machine_ip>:/path
```

36.1.6.1.5 Carregar as imagens da versão do Edge para sua máquina air-gapped

Na máquina air-gapped:

1. Faça login no seu registro particular (se necessário):

```
podman login <REGISTRY.YOURDOMAIN.COM:PORT>
```

2. Torne o `edge-load-images.sh` executável:

```
chmod +x edge-load-images.sh
```

3. Execute o script, especificando o arquivo já **copiado** `edge-images.tar.gz`:

```
./edge-load-images.sh --source-registry registry.suse.com --registry
 <REGISTRY.YOURDOMAIN.COM:PORT> --images edge-images.tar.gz
```

Nota

Desse modo, todas as imagens do `edge-images.tar.gz` serão carregadas, remarcadas e enviadas ao registro especificado na opção `--registry`.

36.1.6.1.6 Carregar as imagens de gráfico OCI do Edge em sua máquina air-gapped

Na máquina air-gapped:

1. Faça login no seu registro particular (se necessário):

```
podman login <REGISTRY.YOURDOMAIN.COM:PORT>
```

2. Torne o `edge-load-oci-artefacts.sh` executável:

```
chmod +x edge-load-oci-artefacts.sh
```

3. Descompacte o arquivo `oci-artefacts.tar.gz` copiado:

```
tar -xvf oci-artefacts.tar.gz
```

4. Isso cria um diretório com o gabarito de nomenclatura `edge-release-oci-tgz-<data>`.

5. Especifique esse diretório no script `edge-load-oci-artefacts.sh` para carregar as imagens de gráfico OCI do Edge em seu registro particular:

Nota

Esse script assume que a CLI `helm` foi pré-instalada em seu ambiente. Para obter instruções de instalação do Helm, consulte Installing Helm (https://helm.sh/docs/intro/install/) (Instalando o Helm).

```
./edge-load-oci-artefacts.sh --archive-directory edge-release-oci-tgz-<date> --
registry <REGISTRY.YOURDOMAIN.COM:PORT> --source-registry registry.suse.com
```

36.1.6.1.7 Configurar o registro particular na distribuição Kubernetes

Para RKE2, consulte Private Registry Configuration (https://docs.rke2.io/install/private_registry) (Configuração de registro particular)

Para K3s, consulte Private Registry Configuration (https://docs.k3s.io/installation/private-registry) (Configuração de registro particular)

36.1.6.2 Procedimento de upgrade

Esta seção tem como foco os seguintes casos de uso do procedimento de upgrade do Helm:

1. *Seção 36.1.6.2.1, "Eu tenho um novo cluster e desejo implantar e gerenciar um gráfico Helm do Edge"*
2. *Seção 36.1.6.2.2, "Quero fazer upgrade de um gráfico Helm gerenciado pelo Fleet"*
3. *Seção 36.1.6.2.3, "Quero fazer upgrade de um gráfico Helm implantado pelo EIB"*

Importante

Não é possível fazer upgrade confiável de gráficos Helm implantados manualmente. Sugerimos reimplantá-los usando o método da *Seção 36.1.6.2.1, "Eu tenho um novo cluster e desejo implantar e gerenciar um gráfico Helm do Edge"*.

36.1.6.2.1 Eu tenho um novo cluster e desejo implantar e gerenciar um gráfico Helm do Edge

Esta seção aborda o seguinte:

1. *Seção 36.1.6.2.1.1, "Preparar os recursos do Fleet para seu gráfico"*.
2. *Seção 36.1.6.2.1.2, "Implantar o Fleet para seu gráfico"*.
3. *Seção 36.1.6.2.1.3, "Gerenciar o gráfico Helm implantado"*.

36.1.6.2.1.1 Preparar os recursos do Fleet para seu gráfico

1. Adquira os recursos do Fleet do gráfico com base na tag da versão (https://github.com/suse-edge/fleet-examples/releases) do Edge que deseja usar.
2. Navegue até a instância do Fleet do gráfico Helm (`fleets/day2/chart-templates/<gráfico>`).
3. **Se você pretende usar um fluxo de trabalho do GitOps**, copie o diretório do Fleet do gráfico para o repositório Git do qual você vai executar o GitOps.

4. **Opcionalmente**, se o gráfico Helm exigir configurações em seus **valores**, edite a configuração `.helm.values` no arquivo `fleet.yaml` do diretório copiado.

5. **Opcionalmente**, pode haver casos de uso em que você tenha que adicionar outros recursos à instância do Fleet do seu gráfico para que se adapte melhor ao seu ambiente. Para obter informações de como aprimorar o diretório do Fleet, consulte Git Repository Contents (https://fleet.rancher.io/gitrepo-content) (Conteúdo do repositório Git).

Nota

Em alguns casos, o tempo limite padrão que o Fleet usa nas operações do Helm pode ser insuficiente e resultar no seguinte erro:

```
failed pre-install: context deadline exceeded
```

Nesses casos, adicione a propriedade timeoutSeconds (https://fleet.rancher.io/ref-crds#helmoptions) à configuração `helm` do arquivo `fleet.yaml`.

Esta é a aparência de um **exemplo** de gráfico Helm do `longhorn`:

- Estrutura de repositórios Git do usuário:

```
<user_repository_root>
├── longhorn
│   └── fleet.yaml
└── longhorn-crd
    └── fleet.yaml
```

- Conteúdo do `fleet.yaml` preenchido com os dados do `Longhorn` do usuário:

```
defaultNamespace: longhorn-system

helm:
  # timeoutSeconds: 10
  releaseName: "longhorn"
  chart: "longhorn"
  repo: "https://charts.rancher.io/"
  version: "106.2.0+up1.8.1"
  takeOwnership: true
  # custom chart value overrides
  values:
    # Example for user provided custom values content
    defaultSettings:
```

```
      deletingConfirmationFlag: true

# https://fleet.rancher.io/bundle-diffs
diff:
  comparePatches:
  - apiVersion: apiextensions.k8s.io/v1
    kind: CustomResourceDefinition
    name: engineimages.longhorn.io
    operations:
    - {"op":"remove", "path":"/status/conditions"}
    - {"op":"remove", "path":"/status/storedVersions"}
    - {"op":"remove", "path":"/status/acceptedNames"}
  - apiVersion: apiextensions.k8s.io/v1
    kind: CustomResourceDefinition
    name: nodes.longhorn.io
    operations:
    - {"op":"remove", "path":"/status/conditions"}
    - {"op":"remove", "path":"/status/storedVersions"}
    - {"op":"remove", "path":"/status/acceptedNames"}
  - apiVersion: apiextensions.k8s.io/v1
    kind: CustomResourceDefinition
    name: volumes.longhorn.io
    operations:
    - {"op":"remove", "path":"/status/conditions"}
    - {"op":"remove", "path":"/status/storedVersions"}
    - {"op":"remove", "path":"/status/acceptedNames"}
```

Nota

Estes são apenas valores de exemplo usados para ilustrar as configurações comuns no gráfico do `longhorn`. Eles **NÃO** devem ser considerados diretrizes de implantação para o gráfico do `longhorn`.

36.1.6.2.1.2 Implantar o Fleet para seu gráfico

Você pode implantar a instância do Fleet para seu gráfico usando o GitRepo (*Seção 36.1.6.2.1.2.1, "GitRepo"*) ou o bundle (*Seção 36.1.6.2.1.2.2, "Bundle"*).

Nota

Durante a implantação do Fleet, se você receber a mensagem `Modified` (Modificado), adicione uma entrada `comparePatches` correspondente à seção `diff` do Fleet. Para obter mais informações, consulte Generating Diffs to Ignore Modified GitRepos (https://fleet.rancher.io/bundle-diffs) (Gerando diffs para ignorar GitRepos modificados).

36.1.6.2.1.2.1 GitRepo

O recurso GitRepo (https://fleet.rancher.io/ref-gitrepo) do Fleet armazena as informações sobre como acessar os recursos do Fleet do seu gráfico e a quais clusters ele precisa aplicar esses recursos.

É possível implantar o recurso `GitRepo` pela IU do Rancher (https://ranchermanager.docs.rancher.com/v2.11/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui) ou manualmente pela implantação (https://fleet.rancher.io/tut-deployment) do recurso no `cluster de gerenciamento`.

Exemplo de recurso `GitRepo` do **Longhorn** para implantação **manual**:

```
apiVersion: fleet.cattle.io/v1alpha1
kind: GitRepo
metadata:
  name: longhorn-git-repo
  namespace: fleet-default
spec:
  # If using a tag
  # revision: user_repository_tag
  #
  # If using a branch
  # branch: user_repository_branch
  paths:
  # As seen in the 'Prepare your Fleet resources' example
  - longhorn
  - longhorn-crd
  repo: user_repository_url
  targets:
  # Match all clusters
  - clusterSelector: {}
```

36.1.6.2.1.2.2 Bundle

Os recursos Bundle (https://fleet.rancher.io/bundle-add) armazenam os recursos brutos do Kubernetes que o Fleet precisa implantar. Normalmente, a recomendação é usar a abordagem do `GitRepo`, mas para casos de uso em que o ambiente é air-gapped e não oferece suporte a um servidor Git local, os `Bundles` podem ajudar na propagação do Fleet do gráfico Helm aos clusters de destino.

É possível implantar um `Bundle` pela IU do Rancher (`Continuous Delivery → Advanced → Bundles → Create from YAML` (Entrega contínua → Avançado → Bundles → Criar do YAML)) ou pela implantação manual do recurso `Bundle` no namespace correto do Fleet. Para obter informações sobre os namespaces do Fleet, consulte a documentação (https://fleet.rancher.io/namespaces#gitrepos-bundles-clusters-clustergroups) upstream.

É possível criar `Bundles` para gráficos Helm do Edge usando a abordagem Converter um gráfico Helm em bundle (https://fleet.rancher.io/bundle-add#convert-a-helm-chart-into-a-bundle) do Fleet.

Veja a seguir um exemplo de como criar um recurso `Bundle` com base nos gabaritos de gráfico Helm longhorn (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/fleets/day2/chart-templates/longhorn/longhorn/fleet.yaml) e longhorn-crd (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/fleets/day2/chart-templates/longhorn/longhorn-crd/fleet.yaml) do Fleet e implantar manualmente esse bundle no `cluster de gerenciamento`.

Nota

Para ilustrar o fluxo de trabalho, o exemplo a seguir usa a estrutura de diretório suse-edge/fleet-examples (https://github.com/suse-edge/fleet-examples).

1. Navegue até o gabarito de gráfico longhorn (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/fleets/day2/chart-templates/longhorn/longhorn/fleet.yaml) do Fleet:

```
cd fleets/day2/chart-templates/longhorn/longhorn
```

2. Crie um arquivo `targets.yaml` para instruir o Fleet sobre os clusters em que ele deve implantar o gráfico Helm:

```
cat > targets.yaml <<EOF
targets:
# Matches all downstream clusters
```

```
- clusterSelector: {}
EOF
```

Para uma seleção mais granular de clusters downstream, consulte Mapping to Downstream Clusters (https://fleet.rancher.io/gitrepo-targets) (Mapeando para clusters downstream).

3. Converta o gráfico Helm `Longhorn` do Fleet em um recurso bundle usando fleet-cli (https://fleet.rancher.io/cli/fleet-cli/fleet).

Nota

É possível recuperar a CLI do Fleet da página **Assets** (Ativos) da respectiva versão (https://github.com/rancher/fleet/releases/tag/v0.12.2) (`fleet-linux-amd64`).

Para usuários do Mac, existe um Homebrew Formulae fleet-cli (https://formulae.brew.sh/formula/fleet-cli).

```
fleet apply --compress --targets-file=targets.yaml -n fleet-default -o - longhorn-
bundle > longhorn-bundle.yaml
```

4. Navegue até o gabarito de gráfico longhorn-crd (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/fleets/day2/chart-templates/longhorn/longhorn-crd/fleet.yaml) do Fleet:

```
cd fleets/day2/chart-templates/longhorn/longhorn-crd
```

5. Crie um arquivo `targets.yaml` para instruir o Fleet sobre os clusters em que ele deve implantar o gráfico Helm:

```
cat > targets.yaml <<EOF
targets:
# Matches all downstream clusters
- clusterSelector: {}
EOF
```

6. Converta o gráfico Helm `Longhorn CRD` do Fleet em um recurso bundle usando fleet-cli (https://fleet.rancher.io/cli/fleet-cli/fleet).

```
fleet apply --compress --targets-file=targets.yaml -n fleet-default -o - longhorn-
crd-bundle > longhorn-crd-bundle.yaml
```

7. Implante os arquivos `longhorn-bundle.yaml` e `longhorn-crd-bundle.yaml` no `cluster de gerenciamento`:

```
kubectl apply -f longhorn-crd-bundle.yaml
kubectl apply -f longhorn-bundle.yaml
```

Siga estas etapas para garantir que o `SUSE Storage` seja implantado em todos os clusters downstream especificados.

36.1.6.2.1.3 Gerenciar o gráfico Helm implantado

Após a implantação com o Fleet, para fazer upgrade dos gráficos Helm, consulte a *Seção 36.1.6.2.2, "Quero fazer upgrade de um gráfico Helm gerenciado pelo Fleet"*.

36.1.6.2.2 Quero fazer upgrade de um gráfico Helm gerenciado pelo Fleet

1. Determine a versão para a qual você precisa fazer upgrade do seu gráfico para que ele fique compatível com o lançamento desejado do Edge. Você encontra a versão do gráfico Helm por lançamento do Edge nas Notas de lançamento (*Seção 52.1, "Resumo"*).
2. No repositório Git monitorado pelo Fleet, edite o arquivo `fleet.yaml` do gráfico Helm com a **versão** e o **repositório** corretos do gráfico conforme as Notas de lançamento (*Seção 52.1, "Resumo"*).
3. Depois de confirmar e enviar as alterações ao repositório, será acionado um upgrade do gráfico Helm desejado.

36.1.6.2.3 Quero fazer upgrade de um gráfico Helm implantado pelo EIB

O *Capítulo 11, Edge Image Builder* implanta gráficos Helm criando um recurso `HelmChart` e usando o `helm-controller` introduzido pelo recurso de integração do Helm RKE2 (https://docs.rke2.io/helm)/K3s (https://docs.k3s.io/helm).

Para garantir que o upgrade do gráfico Helm implantado pelo `EIB` seja feito com sucesso, os usuários precisam fazer upgrade dos respectivos recursos `HelmChart`.

Veja a seguir informações sobre:

- A visão geral (*Seção 36.1.6.2.3.1, "Visão geral"*) do processo de upgrade.
- As etapas de upgrade necessárias (*Seção 36.1.6.2.3.2, "Etapas de upgrade"*).
- Um exemplo (*Seção 36.1.6.2.3.3, "Exemplo"*) que demonstra o upgrade de um gráfico Longhorn (https://longhorn.io) usando o método explicado.
- Como usar o processo de upgrade com uma ferramenta GitOps diferente (*Seção 36.1.6.2.3.4, "Upgrade do gráfico Helm usando uma ferramenta GitOps de terceiros"*).

36.1.6.2.3.1 Visão geral

O upgrade dos gráficos Helm que são implantados pelo `EIB` é feito por uma instância do `Fleet` chamada eib-charts-upgrader (https://github.com/suse-edge/fleet-examples/tree/release-3.3.0/fleets/day2/eib-charts-upgrader).

Essa instância do `Fleet` processa os dados **fornecidos pelo usuário** para **atualizar** um conjunto específico de recursos HelmChart.

A atualização desses recursos aciona o helm-controller (https://github.com/k3s-io/helm-controller), que faz **upgrade** dos gráficos Helm associados aos recursos `HelmChart` modificados.

O usuário precisa apenas:

1. Obter (https://helm.sh/docs/helm/helm_pull/) localmente os arquivos para cada gráfico Helm que precisa de upgrade.
2. Especificar esses arquivos no script generate-chart-upgrade-data.sh (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/scripts/day2/generate-chart-upgrade-data.sh) `generate-chart-upgrade-data.sh`, que incluirá os dados dos arquivos na instância do Fleet `eib-charts-upgrader`.
3. Implantar a instância do Fleet `eib-charts-upgrader` no respectivo `cluster de gerenciamento`. Isso é feito por um recurso `GitRepo` ou `Bundle`.

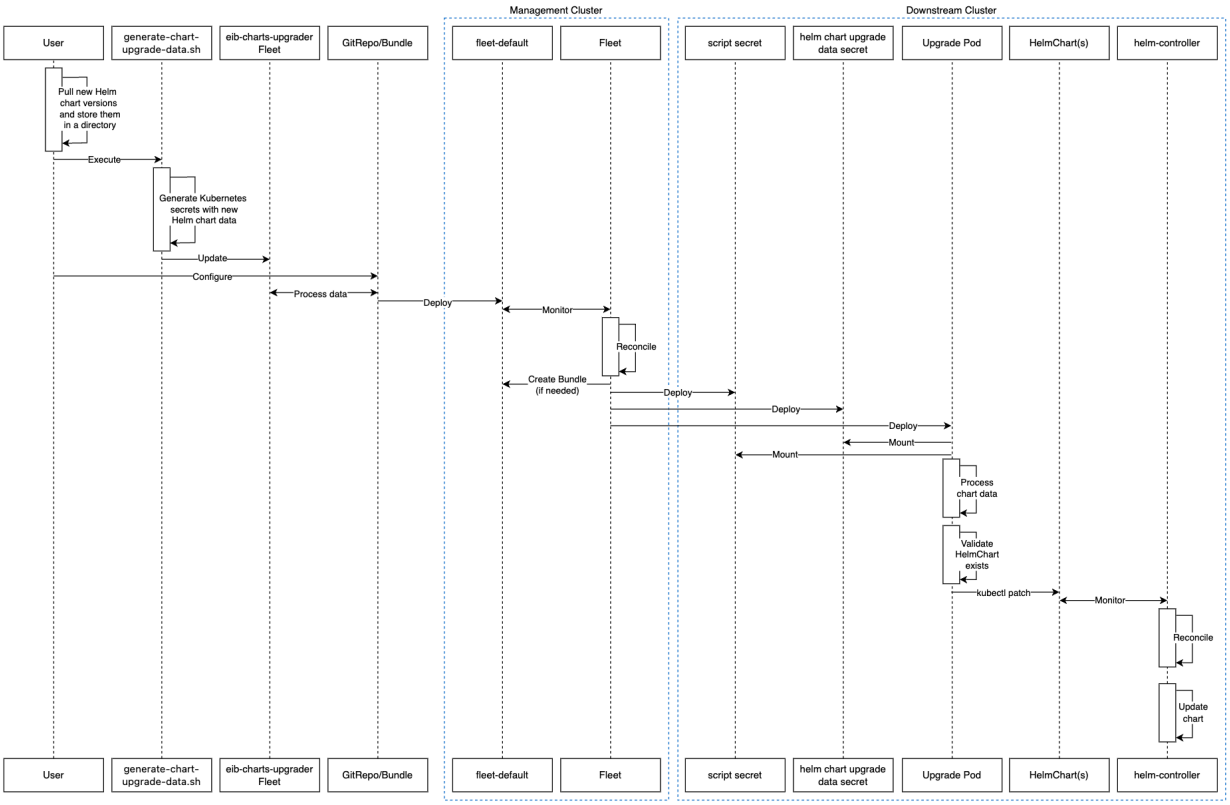
Após a implantação, o `eib-charts-upgrader`, com ajuda do Fleet, enviará os recursos para o cluster downstream desejado.

Esses recursos incluem:

1. Um conjunto de `segredos` com os dados do gráfico Helm **fornecidos pelo usuário**.
2. Um `Kubernetes Job` para implantar o `pod` que vai montar os `segredos` já mencionados e, com base neles, aplicar o patch (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_patch/) aos recursos HelmChart correspondentes.

Como já foi mencionado, isso acionará o `helm-controller`, que faz upgrade do gráfico Helm real.

Veja a seguir um diagrama da descrição acima:



36.1.6.2.3.2 Etapas de upgrade

1. Clone o repositório `suse-edge/fleet-examples` da tag (https://github.com/suse-edge/fleet-examples/releases/tag/release-3.3.0) da versão correta.
2. Crie um diretório para armazenar um ou mais arquivos dos gráficos Helm enviados.

```
mkdir archives
```

3. Dentro do diretório do arquivo recém-criado, extraia (https://helm.sh/docs/helm/helm_pull/) os arquivos dos gráficos Helm dos quais você deseja fazer upgrade:

```
cd archives
helm pull [chart URL | repo/chartname]

# Alternatively if you want to pull a specific version:
# helm pull [chart URL | repo/chartname] --version 0.0.0
```

4. Em **Assets** (Ativos) da tag da versão (https://github.com/suse-edge/fleet-examples/releases/tag/release-3.3.0) desejada, faça download do script `generate-chart-upgrade-data.sh`.

5. Execute o script `generate-chart-upgrade-data.sh`:

```
chmod +x ./generate-chart-upgrade-data.sh

./generate-chart-upgrade-data.sh --archive-dir /foo/bar/archives/ --fleet-path /foo/bar/fleet-examples/fleets/day2/eib-charts-upgrader
```

   Para cada arquivo de gráfico no diretório `--archive-dir`, o script gera um arquivo `YAML de segredo do Kubernetes` com os dados de upgrade do gráfico e o armazena no diretório `base/secrets` da instância do Fleet especificada por `--fleet-path`.
   O script `generate-chart-upgrade-data.sh` também aplica modificações adicionais à instância do Fleet para garantir que os arquivos `YAML de segredo do Kubernetes` gerados sejam corretamente usados pela carga de trabalho implantada pelo Fleet.

Importante

Os usuários não devem fazer alterações no conteúdo que é gerado pelo script `generate-chart-upgrade-data.sh`.

As etapas abaixo dependem do ambiente que está em execução:

1. Para um ambiente com suporte a GitOps (por exemplo, não air-gapped ou air-gapped, mas com suporte a servidor Git local):

   a. Copie a instância do Fleet `fleets/day2/eib-charts-upgrader` para o repositório que você vai usar com o GitOps.

Nota

Garanta que o Fleet inclua as alterações feitas pelo script `generate-chart-upgrade-data.sh`.

b. Configure o recurso `GitRepo` que será usado para enviar todos os recursos da instância do Fleet `eib-charts-upgrader`.

   i. Para configuração e implantação do `GitRepo` pela IU do Rancher, consulte Accessing Fleet in the Rancher UI (https://ranchermanager.docs.rancher.com/v2.11/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui) (Acessando o Fleet na IU do Racher).

   ii. Para configuração e implantação manuais do `GitRepo`, consulte Creating a Deployment (https://fleet.rancher.io/tut-deployment) (Criando uma implantação).

2. Para um ambiente sem suporte a GitOps (por exemplo, air-gapped que não permite o uso de servidor Git local):

   a. Faça download do binário `fleet-cli` na página da versão (https://github.com/rancher/fleet/releases/tag/v0.12.2) do `rancher/fleet` (`fleet-linux-amd64` para Linux). Para usuários do Mac, existe um Homebrew Formulae que pode ser usado: fleet-cli (https://formulae.brew.sh/formula/fleet-cli).

   b. Navegue até a instância do Fleet `eib-charts-upgrader`:

```
cd /foo/bar/fleet-examples/fleets/day2/eib-charts-upgrader
```

   c. Crie um arquivo `targets.yaml` para instruir o Fleet sobre onde implantar seus recursos:

```
cat > targets.yaml <<EOF
targets:
# To match all downstream clusters
- clusterSelector: {}
EOF
```

Para obter informações sobre como mapear clusters de destino, consulte a documentação (https://fleet.rancher.io/gitrepo-targets) upstream.

d. Use `fleet-cli` para converter o Fleet em um recurso `Bundle`:

```
fleet apply --compress --targets-file=targets.yaml -n fleet-default -o - eib-
charts-upgrade > bundle.yaml
```

Isso cria um bundle (`bundle.yaml`) para armazenar todos os recursos usados como gabaritos da instância do Fleet `eib-charts-upgrader`.
Para obter mais informações sobre o comando `fleet apply`, consulte fleet apply (https://fleet.rancher.io/cli/fleet-cli/fleet_apply).
Para obter mais informações sobre como converter instâncias do Fleet em bundles, consulte Convert a Helm Chart into a Bundle (https://fleet.rancher.io/bundle-add#convert-a-helm-chart-into-a-bundle) (Converter um gráfico Helm em bundle).

e. Implante o `Bundle`. É possível fazer isso de uma destas maneiras:

   i. Pela IU do Rancher: navegue até **Continuous Delivery** → **Advanced** → **Bundles** → **Create from YAML** (Entrega contínua → Avançado → Bundles → Criar do YAML) e cole o conteúdo do `bundle.yaml` ou clique na opção `Read from File` (Ler arquivo) e especifique o arquivo.

   ii. Manualmente: implante o arquivo `bundle.yaml` manualmente no `cluster de gerenciamento`.

A execução dessas etapas resulta na implantação bem-sucedida do recurso `GitRepo/Bundle`. O Fleet seleciona o recurso e seu conteúdo é implantado nos clusters de destino que o usuário especificou nas etapas anteriores. Para obter uma visão geral do processo, consulte a *Seção 36.1.6.2.3.1, "Visão geral"*.

Para obter informações sobre como acompanhar o processo de upgrade, consulte a *Seção 36.1.6.2.3.3, "Exemplo"*.

Importante

Após a verificação bem-sucedida da atualização do gráfico, remova o recurso `Bundle/GitRepo`.

Isso removerá os recursos de upgrade que não são mais necessários do cluster `downstream`, garantindo que não haja conflitos com versões futuras.

36.1.6.2.3.3 Exemplo

Nota

No exemplo a seguir, demonstramos como fazer upgrade de um gráfico Helm implantado pelo `EIB` de uma versão para outra em um cluster `downstream`. Observe que as versões usadas neste exemplo **não** são recomendações. Para recomendações de versão específicas a um lançamento do Edge, consulte as Notas de lançamento (*Seção 52.1, "Resumo"*).

Caso de uso:

- Um cluster chamado `doc-example` executa uma versão mais antiga do Longhorn (https://longhorn.io).
- O cluster foi implantado pelo EIB usando o seguinte *trecho* de definição da imagem:

```
kubernetes:
  helm:
    charts:
    - name: longhorn-crd
      repositoryName: rancher-charts
      targetNamespace: longhorn-system
      createNamespace: true
      version: 104.2.0+up1.7.1
      installationNamespace: kube-system
    - name: longhorn
      repositoryName: rancher-charts
      targetNamespace: longhorn-system
      createNamespace: true
      version: 104.2.0+up1.7.1
      installationNamespace: kube-system
    repositories:
    - name: rancher-charts
      url: https://charts.rancher.io/
...
```

- É preciso fazer upgrade do `SUSE Storage` para uma versão compatível com o Edge 3.3.1, ou seja, para a versão `106.2.0+up1.8.1`.
- Presume-se que o `cluster de gerenciamento` responsável pelo gerenciamento do `doc-example` seja **air-gapped**, sem suporte a servidor Git local e com uma instalação ativa do Rancher.

Siga as etapas de upgrade (*Seção 36.1.6.2.3.2, "Etapas de upgrade"*):

1. Clone o repositório `suse-edge/fleet-example` da tag `release-3.3.0`.

```
git clone -b release-3.3.0 https://github.com/suse-edge/fleet-examples.git
```

2. Crie um diretório para armazenar o arquivo de upgrade do `Longhorn`.

```
mkdir archives
```

3. Extraia a versão desejada do arquivo de gráficos do `Longhorn`:

```
# First add the Rancher Helm chart repository
helm repo add rancher-charts https://charts.rancher.io/

# Pull the Longhorn 1.8.1 CRD archive
helm pull rancher-charts/longhorn-crd --version 106.2.0+up1.8.1

# Pull the Longhorn 1.8.1 chart archive
helm pull rancher-charts/longhorn --version 106.2.0+up1.8.1
```

4. Fora do diretório `archives`, faça download do script `generate-chart-upgrade-data.sh` da tag (https://github.com/suse-edge/fleet-examples/releases/tag/release-3.3.0) da versão do `suse-edge/fleet-examples`.

5. A configuração do diretório deve ter uma aparência como esta:

```
.
├── archives
|   ├── longhorn-106.2.0+up1.8.1.tgz
|   └── longhorn-crd-106.2.0+up1.8.1.tgz
├── fleet-examples
...
|   ├── fleets
|   |   ├── day2
|   |   |   ├── ...
|   |   |   ├── eib-charts-upgrader
|   |   |   |   ├── base
|   |   |   |   |   ├── job.yaml
|   |   |   |   |   ├── kustomization.yaml
|   |   |   |   |   ├── patches
|   |   |   |   |   |   └── job-patch.yaml
|   |   |   |   |   ├── rbac
|   |   |   |   |   |   ├── cluster-role-binding.yaml
|   |   |   |   |   |   ├── cluster-role.yaml
```

```
|   |   |   |   |   |   ├── kustomization.yaml
|   |   |   |   |   |   └── sa.yaml
|   |   |   |   |   └── secrets
|   |   |   |   |       ├── eib-charts-upgrader-script.yaml
|   |   |   |   |       └── kustomization.yaml
|   |   |   |   ├── fleet.yaml
|   |   |   |   └── kustomization.yaml
|   |   |   └── ...
|   └── ...
└── generate-chart-upgrade-data.sh
```

6. Execute o script `generate-chart-upgrade-data.sh`:

```
# First make the script executable
chmod +x ./generate-chart-upgrade-data.sh

# Then execute the script
./generate-chart-upgrade-data.sh --archive-dir ./archives --fleet-path ./fleet-
examples/fleets/day2/eib-charts-upgrader
```

A estrutura de diretórios após a execução do script deve ser parecida com esta:

```
.
├── archives
|   ├── longhorn-106.2.0+up1.8.1.tgz
|   └── longhorn-crd-106.2.0+up1.8.1.tgz
├── fleet-examples
...
|   ├── fleets
|   |   ├── day2
|   |   |   ├── ...
|   |   |   ├── eib-charts-upgrader
|   |   |   |   ├── base
|   |   |   |   |   ├── job.yaml
|   |   |   |   |   ├── kustomization.yaml
|   |   |   |   |   ├── patches
|   |   |   |   |   |   └── job-patch.yaml
|   |   |   |   |   ├── rbac
|   |   |   |   |   |   ├── cluster-role-binding.yaml
|   |   |   |   |   |   ├── cluster-role.yaml
|   |   |   |   |   |   ├── kustomization.yaml
|   |   |   |   |   |   └── sa.yaml
|   |   |   |   |   └── secrets
|   |   |   |   |       ├── eib-charts-upgrader-script.yaml
|   |   |   |   |       ├── kustomization.yaml
|   |   |   |   |       ├── longhorn-VERSION.yaml - secret created by the generate-
chart-upgrade-data.sh script
```

```
|   |   |   |   |       └── longhorn-crd-VERSION.yaml - secret created by the
 generate-chart-upgrade-data.sh script
|   |   |   |   ├── fleet.yaml
|   |   |   |   └── kustomization.yaml
|   |   |   └── ...
|   └── ...
└── generate-chart-upgrade-data.sh
```

Os arquivos alterados no Git devem ter aparência similar a esta:

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
 modified:   fleets/day2/eib-charts-upgrader/base/patches/job-patch.yaml
 modified:   fleets/day2/eib-charts-upgrader/base/secrets/kustomization.yaml

Untracked files:
  (use "git add <file>..." to include in what will be committed)
 fleets/day2/eib-charts-upgrader/base/secrets/longhorn-VERSION.yaml
 fleets/day2/eib-charts-upgrader/base/secrets/longhorn-crd-VERSION.yaml
```

7. Crie um `Bundle` para a instância do Fleet `eib-charts-upgrader`:

   a. Primeiro, navegue até o Fleet:

   ```
   cd ./fleet-examples/fleets/day2/eib-charts-upgrader
   ```

   b. Em seguida, crie um arquivo `targets.yaml`:

   ```
   cat > targets.yaml <<EOF
   targets:
   - clusterName: doc-example
   EOF
   ```

   c. Depois disso, use o binário `fleet-cli` para converter a instância do Fleet em um bundle:

   ```
   fleet apply --compress --targets-file=targets.yaml -n fleet-default -o - eib-
   charts-upgrade > bundle.yaml
   ```

   d. Agora transfira o `bundle.yaml` para a máquina do `cluster de gerenciamento`.

8. Implante o bundle usando a IU do Rancher:

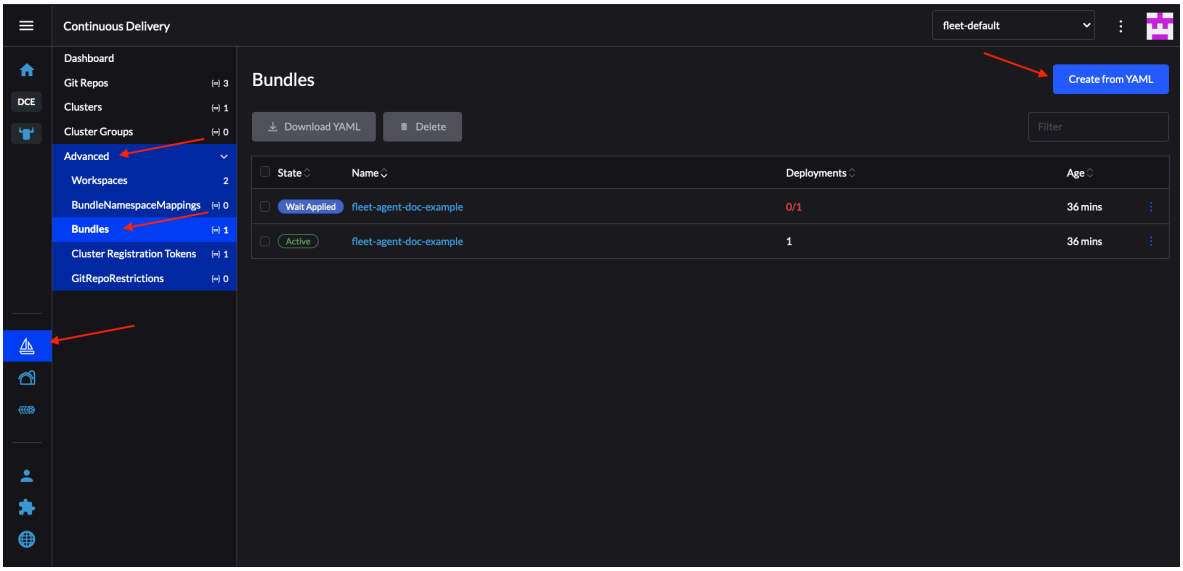


FIGURA 36.1: IMPLANTAR O BUNDLE USANDO A IU DO RANCHER

Agora selecione **Read from File** (Ler arquivo) e encontre o arquivo `bundle.yaml` no sistema.
Isso preencherá automaticamente o `Bundle` na IU do Rancher.
Selecione **Create** (Criar).

9. Após a implantação bem-sucedida, o bundle terá uma aparência similar a esta:

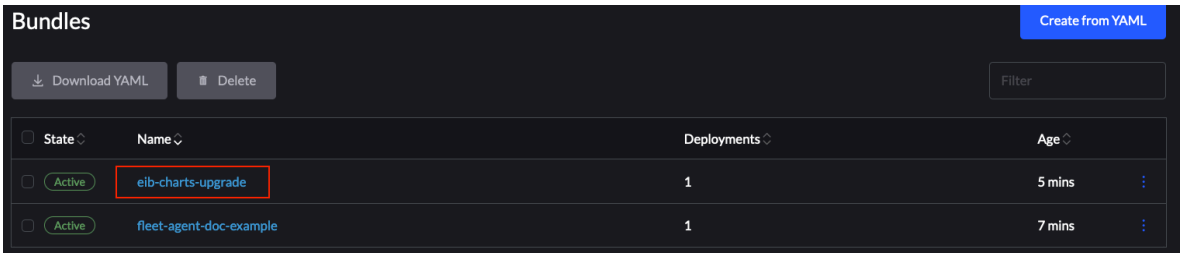
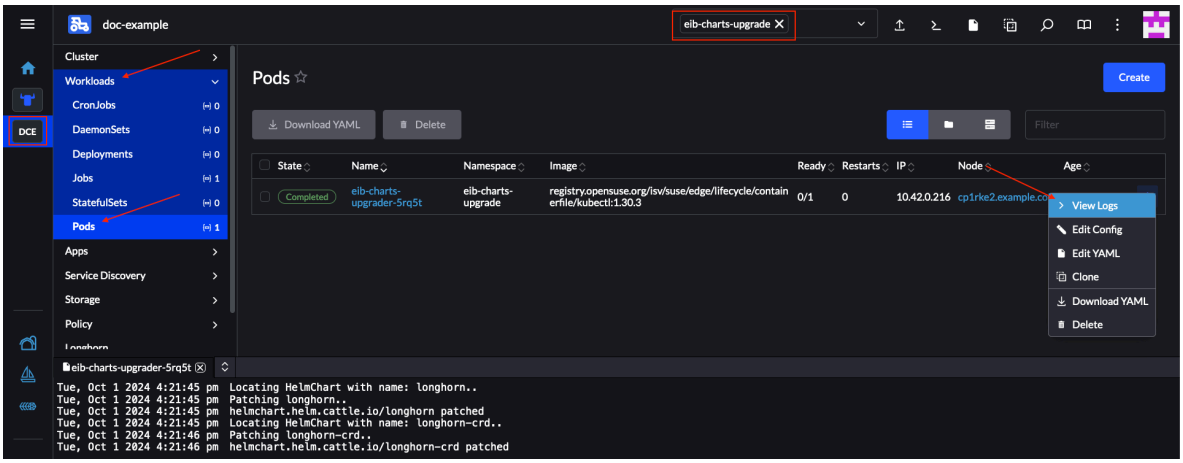


FIGURA 36.2: BUNDLE IMPLANTADO COM SUCESSO

Após a implantação bem-sucedida do `Bundle`, monitore o processo de upgrade da seguinte maneira:

1. Consulte os registros do `pod de upgrade`:



2. Agora consulte os registros do pod criado pelo helm-controller para o upgrade:

    a. O nome do pod seguirá o seguinte gabarito: `helm-install-longhorn-<sufixo-aleatório>`

    b. O pod estará no namespace em que o recurso `HelmChart` foi implantado. No nosso caso, o namespace é `kube-system`.

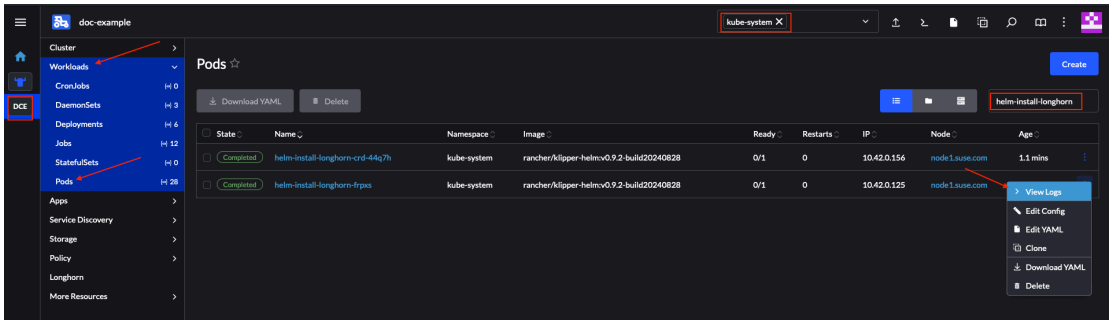


FIGURA 36.3: REGISTROS DO GRÁFICO DO LONGHORN ATUALIZADO COM SUCESSO

3. Verifique se a versão do `HelmChart` foi atualizada navegando até a seção `HelmCharts` do Rancher (`More Resources → HelmCharts` "Mais recursos → HelmCharts"). Selecione o namespace em que o gráfico foi implantado; que, neste exemplo, é `kube-system`.

4. Por fim, verifique se os pods do Longhorn estão em execução.

Depois de fazer as validações acima, é seguro pressupor que o upgrade do gráfico Helm do Longhorn foi feito para a versão `106.2.0+up1.8.1`.

36.1.6.2.3.4 Upgrade do gráfico Helm usando uma ferramenta GitOps de terceiros

Em alguns casos de uso, os usuários talvez queiram seguir esse procedimento de upgrade com um fluxo de trabalho do GitOps diferente do Fleet (por exemplo, `Flux`).

Para gerar os recursos necessários ao procedimento de upgrade, você pode usar o script `generate-chart-upgrade-data.sh` para preencher a instância do Fleet `eib-charts-upgrader` com os dados fornecidos pelo usuário. Para obter mais informações sobre como fazer isso, consulte a *Seção 36.1.6.2.3.2, "Etapas de upgrade"*.

Com a configuração completa, você pode usar o kustomize (https://kustomize.io) para gerar uma solução totalmente funcional para ser implantada no seu cluster:

```
cd /foo/bar/fleets/day2/eib-charts-upgrader

kustomize build .
```

Para incluir a solução no fluxo de trabalho do GitOps, remova o arquivo `fleet.yaml` e use o que sobrou como configuração válida do `Kustomize`. Lembre-se de executar primeiro o script `generate-chart-upgrade-data.sh` para que ele possa preencher a configuração do `Kustomize` com os dados dos gráficos Helm para o qual você quer fazer upgrade.

Para saber qual é a finalidade de uso desse fluxo de trabalho, consulte a *Seção 36.1.6.2.3.1, "Visão geral"* e a *Seção 36.1.6.2.3.2, "Etapas de upgrade"*.

VII Documentação do produto

Aqui você encontra a documentação do SUSE Edge for Telco

37 SUSE Edge for Telco

O SUSE Edge for Telco (antes conhecido como Adaptive Telco Infrastructure Platform/ATIP) é uma plataforma de computação de borda otimizada para telecomunicações que permite que as empresas de telecomunicações inovem e acelerem a modernização de suas redes.

SUSE Edge for Telco é uma pilha de nuvem de telecomunicações completa para hospedar CNFs, como pacote 5G Core e Cloud RAN.

- Automatiza a distribuição "zero touch" e o gerenciamento do ciclo de vida de configurações complexas da pilha de borda de acordo com a escala de telecomunicações.
- Garante a qualidade contínua do hardware voltado para telecomunicações usando as configurações e cargas de trabalho específicas dessa área.
- Consiste em componentes desenvolvidos especificamente para a borda e, portanto, com menor pegada e maior desempenho por Watt.
- Mantém uma estratégia de plataforma flexível com APIs independentes de fornecedor e código 100% aberto.

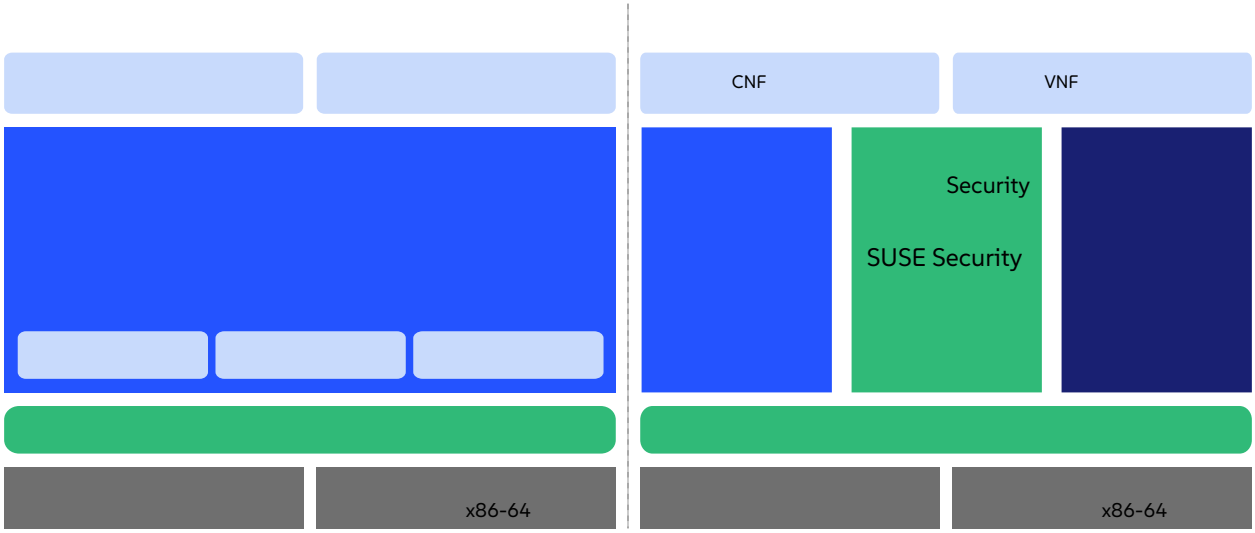
38 Conceito e arquitetura

O SUSE Edge for Telco é uma plataforma desenvolvida para hospedar em escala aplicativos de telecomunicações modernos e nativos de nuvem, do núcleo à borda.

Esta página explica a arquitetura e os componentes usados no SUSE Edge for Telco.

38.1 Arquitetura do SUSE Edge for Telco

O diagrama abaixo mostra a arquitetura de alto nível do SUSE Edge for Telco:



38.2 Componentes

Há dois blocos diferentes; pilha de gerenciamento e pilha de runtime:

- **Pilha de gerenciamento**: trata-se da parte do SUSE Edge for Telco usada para gerenciar o provisionamento e o ciclo de vida das pilhas de runtime. Ela inclui os seguintes componentes:
  - Gerenciamento multicluster em ambientes de nuvem pública e privada com o Rancher (*Capítulo 5, Rancher*)
  - Suporte a bare metal com os provedores de infraestrutura Metal3 (*Capítulo 10, Metal*^{3}), MetalLB (*Capítulo 19, MetalLB*) e `CAPI` (Cluster API)

    - Isolamento completo de locatários e integrações de `IDP` (provedor de identidade)
    - Amplo mercado de integrações e extensões de terceiros
    - API independente de fornecedor e vasto ecossistema de provedores
    - Controle das atualizações transacionais do SUSE Linux Micro
    - GitOps Engine para gerenciar o ciclo de vida dos clusters usando repositórios Git com o Fleet (*Capítulo 8, Fleet*)
- **Pilha de runtime**: trata-se da parte do SUSE Edge for Telco usada para executar as cargas de trabalho.
    - Kubernetes com distribuições seguras e leves como K3s (*Capítulo 15, K3s*) e RKE2 (*Capítulo 16, RKE2*) (o `RKE2` é protegido, certificado e otimizado para uso em órgãos governamentais e setores regulamentados).
    - SUSE Security (*Capítulo 18, SUSE Security*) para habilitar recursos de segurança, como verificação de vulnerabilidades de imagem, inspeção detalhada de pacotes e controle automático do tráfego intracluster.
    - Armazenamento em blocos com SUSE Storage (*Capítulo 17, SUSE Storage*) para permitir um meio simples e fácil de usar uma solução de armazenamento nativa de nuvem.
    - Sistema operacional otimizado com SUSE Linux Micro (*Capítulo 9, SUSE Linux Micro*) para oferecer um sistema operacional seguro, leve e imutável (sistema de arquivos transacional) para execução de contêineres. O SUSE Linux Micro está disponível nas arquiteturas AArch64 e AMD64/Intel 64 e oferece suporte ao `kernel Real-Time` para casos de uso de telecomunicações e de borda.

38.3 Fluxos de implantação de exemplo

Veja a seguir exemplos de alto nível de fluxos de trabalho para entender o relacionamento entre os componentes de gerenciamento e de runtime.

O provisionamento de rede direcionado é o fluxo de trabalho que permite a implantação de um novo cluster downstream com todos os componentes pré-configurados e prontos para executar as cargas de trabalho sem intervenção manual.

38.3.1 Exemplo 1: implantação de um novo cluster de gerenciamento com todos os componentes instalados

Uso do Edge Image Builder (*Capítulo 11, Edge Image Builder*) para criar uma nova imagem `ISO` com a pilha de gerenciamento incluída. Você pode usar essa imagem `ISO` para instalar um novo cluster de gerenciamento em VMs ou bare metal.

Nota

Para obter mais informações sobre como implantar um novo cluster de gerenciamento, consulte o guia sobre cluster de gerenciamento do SUSE Edge for Telco (*Capítulo 40, Configurando o cluster de gerenciamento*).

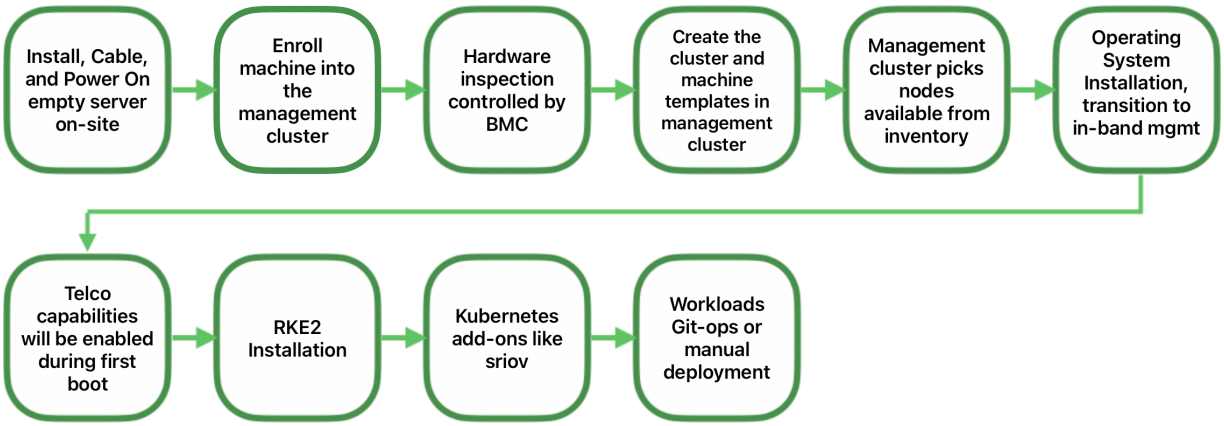
Nota

Para obter mais informações sobre como usar o Edge Image Builder, consulte o respectivo guia (*Capítulo 3, Clusters independentes com o Edge Image Builder*).

38.3.2 Exemplo 2: implantação de um cluster downstream de nó único com perfis de telecomunicações para que ele possa executar cargas de trabalho de telecomunicações

Quando o cluster de gerenciamento estiver em funcionamento, poderemos usá-lo para implantar um cluster downstream de nó único com todos os recursos de telecomunicações habilitados e configurados por meio do fluxo de trabalho de provisionamento de rede direcionado.

O seguinte diagrama mostra o fluxo de trabalho de alto nível para implantá-lo:



Nota

Para obter mais informações sobre como implantar um cluster downstream, consulte o guia sobre provisionamento automatizado do SUSE Edge for Telco (*Capítulo 42, Provisionamento de rede direcionado totalmente automatizado*).

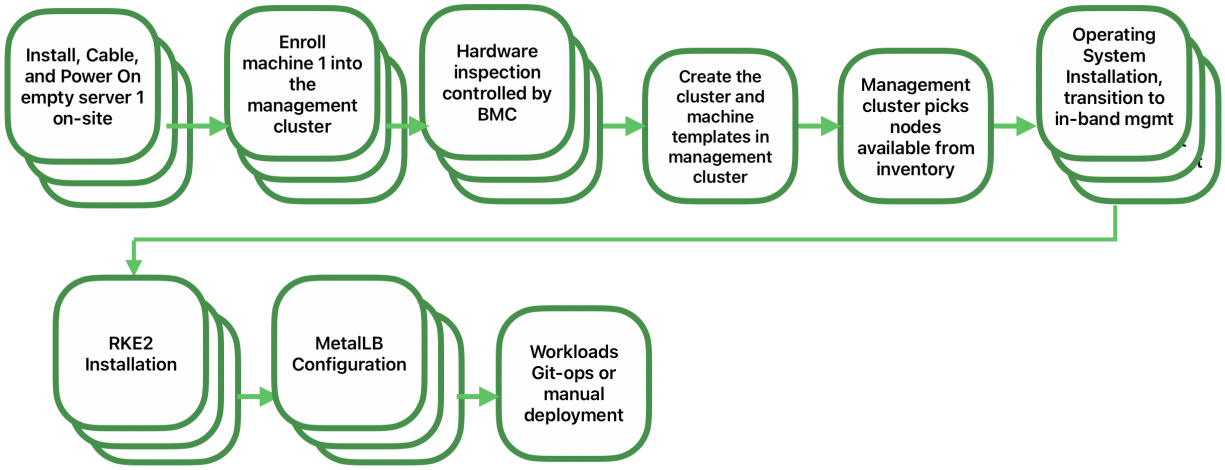
Nota

Para obter mais informações sobre os recursos de telecomunicações, consulte o guia sobre recursos de telecomunicações do SUSE Edge for Telco (*Capítulo 41, Configuração dos recursos de telecomunicações*).

38.3.3 Exemplo 3: implantação de um cluster downstream de alta disponibilidade usando o MetalLB como balanceador de carga

Depois que o cluster de gerenciamento estiver em funcionamento, poderemos usá-lo para implantar um cluster downstream de alta disponibilidade com o `MetalLB` como balanceador de carga, usando o fluxo de trabalho de provisionamento de rede direcionado.

O seguinte diagrama mostra o fluxo de trabalho de alto nível para implantá-lo:



Nota

Para obter mais informações sobre como implantar um cluster downstream, consulte o guia sobre provisionamento automatizado do SUSE Edge for Telco (*Capítulo 42, Provisionamento de rede direcionado totalmente automatizado*).

Nota

Para obter mais informações sobre o `MetalLB`, consulte: (*Capítulo 19, MetalLB*)

39 Requisitos e suposições

39.1 Hardware

Veja a seguir os requisitos de hardware para o SUSE Edge for Telco:

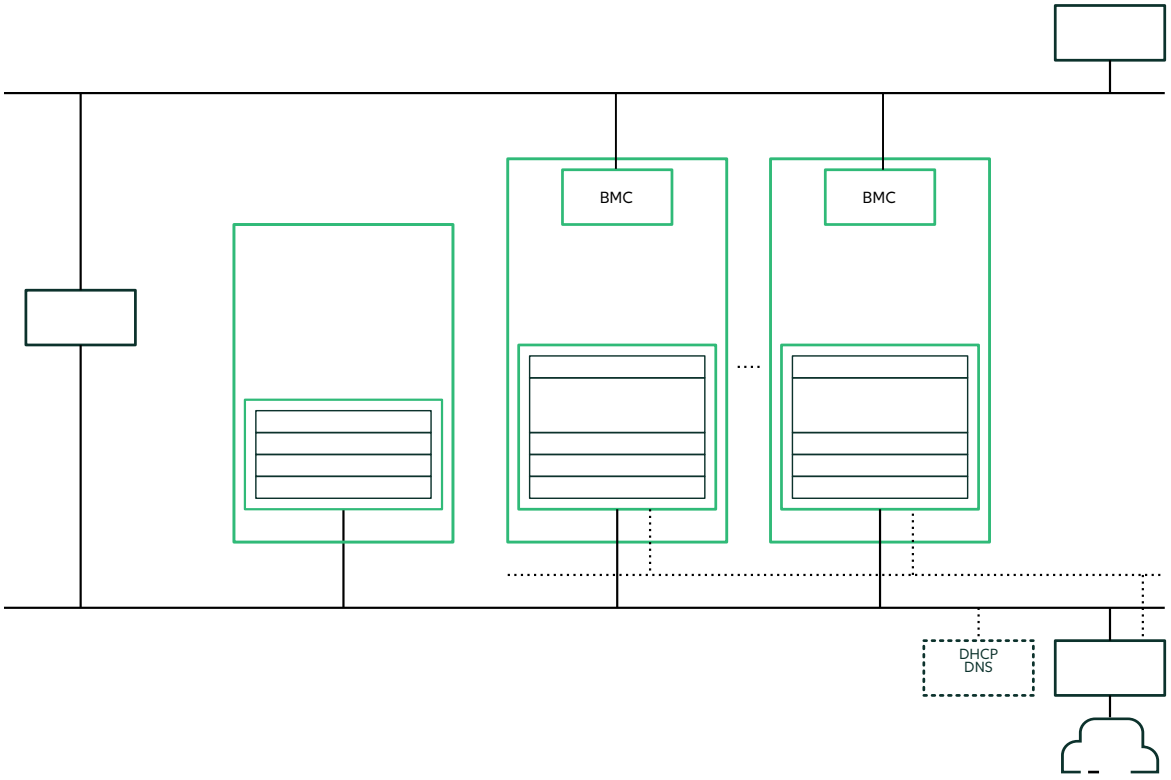
- **Cluster de gerenciamento**: contém componentes, como `SUSE Linux Micro`, `RKE2`, `SUSE Rancher Prime` e `Metal3`, e é usado para gerenciar vários clusters downstream. Dependendo do número de clusters downstream que será gerenciado, os requisitos de hardware para o servidor podem variar.
    - Os requisitos mínimos para o servidor (`VM` ou `bare metal`) são:
        - RAM: mínimo de 8 GB (recomendamos pelo menos 16 GB)
        - CPU: mínimo de 2 (recomendamos pelo menos 4)
- **Clusters downstream**: trata-se dos clusters implantados para executar as cargas de trabalho de telecomunicações. Requisitos específicos são necessários para habilitar determinados recursos de telecomunicações, como `SR-IOV`, `otimização de desempenho da CPU` etc.
    - SR-IOV: para anexar funções virtuais (VFs, Virtual Functions) no modo de passagem a CNFs/VNFs, a NIC deve permitir que SR-IOV e VT-d/AMD-Vi sejam habilitados no BIOS.
    - Processadores de CPU: para executar cargas de trabalho de telecomunicações, o modelo de processador da CPU deve ser adaptado para disponibilizar a maior parte dos recursos nesta tabela de referência (*Capítulo 41, Configuração dos recursos de telecomunicações*).
    - Requisitos de firmware para instalação com mídia virtual:

Hardware de servidor	Modelo de BMC	Gerenciamento
Hardware Dell	15ª geração	iDRAC9
Hardware Supermicro	01.00.25	Supermicro SMC – redfish

Hardware HPE	1.50	iLO6

39.2 Rede

Como referência à arquitetura de rede, o seguinte diagrama mostra uma arquitetura de rede típica para um ambiente de telecomunicações:



A arquitetura de rede baseia-se nos seguintes componentes:

- **Rede de gerenciamento**: essa rede é usada para gerenciamento de nós do cluster downstream. Ela é usada para gerenciamento fora da banda. Em geral, essa rede também é conectada a um comutador de gerenciamento separado, mas pode ser conectada ao mesmo comutador de serviço usando VLANs para isolar o tráfego.
- **Rede do plano de controle**: essa rede é usada para comunicação entre os nós do cluster downstream e os serviços executados neles. Ela também é usada para comunicação entre os nós e os serviços externos, como servidores `DHCP` ou `DNS`. Em alguns casos, o comutador/roteador processa o tráfego pela Internet nos ambientes conectados.
- **Outras redes**: em alguns casos, os nós podem se conectar a outras redes para finalidades específicas.

Nota

Para usar o fluxo de trabalho de provisionamento de rede direcionado, o cluster de gerenciamento deve ter conectividade de rede com o Baseboard Management Controller (BMC) do servidor de cluster downstream para que seja possível automatizar a preparação e o provisionamento do host.

39.3 Serviços (DHCP, DNS etc.)

Alguns serviços externos, como `DHCP`, `DNS` etc., podem ser necessários, dependendo do tipo de ambiente em que estão implantados:

- **Ambiente conectado**: nesse caso, os nós são conectados à Internet (por protocolos de roteamento L3), e o cliente providencia os serviços externos.
- **Ambiente desconectado/air-gapped**: nesse caso, os nós não têm conectividade IP de Internet e são exigidos serviços adicionais para espelhar localmente o conteúdo necessário pelo fluxo de trabalho de provisionamento de rede direcionado.
- **Servidor de arquivos**: usado para armazenar as imagens de sistema operacional que são provisionadas nos nós do cluster downstream durante o fluxo de trabalho de provisionamento de rede direcionado. O gráfico Helm do `Metal3` pode implantar um servidor de mídia para armazenar as imagens de sistema operacional. Consulte a seguinte seção (*Nota*), mas também é possível usar um servidor web local existente.

39.4 Desabilitando os serviços systemd

Para cargas de trabalho de telecomunicações, é importante desabilitar ou configurar apropriadamente alguns dos serviços executados nos nós para evitar qualquer impacto no desempenho das cargas de trabalho em execução nos nós (latência).

- `rebootmgr` é um serviço que permite configurar uma estratégia de reinicialização quando o sistema tem atualizações pendentes. Para cargas de trabalho de telecomunicações, é realmente importante desabilitar ou configurar apropriadamente o serviço `rebootmgr` para evitar a reinicialização dos nós em caso de atualizações programadas pelo sistema, a fim de evitar qualquer impacto nos serviços executados nos nós.

Nota

Para obter mais informações sobre o `rebootmgr`, consulte o repositório rebootmgr do GitHub (https://github.com/SUSE/rebootmgr).

Verifique a estratégia usada executando este comando:

```
cat /etc/rebootmgr.conf
[rebootmgr]
window-start=03:30
window-duration=1h30m
strategy=best-effort
lock-group=default
```

e você pode desabilitá-la executando:

```
sed -i 's/strategy=best-effort/strategy=off/g' /etc/rebootmgr.conf
```

ou usando o comando `rebootmgrctl`:

```
rebootmgrctl strategy off
```

Nota

É possível automatizar essa configuração para definir a estratégia `rebootmgr` usando o fluxo de trabalho de provisionamento de rede direcionado. Para obter mais informações, consulte a documentação sobre provisionamento automatizado (*Capítulo 42, Provisionamento de rede direcionado totalmente automatizado*).

- `transactional-update` é um serviço que permite que o sistema controle as atualizações automáticas. Para cargas de trabalho de telecomunicações, é importante desabilitar as atualizações automáticas para evitar qualquer impacto nos serviços executados nos nós.

Para desabilitar as atualizações automáticas, execute:

```
systemctl --now disable transactional-update.timer
systemctl --now disable transactional-update-cleanup.timer
```

- `fstrim` é um serviço que permite cortar os sistemas de arquivos automaticamente toda semana. Para cargas de trabalho de telecomunicações, é importante desabilitar o corte automático para evitar qualquer impacto nos serviços executados nos nós.

Para desabilitar o corte automático, execute:

```
systemctl --now disable fstrim.timer
```

40 Configurando o cluster de gerenciamento

40.1 Introdução

O cluster de gerenciamento é a parte do SUSE Edge for Telco usada para gerenciar o provisionamento e o ciclo de vida das pilhas de runtime. Do ponto de vista técnico, esse cluster contém os seguintes componentes:

- `SUSE Linux Micro` como sistema operacional. Dependendo do caso de uso, é possível personalizar algumas configurações, como rede, armazenamento, usuários e argumentos do kernel.
- `RKE2` como cluster Kubernetes. Dependendo do caso de uso, é possível configurá-lo para usar plug-ins de CNI específicos, como `Multus`, `Cilium`, `Calico` etc.
- `Rancher` como plataforma de gerenciamento para gerenciar o ciclo de vida dos clusters.
- `Metal3` como componente para gerenciar o ciclo de vida dos nós bare metal.
- `CAPI` como componente para gerenciar o ciclo de vida dos clusters Kubernetes (downstream). O `provedor CAPI RKE2` é usado para gerenciar o ciclo de vida dos clusters RKE2.

Com todos os componentes mencionados acima, o cluster de gerenciamento pode gerenciar o ciclo de vida dos clusters downstream, aplicando uma abordagem declarativa para gerenciar a infraestrutura e os aplicativos.

Nota

Para obter mais informações sobre o `SUSE Linux Micro`, consulte: SUSE Linux Micro (*Capítulo 9, SUSE Linux Micro*)

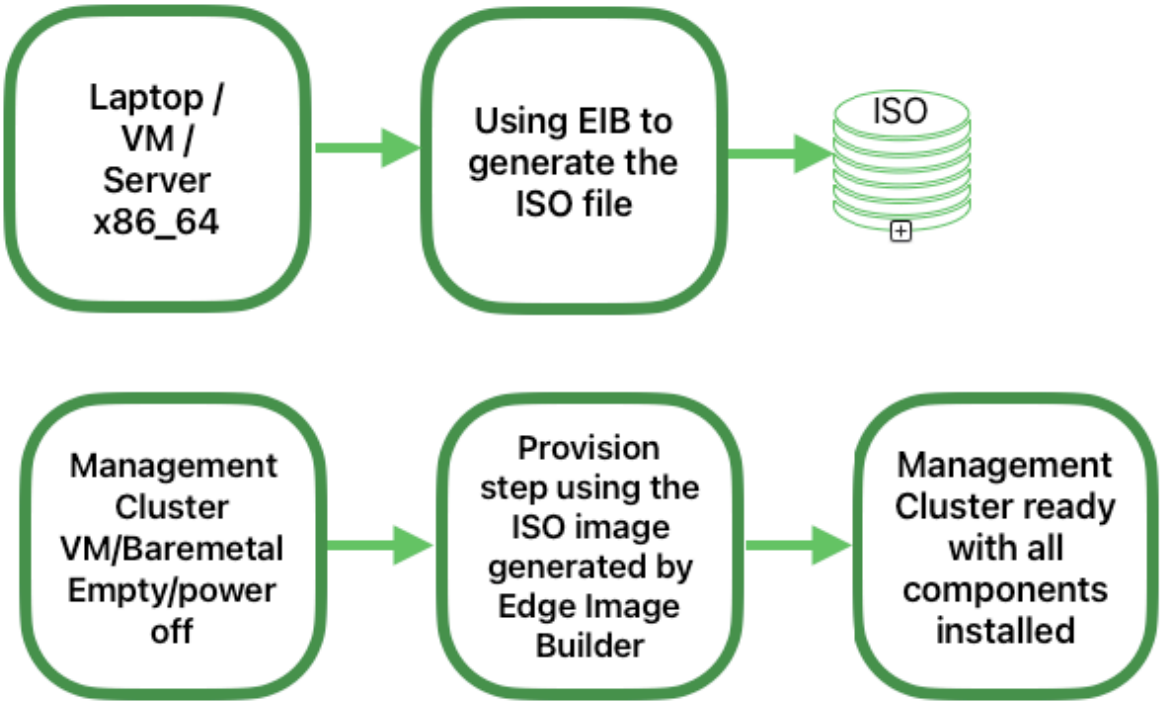
Para obter mais informações sobre o `RKE2`, consulte: RKE2 (*Capítulo 16, RKE2*)

Para obter mais informações sobre o `Rancher`, consulte: Rancher (*Capítulo 5, Rancher*)

Para obter mais informações sobre o `Metal3`, consulte: Metal3 (*Capítulo 10, Metal*^{3})

40.2 Etapas para configurar o cluster de gerenciamento

As seguintes etapas são necessárias para configurar o cluster de gerenciamento (de nó único):



Veja a seguir as etapas principais para configurar o cluster de gerenciamento com uma abordagem declarativa:

1. **Preparação da imagem para ambientes conectados (*Seção 40.3, "Preparação da imagem para ambientes conectados"*)**: a primeira etapa é preparar o manifesto e os arquivos com todas as configurações necessárias aos ambientes conectados.

- Estrutura de diretórios para ambientes conectados (*Seção 40.3.1, "Estrutura de diretórios"*): esta etapa cria uma estrutura de diretórios que o Edge Image Builder usará para armazenar os arquivos de configuração e a própria imagem.
- Arquivo de definição do cluster de gerenciamento (*Seção 40.3.2, "Arquivo de definição do cluster de gerenciamento"*): o `mgmt-cluster.yaml` é o principal arquivo de definição do cluster de gerenciamento. Ele inclui as seguintes informações sobre a imagem que será criada:
  - Informações da imagem: as informações relacionadas à imagem que será criada usando a imagem base.
  - Sistema operacional: a configuração do sistema operacional que será usada na imagem.
  - Kubernetes: repositórios e gráficos Helm, versão do kubernetes, configuração de rede e os nós que serão usados no cluster.
- Pasta custom (*Seção 40.3.3, "Pasta custom"*): a pasta `custom` contém os arquivos de configuração e os scripts que o Edge Image Builder usará para implantar um cluster totalmente funcional.
  - Arquivos: contêm os arquivos de configuração que serão usados pelo cluster de gerenciamento.
  - Scripts: contêm os scripts que serão usados pelo cluster de gerenciamento.
- Pasta kubernetes (*Seção 40.3.4, "Pasta kubernetes"*): a pasta `kubernetes` contém os arquivos de configuração que serão usados pelo cluster de gerenciamento.

- Manifestos: contêm os manifestos que serão usados pelo cluster de gerenciamento.
- Helm: contém os arquivos de valores do Helm que serão usados pelo cluster de gerenciamento.
- Config: contém os arquivos de configuração que serão usados pelo cluster de gerenciamento.

- Pasta network (*Seção 40.3.5, "Pasta de rede"*): a pasta `network` contém os arquivos de configuração de rede que serão usados pelos nós do cluster de gerenciamento.

2. **Preparação da imagem para ambientes air-gapped (*Seção 40.4, "Preparação da imagem para ambientes air-gapped"*)**: a etapa mostra as diferenças para preparar os manifestos e os arquivos que serão usados em um cenário air-gapped.

   - Modificações no arquivo de definição (*Seção 40.4.1, "Modificações no arquivo de definição"*): o arquivo `mgmt-cluster.yaml` deve ser modificado para incluir a seção `embeddedArtifactRegistry` com o campo `images` definido para que todas as imagens do contêiner sejam incluídas na imagem de saída do EIB.
   - Modificações na pasta custom (*Seção 40.4.2, "Modificações na pasta custom"*): a pasta `custom` deve ser modificada para incluir os recursos necessários para executar o cluster de gerenciamento em um ambiente air-gapped.
     - Script de registro: o script `custom/scripts/99-register.sh` deve ser removido quando você usa um ambiente air-gapped.
   - Modificações na pasta de valores do Helm (*Seção 40.4.3, "Modificações na pasta de valores do Helm"*): a pasta `helm/values` deve ser modificada para incluir a configuração necessária para executar o cluster de gerenciamento em um ambiente air-gapped.

3. **Criação da imagem (*Seção 40.5, "Criação de imagem"*)**: esta etapa abrange a criação da imagem com a ferramenta Edge Image Builder (para os cenários tanto conectados quanto air-gapped). Consulte os pré-requisitos (*Capítulo 11, Edge Image Builder*) para executar a ferramenta Edge Image Builder no seu sistema.

4. **Provisionamento do cluster de gerenciamento (*Seção 40.6, "Provisionar o cluster de gerenciamento"*)**: esta etapa aborda o provisionamento do cluster de gerenciamento usando a imagem criada na etapa anterior (para os cenários tanto conectados quanto air-gapped). Para executá-la, é possível usar um laptop, um servidor, uma VM ou qualquer outro sistema AMD64/Intel 64 com uma porta USB.

Nota

Para obter mais informações sobre o Edge Image Builder, consulte o respectivo guia (*Capítulo 11, Edge Image Builder*) e o Guia de Início Rápido (*Capítulo 3, Clusters independentes com o Edge Image Builder*).

40.3 Preparação da imagem para ambientes conectados

O Edge Image Builder é usado para criar a imagem do cluster de gerenciamento. Neste documento, explicamos a configuração mínima necessária para configurar o cluster de gerenciamento.

O Edge Image Builder é executado dentro de um contêiner, portanto, um tempo de execução do contêiner é necessário, como Podman (https://podman.io) ou Rancher Desktop (https://rancherdesktop.io). Neste guia, consideramos que o podman esteja disponível.

Também como pré-requisito para implantar um cluster de gerenciamento altamente disponível, você precisa reservar três IPs na sua rede:

- `apiVIP` para o endereço VIP da API (usado para acessar o servidor da API Kubernetes).
- `ingressVIP` para o endereço VIP de entrada (consumido pela IU do Rancher, por exemplo).
- `metal3VIP` para o endereço VIP do Metal3.

40.3.1 Estrutura de diretórios

Ao executar o EIB, um diretório é montado do host, portanto, a primeira etapa é criar uma estrutura de diretórios para o EIB usar para armazenar os arquivos de configuração e a própria imagem. Esse diretório tem a seguinte estrutura:

```
eib
├── mgmt-cluster.yaml
├── network
│   └── mgmt-cluster-node1.yaml
├── kubernetes
│   ├── manifests
│   │   ├── rke2-ingress-config.yaml
│   │   ├── neuvector-namespace.yaml
│   │   ├── ingress-l2-adv.yaml
│   │   └── ingress-ippool.yaml
│   ├── helm
│   │   └── values
│   │       ├── rancher.yaml
│   │       ├── neuvector.yaml
│   │       ├── metal3.yaml
│   │       └── certmanager.yaml
│   └── config
│       └── server.yaml
├── custom
│   ├── scripts
│   │   ├── 99-register.sh
│   │   ├── 99-mgmt-setup.sh
│   │   └── 99-alias.sh
│   └── files
│       ├── rancher.sh
│       ├── mgmt-stack-setup.service
│       ├── metal3.sh
│       └── basic-setup.sh
└── base-images
```

Nota

É necessário fazer download da imagem `SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso` do SUSE Customer Center (https://scc.suse.com/) ou da página de downloads da SUSE (https://www.suse.com/download/sle-micro/) e salvá-la na pasta `base-images`.

Confira o checksum SHA256 da imagem para garantir que ela não tenha sido adulterada. O checksum está no mesmo local em que a imagem foi baixada.

Há um exemplo da estrutura de diretórios disponível no repositório SUSE Edge do GitHub na pasta "telco-examples" (https://github.com/suse-edge/atip).

40.3.2 Arquivo de definição do cluster de gerenciamento

O `mgmt-cluster.yaml` é o principal arquivo de definição do cluster de gerenciamento. Ele contém as seguintes informações:

```
apiVersion: 1.2
image:
  imageType: iso
  arch: x86_64
  baseImage: SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso
  outputImageName: eib-mgmt-cluster-image.iso
operatingSystem:
  isoConfiguration:
    installDevice: /dev/sda
  users:
  - username: root
    encryptedPassword: $ROOT_PASSWORD
  packages:
    packageList:
    - git
    - jq
    sccRegistrationCode: $SCC_REGISTRATION_CODE
kubernetes:
  version: v1.32.4+rke2r1
  helm:
    charts:
      - name: cert-manager
        repositoryName: jetstack
        version: 1.15.3
        targetNamespace: cert-manager
        valuesFile: certmanager.yaml
        createNamespace: true
        installationNamespace: kube-system
      - name: longhorn-crd
        version: 106.2.0+up1.8.1
        repositoryName: rancher-charts
        targetNamespace: longhorn-system
        createNamespace: true
        installationNamespace: kube-system
      - name: longhorn
        version: 106.2.0+up1.8.1
```

```
      repositoryName: rancher-charts
      targetNamespace: longhorn-system
      createNamespace: true
      installationNamespace: kube-system
    - name: metal3
      version: 303.0.7+up0.11.5
      repositoryName: suse-edge-charts
      targetNamespace: metal3-system
      createNamespace: true
      installationNamespace: kube-system
      valuesFile: metal3.yaml
    - name: rancher-turtles
      version: 303.0.4+up0.20.0
      repositoryName: suse-edge-charts
      targetNamespace: rancher-turtles-system
      createNamespace: true
      installationNamespace: kube-system
    - name: neuvector-crd
      version: 106.0.1+up2.8.6
      repositoryName: rancher-charts
      targetNamespace: neuvector
      createNamespace: true
      installationNamespace: kube-system
      valuesFile: neuvector.yaml
    - name: neuvector
      version: 106.0.1+up2.8.6
      repositoryName: rancher-charts
      targetNamespace: neuvector
      createNamespace: true
      installationNamespace: kube-system
      valuesFile: neuvector.yaml
    - name: rancher
      version: 2.11.2
      repositoryName: rancher-prime
      targetNamespace: cattle-system
      createNamespace: true
      installationNamespace: kube-system
      valuesFile: rancher.yaml
  repositories:
    - name: jetstack
      url: https://charts.jetstack.io
    - name: rancher-charts
      url: https://charts.rancher.io/
    - name: suse-edge-charts
      url: oci://registry.suse.com/edge/charts
    - name: rancher-prime
      url: https://charts.rancher.com/server-charts/prime
```

```
  network:
    apiHost: $API_HOST
    apiVIP: $API_VIP
  nodes:
    - hostname: mgmt-cluster-node1
      initializer: true
      type: server
#   - hostname: mgmt-cluster-node2
#     type: server
#   - hostname: mgmt-cluster-node3
#     type: server
```

Para explicar os campos e os valores no arquivo de definição `mgmt-cluster.yaml`, dividimos esse arquivo nas seções a seguir.

- Seção da imagem (arquivo de definição):

```
image:
  imageType: iso
  arch: x86_64
  baseImage: SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso
  outputImageName: eib-mgmt-cluster-image.iso
```

em que `baseImage` é a imagem original que você baixou do SUSE Customer Center ou da página de downloads da SUSE. `outputImageName` é o nome da nova imagem que será usada para provisionar o cluster de gerenciamento.

- Seção do sistema operacional (arquivo de definição):

```
operatingSystem:
  isoConfiguration:
    installDevice: /dev/sda
  users:
  - username: root
    encryptedPassword: $ROOT_PASSWORD
  packages:
    packageList:
    - jq
    sccRegistrationCode: $SCC_REGISTRATION_CODE
```

em que `installDevice` é o dispositivo usado para instalar o sistema operacional, `username` e `encryptedPassword` são as credenciais usadas para acessar o sistema, `packageList` é a lista dos pacotes que devem ser instalados (`jq` é obrigatório internamente durante o processo de

instalação) e `sccRegistrationCode` é o código de registro usado para obter os pacotes e as dependências no momento da criação e pode ser recuperado do SUSE Customer Center. É possível gerar a senha criptografada usando o comando `openssl` da seguinte maneira:

```
openssl passwd -6 MyPassword!123
```

A saída é similar a esta:

```
$6$UrXB1sAGs46DOiSq$HSwi9GFJLCorm0J53nF2Sq8YEoyINhHcObHzX2R8h13mswUIsMwzx4eUzn/
rRx0QPV4JIb0eWCoNrxGiKH4R31
```

- Seção do Kubernetes (arquivo de definição):

```
kubernetes:
  version: v1.32.4+rke2r1
  helm:
    charts:
      - name: cert-manager
        repositoryName: jetstack
        version: 1.15.3
        targetNamespace: cert-manager
        valuesFile: certmanager.yaml
        createNamespace: true
        installationNamespace: kube-system
      - name: longhorn-crd
        version: 106.2.0+up1.8.1
        repositoryName: rancher-charts
        targetNamespace: longhorn-system
        createNamespace: true
        installationNamespace: kube-system
      - name: longhorn
        version: 106.2.0+up1.8.1
        repositoryName: rancher-charts
        targetNamespace: longhorn-system
        createNamespace: true
        installationNamespace: kube-system
      - name: metal3
        version: 303.0.7+up0.11.5
        repositoryName: suse-edge-charts
        targetNamespace: metal3-system
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: metal3.yaml
      - name: rancher-turtles
        version: 303.0.4+up0.20.0
        repositoryName: suse-edge-charts
```

```
        targetNamespace: rancher-turtles-system
        createNamespace: true
        installationNamespace: kube-system
      - name: neuvector-crd
        version: 106.0.1+up2.8.6
        repositoryName: rancher-charts
        targetNamespace: neuvector
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: neuvector.yaml
      - name: neuvector
        version: 106.0.1+up2.8.6
        repositoryName: rancher-charts
        targetNamespace: neuvector
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: neuvector.yaml
      - name: rancher
        version: 2.11.2
        repositoryName: rancher-prime
        targetNamespace: cattle-system
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: rancher.yaml
    repositories:
      - name: jetstack
        url: https://charts.jetstack.io
      - name: rancher-charts
        url: https://charts.rancher.io/
      - name: suse-edge-charts
        url: oci://registry.suse.com/edge/charts
      - name: rancher-prime
        url: https://charts.rancher.com/server-charts/prime
    network:
      apiHost: $API_HOST
      apiVIP: $API_VIP
    nodes:
    - hostname: mgmt-cluster-node1
      initializer: true
      type: server
#   - hostname: mgmt-cluster-node2
#     type: server
#   - hostname: mgmt-cluster-node3
#     type: server
```

A seção `helm` contém a lista de gráficos Helm para instalação, os repositórios que serão usados e a configuração da versão de todos eles.

A seção `network` contém a configuração da rede, como o `apiHost` e o `apiVIP` que o componente `RKE2` usará. O `apiVIP` deve ser um endereço IP não usado na rede e que não faça parte do pool DHCP (se DHCP for usado). Além disso, quando usamos o `apiVIP` em um cluster de vários nós, ele é usado para acessar o servidor da API Kubernetes. O `apiHost` é a resolução de nome para o `apiVIP` que o componente `RKE2` usará.

A seção `nodes` contém a lista dos nós usados no cluster. Neste exemplo, usamos um cluster de nó único, mas é possível estendê-lo para um cluster de vários nós adicionando nós à lista (removendo o marcador de comentário das linhas).

Nota

- Os nomes dos nós devem ser exclusivos no cluster.
- Se preferir, use o campo `initializer` para especificar o host de inicialização; do contrário, ele será o primeiro nó da lista.
- Os nomes dos nós devem ser iguais aos nomes de host definidos na pasta de rede (*Seção 40.3.5, "Pasta de rede"*) quando a configuração de rede é obrigatória.

40.3.3 Pasta custom

A pasta `custom` contém as seguintes subpastas:

```
...
├── custom
│ ├── scripts
│ │ ├── 99-register.sh
│ │ ├── 99-mgmt-setup.sh
│ │ └── 99-alias.sh
│ └── files
│     ├── rancher.sh
│     ├── mgmt-stack-setup.service
│     ├── metal3.sh
│     └── basic-setup.sh
...
```

- A pasta `custom/files` contém os arquivos de configuração usados pelo cluster de gerenciamento.
- A pasta `custom/scripts` contém os scripts usados pelo cluster de gerenciamento.

A pasta `custom/files` contém os seguintes arquivos:

- `basic-setup.sh`: inclui os parâmetros de configuração para `Metal3`, `Rancher` e `MetalLB`. Modifique esse arquivo apenas para alterar os namespaces usados.

```
#!/bin/bash
# Pre-requisites. Cluster already running
export KUBECTL="/var/lib/rancher/rke2/bin/kubectl"
export KUBECONFIG="/etc/rancher/rke2/rke2.yaml"

##################
# METAL3 DETAILS #
##################
export METAL3_CHART_TARGETNAMESPACE="metal3-system"

############
# METALLB #
############
export METALLBNAMESPACE="metallb-system"

############
# RANCHER #
############
export RANCHER_CHART_TARGETNAMESPACE="cattle-system"
export RANCHER_FINALPASSWORD="adminadminadmin"

die(){
  echo ${1} 1>&2
  exit ${2}
}
```

- `metal3.sh`: inclui a configuração para o componente `Metal3` usado (sem necessidade de modificação). Em versões futuras, esse script será usado no lugar do `Rancher Turtles` para facilitar o uso.

```
#!/bin/bash
set -euo pipefail

BASEDIR="$(dirname "$0")"
source ${BASEDIR}/basic-setup.sh
```

```
METAL3LOCKNAMESPACE="default"
METAL3LOCKCMNAME="metal3-lock"

trap 'catch $? $LINENO' EXIT

catch() {
  if [ "$1" != "0" ]; then
    echo "Error $1 occurred on $2"
    ${KUBECTL} delete configmap ${METAL3LOCKCMNAME} -n ${METAL3LOCKNAMESPACE}
  fi
}

# Get or create the lock to run all those steps just in a single node
# As the first node is created WAY before the others, this should be enough
# TODO: Investigate if leases is better
if [ $(${KUBECTL} get cm -n ${METAL3LOCKNAMESPACE} ${METAL3LOCKCMNAME} -o name | wc
 -l) -lt 1 ]; then
  ${KUBECTL} create configmap ${METAL3LOCKCMNAME} -n ${METAL3LOCKNAMESPACE} --from-
literal foo=bar
else
  exit 0
fi

# Wait for metal3
while ! ${KUBECTL} wait --for condition=ready -n ${METAL3_CHART_TARGETNAMESPACE}
 $(${KUBECTL} get pods -n ${METAL3_CHART_TARGETNAMESPACE} -l app.kubernetes.io/
name=metal3-ironic -o name) --timeout=10s; do sleep 2 ; done

# Get the ironic IP
IRONICIP=$(${KUBECTL} get cm -n ${METAL3_CHART_TARGETNAMESPACE} ironic-bmo -o
 jsonpath='{.data.IRONIC_IP}')

# If LoadBalancer, use metallb, else it is NodePort
if [ $(${KUBECTL} get svc -n ${METAL3_CHART_TARGETNAMESPACE} metal3-metal3-ironic -o
 jsonpath='{.spec.type}') == "LoadBalancer" ]; then
  # Wait for metallb
  while ! ${KUBECTL} wait --for condition=ready -n ${METALLBNAMESPACE} $(${KUBECTL}
 get pods -n ${METALLBNAMESPACE} -l app.kubernetes.io/component=controller -o name)
 --timeout=10s; do sleep 2 ; done

  # Do not create the ippool if already created
  ${KUBECTL} get ipaddresspool -n ${METALLBNAMESPACE} ironic-ip-pool -o name || cat
 <<-EOF | ${KUBECTL} apply -f -
  apiVersion: metallb.io/v1beta1
  kind: IPAddressPool
  metadata:
    name: ironic-ip-pool
```

```
    namespace: ${METALLBNAMESPACE}
  spec:
    addresses:
    - ${IRONICIP}/32
    serviceAllocation:
      priority: 100
      serviceSelectors:
      - matchExpressions:
        - {key: app.kubernetes.io/name, operator: In, values: [metal3-ironic]}
 EOF

  # Same for L2 Advs
  ${KUBECTL} get L2Advertisement -n ${METALLBNAMESPACE} ironic-ip-pool-l2-adv -o
 name || cat <<-EOF | ${KUBECTL} apply -f -
  apiVersion: metallb.io/v1beta1
  kind: L2Advertisement
  metadata:
    name: ironic-ip-pool-l2-adv
    namespace: ${METALLBNAMESPACE}
  spec:
    ipAddressPools:
    - ironic-ip-pool
 EOF
fi

# If rancher is deployed
if [ $(${KUBECTL} get pods -n ${RANCHER_CHART_TARGETNAMESPACE} -l app=rancher -o
 name | wc -l) -ge 1 ]; then
  cat <<-EOF | ${KUBECTL} apply -f -
 apiVersion: management.cattle.io/v3
 kind: Feature
 metadata:
   name: embedded-cluster-api
 spec:
   value: false
 EOF

  # Disable Rancher webhooks for CAPI
  ${KUBECTL} delete --ignore-not-found=true
 mutatingwebhookconfiguration.admissionregistration.k8s.io mutating-webhook-
configuration
  ${KUBECTL} delete --ignore-not-found=true
 validatingwebhookconfigurations.admissionregistration.k8s.io validating-webhook-
configuration
  ${KUBECTL} wait --for=delete namespace/cattle-provisioning-capi-system --
timeout=300s
fi
```

```
# Clean up the lock cm

${KUBECTL} delete configmap ${METAL3LOCKCMNAME} -n ${METAL3LOCKNAMESPACE}
```

- `rancher.sh`: inclui a configuração para o componente `Rancher` usado (sem necessidade de modificação).

```
#!/bin/bash
set -euo pipefail

BASEDIR="$(dirname "$0")"
source ${BASEDIR}/basic-setup.sh

RANCHERLOCKNAMESPACE="default"
RANCHERLOCKCMNAME="rancher-lock"

if [ -z "${RANCHER_FINALPASSWORD}" ]; then
  # If there is no final password, then finish the setup right away
  exit 0
fi

trap 'catch $? $LINENO' EXIT

catch() {
  if [ "$1" != "0" ]; then
    echo "Error $1 occurred on $2"
    ${KUBECTL} delete configmap ${RANCHERLOCKCMNAME} -n ${RANCHERLOCKNAMESPACE}
  fi
}

# Get or create the lock to run all those steps just in a single node
# As the first node is created WAY before the others, this should be enough
# TODO: Investigate if leases is better
if [ $(${KUBECTL} get cm -n ${RANCHERLOCKNAMESPACE} ${RANCHERLOCKCMNAME} -o
 name | wc -l) -lt 1 ]; then
  ${KUBECTL} create configmap ${RANCHERLOCKCMNAME} -n ${RANCHERLOCKNAMESPACE}
 --from-literal foo=bar
else
  exit 0
fi

# Wait for rancher to be deployed
while ! ${KUBECTL} wait --for condition=ready -n
 ${RANCHER_CHART_TARGETNAMESPACE} $(${KUBECTL} get pods -n
```

```
 ${RANCHER_CHART_TARGETNAMESPACE} -l app=rancher -o name) --timeout=10s; do
 sleep 2 ; done
until ${KUBECTL} get ingress -n ${RANCHER_CHART_TARGETNAMESPACE} rancher > /
dev/null 2>&1; do sleep 10; done

RANCHERBOOTSTRAPPASSWORD=$(${KUBECTL} get secret -n
 ${RANCHER_CHART_TARGETNAMESPACE} bootstrap-secret -o
 jsonpath='{.data.bootstrapPassword}' | base64 -d)
RANCHERHOSTNAME=$(${KUBECTL} get ingress -n ${RANCHER_CHART_TARGETNAMESPACE}
 rancher -o jsonpath='{.spec.rules[0].host}')

# Skip the whole process if things have been set already
if [ -z $(${KUBECTL} get settings.management.cattle.io first-login -
ojsonpath='{.value}') ]; then
  # Add the protocol
  RANCHERHOSTNAME="https://${RANCHERHOSTNAME}"
  TOKEN=""
  while [ -z "${TOKEN}" ]; do
    # Get token
    sleep 2
    TOKEN=$(curl -sk -X POST ${RANCHERHOSTNAME}/v3-public/localProviders/local?
action=login -H 'content-type: application/json' -d "{\"username\":\"admin\",
\"password\":\"${RANCHERBOOTSTRAPPASSWORD}\"}" | jq -r .token)
  done

  # Set password
  curl -sk ${RANCHERHOSTNAME}/v3/users?action=changepassword -H 'content-type:
 application/json' -H "Authorization: Bearer $TOKEN" -d "{\"currentPassword\":
\"${RANCHERBOOTSTRAPPASSWORD}\",\"newPassword\":\"${RANCHER_FINALPASSWORD}\"}"

  # Create a temporary API token (ttl=60 minutes)
  APITOKEN=$(curl -sk ${RANCHERHOSTNAME}/v3/token -H 'content-
type: application/json' -H "Authorization: Bearer ${TOKEN}" -d
 '{"type":"token","description":"automation","ttl":3600000}' | jq -r .token)

  curl -sk ${RANCHERHOSTNAME}/v3/settings/server-url -H 'content-type:
 application/json' -H "Authorization: Bearer ${APITOKEN}" -X PUT -d "{\"name\":
\"server-url\",\"value\":\"${RANCHERHOSTNAME}\"}"
  curl -sk ${RANCHERHOSTNAME}/v3/settings/telemetry-opt -X PUT -H 'content-
type: application/json' -H 'accept: application/json' -H "Authorization: Bearer
 ${APITOKEN}" -d '{"value":"out"}'
fi

# Clean up the lock cm
```

```
${KUBECTL} delete configmap ${RANCHERLOCKCMNAME} -n ${RANCHERLOCKNAMESPACE}
```

- `mgmt-stack-setup.service`: contém a configuração para criar o serviço systemd para executar os scripts durante a primeira inicialização (nenhuma modificação é necessária).

```
[Unit]
Description=Setup Management stack components
Wants=network-online.target
# It requires rke2 or k3s running, but it will not fail if those services are
 not present
After=network.target network-online.target rke2-server.service k3s.service
# At least, the basic-setup.sh one needs to be present
ConditionPathExists=/opt/mgmt/bin/basic-setup.sh

[Service]
User=root
Type=forking
# Metal3 can take A LOT to download the IPA image
TimeoutStartSec=1800

ExecStartPre=/bin/sh -c "echo 'Setting up Management components...'"
# Scripts are executed in StartPre because Start can only run a single one
ExecStartPre=/opt/mgmt/bin/rancher.sh
ExecStartPre=/opt/mgmt/bin/metal3.sh
ExecStart=/bin/sh -c "echo 'Finished setting up Management components'"
RemainAfterExit=yes
KillMode=process
# Disable & delete everything
ExecStartPost=rm -f /opt/mgmt/bin/rancher.sh
ExecStartPost=rm -f /opt/mgmt/bin/metal3.sh
ExecStartPost=rm -f /opt/mgmt/bin/basic-setup.sh
ExecStartPost=/bin/sh -c "systemctl disable mgmt-stack-setup.service"
ExecStartPost=rm -f /etc/systemd/system/mgmt-stack-setup.service

[Install]
WantedBy=multi-user.target
```

A pasta `custom/scripts` contém os seguintes arquivos:

- Script `99-alias.sh`: contém o álias que o cluster de gerenciamento usa para carregar o arquivo kubeconfig na primeira inicialização (sem necessidade de modificação).

```
#!/bin/bash
echo "alias k=kubectl" >> /etc/profile.local
echo "alias kubectl=/var/lib/rancher/rke2/bin/kubectl" >> /etc/profile.local
```

```
echo "export KUBECONFIG=/etc/rancher/rke2/rke2.yaml" >> /etc/profile.local
```

- Script `99-mgmt-setup.sh`: contém a configuração para copiar os scripts durante a primeira inicialização (sem necessidade de modificação).

```
#!/bin/bash

# Copy the scripts from combustion to the final location
mkdir -p /opt/mgmt/bin/
for script in basic-setup.sh rancher.sh metal3.sh; do
 cp ${script} /opt/mgmt/bin/
done

# Copy the systemd unit file and enable it at boot
cp mgmt-stack-setup.service /etc/systemd/system/mgmt-stack-setup.service
systemctl enable mgmt-stack-setup.service
```

- Script `99-register.sh`: contém a configuração para registrar o sistema usando o código de registro do SCC. O `${SCC_ACCOUNT_EMAIL}` e o `${SCC_REGISTRATION_CODE}` devem ser devidamente definidos para registrar o sistema com a sua conta.

```
#!/bin/bash
set -euo pipefail

# Registration https://www.suse.com/support/kb/doc/?id=000018564
if ! which SUSEConnect > /dev/null 2>&1; then
 zypper --non-interactive install suseconnect-ng
fi
SUSEConnect --email "${SCC_ACCOUNT_EMAIL}" --url "https://scc.suse.com" --regcode
 "${SCC_REGISTRATION_CODE}"
```

40.3.4 Pasta kubernetes

A pasta `kubernetes` contém as seguintes subpastas:

```
...
├── kubernetes
│ ├── manifests
│ │ ├── rke2-ingress-config.yaml
│ │ ├── neuvector-namespace.yaml
│ │ ├── ingress-l2-adv.yaml
│ │ └── ingress-ippool.yaml
│ ├── helm
│ │ └── values
```

```
| |     ├── rancher.yaml
| |     ├── neuvector.yaml
| |     ├── metal3.yaml
| |     └── certmanager.yaml
| └── config
|     └── server.yaml
...
```

A pasta `kubernetes/config` contém os seguintes arquivos:

- `server.yaml`: por padrão, o plug-in de `CNI` instalado é o `Cilium`, portanto, você não precisa criar a pasta nem o arquivo. Caso seja necessário personalizar o plug-in de `CNI`, use o arquivo `server.yaml` na pasta `kubernetes/config`, que contém as seguintes informações:

```
cni:
- multus
- cilium
```

Nota

Esse é um arquivo opcional para definir personalizações do Kubernetes, como os plug-ins de CNI usados ou várias opções que você pode conferir na documentação oficial (https://docs.rke2.io/install/configuration).

A pasta `kubernetes/manifests` contém os seguintes arquivos:

- `rke2-ingress-config.yaml`: contém a configuração para criar o serviço `Ingress` para o cluster de gerenciamento (sem necessidade de modificação).

```
apiVersion: helm.cattle.io/v1
kind: HelmChartConfig
metadata:
  name: rke2-ingress-nginx
  namespace: kube-system
spec:
  valuesContent: |-
    controller:
      config:
        use-forwarded-headers: "true"
        enable-real-ip: "true"
      publishService:
        enabled: true
      service:
```

```
        enabled: true
        type: LoadBalancer
        externalTrafficPolicy: Local
```

- `neuvector-namespace.yaml`: contém a configuração para criar o namespace `NeuVector` (sem necessidade de modificação).

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    pod-security.kubernetes.io/enforce: privileged
  name: neuvector
```

- `ingress-l2-adv.yaml`: contém a configuração para criar o `L2Advertisement` para o componente `MetalLB` (sem necessidade de modificação).

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: ingress-l2-adv
  namespace: metallb-system
spec:
  ipAddressPools:
    - ingress-ippool
```

- `ingress-ippool.yaml`: contém a configuração para criar o `IPAddressPool` para o componente `rke2-ingress-nginx`. O `${INGRESS_VIP}` deve ser definido apropriadamente para especificar o endereço IP reservado que o componente `rke2-ingress-nginx` usará.

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: ingress-ippool
  namespace: metallb-system
spec:
  addresses:
    - ${INGRESS_VIP}/32
  serviceAllocation:
    priority: 100
    serviceSelectors:
      - matchExpressions:
          - {key: app.kubernetes.io/name, operator: In, values: [rke2-ingress-
nginx]}
```

A pasta `kubernetes/helm/values` contém os seguintes arquivos:

- `rancher.yaml`: contém a configuração para criar o componente `Rancher`. É necessário definir apropriadamente o `${INGRESS_VIP}` para especificar o endereço IP que o componente `Rancher` consumirá. O URL para acessar o componente `Rancher` será `https://rancher-${INGRESS_VIP}.sslip.io`.

```
hostname: rancher-${INGRESS_VIP}.sslip.io
bootstrapPassword: "foobar"
replicas: 1
global.cattle.psp.enabled: "false"
```

- `neuvector.yaml`: contém a configuração para criar o componente `NeuVector` (sem necessidade de modificação).

```
controller:
  replicas: 1
  ranchersso:
    enabled: true
manager:
  enabled: false
cve:
  scanner:
    enabled: false
    replicas: 1
k3s:
  enabled: true
crdwebhook:
  enabled: false
```

- `metal3.yaml`: contém a configuração para criar o componente `Metal3`. É necessário definir apropriadamente o `${METAL3_VIP}` para especificar o endereço IP que o componente `Metal3` consumirá.

```
global:
  ironicIP: ${METAL3_VIP}
  enable_vmedia_tls: false
  additionalTrustedCAs: false
metal3-ironic:
  global:
    predictableNicNames: "true"
  persistence:
    ironic:
      size: "5Gi"
```

Para implantar clusters downstream arm64 usando esse cluster de gerenciamento x86_64, você precisa adicionar a seguinte `deployArchitecture: arm64` à seção `global` do arquivo `metal3.yaml`:

```
global:
  ironicIP: ${METAL3_VIP}
  enable_vmedia_tls: false
  additionalTrustedCAs: false
  deployArchitecture: arm64
metal3-ironic:
  global:
    predictableNicNames: "true"
  persistence:
    ironic:
      size: "5Gi"
```

Nota

Na versão atual, existe uma limitação referente ao uso da `deployArchitecture: arm64`. Especificamente, se você permitir a implantação de clusters downstream arm64 usando essa diretiva, o cluster de gerenciamento apenas poderá implantar essa arquitetura no futuro. Para implantar clusters nas duas arquiteturas (x86_64 e arm64), provisione dois clusters de gerenciamento separados. Essa limitação será removida em uma versão futura.

Nota

O servidor de mídia é um recurso opcional incluído no Metal^3 (que está desabilitado por padrão). Para usar o recurso Metal3, configure-o no manifesto anterior. Para usar o servidor de mídia Metal^3, especifique a seguinte variável:

- adicione `enable_metal3_media_server` a `true` para habilitar o recurso de servidor de mídia na seção global.
- inclua a seguinte configuração sobre o servidor de mídia, em que ${MEDIA_VOLUME_PATH} é o caminho para o volume da mídia (por exemplo, `/home/metal3/bmh-image-cache`).

```
metal3-media:
  mediaVolume:
    hostPath: ${MEDIA_VOLUME_PATH}
```

É possível usar um servidor de mídia externo para armazenar as imagens e, caso você queira usá-lo com TLS, precisará modificar as seguintes configurações:

- defina `additionalTrustedCAs` como `true` no arquivo `metal3.yaml` anterior para permitir as CAs confiáveis adicionais do servidor de mídia externo.
- inclua a seguinte configuração de segredo na pasta `kubernetes/manifests/metal3-cacert-secret.yaml` para armazenar o certificado CA do servidor de mídia externo.

```
apiVersion: v1
kind: Namespace
metadata:
  name: metal3-system
---
apiVersion: v1
kind: Secret
metadata:
  name: tls-ca-additional
  namespace: metal3-system
type: Opaque
data:
  ca-additional.crt: {{ additional_ca_cert | b64encode }}
```

O `additional_ca_cert` é o certificado CA codificado em base64 do servidor de mídia externo. É possível usar o seguinte comando para codificar o certificado e gerar o segredo manualmente:

```
kubectl -n meta3-system create secret generic tls-ca-additional --from-file=ca-
additional.crt=./ca-additional.crt
```

- `certmanager.yaml`: contém a configuração para criar o componente `Cert-Manager` (sem necessidade de modificação).

```
installCRDs: "true"
```

40.3.5 Pasta de rede

A pasta `network` contém o mesmo número de arquivos e nós no cluster de gerenciamento. Em nosso caso, temos apenas um nó, portanto, temos somente um arquivo chamado `mgmt-cluster-node1.yaml`. O nome do arquivo deve ser igual ao nome de host especificado no arquivo de definição `mgmt-cluster.yaml` incluído na seção network/node descrita acima.

Se você precisa personalizar a configuração de rede, por exemplo, para usar um endereço IP estático específico (cenário sem DHCP), use o arquivo `mgmt-cluster-node1.yaml` na pasta `network`, que contém as seguintes informações:

- `${MGMT_GATEWAY}`: o endereço IP do gateway.
- `${MGMT_DNS}`: o endereço IP do servidor DNS.
- `${MGMT_MAC}`: o endereço MAC da interface de rede.
- `${MGMT_NODE_IP}`: o endereço IP do cluster de gerenciamento.

```
routes:
  config:
  - destination: 0.0.0.0/0
    metric: 100
    next-hop-address: ${MGMT_GATEWAY}
    next-hop-interface: eth0
    table-id: 254
dns-resolver:
  config:
    server:
    - ${MGMT_DNS}
    - 8.8.8.8
interfaces:
- name: eth0
  type: ethernet
  state: up
  mac-address: ${MGMT_MAC}
  ipv4:
    address:
    - ip: ${MGMT_NODE_IP}
      prefix-length: 24
    dhcp: false
```

```
    enabled: true
  ipv6:
    enabled: false
```

Para usar DHCP para obter o endereço IP, você pode definir a seguinte configuração (o endereço `MAC` deve ser devidamente definido por meio da variável `${MGMT_MAC}`):

```
## This is an example of a dhcp network configuration for a management cluster
interfaces:
- name: eth0
  type: ethernet
  state: up
  mac-address: ${MGMT_MAC}
  ipv4:
    dhcp: true
    enabled: true
  ipv6:
    enabled: false
```

Nota

- Dependendo do número de nós no cluster de gerenciamento, você poderá criar mais arquivos, como `mgmt-cluster-node2.yaml`, `mgmt-cluster-node3.yaml` etc., para configurar o restante dos nós.
- A seção `routes` é usada para definir a tabela de roteamento para o cluster de gerenciamento.

40.4 Preparação da imagem para ambientes air-gapped

Esta seção descreve como preparar a imagem para ambientes air-gapped mostrando apenas as diferenças das seções anteriores. As seguintes alterações na seção anterior (Preparação da imagem para ambientes conectados (*Seção 40.3, "Preparação da imagem para ambientes conectados"*)) são necessárias para preparar a imagem para os ambientes air-gapped:

- É necessário modificar o arquivo `mgmt-cluster.yaml` para incluir a seção `embeddedArtifactRegistry` com o campo `images` definido para que todas as imagens do contêiner sejam incluídas na imagem de saída do EIB.
- É necessário modificar o arquivo `mgmt-cluster.yaml` para incluir o gráfico Helm `rancher-turtles-airgap-resources`.
- É necessário remover o script `custom/scripts/99-register.sh` ao usar um ambiente air-gapped.

40.4.1 Modificações no arquivo de definição

É necessário modificar o arquivo `mgmt-cluster.yaml` para incluir a seção `embeddedArtifactRegistry`. Nessa seção, o campo `images` deve conter a lista de todas as imagens do contêiner que serão incluídas na imagem de saída.

Nota

Veja a seguir um exemplo do arquivo `mgmt-cluster.yaml` com a seção `embeddedArtifactRegistry` incluída. Garanta que as imagens listadas contenham as versões dos componentes necessárias.

É necessário também adicionar o gráfico Helm `rancher-turtles-airgap-resources` para criar os recursos conforme descrito na documentação do Rancher Turtles sobre ambientes air-gapped (https://documentation.suse.com/cloudnative/cluster-api/v0.19/en/getting-started/air-gapped-environment.html). Isso também requer um arquivo de valores turtles.yaml para que o gráfico rancher-turtles especifique a configuração necessária.

```
apiVersion: 1.2
image:
  imageType: iso
```

```
  arch: x86_64
  baseImage: SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso
  outputImageName: eib-mgmt-cluster-image.iso
operatingSystem:
  isoConfiguration:
    installDevice: /dev/sda
  users:
  - username: root
    encryptedPassword: $ROOT_PASSWORD
  packages:
    packageList:
    - jq
    sccRegistrationCode: $SCC_REGISTRATION_CODE
kubernetes:
  version: v1.32.4+rke2r1
  helm:
    charts:
      - name: cert-manager
        repositoryName: jetstack
        version: 1.15.3
        targetNamespace: cert-manager
        valuesFile: certmanager.yaml
        createNamespace: true
        installationNamespace: kube-system
      - name: longhorn-crd
        version: 106.2.0+up1.8.1
        repositoryName: rancher-charts
        targetNamespace: longhorn-system
        createNamespace: true
        installationNamespace: kube-system
      - name: longhorn
        version: 106.2.0+up1.8.1
        repositoryName: rancher-charts
        targetNamespace: longhorn-system
        createNamespace: true
        installationNamespace: kube-system
      - name: metal3
        version: 303.0.7+up0.11.5
        repositoryName: suse-edge-charts
        targetNamespace: metal3-system
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: metal3.yaml
      - name: rancher-turtles
        version: 303.0.4+up0.20.0
        repositoryName: suse-edge-charts
        targetNamespace: rancher-turtles-system
```

```
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: turtles.yaml
      - name: rancher-turtles-airgap-resources
        version: 303.0.4+up0.20.0
        repositoryName: suse-edge-charts
        targetNamespace: rancher-turtles-system
        createNamespace: true
        installationNamespace: kube-system
      - name: neuvector-crd
        version: 106.0.1+up2.8.6
        repositoryName: rancher-charts
        targetNamespace: neuvector
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: neuvector.yaml
      - name: neuvector
        version: 106.0.1+up2.8.6
        repositoryName: rancher-charts
        targetNamespace: neuvector
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: neuvector.yaml
      - name: rancher
        version: 2.11.2
        repositoryName: rancher-prime
        targetNamespace: cattle-system
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: rancher.yaml
    repositories:
      - name: jetstack
        url: https://charts.jetstack.io
      - name: rancher-charts
        url: https://charts.rancher.io/
      - name: suse-edge-charts
        url: oci://registry.suse.com/edge/charts
      - name: rancher-prime
        url: https://charts.rancher.com/server-charts/prime
    network:
      apiHost: $API_HOST
      apiVIP: $API_VIP
    nodes:
    - hostname: mgmt-cluster-node1
      initializer: true
      type: server
#   - hostname: mgmt-cluster-node2
```

```
#     type: server
#   - hostname: mgmt-cluster-node3
#     type: server
#       type: server
embeddedArtifactRegistry:
  images:
    - name: registry.suse.com/rancher/hardened-cluster-autoscaler:v1.9.0-build20241203
    - name: registry.suse.com/rancher/hardened-cni-plugins:v1.6.2-build20250306
    - name: registry.suse.com/rancher/hardened-coredns:v1.12.1-build20250401
    - name: registry.suse.com/rancher/hardened-k8s-metrics-server:v0.7.2-build20250110
    - name: registry.suse.com/rancher/hardened-multus-cni:v4.2.0-build20250326
    - name: registry.suse.com/rancher/klipper-helm:v0.9.5-build20250306
    - name: registry.suse.com/rancher/mirrored-cilium-cilium:v1.17.3
    - name: registry.suse.com/rancher/mirrored-cilium-operator-generic:v1.17.3
    - name: registry.suse.com/rancher/mirrored-longhornio-csi-attacher:v4.8.1
    - name: registry.suse.com/rancher/mirrored-longhornio-csi-node-driver-
registrar:v2.13.0
    - name: registry.suse.com/rancher/mirrored-longhornio-csi-provisioner:v5.2.0
    - name: registry.suse.com/rancher/mirrored-longhornio-csi-resizer:v1.13.2
    - name: registry.suse.com/rancher/mirrored-longhornio-csi-snapshotter:v8.2.0
    - name: registry.suse.com/rancher/mirrored-longhornio-livenessprobe:v2.15.0
    - name: registry.suse.com/rancher/mirrored-longhornio-longhorn-engine:v1.8.1
    - name: registry.suse.com/rancher/mirrored-longhornio-longhorn-instance-
manager:v1.8.1
    - name: registry.suse.com/rancher/mirrored-longhornio-longhorn-manager:v1.8.1
    - name: registry.suse.com/rancher/mirrored-longhornio-longhorn-share-manager:v1.8.1
    - name: registry.suse.com/rancher/mirrored-longhornio-longhorn-ui:v1.8.1
    - name: registry.suse.com/rancher/mirrored-sig-storage-snapshot-controller:v8.2.0
    - name: registry.suse.com/rancher/neuvector-compliance-config:1.0.5
    - name: registry.suse.com/rancher/neuvector-controller:5.4.4
    - name: registry.suse.com/rancher/neuvector-enforcer:5.4.4
    - name: registry.suse.com/rancher/nginx-ingress-controller:v1.12.1-hardened3
    - name: registry.rancher.com/rancher/cluster-api-addon-provider-fleet:v0.10.0
    - name: registry.rancher.com/rancher/cluster-api-operator:v0.17.0
    - name: registry.rancher.com/rancher/fleet-agent:v0.12.3
    - name: registry.rancher.com/rancher/fleet:v0.12.3
    - name: registry.rancher.com/rancher/hardened-node-feature-discovery:v0.15.7-
build20250425
    - name: registry.rancher.com/rancher/rancher-webhook:v0.7.2
    - name: registry.rancher.com/rancher/rancher/turtles:v0.20.0
    - name: registry.rancher.com/rancher/rancher:v2.11.2
    - name: registry.rancher.com/rancher/shell:v0.4.1
    - name: registry.rancher.com/rancher/system-upgrade-controller:v0.15.2
    - name: registry.suse.com/rancher/cluster-api-controller:v1.9.5
    - name: registry.suse.com/rancher/cluster-api-provider-metal3:v1.9.3
    - name: registry.suse.com/rancher/cluster-api-provider-rke2-bootstrap:v0.16.1
    - name: registry.suse.com/rancher/cluster-api-provider-rke2-controlplane:v0.16.1
```

```
    - name: registry.suse.com/rancher/hardened-sriov-network-operator:v1.5.0-
build20250425
    - name: registry.suse.com/rancher/ip-address-manager:v1.9.4
    - name: registry.rancher.com/rancher/kubectl:v1.32.2
```

40.4.2 Modificações na pasta custom

- É necessário remover o script `custom/scripts/99-register.sh` ao usar um ambiente air-gapped. Conforme você pode observar na estrutura de diretórios, o script `99-register.sh` não está incluído na pasta `custom/scripts`.

40.4.3 Modificações na pasta de valores do Helm

- O `turtles.yaml`: contém a configuração necessária para especificar a operação air-gapped para o Rancher Turtles. Observe que isso depende da instalação do gráfico rancher-turtles-airgap-resources.

```
cluster-api-operator:
  cluster-api:
    core:
      fetchConfig:
        selector: "{\"matchLabels\": {\"provider-components\": \"core\"}}"
    rke2:
      bootstrap:
        fetchConfig:
          selector: "{\"matchLabels\": {\"provider-components\": \"rke2-bootstrap
\"}}"
      controlPlane:
        fetchConfig:
          selector: "{\"matchLabels\": {\"provider-components\": \"rke2-control-
plane\"}}"
    metal3:
      infrastructure:
        fetchConfig:
```

```
            selector: "{\"matchLabels\": {\"provider-components\": \"metal3\"}}"
```

40.5 Criação de imagem

Depois que a estrutura de diretórios foi preparada de acordo com as seções anteriores (para os cenários tanto conectados quanto air-gapped), execute o comando abaixo para criar a imagem:

```
podman run --rm --privileged -it -v $PWD:/eib \
 registry.suse.com/edge/3.3/edge-image-builder:1.2.1 \
 build --definition-file mgmt-cluster.yaml
```

Isso cria o arquivo de imagem ISO de saída que, com base na definição da imagem descrita acima, no nosso caso é `eib-mgmt-cluster-image.iso`.

40.6 Provisionar o cluster de gerenciamento

A imagem anterior contém todos os componentes explicados acima e pode ser usada para provisionar o cluster de gerenciamento usando uma máquina virtual ou um servidor bare metal (com o recurso de mídia virtual).

41 Configuração dos recursos de telecomunicações

Esta seção documenta e explica a configuração dos recursos específicos de telecomunicações nos clusters implantados pelo SUSE Edge for Telco.

O método de implantação de provisionamento de rede direcionado é usado, conforme descrito na seção sobre provisionamento automatizado (*Capítulo 42, Provisionamento de rede direcionado totalmente automatizado*).

Os seguintes tópicos são abordados nesta seção:

- Imagem do kernel para tempo real (*Seção 41.1, "Imagem do kernel para tempo real"*): que será usada pelo kernel Real-Time.
- Argumentos do kernel para baixa latência e alto desempenho (*Seção 41.2, "Argumentos do kernel para baixa latência e alto desempenho"*): usados pelo kernel Real-Time para máximo desempenho e baixa latência na execução de cargas de trabalho de telecomunicações.
- Fixação de CPU via TuneD e argumentos do kernel (*Seção 41.3, "Fixação de CPU via TuneD e argumentos do kernel"*): isolamento de CPUs por meio de argumentos do kernel e perfil do TuneD.
- Configuração da CNI (*Seção 41.4, "Configuração da CNI"*): usada pelo cluster Kubernetes.
- Configuração do SR-IOV (*Seção 41.5, "SR-IOV"*): usada pelas cargas de trabalho Kubernetes.
- Configuração do DPDK (*Seção 41.6, "DPDK"*): usada pelo sistema.
- Placa aceleradora vRAN (*Seção 41.7, "Aceleração vRAN (`Intel ACC100/ACC200`)"*): configuração da placa aceleradora usada pelas cargas de trabalho Kubernetes.
- HugePages (*Seção 41.8, "HugePages"*): configuração do HugePages usada pelas cargas de trabalho Kubernetes.
- Fixação de CPU no Kubernetes (*Seção 41.9, "Fixação da CPU em Kubernetes"*): configuração do Kubernetes e de aplicativos para aproveitar a fixação de CPU.
- Configuração da programação com reconhecimento de NUMA (*Seção 41.10, "Programação com reconhecimento de NUMA"*): usada pelas cargas de trabalho Kubernetes.
- Configuração do Metal LB (*Seção 41.11, "MetalLB"*): usada pelas cargas de trabalho Kubernetes.

- Configuração do registro particular (*Seção 41.12, "Configuração do registro particular"*): usada pelas cargas de trabalho Kubernetes.
- Configuração do Precision Time Protocol (*Seção 41.13, "Precision Time Protocol"*): arquivos de configuração para execução de perfis de telecomunicações PTP.

41.1 Imagem do kernel para tempo real

A imagem do kernel Real-Time não é necessariamente melhor do que o kernel padrão. Trata-se de um kernel diferente adaptado a um caso de uso específico. O kernel Real-Time é adaptado para latência mais baixa em detrimento da taxa de transferência. Ele não é recomendado para fins de uso geral; mas, no nosso caso, esse é o kernel recomendado para cargas de trabalho de telecomunicações em que a latência é um fator importante.

Há quatro recursos principais:

- Execução determinística:
  Aumente a previsibilidade: garanta que os processos de negócios críticos sejam sempre concluídos dentro do prazo e ofereçam um serviço de alta qualidade, mesmo com cargas pesadas do sistema. Ao proteger os recursos importantes do sistema nos processos de alta prioridade, você garante maior previsibilidade para aplicativos urgentes.
- Baixa instabilidade:
  A baixa instabilidade decorrente da tecnologia altamente determinística ajuda a manter a sincronização dos aplicativos com o mundo real. Isso ajuda os serviços que precisam de cálculo contínuo e repetido.
- Herança de prioridade:
  A herança de prioridade refere-se à capacidade de um processo de prioridade mais baixa assumir a prioridade mais alta quando há um processo de maior prioridade que requer a conclusão do processo de menor prioridade para então finalizar sua tarefa. O SUSE Linux Enterprise Real Time resolve esses problemas de inversão de prioridade para processos de extrema importância.
- Interrupções de threads:
  Os processos executados no modo de interrupção em um sistema operacional de uso geral não são preemptíveis. Com o SUSE Linux Enterprise Real Time, essas interrupções foram encapsuladas pelos threads do kernel, que são interrompíveis e permitem a preempção de interrupções fixas e flexíveis por processos de prioridade mais alta definidos pelo usuário.

No nosso caso, se você instalou uma imagem em tempo real, como `SUSE Linux Micro RT`, o kernel Real-Time já está instalado. Você pode fazer download da imagem do kernel Real-Time pelo SUSE Customer Center (https://scc.suse.com/).

Nota

Para obter mais informações sobre o kernel Real-Time, visite SUSE Real Time (https://www.suse.com/products/realtime/).

41.2 Argumentos do kernel para baixa latência e alto desempenho

É importante configurar os argumentos do kernel para que o kernel Real-Time funcione corretamente, apresentando o melhor desempenho e a baixa latência para executar as cargas de trabalho de telecomunicações. Há alguns conceitos importantes para manter em mente na hora de configurar os argumentos do kernel para este caso de uso:

- Remova o `kthread_cpus` ao usar o kernel Real-Time da SUSE. Esse parâmetro controla em quais CPUs os threads do kernel serão criados. Ele também controla quais CPUs têm permissão para PID 1 e para carregar módulos do kernel (o auxiliar de espaço do usuário kmod). Esse parâmetro não é reconhecido e não tem nenhum efeito.

- Isole os núcleos da CPU usando `isolcpus`, `nohz_full`, `rcu_nocbs` e `irqaffinity`. Para acessar a lista completa de técnicas de fixação de CPU, consulte o capítulo Fixação de CPU via TuneD e argumentos do kernel (*Seção 41.3, "Fixação de CPU via TuneD e argumentos do kernel"*).

- Adicione os sinalizadores `domain,nohz,managed_irq` ao argumento do kernel `isolcpus`. Sem os sinalizadores, o `isolcpus` equivale a especificar apenas o sinalizador `domain`. Isso isola as CPUs especificadas da programação, incluindo as tarefas do kernel. O sinalizador `nohz` interrompe o tick do programador nas CPUs especificadas (se apenas uma tarefa for executável em determinada CPU), e o sinalizador `managed_irq` evita o roteamento de interrupções externas gerenciadas (dispositivos) nas CPUs especificadas. Observe que as linhas da IRQ dos dispositivos NVMe são totalmente gerenciadas pelo kernel e serão

roteadas para os núcleos não isolados (manutenção) como consequência. Por exemplo, a linha de comando inserida no fim desta seção resultará apenas em quatro filas (mais uma fila de admin/controle) alocadas no sistema:

```
for I in $(grep nvme0 /proc/interrupts | cut -d ':' -f1); do cat /proc/irq/${I}/
effective_affinity_list; done | column
39      0       19      20      39
```

Esse comportamento impede interrupções causadas por E/S do disco em qualquer aplicativo urgente executado nos núcleos isolados, mas pode exigir atenção e definição cuidadosa para cargas de trabalho com foco em armazenamento.

- Ajuste os ticks (interrupções periódicas do temporizador do kernel):
    - `skew_tick=1`: os ticks às vezes podem ocorrer ao mesmo tempo. Em vez de todas as CPUs receberem o tick do temporizador exatamente no mesmo momento, o `skew_tick=1` faz com que ele ocorra em horários um pouco diferentes. Isso ajuda a reduzir a instabilidade do sistema, resultando em tempos de resposta mais consistentes e com menos interrupções (um requisito essencial para aplicativos sensíveis à latência).
    - `nohz=on`: interrompe o tick periódico do temporizador em CPUs ociosas.
    - `nohz_full=<núcleos-cpu>`: interrompe o tick periódico do temporizador nas CPUs especificadas que são dedicadas a aplicativos em tempo real.
- Desabilite o processamento de exceção de verificação de máquina (MCE, Machine Check Exception) especificando `mce=off`. As MCEs são erros do hardware detectados pelo processador, e sua desabilitação pode evitar registros com muito ruído.
- Adicione `nowatchdog` para desabilitar o watchdog de bloqueio flexível que é implementado como um temporizador em execução no contexto de interrupção fixa do temporizador. Quando ele expira (ou seja, um bloqueio flexível é detectado), um aviso é exibido (no contexto de interrupção fixa), executando os destinos de latência. Mesmo que nunca expire, ele é incluído na lista do temporizador, o que aumenta levemente a sobrecarga de cada interrupção do temporizador. Essa opção também desabilita o watchdog de NMI, assim as NMIs não podem interferir.
- `nmi_watchdog=0` desabilita o watchdog de NMI (interrupção não mascarável). Para omiti-lo, use `nowatchdog`.

- RCU (Read-Copy-Update, Ler-Copiar-Atualizar) é um mecanismo que permite o acesso simultâneo e sem bloqueio de vários leitores aos dados compartilhados. O retorno de chamada RCU, função acionada após um "período extra", garante que todos os leitores anteriores tenham finalizado para que os dados antigos possam ser recuperados com segurança. Ajustamos o RCU, especificamente para cargas de trabalho confidenciais, para descarregar esses retornos de chamada das CPUs dedicadas (fixadas), evitando que as operações do kernel interfiram em tarefas críticas e urgentes.
    - Especifique as CPUs fixadas em `rcu_nocbs` para que os retornos de chamada RCU não sejam executados nelas. Isso ajuda a reduzir a instabilidade e a latência para cargas de trabalho em tempo real.
    - O `rcu_nocb_poll` faz com que as CPUs sem retorno de chamada realizem sondagens regulares para ver se há necessidade de gerenciar retornos de chamadas. Isso pode reduzir a sobrecarga de interrupções.
    - `rcupdate.rcu_cpu_stall_suppress=1` suprime os avisos de parada de RCU da CPU, que às vezes podem ser falsos positivos nos sistemas em tempo real com carga elevada.
    - `rcupdate.rcu_expedited=1` acelera o período extra das operações RCU, o que torna as seções críticas do lado da leitura mais responsivas.
    - `rcupdate.rcu_normal_after_boot=1`, quando usado com rcu_expedited, permite que o RCU volte à operação normal (não acelerada) após a inicialização do sistema.
    - `rcupdate.rcu_task_stall_timeout=0` desabilita o detector de paradas de tarefas do RCU, evitando possíveis avisos ou paralisações do sistema provocadas por tarefas do RCU de longa duração.
    - `rcutree.kthread_prio=99` define a prioridade do thread do kernel de retorno de chamada RCU como a mais alta possível (99), garantindo que ele seja programado e processe os retornos de chamada RCU prontamente, quando necessário.
- Adicione o `ignition.platform.id=openstack` para que o Metal3 e a Cluster API provisionem/desprovisionem o cluster com sucesso. Ele é usado pelo agente Metal3 Python, que teve sua origem no Openstack Ironic.

- Remova `intel_pstate=passive`. Essa opção configura o `intel_pstate` para operar com controladores cpufreq genéricos. No entanto, para que isso funcione, ele desabilita os estados P gerenciados pelo hardware (`HWP`) como efeito colateral. Para reduzir a latência do hardware, essa opção não é recomendada para cargas de trabalho em tempo real.
- Substitua `intel_idle.max_cstate=0 processor.max_cstate=1` por `idle=poll`. Para evitar transições de estado C, a opção `idle=poll` é usada para desabilitar essas transições e manter a CPU no estado C mais alto. A opção `intel_idle.max_cstate=0` desabilita `intel_idle` para que `acpi_idle` seja usado e, em seguida, o `acpi_idle.max_cstate=1` define o estado C máximo para acpi_idle. Nas arquiteturas AMD64/Intel 64, o primeiro estado C da ACPI sempre é `POLL`, mas ela usa uma função `poll_idle()`, que pode gerar uma pequena latência com a leitura periódica do relógio e a reinicialização do loop principal em `do_idle()` após o tempo limite (isso também envolve limpar e definir o sinalizador da tarefa `TIF_POLL`). Por outro lado, `idle=poll` é executado em um loop restrito, mantendo-se em espera ocupada até que uma tarefa seja reprogramada. Isso minimiza a latência de sair do estado ocioso, mas à custa de manter a CPU em execução na máxima velocidade no thread ocioso.
- Desabilite C1E no BIOS. Essa opção é importante para desabilitar o estado C1E no BIOS para evitar que a CPU entre no estado C1E quando estiver ociosa. C1E é um estado de baixo consumo que pode gerar latência quando a CPU está ociosa.

O restante desta documentação aborda parâmetros adicionais, incluindo HugePages e IOMMU.

Este é um exemplo de argumentos do kernel para um servidor Intel de 32 núcleos, incluindo os ajustes já mencionados:

```
$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-6.4.0-9-rt root=UUID=77b713de-5cc7-4d4c-8fc6-f5eca0a43cf9
 skew_tick=1 rd.timeout=60 rd.retry=45 console=ttyS1,115200 console=tty0
 default_hugepagesz=1G hugepagesz=1G hugepages=40 hugepagesz=2M hugepages=0
 ignition.platform.id=openstack intel_iommu=on iommu=pt irqaffinity=0,31,32,63
 isolcpus=domain,nohz,managed_irq,1-30,33-62 nohz_full=1-30,33-62 nohz=on
 mce=off net.ifnames=0 nosoftlockup nowatchdog nmi_watchdog=0 quiet rcu_nocb_poll
 rcu_nocbs=1-30,33-62 rcupdate.rcu_cpu_stall_suppress=1 rcupdate.rcu_expedited=1
 rcupdate.rcu_normal_after_boot=1 rcupdate.rcu_task_stall_timeout=0
 rcutree.kthread_prio=99 security=selinux selinux=1 idle=poll
```

Este é outro exemplo de configuração para um servidor AMD de 64 núcleos. Dentre os 128 processadores lógicos (`0-127`), os primeiros 8 núcleos (`0-7`) são destinados à manutenção, enquanto os 120 núcleos restantes (`8-127`) são fixados para os aplicativos:

```
$ cat /proc/cmdline
```

```
BOOT_IMAGE=/boot/vmlinuz-6.4.0-9-rt root=UUID=575291cf-74e8-42cf-8f2c-408a20dc00b8
 skew_tick=1 console=ttyS1,115200 console=tty0 default_hugepagesz=1G hugepagesz=1G
 hugepages=40 hugepagesz=2M hugepages=0 ignition.platform.id=openstack amd_iommu=on
 iommu=pt irqaffinity=0-7 isolcpus=domain,nohz,managed_irq,8-127 nohz_full=8-127
 rcu_nocbs=8-127 mce=off nohz=on net.ifnames=0 nowatchdog nmi_watchdog=0 nosoftlockup
 quiet rcu_nocb_poll rcupdate.rcu_cpu_stall_suppress=1 rcupdate.rcu_expedited=1
 rcupdate.rcu_normal_after_boot=1 rcupdate.rcu_task_stall_timeout=0
 rcutree.kthread_prio=99 security=selinux selinux=1 idle=poll
```

41.3 Fixação de CPU via TuneD e argumentos do kernel

`tuned` é uma ferramenta de ajuste de sistema que monitora as condições do sistema para otimizar o desempenho usando vários perfis predefinidos. Um recurso importante é sua capacidade de isolar os núcleos da CPU para cargas de trabalho específicas, como os aplicativos em tempo real. Isso impede que o sistema operacional use esses núcleos e, possivelmente, aumente a latência.

Para habilitar e configurar esse recurso, a primeira etapa é criar um perfil para os núcleos da CPU que desejamos isolar. Neste exemplo, dos 64 núcleos, dedicamos 60 (`1-30,33-62`) para o aplicativo e os 4 restantes são usados para manutenção. Observe que o design das CPUs isoladas depende significativamente dos aplicativos em tempo real.

```
$ echo "export tuned_params" >> /etc/grub.d/00_tuned

$ echo "isolated_cores=1-30,33-62" >> /etc/tuned/cpu-partitioning-variables.conf

$ tuned-adm profile cpu-partitioning
Tuned (re)started, changes applied.
```

Na sequência, precisamos modificar a opção GRUB para isolar os núcleos da CPU e outros parâmetros importantes para uso da CPU. É importante personalizar as seguintes opções com suas especificações de hardware atuais:

parâmetro	valor	descrição
isolcpus	domain,nohz,managed_irq,1-30,33-62	Isola os núcleos 1-30 e 33-62. `domain` indica que as CPUs fazem parte do domínio de isolamento. `nohz` permite a operação sem ticks nessas CPUs isoladas quando

parâmetro	valor	descrição
		estão ociosas, para reduzir interrupções. `managed_irq` isola as CPUs fixadas para que não sejam destinos das IRQs. Isso contempla o `irqaffinity=0-7`, que já direciona a maiorias das IRQs para os núcleos de manutenção.
skew_tick	1	Essa opção permite que o kernel distribua as interrupções do temporizador entre as CPUs isoladas.
nohz	on	Quando habilitada, a interrupção periódica do temporizador do kernel ("tick") vai parar em qualquer núcleo da CPU que esteja ocioso. Isso beneficia principalmente as CPUs de manutenção (`0,31,32,63`), além de economizar energia e reduzir ativações desnecessárias nos núcleos de uso geral.
nohz_full	1-30,33-62	Para os núcleos isolados, esse processo interrompe o tick, mesmo quando a CPU está executando uma única tarefa ativa. Dessa forma, ele faz com que a CPU seja executada no modo sem ticks

parâmetro	valor	descrição
		total (ou "dyntick"). O kernel apenas enviará interrupções do temporizador quando forem de fato necessárias.
rcu_nocbs	1-30,33-62	Essa opção descarrega o processamento de retorno de chamada RCU dos núcleos especificados da CPU.
rcu_nocb_poll		Quando essa opção é definida, as CPUs sem retorno de chamada RCU fazem uma sondagem regular para ver se o processamento de retornos de chamada é necessário, em vez de ser explicitamente ativadas por outras CPUs. Isso pode reduzir a sobrecarga das interrupções.
irqaffinity	0,31,32,63	Essa opção permite que o kernel execute as interrupções nos núcleos de manutenção.
idle	poll	Isso minimiza a latência de sair do estado ocioso, mas à custa de manter a CPU em execução na velocidade máxima no thread ocioso.
nmi_watchdog	0	Essa opção desabilita apenas o watchdog de NMI. Para omiti-la, defina `nowatchdog`.

parâmetro	valor	descrição
nowatchdog		Essa opção desabilita o watchdog de bloqueio flexível, que é implementado como um temporizador executado no contexto de interrupção fixa do temporizador.

Os seguintes comandos modificam a configuração do GRUB e aplicam as alterações mencionadas acima para que estejam presentes na próxima inicialização:

Edite o arquivo `/etc/default/grub` com os parâmetros acima, e ele terá esta aparência:

```
GRUB_CMDLINE_LINUX="BOOT_IMAGE=/boot/vmlinuz-6.4.0-9-rt
 root=UUID=77b713de-5cc7-4d4c-8fc6-f5eca0a43cf9 skew_tick=1 rd.timeout=60 rd.retry=45
 console=ttyS1,115200 console=tty0 default_hugepagesz=1G hugepagesz=1G hugepages=40
 hugepagesz=2M hugepages=0 ignition.platform.id=openstack intel_iommu=on iommu=pt
 irqaffinity=0,31,32,63 isolcpus=domain,nohz,managed_irq,1-30,33-62 nohz_full=1-30,33-62
 nohz=on mce=off net.ifnames=0 nosoftlockup nowatchdog nmi_watchdog=0 quiet rcu_nocb_poll
 rcu_nocbs=1-30,33-62 rcupdate.rcu_cpu_stall_suppress=1 rcupdate.rcu_expedited=1
 rcupdate.rcu_normal_after_boot=1 rcupdate.rcu_task_stall_timeout=0
 rcutree.kthread_prio=99 security=selinux selinux=1 idle=poll"
```

Atualize a configuração do GRUB:

```
$ transactional-update grub.cfg
$ reboot
```

Para validar a aplicação dos parâmetros após a reinicialização, é possível usar o seguinte comando para verificar a linha de comando do kernel:

```
$ cat /proc/cmdline
```

Existe outro script que pode ser usado para ajustar a configuração da CPU que, basicamente, executa as seguintes etapas:

- Definir o controlador da CPU como `performance`.
- Cancelar a definição da migração do temporizador para as CPUs isoladas.
- Migrar os threads kdaemon para as CPUs de manutenção.

- Definir a latência das CPUs isoladas como o valor mais baixo possível.
- Atrasar as atualizações de vmstat para 300 segundos.

O script está disponível no repositório de exemplos do SUSE Edge for Telco (https://raw.githubusercontent.com/suse-edge/atip/refs/heads/release-3.3/telco-examples/edge-clusters/dhcp-less/eib/custom/files/performance-settings.sh).

41.4 Configuração da CNI

41.4.1 Cilium

`Cilium` é o plug-in de CNI padrão para o SUSE Edge for Telco. Para habilitar o Cilium no cluster RKE2 como plug-in padrão, as seguintes configurações são necessárias no arquivo `/etc/rancher/rke2/config.yaml`:

```
cni:
- cilium
```

Também é possível especificar isso com argumentos de linha de comando, ou seja, `--cni=cilium` na linha do servidor no arquivo `/etc/systemd/system/rke2-server`.

Para usar o operador de rede `SR-IOV` descrito na próxima seção (*Seção 41.5, "SR-IOV"* (p 447)), use o `Multus` com outro plug-in de CNI, como `Cilium` ou `Calico`, como o plug-in secundário.

```
cni:
- multus
- cilium
```

Nota

Para obter mais informações sobre os plug-ins de CNI, visite Network Options (https://docs.rke2.io/install/network_options) (Opções de rede).

41.5 SR-IOV

O SR-IOV permite que um dispositivo, por exemplo, adaptador de rede, separe o acesso a seus recursos entre várias funções de hardware `PCIe`. Há diversas maneiras de implantar o `SR-IOV` e, neste documento, mostramos duas opções diferentes:

- Opção 1: usar os plug-ins de dispositivo CNI `SR-IOV` e um mapa de configuração para configurá-lo de maneira apropriada.
- Opção 2 (recomendada): usar o gráfico Helm do `SR-IOV` do Rancher Prime para facilitar a implantação.

Opção 1 – Instalação dos plug-ins de dispositivo CNI SR-IOV e um mapa de configuração para configurá-lo de maneira apropriada

- Preparar o mapa de configuração para o plug-in de dispositivo

Obtenha as informações para preencher o mapa de configuração executando o comando `lspci`:

```
$ lspci | grep -i acc
8a:00.0 Processing accelerators: Intel Corporation Device 0d5c

$ lspci | grep -i net
19:00.0 Ethernet controller: Broadcom Inc. and subsidiaries BCM57504 NetXtreme-E
 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet (rev 11)
19:00.1 Ethernet controller: Broadcom Inc. and subsidiaries BCM57504 NetXtreme-E
 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet (rev 11)
19:00.2 Ethernet controller: Broadcom Inc. and subsidiaries BCM57504 NetXtreme-E
 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet (rev 11)
19:00.3 Ethernet controller: Broadcom Inc. and subsidiaries BCM57504 NetXtreme-E
 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet (rev 11)
51:00.0 Ethernet controller: Intel Corporation Ethernet Controller E810-C for QSFP (rev
 02)
51:00.1 Ethernet controller: Intel Corporation Ethernet Controller E810-C for QSFP (rev
 02)
51:01.0 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev
 02)
51:01.1 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev
 02)
51:01.2 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev
 02)
51:01.3 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev
 02)
51:11.0 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev
 02)
```

```
51:11.1 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev
 02)
51:11.2 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev
 02)
51:11.3 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev
 02)
```

O mapa de configuração consiste em um arquivo `JSON` que descreve os dispositivos usando filtros para descobri-los e cria grupos para as interfaces. O principal é entender os filtros e os grupos. Os filtros são usados para descobrir os dispositivos, e os grupos para criar as interfaces.

É possível definir os filtros desta maneira:

- vendorID: `8086` (Intel)
- deviceID: `0d5c` (placa aceleradora)
- driver: `vfio-pci` (driver)
- pfNames: `p2p1` (nome da interface física)

É possível também definir os filtros para corresponder à uma sintaxe de interface mais complexa, por exemplo:

- pfNames: `["eth1#1,2,3,4,5,6"]` ou `[eth1#1-6]` (nome da interface física)

Em relação aos grupos, podemos criar um para a placa `FEC` e outro para a placa `Intel`, e até criar um prefixo dependendo do nosso caso de uso:

- resourceName: `pci_sriov_net_bh_dpdk`
- resourcePrefix: `Rancher.io`

Há inúmeras combinações para descobrir e criar o grupo de recursos para alocar algumas `VFs` aos pods.

Nota

Para obter mais informações sobre filtros e grupos, visite sr-iov network device plug-in (https://github.com/k8snetworkplumbingwg/sriov-network-device-plugin) (Plug-in de dispositivo de rede sr-iov).

Depois de definir os filtros e os grupos para corresponder as interfaces, dependendo do hardware e do caso de uso, o seguinte mapa de configuração mostrará um exemplo para ser usado:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: sriovdp-config
  namespace: kube-system
data:
  config.json: |
    {
        "resourceList": [
            {
                "resourceName": "intel_fec_5g",
                "devicetype": "accelerator",
                "selectors": {
                    "vendors": ["8086"],
                    "devices": ["0d5d"]
                }
            },
            {
                "resourceName": "intel_sriov_odu",
                "selectors": {
                    "vendors": ["8086"],
                    "devices": ["1889"],
                    "drivers": ["vfio-pci"],
                    "pfNames": ["p2p1"]
                }
            },
            {
                "resourceName": "intel_sriov_oru",
                "selectors": {
                    "vendors": ["8086"],
                    "devices": ["1889"],
                    "drivers": ["vfio-pci"],
                    "pfNames": ["p2p2"]
                }
            }
        ]
    }
```

- Preparar o arquivo `daemonset` para implantar o plug-in de dispositivo

O plug-in de dispositivo oferece suporte a várias arquiteturas (`arm`, `amd`, `ppc64le`), portanto, é possível usar o mesmo arquivo para arquiteturas diferentes ao implantar vários `daemonset` para cada arquitetura.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sriov-device-plugin
  namespace: kube-system
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: kube-sriov-device-plugin-amd64
  namespace: kube-system
  labels:
    tier: node
    app: sriovdp
spec:
  selector:
    matchLabels:
      name: sriov-device-plugin
  template:
    metadata:
      labels:
        name: sriov-device-plugin
        tier: node
        app: sriovdp
    spec:
      hostNetwork: true
      nodeSelector:
        kubernetes.io/arch: amd64
      tolerations:
      - key: node-role.kubernetes.io/master
        operator: Exists
        effect: NoSchedule
      serviceAccountName: sriov-device-plugin
      containers:
      - name: kube-sriovdp
        image: rancher/hardened-sriov-network-device-plugin:v3.7.0-build20240816
        imagePullPolicy: IfNotPresent
        args:
        - --log-dir=sriovdp
        - --log-level=10
        securityContext:
          privileged: true
        resources:
          requests:
            cpu: "250m"
            memory: "40Mi"
          limits:
```

```
          cpu: 1
          memory: "200Mi"
      volumeMounts:
      - name: devicesock
        mountPath: /var/lib/kubelet/
        readOnly: false
      - name: log
        mountPath: /var/log
      - name: config-volume
        mountPath: /etc/pcidp
      - name: device-info
        mountPath: /var/run/k8s.cni.cncf.io/devinfo/dp
    volumes:
      - name: devicesock
        hostPath:
          path: /var/lib/kubelet/
      - name: log
        hostPath:
          path: /var/log
      - name: device-info
        hostPath:
          path: /var/run/k8s.cni.cncf.io/devinfo/dp
          type: DirectoryOrCreate
      - name: config-volume
        configMap:
          name: sriovdp-config
          items:
          - key: config.json
            path: config.json
```

- Depois de aplicar o mapa de configuração e o `daemonset`, o plug-in de dispositivo será implantado, e as interfaces serão descobertas e estarão disponíveis para os pods.

```
$ kubectl get pods -n kube-system | grep sriov
kube-system  kube-sriov-device-plugin-amd64-twjfl  1/1  Running  0  2m
```

- Verifique as interfaces descobertas e disponíveis nos nós usados pelos pods:

```
$ kubectl get $(kubectl get nodes -oname) -o jsonpath='{.status.allocatable}' | jq
{
  "cpu": "64",
  "ephemeral-storage": "256196109726",
  "hugepages-1Gi": "40Gi",
  "hugepages-2Mi": "0",
  "intel.com/intel_fec_5g": "1",
  "intel.com/intel_sriov_odu": "4",
  "intel.com/intel_sriov_oru": "4",
```

```
  "memory": "221396384Ki",
  "pods": "110"
}
```

- O `FEC` é `intel.com/intel_fec_5g` e o valor é 1.
- A `VF` é `intel.com/intel_sriov_odu` ou `intel.com/intel_sriov_oru`, se você a implantar com um plug-in de dispositivo e um mapa de configuração sem gráficos Helm.

Importante

Se não há interfaces neste ponto, não faz muito sentido continuar porque a interface não estará disponível para os pods. Revise o mapa de configuração e os filtros para resolver o problema primeiro.

Opção 2 (recomendada) – Instalação usando o Rancher com gráficos Helm para plug-ins de dispositivo CNI SR-IOV

- Obtenha o Helm se não estiver presente:

```
$ curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
```

- Instale o SR-IOV.

```
helm install sriov-crd oci://registry.suse.com/edge/charts/sriov-crd -n sriov-network-
operator
helm install sriov-network-operator oci://registry.suse.com/edge/charts/sriov-network-
operator -n sriov-network-operator
```

- Verifique os recursos crd e os pods implantados:

```
$ kubectl get crd
$ kubectl -n sriov-network-operator get pods
```

- Verifique o rótulo nos nós.

Com todos os recursos em execução, o rótulo aparecerá automaticamente em seu nó:

```
$ kubectl get nodes -oyaml | grep feature.node.kubernetes.io/network-sriov.capable

feature.node.kubernetes.io/network-sriov.capable: "true"
```

- Revise o `daemonset` para ver os novos `sriov-network-config-daemon` e `sriov-rancher-nfd-worker` já ativos e prontos:

```
$ kubectl get daemonset -A
NAMESPACE             NAME                              DESIRED   CURRENT   READY   UP-TO-
DATE   AVAILABLE   NODE SELECTOR                                          AGE
calico-system             calico-node                       1         1         1       1
         1           kubernetes.io/os=linux                                 15h
sriov-network-operator    sriov-network-config-daemon       1         1         1       1
         1           feature.node.kubernetes.io/network-sriov.capable=true   45m
sriov-network-operator    sriov-rancher-nfd-worker          1         1         1       1
         1           <none>                                                 45m
kube-system               rke2-ingress-nginx-controller     1         1         1       1
         1           kubernetes.io/os=linux                                 15h
kube-system               rke2-multus-ds                    1         1         1       1
         1           kubernetes.io/arch=amd64,kubernetes.io/os=linux        15h
```

Em poucos minutos (a atualização pode levar até 10 minutos), os nós são detectados e configurados com os recursos do `SR-IOV`:

```
$ kubectl get sriovnetworknodestates.sriovnetwork.openshift.io -A
NAMESPACE             NAME     AGE
sriov-network-operator   xr11-2   83s
```

- Verifique as interfaces detectadas.

As interfaces descobertas devem ser o endereço PCI do dispositivo de rede. Verifique essa informação com o comando `lspci` no host.

```
$ kubectl get sriovnetworknodestates.sriovnetwork.openshift.io -n kube-system -oyaml
apiVersion: v1
items:
- apiVersion: sriovnetwork.openshift.io/v1
  kind: SriovNetworkNodeState
  metadata:
    creationTimestamp: "2023-06-07T09:52:37Z"
    generation: 1
    name: xr11-2
    namespace: sriov-network-operator
    ownerReferences:
    - apiVersion: sriovnetwork.openshift.io/v1
      blockOwnerDeletion: true
      controller: true
      kind: SriovNetworkNodePolicy
      name: default
```

```
      uid: 80b72499-e26b-4072-a75c-f9a6218ec357
    resourceVersion: "356603"
    uid: e1f1654b-92b3-44d9-9f87-2571792cc1ad
  spec:
    dpConfigVersion: "356507"
  status:
    interfaces:
    - deviceID: "1592"
      driver: ice
      eSwitchMode: legacy
      linkType: ETH
      mac: 40:a6:b7:9b:35:f0
      mtu: 1500
      name: p2p1
      pciAddress: "0000:51:00.0"
      totalvfs: 128
      vendor: "8086"
    - deviceID: "1592"
      driver: ice
      eSwitchMode: legacy
      linkType: ETH
      mac: 40:a6:b7:9b:35:f1
      mtu: 1500
      name: p2p2
      pciAddress: "0000:51:00.1"
      totalvfs: 128
      vendor: "8086"
    syncStatus: Succeeded
kind: List
metadata:
  resourceVersion: ""
```

Nota

Se a sua interface não foi detectada neste momento, verifique se ela está presente no próximo mapa de configuração:

```
$ kubectl get cm supported-nic-ids -oyaml -n sriov-network-operator
```

Se o seu dispositivo não estiver presente lá, edite o mapa de configuração adicionando os valores corretos a serem descobertos (deve ser necessário reiniciar o daemonset `sriov-network-config-daemon`).

- Crie a `política NetworkNode` para configurar as `VFs`.

Serão criadas algumas `VFs` (`numVfs`) do dispositivo (`rootDevices`), e ela será configurada com o `deviceType` do driver e a `MTU`:

Nota

O campo `resourceName` não deve conter caracteres especiais e deve ser exclusivo em todo o cluster. O exemplo usa `deviceType: vfio-pci` porque `dpdk` será usado em conjunto com `sr-iov`. Se você não usar `dpdk`, o deviceType deverá ser `deviceType: netdevice` (valor padrão).

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-dpdk
  namespace: sriov-network-operator
spec:
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  resourceName: intelnicsDpdk
  deviceType: vfio-pci
  numVfs: 8
  mtu: 1500
  nicSelector:
    deviceID: "1592"
    vendor: "8086"
    rootDevices:
    - 0000:51:00.0
```

- Valide as configurações:

```
$ kubectl get $(kubectl get nodes -oname) -o jsonpath='{.status.allocatable}' | jq
{
  "cpu": "64",
  "ephemeral-storage": "256196109726",
  "hugepages-1Gi": "60Gi",
  "hugepages-2Mi": "0",
  "intel.com/intel_fec_5g": "1",
  "memory": "200424836Ki",
  "pods": "110",
  "rancher.io/intelnicsDpdk": "8"
}
```

- Crie a rede sr-iov (opcional, caso seja necessária uma rede diferente):

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: network-dpdk
  namespace: sriov-network-operator
spec:
  ipam: |
    {
      "type": "host-local",
      "subnet": "192.168.0.0/24",
      "rangeStart": "192.168.0.20",
      "rangeEnd": "192.168.0.60",
      "routes": [{
        "dst": "0.0.0.0/0"
      }],
      "gateway": "192.168.0.1"
    }
  vlan: 500
  resourceName: intelnicsDpdk
```

- Verifique a rede criada:

```
$ kubectl get network-attachment-definitions.k8s.cni.cncf.io -A -oyaml

apiVersion: v1
items:
- apiVersion: k8s.cni.cncf.io/v1
  kind: NetworkAttachmentDefinition
  metadata:
    annotations:
      k8s.v1.cni.cncf.io/resourceName: rancher.io/intelnicsDpdk
    creationTimestamp: "2023-06-08T11:22:27Z"
    generation: 1
    name: network-dpdk
    namespace: sriov-network-operator
    resourceVersion: "13124"
    uid: df7c89f5-177c-4f30-ae72-7aef3294fb15
  spec:
    config: '{ "cniVersion":"0.4.0", "name":"network-
dpdk","type":"sriov","vlan":500,"vlanQoS":0,"ipam":{"type":"host-
local","subnet":"192.168.0.0/24","rangeStart":"192.168.0.10","rangeEnd":"192.168.0.60","routes":
[{"dst":"0.0.0.0/0"}],"gateway":"192.168.0.1"}
      }'
kind: List
metadata:
```

```
  resourceVersion: ""
```

41.6 DPDK

DPDK (Data Plane Development Kit) é um conjunto de bibliotecas e drivers para processamento rápido de pacotes. Ele é usado para acelerar as cargas de trabalho de processamento de pacotes executadas em uma ampla variedade de arquiteturas de CPU. O DPDK inclui as bibliotecas de plano de controle e os drivers otimizados de placa de interface de rede (NIC) para o seguinte:

1. Um gerenciador de fila que implementa filas sem bloqueio.
2. Um gerenciador de buffer que pré-aloca buffers de tamanho fixo.
3. Um gerenciador de memória que aloca pools de objetos na memória e usa um anel para armazenar objetos livres; garante que os objetos sejam igualmente distribuídos por todos os canais DRAM.
4. Drivers de modo de sondagem (PMD) desenvolvidos para operar sem notificações assíncronas, reduzindo a sobrecarga.
5. Uma estrutura de pacotes como um conjunto de bibliotecas auxiliares para desenvolver o processamento de pacotes.

As seguintes etapas mostram como habilitar o DPDK e criar VFs das NICs usadas pelas interfaces do DPDK:

- Instale o pacote DPDK:

```
$ transactional-update pkg install dpdk dpdk-tools libdpdk-23
$ reboot
```

- Parâmetros do kernel:

Para usar o DPDK, aplique alguns drivers para habilitar determinados parâmetros no kernel:

parâmetro	valor	descrição
iommu	pt	Essa opção permite usar o driver vfio para as interfaces do DPDK.

parâmetro	valor	descrição
intel_iommu ou amd_iommu	on	Essa opção permite usar o `vfio` para `VFs`.

Para habilitar os parâmetros, adicione-os ao arquivo `/etc/default/grub`:

```
GRUB_CMDLINE_LINUX="BOOT_IMAGE=/boot/vmlinuz-6.4.0-9-rt
 root=UUID=77b713de-5cc7-4d4c-8fc6-f5eca0a43cf9 skew_tick=1 rd.timeout=60 rd.retry=45
 console=ttyS1,115200 console=tty0 default_hugepagesz=1G hugepagesz=1G hugepages=40
 hugepagesz=2M hugepages=0 ignition.platform.id=openstack intel_iommu=on iommu=pt
 irqaffinity=0,31,32,63 isolcpus=domain,nohz,managed_irq,1-30,33-62 nohz_full=1-30,33-62
 nohz=on mce=off net.ifnames=0 nosoftlockup nowatchdog nmi_watchdog=0 quiet rcu_nocb_poll
 rcu_nocbs=1-30,33-62 rcupdate.rcu_cpu_stall_suppress=1 rcupdate.rcu_expedited=1
 rcupdate.rcu_normal_after_boot=1 rcupdate.rcu_task_stall_timeout=0
 rcutree.kthread_prio=99 security=selinux selinux=1 idle=poll"
```

Atualize a configuração do GRUB e reinicialize o sistema para aplicar as alterações:

```
$ transactional-update grub.cfg
$ reboot
```

- Carregue o módulo do kernel `vfio-pci` e habilite o `SR-IOV` nas `NICs`:

```
$ modprobe vfio-pci enable_sriov=1 disable_idle_d3=1
```

- Crie algumas funções virtuais (`VFs`) das `NICs`.

Por exemplo, para criar `VFs` para duas `NICs` diferentes, os seguintes comandos são necessários:

```
$ echo 4 > /sys/bus/pci/devices/0000:51:00.0/sriov_numvfs
$ echo 4 > /sys/bus/pci/devices/0000:51:00.1/sriov_numvfs
```

- Vincule as novas VFs ao driver `vfio-pci`:

```
$ dpdk-devbind.py -b vfio-pci 0000:51:01.0 0000:51:01.1 0000:51:01.2 0000:51:01.3 \
                              0000:51:11.0 0000:51:11.1 0000:51:11.2 0000:51:11.3
```

- Verifique se a configuração foi aplicada corretamente:

```
$ dpdk-devbind.py -s
```

```
Network devices using DPDK-compatible driver
============================================
0000:51:01.0 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci unused=iavf,igb_uio
0000:51:01.1 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci unused=iavf,igb_uio
0000:51:01.2 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci unused=iavf,igb_uio
0000:51:01.3 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci unused=iavf,igb_uio
0000:51:01.0 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci unused=iavf,igb_uio
0000:51:11.1 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci unused=iavf,igb_uio
0000:51:21.2 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci unused=iavf,igb_uio
0000:51:31.3 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci unused=iavf,igb_uio

Network devices using kernel driver
===================================
0000:19:00.0 'BCM57504 NetXtreme-E 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet 1751' if=em1
 drv=bnxt_en unused=igb_uio,vfio-pci *Active*
0000:19:00.1 'BCM57504 NetXtreme-E 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet 1751' if=em2
 drv=bnxt_en unused=igb_uio,vfio-pci
0000:19:00.2 'BCM57504 NetXtreme-E 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet 1751' if=em3
 drv=bnxt_en unused=igb_uio,vfio-pci
0000:19:00.3 'BCM57504 NetXtreme-E 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet 1751' if=em4
 drv=bnxt_en unused=igb_uio,vfio-pci
0000:51:00.0 'Ethernet Controller E810-C for QSFP 1592' if=eth13 drv=ice
 unused=igb_uio,vfio-pci
0000:51:00.1 'Ethernet Controller E810-C for QSFP 1592' if=rename8 drv=ice
 unused=igb_uio,vfio-pci
```

41.7 Aceleração vRAN (Intel ACC100/ACC200)

À medida que os provedores de serviços de comunicação migram da rede 4G para 5G, muitos deles estão adotando as arquiteturas de rede de acesso por rádio virtualizada (vRAN) para maior capacidade dos canais e implantação mais fácil dos serviços e aplicativos de borda. As soluções vRAN estão no local ideal para oferecer serviços de baixa latência com flexibilidade para aumentar ou reduzir a capacidade de acordo com o volume do tráfego e da demanda em tempo real na rede.

Uma das cargas de trabalho 4G e 5G com uso mais intenso de recursos é a FEC da RAN de camada 1 (L1), que resolve os erros de transmissão de dados por meio de canais de comunicação incertos ou ruidosos. A tecnologia FEC detecta e corrige um número limitado de erros em dados 4G ou 5G, eliminando a necessidade de retransmissão. Como a transação de aceleração da FEC não contém informações de estado da célula, é possível virtualizá-la com facilidade para aproveitar os benefícios dos agrupamentos e facilitar a migração de células.

- Parâmetros do kernel

Para habilitar a aceleração da vRAN, precisamos habilitar os seguintes parâmetros do kernel (se ainda não estiverem presentes):

parâmetro	valor	descrição
iommu	pt	Essa opção permite usar vfio paras as interfaces do DPDK.
intel_iommu ou amd_iommu	on	Essa opção permite usar vfio para VFs.

Modifique o arquivo GRUB `/etc/default/grub` para adicioná-los à linha de comando do kernel:

```
GRUB_CMDLINE_LINUX="BOOT_IMAGE=/boot/vmlinuz-6.4.0-9-rt
 root=UUID=77b713de-5cc7-4d4c-8fc6-f5eca0a43cf9 skew_tick=1 rd.timeout=60 rd.retry=45
 console=ttyS1,115200 console=tty0 default_hugepagesz=1G hugepagesz=1G hugepages=40
 hugepagesz=2M hugepages=0 ignition.platform.id=openstack intel_iommu=on iommu=pt
 irqaffinity=0,31,32,63 isolcpus=domain,nohz,managed_irq,1-30,33-62 nohz_full=1-30,33-62
 nohz=on mce=off net.ifnames=0 nosoftlockup nowatchdog nmi_watchdog=0 quiet rcu_nocb_poll
 rcu_nocbs=1-30,33-62 rcupdate.rcu_cpu_stall_suppress=1 rcupdate.rcu_expedited=1
 rcupdate.rcu_normal_after_boot=1 rcupdate.rcu_task_stall_timeout=0
 rcutree.kthread_prio=99 security=selinux selinux=1 idle=poll"
```

Atualize a configuração do GRUB e reinicialize o sistema para aplicar as alterações:

```
$ transactional-update grub.cfg
$ reboot
```

Para verificar se os parâmetros foram aplicados após a reinicialização, consulte a linha de comando:

```
$ cat /proc/cmdline
```

- Carregue os módulos do kernel vfio-pci para habilitar a aceleração da vRAN:

```
$ modprobe vfio-pci enable_sriov=1 disable_idle_d3=1
```

- Obtenha as informações da interface Acc100:

```
$ lspci | grep -i acc
```

```
8a:00.0 Processing accelerators: Intel Corporation Device 0d5c
```

- Vincule a interface física (PF) ao driver vfio-pci:

```
$ dpdk-devbind.py -b vfio-pci 0000:8a:00.0
```

- Crie as funções virtuais (VFs) da interface física (PF).

Crie 2 VFs da PF e vincule a elas o vfio-pci seguindo estas etapas:

```
$ echo 2 > /sys/bus/pci/devices/0000:8a:00.0/sriov_numvfs
$ dpdk-devbind.py -b vfio-pci 0000:8b:00.0
```

- Configure a acc100 com o arquivo de configuração proposto:

```
$ pf_bb_config ACC100 -c /opt/pf-bb-config/acc100_config_vf_5g.cfg
Tue Jun  6 10:49:20 2023:INFO:Queue Groups: 2 5GUL, 2 5GDL, 2 4GUL, 2 4GDL
Tue Jun  6 10:49:20 2023:INFO:Configuration in VF mode
Tue Jun  6 10:49:21 2023:INFO: ROM version MM 99AD92
Tue Jun  6 10:49:21 2023:WARN:* Note: Not on DDR PRQ version  1302020 != 10092020
Tue Jun  6 10:49:21 2023:INFO:PF ACC100 configuration complete
Tue Jun  6 10:49:21 2023:INFO:ACC100 PF [0000:8a:00.0] configuration complete!
```

- Verifique as novas VFs criadas da PF FEC:

```
$ dpdk-devbind.py -s
Baseband devices using DPDK-compatible driver
==============================================
0000:8a:00.0 'Device 0d5c' drv=vfio-pci unused=
0000:8b:00.0 'Device 0d5d' drv=vfio-pci unused=

Other Baseband devices
======================
0000:8b:00.1 'Device 0d5d' unused=
```

41.8 HugePages

Quando um processo usa RAM, a CPU a marca como usada por esse processo. Para manter a eficiência, a CPU aloca a RAM em blocos de 4K bytes, que é o valor padrão em muitas plataformas. Esses blocos são chamados de páginas. As páginas podem ser substituídas por discos, entre outros.

Como o espaço do endereço do processo é virtual, a `CPU` e o sistema operacional precisam memorizar quais páginas pertencem a qual processo, e onde cada página é armazenada. Quanto maior o número de páginas, mais longa a pesquisa de mapeamento de memória. Quando um processo usa `1 GB` de memória, isso equivale a 262144 entradas para pesquisa (`1 GB/4 K`). Se uma entrada da tabela de páginas consome 8 bytes, isso equivale a `2 MB` (262144 * 8) para pesquisa.

As arquiteturas de `CPU` mais atuais oferecem suporte às páginas maiores que o padrão, o que reduz o número de entradas para a `CPU/SO` pesquisar.

- Parâmetros do kernel

Para habilitar o HugePages, devemos adicionar os seguintes parâmetros do kernel. Neste exemplo, configuramos 40 páginas de 1G, portanto, o tamanho e o número exato de páginas enormes devem ser adaptados aos requisitos de memória do seu aplicativo:

parâmetro	**valor**	**descrição**
hugepagesz	1G	Essa opção permite definir o tamanho das páginas enormes como 1 G
hugepages	40	Esse é o número de páginas enormes definido antes
default_hugepagesz	1G	Esse é o valor padrão para obter as páginas enormes

Modifique o arquivo GRUB `/etc/default/grub` para adicionar esses parâmetros a `GRUB_CMDLINE_LINUX`:

```
default_hugepagesz=1G hugepagesz=1G hugepages=40 hugepagesz=2M hugepages=0
```

Atualize a configuração do GRUB e reinicialize o sistema para aplicar as alterações:

```
$ transactional-update grub.cfg
$ reboot
```

Para validar se os parâmetros foram aplicados após a reinicialização, verifique a linha de comando:

```
$ cat /proc/cmdline
```

- Usando o HugePages

Para usar o HugePages, precisamos montá-lo:

```
$ mkdir -p /hugepages
$ mount -t hugetlbfs nodev /hugepages
```

Implante a carga de trabalho Kubernetes criando os recursos e os volumes:

```
...
 resources:
   requests:
     memory: "24Gi"
     hugepages-1Gi: 16Gi
     intel.com/intel_sriov_oru: '4'
   limits:
     memory: "24Gi"
     hugepages-1Gi: 16Gi
     intel.com/intel_sriov_oru: '4'
...
```

```
...
volumeMounts:
  - name: hugepage
    mountPath: /hugepages
...
volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
...
```

41.9 Fixação da CPU em Kubernetes

41.9.1 Pré-requisito

A CPU deve estar ajustada de acordo com o perfil de desempenho abordado nesta seção (*Seção 41.3, "Fixação de CPU via TuneD e argumentos do kernel"*).

41.9.2 Configurar o Kubernetes para fixação da CPU

Configure os argumentos de kubelet para implementar o gerenciamento da CPU no cluster `RKE2`. Adicione o seguinte bloco de configuração, como no exemplo abaixo, ao arquivo `/etc/rancher/rke2/config.yaml`. Especifique os núcleos da CPU de manutenção nos argumentos `kubelet-reserved` e `system-reserved`:

```
kubelet-arg:
- "cpu-manager-policy=static"
- "cpu-manager-policy-options=full-pcpus-only=true"
- "cpu-manager-reconcile-period=0s"
- "kubelet-reserved=cpu=0,31,32,63"
- "system-reserved=cpu=0,31,32,63"
```

41.9.3 Aproveitando as CPUs fixadas para as cargas de trabalho

Há três maneiras de usar este recurso com a `política estática` definida no kubelet, dependendo das solicitações e dos limites definidos em sua carga de trabalho:

1. Classe de QoS `BestEffort`: se você não definir uma solicitação ou um limite de `CPU`, o pod será programado na primeira `CPU` disponível no sistema.
   Um exemplo de uso da classe de QoS `BestEffort` é:

   ```
   spec:
     containers:
     - name: nginx
       image: nginx
   ```

2. Classe de QoS `Burstable`: se você definir uma solicitação de CPU que não é igual aos limites, ou se não houver solicitações de CPU.
   Alguns exemplos de uso da classe de QoS `Burstable` são:

   ```
   spec:
     containers:
     - name: nginx
       image: nginx
       resources:
         limits:
           memory: "200Mi"
         requests:
           memory: "100Mi"
   ```

ou

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
        cpu: "2"
      requests:
        memory: "100Mi"
        cpu: "1"
```

3. Classe de QoS `Guaranteed`: se você definir uma solicitação de CPU igual aos limites. Um exemplo de uso da classe de QoS `Guaranteed` é:

```
spec:
  containers:
    - name: nginx
      image: nginx
      resources:
        limits:
          memory: "200Mi"
          cpu: "2"
        requests:
          memory: "200Mi"
          cpu: "2"
```

41.10 Programação com reconhecimento de NUMA

Acesso não uniforme à memória ou arquitetura não uniforme de acesso à memória (`NUMA`, Non-Uniform Memory Access ou Non-Uniform Memory Architecture) é um projeto de memória física usado na arquitetura `SMP` (multiprocessadores), em que o tempo de acesso à memória depende do local da memória relativo ao processador. No `NUMA`, um processador pode acessar a própria memória local com mais rapidez do que a memória não local, ou seja, a memória local de outro processador ou a memória compartilhada entre processadores.

41.10.1 Identificando os nós NUMA

Para identificar os nós NUMA, execute o seguinte comando em seu sistema:

```
$ lscpu | grep NUMA
NUMA node(s):                       1
NUMA node0 CPU(s):                  0-63
```

Nota

Para este exemplo, temos apenas um nó NUMA com 64 CPUs.

É necessário habilitar o NUMA no BIOS. Se o dmesg não tem registros de inicialização do NUMA durante o bootup, as mensagens relacionadas ao NUMA no buffer de anel do kernel podem ter sido substituídas.

41.11 MetalLB

MetalLB é uma implementação de balanceador de carga para clusters Kubernetes bare metal, que usa os protocolos de roteamento padrão, como L2 e BGP, como protocolos de anúncio. Trata-se de um balanceador de carga de rede que pode ser usado para expor serviços em um cluster Kubernetes ao ambiente externo por causa da necessidade de usar serviços do Kubernetes do tipo LoadBalancer com bare metal.

Para habilitar o MetalLB no cluster RKE2, são necessárias as seguintes etapas:

- Instale o MetalLB usando o seguinte comando:

```
$ kubectl apply <<EOF -f
apiVersion: helm.cattle.io/v1
kind: HelmChart
metadata:
  name: metallb
  namespace: kube-system
spec:
  chart: oci://registry.suse.com/edge/charts/metallb
  targetNamespace: metallb-system
  version: 303.0.0+up0.14.9
  createNamespace: true
---
apiVersion: helm.cattle.io/v1
kind: HelmChart
```

```
metadata:
  name: endpoint-copier-operator
  namespace: kube-system
spec:
  chart: oci://registry.suse.com/edge/charts/endpoint-copier-operator
  targetNamespace: endpoint-copier-operator
  version: 303.0.0+up0.2.1
  createNamespace: true
EOF
```

- Crie a configuração de `IpAddressPool` e `L2advertisement`:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: kubernetes-vip-ip-pool
  namespace: metallb-system
spec:
  addresses:
    - 10.168.200.98/32
  serviceAllocation:
    priority: 100
    namespaces:
      - default
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: ip-pool-l2-adv
  namespace: metallb-system
spec:
  ipAddressPools:
    - kubernetes-vip-ip-pool
```

- Crie o serviço de endpoint para expor o `VIP`:

```
apiVersion: v1
kind: Service
metadata:
  name: kubernetes-vip
  namespace: default
spec:
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
```

```
  ports:
  - name: rke2-api
    port: 9345
    protocol: TCP
    targetPort: 9345
  - name: k8s-api
    port: 6443
    protocol: TCP
    targetPort: 6443
  sessionAffinity: None
  type: LoadBalancer
```

- Verifique se o `VIP` foi criado e se os pods do `MetalLB` estão em execução:

```
$ kubectl get svc -n default
$ kubectl get pods -n default
```

41.12 Configuração do registro particular

É possível configurar o `Containerd` para conexão com os registros particulares e usá-los para extrair as imagens particulares de cada nó.

Na inicialização, o `RKE2` verifica se existe um arquivo `registries.yaml` em `/etc/rancher/rke2/` e instrui o `containerd` a usar os registros definidos no arquivo. Para usar um registro particular, crie esse arquivo como raiz em cada nó que usará o registro.

Para adicionar o registro particular, crie o arquivo `/etc/rancher/rke2/registries.yaml` com o seguinte conteúdo:

```
mirrors:
  docker.io:
    endpoint:
      - "https://registry.example.com:5000"
configs:
  "registry.example.com:5000":
    auth:
      username: xxxxxx # this is the registry username
      password: xxxxxx # this is the registry password
    tls:
      cert_file:            # path to the cert file used to authenticate to the registry
      key_file:             # path to the key file for the certificate used to
 authenticate to the registry
      ca_file:              # path to the ca file used to verify the registry's
 certificate
```

```
      insecure_skip_verify: # may be set to true to skip verifying the registry's
 certificate
```

ou sem autenticação:

```
mirrors:
  docker.io:
    endpoint:
      - "https://registry.example.com:5000"
configs:
  "registry.example.com:5000":
    tls:
      cert_file:            # path to the cert file used to authenticate to the registry
      key_file:             # path to the key file for the certificate used to
 authenticate to the registry
      ca_file:              # path to the ca file used to verify the registry's
 certificate
      insecure_skip_verify: # may be set to true to skip verifying the registry's
 certificate
```

Para que as alterações no registro entrem em vigor, você precisa configurar esse arquivo antes de iniciar o RKE2 no nó ou reiniciar o RKE2 em cada nó configurado.

Nota

Para obter mais informações sobre isso, acesse a configuração de registro containerd para RKE2 (https://documentation.suse.com/cloudnative/rke2/latest/en/install/containerd_registry_configuration.html#_registries_configuration_file).

41.13 Precision Time Protocol

Precision Time Protocol (PTP) é um protocolo de rede desenvolvido pelo Institute of Electrical and Electronics Engineers (IEEE) para permitir a sincronização de tempo em submicrossegundos em uma rede de computadores. Desde a sua origem e durante as duas últimas décadas, o PTP tem sido usado em diversos setores. Recentemente, observamos uma crescente adoção nas redes de telecomunicações como elemento essencial a redes 5G. Apesar de ser um protocolo relativamente simples, sua configuração pode mudar bastante de acordo com o aplicativo. Por essa razão, foram definidos e padronizados vários perfis.

Nesta seção, apenas os perfis específicos de telecomunicações serão apresentados. Portanto, vamos considerar o recurso de marcação de data e hora e um relógio de hardware PTP (PHC, PTP Hardware Clock) na NIC. Hoje em dia, todos os adaptadores de rede para telecomunicações contam com suporte a PTP no hardware, mas você pode verificar esses recursos com o seguinte comando:

```
# ethtool -T p1p1
Time stamping parameters for p1p1:
Capabilities:
        hardware-transmit
        software-transmit
        hardware-receive
        software-receive
        software-system-clock
        hardware-raw-clock
PTP Hardware Clock: 0
Hardware Transmit Timestamp Modes:
        off
        on
Hardware Receive Filter Modes:
        none
        all
```

Substitua `p1p1` pelo nome da interface usada para PTP.

As seções a seguir orientam como instalar e configurar o PTP especificamente no SUSE Edge, mas deve haver uma familiaridade com os conceitos básicos do PTP. Para uma breve visão geral do PTP e a implementação incluída no SUSE Edge for Telco, acesse https://documentation.suse.com/sles/html/SLES-all/cha-tuning-ptp.html.

41.13.1 Instalar os componentes de software PTP

No SUSE Edge for Telco, a implementação do PTP é fornecida pelo pacote `linuxptp`, que inclui dois componentes:

- `ptp4l`: um daemon que controla o PHC na NIC e executa o protocolo PTP
- `phc2sys`: um daemon que mantém a sincronização do relógio do sistema com o PHC sincronizado por PTP na NIC

Os dois daemons são necessários para que a sincronização do sistema funcione por completo e devem ser definidos de maneira correta com a sua configuração, o que foi abordado na *Seção 41.13.2, "Configurar o PTP para implantações de telecomunicações"*.

A maneira melhor e mais fácil de integrar o PTP ao cluster downstream é adicionar o pacote `linuxptp` em `packageList` ao arquivo de definição do Edge Image Builder (EIB). Desse modo, o software de plano de controle PTP será instalado automaticamente durante o provisionamento do cluster. Consulte a documentação do EIB (*Seção 3.3.4, "Configurando pacotes RPM"*) para obter mais informações sobre como instalar os pacotes.

Veja a seguir um manifesto do EIB de amostra com `linuxptp`:

```
apiVersion: 1.0
image:
  imageType: RAW
  arch: x86_64
  baseImage: {micro-base-rt-image-raw}
  outputImageName: eibimage-slmicrort-telco.raw
operatingSystem:
  time:
    timezone: America/New_York
  kernelArgs:
    - ignition.platform.id=openstack
    - net.ifnames=1
  systemd:
    disable:
      - rebootmgr
      - transactional-update.timer
      - transactional-update-cleanup.timer
      - fstrim
      - time-sync.target
    enable:
      - ptp4l
      - phc2sys
  users:
    - username: root
      encryptedPassword: ${ROOT_PASSWORD}
  packages:
    packageList:
      - jq
      - dpdk
      - dpdk-tools
      - libdpdk-23
      - pf-bb-config
      - open-iscsi
      - tuned
      - cpupower
      - linuxptp
    sccRegistrationCode: ${SCC_REGISTRATION_CODE}
```

Nota

O pacote `linuxptp` incluído no SUSE Edge for Telco não habilita o `ptp4l` e o `phc2sys` por padrão. Se os arquivos de configuração específicos do sistema forem implantados no momento do provisionamento (consulte a *Seção 41.13.3, "Integração da Cluster API"*), eles deverão ser habilitados. Para isso, adicione-os à seção `systemd` do manifesto, conforme o exemplo acima.

Siga o processo normal para criar a imagem conforme descrito na documentação do EIB (*Seção 3.4, "Criando a imagem"*) e use-a para implantar o cluster. Se você não tem experiência com o EIB, comece pelo *Capítulo 11, Edge Image Builder*.

41.13.2 Configurar o PTP para implantações de telecomunicações

Muitos aplicativos de telecomunicações exigem uma sincronização rígida de fase e de tempo com pouca variação, o que resultou na definição de dois perfis orientados a telecomunicações: ITU-T G.8275.1 e ITU-T G.8275.2. Os dois têm alta taxa de mensagens de sincronização e outros aspectos diferenciados, como uso de um algoritmo BMCA (Best Master Clock Algorithm) alternativo. Esse comportamento exige definições específicas no arquivo de configuração consumido pelo `ptp4l`, apresentadas nas seções a seguir como referência.

Nota

- As duas seções abordam apenas o caso de um relógio comum na configuração de receptor de tempo.
- Esse tipo de perfil deve ser usado em uma infraestrutura PTP bem planejada.
- Sua rede PTP específica pode exigir um ajuste de configuração adicional. Revise os exemplos apresentados e adapte-os se necessário.

41.13.2.1 Perfil de PTP ITU-T G.8275.1

O perfil G.8275.1 tem as seguintes especificações:

- Executado diretamente em Ethernet e requer suporte completo à rede (nós/comutadores adjacentes devem dar suporte a PTP).
- A configuração de domínio padrão é 24.
- A comparação de conjunto de dados baseia-se no algoritmo G.8275.x e nos valores `localPriority` depois de `priority2`.

Copie o seguinte conteúdo para um arquivo chamado `/etc/ptp4l-G.8275.1.conf`:

```
# Telecom G.8275.1 example configuration
[global]
domainNumber                    24
priority2   255
dataset_comparison              G.8275.x
G.8275.portDS.localPriority     128
G.8275.defaultDS.localPriority  128
maxStepsRemoved                 255
logAnnounceInterval             -3
logSyncInterval                 -4
logMinDelayReqInterval          -4
announceReceiptTimeout  3
serverOnly                      0
ptp_dst_mac                     01:80:C2:00:00:0E
network_transport               L2
```

Após a criação do arquivo, faça referência a ele em `/etc/sysconfig/ptp4l` para o daemon ser iniciado corretamente. Para fazer isso, altere a linha `OPTIONS=`:

```
OPTIONS="-f /etc/ptp4l-G.8275.1.conf -i $IFNAME --message_tag ptp-8275.1"
```

Mais precisamente:

- `-f` requer o nome do arquivo de configuração que será usado; neste caso, `/etc/ptp4l-G.8275.1.conf`.
- `-i` requer o nome da interface que será usada. Substitua `$IFNAME` pelo nome da interface real.
- `--message_tag` permite identificar melhor a saída de ptp4l nos registros do sistema e é opcional.

Após a conclusão das etapas acima, o daemon `ptp4l` deverá ser (re)iniciado:

```
# systemctl restart ptp4l
```

Verifique o status da sincronização observando os registros com:

```
# journalctl -e -u ptp4l
```

41.13.2.2 Perfil de PTP ITU-T G.8275.2

O perfil G.8275.2 tem as seguintes especificações:

- Executado em IP e não requer suporte total à rede (nós/comutadores adjacentes podem não dar suporte a PTP).
- A configuração de domínio padrão é 44.
- A comparação de conjunto de dados baseia-se no algoritmo G.8275.x e nos valores `localPriority` depois de `priority2`.

Copie o seguinte conteúdo para o arquivo chamado `/etc/ptp4l-G.8275.2.conf`:

```
# Telecom G.8275.2 example configuration
[global]
domainNumber                    44
priority2   255
dataset_comparison              G.8275.x
G.8275.portDS.localPriority     128
G.8275.defaultDS.localPriority  128
maxStepsRemoved                 255
logAnnounceInterval             0
serverOnly                      0
hybrid_e2e                      1
inhibit_multicast_service       1
unicast_listen                  1
unicast_req_duration            60
logSyncInterval                 -5
logMinDelayReqInterval          -4
announceReceiptTimeout  2
#
# Customize the following for slave operation:
#
[unicast_master_table]
table_id                        1
logQueryInterval                2
```

```
UDPv4                           $PEER_IP_ADDRESS
[$IFNAME]
unicast_master_table            1
```

Substitua os seguintes espaços reservados:

- `$PEER_IP_ADDRESS`: o endereço IP do nó PTP seguinte com o qual se comunicar, como o relógio mestre ou de limite que fornecerá a sincronização.
- `$IFNAME`: instrui o `ptp4l` sobre qual interface usar para PTP.

Após a criação do arquivo, faça referência a ele, junto com o nome da interface para PTP, em `/etc/sysconfig/ptp4l` para que o daemon seja iniciado corretamente. Para fazer isso, altere a linha `OPTIONS=` para:

```
OPTIONS="-f /etc/ptp4l-G.8275.2.conf --message_tag ptp-8275.2"
```

Mais precisamente:

- `-f` requer o nome do arquivo de configuração que será usado; neste caso, `/etc/ptp4l-G.8275.2.conf`.
- `--message_tag` permite identificar melhor a saída de ptp4l nos registros do sistema e é opcional.

Após a conclusão das etapas acima, o daemon `ptp4l` deverá ser (re)iniciado:

```
# systemctl restart ptp4l
```

Verifique o status da sincronização observando os registros com:

```
# journalctl -e -u ptp4l
```

41.13.2.3 Configuração do phc2sys

Embora não seja obrigatório, é recomendado completar toda a configuração de `ptp4l` antes de passar para o `phc2sys`. O `phc2sys` não requer um arquivo de configuração, e seus parâmetros de execução podem ser controlados unicamente pela variável `OPTIONS=` presente em `/etc/sysconfig/ptp4l`, de maneira similar a `ptp4l`:

```
OPTIONS="-s $IFNAME -w"
```

Em que `$IFNAME` é o nome da interface já configurada em ptp4l, que será usada como fonte para o relógio do sistema. Isso é usado para identificar o PHC de origem.

41.13.3 Integração da Cluster API

Sempre que um cluster é implantado por cluster de gerenciamento e provisionamento de rede direcionado, tanto o arquivo de configuração quanto as duas variáveis de configuração em `/etc/sysconfig` podem ser implantados no host no momento do provisionamento. Veja abaixo o trecho de uma definição de cluster, com foco no objeto `RKE2ControlPlane` modificado que implanta o mesmo arquivo de configuração G.8275.1 em todos os hosts:

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
metadata:
  name: single-node-cluster
  namespace: default
spec:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3MachineTemplate
    name: single-node-cluster-controlplane
  replicas: 1
  version: ${RKE2_VERSION}
  rolloutStrategy:
    type: "RollingUpdate"
    rollingUpdate:
      maxSurge: 0
  registrationMethod: "control-plane-endpoint"
  serverConfig:
    cni: canal
  agentConfig:
    format: ignition
    cisProfile: cis
    additionalUserData:
      config: |
        variant: fcos
        version: 1.4.0
        systemd:
          units:
            - name: rke2-preinstall.service
              enabled: true
              contents: |
                [Unit]
                Description=rke2-preinstall
                Wants=network-online.target
                Before=rke2-install.service
                ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
                [Service]
                Type=oneshot
```

```
                User=root
                ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
                ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/$(jq -r .uuid /mnt/
openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
                ExecStart=/bin/sh -c "echo \"node-name: $(jq -r .name /mnt/openstack/
latest/meta_data.json)\" >> /etc/rancher/rke2/config.yaml"
                ExecStartPost=/bin/sh -c "umount /mnt"
                [Install]
                WantedBy=multi-user.target
        storage:
          files:
            - path: /etc/ptp4l-G.8275.1.conf
              overwrite: true
              contents:
                inline: |
                  # Telecom G.8275.1 example configuration
                  [global]
                  domainNumber                    24
                  priority2                       255
                  dataset_comparison              G.8275.x
                  G.8275.portDS.localPriority     128
                  G.8275.defaultDS.localPriority  128
                  maxStepsRemoved                 255
                  logAnnounceInterval             -3
                  logSyncInterval                 -4
                  logMinDelayReqInterval          -4
                  announceReceiptTimeout          3
                  serverOnly                      0
                  ptp_dst_mac                     01:80:C2:00:00:0E
                  network_transport               L2
              mode: 0644
              user:
                name: root
              group:
                name: root
            - path: /etc/sysconfig/ptp4l
              overwrite: true
              contents:
                inline: |
                  ## Path:           Network/LinuxPTP
                  ## Description:    Precision Time Protocol (PTP): ptp4l settings
                  ## Type:           string
                  ## Default:        "-i eth0 -f /etc/ptp4l.conf"
                  ## ServiceRestart: ptp4l
                  #
                  # Arguments when starting ptp4l(8).
                  #
```

```
                OPTIONS="-f /etc/ptp4l-G.8275.1.conf -i $IFNAME --message_tag
ptp-8275.1"
            mode: 0644
            user:
              name: root
            group:
              name: root
          - path: /etc/sysconfig/phc2sys
            overwrite: true
            contents:
              inline: |
                ## Path:           Network/LinuxPTP
                ## Description:    Precision Time Protocol (PTP): phc2sys settings
                ## Type:           string
                ## Default:        "-s eth0 -w"
                ## ServiceRestart: phc2sys
                #
                # Arguments when starting phc2sys(8).
                #
                OPTIONS="-s $IFNAME -w"
            mode: 0644
            user:
              name: root
            group:
              name: root
  kubelet:
    extraArgs:
      - provider-id=metal3://BAREMETALHOST_UUID
  nodeName: "localhost.localdomain"
```

Além de outras variáveis, é necessário preencher a definição acima com o nome da interface e os outros objetos da Cluster API, conforme descrito no *Capítulo 42, Provisionamento de rede direcionado totalmente automatizado*.

Nota

- Essa é uma abordagem prática apenas quando o hardware no cluster é uniforme e a mesma configuração é necessária em todos os hosts, inclusive o nome da interface.
- Há outras abordagens possíveis que serão explicadas em versões futuras.

Neste ponto, os hosts devem ter uma pilha PTP em funcionamento e começarão a negociar sua função de PTP.

42 Provisionamento de rede direcionado totalmente automatizado

42.1 Introdução

O provisionamento de rede direcionado é um recurso que permite automatizar o provisionamento de clusters downstream. Ele é útil quando você tem que provisionar muitos clusters downstream e deseja automatizar o processo.

Um cluster de gerenciamento (*Capítulo 40, Configurando o cluster de gerenciamento*) automatiza a implantação dos seguintes componentes:

- `SUSE Linux Micro RT` como sistema operacional. Dependendo do caso de uso, é possível personalizar configurações como rede, armazenamento, usuários e argumentos do kernel.
- `RKE2` como cluster Kubernetes. O plug-in de `CNI` padrão é `Cilium`. Dependendo do caso de uso, é possível usar determinados plug-ins de `CNI`, como `Cilium+Multus`.
- `SUSE Storage`
- `SUSE Security`
- É possível usar o `MetalLB` como balanceador de carga para clusters de vários nós altamente disponíveis.

Nota

Para obter mais informações sobre o `SUSE Linux Micro`, consulte o *Capítulo 9, SUSE Linux Micro*. Para obter mais informações sobre o `RKE2`, consulte o *Capítulo 16, RKE2*. Para obter mais informações sobre o `SUSE Storage`, consulte o *Capítulo 17, SUSE Storage*. Para obter mais informações sobre o `SUSE Security`, consulte o *Capítulo 18, SUSE Security*.

As seguintes seções descrevem os diferentes fluxos de trabalho de provisionamento de rede direcionado e alguns recursos adicionais que podem ser incluídos no processo de provisionamento:

- *Seção 42.2, “Preparar uma imagem de cluster downstream para cenários conectados”*
- *Seção 42.3, “Preparar uma imagem de cluster downstream para cenários air-gapped”*

- *Seção 42.4, “Provisionamento de cluster downstream com provisionamento de rede direcionado (nó único)”*
- *Seção 42.5, “Provisionamento de cluster downstream com provisionamento de rede direcionado (vários nós)”*
- *Seção 42.6, “Configuração avançada de rede”*
- *Seção 42.7, “Recursos de telecomunicações (DPDK, SR-IOV, isolamento de CPU, HugePages, NUMA etc.)”*
- *Seção 42.8, “Registro particular”*
- *Seção 42.9, “Provisionamento de cluster downstream em cenários air-gapped”*

Nota

As seções a seguir mostram como preparar os diversos cenários para o fluxo de trabalho de provisionamento de rede direcionado usando o SUSE Edge for Telco. Para ver exemplos de opções de configurações diferentes para implantação (incluindo ambientes air-gapped, redes DHCP e sem DHCP, registros de contêiner particulares etc.), consulte o repositório SUSE do SUSE Edge for Telco (https://github.com/suse-edge/atip/tree/release-3.3/telco-examples/edge-clusters).

42.2 Preparar uma imagem de cluster downstream para cenários conectados

O Edge Image Builder (*Capítulo 11, Edge Image Builder*) é usado para preparar uma imagem base do SLEMicro modificada que será provisionada aos hosts de cluster downstream.

Grande parte da configuração é possível pelo Edge Image Builder, mas neste guia, abordamos as configurações mínimas necessárias para preparar o cluster downstream.

42.2.1 Pré-requisitos para cenários conectados

- É necessário um tempo de execução do contêiner, como Podman (https://podman.io) ou Rancher Desktop (https://rancherdesktop.io), para executar o Edge Image Builder.
- A imagem base será criada conforme este guia *Capítulo 28, Criando imagens atualizadas do SUSE Linux Micro com o Kiwi* com o perfil `Base-SelfInstall` (ou `Base-RT-SelfInstall` para o kernel Real-Time). O processo é o mesmo para as duas arquiteturas (x86-64 e aarch64).
- Para implantar clusters downstream aarch64, você deve definir o parâmetro `deployArchitecture: arm64` no arquivo `metal3.yaml` antes da implantação do cluster de gerenciamento, o que está explicado na documentação sobre o cluster de gerenciamento (*??? (p 421)*). Isso é necessário para garantir o uso da arquitetura correta para o cluster downstream.

Nota

É necessário usar um host de build com a mesma arquitetura das imagens que estão sendo criadas. Em outras palavras, para criar uma imagem `aarch64`, é necessário usar um host de build `aarch64`, e vice-versa para `x86-64` (não há suporte para builds cruzados no momento).

42.2.2 Configuração da imagem para cenários conectados

Ao executar o Edge Image Builder, um diretório é montado com base no host, portanto, é necessário criar uma estrutura de diretórios para armazenar os arquivos de configuração usados para definir a imagem de destino.

- `downstream-cluster-config.yaml` é o arquivo de definição da imagem. Consulte o *Capítulo 3, Clusters independentes com o Edge Image Builder* para obter mais detalhes.
- A pasta da imagem base inclui a imagem bruta de saída gerada conforme o guia *Capítulo 28, Criando imagens atualizadas do SUSE Linux Micro com o Kiwi* com o perfil `Base-SelfInstall` (ou `Base-RT-SelfInstall` para o kernel Real-Time) e deve ser copiada/movida para a pasta `base-images`.
- A pasta `network` é opcional. Consulte a *Seção 42.2.2.6, "Script adicional para configuração de rede avançada"* para obter mais detalhes.

- O diretório `custom/scripts` contém scripts para execução na primeira inicialização:
    1. O script `01-fix-growfs.sh` é necessário para redimensionar a partição raiz do sistema operacional na implantação.
    2. O script `02-performance.sh` é opcional e pode ser usado para configurar o sistema para ajuste de desempenho.
    3. O script `03-sriov.sh` é opcional e pode ser usado para configurar o sistema para SR-IOV.
- O diretório `custom/files` contém os arquivos `performance-settings.sh` e `sriov-auto-filler.sh` para copiar na imagem durante o processo de criação da imagem.

```
├── downstream-cluster-config.yaml
├── base-images/
|   └ SL-Micro.x86_64-6.1-Base-GM.raw
├── network/
|   └ configure-network.sh
└── custom/
    └ scripts/
    |   └ 01-fix-growfs.sh
    |   └ 02-performance.sh
    |   └ 03-sriov.sh
    └ files/
        └ performance-settings.sh
        └ sriov-auto-filler.sh
```

42.2.2.1 Arquivo de definição da imagem do cluster downstream

O `downstream-cluster-config.yaml` é o arquivo de configuração principal para a imagem do cluster downstream. Veja a seguir um exemplo mínimo de implantação por meio do Metal[3]:

```
apiVersion: 1.2
image:
  imageType: raw
  arch: x86_64
  baseImage: SL-Micro.x86_64-6.1-Base-GM.raw
  outputImageName: eibimage-output-telco.raw
operatingSystem:
  kernelArgs:
    - ignition.platform.id=openstack
    - net.ifnames=1
  systemd:
```

```
    disable:
      - rebootmgr
      - transactional-update.timer
      - transactional-update-cleanup.timer
      - fstrim
      - time-sync.target
  users:
    - username: root
      encryptedPassword: $ROOT_PASSWORD
      sshKeys:
      - $USERKEY1
  packages:
    packageList:
      - jq
    sccRegistrationCode: $SCC_REGISTRATION_CODE
```

Em que `$SCC_REGISTRATION_CODE` é o código de registro copiado do SUSE Customer Center (https://scc.suse.com/), e a lista de pacotes inclui o `jq`, que é obrigatório.

`$ROOT_PASSWORD` é a senha criptografada do usuário root, que pode ser útil para teste/depuração. É possível gerá-la com o comando `openssl passwd -6 PASSWORD`.

Para os ambientes de produção, a recomendação é usar as chaves SSH que podem ser adicionadas ao bloco de usuários substituindo a `$USERKEY1` pelas chaves SSH reais.

Nota

`arch: x86_64` é a arquitetura da imagem. Para arm64, use `arch: aarch64`.

O `net.ifnames=1` habilita a Nomenclatura de interface de rede previsível (https://documentation.suse.com/smart/network/html/network-interface-predictable-naming/index.html).

Isso corresponde à configuração padrão para o gráfico metal3, mas a configuração deve corresponder ao valor `predictableNicNames` do gráfico configurado.

Veja também que o `ignition.platform.id=openstack` é obrigatório. Sem esse argumento, a configuração do SLEMicro pelo ignition vai falhar no fluxo automatizado do Metal[3].

42.2.2.2 Script growfs

Atualmente, é necessário um script personalizado (`custom/scripts/01-fix-growfs.sh`) para expandir o sistema de arquivos de acordo com o tamanho do disco na primeira inicialização após o provisionamento. O script `01-fix-growfs.sh` contém as seguintes informações:

```
#!/bin/bash
growfs() {
  mnt="$1"
  dev="$(findmnt --fstab --target ${mnt} --evaluate --real --output SOURCE --noheadings)"
  # /dev/sda3 -> /dev/sda, /dev/nvme0n1p3 -> /dev/nvme0n1
  parent_dev="/dev/$(lsblk --nodeps -rno PKNAME "${dev}")"
  # Last number in the device name: /dev/nvme0n1p42 -> 42
  partnum="$(echo "${dev}" | sed 's/^.*[^0-9]\([0-9]\+\)$/\1/')"
  ret=0
  growpart "$parent_dev" "$partnum" || ret=$?
  [ $ret -eq 0 ] || [ $ret -eq 1 ] || exit 1
  /usr/lib/systemd/systemd-growfs "$mnt"
}
growfs /
```

42.2.2.3 Script de desempenho

O seguinte script opcional (`custom/scripts/02-performance.sh`) pode ser usado para configurar o sistema para ajuste de desempenho:

```
#!/bin/bash

# create the folder to extract the artifacts there
mkdir -p /opt/performance-settings

# copy the artifacts
cp performance-settings.sh /opt/performance-settings/
```

O conteúdo do `custom/files/performance-settings.sh` é um script que pode ser usado para configurar o sistema para ajuste de desempenho e pode ser baixado acessando este link (https://github.com/suse-edge/atip/blob/release-3.3/telco-examples/edge-clusters/dhcp/eib/custom/files/performance-settings.sh).

42.2.2.4 Script SR-IOV

O seguinte script opcional (`custom/scripts/03-sriov.sh`) pode ser usado para configurar o sistema para SR-IOV:

```
#!/bin/bash

# create the folder to extract the artifacts there
mkdir -p /opt/sriov
# copy the artifacts
cp sriov-auto-filler.sh /opt/sriov/sriov-auto-filler.sh
```

O conteúdo do `custom/files/sriov-auto-filler.sh` é um script que pode ser usado para configurar o sistema para SR-IOV e pode ser baixado acessando este link (https://github.com/suse-edge/atip/blob/release-3.3/telco-examples/edge-clusters/dhcp/eib/custom/files/sriov-auto-filler.sh).

Nota

Adicione seus próprios scripts personalizados para execução durante o processo de provisionamento usando a mesma abordagem. Para obter mais informações, consulte o *Capítulo 3, Clusters independentes com o Edge Image Builder*.

42.2.2.5 Configuração adicional para cargas de trabalho de telecomunicações

Para habilitar recursos de telecomunicações, como `dpdk`, `sr-iov` ou `FEC`, podem ser necessários pacotes adicionais, conforme mostrado no exemplo a seguir.

```
apiVersion: 1.2
image:
  imageType: raw
  arch: x86_64
  baseImage: SL-Micro.x86_64-6.1-Base-GM.raw
  outputImageName: eibimage-output-telco.raw
operatingSystem:
  kernelArgs:
    - ignition.platform.id=openstack
    - net.ifnames=1
  systemd:
    disable:
```

```
      - rebootmgr
      - transactional-update.timer
      - transactional-update-cleanup.timer
      - fstrim
      - time-sync.target
  users:
    - username: root
      encryptedPassword: $ROOT_PASSWORD
      sshKeys:
      - $user1Key1
  packages:
    packageList:
      - jq
      - dpdk
      - dpdk-tools
      - libdpdk-23
      - pf-bb-config
    sccRegistrationCode: $SCC_REGISTRATION_CODE
```

Em que `$SCC_REGISTRATION_CODE` é o código de registro copiado do SUSE Customer Center (https://scc.suse.com/), e a lista de pacotes inclui os pacotes mínimos usados nos perfis de telecomunicações.

Nota

`arch: x86_64` é a arquitetura da imagem. Para arm64, use `arch: aarch64`.

42.2.2.6 Script adicional para configuração de rede avançada

Se você precisa configurar IPs estáticos ou cenários de rede mais avançados, conforme descrito na *Seção 42.6, "Configuração avançada de rede"*, a configuração adicional abaixo é necessária.

Na pasta `network`, crie o seguinte arquivo `configure-network.sh`, que consome os dados da unidade de configuração na primeira inicialização e configura a rede do host usando a ferramenta NM Configurator (https://github.com/suse-edge/nm-configurator).

```
#!/bin/bash

set -eux

# Attempt to statically configure a NIC in the case where we find a network_data.json
# In a configuration drive
```

```
CONFIG_DRIVE=$(blkid --label config-2 || true)
if [ -z "${CONFIG_DRIVE}" ]; then
  echo "No config-2 device found, skipping network configuration"
  exit 0
fi

mount -o ro $CONFIG_DRIVE /mnt

NETWORK_DATA_FILE="/mnt/openstack/latest/network_data.json"

if [ ! -f "${NETWORK_DATA_FILE}" ]; then
  umount /mnt
  echo "No network_data.json found, skipping network configuration"
  exit 0
fi

DESIRED_HOSTNAME=$(cat /mnt/openstack/latest/meta_data.json | tr ',{}' '\n' | grep
 '\"metal3-name\"' | sed 's/.*\"metal3-name\": \"\(.*\)\"/\1/')
echo "${DESIRED_HOSTNAME}" > /etc/hostname

mkdir -p /tmp/nmc/{desired,generated}
cp ${NETWORK_DATA_FILE} /tmp/nmc/desired/_all.yaml
umount /mnt

./nmc generate --config-dir /tmp/nmc/desired --output-dir /tmp/nmc/generated
./nmc apply --config-dir /tmp/nmc/generated
```

42.2.3 Criação de imagem

Depois que a estrutura de diretórios for preparada de acordo com as seções anteriores, execute o seguinte comando para criar a imagem:

```
podman run --rm --privileged -it -v $PWD:/eib \
 registry.suse.com/edge/3.3/edge-image-builder:1.2.1 \
 build --definition-file downstream-cluster-config.yaml
```

Esse procedimento cria o arquivo da imagem ISO de saída chamado `eibimage-output-telco.raw`, com base na definição descrita acima.

Em seguida, disponibilize a imagem de saída por um servidor web, ou o contêiner de servidor de mídia habilitado de acordo com as instruções na documentação sobre o cluster de gerenciamento (*Nota*) ou algum outro servidor acessível localmente. Nos exemplos abaixo, esse servidor é mencionado como `imagecache.local:8080`.

42.3 Preparar uma imagem de cluster downstream para cenários air-gapped

O Edge Image Builder (*Capítulo 11, Edge Image Builder*) é usado para preparar uma imagem base do SLEMicro modificada que será provisionada aos hosts de cluster downstream.

Grande parte da configuração é possível com o Edge Image Builder, mas neste guia, abordamos as configurações mínimas necessárias para preparar o cluster downstream em cenários air-gapped.

42.3.1 Pré-requisitos para cenários air-gapped

- É necessário um tempo de execução do contêiner, como Podman (https://podman.io) ou Rancher Desktop (https://rancherdesktop.io), para executar o Edge Image Builder.
- A imagem base será criada conforme este guia *Capítulo 28, Criando imagens atualizadas do SUSE Linux Micro com o Kiwi* com o perfil `Base-SelfInstall` (ou `Base-RT-SelfInstall` para o kernel Real-Time). O processo é o mesmo para as duas arquiteturas (x86-64 e aarch64).
- Para implantar clusters downstream aarch64, você deve definir o parâmetro `deployArchitecture: arm64` no arquivo `metal3.yaml` antes da implantação do cluster de gerenciamento, o que está explicado na documentação sobre o cluster de gerenciamento (*??? (p 421)*). Isso é necessário para garantir o uso da arquitetura correta para o cluster downstream.
- Para usar SR-IOV ou qualquer outra carga de trabalho que exija uma imagem de contêiner, é necessário implantar e já configurar um registro particular local (com/sem TLS e/ou autenticação). Esse registro será usado para armazenar as imagens comuns e as imagens OCI de gráfico Helm.

Nota

É necessário usar um host de build com a mesma arquitetura das imagens que estão sendo criadas. Em outras palavras, para criar uma imagem `aarch64`, é necessário usar um host de build `aarch64`, e vice-versa para `x86-64` (não há suporte para builds cruzados no momento).

42.3.2 Configuração da imagem para cenários air-gapped

Ao executar o Edge Image Builder, um diretório é montado com base no host, portanto, é necessário criar uma estrutura de diretórios para armazenar os arquivos de configuração usados para definir a imagem de destino.

- `downstream-cluster-airgap-config.yaml` é o arquivo de definição da imagem, consulte o *Capítulo 3, Clusters independentes com o Edge Image Builder* para obter mais detalhes.
- A pasta da imagem base inclui a imagem bruta de saída gerada conforme o guia *Capítulo 28, Criando imagens atualizadas do SUSE Linux Micro com o Kiwi* com o perfil `Base-SelfInstall` (ou `Base-RT-SelfInstall` para o kernel Real-Time) e deve ser copiada/movida para a pasta `base-images`.
- A pasta `network` é opcional. Consulte a *Seção 42.2.2.6, "Script adicional para configuração de rede avançada"* para obter mais detalhes.
- O diretório `custom/scripts` contém scripts para execução na primeira inicialização:
    1. O script `01-fix-growfs.sh` é necessário para redimensionar a partição raiz do sistema operacional na implantação.
    2. O script `02-airgap.sh` é necessário para copiar as imagens no local certo durante o processo de criação da imagem para ambientes air-gapped.
    3. O script `03-performance.sh` é opcional e pode ser usado para configurar o sistema para ajuste de desempenho.
    4. O script `04-sriov.sh` é opcional e pode ser usado para configurar o sistema para SR-IOV.
- O diretório `custom/files` contém as imagens `rke2` e `cni` para copiar na imagem durante o processo de criação da imagem. É possível também incluir os arquivos opcionais `performance-settings.sh` e `sriov-auto-filler.sh`.

```
├── downstream-cluster-airgap-config.yaml
├── base-images/
|   └ SL-Micro.x86_64-6.1-Base-GM.raw
├── network/
|   └ configure-network.sh
└── custom/
```

```
    └ files/
    |   └ install.sh
    |   └ rke2-images-cilium.linux-amd64.tar.zst
    |   └ rke2-images-core.linux-amd64.tar.zst
    |   └ rke2-images-multus.linux-amd64.tar.zst
    |   └ rke2-images.linux-amd64.tar.zst
    |   └ rke2.linux-amd64.tar.zst
    |   └ sha256sum-amd64.txt
    |   └ performance-settings.sh
    |   └ sriov-auto-filler.sh
    └ scripts/
        └ 01-fix-growfs.sh
        └ 02-airgap.sh
        └ 03-performance.sh
        └ 04-sriov.sh
```

42.3.2.1 Arquivo de definição da imagem do cluster downstream

O `downstream-cluster-airgap-config.yaml` é o principal arquivo de configuração da imagem do cluster downstream, e o conteúdo foi descrito na seção anterior (*Seção 42.2.2.5, "Configuração adicional para cargas de trabalho de telecomunicações"*).

42.3.2.2 Script growfs

Atualmente, é necessário um script personalizado (`custom/scripts/01-fix-growfs.sh`) para expandir o sistema de arquivos de acordo com o tamanho do disco na primeira inicialização após o provisionamento. O script `01-fix-growfs.sh` contém as seguintes informações:

```
#!/bin/bash
growfs() {
  mnt="$1"
  dev="$(findmnt --fstab --target ${mnt} --evaluate --real --output SOURCE --noheadings)"
  # /dev/sda3 -> /dev/sda, /dev/nvme0n1p3 -> /dev/nvme0n1
  parent_dev="/dev/$(lsblk --nodeps -rno PKNAME "${dev}")"
  # Last number in the device name: /dev/nvme0n1p42 -> 42
  partnum="$(echo "${dev}" | sed 's/^.*[^0-9]\([0-9]\+\)$/\1/')"
  ret=0
  growpart "$parent_dev" "$partnum" || ret=$?
  [ $ret -eq 0 ] || [ $ret -eq 1 ] || exit 1
  /usr/lib/systemd/systemd-growfs "$mnt"
}
growfs /
```

42.3.2.3 Script air-gap

O seguinte script (`custom/scripts/02-airgap.sh`) é necessário para copiar as imagens no local certo durante o processo de criação da imagem:

```
#!/bin/bash

# create the folder to extract the artifacts there
mkdir -p /opt/rke2-artifacts
mkdir -p /var/lib/rancher/rke2/agent/images

# copy the artifacts
cp install.sh /opt/
cp rke2-images*.tar.zst rke2.linux-amd64.tar.gz sha256sum-amd64.txt /opt/rke2-artifacts/
```

42.3.2.4 Script de desempenho

O seguinte script opcional (`custom/scripts/03-performance.sh`) pode ser usado para configurar o sistema para ajuste de desempenho:

```
#!/bin/bash

# create the folder to extract the artifacts there
mkdir -p /opt/performance-settings

# copy the artifacts
cp performance-settings.sh /opt/performance-settings/
```

O conteúdo do `custom/files/performance-settings.sh` é um script que pode ser usado para configurar o sistema para ajuste de desempenho e pode ser baixado acessando este link (https://github.com/suse-edge/atip/blob/release-3.3/telco-examples/edge-clusters/dhcp/eib/custom/files/performance-settings.sh).

42.3.2.5 Script SR-IOV

O seguinte script opcional (`custom/scripts/04-sriov.sh`) pode ser usado para configurar o sistema para SR-IOV:

```
#!/bin/bash

# create the folder to extract the artifacts there
mkdir -p /opt/sriov
```

```
# copy the artifacts
cp sriov-auto-filler.sh /opt/sriov/sriov-auto-filler.sh
```

O conteúdo do `custom/files/sriov-auto-filler.sh` é um script que pode ser usado para configurar o sistema para SR-IOV e pode ser baixado acessando este link (https://github.com/suse-edge/atip/blob/release-3.3/telco-examples/edge-clusters/dhcp/eib/custom/files/sriov-auto-filler.sh).

42.3.2.6 Arquivos personalizados para cenários air-gapped

O diretório `custom/files` contém as imagens do `rke2` e `cni` que devem ser copiadas para a imagem durante o processo de criação dela. Para gerar as imagens com facilidade, prepare-as no local usando o seguinte script (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/scripts/day2/edge-save-images.sh) e a lista de imagens aqui (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/scripts/day2/edge-release-rke2-images.txt) para gerar os artefatos necessários para incluir em `custom/files`. Você também pode fazer download do script `rke2-install` mais recente aqui (https://get.rke2.io/).

```
$ ./edge-save-rke2-images.sh -o custom/files -l ~/edge-release-rke2-images.txt
```

Após o download das imagens, a estrutura de diretórios deve ter esta aparência:

```
└── custom/
    └ files/
        └ install.sh
        └ rke2-images-cilium.linux-amd64.tar.zst
        └ rke2-images-core.linux-amd64.tar.zst
        └ rke2-images-multus.linux-amd64.tar.zst
        └ rke2-images.linux-amd64.tar.zst
        └ rke2.linux-amd64.tar.zst
        └ sha256sum-amd64.txt
```

42.3.2.7 Pré-carregar seu registro particular com as imagens necessárias para cenários air-gapped e SR-IOV (opcional)

Para usar SR-IOV em seu cenário air-gapped ou qualquer outra imagem de carga de trabalho, você deve pré-carregar o registro particular local com as imagens seguindo as próximas etapas:

- Faça download, extraia e envie as imagens OCI de gráfico Helm ao registro particular
- Faça download, extraia e envie o restante das imagens necessárias ao registro particular

É possível usar os seguintes scripts para fazer download, extrair e enviar as imagens ao registro particular. Vamos mostrar um exemplo para pré-carregar as imagens SR-IOV, mas você também pode usar a mesma abordagem para pré-carregar qualquer outra imagem personalizada:

1. Pré-carregamento com imagens OCI de gráfico Helm para SR-IOV:

    a. Você deve criar uma lista com as imagens OCI de gráfico Helm necessárias:

    ```
    $ cat > edge-release-helm-oci-artifacts.txt <<EOF
    edge/sriov-network-operator-chart:303.0.2+up1.5.0
    edge/sriov-crd-chart:303.0.2+up1.5.0
    EOF
    ```

    b. Gere um arquivo tarball local usando o seguinte script (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/scripts/day2/edge-save-oci-artefacts.sh) e a lista criada acima:

    ```
    $ ./edge-save-oci-artefacts.sh -al ./edge-release-helm-oci-artifacts.txt -s
     registry.suse.com
    Pulled: registry.suse.com/edge/charts/sriov-network-operator:303.0.2+up1.5.0
    Pulled: registry.suse.com/edge/charts/sriov-crd:303.0.2+up1.5.0
    a edge-release-oci-tgz-20240705
    a edge-release-oci-tgz-20240705/sriov-network-operator-
    chart-303.0.2+up1.5.0.tgz
    a edge-release-oci-tgz-20240705/sriov-crd-chart-303.0.2+up1.5.0.tgz
    ```

    c. Faça upload do arquivo tarball para seu registro particular (por exemplo, `myregistry:5000`) usando o seguinte script (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/scripts/day2/edge-load-oci-artefacts.sh) para pré-carregar o registro com as imagens OCI do gráfico Helm baixadas na etapa anterior:

    ```
    $ tar zxvf edge-release-oci-tgz-20240705.tgz
    $ ./edge-load-oci-artefacts.sh -ad edge-release-oci-tgz-20240705 -r
     myregistry:5000
    ```

2. Pré-carregamento com o restante das imagens necessárias para SR-IOV:

    a. Nesse caso, devemos incluir as imagens do contêiner `sr-iov para cargas de trabalho de telecomunicações (por exemplo, como referência, você pode obtê-las dos valores de gráfico Helm (https://github.com/suse-edge/charts/blob/main/charts/sriov-network-operator/1.5.0/values.yaml))

    ```
    $ cat > edge-release-images.txt <<EOF
    ```

```
rancher/hardened-sriov-network-operator:v1.3.0-build20240816
rancher/hardened-sriov-network-config-daemon:v1.3.0-build20240816
rancher/hardened-sriov-cni:v2.8.1-build20240820
rancher/hardened-ib-sriov-cni:v1.1.1-build20240816
rancher/hardened-sriov-network-device-plugin:v3.7.0-build20240816
rancher/hardened-sriov-network-resources-injector:v1.6.0-build20240816
rancher/hardened-sriov-network-webhook:v1.3.0-build20240816
EOF
```

b. Usando o seguinte script (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/scripts/day2/edge-save-images.sh) e a lista criada acima, você deve gerar localmente o arquivo tarball com as imagens necessárias:

```
$ ./edge-save-images.sh -l ./edge-release-images.txt -s registry.suse.com
Image pull success: registry.suse.com/rancher/hardened-sriov-network-
operator:v1.3.0-build20240816
Image pull success: registry.suse.com/rancher/hardened-sriov-network-config-
daemon:v1.3.0-build20240816
Image pull success: registry.suse.com/rancher/hardened-sriov-cni:v2.8.1-
build20240820
Image pull success: registry.suse.com/rancher/hardened-ib-sriov-cni:v1.1.1-
build20240816
Image pull success: registry.suse.com/rancher/hardened-sriov-network-device-
plugin:v3.7.0-build20240816
Image pull success: registry.suse.com/rancher/hardened-sriov-network-resources-
injector:v1.6.0-build20240816
Image pull success: registry.suse.com/rancher/hardened-sriov-network-
webhook:v1.3.0-build20240816
Creating edge-images.tar.gz with 7 images
```

c. Faça upload do arquivo tarball para seu registro particular (por exemplo, `myregistry:5000`) usando o seguinte script (https://github.com/suse-edge/fleet-examples/blob/release-3.3.0/scripts/day2/edge-load-images.sh) para pré-carregar o registro particular com as imagens baixadas na etapa anterior:

```
$ tar zxvf edge-release-images-tgz-20240705.tgz
$ ./edge-load-images.sh -ad edge-release-images-tgz-20240705 -r myregistry:5000
```

42.3.3 Criação da imagem para cenários air-gapped

Depois que a estrutura de diretórios for preparada de acordo com as seções anteriores, execute o seguinte comando para criar a imagem:

```
podman run --rm --privileged -it -v $PWD:/eib \
 registry.suse.com/edge/3.3/edge-image-builder:1.2.1 \
 build --definition-file downstream-cluster-airgap-config.yaml
```

Esse procedimento cria o arquivo da imagem ISO de saída chamado `eibimage-output-telco.raw`, com base na definição descrita acima.

Em seguida, disponibilize a imagem de saída por um servidor web, ou o contêiner de servidor de mídia habilitado de acordo com as instruções na documentação sobre o cluster de gerenciamento (*Nota*) ou algum outro servidor acessível localmente. Nos exemplos abaixo, esse servidor é mencionado como `imagecache.local:8080`.

42.4 Provisionamento de cluster downstream com provisionamento de rede direcionado (nó único)

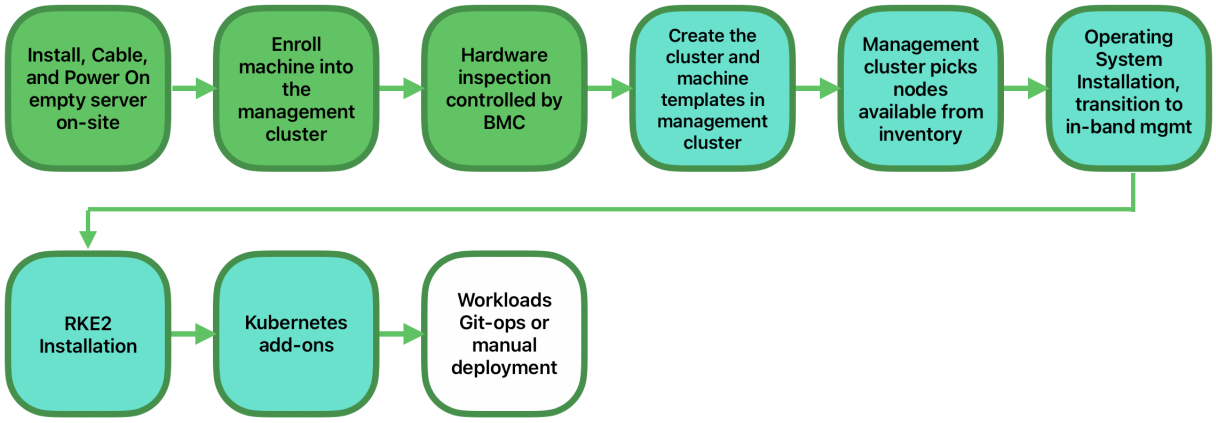
Esta seção descreve o fluxo de trabalho usado para automatizar o provisionamento de um cluster downstream de nó único usando o provisionamento de rede direcionado. Essa é a maneira mais fácil de automatizar o provisionamento de um cluster downstream.

Requisitos

- A imagem gerada por meio do `EIB`, conforme descrito na seção anterior (*Seção 42.2, "Preparar uma imagem de cluster downstream para cenários conectados"*), com a configuração mínima para definir o cluster downstream, deve estar localizada no cluster de gerenciamento exatamente no caminho configurado nesta seção (*Nota*).
- O servidor de gerenciamento criado e disponível para uso nas seções a seguir. Para obter mais informações, consulte a seção sobre o cluster de gerenciamento *Capítulo 40, Configurando o cluster de gerenciamento*.

Fluxo de trabalho

O diagrama a seguir mostra o fluxo de trabalho usado para automatizar o provisionamento de um cluster downstream de nó único por meio do provisionamento de rede direcionado:



Há duas etapas diferentes para automatizar o provisionamento de um cluster downstream de nó único por meio do provisionamento de rede direcionado:

1. Registre o host bare metal para disponibilizá-lo ao processo de provisionamento.
2. Provisione o host bare metal para instalar e configurar o sistema operacional e o cluster Kubernetes.

Registrar o host bare metal

A primeira etapa é registrar o novo host bare metal no cluster de gerenciamento para disponibilizá-lo para provisionamento. Para fazer isso, é necessário criar este arquivo (`bmh-example.yaml`) no cluster de gerenciamento, para especificar as credenciais que serão usadas do `BMC` e o objeto `BaremetalHost` para registro:

```
apiVersion: v1
kind: Secret
metadata:
  name: example-demo-credentials
type: Opaque
data:
  username: ${BMC_USERNAME}
  password: ${BMC_PASSWORD}
---
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
```

```
  name: example-demo
  labels:
    cluster-role: control-plane
spec:
  online: true
  bootMACAddress: ${BMC_MAC}
  rootDeviceHints:
    deviceName: /dev/nvme0n1
  bmc:
    address: ${BMC_ADDRESS}
    disableCertificateVerification: true
    credentialsName: example-demo-credentials
```

em que:

- `${BMC_USERNAME}`: o nome de usuário para o `BMC` do novo host bare metal.
- `${BMC_PASSWORD}`: a senha para o `BMC` do novo host bare metal.
- `${BMC_MAC}`: o endereço `MAC` do novo host bare metal que será usado.
- `${BMC_ADDRESS}`: o `URL` do host bare metal `BMC` (por exemplo, `redfish-virtualmedia://192.168.200.75/redfish/v1/Systems/1/`). Para saber mais sobre as diversas opções disponíveis de acordo com o seu provedor de hardware, acesse este link (https://github.com/metal3-io/baremetal-operator/blob/main/docs/api.md).

Nota

Se não foi especificada uma configuração de rede para o host, seja no momento da criação da imagem, seja pela definição `BareMetalHost`, é usado um mecanismo de configuração automática (DHCP, DHCPv6, SLAAC). Para obter mais detalhes ou ver configurações mais complexas, consulte a *Seção 42.6, "Configuração avançada de rede"*.

Depois que o arquivo for criado, execute o seguinte comando no cluster de gerenciamento para iniciar o registro do novo host bare metal:

```
$ kubectl apply -f bmh-example.yaml
```

O novo objeto host bare metal será registrado, o que altera seu estado de "registrando" para "inspecionando" e "disponível". Para verificar as alterações, use o seguinte comando:

```
$ kubectl get bmh
```

Nota

O objeto `BaremetalHost` fica no estado `registering` até a validação das credenciais do `BMC`. Depois disso, o estado do objeto `BaremetalHost` muda para `inspecting`, e essa etapa pode levar algum tempo dependendo do hardware (até 20 minutos). Durante a fase de inspeção, as informações do hardware são recuperadas e o objeto Kubernetes é atualizado. Para verificar as informações, use o seguinte comando: `kubectl get bmh -o yaml`.

Etapa de provisionamento

Depois que o host bare metal for registrado e estiver disponível, a próxima etapa será provisioná-lo para instalar e configurar o sistema operacional e o cluster Kubernetes. Para fazer isso, é necessário criar este arquivo (`capi-provisioning-example.yaml`) no cluster de gerenciamento com as seguintes informações (é possível gerar o `capi-provisioning-example.yaml` unindo os blocos abaixo).

Nota

Apenas os valores entre `$\{…\}` devem ser substituídos pelos valores reais.

O bloco a seguir é a definição do cluster, em que a rede pode ser configurada usando os blocos `pods` e `services`. Ele também contém as referências aos objetos de plano de controle e infraestrutura (usando o provedor `Metal3`) a serem usados.

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: single-node-cluster
  namespace: default
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - 192.168.0.0/18
    services:
      cidrBlocks:
        - 10.96.0.0/12
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: RKE2ControlPlane
    name: single-node-cluster
```

```
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3Cluster
    name: single-node-cluster
```

Para uma implantação com pods e serviços de pilha dupla, a alternativa é usar a seguinte definição:

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: single-node-cluster
  namespace: default
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - 192.168.0.0/18
        - fd00:bad:cafe::/48
    services:
      cidrBlocks:
        - 10.96.0.0/12
        - fd00:bad:bad:cafe::/112
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: RKE2ControlPlane
    name: single-node-cluster
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3Cluster
    name: single-node-cluster
```

Importante

As implantações de IPv6 e pilha dupla estão com status de prévia de tecnologia e não contam com suporte oficial.

O objeto `Metal3Cluster` especifica o endpoint do plano de controle (para substituir o `${DOWNSTREAM_CONTROL_PLANE_IP}`) que será configurado e o `noCloudProvider`, pois é usado um nó bare metal.

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3Cluster
metadata:
  name: single-node-cluster
```

```
  namespace: default
spec:
  controlPlaneEndpoint:
    host: ${DOWNSTREAM_CONTROL_PLANE_IP}
    port: 6443
  noCloudProvider: true
```

O objeto `RKE2ControlPlane` especifica a configuração do plano de controle e o objeto `Metal3MachineTemplate` especifica a imagem do plano de controle que serão usadas. Ele também contém as informações sobre o número de réplicas (neste caso, uma) e o plug-in de `CNI` (neste caso, `Cilium`) que serão usados. O bloco agentConfig contém o formato do `Ignition` e o `additionalUserData` a serem usados para configurar o nó `RKE2`, com informações como um systemd chamado `rke2-preinstall.service` para substituir automaticamente o `BAREMETALHOST_UUID` e o `node-name` durante o processo de provisionamento usando as informações do Ironic. Para permitir o multus com o cilium, um arquivo é criado no diretório de manifestos do servidor `rke2` chamado `rke2-cilium-config.yaml` com a configuração que será usada. O último bloco de informações inclui a versão do Kubernetes a ser usada. `${RKE2_VERSION}` é a versão do `RKE2` que substituirá esse valor (por exemplo, `v1.32.4+rke2r1`).

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
metadata:
  name: single-node-cluster
  namespace: default
spec:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3MachineTemplate
    name: single-node-cluster-controlplane
  replicas: 1
  version: ${RKE2_VERSION}
  rolloutStrategy:
    type: "RollingUpdate"
    rollingUpdate:
      maxSurge: 0
  serverConfig:
    cni: cilium
  agentConfig:
    format: ignition
    additionalUserData:
      config: |
        variant: fcos
        version: 1.4.0
        systemd:
          units:
```

```
            - name: rke2-preinstall.service
              enabled: true
              contents: |
                [Unit]
                Description=rke2-preinstall
                Wants=network-online.target
                Before=rke2-install.service
                ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
                [Service]
                Type=oneshot
                User=root
                ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
                ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/$(jq -r .uuid /mnt/
openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
                ExecStart=/bin/sh -c "echo \"node-name: $(jq -r .name /mnt/openstack/
latest/meta_data.json)\" >> /etc/rancher/rke2/config.yaml"
                ExecStartPost=/bin/sh -c "umount /mnt"
                [Install]
                WantedBy=multi-user.target
        storage:
          files:
            # https://docs.rke2.io/networking/multus_sriov#using-multus-with-cilium
            - path: /var/lib/rancher/rke2/server/manifests/rke2-cilium-config.yaml
              overwrite: true
              contents:
                inline: |
                  apiVersion: helm.cattle.io/v1
                  kind: HelmChartConfig
                  metadata:
                    name: rke2-cilium
                    namespace: kube-system
                  spec:
                    valuesContent: |-
                      cni:
                        exclusive: false
              mode: 0644
              user:
                name: root
              group:
                name: root
    kubelet:
      extraArgs:
        - provider-id=metal3://BAREMETALHOST_UUID
    nodeName: "localhost.localdomain"
```

O objeto `Metal3MachineTemplate` especifica as seguintes informações:

- O `dataTemplate` para usar como referência para o gabarito.
- O `hostSelector` a ser usado correspondente ao rótulo criado durante o processo de registro.
- A `image` que será usada como referência à imagem gerada pelo `EIB` na seção anterior (*Seção 42.2, "Preparar uma imagem de cluster downstream para cenários conectados"*), e o `checksum` e o `checksumType` usados para validar a imagem.

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3MachineTemplate
metadata:
  name: single-node-cluster-controlplane
  namespace: default
spec:
  template:
    spec:
      dataTemplate:
        name: single-node-cluster-controlplane-template
      hostSelector:
        matchLabels:
          cluster-role: control-plane
      image:
        checksum: http://imagecache.local:8080/eibimage-output-telco.raw.sha256
        checksumType: sha256
        format: raw
        url: http://imagecache.local:8080/eibimage-output-telco.raw
```

O objeto `Metal3DataTemplate` especifica o `metaData` para o cluster downstream.

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3DataTemplate
metadata:
  name: single-node-cluster-controlplane-template
  namespace: default
spec:
  clusterName: single-node-cluster
  metaData:
    objectNames:
      - key: name
        object: machine
      - key: local-hostname
        object: machine
```

```
      - key: local_hostname
        object: machine
```

Após a criação do arquivo unindo os blocos anteriores, execute o seguinte comando no cluster de gerenciamento para iniciar o provisionamento do novo host bare metal:

```
$ kubectl apply -f capi-provisioning-example.yaml
```

42.5 Provisionamento de cluster downstream com provisionamento de rede direcionado (vários nós)

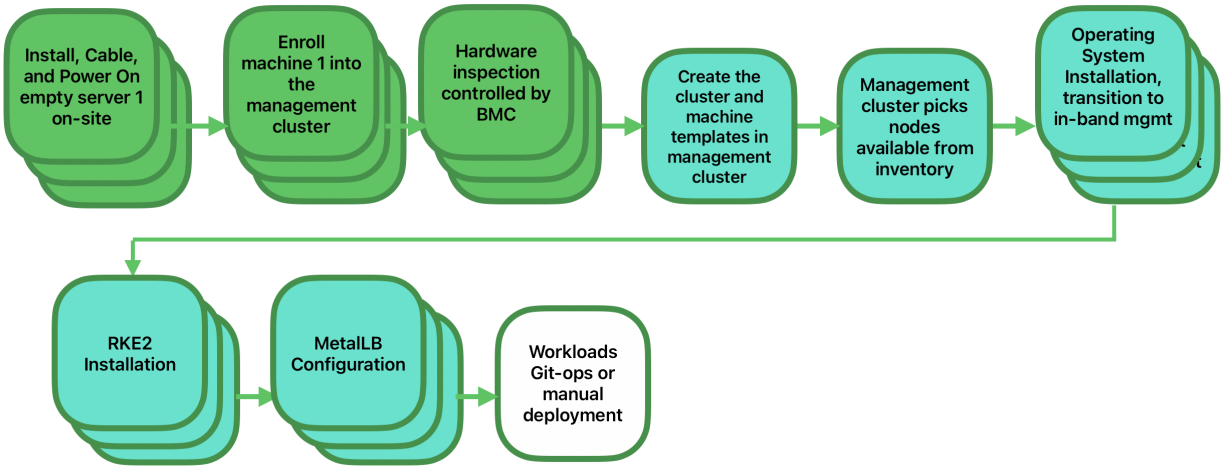
Esta seção descreve o fluxo de trabalho usado para automatizar o provisionamento de um cluster downstream de vários nós usando o provisionamento de rede direcionado e o `MetalLB` como estratégia de balanceador de carga. Essa é a maneira mais simples de automatizar o provisionamento de clusters downstream. O diagrama abaixo mostra o fluxo de trabalho usado para automatizar o provisionamento de um cluster downstream de vários nós usando o provisionamento de rede direcionado e o `MetalLB`.

Requisitos

- A imagem gerada por meio do `EIB`, conforme descrito na seção anterior (*Seção 42.2, "Preparar uma imagem de cluster downstream para cenários conectados"*), com a configuração mínima para definir o cluster downstream, deve estar localizada no cluster de gerenciamento exatamente no caminho configurado nesta seção (*Nota*).
- O servidor de gerenciamento criado e disponível para uso nas seções a seguir. Para obter mais informações, consulte a seção sobre o cluster de gerenciamento: *Capítulo 40, Configurando o cluster de gerenciamento*.

Fluxo de trabalho

O diagrama a seguir mostra o fluxo de trabalho usado para automatizar o provisionamento de um cluster downstream de vários nós por meio do provisionamento de rede direcionado:



1. Registre os três hosts bare metal para disponibilizá-los ao processo de provisionamento.
2. Provisione os três hosts bare metal para instalar e configurar o sistema operacional e o cluster Kubernetes usando o `MetalLB`.

Registrar os hosts bare metal

A primeira etapa é registrar os três hosts bare metal no cluster de gerenciamento para disponibilizá-los para provisionamento. Para fazer isso, é necessário criar estes arquivos (`bmh-example-node1.yaml`, `bmh-example-node2.yaml` e `bmh-example-node3.yaml`) no cluster de gerenciamento, para especificar as credenciais que serão usadas do `BMC` e o objeto `BaremetalHost` para registro no cluster de gerenciamento.

Nota

- Apenas os valores entre `$\{…\}` devem ser substituídos pelos valores reais.
- Vamos orientar você apenas pelo processo de um host. As mesmas etapas são válidas para os outros dois nós.

```
apiVersion: v1
kind: Secret
```

```
metadata:
  name: node1-example-credentials
type: Opaque
data:
  username: ${BMC_NODE1_USERNAME}
  password: ${BMC_NODE1_PASSWORD}
---
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: node1-example
  labels:
    cluster-role: control-plane
spec:
  online: true
  bootMACAddress: ${BMC_NODE1_MAC}
  bmc:
    address: ${BMC_NODE1_ADDRESS}
    disableCertificateVerification: true
    credentialsName: node1-example-credentials
```

Em que:

- `${BMC_NODE1_USERNAME}`: o nome de usuário para o BMC do primeiro host bare metal.
- `${BMC_NODE1_PASSWORD}`: a senha para o BMC do primeiro host bare metal.
- `${BMC_NODE1_MAC}`: o endereço MAC do primeiro host bare metal que será usado.
- `${BMC_NODE1_ADDRESS}`: o URL do primeiro host bare metal BMC (por exemplo, `redfish-virtualmedia://192.168.200.75/redfish/v1/Systems/1/`). Para saber mais sobre as diversas opções disponíveis de acordo com o seu provedor de hardware, acesse este link (https://github.com/metal3-io/baremetal-operator/blob/main/docs/api.md).

Nota

- Se não foi especificada uma configuração de rede para o host, seja no momento da criação da imagem, seja pela definição `BareMetalHost`, é usado um mecanismo de configuração automática (DHCP, DHCPv6, SLAAC). Para obter mais detalhes ou ver configurações mais complexas, consulte a *Seção 42.6, "Configuração avançada de rede"*.
- Ainda não há suporte para clusters de pilha dupla de vários nós ou somente IPv6.

Depois que o arquivo for criado, execute o seguinte comando no cluster de gerenciamento para iniciar o registro dos hosts bare metal:

```
$ kubectl apply -f bmh-example-node1.yaml
$ kubectl apply -f bmh-example-node2.yaml
$ kubectl apply -f bmh-example-node3.yaml
```

Os novos objetos host bare metal serão registrados, o que altera o estado deles de "registrando" para "inspecionando" e "disponíveis". Para verificar as alterações, use o seguinte comando:

```
$ kubectl get bmh -o wide
```

Nota

O objeto `BaremetalHost` fica no estado `registering` até a validação das credenciais do `BMC`. Depois disso, o estado do objeto `BaremetalHost` muda para `inspecting`, e essa etapa pode levar algum tempo dependendo do hardware (até 20 minutos). Durante a fase de inspeção, as informações do hardware são recuperadas e o objeto Kubernetes é atualizado. Para verificar as informações, use o seguinte comando: `kubectl get bmh -o yaml`.

Etapa de provisionamento

Depois que os três hosts bare metal forem registrados e estiverem disponíveis, a próxima etapa será provisioná-los para instalar e configurar o sistema operacional e o cluster Kubernetes, criando um balanceador de carga para gerenciá-los. Para fazer isso, é necessário criar este arquivo (`capi-provisioning-example.yaml`) no cluster de gerenciamento com as seguintes informações (é possível gerar o `capi-provisioning-example.yaml unindo os blocos abaixo).

Nota

- Apenas os valores entre `$\{…\}` devem ser substituídos pelos valores reais.
- O endereço `VIP` é um IP reservado não atribuído a nenhum nó e usado para configurar o balanceador de carga.

Veja a seguir a definição do cluster, em que a rede do cluster pode ser configurada usando os blocos `pods` e `services`. Ela também contém as referências aos objetos de plano de controle e infraestrutura (usando o provedor `Metal3`) a serem usados.

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: multinode-cluster
  namespace: default
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - 192.168.0.0/18
    services:
      cidrBlocks:
        - 10.96.0.0/12
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: RKE2ControlPlane
    name: multinode-cluster
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3Cluster
    name: multinode-cluster
```

O objeto `Metal3Cluster` especifica o endpoint do plano de controle que usa o endereço `VIP` já reservado (para substituir o `${DOWNSTREAM_VIP_ADDRESS}`) que será configurado e o `noCloudProvider`, pois são usados três nós bare metal.

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3Cluster
metadata:
  name: multinode-cluster
  namespace: default
spec:
  controlPlaneEndpoint:
    host: ${EDGE_VIP_ADDRESS}
    port: 6443
  noCloudProvider: true
```

O objeto `RKE2ControlPlane` especifica a configuração do plano de controle e o objeto `Metal3MachineTemplate` especifica a imagem do plano de controle que serão usadas.

- O número de réplicas a ser usado (neste caso, três).
- O modo de anúncio usado pelo balanceador de carga (`address` usa a implementação L2) e o endereço que será usado (substituindo o `${EDGE_VIP_ADDRESS}` pelo endereço `VIP`).
- O `serverConfig` com o plug-in de `CNI` usado (neste caso, `Cilium`) e o `tlsSan` usado para configurar o endereço `VIP`.

- O bloco agentConfig inclui o formato usado do `Ignition` e o `additionalUserData` usado para configurar o nó `RKE2` com informações como:
    - O serviço systemd chamado `rke2-preinstall.service` para substituir automaticamente o `BAREMETALHOST_UUID` e o `node-name` durante o processo de provisionamento com as informações do Ironic.
    - O bloco `storage` que contém os gráficos Helm que serão usados para instalar o `MetalLB` e o `endpoint-copier-operator`.
    - O arquivo de recurso personalizado `metalLB` com o `IPaddressPool` e o `L2Advertisement` que serão usados (substituindo `${EDGE_VIP_ADDRESS}` pelo endereço `VIP`).
    - O arquivo `endpoint-svc.yaml` que será usado para configurar o serviço `kubernetes-vip` usado pelo `MetalLB` para gerenciar o endereço `VIP`.
- Os último bloco de informações contém a versão do Kubernetes que será usada. A `${RKE2_VERSION}` é a versão do `RKE2` que será usada no lugar desse valor (por exemplo, `v1.32.4+rke2r1`).

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
metadata:
  name: multinode-cluster
  namespace: default
spec:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3MachineTemplate
    name: multinode-cluster-controlplane
  replicas: 3
  version: ${RKE2_VERSION}
  rolloutStrategy:
    type: "RollingUpdate"
    rollingUpdate:
      maxSurge: 0
  registrationMethod: "control-plane-endpoint"
  registrationAddress: ${EDGE_VIP_ADDRESS}
  serverConfig:
    cni: cilium
    tlsSan:
      - ${EDGE_VIP_ADDRESS}
      - https://${EDGE_VIP_ADDRESS}.sslip.io
```

```
  agentConfig:
    format: ignition
    additionalUserData:
      config: |
        variant: fcos
        version: 1.4.0
        systemd:
          units:
            - name: rke2-preinstall.service
              enabled: true
              contents: |
                [Unit]
                Description=rke2-preinstall
                Wants=network-online.target
                Before=rke2-install.service
                ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
                [Service]
                Type=oneshot
                User=root
                ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
                ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/$(jq -r .uuid /mnt/
openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
                ExecStart=/bin/sh -c "echo \"node-name: $(jq -r .name /mnt/openstack/
latest/meta_data.json)\" >> /etc/rancher/rke2/config.yaml"
                ExecStartPost=/bin/sh -c "umount /mnt"
                [Install]
                WantedBy=multi-user.target
        storage:
          files:
            # https://docs.rke2.io/networking/multus_sriov#using-multus-with-cilium
            - path: /var/lib/rancher/rke2/server/manifests/rke2-cilium-config.yaml
              overwrite: true
              contents:
                inline: |
                  apiVersion: helm.cattle.io/v1
                  kind: HelmChartConfig
                  metadata:
                    name: rke2-cilium
                    namespace: kube-system
                  spec:
                    valuesContent: |-
                      cni:
                        exclusive: false
              mode: 0644
              user:
                name: root
              group:
```

```
    name: root
- path: /var/lib/rancher/rke2/server/manifests/endpoint-copier-operator.yaml
  overwrite: true
  contents:
    inline: |
      apiVersion: helm.cattle.io/v1
      kind: HelmChart
      metadata:
        name: endpoint-copier-operator
        namespace: kube-system
      spec:
        chart: oci://registry.suse.com/edge/charts/endpoint-copier-operator
        targetNamespace: endpoint-copier-operator
        version: 303.0.0+up0.2.1
        createNamespace: true
- path: /var/lib/rancher/rke2/server/manifests/metallb.yaml
  overwrite: true
  contents:
    inline: |
      apiVersion: helm.cattle.io/v1
      kind: HelmChart
      metadata:
        name: metallb
        namespace: kube-system
      spec:
        chart: oci://registry.suse.com/edge/charts/metallb
        targetNamespace: metallb-system
        version: 303.0.0+up0.14.9
        createNamespace: true

- path: /var/lib/rancher/rke2/server/manifests/metallb-cr.yaml
  overwrite: true
  contents:
    inline: |
      apiVersion: metallb.io/v1beta1
      kind: IPAddressPool
      metadata:
        name: kubernetes-vip-ip-pool
        namespace: metallb-system
      spec:
        addresses:
          - ${EDGE_VIP_ADDRESS}/32
        serviceAllocation:
          priority: 100
          namespaces:
            - default
          serviceSelectors:
```

```
                    - matchExpressions:
                      - {key: "serviceType", operator: In, values: [kubernetes-vip]}
                ---
                apiVersion: metallb.io/v1beta1
                kind: L2Advertisement
                metadata:
                  name: ip-pool-l2-adv
                  namespace: metallb-system
                spec:
                  ipAddressPools:
                    - kubernetes-vip-ip-pool
          - path: /var/lib/rancher/rke2/server/manifests/endpoint-svc.yaml
            overwrite: true
            contents:
              inline: |
                apiVersion: v1
                kind: Service
                metadata:
                  name: kubernetes-vip
                  namespace: default
                  labels:
                    serviceType: kubernetes-vip
                spec:
                  ports:
                  - name: rke2-api
                    port: 9345
                    protocol: TCP
                    targetPort: 9345
                  - name: k8s-api
                    port: 6443
                    protocol: TCP
                    targetPort: 6443
                  type: LoadBalancer
    kubelet:
      extraArgs:
        - provider-id=metal3://BAREMETALHOST_UUID
    nodeName: "Node-multinode-cluster"
```

O objeto `Metal3MachineTemplate` especifica as seguintes informações:

- O `dataTemplate` para usar como referência para o gabarito.
- O `hostSelector` a ser usado correspondente ao rótulo criado durante o processo de registro.
- A `image` que será usada como referência à imagem gerada pelo `EIB` na seção anterior (*Seção 42.2, "Preparar uma imagem de cluster downstream para cenários conectados"*), e o `checksum` e o `checksumType` usados para validar a imagem.

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3MachineTemplate
metadata:
  name: multinode-cluster-controlplane
  namespace: default
spec:
  template:
    spec:
      dataTemplate:
        name: multinode-cluster-controlplane-template
      hostSelector:
        matchLabels:
          cluster-role: control-plane
      image:
        checksum: http://imagecache.local:8080/eibimage-output-telco.raw.sha256
        checksumType: sha256
        format: raw
        url: http://imagecache.local:8080/eibimage-output-telco.raw
```

O objeto `Metal3DataTemplate` especifica o `metaData` para o cluster downstream.

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3DataTemplate
metadata:
  name: multinode-cluster-controlplane-template
  namespace: default
spec:
  clusterName: multinode-cluster
  metaData:
    objectNames:
      - key: name
        object: machine
      - key: local-hostname
        object: machine
      - key: local_hostname
```

```
        object: machine
```

Após a criação do arquivo unindo os blocos anteriores, execute o seguinte comando no cluster de gerenciamento para iniciar o provisionamento dos três novos hosts bare metal:

```
$ kubectl apply -f capi-provisioning-example.yaml
```

42.6 Configuração avançada de rede

O fluxo de trabalho de provisionamento de rede direcionado permite fazer configurações de rede específicas nos clusters downstream, como IPs estáticos, vínculo, VLANs, IPv6 etc.

As seções a seguir descrevem as etapas adicionais necessárias para permitir o provisionamento de clusters downstream usando a configuração de rede avançada.

Requisitos

- A imagem gerada pelo `EIB` deve incluir a pasta de rede e o script, de acordo com as instruções nesta seção (*Seção 42.2.2.6, "Script adicional para configuração de rede avançada"*).

Configuração

Antes de continuar, consulte uma das seções a seguir para obter orientação sobre as etapas necessárias para registrar e provisionar os hosts:

- Provisionamento de cluster downstream com provisionamento de rede direcionado (nó único) (*Seção 42.4, "Provisionamento de cluster downstream com provisionamento de rede direcionado (nó único)"*)
- Provisionamento de cluster downstream com provisionamento de rede direcionado (vários nós) (*Seção 42.5, "Provisionamento de cluster downstream com provisionamento de rede direcionado (vários nós)"*)

É necessário aplicar uma configuração de rede avançada no momento do registro por meio de uma definição do host `BareMetalHost` e um segredo associado que contém um bloco `networkData` no formato `nmstate`. O arquivo de exemplo a seguir define um segredo com o `networkData` necessário que solicita um `IP` estático e a `VLAN` para o host do cluster downstream:

```
apiVersion: v1
kind: Secret
metadata:
  name: controlplane-0-networkdata
type: Opaque
```

```
stringData:
  networkData: |
    interfaces:
    - name: ${CONTROLPLANE_INTERFACE}
      type: ethernet
      state: up
      mtu: 1500
      identifier: mac-address
      mac-address: "${CONTROLPLANE_MAC}"
      ipv4:
        address:
        - ip:  "${CONTROLPLANE_IP}"
          prefix-length: "${CONTROLPLANE_PREFIX}"
        enabled: true
        dhcp: false
    - name: floating
      type: vlan
      state: up
      vlan:
        base-iface: ${CONTROLPLANE_INTERFACE}
        id: ${VLAN_ID}
    dns-resolver:
      config:
        server:
        - "${DNS_SERVER}"
    routes:
      config:
      - destination: 0.0.0.0/0
        next-hop-address: "${CONTROLPLANE_GATEWAY}"
        next-hop-interface: ${CONTROLPLANE_INTERFACE}
```

Como você pode ver, o exemplo mostra a configuração para habilitar a interface com os IPs estáticos e a VLAN usando a interface base, depois que as seguintes variáveis são substituídas pelos valores reais, de acordo com a sua infraestrutura:

- `${CONTROLPLANE1_INTERFACE}`: a interface do plano de controle que será usada para o cluster Edge (por exemplo, `eth0`). Incluindo o `identifier: mac-address`, o endereço MAC inspeciona automaticamente a nomenclatura para que qualquer nome de interface possa ser usado.
- `${CONTROLPLANE1_IP}`: o endereço IP usado como endpoint para o cluster de borda (deve corresponder ao endpoint do servidor kubeapi).
- `${CONTROLPLANE1_PREFIX}`: o CIDR usado para o cluster de borda (por exemplo, `24` para `/24` ou `255.255.255.0`).

- `${CONTROLPLANE1_GATEWAY}`: o gateway usado para o cluster de borda (por exemplo, `192.168.100.1`).
- `${CONTROLPLANE1_MAC}`: o endereço MAC usado para a interface do plano de controle (por exemplo, `00:0c:29:3e:3e:3e`).
- `${DNS_SERVER}`: o DNS usado para o cluster de borda (por exemplo, `192.168.100.2`).
- `${VLAN_ID}`: o ID da VLAN usado para o cluster de borda (por exemplo, `100`).

É possível usar qualquer outra definição adequada a `nmstate` para configurar a rede do cluster downstream de acordo com os requisitos específicos. Por exemplo, é possível especificar uma configuração de pilha dupla estática:

```
apiVersion: v1
kind: Secret
metadata:
  name: controlplane-0-networkdata
type: Opaque
stringData:
  networkData: |
    interfaces:
    - name: ${CONTROLPLANE_INTERFACE}
      type: ethernet
      state: up
      mac-address: ${CONTROLPLANE_MAC}
      ipv4:
        enabled: true
        dhcp: false
        address:
        - ip: ${CONTROLPLANE_IP_V4}
          prefix-length: ${CONTROLPLANE_PREFIX_V4}
      ipv6:
        enabled: true
        dhcp: false
        autoconf: false
        address:
        - ip: ${CONTROLPLANE_IP_V6}
          prefix-length: ${CONTROLPLANE_PREFIX_V6}
    routes:
      config:
      - destination: 0.0.0.0/0
        next-hop-address: ${CONTROLPLANE_GATEWAY_V4}
        next-hop-interface: ${CONTROLPLANE_INTERFACE}
      - destination: ::/0
        next-hop-address: ${CONTROLPLANE_GATEWAY_V6}
```

```
      next-hop-interface: ${CONTROLPLANE_INTERFACE}
  dns-resolver:
    config:
      server:
      - ${DNS_SERVER_V4}
      - ${DNS_SERVER_V6}
```

Assim como no exemplo anterior, substitua as seguintes variáveis pelos valores reais, de acordo com a sua infraestrutura:

- `${CONTROLPLANE_IP_V4}`: o endereço IPv4 para atribuir ao host
- `${CONTROLPLANE_PREFIX_V4}`: o prefixo IPv4 da rede à qual pertence o IP do host
- `${CONTROLPLANE_IP_V6}`: o endereço IPv6 para atribuir ao host
- `${CONTROLPLANE_PREFIX_V6}`: o prefixo IPv6 da rede à qual pertence o IP do host
- `${CONTROLPLANE_GATEWAY_V4}`: o endereço IPv4 do gateway para o tráfego correspondente à rota padrão
- `${CONTROLPLANE_GATEWAY_V6}`: o endereço IPv6 do gateway para o tráfego correspondente à rota padrão
- `${CONTROLPLANE_INTERFACE}`: o nome da interface à qual atribuir os endereços e que será usada para o tráfego de saída correspondente à rota padrão, para ambos IPv4 e IPv6
- `${DNS_SERVER_V4}` e/ou `${DNS_SERVER_V6}`: os endereços IP dos servidores DNS, que podem ser especificados como uma única ou várias entradas. Há suporte para os endereços tanto IPv4 quanto IPv6

Importante

As implantações de IPv6 e pilha dupla estão com status de prévia de tecnologia e não contam com suporte oficial.

Nota

Acesse o repositório de exemplos do SUSE Edge for Telco (https://github.com/suse-edge/atip/tree/main/telco-examples/edge-clusters) para ver exemplos mais complexos, incluindo configurações somente IPv6 e de pilha dupla.

Por fim, sejam quais forem os detalhes da configuração de rede, garanta que o segredo seja referenciado anexando `preprovisioningNetworkDataName` ao objeto `BaremetalHost` para registrar o host no cluster de gerenciamento com sucesso.

```
apiVersion: v1
kind: Secret
metadata:
  name: example-demo-credentials
type: Opaque
data:
  username: ${BMC_USERNAME}
  password: ${BMC_PASSWORD}
---
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: example-demo
  labels:
    cluster-role: control-plane
spec:
  online: true
  bootMACAddress: ${BMC_MAC}
  rootDeviceHints:
    deviceName: /dev/nvme0n1
  bmc:
    address: ${BMC_ADDRESS}
    disableCertificateVerification: true
    credentialsName: example-demo-credentials
  preprovisioningNetworkDataName: controlplane-0-networkdata
```

Nota

- Se você precisa implantar um cluster de vários nós, siga o mesmo processo para cada nó.
- Atualmente, o `Metal3DataTemplate`, o `networkData` e o `Metal3 IPAM` não são suportados. Apenas a configuração por meio de segredos estáticos conta com suporte total.

42.7 Recursos de telecomunicações (DPDK, SR-IOV, isolamento de CPU, HugePages, NUMA etc.)

O fluxo de trabalho de provisionamento de rede direcionado permite automatizar os recursos de telecomunicações que serão usados nos clusters downstream para executar as cargas de trabalho de telecomunicações nesses servidores.

Requisitos

- A imagem gerada por meio do `EIB`, conforme descrito na seção anterior (*Seção 42.2, "Preparar uma imagem de cluster downstream para cenários conectados"*), deve estar localizada no cluster de gerenciamento exatamente no caminho configurado nesta seção (*Nota*).
- A imagem gerada por meio do `EIB` deve incluir os pacotes de telecomunicações específicos de acordo com as instruções nesta seção (*Seção 42.2.2.5, "Configuração adicional para cargas de trabalho de telecomunicações"*).
- O servidor de gerenciamento criado e disponível para uso nas seções a seguir. Para obter mais informações, consulte a seção sobre o cluster de gerenciamento: *Capítulo 40, Configurando o cluster de gerenciamento*.

Configuração

Use as duas seções a seguir como base para registrar e provisionar os hosts:

- Provisionamento de cluster downstream com provisionamento de rede direcionado (nó único) (*Seção 42.4, "Provisionamento de cluster downstream com provisionamento de rede direcionado (nó único)"*)
- Provisionamento de cluster downstream com provisionamento de rede direcionado (vários nós) (*Seção 42.5, "Provisionamento de cluster downstream com provisionamento de rede direcionado (vários nós)"*)

Os recursos de telecomunicações abordados nesta seção são:

- DPDK e criação de VFs
- Alocação de SR-IOV e VFs para as cargas de trabalho usarem
- Isolamento de CPU e ajuste de desempenho
- Configuração do HugePages
- Ajuste de parâmetros do kernel

Nota

Para obter mais informações sobre os recursos de telecomunicações, consulte o *Capítulo 41, Configuração dos recursos de telecomunicações*.

As alterações necessárias para habilitar os recursos de telecomunicações mostrados acima estão todas no bloco `RKE2ControlPlane` do arquivo de provisionamento `capi-provisioning-example.yaml`. O restante das informações no arquivo `capi-provisioning-example.yaml` é o mesmo que as informações apresentadas na seção de provisionamento (*Seção 42.4, "Provisionamento de cluster downstream com provisionamento de rede direcionado (nó único)"* (p 493)).

Para esclarecer o processo, as alterações necessárias neste bloco (`RKE2ControlPlane`) para habilitar os recursos de telecomunicações são:

- Os `preRKE2Commands` usados para executar os comandos antes do processo de instalação do `RKE2`. Neste caso, use o comando `modprobe` para habilitar os módulos do kernel `vfio-pci` e `SR-IOV`.
- O arquivo do Ignition `/var/lib/rancher/rke2/server/manifests/configmap-sriov-custom-auto.yaml` que será usado para definir as interfaces, os drivers e o número de `VFs` que será criado e exposto às cargas de trabalho.
    - Os valores no mapa de configuração `sriov-custom-auto-config` são os únicos que precisam ser substituídos pelos valores reais.
        - `${RESOURCE_NAME1}`: o nome do recurso usado para a primeira interface `PF` (por exemplo, `sriov-resource-du1`). Ele é adicionado ao prefixo `rancher.io` usado como o rótulo que as cargas de trabalho usarão (por exemplo, `rancher.io/sriov-resource-du1`).
        - `${SRIOV-NIC-NAME1}`: o nome da primeira interface `PF` que será usada (por exemplo, `eth0`).
        - `${PF_NAME1}`: o nome da primeira função física `PF` que será usada. Gere filtros mais complexos usando este comando (por exemplo, `eth0#2-5`).

    - `${DRIVER_NAME1}`: o nome do driver que será usado para a primeira interface `VF` (por exemplo, `vfio-pci`).
    - `${NUM_VFS1}`: o número de `VFs` que será criado para a primeira interface `PF` (por exemplo, `8`).
- O `/var/sriov-auto-filler.sh` usado como conversor entre o mapa de configuração de alto nível `sriov-custom-auto-config` e o `sriovnetworknodepolicy`, que contém as informações de hardware de baixo nível. Esse script foi criado para eliminar a complexidade do usuário em ter que saber com antecedência as informações do hardware. Não há necessidade de alterações no arquivo, mas ele deverá estar presente se precisarmos habilitar `sr-iov` e criar `VFs`.
- Os argumentos do kernel que serão usados para habilitar os seguintes recursos:

Parâmetro	Valor	Descrição
isolcpus	domain,nohz,managed_irq,1-30,33-62	Isola os núcleos 1-30 e 33-62.
skew_tick	1	Permite que o kernel distribua as interrupções do temporizador entre as CPUs isoladas.
nohz	on	Permite que o kernel execute o tick do temporizador em uma única CPU quando o sistema está ocioso.
nohz_full	1-30,33-62	O parâmetro de inicialização do kernel é a interface principal atual para configurar dynticks completos junto com o isolamento de CPU.

rcu_nocbs	1-30,33-62	Permite que o kernel execute os retornos de chamada RCU em uma única CPU quando o sistema está ocioso.
irqaffinity	0,31,32,63	Permite que o kernel execute as interrupções em uma única CPU quando o sistema está ocioso.
idle	poll	Minimiza a latência de sair do estado ocioso.
iommu	pt	Permite usar vfio para as interfaces dpdk.
intel_iommu	on	Permite usar vfio para VFs.
hugepagesz	1G	Permite definir o tamanho das páginas enormes como 1 G.
hugepages	40	O número de páginas enormes definido antes.
default_hugepagesz	1G	O valor padrão para habilitar o HugePages.
nowatchdog		Desabilita o watchdog.
nmi_watchdog	0	Desabilita o watchdog de NMI.

- Os serviços systemd abaixo são usados para habilitar o seguinte:
  - `rke2-preinstall.service` para substituir automaticamente o `BAREMETALHOST_UUID` e o `node-name` durante o processo de provisionamento com as informações do Ironic.
  - `cpu-partitioning.service` para habilitar os núcleos de isolamento da `CPU` (por exemplo, `1-30,33-62`).
  - `performance-settings.service` para habilitar o ajuste de desempenho da CPU.
  - `sriov-custom-auto-vfs.service` para instalar o gráfico Helm `sriov`. Aguarde a criação dos recursos personalizados e execute o `/var/sriov-auto-filler.sh` para substituir os valores no mapa de configuração `sriov-custom-auto-config` e criar o `sriovnetworknodepolicy` que as cargas de trabalho usarão.
- A `${RKE2_VERSION}` é a versão do `RKE2` usada no lugar desse valor (por exemplo, `v1.32.4+rke2r1`).

Com todas essas alterações mencionadas, o bloco `RKE2ControlPlane` no `capi-provisioning-example.yaml` terá a seguinte aparência:

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
metadata:
  name: single-node-cluster
  namespace: default
spec:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3MachineTemplate
    name: single-node-cluster-controlplane
  replicas: 1
  version: ${RKE2_VERSION}
  rolloutStrategy:
    type: "RollingUpdate"
    rollingUpdate:
      maxSurge: 0
  serverConfig:
    cni: calico
    cniMultusEnable: true
  preRKE2Commands:
    - modprobe vfio-pci enable_sriov=1 disable_idle_d3=1
  agentConfig:
    format: ignition
```

```
    additionalUserData:
      config: |
        variant: fcos
        version: 1.4.0
        storage:
          files:
            - path: /var/lib/rancher/rke2/server/manifests/configmap-sriov-custom-
auto.yaml
              overwrite: true
              contents:
                inline: |
                  apiVersion: v1
                  kind: ConfigMap
                  metadata:
                    name: sriov-custom-auto-config
                    namespace: kube-system
                  data:
                    config.json: |
                      [
                         {
                           "resourceName": "${RESOURCE_NAME1}",
                           "interface": "${SRIOV-NIC-NAME1}",
                           "pfname": "${PF_NAME1}",
                           "driver": "${DRIVER_NAME1}",
                           "numVFsToCreate": ${NUM_VFS1}
                         },
                         {
                           "resourceName": "${RESOURCE_NAME2}",
                           "interface": "${SRIOV-NIC-NAME2}",
                           "pfname": "${PF_NAME2}",
                           "driver": "${DRIVER_NAME2}",
                           "numVFsToCreate": ${NUM_VFS2}
                         }
                      ]
              mode: 0644
              user:
                name: root
              group:
                name: root
            - path: /var/lib/rancher/rke2/server/manifests/sriov-crd.yaml
              overwrite: true
              contents:
                inline: |
                  apiVersion: helm.cattle.io/v1
                  kind: HelmChart
                  metadata:
                    name: sriov-crd
```

```
                namespace: kube-system
              spec:
                chart: oci://registry.suse.com/edge/charts/sriov-crd
                targetNamespace: sriov-network-operator
                version: 303.0.2+up1.5.0
                createNamespace: true
      - path: /var/lib/rancher/rke2/server/manifests/sriov-network-operator.yaml
        overwrite: true
        contents:
          inline: |
            apiVersion: helm.cattle.io/v1
            kind: HelmChart
            metadata:
              name: sriov-network-operator
              namespace: kube-system
            spec:
              chart: oci://registry.suse.com/edge/charts/sriov-network-operator
              targetNamespace: sriov-network-operator
              version: 303.0.2+up1.5.0
              createNamespace: true
    kernel_arguments:
      should_exist:
        - intel_iommu=on
        - iommu=pt
        - idle=poll
        - mce=off
        - hugepagesz=1G hugepages=40
        - hugepagesz=2M hugepages=0
        - default_hugepagesz=1G
        - irqaffinity=${NON-ISOLATED_CPU_CORES}
        - isolcpus=domain,nohz,managed_irq,${ISOLATED_CPU_CORES}
        - nohz_full=${ISOLATED_CPU_CORES}
        - rcu_nocbs=${ISOLATED_CPU_CORES}
        - rcu_nocb_poll
        - nosoftlockup
        - nowatchdog
        - nohz=on
        - nmi_watchdog=0
        - skew_tick=1
        - quiet
    systemd:
      units:
        - name: rke2-preinstall.service
          enabled: true
          contents: |
            [Unit]
            Description=rke2-preinstall
```

```
                Wants=network-online.target
                Before=rke2-install.service
                ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
                [Service]
                Type=oneshot
                User=root
                ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
                ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/$(jq -r .uuid /mnt/
openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
                ExecStart=/bin/sh -c "echo \"node-name: $(jq -r .name /mnt/openstack/
latest/meta_data.json)\" >> /etc/rancher/rke2/config.yaml"
                ExecStartPost=/bin/sh -c "umount /mnt"
                [Install]
                WantedBy=multi-user.target
            - name: cpu-partitioning.service
              enabled: true
              contents: |
                [Unit]
                Description=cpu-partitioning
                Wants=network-online.target
                After=network.target network-online.target
                [Service]
                Type=oneshot
                User=root
                ExecStart=/bin/sh -c "echo isolated_cores=${ISOLATED_CPU_CORES} > /etc/
tuned/cpu-partitioning-variables.conf"
                ExecStartPost=/bin/sh -c "tuned-adm profile cpu-partitioning"
                ExecStartPost=/bin/sh -c "systemctl enable tuned.service"
                [Install]
                WantedBy=multi-user.target
            - name: performance-settings.service
              enabled: true
              contents: |
                [Unit]
                Description=performance-settings
                Wants=network-online.target
                After=network.target network-online.target cpu-partitioning.service
                [Service]
                Type=oneshot
                User=root
                ExecStart=/bin/sh -c "/opt/performance-settings/performance-settings.sh"
                [Install]
                WantedBy=multi-user.target
            - name: sriov-custom-auto-vfs.service
              enabled: true
              contents: |
                [Unit]
```

```
                Description=SRIOV Custom Auto VF Creation
                Wants=network-online.target  rke2-server.target
                After=network.target network-online.target rke2-server.target
                [Service]
                User=root
                Type=forking
                TimeoutStartSec=900
                ExecStart=/bin/sh -c "while ! /var/lib/rancher/rke2/bin/kubectl --
kubeconfig=/etc/rancher/rke2/rke2.yaml wait --for condition=ready nodes --all ; do sleep
 2 ; done"
                ExecStartPost=/bin/sh -c "while [ $(/var/lib/rancher/
rke2/bin/kubectl --kubeconfig=/etc/rancher/rke2/rke2.yaml get
 sriovnetworknodestates.sriovnetwork.openshift.io --ignore-not-found --no-headers -A | wc
 -l) -eq 0 ]; do sleep 1; done"
                ExecStartPost=/bin/sh -c "/opt/sriov/sriov-auto-filler.sh"
                RemainAfterExit=yes
                KillMode=process
                [Install]
                WantedBy=multi-user.target
    kubelet:
      extraArgs:
        - provider-id=metal3://BAREMETALHOST_UUID
    nodeName: "localhost.localdomain"
```

Após a criação do arquivo unindo os blocos anteriores, execute o seguinte comando no cluster de gerenciamento para iniciar o provisionamento do novo cluster downstream usando os recursos de telecomunicações:

```
$ kubectl apply -f capi-provisioning-example.yaml
```

42.8 Registro particular

É possível configurar um registro particular como espelho para as imagens que as cargas de trabalho usam.

Para isso, criamos o segredo com as informações sobre o registro particular que será usado pelo cluster downstream.

```
apiVersion: v1
kind: Secret
metadata:
  name: private-registry-cert
  namespace: default
data:
  tls.crt: ${TLS_CERTIFICATE}
```

```
  tls.key: ${TLS_KEY}
  ca.crt: ${CA_CERTIFICATE}
type: kubernetes.io/tls
---
apiVersion: v1
kind: Secret
metadata:
  name: private-registry-auth
  namespace: default
data:
  username: ${REGISTRY_USERNAME}
  password: ${REGISTRY_PASSWORD}
```

O `tls.crt`, `tls.key` e `ca.crt` são os certificados usados para autenticar o registro particular. O `username` e a `password` são as credenciais usadas para autenticar o registro particular.

Nota

O `tls.crt`, `tls.key`, `ca.crt` , `username` e `password` devem ser codificados no formato base64 antes de serem usados no segredo.

Com todas essas alterações mencionadas, o bloco `RKE2ControlPlane` no `capi-provisioning-example.yaml` terá a seguinte aparência:

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
metadata:
  name: single-node-cluster
  namespace: default
spec:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3MachineTemplate
    name: single-node-cluster-controlplane
  replicas: 1
  version: ${RKE2_VERSION}
  rolloutStrategy:
    type: "RollingUpdate"
    rollingUpdate:
      maxSurge: 0
  privateRegistriesConfig:
    mirrors:
      "registry.example.com":
        endpoint:
          - "https://registry.example.com:5000"
```

```
    configs:
      "registry.example.com":
        authSecret:
          apiVersion: v1
          kind: Secret
          namespace: default
          name: private-registry-auth
        tls:
          tlsConfigSecret:
            apiVersion: v1
            kind: Secret
            namespace: default
            name: private-registry-cert
  serverConfig:
    cni: calico
    cniMultusEnable: true
  agentConfig:
    format: ignition
    additionalUserData:
      config: |
        variant: fcos
        version: 1.4.0
        systemd:
          units:
            - name: rke2-preinstall.service
              enabled: true
              contents: |
                [Unit]
                Description=rke2-preinstall
                Wants=network-online.target
                Before=rke2-install.service
                ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
                [Service]
                Type=oneshot
                User=root
                ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
                ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/$(jq -r .uuid /mnt/
openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
                ExecStart=/bin/sh -c "echo \"node-name: $(jq -r .name /mnt/openstack/
latest/meta_data.json)\" >> /etc/rancher/rke2/config.yaml"
                ExecStartPost=/bin/sh -c "umount /mnt"
                [Install]
                WantedBy=multi-user.target
    kubelet:
      extraArgs:
        - provider-id=metal3://BAREMETALHOST_UUID
    nodeName: "localhost.localdomain"
```

Em que `registry.example.com` é o nome de exemplo do registro particular usado pelo cluster downstream e deve ser substituído pelos valores reais.

42.9 Provisionamento de cluster downstream em cenários air-gapped

O fluxo de trabalho de provisionamento de rede direcionado permite automatizar o provisionamento de clusters downstream nos cenários air-gapped.

42.9.1 Requisitos para cenários air-gapped

1. A imagem `raw` gerada pelo `EIB` deve incluir as imagens do contêiner específicas (imagens OCI de gráfico Helm e do contêiner) necessárias para executar o cluster downstream em um cenário air-gapped. Para obter mais informações, consulte esta seção (*Seção 42.3, "Preparar uma imagem de cluster downstream para cenários air-gapped"*).

2. No caso de uso de SR-IOV ou qualquer outra carga de trabalho personalizada, as imagens necessárias para executar as cargas de trabalho devem ser pré-carregadas no registro particular, seguindo a seção sobre pré-carregamento no registro particular (*Seção 42.3.2.7, "Pré-carregar seu registro particular com as imagens necessárias para cenários air-gapped e SR-IOV (opcional)"*).

42.9.2 Registrar hosts bare metal em cenários air-gapped

O processo de registro de hosts bare metal no cluster de gerenciamento é o mesmo descrito na seção anterior (*Seção 42.4, "Provisionamento de cluster downstream com provisionamento de rede direcionado (nó único)"* (p 491)).

42.9.3 Provisionar o cluster downstream em cenários air-gapped

Há algumas alterações importantes necessárias para provisionar o cluster downstream em cenários air-gapped:

1. O bloco `RKE2ControlPlane` no arquivo `capi-provisioning-example.yaml` deve incluir a diretiva `spec.agentConfig.airGapped: true`.
2. É necessário incluir a configuração do registro particular no bloco `RKE2ControlPlane` do arquivo `capi-provisioning-airgap-example.yaml` de acordo com a seção de registro particular (*Seção 42.8, "Registro particular"*).
3. Se você usa SR-IOV ou qualquer outra configuração `AdditionalUserData` (script combustion) que requer a instalação de gráfico Helm, deve modificar o conteúdo para fazer referência ao registro particular, em vez de usar o registro público.

O exemplo a seguir mostra a configuração de SR-IOV no bloco `AdditionalUserData` do arquivo `capi-provisioning-airgap-example.yaml` com as modificações necessárias para fazer referência ao registro particular.

- Referências de segredos do registro particular
- Definição do gráfico Helm usando o registro particular em vez das imagens OCI públicas

```
# secret to include the private registry certificates
apiVersion: v1
kind: Secret
metadata:
  name: private-registry-cert
  namespace: default
data:
  tls.crt: ${TLS_BASE64_CERT}
  tls.key: ${TLS_BASE64_KEY}
  ca.crt: ${CA_BASE64_CERT}
type: kubernetes.io/tls
---
# secret to include the private registry auth credentials
apiVersion: v1
kind: Secret
metadata:
  name: private-registry-auth
  namespace: default
data:
  username: ${REGISTRY_USERNAME}
  password: ${REGISTRY_PASSWORD}
```

```
---
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
metadata:
  name: single-node-cluster
  namespace: default
spec:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3MachineTemplate
    name: single-node-cluster-controlplane
  replicas: 1
  version: ${RKE2_VERSION}
  rolloutStrategy:
    type: "RollingUpdate"
    rollingUpdate:
      maxSurge: 0
  privateRegistriesConfig:       # Private registry configuration to add your own mirror
 and credentials
    mirrors:
      docker.io:
        endpoint:
          - "https://$(PRIVATE_REGISTRY_URL)"
    configs:
      "192.168.100.22:5000":
        authSecret:
          apiVersion: v1
          kind: Secret
          namespace: default
          name: private-registry-auth
        tls:
          tlsConfigSecret:
            apiVersion: v1
            kind: Secret
            namespace: default
            name: private-registry-cert
          insecureSkipVerify: false
  serverConfig:
    cni: calico
    cniMultusEnable: true
  preRKE2Commands:
    - modprobe vfio-pci enable_sriov=1 disable_idle_d3=1
  agentConfig:
    airGapped: true       # Airgap true to enable airgap mode
    format: ignition
    additionalUserData:
      config: |
```

```
        variant: fcos
        version: 1.4.0
        storage:
          files:
            - path: /var/lib/rancher/rke2/server/manifests/configmap-sriov-custom-
auto.yaml
              overwrite: true
              contents:
                inline: |
                  apiVersion: v1
                  kind: ConfigMap
                  metadata:
                    name: sriov-custom-auto-config
                    namespace: sriov-network-operator
                  data:
                    config.json: |
                      [
                         {
                           "resourceName": "${RESOURCE_NAME1}",
                           "interface": "${SRIOV-NIC-NAME1}",
                           "pfname": "${PF_NAME1}",
                           "driver": "${DRIVER_NAME1}",
                           "numVFsToCreate": ${NUM_VFS1}
                         },
                         {
                           "resourceName": "${RESOURCE_NAME2}",
                           "interface": "${SRIOV-NIC-NAME2}",
                           "pfname": "${PF_NAME2}",
                           "driver": "${DRIVER_NAME2}",
                           "numVFsToCreate": ${NUM_VFS2}
                         }
                      ]
              mode: 0644
              user:
                name: root
              group:
                name: root
            - path: /var/lib/rancher/rke2/server/manifests/sriov.yaml
              overwrite: true
              contents:
                inline: |
                  apiVersion: v1
                  data:
                    .dockerconfigjson: ${REGISTRY_AUTH_DOCKERCONFIGJSON}
                  kind: Secret
                  metadata:
                    name: privregauth
```

```
    namespace: kube-system
type: kubernetes.io/dockerconfigjson
---
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: kube-system
  name: example-repo-ca
data:
  ca.crt: |-
    -----BEGIN CERTIFICATE-----
    ${CA_BASE64_CERT}
    -----END CERTIFICATE-----
---
apiVersion: helm.cattle.io/v1
kind: HelmChart
metadata:
  name: sriov-crd
  namespace: kube-system
spec:
  chart: oci://${PRIVATE_REGISTRY_URL}/sriov-crd
  dockerRegistrySecret:
    name: privregauth
  repoCAConfigMap:
    name: example-repo-ca
  createNamespace: true
  set:
    global.clusterCIDR: 192.168.0.0/18
    global.clusterCIDRv4: 192.168.0.0/18
    global.clusterDNS: 10.96.0.10
    global.clusterDomain: cluster.local
    global.rke2DataDir: /var/lib/rancher/rke2
    global.serviceCIDR: 10.96.0.0/12
  targetNamespace: sriov-network-operator
  version: 303.0.2+up1.5.0
---
apiVersion: helm.cattle.io/v1
kind: HelmChart
metadata:
  name: sriov-network-operator
  namespace: kube-system
spec:
  chart: oci://${PRIVATE_REGISTRY_URL}/sriov-network-operator
  dockerRegistrySecret:
    name: privregauth
  repoCAConfigMap:
    name: example-repo-ca
```

```
                createNamespace: true
                set:
                  global.clusterCIDR: 192.168.0.0/18
                  global.clusterCIDRv4: 192.168.0.0/18
                  global.clusterDNS: 10.96.0.10
                  global.clusterDomain: cluster.local
                  global.rke2DataDir: /var/lib/rancher/rke2
                  global.serviceCIDR: 10.96.0.0/12
                targetNamespace: sriov-network-operator
                version: 303.0.2+up1.5.0
          mode: 0644
          user:
            name: root
          group:
            name: root
    kernel_arguments:
      should_exist:
        - intel_iommu=on
        - iommu=pt
        - idle=poll
        - mce=off
        - hugepagesz=1G hugepages=40
        - hugepagesz=2M hugepages=0
        - default_hugepagesz=1G
        - irqaffinity=${NON-ISOLATED_CPU_CORES}
        - isolcpus=domain,nohz,managed_irq,${ISOLATED_CPU_CORES}
        - nohz_full=${ISOLATED_CPU_CORES}
        - rcu_nocbs=${ISOLATED_CPU_CORES}
        - rcu_nocb_poll
        - nosoftlockup
        - nowatchdog
        - nohz=on
        - nmi_watchdog=0
        - skew_tick=1
        - quiet
    systemd:
      units:
        - name: rke2-preinstall.service
          enabled: true
          contents: |
            [Unit]
            Description=rke2-preinstall
            Wants=network-online.target
            Before=rke2-install.service
            ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
            [Service]
            Type=oneshot
```

```
                User=root
                ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
                ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/$(jq -r .uuid /mnt/
openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
                ExecStart=/bin/sh -c "echo \"node-name: $(jq -r .name /mnt/openstack/
latest/meta_data.json)\" >> /etc/rancher/rke2/config.yaml"
                ExecStartPost=/bin/sh -c "umount /mnt"
                [Install]
                WantedBy=multi-user.target
            - name: cpu-partitioning.service
              enabled: true
              contents: |
                [Unit]
                Description=cpu-partitioning
                Wants=network-online.target
                After=network.target network-online.target
                [Service]
                Type=oneshot
                User=root
                ExecStart=/bin/sh -c "echo isolated_cores=${ISOLATED_CPU_CORES} > /etc/
tuned/cpu-partitioning-variables.conf"
                ExecStartPost=/bin/sh -c "tuned-adm profile cpu-partitioning"
                ExecStartPost=/bin/sh -c "systemctl enable tuned.service"
                [Install]
                WantedBy=multi-user.target
            - name: performance-settings.service
              enabled: true
              contents: |
                [Unit]
                Description=performance-settings
                Wants=network-online.target
                After=network.target network-online.target cpu-partitioning.service
                [Service]
                Type=oneshot
                User=root
                ExecStart=/bin/sh -c "/opt/performance-settings/performance-settings.sh"
                [Install]
                WantedBy=multi-user.target
            - name: sriov-custom-auto-vfs.service
              enabled: true
              contents: |
                [Unit]
                Description=SRIOV Custom Auto VF Creation
                Wants=network-online.target  rke2-server.target
                After=network.target network-online.target rke2-server.target
                [Service]
                User=root
```

```
                Type=forking
                TimeoutStartSec=1800
                ExecStart=/bin/sh -c "while ! /var/lib/rancher/rke2/bin/kubectl --
kubeconfig=/etc/rancher/rke2/rke2.yaml wait --for condition=ready nodes --timeout=30m --
all ; do sleep 10 ; done"
                ExecStartPost=/bin/sh -c "/opt/sriov/sriov-auto-filler.sh"
                RemainAfterExit=yes
                KillMode=process
                [Install]
                WantedBy=multi-user.target
    kubelet:
      extraArgs:
        - provider-id=metal3://BAREMETALHOST_UUID
    nodeName: "localhost.localdomain"
```

43 Ações de ciclo de vida

Esta seção aborda as ações de gerenciamento do ciclo de vida para clusters implantados pelo SUSE Edge for Telco.

43.1 Upgrades de cluster de gerenciamento

O upgrade do cluster de gerenciamento envolve vários componentes. Para ver a lista dos componentes gerais que exigem upgrade, consulte a documentação do cluster de gerenciamento de `dia 2` (*Capítulo 35, Cluster de gerenciamento*).

O procedimento de upgrade dos componentes específicos desta configuração está descrito abaixo.

Fazendo upgrade do Metal^3

Para fazer upgrade do `Metal3`, use o seguinte comando para atualizar o cache do repositório Helm e buscar o gráfico mais recente para instalar o `Metal3` do repositório de gráficos Helm:

```
helm repo update
helm fetch suse-edge/metal3
```

Depois disso, a maneira fácil de fazer upgrade é exportar suas configurações atuais para um arquivo e fazer upgrade da versão do `Metal3` usando o arquivo anterior. Se for necessário alterar alguma coisa na nova versão, edite o arquivo antes do upgrade.

```
helm get values metal3 -n metal3-system -o yaml > metal3-values.yaml
helm upgrade metal3 suse-edge/metal3 \
  --namespace metal3-system \
  -f metal3-values.yaml \
  --version=303.0.7+up0.11.5
```

43.2 Upgrades de cluster downstream

O upgrade de clusters downstream envolve a atualização de vários componentes. As seções a seguir abordam o processo de upgrade de cada um deles.

Fazendo upgrade do sistema operacional

Para este processo, consulte a seguinte referência (*Seção 42.2, "Preparar uma imagem de cluster downstream para cenários conectados"*) para criar uma imagem com uma nova versão do sistema operacional. Com essa imagem gerada pelo `EIB`, a próxima fase de provisionamento usará a nova versão do sistema operacional fornecida. Na etapa seguinte, a nova imagem será usada para fazer upgrade dos nós.

Fazendo upgrade do cluster RKE2

As alterações necessárias para fazer upgrade do cluster `RKE2` usando o fluxo de trabalho automatizado são as seguintes:

- Altere o bloco `RKE2ControlPlane` no `capi-provisioning-example.yaml` mostrado na seguinte seção (*Seção 42.4, "Provisionamento de cluster downstream com provisionamento de rede direcionado (nó único)"* (p 493)):
  - Especifique a `rolloutStrategy` desejada.
  - Altere a versão do cluster `RKE2` para a nova versão, substituindo `${RKE2_NEW_VERSION}`.

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
metadata:
  name: single-node-cluster
  namespace: default
spec:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3MachineTemplate
    name: single-node-cluster-controlplane
  version: ${RKE2_NEW_VERSION}
  replicas: 1
  rolloutStrategy:
    type: "RollingUpdate"
    rollingUpdate:
      maxSurge: 0
  serverConfig:
    cni: cilium
  rolloutStrategy:
    rollingUpdate:
      maxSurge: 0
  registrationMethod: "control-plane-endpoint"
  agentConfig:
    format: ignition
    additionalUserData:
```

```
      config: |
        variant: fcos
        version: 1.4.0
        systemd:
          units:
            - name: rke2-preinstall.service
              enabled: true
              contents: |
                [Unit]
                Description=rke2-preinstall
                Wants=network-online.target
                Before=rke2-install.service
                ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
                [Service]
                Type=oneshot
                User=root
                ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
                ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/$(jq -r .uuid /mnt/
openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
                ExecStart=/bin/sh -c "echo \"node-name: $(jq -r .name /mnt/openstack/
latest/meta_data.json)\" >> /etc/rancher/rke2/config.yaml"
                ExecStartPost=/bin/sh -c "umount /mnt"
                [Install]
                WantedBy=multi-user.target
    kubelet:
      extraArgs:
        - provider-id=metal3://BAREMETALHOST_UUID
    nodeName: "localhost.localdomain"
```

- Altere o bloco `Metal3MachineTemplate` no `capi-provisioning-example.yaml` mostrado na seguinte seção (*Seção 42.4, "Provisionamento de cluster downstream com provisionamento de rede direcionado (nó único)"* (p 493)):
  - Altere o nome e o checksum da imagem para a nova versão gerada na etapa anterior.
  - Adicione a diretiva `nodeReuse` a `true` para evitar a criação de um novo nó.
  - Adicione a diretiva `automatedCleaningMode` a `metadata` para habilitar a limpeza automatizada do nó.

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3MachineTemplate
metadata:
  name: single-node-cluster-controlplane
  namespace: default
spec:
```

```
  nodeReuse: True
  template:
    spec:
      automatedCleaningMode: metadata
      dataTemplate:
        name: single-node-cluster-controlplane-template
      hostSelector:
        matchLabels:
          cluster-role: control-plane
      image:
        checksum: http://imagecache.local:8080/${NEW_IMAGE_GENERATED}.sha256
        checksumType: sha256
        format: raw
        url: http://imagecache.local:8080/${NEW_IMAGE_GENERATED}.raw
```

Depois de fazer essas alterações, o arquivo `capi-provisioning-example.yaml` poderá ser aplicado ao cluster usando o seguinte comando:

```
kubectl apply -f capi-provisioning-example.yaml
```

VIII Solução de problemas

Esta seção orienta no diagnóstico e na solução dos problemas comuns às implantações e operações do SUSE Edge. Ela aborda vários tópicos, incluindo as etapas de solução de problemas específicas do componente, as ferramentas importantes e os locais relevantes dos registros.

44 Princípios gerais da solução de problemas

Antes de entrar nos detalhes dos problemas específicos dos componentes, avalie estes princípios gerais:

- **Consultar os registros**: os registros são a principal fonte de informações. Na maior parte das vezes, os erros são autoexplicativos e contêm dicas sobre o motivo da falha.
- **Verificar os relógios**: as diferenças entre os relógios dos sistemas podem provocar diversos tipos de erros. Assegure a sincronização dos relógios. É possível instruir o EIB a forçar a sincronização dos relógios no momento da inicialização. Consulte Configurando a hora do sistema operacional (*Capítulo 3, Clusters independentes com o Edge Image Builder*).
- **Problemas na inicialização**: se o sistema travar durante a inicialização, anote as últimas mensagens exibidas. Acesse o console (físico ou via BMC) para observar as mensagens de inicialização.
- **Problemas de rede**: verifique a configuração da interface de rede (`ip a`) e a tabela de roteamento (`ip route`), teste a conectividade com outros nós e serviços externos (`ping`, `nc`) e garanta que as regras de firewall não bloqueiem as portas necessárias.
- **Verificar o status dos componentes**: use `kubectl get` e `kubectl describe` para os recursos do Kubernetes. Use `kubectl get events --sort-by='.lastTimestamp' -n <namespace>` para ver os eventos em um determinado namespace do Kubernetes.
- **Verificar o status dos serviços**: use `systemctl status <serviço>` para os serviços systemd.
- **Verificar a sintaxe**: o software espera uma determinada estrutura e sintaxe nos arquivos de configuração. Para os arquivos yaml, por exemplo, use `yamllint` ou ferramentas similares para verificar a sintaxe adequada.
- **Isolar o problema**: tente restringir o problema a um componente ou uma camada específica (por exemplo, rede, armazenamento, sistema operacional, Kubernetes, Metal[3], Ironic etc.).
- **Documentação**: consulte sempre a documentação oficial do SUSE Edge (https://documentation.suse.com/suse-edge/) e também a documentação upstream para obter informações detalhadas.

- **Versões**: o SUSE Edge é uma versão consistente e completamente testada de diferentes componentes SUSE. Consulte as versões de cada componente por lançamento do SUSE Edge na matriz de suporte do SUSE Edge (https://documentation.suse.com/suse-edge/support-matrix/html/support-matrix/index.html).
- **Problemas conhecidos**: para cada versão do SUSE Edge existe uma seção de "problemas conhecidos" nas Notas de lançamento, com informações dos problemas que serão corrigidos em versões futuras, mas que podem afetar a versão atual.

45 Solução de problemas do Kiwi

O Kiwi é usado para gerar as imagens atualizadas do SUSE Linux Micro que serão usadas com o Edge Image Builder.

PROBLEMAS COMUNS

- **Incompatibilidade de versões do SL Micro**: a versão do sistema operacional do host de build deve ser compatível com a versão do sistema operacional que está sendo criada (host do SL Micro 6.0 → imagem do SL Micro 6.0).
- **SELinux no estado de imposição**: devido a algumas limitações, é necessário desabilitar o SELinux temporariamente para criar imagens com o Kiwi. Verifique o status do SElinux com o comando `getenforce` e desabilite-o antes de executar o processo de criação com `setenforce 0`.
- **Host de build não registrado**: o processo de build usa os registros dos hosts de build para obter os pacotes do SUSE SCC. Se o host não foi registrado, o processo falha.
- **Falha no teste do dispositivo em loop**: na primeira execução do processo de criação do Kiwi, ocorre uma falha logo depois que ele é iniciado com o erro: "ERROR: Early loop device test failed, please retry the container run." (ERRO: Falha no teste do dispositivo no ciclo inicial. Tente executar o contêiner novamente.). Trata-se de um sintoma em que os dispositivos que são criados em loop no sistema host subjacente não ficam imediatamente visíveis dentro da imagem do contêiner. Execute novamente o processo de criação do Kiwi, e ele deverá prosseguir sem problemas.
- **Permissões ausentes**: o processo de build espera sua execução como usuário root (ou via sudo).
- **Privilégios incorretos**: o processo de build espera o sinalizador `--privileged` ao executar o contêiner. Certifique-se de que ele esteja presente.

REGISTROS

- **Registros do contêiner de build**: consulte os registros do contêiner de build. Os registros são gerados no diretório que foi usado para armazenar os artefatos. Consulte também os registros do docker ou do podman para ver as informações necessárias.
- **Diretórios temporários de build**: o Kiwi cria diretórios temporários durante o processo de build. Consulte-os para acessar os registros ou os artefatos intermediários se a saída principal não for suficiente.

1. **Revise a saída de** `build-image`: a mensagem de erro na saída do console costuma ser muito clara.
2. **Verifique o ambiente do build**: garanta que todos os pré-requisitos para o próprio Kiwi (por exemplo, docker/podman, SElinux, espaço em disco suficiente) sejam atendidos na máquina que executa o Kiwi.
3. **Analise os registros do contêiner de build**: revise os registros do contêiner com falha para verificar os erros em mais detalhes (veja acima).
4. **Verifique o arquivo de definição**: se você usa um arquivo de definição de imagem do Kiwi personalizado, revise-o para garantir que não haja erros de sintaxe ou ortografia.

Nota

Consulte o Guia de Solução de Problemas do Kiwi (https://documentation.suse.com/appliance/kiwi-9/html/kiwi/troubleshooting.html) (em inglês).

46 Solucionando problemas no Edge Image Builder (EIB)

O EIB é usado para criar imagens personalizadas do SUSE Edge.

PROBLEMAS COMUNS

- **Código incorreto do SCC**: assegure que o código do SCC usado no arquivo de definição do EIB corresponda à versão e arquitetura do SL Micro.
- **Dependências ausentes**: garanta que não falte nenhum pacote ou ferramenta no ambiente do build.
- **Tamanho da imagem incorreto**: para as imagens brutas, o parâmetro `diskSize` é obrigatório e depende significativamente de que as imagens, os RPMs e outros artefatos sejam incluídos na imagem.
- **Permissões**: se você armazena um script no diretório personalizado/de arquivos, assegure que ele tenha permissões de execução, já que esses arquivos apenas estão disponíveis no momento da combustão, mas o EIB não faz nenhuma alteração.
- **Dependências de grupo do sistema operacional**: ao criar uma imagem com usuários e grupos personalizados, os grupos definidos como "`primaryGroup`" devem ser explicitamente criados.
- **As chaves SSH dos usuários do sistema operacional requerem uma pasta pessoal**: ao criar uma imagem com usuários que têm chaves SSH, também é necessário criar a pasta pessoal com `createHomeDir=true`.
- **Problemas no Combustion**: o EIB usa o Combustion para a personalização do sistema operacional e da implantação de todos os outros componentes do SUSE Edge, o que também inclui os scripts personalizados substituídos na pasta custom/scripts. Observe que o processo do Combustion é executado com o `initrd`, portanto, o sistema não é completamente inicializado quando os scripts são executados.
- **Tamanho da máquina podman**: conforme explicado na seção de dicas e truques do EIB (*Parte IV, "Dicas e truques"*), verifique se a máquina podman tem CPU/memória suficiente para executar o contêiner do EIB em sistemas operacionais não Linux.

REGISTROS

- **Saída do EIB**: a saída do console do comando `eib build` é fundamental.

- **Registros do contêiner de build**: consulte os registros do contêiner de build. Eles são gerados no diretório que foi usado para armazenar os artefatos. Consulte também os `docker logs` ou os `podman logs` para obter as informações necessárias.

Nota

Para obter mais informações, consulte Debugging (https://github.com/suse-edge/edge-image-builder/blob/main/docs/debugging.md) (Depuração).

- **Diretórios temporários de build**: o EIB cria diretórios temporários durante o processo de build. Consulte-os para acessar os registros ou os artefatos intermediários se a saída principal não for suficiente.

- **Registros do Combustion**: se a imagem que está sendo criada com o EIB não for inicializada por algum motivo, um shell root estará disponível. Conecte-se ao console do host (fisicamente, via BMC etc.) e consulte os registros do Combustion com `journalctl -u combustion` e, em geral, todos os registros do sistema operacional com `journalctl`, para encontrar a causa raiz da falha.

ETAPAS DA SOLUÇÃO DE PROBLEMAS

1. **Revise a saída de `eib-build`**: a mensagem de erro na saída do console costuma ser bem clara.

2. **Verifique o ambiente do build**: garanta que todos os pré-requisitos para o próprio EIB (por exemplo, docker/podman, espaço em disco suficiente) sejam atendidos na máquina que executa o EIB.

3. **Analise os registros do contêiner de build**: revise os registros do contêiner com falha para verificar os erros em mais detalhes (veja acima).

4. **Verifique a configuração do `eib`**: revise o arquivo de configuração do `eib` para verificar se não há erros de ortografia ou caminhos incorretos para os arquivos de origem ou scripts de build.

- **Teste os componentes individualmente**: se o build do EIB envolver scripts ou estágios personalizados, execute-os separadamente para isolar falhas.

Nota

Consulte Edge Image Builder Debugging (https://github.com/suse-edge/edge-image-builder/blob/main/docs/debugging.md) (Depuração do Edge Image Builder).

47 Solução de problemas da rede de borda (NMC)

O NMC é injetado nas imagens EIB do SL Micro para configurar a rede dos hosts do Edge no momento da inicialização pelo Combustion. Ele também é executado no fluxo de trabalho do Metal3 como parte do processo de inspeção. Os problemas podem ocorrer quando o host é inicializado pela primeira vez ou no processo de inspeção do Metal3.

PROBLEMAS COMUNS

- **O host não é inicializado corretamente na primeira vez**: arquivos de definição de rede mal elaborados podem provocar falha na fase de combustão e, dessa forma, o host perde o shell root.
- **Os arquivos não são gerados de maneira apropriada**: garanta que os arquivos de rede sigam o formato NMState (https://nmstate.io/examples.html).
- **As interfaces de rede não são configuradas corretamente**: garanta que os endereços MAC correspondam às interfaces usadas no host.
- **Incompatibilidade entre nomes de interface**: o argumento do kernel `net.ifnames=1` habilita o Esquema de nomenclatura previsível para interfaces de rede (https://documentation.suse.com/smart/network/html/network-interface-predictable-naming/index.html), portanto, não existe mais o `eth0`, porém, outro esquema de nomenclatura, como o `enp2s0`.

REGISTROS

- **Registros do Combustion**: como o NMC é usado no momento da execução do Combustion, consulte os registros do Combustion com o comando `journalctl -u combustion` no host que está sendo provisionado.
- **Registros do NetworkManager**: no fluxo de trabalho de implantação do Metal^3, o NMC faz parte da execução do IPA e é executado como uma dependência do serviço NetworkManager, usando a funcionalidade ExecStartPre do systemd. Consulte os registros do NetworkManager no host IPA como `journalctl -u NetworkManager` (consulte a seção de solução de problemas de provisionamento de rede direcionado (*Capítulo 49, Solução de problemas de provisionamento de rede direcionado*) para saber como acessar o host quando inicializado com o IPA).

1. **Verifique a sintaxe do yaml**: os arquivos de configuração do NMC são yaml. Verifique a sintaxe adequada com `yamllint` ou ferramentas similares.
2. **Execute o NMC manualmente**: como o NMC faz parte do contêiner EIB, é possível usar o comando podman local para depurar problemas.
   a. Crie uma pasta temporária para armazenar os arquivos nmc.

   ```
   mkdir -p ${HOME}/tmp/foo
   ```

   b. Salve os arquivos nmc nesse local.

   ```
   ❯ tree --noreport ${HOME}/tmp/foo
   /Users/johndoe/tmp/foo
   ├── host1.example.com.yaml
   └── host2.example.com.yaml
   ```

   c. Execute o contêiner EIB com o NMC como ponto de entrada e o comando generate para executar as mesmas tarefas que o NMC faz no momento da execução do Combustion:

   ```
   podman run -it --rm -v ${HOME}/tmp/foo:/tmp/foo:Z --entrypoint=/usr/bin/nmc
    registry.suse.com/edge/3.3/edge-image-builder:1.2.0 generate --config-dir /
   tmp/foo --output-dir /tmp/foo/

   [2025-06-04T11:58:37Z INFO  nmc::generate_conf] Generating config from "/tmp/
   foo/host2.example.com.yaml"...
   [2025-06-04T11:58:37Z INFO  nmc::generate_conf] Generating config from "/tmp/
   foo/host1.example.com.yaml"...
   [2025-06-04T11:58:37Z INFO  nmc] Successfully generated and stored network
    config
   ```

   d. Observe os registros e os arquivos que são gerados na pasta temporária.

48 Solucionando problemas em cenários "phone home"

Os cenários "phone home" envolvem o uso do Elemental para conectar-se com o cluster de gerenciamento e o EIB criar uma imagem de sistema operacional que inclua os bits do elemental-registration. Pode haver problemas quando o host é inicializado pela primeira vez, durante o processo de criação do EIB ou na tentativa de registro no cluster de gerenciamento.

PROBLEMAS COMUNS

- **Falha no registro do sistema**: o nó não é registrado na IU. Assegure que o host seja inicializado de maneira apropriada e possa se comunicar com o Rancher, o relógio esteja sincronizado e os serviços do Elemental estejam ativos.
- **Falha no provisionamento do sistema**: o nó é registrado, mas não é provisionado. Assegure que o host possa se comunicar com o Rancher, o relógio esteja sincronizado e os serviços do Elemental estejam ativos.

REGISTROS

- **Registros do sistema**: `journalctl`
- **Registros do elemental-system-agent**: `journalctl -u elemental-system-agent`
- **Registros do K3s/RKE2**: `journalctl -u k3s ou journalctl -u rke2-server` (ou `rke2-agent`)
- **Pod do operador Elemental**: `kubectl logs -n cattle-elemental-system -l app=elemental-operator`

ETAPAS DA SOLUÇÃO DE PROBLEMAS

1. **Revise os registros**: consulte os registros do pod do operador Elemental para ver se há problemas. Verifique os registros do host se o nó for inicializado.
2. **Verifique o MachineRegistration e o TPM**: por padrão, o TPM é usado para autenticação (https://elemental.docs.rancher.com/authentication/), mas há alternativas para hosts sem o TPM.

49 Solução de problemas de provisionamento de rede direcionado

Os cenários de provisionamento de rede direcionado envolvem o uso de elementos do Metal^3 e da CAPI para provisionar o cluster downstream. O EIB também é incluído para criar uma imagem do sistema operacional. Pode haver problemas quando o host é inicializado pela primeira vez ou durante os processos de inspeção ou provisionamento.

PROBLEMAS COMUNS

- **Firmware antigo**: garanta que qualquer firmware diferente usado nos hosts físicos esteja atualizado. Isso inclui o firmware BMC, já que às vezes o Metal^3 requer um específico/atualizado (https://book.metal3.io/bmo/supported_hardware#redfish-and-its-variants).
- **Falha no provisionamento com erros de SSL**: se o servidor web que envia as imagens usa https, o Metal^3 precisa ser configurado para injetar e confiar no certificado da imagem do IPA. Consulte a pasta do Kubernetes (*Seção 40.3.4, "Pasta kubernetes"*) para saber como incluir um arquivo `ca-additional.crt` no gráfico do Metal^3.
- **Problemas no certificado ao inicializar os hosts com IPA**: alguns fornecedores de servidor verificam a conexão SSL ao anexar imagens ISO de mídia virtual ao BMC, o que pode causar um problema porque os certificados gerados para a implantação do Metal3 são autoassinados. É possível que o host seja inicializado, mas acabe entrando em um shell UEFI. Consulte Desabilitando TLS para anexo de ISO de mídia virtual (*Seção 1.6.2, "Desabilitando TLS para anexo de ISO de mídia virtual"*) para saber como corrigir isso.
- **Referência a nome ou rótulo incorreto**: se o cluster faz referência ao nome ou rótulo incorreto de um nó, ele aparece como implantado, mas o BMH continua como "Disponível". Revise as referências nos objetos envolvidos dos BMHs.
- **Problemas na comunicação com o BMC**: certifique-se de que os pods do Metal^3 executados no cluster de gerenciamento possam acessar o BMC dos hosts que estão sendo provisionados (normalmente, a rede do BMC é muito restrita).
- **Estado do host bare metal incorreto**: o objeto BMH passa por diferentes estados (inspeção, preparação, provisionamento etc.) durante seu ciclo de vida Ciclo de vida da máquina de estado (https://book.metal3.io/bmo/state_machine) (em inglês). Se for detectado um estado incorreto, verifique o campo `status` do objeto BMH, pois ele contém mais informações, como `kubectl get bmh <nome> -o jsonpath='{.status}'| jq`.

- **O host não é provisionado**: em caso de falha no desprovisionamento de um host, é possível tentar removê-lo depois de adicionar a anotação "detached" ao objeto BMH desta forma: `kubectl annotate bmh/<BMH> baremetalhost.metal3.io/detached=""`.
- **Erros na imagem**: verifique se a imagem que está sendo criada com o EIB para o cluster downstream está disponível, tem um checksum apropriado e não é grande demais para descompactar ou para o disco.
- **Incompatibilidade de tamanho de disco**: por padrão, o disco não expande até se preencher por completo. Conforme explicado na seção do script growfs (*Seção 1.3.4.1.2, "Script growfs"*), é necessário incluir um script growfs na imagem que está sendo criada com o EIB para os hosts de cluster downstream.
- **Processo de limpeza travado**: o processo de limpeza é repetido várias vezes. Se a limpeza não for mais possível devido a um problema com o host, desabilite primeiro a limpeza definindo o campo `automatedCleanMode` como `disabled` no objeto BMH.

Atenção

Não é recomendado remover manualmente o finalizador quando o processo de limpeza está falhando ou levando mais tempo do que o desejado. Se você fizer isso, o registro do host será removido do Kubernetes, mas ficará no Ironic. A ação executada no momento continua em segundo plano, e a tentativa de adicionar o host novamente pode falhar devido ao conflito.

- **Problemas nos pods do Metal3/Rancher Turtles/CAPI**: o fluxo de implantação de todos os componentes obrigatórios é:
  - O controlador Rancher Turtles implanta o controlador do operador CAPI.
  - O controlador do operador CAPI depois implanta os controladores do provedor (núcleo CAPI, CAPM3 e plano de controle/inicialização RKE2).

Verifique se todos os pods estão sendo executados corretamente e consulte os registos caso não estejam.

REGISTROS

- **Registros do Metal^3**: consulte os registros dos diferentes pods.

```
kubectl logs -n metal3-system -l app.kubernetes.io/component=baremetal-operator
```

```
kubectl logs -n metal3-system -l app.kubernetes.io/component=ironic
```

Nota

O pod metal3-ironic contém pelo menos 4 contêineres diferentes (`ironic-httpd`,` ironic-log-watch`, `ironic` & `ironic-ipa-downloader` (init)) no mesmo pod. Use o sinalizador `-c` ao usar o comando `kubectl logs` para verificar os registros de cada um dos contêineres.

Nota

O contêiner `ironic-log-watch` expõe os registros do console dos hosts após a inspeção/provisionamento, desde que a conectividade de rede permita o envio desses registros de volta ao cluster de gerenciamento. Isso pode ser útil quando há erros de provisionamento, mas você não tem acesso direto aos registros do console do BMC.

- **Registros do Rancher Turtles**: consulte os registros dos diferentes pods.

```
kubectl logs -n rancher-turtles-system -l control-plane=controller-manager
kubectl logs -n rancher-turtles-system -l app.kubernetes.io/name=cluster-api-
operator
kubectl logs -n rke2-bootstrap-system -l cluster.x-k8s.io/provider=bootstrap-rke2
kubectl logs -n rke2-control-plane-system -l cluster.x-k8s.io/provider=control-
plane-rke2
kubectl logs -n capi-system -l cluster.x-k8s.io/provider=cluster-api
kubectl logs -n capm3-system -l cluster.x-k8s.io/provider=infrastructure-metal3
```

- **Registros do BMC**: em geral, os BMCs têm uma IU em que é possível realizar grande parte da interação. Normalmente, há uma seção "registros" na qual observar possíveis problemas (imagem inacessível, falhas de hardware etc.).

- **Registros do console**: conecte-se ao console BMC (pela IU da web do BMC, série etc.) e verifique se há erros nos registros gravados.

1. **Verifique o status de** `BareMetalHost`:
    - `kubectl get bmh -A` mostra o estado atual. Procure por `provisioning`, `ready`, `error`, `registering`.
    - `kubectl describe bmh -n <namespace> <nome_do_bmh>` retorna os eventos detalhados e as condições que explicam o motivo da paralisação do BMH.
2. **Teste a conectividade do RedFish**:
    - Use `curl` no plano de controle do Metal^3 para testar a conectividade com os BMCs pelo Redfish.
    - Garanta que as credenciais corretas do BMC sejam fornecidas na definição de `BareMetalHost-Secret`.
3. **Verifique o status dos pods turtles/CAPI/metal3**: garanta que os contêineres no cluster de gerenciamento estejam em funcionamento: `kubectl get pods -n metal3-system` e `kubectl get pods -n rancher-turtles-system` (veja também `capi-system`, `capm3-system`, `rke2-bootstrap-system` e `rke2-control-plane-system`).
4. **Verifique se o endpoint do Ironic é acessível ao host provisionado**: o host provisionado precisa acessar o endpoint do Ironic para retornar os relatórios ao Metal^3. Verifique o IP com `kubectl get svc -n metal3-system metal3-metal3-ironic` e tente acessá-lo via `curl/nc`.
5. **Verifique se a imagem do IPA é acessível ao BMC**: o IPA é fornecido pelo endpoint do Ironic e precisa ser acessível ao BMC porque é usado como CD virtual.
6. **Verifique se a imagem do sistema operacional é acessível ao host provisionado**: a imagem usada para provisionar o host precisa ser acessível ao próprio host (ao executar o IPA) já que ela será baixada temporariamente e gravada no disco.
7. **Examine os registros do componente Metal^3**: veja acima.

8. **Acione novamente a inspeção do BMH**: se houver falha em uma inspeção ou se o hardware de um host disponível for modificado, um novo processo de inspeção poderá ser acionado anotando o objeto BMH com `inspect.metal3.io: ""`. Consulte o guia Metal[3] Controlling inspection (https://book.metal3.io/bmo/inspect_annotation) (Inspeção de controle do Metal[3]) para obter mais informações.

9. **Console do IPA bare metal**: para solucionar problemas do IPA, há algumas alternativas:

   - Habilite o "login automático". Isso habilita o login automático do usuário root ao se conectar ao console do IPA.

     Atenção

     Isso é apenas para fins de depuração, já que concede acesso total ao host.

     Para habilitar o login automático, o valor `global.ironicKernelParams` do Helm do Metal3 deve ser assim: `console=ttyS0 suse.autologin=ttyS0` (dependendo do console, `ttyS0` pode ser alterado). Em seguida, é necessário reimplantar o gráfico do Metal[3]. (Observe que `ttyS0` é um exemplo, ele deve corresponder ao terminal real, por exemplo, em muitos casos, pode ser `tty1` em bare metal. Para confirmar isso, observe a saída do console do ramdisk IPA na inicialização, em que `/etc/issue` mostra o nome do console).
     Outra maneira de fazer isso é alterar o parâmetro `IRONIC_KERNEL_PARAMS` no mapa de configuração `ironic-bmo` do namespace `metal3-system`. Isso pode ser mais fácil porque é feito pela edição de `kubectl`, mas será substituído ao atualizar o gráfico. Depois disso, o pod do Metal[3] precisará ser reiniciado com `kubectl delete pod -n metal3-system -l app.kubernetes.io/component=ironic`.

   - Injete uma chave SSH para o usuário root no IPA.

     Atenção

     Isso é apenas para fins de depuração, já que concede acesso total ao host.

     Para injetar a chave SSH para o usuário root, o valor `debug.ironicRamdiskSshKey` do Helm do Metal[3] deve ser usado. Em seguida, reimplante o gráfico do Metal[3].

Outra maneira de fazer isso é alterar o parâmetro `IRONIC_RAMDISK_SSH_KEY` no `ironic-bmo configmap` do namespace `metal3-system`. Isso pode ser mais fácil porque é feito pela edição de `kubectl`, mas será substituído durante a atualização do gráfico. Em seguida, o pod do Metal[3] precisa ser reiniciado com `kubectl delete pod -n metal3-system -l app.kubernetes.io/component=ironic`

Nota

Consulte os guias de solução de problemas da CAPI (https://cluster-api.sigs.k8s.io/user/troubleshooting) e de solução de problemas do Metal[3] (https://book.metal3.io/troubleshooting).

50 Solução de problemas de outros componentes

Consulte os guias de solução de problemas de outros componentes do SUSE Edge na respectiva documentação oficial:

- Solução de problemas do SUSE Linux Micro (https://documentation.suse.com/smart/micro-clouds/html/SLE-Micro-5.5-admin/index.html#id-1.10)
- Problemas conhecidos do RKE2 (https://docs.rke2.io/known_issues)
- Problemas conhecidos do K3s (https://docs.k3s.io/known-issues)
- Solução de problemas gerais do Rancher (https://ranchermanager.docs.rancher.com/troubleshooting/general-troubleshooting)
- Solução de problemas do SUSE Multi-Linux Manager (https://documentation.suse.com/multi-linux-manager/5.1/en/docs/administration/troubleshooting/tshoot-intro.html)
- Suporte do Elemental (https://elemental.docs.rancher.com/troubleshooting-support/)
- Solução de problemas do Rancher Turtles (https://turtles.docs.rancher.com/turtles/stable/en/troubleshooting/troubleshooting.html)
- Solução de problemas do Longhorn (https://longhorn.io/docs/1.8.1/troubleshoot/troubleshooting/)
- Solução de problemas do Neuvector (https://open-docs.neuvector.com/next/troubleshooting/troubleshooting/)
- Solução de problemas do Fleet (https://fleet.rancher.io/troubleshooting)

Você também pode consultar o SUSE Knowledgebase (https://www.suse.com/support/kb/).

51 Coletando diagnósticos para o suporte

Ao contatar o suporte da SUSE, é essencial passar todas as informações de diagnóstico.

INFORMAÇÕES ESSENCIAIS PARA COLETAR

- **Descrição detalhada do problema**: o que aconteceu, quando aconteceu, o que você estava fazendo, qual era o comportamento esperado e qual foi o comportamento real?
- **Etapas de reprodução**: você pode reproduzir o problema de maneira confiável? Se puder, faça uma lista das etapas exatas.
- **Versões dos componentes**: as versões do SUSE Edge e dos componentes (RKE2/K3, EIB, Metal^3, Elemental etc.).
- **Registros relevantes**:
  - Saída do `journalctl` (filtrada por serviço, se possível, ou os registros completos de inicialização)
  - Registros do pod Kubernetes (registros do kubectl)
  - Registros do componente Metal^3/Elemental
  - Registros do build do EIB e outros registros
- **Informações do sistema**:
  - `uname -a`
  - `df -h`
  - `ip a`
  - `/etc/os-release`
- **Arquivos de configuração**: arquivos de configuração relevantes para Elemental, Metal^3 e EIB, como valores de gráficos Helm, mapas de configuração etc.
- **Informações do Kubernetes**: nós, serviços, implantações etc.
- **Objetos do Kubernetes afetados**: BMH, MachineRegistration etc.

COMO FAZER A COLETA

- **Dos registros**: redirecione a saída do comando para os arquivos (por exemplo, `journalctl -u k3s > k3s_logs.txt`).

- **Dos recursos do Kubernetes**: use `kubectl get <recurso> -o yaml > <nome_do_recurso>.yaml` para obter as definições detalhadas do YAML.
- **Das informações do sistema**: colete a saída dos comandos listados acima.
- **Do SL Micro**: consulte a documentação do Guia de Solução de Problemas do SUSE Linux Micro (https://documentation.suse.com/sle-micro/5.5/html/SLE-Micro-all/cha-adm-support-slemicro.html) sobre como reunir as informações do sistema para o suporte com `supportconfig`.
- **Do RKE2/Rancher**: consulte o artigo The Rancher v2.x Linux log collector script (https://www.suse.com/support/kb/doc/?id=000020191) para executar o script do coletor de registros do Linux para Rancher v2.x.

Contatar o suporte. Leia o artigo disponível em How-to effectively work with SUSE Technical Support (https://www.suse.com/support/kb/doc/?id=000019452) (Como trabalhar de maneira eficiente com o suporte técnico da SUSE) e o manual de suporte localizado em SUSE Technical Support Handbook (https://www.suse.com/support/handbook/) (Manual de Suporte Técnico da SUSE) para obter mais detalhes sobre como contatar o suporte da SUSE.

IX Apêndice

52 Notas de lançamento

52.1 Resumo

O SUSE Edge 3.3 é uma solução completa, de estreita integração e amplamente validada, capaz de abordar os desafios exclusivos da implantação da infraestrutura e dos aplicativos nativos de nuvem na borda. Seu foco principal é oferecer uma plataforma consistente, porém altamente flexível, escalável e segura, que envolva a criação de imagem da implantação inicial, o provisionamento e a integração de nós, a implantação de aplicativos, a observabilidade e o gerenciamento do ciclo de vida.

A solução foi criada com a ideia de que não existe uma plataforma de borda do tipo "tamanho único" porque os requisitos e as expectativas dos clientes variam de maneira significativa. As implantações de borda nos levam a resolver (e sempre aprimorar) alguns dos problemas mais desafiadores, como escalabilidade massiva, disponibilidade de rede restrita, limitações de espaço físico, novas ameaças à segurança e vetores de ataque, variações na arquitetura de hardware e nos recursos de sistema, a necessidade de implantar e estabelecer interface com infraestruturas e aplicativos legados e soluções de clientes com durações estendidas.

O SUSE Edge foi criado do zero, com base no melhor software de código aberto, e condiz com os nossos 30 anos de história como provedores de plataformas SUSE Linux seguras, estáveis e certificadas e com a nossa experiência em oferecer gerenciamento Kubernetes altamente escalável e repleto de recursos com o nosso portfólio Rancher. O SUSE Edge é fundamentado nesses recursos para entregar funcionalidades que atendem a inúmeros segmentos de mercado, como varejo, medicina, transporte, logística, telecomunicações, manufatura inteligente e IoT industrial.

Nota

O SUSE Edge for Telco (antes conhecido como Adaptive Telco Infrastructure Platform/ ATIP) deriva (produto downstream) do SUSE Edge, com otimizações e componentes adicionais que possibilitam que a plataforma atenda aos requisitos constantes nos casos de uso de telecomunicações. A menos que claramente indicado, todas as Notas de lançamento são aplicáveis ao SUSE Edge 3.3 e ao SUSE Edge for Telco 3.3.

52.2 Sobre

As Notas de lançamento, a menos que claramente especificado e explicado, são idênticas em todas as arquiteturas; e a versão mais recente, junto com as Notas de lançamento de todos os outros produtos SUSE, está sempre disponível online em https://www.suse.com/releasenotes.

As entradas são listadas apenas uma vez, mas é possível fazer referência a elas em vários locais, se forem importantes e pertencerem a mais de uma seção. Normalmente, as Notas de lançamento listam somente as alterações feitas entre dois lançamentos subsequentes. Determinadas entradas importantes das Notas de lançamento de versões anteriores dos produtos podem se repetir. Para facilitar a identificação dessas entradas, elas incluem uma observação a esse respeito.

No entanto, as entradas repetidas são fornecidas apenas como conveniência. Portanto, se você pular um ou mais lançamentos, consulte as Notas de lançamento também dos lançamentos que forem pulados. Se você estiver apenas lendo as Notas de lançamento referentes ao lançamento atual, poderá perder alterações importantes que podem afetar o comportamento do sistema. As versões do SUSE Edge são definidas como x.y.z, em que "x" indica a versão principal, "y" indica a versão secundária e "z" indica a versão do patch, também conhecida como "z-stream". Os ciclos de vida do produto SUSE Edge são definidos com base na versão secundária especificada, por exemplo, "3.3", mas acompanham as atualizações de patch subsequentes ao longo do seu ciclo de vida, por exemplo, "3.3.1".

Nota

As versões z-stream do SUSE Edge são estreitamente integradas e foram testadas na íntegra como uma pilha com controle de versão. O upgrade de qualquer componente individual para versões diferentes das que estão listadas acima pode resultar em tempo de inatividade do sistema. É possível executar clusters Edge em configurações não testadas, mas não é recomendado e pode levar mais tempo para conseguir uma resolução por meio dos canais de suporte.

52.3 Versão 3.3.1

Data da disponibilidade: 13 de junho de 2025

Resumo: SUSE Edge 3.3.1 é a primeira versão z-stream no fluxo de versões do SUSE Edge 3.3.

52.3.1 Novos recursos

- Atualizado para o Kubernetes 1.32.4 e o Rancher Prime 2.11.2 Notas de lançamento (https://github.com/rancher/rancher/releases/tag/v2.11.2)
- Atualizado para o SUSE Security (Neuvector) 5.4.4 Notas de lançamento (https://open-docs.neuvector.com/releasenotes/5x#544-may-2025)
- Atualizado para o Rancher Turtles 0.20.0 Notas de lançamento (https://turtles.docs.rancher.com/turtles/stable/en/changelogs/changelogs/v0.20.0.html)

52.3.2 Correções de bugs e de segurança

- Em alguns casos, ao usar um cluster de gerenciamento que foi atualizado do Edge 3.2 para 3.3.0, os upgrades downstream contínuos feitos via CAPI podem fazer com que as máquinas fiquem travadas no estado "em exclusão". Isso foi resolvido com a atualização para o provedor CAPI RKE2 (Upstream RKE2 provider issue 661 (https://github.com/rancher/cluster-api-provider-rke2/issues/661))
- Ao configurar a rede pelo nm-configurator, algumas configurações que identificam as interfaces por MAC na versão 3.3.0 não funcionavam, o que foi resolvido ao atualizar o NMC para 0.3.3, com as atualizações correspondentes nas imagens do contêiner do mecanismo de download IPA do EIB e do Metal^3 Problema com o NM Configurator upstream (https://github.com/suse-edge/nm-configurator/issues/163)
- Nos clusters de gerenciamento Metal^3 de longa execução na versão 3.3.0, o vencimento do certificado poderia provocar falha na conexão do operador bare metal com o Ironic, exigindo como solução alternativa a reinicialização manual do pod, o que foi resolvido por meio de atualizações no gráfico do Metal^3 Problema nos gráficos do SUSE Edge (https://github.com/suse-edge/charts/issues/178)
- Antes, a IU do Rancher não listava os gráficos do SUSE Edge do registro OCI no catálogo de aplicativos. Isso foi resolvido com a atualização para o Rancher 2.11.2 Problema no Rancher (https://github.com/rancher/rancher/issues/48746)

52.3.3 Problemas conhecidos

Atenção

Se você está implantando novos clusters, siga o *Capítulo 28, Criando imagens atualizadas do SUSE Linux Micro com o Kiwi* para primeiro criar as novas imagens, já que esta é a primeira etapa necessária para criar clusters para ambas as arquiteturas AMD64/Intel 64 e AArch64, além dos clusters de gerenciamento e downstream.

- Ao usar o `toolbox` no SUSE Linux Micro 6.1, a imagem do contêiner padrão não inclui algumas ferramentas que eram incluídas na versão 5.5 anterior. A solução alternativa é configurar o toolbox para usar a imagem do contêiner `suse/sle-micro/5.5/toolbox` anterior. Consulte `toolbox --help` para ver opções de configuração da imagem.

- Em alguns casos, os upgrades contínuos feitos via CAPI podem fazer com que as máquinas fiquem travadas no estado "em exclusão". Isso será resolvido com uma atualização futura (Upstream RKE2 provider issue 655 (https://github.com/rancher/cluster-api-provider-rke2/issues/655))

- Devido a correções relacionadas ao CVE-2025-1974 (https://nvd.nist.gov/vuln/detail/CVE-2025-1974), conforme mencionado na versão 3.3.0, o SUSE Linux Micro 6.1 **deve** ser atualizado para incluir o kernel `>=6.4.0-26-default` ou `>=6.4.0-30-rt` (kernel real-time) por causa dos patches de kernel obrigatórios do SELinux. Se não forem aplicados, o pod ingress-nginx continuará no estado `CrashLoopBackOff`. Para aplicar a atualização do kernel, execute `transactional-update` no próprio host (para atualizar todos os pacotes), ou `transactional-update pkg update kernel-default` (ou kernel-rt) para atualizar apenas o kernel e, em seguida, reinicialize o host. Se estiver implantando novos clusters, siga o *Capítulo 28, Criando imagens atualizadas do SUSE Linux Micro com o Kiwi* para criar novas imagens com o kernel mais recente.

- Um bug no controlador de jobs do Kubernetes foi identificado em determinadas condições nas quais ele pode fazer com que os nós do RKE2/K3s permaneçam no estado `NotReady` (consulte o problema #8357 do RKE2 (https://github.com/rancher/rke2/issues/8357)). Os erros podem ter esta aparência:

```
E0605 23:11:18.489721   »···1 job_controller.go:631] "Unhandled Error" err="syncing
 job: tracking status: adding uncounted pods to status: Operation cannot be
 fulfilled on jobs.batch \"helm-install-rke2-ingress-nginx\": StorageError: invalid
```

```
 object, Code: 4, Key: /registry/jobs/kube-system/helm-install-rke2-ingress-nginx,
 ResourceVersion: 0, AdditionalErrorMsg: Precondition failed: UID in precondition:
 0aa6a781-7757-4c61-881a-cb1a4e47802c, UID in object meta: 6a320146-16b8-4f83-88c5-
fc8b5a59a581" logger="UnhandledError"
```

Como solução alternativa, reinicie o pod `kube-controller-manager` com `crictl` como:

```
export CONTAINER_RUNTIME_ENDPOINT=unix:///run/k3s/containerd/containerd.sock
export KUBEMANAGER_POD=$(/var/lib/rancher/rke2/bin/crictl ps --label
 io.kubernetes.container.name=kube-controller-manager --quiet)
/var/lib/rancher/rke2/bin/crictl stop ${KUBEMANAGER_POD} && \
/var/lib/rancher/rke2/bin/crictl rm ${KUBEMANAGER_POD}
```

- Nas versões do RKE2/K3s 1.31 e 1.32, o diretório `/etc/cni` usado para armazenar as configurações de CNI podem não acionar uma notificação sobre os arquivos que estão sendo gravados nele para o `containerd` devido a determinadas condições relacionadas ao `overlayfs` (consulte o problema #8356 do RKE2 (https://github.com/rancher/rke2/issues/8356)). Isso, por sua vez, acaba travando a implantação do RKE2/K3s na espera pela inicialização da CNI, e os nós do RKE2/K3s no estado `NotReady`. Isso pode ser observado no nível do nó com `kubectl describe node <nó_afetado>`:

```
<200b><200b>Conditions:
  Type         »Status  LastHeartbeatTime             »·LastTransitionTime
 »·Reason                   »··Message
  ----         »------  -----------------             »·------------------
 »·------                   »··--------
  Ready        »False   Thu, 05 Jun 2025 17:41:28 +0000   Thu, 05 Jun 2025 14:38:16 +0000
   KubeletNotReady          »··container runtime network not ready: NetworkReady=false
 reason:NetworkPluginNotReady message:Network plugin returns error: cni plugin not
 initialized
```

Como solução alternativa, é possível montar um volume tmpfs no diretório `/etc/cni` antes que o RKE2 seja iniciado. Isso evita o uso do overlayfs, que faz com que o containerd perca notificações e que as configurações sejam regravadas sempre que o nó é reiniciado e os initcontainers dos pods são novamente executados. Se você usa o EIB, isso pode ser um script `04-tmpfs-cni.sh` no diretório `custom/scripts` (conforme explicado aqui [https://github.com/suse-edge/edge-image-builder/blob/release-1.2/docs/building-images.md#custom]) com esta aparência:

```
#!/bin/bash
mkdir -p /etc/cni
mount -t tmpfs -o mode=0700,size=5M tmpfs /etc/cni
echo "tmpfs /etc/cni tmpfs defaults,size=5M,mode=0700 0 0" >> /etc/fstab
```

52.3.4 Versões dos componentes

A tabela a seguir descreve os componentes individuais que compõem o lançamento 3.3.1, incluindo a versão, a versão do gráfico Helm (se aplicável) e de onde obter o artefato lançado no formato binário. Consulte a documentação associada para ver exemplos de uso e implantação.

Nome	Versão	Versão do gráfico Helm	Local do artefato (URL/imagem)
SUSE Linux Micro	6.1 (mais recente)	N/A	Página de downloads do SUSE Linux Micro (https://www.suse.com/download/sle-micro/) SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso (sha256 70b9be28f2d92bc3b228412e4fc2b1 SL-Micro.x86_64-6.1-Base-RT-SelfInstall-GM.install.iso (sha256 9ce83e4545d4b36c7c6a44f7841dc3 SL-Micro.x86_64-6.1-Base-GM.raw.xz (sha256 36e3efa55822113840dd76fdf6914e SL-Micro.x86_64-6.1-Base-RT-GM.raw.xz (sha256 2ee66735da3e1da107b4878e73ae6
SUSE Multi-Linux Manager	5.0.3	N/A	Página de downloads do SUSE Multi-Linux Manager (https://

			www.suse.com/download/suse-manager/)
K3s	**1.32.4**	N/A	Versão upstream do K3s (https://github.com/k3s-io/k3s/releases/tag/v1.32.4%2Bk3s1)
RKE2	**1.32.4**	N/A	Versão upstream do RKE2 (https://github.com/rancher/rke2/releases/tag/v1.32.4%2Brke2r1)
SUSE Rancher Prime	**2.11.2**	2.11.2	Repositório Helm do Rancher Prime (https://charts.rancher.com/server-charts/prime/index.yaml) Imagens do contêiner do Rancher 2.11.1 (https://github.com/rancher/rancher/releases/download/v2.11.1/rancher-images.txt)
SUSE Storage	1.8.1	106.2.0 + up1.8.1	Repositório de gráficos Helm do Rancher (https://charts.rancher.io/index.yaml)

			registry.suse.com/rancher/mirrored-longhornio-csi-attacher:v4.8.1 registry.suse.com/rancher/mirrored-longhornio-csi-provisioner:v5.2.0 registry.suse.com/rancher/mirrored-longhornio-csi-resizer:v1.13.2 registry.suse.com/rancher/mirrored-longhornio-csi-snapshotter:v8.2.0 registry.suse.com/rancher/mirrored-longhornio-csi-node-driver-registrar:v2.13.0 registry.suse.com/rancher/mirrored-longhornio-livenessprobe:v2.15.0 registry.suse.com/rancher/mirrored-longhornio-backing-image-manager:v1.8.1 registry.suse.com/rancher/mirrored-longhornio-longhorn-engine:v1.8.1

			registry.suse.com/rancher/mirrored-longhornio-longhorn-instance-manager:v1.8.1 registry.suse.com/rancher/mirrored-longhornio-longhorn-manager:v1.8.1 registry.suse.com/rancher/mirrored-longhornio-longhorn-share-manager:v1.8.1 registry.suse.com/rancher/mirrored-longhornio-longhorn-ui:v1.8.1 registry.suse.com/rancher/mirrored-longhornio-support-bundle-kit:v0.0.52 registry.suse.com/rancher/mirrored-longhornio-longhorn-cli:v1.8.1
SUSE Security	**5.4.4**	**106.0.1 + up2.8.6**	Repositório de gráficos Helm do Rancher (https://charts.rancher.io/index.yaml) **registry.suse.com/rancher/neuvector-controller:5.4.4**

			registry.suse.com/rancher/neuvector-enforcer:5.4.4 **registry.suse.com/rancher/neuvector-manager:5.4.4** **registry.suse.com/rancher/neuvector-compliance-config:1.0.5** registry.suse.com/rancher/neuvector-registry-adapter:0.1.6 registry.suse.com/rancher/neuvector-scanner:6 **registry.suse.com/rancher/neuvector-updater:0.0.3**
Rancher Turtles (CAPI)	0.20.0	303.0.4 + up0.20.0	**registry.suse.com/edge/charts/rancher-turtles:303.0.3_up0.20.0** **registry.rancher.com/rancher/rancher/turtles:v0.20.0** registry.rancher.com/rancher/cluster-api-operator:v0.17.0 registry.rancher.com/rancher/cluster-api-metal3-controller:v1.9.3

			registry.rancher.com/rancher/cluster-api-metal3-ipam-controller:v1.9.4 registry.suse.com/rancher/cluster-api-controller:v1.9.5 **registry.suse.com/rancher/cluster-api-provider-rke2-bootstrap:v0.16.1 registry.suse.com/rancher/cluster-api-provider-rke2-controlplane:v0.16.1**
Recursos de isolamento do Rancher Turtles	**0.20.0**	**303.0.4 + up0.20.0**	**registry.suse.com/edge/charts/rancher-turtles-airgap-resources:303.0.3_up0.20.0**
Metal[3]	**0.11.5**	**303.0.7 + up0.11.5**	**registry.suse.com/edge/charts/metal3:303.0.7_up0.11.5 registry.suse.com/edge/3.3/baremetal-operator:0.9.1.1** registry.suse.com/edge/3.3/ironic:26.1.2.4 **registry.suse.com/edge/3.3/ironic-ipa-downloader:3.0.7**

			registry.suse.com/edge/mariadb:10.6.15.1
MetalLB	0.14.9	303.0.0+up0.14.9	registry.suse.com/edge/charts/metallb:303.0.0_up0.14.9 registry.suse.com/edge/3.3/metallb-controller:v0.14.8 registry.suse.com/edge/3.3/metallb-speaker:v0.14.8 registry.suse.com/edge/3.3/frr:8.4 registry.suse.com/edge/3.3/frr-k8s:v0.0.14
Elemental	1.6.8	1.6.8	registry.suse.com/rancher/elemental-operator-chart:1.6.8 registry.suse.com/rancher/elemental-operator-crds-chart:1.6.8 registry.suse.com/rancher/elemental-operator:1.6.8
Extensão de dashboard do Elemental	3.0.1	3.0.1	Gráfico Helm da extensão do Elemental (https://github.com/rancher/ui-plugin-charts/tree/4.0.0/charts/elemental/3.0.1)

Edge Image Builder	**1.2.1**	N/A	**registry.suse.com/edge/3.3/edge-image-builder:1.2.1**
NM Configurator	**0.3.3**	N/A	Versão upstream do NMConfigurator (https://github.com/suse-edge/nm-configurator/releases/tag/v0.3.3)
KubeVirt	1.4.0	303.0.0 + up0.5.0	registry.suse.com/edge/charts/kubevirt:303.0.0_up0.5.0 registry.suse.com/suse/sles/15.6/virt-operator:1.4.0 registry.suse.com/suse/sles/15.6/virt-api:1.4.0 registry.suse.com/suse/sles/15.6/virt-controller:1.4.0 registry.suse.com/suse/sles/15.6/virt-exportproxy:1.4.0 registry.suse.com/suse/sles/15.6/virt-exportserver:1.4.0 registry.suse.com/suse/sles/15.6/virt-handler:1.4.0 registry.suse.com/suse/sles/15.6/virt-launcher:1.4.0

Extensão de dashboard KubeVirt	1.3.2	303.0.2+up1.3.2	registry.suse.com/edge/charts/kubevirt-dashboard-extension:303.0.2_up1.3.2
Containerized Data Importer	1.61.0	303.0.0+up0.5.0	registry.suse.com/edge/charts/cdi:303.0.0_up0.5.0 registry.suse.com/suse/sles/15.6/cdi-operator:1.61.0 registry.suse.com/suse/sles/15.6/cdi-controller:1.61.0 registry.suse.com/suse/sles/15.6/cdi-importer:1.61.0 registry.suse.com/suse/sles/15.6/cdi-cloner:1.61.0 registry.suse.com/suse/sles/15.6/cdi-apiserver:1.61.0 registry.suse.com/suse/sles/15.6/cdi-uploadserver:1.61.0 registry.suse.com/suse/sles/15.6/cdi-uploadproxy:1.61.0
Endpoint Copier Operator	0.2.0	303.0.0+up0.2.1	registry.suse.com/edge/charts/endpoint-copier-operator:303.0.0_up0.2.1

			registry.suse.com/edge/3.3/endpoint-copier-operator:0.2.0
Akri (prévia de tecnologia)	0.12.20	303.0.0 + up0.12.20	registry.suse.com/edge/charts/akri:303.0.0_up0.12.20 registry.suse.com/edge/charts/akri-dashboard-extension:303.0.0_up1.3.1 registry.suse.com/edge/3.3/akri-agent:v0.12.20 registry.suse.com/edge/3.3/akri-controller:v0.12.20 registry.suse.com/edge/3.3/akri-debug-echo-discovery-handler:v0.12.20 registry.suse.com/edge/3.3/akri-onvif-discovery-handler:v0.12.20 registry.suse.com/edge/3.3/akri-opcua-discovery-handler:v0.12.20 registry.suse.com/edge/3.3/akri-udev-discovery-handler:v0.12.20

			registry.suse.com/edge/3.3/akri-webhook-configuration:v0.12.20
SR-IOV Network Operator	1.5.0	303.0.2+up1.5.0	registry.suse.com/edge/charts/sriov-network-operator:303.0.2_up1.5.0 registry.suse.com/edge/charts/sriov-crd:303.0.2_up1.5.0
System Upgrade Controller	0.15.2	106.0.0	Repositório de gráficos Helm do Rancher (https://charts.rancher.io/index.yaml) registry.suse.com/rancher/system-upgrade-controller:v0.15.2
Controller de upgrade	**0.1.1**	**303.0.1+up0.1.1**	**registry.suse.com/edge/charts/upgrade-controller:303.0.1_up0.1.1** registry.suse.com/edge/3.3/upgrade-controller:0.1.1 **registry.suse.com/edge/3.3/kubectl:1.32.4** **registry.suse.com/edge/3.3/release-manifest:3.3.1**

Construtor Kiwi	10.2.12.0	N/A	registry.suse.com/edge/3.3/kiwi-builder:10.2.12.0

52.4 Versão 3.3.0

Data da disponibilidade: 20 de maio de 2025

Resumo: SUSE Edge 3.3.0 é a primeira versão no fluxo de versões do SUSE Edge 3.3.

52.4.1 Novos recursos

- Atualizado para o Kubernetes 1.32 e o Rancher Prime 2.11
- Sistema operacional atualizado para o SUSE Linux Micro 6.1 (https://documentation.suse.com/sle-micro/6.1)
- Versões atualizadas do Rancher Turtles, da Cluster API e do Metal3/Ironic
- Agora uma imagem do contêiner é fornecida para criar imagens atualizadas do SUSE Linux Micro. Consulte o *Capítulo 28, Criando imagens atualizadas do SUSE Linux Micro com o Kiwi* para obter mais detalhes.
- Agora a implantação de clusters downstream AArch64 é possível pelo fluxo de provisionamento de rede direcionado. Consulte o *Capítulo 42, Provisionamento de rede direcionado totalmente automatizado* para obter mais detalhes.
- Agora a implantação de clusters downstream de pilha dupla é possível por meio do fluxo de provisionamento de rede direcionado como prévia de tecnologia.
- Agora a configuração do Precision Time Protocol (PTP) é possível como prévia de tecnologia. Consulte a *Seção 52.5, "Prévias de tecnologia"* para obter mais detalhes.

52.4.2 Correções de bugs e de segurança

- Endereços CVE-2025-1974 (https://nvd.nist.gov/vuln/detail/CVE-2025-1974) com patches para ingress-nginx no RKE2. Há mais informações disponíveis aqui (https://kubernetes.io/blog/2025/03/24/ingress-nginx-cve-2025-1974/).
- SUSE Storage (Longhorn) 1.8.1 com várias correções de bugs, incluindo:
  - Correção do problema de volume FailedMount que pode provocar falha no anexo de volume Problema do Longhorn upstream (https://github.com/longhorn/longhorn/issues/9939)
  - Correção do problema do mecanismo travado no estado parado que pode impedir o anexo de volume Problema do Longhorn upstream (https://github.com/longhorn/longhorn/issues/9938)
- A atualização do gráfico do Metal[3] contém várias correções de bugs, incluindo:
  - Foi resolvido um bug ao implantar clusters com IPs estáticos nas redes com servidores DHCP Problema upstream (https://github.com/suse-edge/charts/pull/196)
- O gráfico do MetalLB contém uma correção para garantir o uso das imagens downstream quando frr-k8s estiver habilitado
- O construtor Kiwi foi atualizado para a versão 10.2.12 para se adequar às alterações de segurança recentes no Kiwi, mudando dos métodos de checksum md5 para sha256 (https://github.com/OSInside/kiwi/commit/d4d39e481aaff8be28337a9c76c3913a8a482628) para verificação de imagens.
- A imagem do Edge Image Builder foi recriada para incluir a versão atualizada do MetalLB e abordar as alterações do Kiwi, ambas mencionadas acima.

52.4.3 Problemas conhecidos

Atenção

Se você está implantando novos clusters, siga o *Capítulo 28, Criando imagens atualizadas do SUSE Linux Micro com o Kiwi* para primeiro criar as novas imagens, já que esta é a primeira etapa necessária para criar clusters para ambas as arquiteturas AMD64/Intel 64 e AArch64, além dos clusters de gerenciamento e downstream.

- Ao usar o `toolbox` no SUSE Linux Micro 6.1, a imagem do contêiner padrão não inclui algumas ferramentas que eram incluídas na versão 5.5 anterior. A solução alternativa é configurar o toolbox para usar a imagem do contêiner `suse/sle-micro/5.5/toolbox` anterior. Consulte `toolbox --help` para ver opções de configuração da imagem.

- Em alguns casos, os upgrades contínuos feitos via CAPI podem fazer com que as máquinas fiquem travadas no estado "em exclusão". Isso será resolvido com uma atualização futura (Upstream RKE2 provider issue 655 (https://github.com/rancher/cluster-api-provider-rke2/issues/655))

- Em alguns casos, ao usar um cluster de gerenciamento que foi atualizado do Edge 3.2, os upgrades downstream contínuos feitos via CAPI podem fazer com que as máquinas fiquem travadas no estado "em exclusão". Isso será resolvido com uma atualização futura (Upstream RKE2 provider issue 661 (https://github.com/rancher/cluster-api-provider-rke2/issues/661))

- Ao usar o RKE2 1.32.3, que é resolvido como CVE-2025-1974 (https://nvd.nist.gov/vuln/detail/CVE-2025-1974), o SUSE Linux Micro 6.1 **deve** ser atualizado para incluir o kernel `>=6.4.0-26-default` ou `>=6.4.0-30-rt` (kernel real-time) por causa dos patches de kernel obrigatórios do SELinux. Se não forem aplicados, o pod ingress-nginx continuará no estado `CrashLoopBackOff`. Para aplicar a atualização do kernel, execute `transactional-update` no próprio host (para atualizar todos os pacotes), ou `transactional-update pkg update kernel-default` (ou kernel-rt) para atualizar apenas o kernel e, em seguida, reinicialize o host. Se estiver implantando novos clusters, siga o *Capítulo 28, Criando imagens atualizadas do SUSE Linux Micro com o Kiwi* para criar novas imagens com o kernel mais recente.

- Ao configurar a rede via nm-congfigurator, determinadas configurações que identificam as interfaces por MAC atualmente não funcionam, o que será resolvido em uma atualização futura Problema no NM Configurator upstream (https://github.com/suse-edge/nm-configurator/issues/163)

- Nos clusters de gerenciamento Metal[3] de longa execução, o vencimento do certificado pode provocar falha na conexão do operador bare metal com o Ironic, exigindo como solução alternativa a reinicialização manual do pod Problema nos gráficos do SUSE Edge (https://github.com/suse-edge/charts/issues/178)

- Um bug no controlador de jobs do Kubernetes foi identificado em determinadas condições nas quais ele pode fazer com que os nós do RKE2/K3s permaneçam no estado `NotReady` (consulte o problema #8357 do RKE2 (https://github.com/rancher/rke2/issues/8357)). Os erros podem ter esta aparência:

```
E0605 23:11:18.489721    1 job_controller.go:631] "Unhandled Error" err="syncing
 job: tracking status: adding uncounted pods to status: Operation cannot be
 fulfilled on jobs.batch \"helm-install-rke2-ingress-nginx\": StorageError: invalid
 object, Code: 4, Key: /registry/jobs/kube-system/helm-install-rke2-ingress-nginx,
 ResourceVersion: 0, AdditionalErrorMsg: Precondition failed: UID in precondition:
 0aa6a781-7757-4c61-881a-cb1a4e47802c, UID in object meta: 6a320146-16b8-4f83-88c5-
fc8b5a59a581" logger="UnhandledError"
```

Como solução alternativa, reinicie o pod `kube-controller-manager` com `crictl` como:

```
export CONTAINER_RUNTIME_ENDPOINT=unix:///run/k3s/containerd/containerd.sock
export KUBEMANAGER_POD=$(/var/lib/rancher/rke2/bin/crictl ps --label
 io.kubernetes.container.name=kube-controller-manager --quiet)
/var/lib/rancher/rke2/bin/crictl stop ${KUBEMANAGER_POD} && \
/var/lib/rancher/rke2/bin/crictl rm ${KUBEMANAGER_POD}
```

- Nas versões do RKE2/K3s 1.31 e 1.32, o diretório `/etc/cni` usado para armazenar as configurações de CNI podem não acionar uma notificação sobre os arquivos que estão sendo gravados nele para o `containerd` devido a determinadas condições relacionadas ao `overlayfs` (consulte o problema #8356 do RKE2 (https://github.com/rancher/rke2/issues/8356)). Isso, por sua vez, acaba travando a implantação do RKE2/K3s na espera pela inicialização da CNI, e os nós do RKE2/K3s no estado `NotReady`. Isso pode ser observado no nível do nó com `kubectl describe node <nó_afetado>`:

```
Conditions:
  Type          Status  LastHeartbeatTime              LastTransitionTime
 Reason                     Message
  ----          ------  -----------------              ------------------
 ------                     -------
  Ready         False   Thu, 05 Jun 2025 17:41:28 +0000   Thu, 05 Jun 2025 14:38:16 +0000
   KubeletNotReady           container runtime network not ready: NetworkReady=false
 reason:NetworkPluginNotReady message:Network plugin returns error: cni plugin not
 initialized
```

Como solução alternativa, é possível montar um volume tmpfs no diretório `/etc/cni` antes que o RKE2 seja iniciado. Isso evita o uso do overlayfs, que faz com que o containerd perca notificações e que as configurações sejam regravadas sempre que o nó é reiniciado e os initcontainers dos pods são novamente executados. Se você usa o EIB, isso pode ser um script `04-tmpfs-cni.sh` no diretório `custom/scripts` (conforme explicado aqui [https://github.com/suse-edge/edge-image-builder/blob/release-1.2/docs/building-images.md#custom]) com esta aparência:

```
#!/bin/bash
mkdir -p /etc/cni
mount -t tmpfs -o mode=0700,size=5M tmpfs /etc/cni
echo "tmpfs /etc/cni tmpfs defaults,size=5M,mode=0700 0 0" >> /etc/fstab
```

52.4.4 Versões dos componentes

A tabela a seguir descreve os componentes individuais que compõem o lançamento 3.3.0, incluindo a versão, a versão do gráfico Helm (se aplicável) e de onde obter o artefato lançado no formato binário. Consulte a documentação associada para ver exemplos de uso e implantação.

Nome	Versão	Versão do gráfico Helm	Local do artefato (URL/imagem)
SUSE Linux Micro	6.1 (mais recente)	N/A	Página de downloads do SUSE Linux Micro (https://www.suse.com/download/sle-micro/) SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso (sha256 70b9be28f2d92bc3b228412e4fc2b1 SL-Micro.x86_64-6.1-Base-RT-SelfInstall-GM.install.iso (sha256 9ce83e4545d4b36c7c6a44f7841dc3

			SL-Micro.x86_64-6.1-Base-GM.raw.xz (sha256 36e3efa55822113840dd76fdf6914e SL-Micro.x86_64-6.1-Base-RT-GM.raw.xz (sha256 2ee66735da3e1da107b4878e73ae6
SUSE Multi-Linux Manager	5.0.3	N/A	Página de downloads do SUSE Multi-Linux Manager (https://www.suse.com/download/suse-manager/)
K3s	1.32.3	N/A	Versões upstream do K3s (https://github.com/k3s-io/k3s/releases/tag/v1.32.3%2Bk3s1)
RKE2	1.32.3	N/A	Versão upstream do RKE2 (https://github.com/rancher/rke2/releases/tag/v1.32.3%2Brke2r1)
SUSE Rancher Prime	2.11.1	2.11.1	Repositório Helm do Rancher Prime (https://charts.rancher.com/server-charts/prime/index.yaml) Imagens do contêiner do Rancher 2.11.1 (https://github.com/

			rancher/rancher/releases/download/v2.11.1/rancher-images.txt)
SUSE Storage	1.8.1	106.2.0 + up1.8.1	Repositório de gráficos Helm do Rancher (https://charts.rancher.io/index.yaml) registry.suse.com/rancher/mirrored-longhornio-csi-attacher:v4.7.0 registry.suse.com/rancher/mirrored-longhornio-csi-provisioner:v4.0.1-20241007 registry.suse.com/rancher/mirrored-longhornio-csi-resizer:v1.12.0 registry.suse.com/rancher/mirrored-longhornio-csi-snapshotter:v7.0.2-20241007 registry.suse.com/rancher/mirrored-longhornio-csi-node-driver-registrar:v2.12.0 registry.suse.com/rancher/mirrored-longhornio-livenessprobe:v2.14.0

			registry.suse.com/rancher/mirrored-longhornio-backing-image-manager:v1.7.2 registry.suse.com/rancher/mirrored-longhornio-longhorn-engine:v1.7.2 registry.suse.com/rancher/mirrored-longhornio-longhorn-instance-manager:v1.7.2 registry.suse.com/rancher/mirrored-longhornio-longhorn-manager:v1.7.2 registry.suse.com/rancher/mirrored-longhornio-longhorn-share-manager:v1.7.2 registry.suse.com/rancher/mirrored-longhornio-longhorn-ui:v1.7.2 registry.suse.com/rancher/mirrored-longhornio-support-bundle-kit:v0.0.45 registry.suse.com/rancher/mirrored-longhornio-longhorn-cli:v1.7.2

SUSE Security	5.4.3	106.0.0 + up2.8.5	Repositório de gráficos Helm do Rancher (https://charts.rancher.io/index.yaml) registry.suse.com/rancher/neuvector-controller:5.4.3 registry.suse.com/rancher/neuvector-enforcer:5.4.3 registry.suse.com/rancher/neuvector-manager:5.4.3 registry.suse.com/rancher/neuvector-compliance-config:1.0.4 registry.suse.com/rancher/neuvector-registry-adapter:0.1.6 registry.suse.com/rancher/neuvector-scanner:6 registry.suse.com/rancher/neuvector-updater:0.0.2
Rancher Turtles (CAPI)	0.19.0	303.0.2 + up0.19.0	registry.suse.com/edge/charts/rancher-turtles:303.0.2_up0.19.0 registry.rancher.com/rancher/rancher/turtles:v0.19.0

			registry.rancher.com/rancher/cluster-api-operator:v0.17.0 registry.rancher.com/rancher/cluster-api-metal3-controller:v1.9.3 registry.rancher.com/rancher/cluster-api-metal3-ipam-controller:v1.9.4 registry.suse.com/rancher/cluster-api-controller:v1.9.5 registry.suse.com/rancher/cluster-api-provider-rke2-bootstrap:v0.15.1 registry.suse.com/rancher/cluster-api-provider-rke2-controlplane:v0.15.1
Recursos de isolamento do Rancher Turtles	0.19.0	303.0.2 + up0.19.0	registry.suse.com/edge/charts/rancher-turtles-airgap-resources:303.0.2_up0.19.0
Metal[3]	0.11.3	303.0.5 + up0.11.3	registry.suse.com/edge/charts/metal3:303.0.5_up0.11.3 registry.suse.com/edge/3.3/baremetal-operator:0.9.1

			registry.suse.com/edge/3.3/ironic:26.1.2.4 registry.suse.com/edge/3.3/ironic-ipa-downloader:3.0.6 registry.suse.com/edge/mariadb:10.6.15.1
MetalLB	0.14.9	303.0.0 + up0.14.9	registry.suse.com/edge/charts/metallb:303.0.0_up0.14.9 registry.suse.com/edge/3.3/metallb-controller:v0.14.8 registry.suse.com/edge/3.3/metallb-speaker:v0.14.8 registry.suse.com/edge/3.3/frr:8.4 registry.suse.com/edge/3.3/frr-k8s:v0.0.14
Elemental	1.6.8	1.6.8	registry.suse.com/rancher/elemental-operator-chart:1.6.8 registry.suse.com/rancher/elemental-operator-crds-chart:1.6.8 registry.suse.com/rancher/elemental-operator:1.6.8

Extensão de dashboard do Elemental	3.0.1	3.0.1	Gráfico Helm da extensão do Elemental (https://github.com/rancher/ui-plugin-charts/tree/4.0.0/charts/elemental/3.0.1)
Edge Image Builder	1.2.0	N/A	registry.suse.com/edge/3.3/edge-image-builder:1.2.0
NM Configurator	0.3.2	N/A	Versão upstream do NMConfigurator (https://github.com/suse-edge/nm-configurator/releases/tag/v0.3.2)
KubeVirt	1.4.0	303.0.0 + up0.5.0	registry.suse.com/edge/charts/kubevirt:303.0.0_up0.5.0 registry.suse.com/suse/sles/15.6/virt-operator:1.4.0 registry.suse.com/suse/sles/15.6/virt-api:1.4.0 registry.suse.com/suse/sles/15.6/virt-controller:1.4.0 registry.suse.com/suse/sles/15.6/virt-exportproxy:1.4.0

			registry.suse.com/suse/sles/15.6/virt-exportserver:1.4.0 registry.suse.com/suse/sles/15.6/virt-handler:1.4.0 registry.suse.com/suse/sles/15.6/virt-launcher:1.4.0
Extensão de dashboard KubeVirt	1.3.2	303.0.2 + up1.3.2	registry.suse.com/edge/charts/kubevirt-dashboard-extension:303.0.2_up1.3.2
Containerized Data Importer	1.61.0	303.0.0 + up0.5.0	registry.suse.com/edge/charts/cdi:303.0.0_up0.5.0 registry.suse.com/suse/sles/15.6/cdi-operator:1.61.0 registry.suse.com/suse/sles/15.6/cdi-controller:1.61.0 registry.suse.com/suse/sles/15.6/cdi-importer:1.61.0 registry.suse.com/suse/sles/15.6/cdi-cloner:1.61.0 registry.suse.com/suse/sles/15.6/cdi-apiserver:1.61.0 registry.suse.com/suse/sles/15.6/cdi-uploadserver:1.61.0

			registry.suse.com/suse/sles/15.6/cdi-uploadproxy:1.61.0
Endpoint Copier Operator	0.2.0	303.0.0 + up0.2.1	registry.suse.com/edge/charts/endpoint-copier-operator:303.0.0_up0.2.1 registry.suse.com/edge/3.3/endpoint-copier-operator:0.2.0
Akri (prévia de tecnologia)	0.12.20	303.0.0 + up0.12.20	registry.suse.com/edge/charts/akri:303.0.0_up0.12.20 registry.suse.com/edge/charts/akri-dashboard-extension:303.0.0_up1.3.1 registry.suse.com/edge/3.3/akri-agent:v0.12.20 registry.suse.com/edge/3.3/akri-controller:v0.12.20 registry.suse.com/edge/3.3/akri-debug-echo-discovery-handler:v0.12.20 registry.suse.com/edge/3.3/akri-onvif-discovery-handler:v0.12.20

			registry.suse.com/edge/3.3/akri-opcua-discovery-handler:v0.12.20 registry.suse.com/edge/3.3/akri-udev-discovery-handler:v0.12.20 registry.suse.com/edge/3.3/akri-webhook-configuration:v0.12.20
SR-IOV Network Operator	1.5.0	303.0.2+up1.5.0	registry.suse.com/edge/charts/sriov-network-operator:303.0.2_up1.5.0 registry.suse.com/edge/charts/sriov-crd:303.0.2_up1.5.0
System Upgrade Controller	0.15.2	106.0.0	Repositório de gráficos Helm do Rancher (https://charts.rancher.io/index.yaml) registry.suse.com/rancher/system-upgrade-controller:v0.15.2
Controller de upgrade	0.1.1	303.0.0+up0.1.1	registry.suse.com/edge/charts/upgrade-controller:303.0.0_up0.1.1 registry.suse.com/edge/3.3/upgrade-controller:0.1.1

			registry.suse.com/edge/3.3/kubectl:1.30.3 registry.suse.com/edge/3.3/release-manifest:3.3.0
Construtor Kiwi	10.2.12.0	N/A	registry.suse.com/edge/3.3/kiwi-builder:10.2.12.0

52.5 Prévias de tecnologia

Exceto se especificado de outra forma, elas são aplicáveis à versão 3.3.0 e a todas as versões z-stream subsequentes.

- Akri é uma oferta de prévia de tecnologia e não está sujeita ao escopo padrão de suporte.
- As implantações downstream IPv6 e de pilha dupla são uma oferta de prévia de tecnologia e não estão sujeitas ao escopo padrão de suporte.
- O Precision Time Protocol (PTP) em implantações downstream é uma oferta de prévia de tecnologia e não está sujeito ao escopo padrão de suporte.

52.6 Verificação de componentes

Os componentes mencionados acima podem ser verificados usando os dados do Software Bill Of Materials (SBOM), por exemplo, com o comando `cosign` conforme descrito abaixo:

Faça download da chave pública do SUSE Edge Container da fonte de chaves de assinatura da SUSE (https://www.suse.com/support/security/keys/):

```
> cat key.pem
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEA7N0S2d8LFKW4WU43bq7Z
IZT537xlKe17OQEpYjNrdtqnSwA0/jLtK83m7bTzfYRK4wty/so0g3BGo+x6yDFt
SVXTPBqnYvabU/j7UKaybJtX3jc4SjaezeBqdi96h6yEslvg4VTZDpy6TFP5ZHxZ
A0fX6m5kU2/RYhGXItoeUmL5hZ+APYgYG4/455NBaZT2yOywJ6+1zRgpR0cRAekI
```

```
OZXl51k0ebsGV6ui/NGECO6MB5e3arAhszf8eHDE02FeNJw5cimXkgDh/1Lg3KpO
dvUNm0EPWvnkNYeMCKR+687QG0bXqSVyCbY6+HG/HLkeBWkv6Hn41oeTSLrjYVGa
T3zxPVQM726sami6pgZ5vULyOleQuKBZrlFhFLbFyXqv1/DokUqEppm2Y3xZQv77
fMNogapp0qYz+nE3wSK4UHPd9z+2bq5WEkQSalYxadyuqOzxqZgSoCNoX5iIuWte
Zf1RmHjiEndg/2UgxKUysVnyCpiWoGbalM4dnWE24102050Gj6M4B5fe73hbaRlf
NBqP+97uznnRlSl8FizhXzdzJiVPcRav1tDdRUyDE2XkNRXmGfD3aCmILhB27SOA
Lppkouw849PWBt9kDMvzelUYLpINYpHRi2+/eyhHNlufeyJ7e7d6N9VcvjR/6qWG
64iSkcF2DTW61CN5TrCe0k0CAwEAAQ==
-----END PUBLIC KEY-----
```

Verifique o hash da imagem do contêiner, usando, por exemplo, o `crane`:

```
> crane digest registry.suse.com/edge/3.3/baremetal-operator:0.9.1 --platform linux/amd64
sha256:02c5590cd51b1a1ea02f9908f2184ef4fbc856eb0197e804a7d57566d9278ddd
```

Nota

Para imagens de várias arquiteturas, também é necessário especificar uma plataforma ao obter o resumo, por exemplo, `--platform linux/amd64` ou `--platform linux/arm64`. Se isso não for feito, um erro será retornado na etapa seguinte (`Error: no matching attestations`, "Erro: não há atestados correspondentes").

Faça a verificação com o comando `cosign`:

```
> cosign verify-attestation --type spdxjson --key key.pem registry.suse.com/edge/3.3/
baremetal-
operator@sha256:02c5590cd51b1a1ea02f9908f2184ef4fbc856eb0197e804a7d57566d9278ddd > /dev/
null
#
Verification for registry.suse.com/edge/3.3/baremetal-
operator@sha256:02c5590cd51b1a1ea02f9908f2184ef4fbc856eb0197e804a7d57566d9278ddd --
The following checks were performed on each of these signatures:
  - The cosign claims were validated
  - The claims were present in the transparency log
  - The signatures were integrated into the transparency log when the certificate was
 valid
  - The signatures were verified against the specified public key
```

Extraia os dados do SBOM conforme descrito na documentação do SBOM da SUSE (https://www.suse.com/support/security/sbom/):

```
> cosign verify-attestation --type spdxjson --key key.pem registry.suse.com/edge/3.2/
baremetal-
```

```
operator@sha256:d85c1bcd286dec81a3806a8fb8b66c0e0741797f23174f5f6f41281b1e27c52f | jq
 '.payload | @base64d | fromjson | .predicate'
```

52.7 Etapas de upgrade

Consulte a *Parte VI, "Operações de dia 2"* para obter detalhes sobre como fazer upgrade para uma nova versão.

52.8 Ciclo de vida de suporte do produto

O SUSE Edge conta com o suporte reconhecido da SUSE, líder consagrada em tecnologia com um histórico comprovado de prestação de serviços de suporte de qualidade empresarial. Para obter mais informações, visite https://www.suse.com/lifecycle e a página da política de suporte em https://www.suse.com/support/policy.html. Se você tiver dúvidas sobre como criar um caso de suporte, como a SUSE classifica os níveis de gravidade ou sobre o escopo do suporte, consulte o Manual de Suporte Técnico em https://www.suse.com/support/handbook/.

O SUSE Edge "3.3" tem suporte de produção de 18 meses, com os primeiros 6 meses de "suporte completo", seguidos de 12 meses de "suporte de manutenção". Após essas fases de suporte, o produto chega ao "fim do serviço" (EOL, End of Life) e não é mais suportado. Encontre mais informações sobre as fases do ciclo de vida na tabela abaixo:

Suporte completo (6 meses)	Correções de bugs urgentes e de alta prioridade selecionadas serão lançadas durante a janela de suporte completo, e todos os outros patches (não urgentes, melhorias, novos recursos) serão lançados de acordo com a programação de lançamento regular.
Suporte de manutenção (12 meses)	Durante esse período, apenas correções críticas serão lançadas por meio de patches. Outras correções de bugs poderão ser lançadas a critério da SUSE, mas não devem ser esperadas.

Fim do serviço (EOL, End of Life)	Quando um produto chega à data de fim do serviço, o cliente ainda pode usá-lo nos termos do contrato de licenciamento do produto. Os planos de suporte da SUSE não são válidos para versões de produtos que ultrapassaram a data EOL.

Exceto se for claramente especificado, todos os componentes listados são considerados em disponibilidade geral (GA, Generally Available) e cobertos pelo escopo padrão de suporte da SUSE. Alguns componentes podem constar na lista como "prévia de tecnologia", em que a SUSE concede aos clientes acesso antecipado a recursos e funcionalidades pré-GA para avaliação, mas que não estão sujeitos às políticas de suporte padrão e não são recomendados em casos de uso de produção. A SUSE tem grande consideração pelo feedback e pelas sugestões sobre melhorias que possam ser feitas nos componentes em prévia de tecnologia, mas reserva-se o direito de descontinuar um recurso em prévia de tecnologia antes de lançá-lo em disponibilidade geral, caso ele não atenda às necessidades de nossos clientes ou não atinja o estado de maturidade exigido pela SUSE.

Observe que a SUSE deve, ocasionalmente, descontinuar recursos ou alterar especificações de APIs. Alguns motivos para descontinuar um recurso ou alterar uma API são a atualização do recurso ou sua substituição por uma nova implementação, um novo conjunto de recursos, uma tecnologia upstream que não está mais disponível ou a comunidade upstream que fez alterações incompatíveis. Não há intenção de que isso aconteça dentro de uma determinada versão secundária (x.z) e, portanto, todas as versões z-stream manterão a compatibilidade com as APIs e as funcionalidades dos recursos. A SUSE se empenhará em enviar avisos sobre descontinuação com muito tempo de antecedência como parte das Notas de lançamento, junto com soluções alternativas, sugestões e mitigações para minimizar interrupções de serviços.

A equipe do SUSE Edge também agradece o feedback da comunidade, em que é possível relatar problemas dentro do respectivo repositório de códigos em https://www.github.com/suse-edge.

52.9 Obtendo o código-fonte

Este produto SUSE inclui materiais licenciados à SUSE de acordo com a Licença Pública Geral (GPL, General Public License) GNU e várias outras licenças de código aberto. A GPL exige que a SUSE providencie o código-fonte em conformidade com o material licenciado sob a GPL, e

a SUSE cumpre todos os outros requisitos de licença de código aberto. Sendo assim, a SUSE disponibiliza todo o código-fonte que, de modo geral, está armazenado no repositório SUSE Edge do GitHub (https://www.github.com/suse-edge), no repositório SUSE Rancher do GitHub (https://www.github.com/rancher) para componentes dependentes, e especificamente para o SUSE Linux Micro, o código-fonte está disponível para download em https://www.suse.com/download/sle-micro (https://www.suse.com/download/sle-micro/) na "mídia 2".

52.10 Avisos legais

A SUSE não faz representações ou garantias no que se refere ao conteúdo ou ao uso desta documentação e, especificamente, isenta-se de quaisquer garantias expressas ou implícitas de comercialização ou adequação para uma finalidade específica. Além disso, a SUSE reserva o direito de revisar esta publicação e fazer alterações em seu conteúdo, a qualquer momento, sem a obrigação de notificar qualquer pessoa ou entidade de tais revisões ou alterações.

A SUSE não faz representações ou garantias em relação a nenhum software e, especificamente, isenta-se de quaisquer garantias expressas ou implícitas de comercialização ou adequação a uma finalidade específica. Além disso, a SUSE reserva o direito de fazer alterações em qualquer parte do software SUSE, a qualquer momento, sem a obrigação de notificar qualquer pessoa ou entidade de tais alterações.

Os produtos ou as informações técnicas constantes neste Contrato podem estar sujeitos aos controles de exportação dos EUA e à legislação comercial de outros países. Você concorda em cumprir todas as normas de controle de exportação e obter as licenças ou as classificações necessárias para exportar, reexportar ou importar produtos entregáveis. Você concorda em não exportar ou reexportar para entidades que façam parte das listas atuais de exclusão de exportação dos EUA nem para países embargados ou terroristas, conforme especificado nas leis de exportação dos EUA. Você concorda em não usar os produtos entregáveis para fins proibidos como armas nucleares, mísseis ou armamento químico/biológico. Visite https://www.suse.com/company/legal/ para obter mais informações sobre exportação do software SUSE. A SUSE não assumirá nenhuma responsabilidade se você não obtiver as aprovações de exportação necessárias.

Copyright © 2024 SUSE LLC.

Este documento de Notas de lançamento é protegido por uma licença Creative Commons Atribuição-NãoComercial-SemDerivações 4.0 Internacional (CC-BY-ND-4.0). Você deve ter recebido uma cópia da licença junto com este documento. Do contrário, acesse https://creativecommons.org/licenses/by-nd/4.0/.

A SUSE tem direitos de propriedade intelectual pertinentes à tecnologia incorporada ao produto descrito neste documento. Em particular, e sem limitação, esses direitos de propriedade intelectual podem incluir uma ou mais patentes nos EUA, relacionadas em https://www.suse.com/company/legal/, e uma ou mais patentes adicionais ou solicitações de patentes pendentes nos EUA e em outros países.

Para conhecer as marcas registradas da SUSE, consulte a lista de marcas registradas e marcas de serviço da SUSE (https://www.suse.com/company/legal/). Todas as marcas registradas de terceiros pertencem aos respectivos proprietários. Para saber as informações e os requisitos de uso da marca SUSE, consulte as diretrizes publicadas em https://brand.suse.com/.