



SUSE Edge Documentation

SUSE Edge Documentation

Publication Date: 2025-03-06

<https://documentation.suse.com> 

Contents

SUSE Edge Documentation **xv**

- 1 What is SUSE Edge? **xv**
- 2 Design Philosophy **xv**
- 3 Which Quick Start should you use? **xvi**
 - Directed network provisioning **xvi** • "Phone home" network provisioning **xvii** • Image-based provisioning **xvii**
- 4 Components used in SUSE Edge **xviii**

I QUICK STARTS **1**

1 BMC automated deployments with Metal³ **2**

- 1.1 Why use this method **2**
- 1.2 High-level architecture **3**
- 1.3 Prerequisites **4**
 - Setup Management Cluster **4** • Installing Metal³ dependencies **5** • Installing cluster API dependencies **7** • Prepare downstream cluster image **8** • Adding BareMetalHost inventory **11** • Creating downstream clusters **15** • Control plane deployment **15** • Worker/Compute deployment **18** • Cluster deprovisioning **21**
- 1.4 Known issues **22**
- 1.5 Planned changes **22**
- 1.6 Additional resources **22**
 - Single-node configuration **23** • Disabling TLS for virtualmedia ISO attachment **23**

2 Remote host onboarding with Elemental 24

- 2.1 High-level architecture 25
- 2.2 Resources needed 26
- 2.3 How to use Elemental 27
 - Build bootstrap cluster 27 • Install Rancher 28 • Install Elemental 29 • Build the installation media 42 • Boot the downstream nodes 43 • Create downstream clusters 43
- 2.4 Node Reset 46
- 2.5 Next steps 47

3 Standalone clusters with Edge Image Builder 48

- 3.1 Prerequisites 48
 - Getting the EIB Image 48
- 3.2 Creating the image configuration directory 49
- 3.3 Creating the image definition file 49
 - Configuring OS Users 50 • Configuring RPM packages 51 • Configuring Kubernetes cluster and user workloads 53 • Configuring the network 55
- 3.4 Building the image 57
- 3.5 Debugging the image build process 60
- 3.6 Testing your newly built image 60

II COMPONENTS USED 61

4 Rancher 62

- 4.1 Key Features of Rancher 62
- 4.2 Rancher's use in SUSE Edge 62
 - Centralized Kubernetes management 62 • Simplified cluster deployment 63 • Application deployment and management 63 • Security and policy enforcement 63

- 4.3 Best practices 63
 - GitOps 63 • Observability 63
- 4.4 Installing with Edge Image Builder 63
- 4.5 Additional Resources 64
- 5 Rancher Dashboard Extensions 65**
- 5.1 Prerequisites 65
- 5.2 Installation 65
 - Installing with Helm 65 • Installing with Fleet 66
- 5.3 KubeVirt Dashboard Extension 69
- 5.4 Akri Dashboard Extension 69
- 6 Fleet 70**
- 6.1 Installing Fleet with Helm 70
- 6.2 Using Fleet with Rancher 70
- 6.3 Accessing Fleet in the Rancher UI 70
 - Dashboard 71 • Git repos 72 • Clusters 72 • Cluster groups 72 • Advanced 72
- 6.4 Example of installing KubeVirt with Rancher and Fleet using Rancher dashboard 72
- 6.5 Debugging and troubleshooting 77
- 6.6 Fleet examples 80
- 7 SLE Micro 81**
- 7.1 How does SUSE Edge use SLE Micro? 81
- 7.2 Best practices 81
 - Installation media 81 • Local administration 81
- 7.3 Known issues 82

8 Metal³ 83

8.1 How does SUSE Edge use Metal3? 83

8.2 Known issues 83

9 Edge Image Builder 84

9.1 How does SUSE Edge use Edge Image Builder? 84

9.2 Getting started 85

9.3 Known issues 85

10 Edge Networking 86

10.1 Overview of NetworkManager 86

10.2 Overview of nmstate 86

10.3 Enter: NetworkManager Configurator (nmc) 86

10.4 How does SUSE Edge use NetworkManager Configurator? 87

10.5 Configuring with Edge Image Builder 87

Prerequisites 87 • Getting the Edge Image Builder container image 87 • Creating the image configuration directory 88 • Creating the image definition file 88 • Defining the network configurations 89 • Building the OS image 94 • Provisioning the edge nodes 95 • Unified node configurations 102 • Custom network configurations 105

11 Elemental 109

11.1 How does SUSE Edge use Elemental? 109

11.2 Best practices 110

Installation media 110 • Labels 110

11.3 Known issues 110

12 Akri 111

- 12.1 How does SUSE Edge use Akri? 111
Installing Akri 111 • Configuring Akri 111 • Writing and deploying additional Discovery Handlers 113 • Akri Rancher Dashboard Extension 113

13 K3s 120

- 13.1 How does SUSE Edge use K3s 120
- 13.2 Best practices 120
Installation 120 • Fleet for GitOps workflow 120 • Storage management 120 • Load balancing and HA 121

14 RKE2 122

- 14.1 RKE2 vs K3s 122
- 14.2 How does SUSE Edge use RKE2? 122
- 14.3 Best practices 123
Installation 123 • High availability 123 • Networking 124 • Storage 124

15 Longhorn 125

- 15.1 Prerequisites 125
- 15.2 Manual installation of Longhorn 125
Installing Open-iSCSI 125 • Installing Longhorn 126
- 15.3 Creating Longhorn volumes 127
- 15.4 Accessing the UI 130
- 15.5 Installing with Edge Image Builder 130

16 NeuVector 133

- 16.1 How does SUSE Edge use NeuVector? 134
- 16.2 Important notes 134
- 16.3 Installing with Edge Image Builder 134

17 MetalLB 135

17.1 How does SUSE Edge use MetalLB? 135

17.2 Best practices 136

17.3 Known issues 136

18 Edge Virtualization 137

18.1 KubeVirt overview 137

18.2 Prerequisites 138

18.3 Manual installation of Edge Virtualization 138

18.4 Deploying virtual machines 142

18.5 Using virtctl 145

18.6 Simple ingress networking 147

18.7 Using the Rancher UI extension 149

Installation 149 • Using KubeVirt Rancher Dashboard Extension 149

18.8 Installing with Edge Image Builder 153

III HOW-TO GUIDES 154

19 MetalLB on K3s (using L2) 155

19.1 Why use this method 155

19.2 MetalLB on K3s (using L2) 155

19.3 Prerequisites 156

Deployment 156 • Configuration 157 • Traefik and MetalLB 158 • Usage 158

19.4 Ingress with MetalLB 161

20 MetalLB in front of the Kubernetes API server 164

20.1 Prerequisites 164

20.2 Installing RKE2/K3s 164

20.3	Configuring an existing cluster	166
20.4	Installing MetalLB	166
20.5	Installing the Endpoint Copier Operator	167
20.6	Adding control-plane nodes	169
21	Air-gapped deployments with Edge Image Builder	171
21.1	Intro	171
21.2	Prerequisites	171
21.3	Libvirt Network Configuration	172
21.4	Base Directory Configuration	172
21.5	Base Definition File	174
21.6	Rancher Installation	175
21.7	NeuVector Installation	187
21.8	Longhorn Installation	189
21.9	KubeVirt and CDI Installation	193
21.10	Troubleshooting	196
IV	THIRD-PARTY INTEGRATION	197
22	NATS	198
22.1	Architecture	198
	NATS client applications	198
	NATS service infrastructure	198
	Simple messaging design	199
	NATS JetStream	199
22.2	Installation	199
	Installing NATS on top of K3s	199
	NATS as a back-end for K3s	201
23	NVIDIA GPUs on SLE Micro	203
23.1	Intro	203

- 23.2 Prerequisites 204
- 23.3 Manual installation 204
- 23.4 Further validation of the manual installation 209
- 23.5 Implementation with Kubernetes 212
- 23.6 Bringing it together via Edge Image Builder 215
- 23.7 Resolving issues 218
 - nvidia-smi does not find the GPU 218

V DAY 2 OPERATIONS 219

24 Management Cluster 220

- 24.1 RKE2 upgrade 220
- 24.2 OS upgrade 221
- 24.3 Helm upgrade 222
 - EIB deployed helm chart 222
 - Non-EIB deployed helm chart 228
- 24.4 Cluster API upgrade 231

25 Downstream clusters 232

- 25.1 Introduction 232
 - Components 232
 - Determine your use-case 234
 - Day 2 workflow 235
- 25.2 System upgrade controller deployment guide 235
 - Deployment 236
 - Monitor SUC resources using Rancher 241
- 25.3 OS package update 250
 - Components 250
 - Requirements 251
 - Update procedure 253
 - OS package update - SUC Plan deployment 257
- 25.4 Kubernetes version upgrade 262
 - Components 262
 - Requirements 263
 - Upgrade procedure 264
 - Kubernetes version upgrade - SUC Plan deployment 269

25.5	Helm chart upgrade	275
	Components	275 • Preparation for air-gapped environments 275 • Upgrade procedure 279
VI	PRODUCT DOCUMENTATION	303
26	SUSE Adaptive Telco Infrastructure Platform (ATIP)	304
27	Concept & Architecture	305
27.1	ATIP Architecture	306
27.2	Components	307
27.3	Example deployment flows	308
	Example 1: Deploying a new management cluster with all components installed	308 • Example 2: Deploying a single-node downstream cluster with Telco profiles to enable it to run Telco workloads 309 • Example 3: Deploying a high availability downstream cluster using MetalLB as a Load Balancer 310
28	Requirements & Assumptions	313
28.1	Hardware	313
28.2	Network	314
28.3	Services (DHCP, DNS, etc.)	315
28.4	Disabling rebootmgr	316
29	Setting up the management cluster	317
29.1	Introduction	317
29.2	Steps to set up the management cluster	318
29.3	Image preparation for connected environments	321
	Directory structure	321 • Management cluster definition file 322 • Custom folder 328 • Kubernetes folder 336 • Networking folder 341

- 29.4 Image preparation for air-gap environments **343**
 - Directory structure for air-gap environments **343** • Modifications in the definition file **344** • Modifications in the custom folder **349**
- 29.5 Image creation **354**
- 29.6 Provision the management cluster **354**
- 30 Telco features configuration 355**
- 30.1 Kernel image for real time **356**
- 30.2 CPU tuned configuration **357**
- 30.3 CNI Configuration **359**
 - Cilium **359**
- 30.4 SR-IOV **360**
- 30.5 DPDK **370**
- 30.6 vRAN acceleration (Intel ACC100/ACC200) **372**
- 30.7 Huge pages **374**
- 30.8 CPU pinning configuration **376**
- 30.9 NUMA-aware scheduling **378**
 - Identifying NUMA nodes **378**
- 30.10 Metal LB **379**
- 30.11 Private registry configuration **380**
- 31 Fully automated directed network provisioning 383**
- 31.1 Introduction **383**
- 31.2 Prepare downstream cluster image for connected scenarios **384**
 - Prerequisites for connected scenarios **384** • Image configuration for connected scenarios **384** • Image creation **388**
- 31.3 Prepare downstream cluster image for air-gap scenarios **389**
 - Prerequisites for air-gap scenarios **389** • Image configuration for air-gap scenarios **389** • Image creation for air-gap scenarios **394**

- 31.4 Downstream cluster provisioning with Directed network provisioning (single-node) 394
- 31.5 Downstream cluster provisioning with Directed network provisioning (multi-node) 401
- 31.6 Advanced Network Configuration 410
- 31.7 Telco features (DPDK, SR-IOV, CPU isolation, huge pages, NUMA, etc.) 414
- 31.8 Private registry 423
- 31.9 Downstream cluster provisioning in air-gapped scenarios 426
 - Requirements for air-gapped scenarios 426 • Enroll the bare-metal hosts in air-gap scenarios 426 • Provision the downstream cluster in air-gap scenarios 427

32 Lifecycle actions 435

- 32.1 Management cluster upgrades 435
- 32.2 Downstream cluster upgrades 435

VII APPENDIX 439

33 Release Notes 440

- 33.1 Abstract 440
- 33.2 About 441
- 33.3 Release 3.0.3 441
 - Bug & Security Fixes 442 • Components Versions 442
- 33.4 Release 3.0.2 451
 - New Features 452 • Bug & Security Fixes 452 • Components Versions 452
- 33.5 Release 3.0.1 462
 - New Features 462 • Bug & Security Fixes 462 • Components Versions 462

- 33.6 Release 3.0.0 472
 - New Features 472 • Bug & Security Fixes 472 • Components Versions 472
- 33.7 Components Verification 481
- 33.8 Upgrade Steps 482
- 33.9 Known Limitations 482
- 33.10 Product Support Lifecycle 483
- 33.11 Obtaining source code 484
- 33.12 Legal notices 484

SUSE Edge Documentation

Welcome to the SUSE Edge documentation. You will find quick start guides, validated designs, guidance on using components, third-party integrations, and best practices for managing your edge computing infrastructure and workloads.

1 What is SUSE Edge?

SUSE Edge is a purpose-built, tightly integrated, and comprehensively validated end-to-end solution for addressing the unique challenges of the deployment of infrastructure and cloud-native applications at the edge. Its driving focus is to provide an opinionated, yet highly flexible, highly scalable, and secure platform that spans initial deployment image building, node provisioning and onboarding, application deployment, observability, and complete lifecycle operations. The platform is built on best-of-breed open source software from the ground up, consistent with both our 30-year history in delivering secure, stable, and certified SUSE Linux platforms and our experience in providing highly scalable and feature-rich Kubernetes management with our Rancher portfolio. SUSE Edge builds on-top of these capabilities to deliver functionality that can address a wide number of market segments, including retail, medical, transportation, logistics, telecommunications, smart manufacturing, and Industrial IoT.

2 Design Philosophy

The solution is designed with the notion that there is no "one-size-fits-all" edge platform due to customers' widely varying requirements and expectations. Edge deployments push us to solve, and continually evolve, some of the most challenging problems, including massive scalability, restricted network availability, physical space constraints, new security threats and attack vectors, variations in hardware architecture and system resources, the requirement to deploy and interface with legacy infrastructure and applications, and customer solutions that have extended lifespans. Since many of these challenges are different from traditional ways of thinking, e.g. deployment of infrastructure and applications within data centers or in the public cloud, we have to look into the design in much more granular detail, and rethinking many common assumptions.

For example, we find value in minimalism, modularity, and ease of operations. Minimalism is important for edge environments since the more complex a system is, the more likely it is to break. When looking at hundreds of locations, up to hundreds of thousands, complex systems will break in complex ways. Modularity in our solution allows for more user choice while removing unneeded complexity in the deployed platform. We also need to balance these with the ease of operations. Humans may make mistakes when repeating a process thousands of times, so the platform should make sure any potential mistakes are recoverable, eliminating the need for on-site technician visits, but also strive for consistency and standardization.

3 Which Quick Start should you use?

Due to the varying set of operating environments and lifecycle requirements, we've implemented support for a number of distinct deployment patterns that loosely align to market segments and use-cases that SUSE Edge operates in. We have documented a quickstart guide for each of these deployment patterns to help you get familiar with the SUSE Edge platform based around your needs. The three deployment patterns that we support today are described below, with a link to the respective quickstart page.

3.1 Directed network provisioning

Directed network provisioning is where you know the details of the hardware you wish to deploy to and have direct access to the out-of-band management interface to orchestrate and automate the entire provisioning process. In this scenario, our customers expect a solution to be able to provision edge sites fully automated from a centralized location, going much further than the creation of a boot image by minimizing the manual operations at the edge location; simply rack, power, and attach the required networks to the physical hardware, and the automation process powers up the machine via the out-of-band management (e.g. via the Redfish API) and handles the provisioning, onboarding, and deployment of infrastructure without user intervention. The key for this to work is that the systems are known to the administrators; they know which hardware is in which location, and that deployment is expected to be handled centrally.

This solution is the most robust since you are directly interacting with the hardware's management interface, are dealing with known hardware, and have fewer constraints on network availability. Functionality wise, this solution extensively uses Cluster API and Metal³ for automated provisioning from baremetal, through operating system, Kubernetes, and layered applications,

and provides the ability to link into the rest of the common lifecycle management capabilities of SUSE Edge post-deployment. The quickstart for this solution can be found in [Chapter 1, BMC automated deployments with Metal³](#).

3.2 "Phone home" network provisioning

Sometimes you are operating in an environment where the central management cluster cannot manage the hardware directly (for example, your remote network is behind a firewall or there is no out-of-band management interface; common in "PC" type hardware often found at the edge). In this scenario, we provide tooling to remotely provision clusters and their workloads with no need to know where hardware is being shipped when it is bootstrapped. This is what most people think of when they think about edge computing; it's the thousands or tens of thousands of somewhat unknown systems booting up at edge locations and securely phoning home, validating who they are, and receiving their instructions on what they're supposed to do. Our requirements here expect provisioning and lifecycle management with very little user-intervention other than either pre-imaging the machine at the factory, or simply attaching a boot image, e.g. via USB, and switching the system on. The primary challenges in this space are addressing scale, consistency, security, and lifecycle of these devices in the wild.

This solution provides a great deal of flexibility and consistency in the way that systems are provisioned and on-boarded, regardless of their location, system type or specification, or when they're powered on for the first time. SUSE Edge enables full flexibility and customization of the system via Edge Image Builder, and leverages the registration capabilities Rancher's Elemental offering for node on-boarding and Kubernetes provisioning, along with SUSE Manager for operating system patching. The quick start for this solution can be found in [Chapter 2, Remote host onboarding with Elemental](#).

3.3 Image-based provisioning

For customers that need to operate in standalone, air-gapped, or network limited environments, SUSE Edge provides a solution that enables customers to generate fully customized installation media that contains all of the required deployment artifacts to enable both single-node and multi-node highly-available Kubernetes clusters at the edge, including any workloads or additional layered components required, all without any network connectivity to the outside world, and without the intervention of a centralized management platform. The user-experience follows closely to the "phone home" solution in that installation media is provided to the target systems,

but the solution will "bootstrap in-place". In this scenario, it's possible to attach the resulting clusters into Rancher for ongoing management (i.e. going from a "disconnected" to "connected" mode of operation without major reconfiguration or redeployment), or can continue to operate in isolation. Note that in both cases the same consistent mechanism for automating lifecycle operations can be applied.

Furthermore, this solution can be used to quickly create management clusters that may host the centralized infrastructure that supports both the "directed network provisioning" and "phone home network provisioning" models as it can be the quickest and most simple way to provision all types of Edge infrastructure. This solution heavily utilizes the capabilities of SUSE Edge Image Builder to create fully customized and unattended installation media; the quickstart can be found in [Chapter 3, Standalone clusters with Edge Image Builder](#).

4 Components used in SUSE Edge

SUSE Edge is comprised of both existing SUSE components, including those from the Linux and Rancher teams, along with additional features and components built by the Edge team to enable SUSE to address both the infrastructure requirements and intricacies. The list of components, along with a link to a high-level description of each and how it's used in SUSE Edge can be found below:

- Rancher ([Chapter 4, Rancher](#))
- Rancher Dashboard Extensions ([Chapter 5, Rancher Dashboard Extensions](#))
- Fleet ([Chapter 6, Fleet](#))
- SLE Micro ([Chapter 7, SLE Micro](#))
- Metal³ ([Chapter 8, Metal³](#))
- Edge Image Builder ([Chapter 9, Edge Image Builder](#))
- NetworkManager Configurator ([Chapter 10, Edge Networking](#))
- Elemental ([Chapter 11, Elemental](#))
- Akri ([Chapter 12, Akri](#))
- K3s ([Chapter 13, K3s](#))
- RKE2 ([Chapter 14, RKE2](#))

- Longhorn (*Chapter 15, Longhorn*)
- NeuVector (*Chapter 16, NeuVector*)
- MetalLB (*Chapter 17, MetalLB*)
- KubeVirt (*Chapter 18, Edge Virtualization*)

I Quick Starts

- 1 BMC automated deployments with Metal³ 2
- 2 Remote host onboarding with Elemental 24
- 3 Standalone clusters with Edge Image Builder 48

Quick Starts here

1 BMC automated deployments with Metal³

Metal³ is a [CNCF project \(https://metal3.io/\)](https://metal3.io/) which provides bare-metal infrastructure management capabilities for Kubernetes.

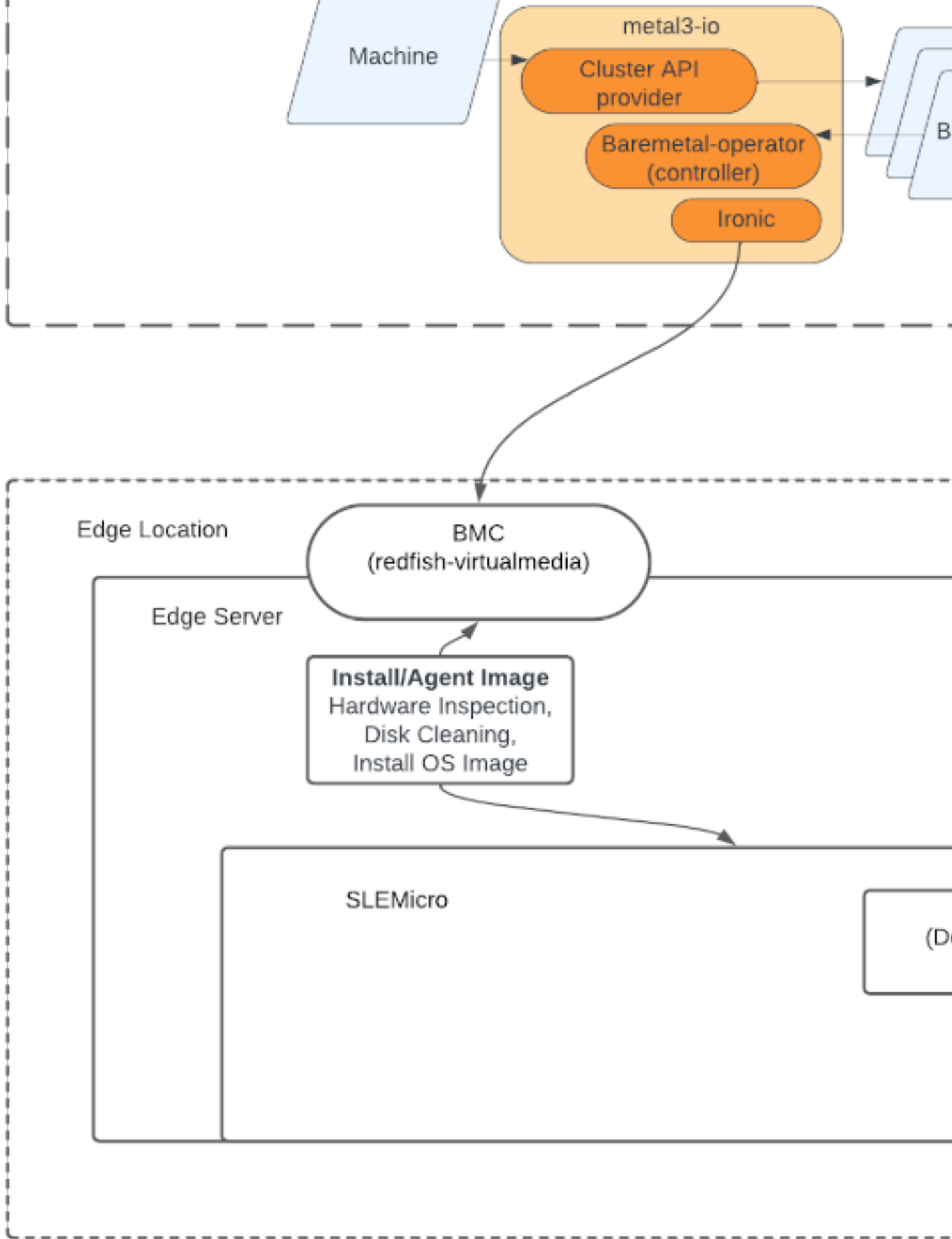
Metal³ provides Kubernetes-native resources to manage the lifecycle of bare-metal servers which support management via out-of-band protocols such as [Redfish \(https://www.dmtf.org/standards/redfish\)](https://www.dmtf.org/standards/redfish).

It also has mature support for [Cluster API \(CAPI\) \(https://cluster-api.sigs.k8s.io/\)](https://cluster-api.sigs.k8s.io/) which enables management of infrastructure resources across multiple infrastructure providers via broadly adopted vendor-neutral APIs.

1.1 Why use this method

This method is useful for scenarios where the target hardware supports out-of-band management, and a fully automated infrastructure management flow is desired.

A management cluster is configured to provide declarative APIs that enable inventory and state management of downstream cluster bare-metal servers, including automated inspection, cleaning and provisioning/deprovisioning.



1.3 Prerequisites

There are some specific constraints related to the downstream cluster server hardware and networking:

- Management cluster
 - Must have network connectivity to the target server management/BMC API
 - Must have network connectivity to the target server control plane network
 - For multi-node management clusters, an additional reserved IP address is required
- Hosts to be controlled
 - Must support out-of-band management via Redfish, iDRAC or iLO interfaces
 - Must support deployment via virtual media (PXE is not currently supported)
 - Must have network connectivity to the management cluster for access to the Metal³ provisioning APIs

Some tools are required, these can be installed either on the management cluster, or on a host which can access it.

- Kubectl (<https://kubernetes.io/docs/reference/kubectl/kubectl/>), Helm (<https://helm.sh>) and Clusterctl (<https://cluster-api.sigs.k8s.io/user/quick-start.html#install-clusterctl>)
- A container runtime such as Podman (<https://podman.io>) or Rancher Desktop (<https://rancherdesktop.io>)

The [SLE-Micro.x86_64-5.5.0-Default-GM.raw.xz](#) OS image file must be downloaded from the [SUSE Customer Center \(https://scc.suse.com/\)](https://scc.suse.com/) or the [SUSE Download page \(https://www.suse.com/download/sle-micro/\)](https://www.suse.com/download/sle-micro/).

1.3.1 Setup Management Cluster

The basic steps to install a management cluster and use Metal³ are:

1. Install an RKE2 management cluster
2. Install Rancher

3. Install a storage provider
4. Install the Metal³ dependencies
5. Install CAPI dependencies
6. Build a SLEMicro OS image for downstream cluster hosts
7. Register BareMetalHost CRs to define the bare-metal inventory
8. Create a downstream cluster by defining CAPI resources

This guide assumes an existing RKE2 cluster and Rancher (including cert-manager) has been installed, for example by using Edge Image Builder ([Chapter 9, Edge Image Builder](#)).





Tip

The steps here can also be fully automated as described in the ATIP management cluster documentation ([Chapter 29, Setting up the management cluster](#)).

1.3.2 Installing Metal³ dependencies

If not already installed as part of the Rancher installation, cert-manager must be installed and running.

A persistent storage provider must be installed. Longhorn is recommended but local-path can also be used for dev/PoC environments. The instructions below assume a StorageClass has been marked as default (<https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/>) , otherwise additional configuration for the Metal³ chart is required.

An additional IP is required, which is managed by MetalLB (<https://metallb.universe.tf/>)  to provide a consistent endpoint for the Metal³ management services. This IP must be part of the control plane subnet and reserved for static configuration (not part of any DHCP pool).



Tip

If the management cluster is a single node, the requirement for an additional floating IP managed via MetalLB can be avoided, see Single-node configuration ([Section 1.6.1, "Single-node configuration"](#))

1. First, we install MetalLB:

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

2. Then we define an IPAddressPool and L2Advertisement using the reserved IP, defined as STATIC_IRONIC_IP below:

```
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. Now Metal³ can be installed:

```
helm install \
  metal3 oci://registry.suse.com/edge/metal3-chart \
  --namespace metal3-system \
  --create-namespace \
```

```
--set global.ironicIP="${STATIC_IRONIC_IP}"
```

4. It can take around two minutes for the `initContainer` to run on this deployment, so ensure the pods are all running before proceeding:

```
kubectl get pods -n metal3-system
```

NAME	READY	STATUS	RESTARTS
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			



Warning

Do not proceed to the following steps until all pods in the `metal3-system` namespace are running

1.3.3 Installing cluster API dependencies

First, we need to disable the Rancher-embedded CAPI controller:

```
cat <<-EOF | kubectl apply -f -
apiVersion: management.cattle.io/v3
kind: Feature
metadata:
  name: embedded-cluster-api
spec:
  value: false
EOF

kubectl delete mutatingwebhookconfiguration.admissionregistration.k8s.io mutating-
webhook-configuration
kubectl delete validatingwebhookconfigurations.admissionregistration.k8s.io validating-
webhook-configuration
kubectl wait --for=delete namespace/cattle-provisioning-capi-system --timeout=300s
```

Then, to use the SUSE images, a configuration file is needed:

```
mkdir ~/.cluster-api
cat > ~/.cluster-api/clusterctl.yaml <<EOF
```

```
images:
  all:
    repository: registry.suse.com/edge
EOF
```

Install [clusterctl](https://cluster-api.sigs.k8s.io/user/quick-start.html#install-clusterctl) (https://cluster-api.sigs.k8s.io/user/quick-start.html#install-clusterctl) 1.6.x, after which we will install the core, infrastructure, bootstrap and control plane providers as follows:

```
clusterctl init --core "cluster-api:v1.6.2" --infrastructure "metal3:v1.6.0" --bootstrap "rke2:v0.4.1" --control-plane "rke2:v0.4.1"
```

After some time, the controller pods should be running in the `capi-system`, `capm3-system`, `rke2-bootstrap-system` and `rke2-control-plane-system` namespaces.

1.3.4 Prepare downstream cluster image

Edge Image Builder ([Chapter 9, Edge Image Builder](#)) is used to prepare a modified SLEMicro base image which is provisioned on downstream cluster hosts.

In this guide, we cover the minimal configuration necessary to deploy the downstream cluster.

1.3.4.1 Image configuration

When running Edge Image Builder, a directory is mounted from the host, so it is necessary to create a directory structure to store the configuration files used to define the target image.

- `downstream-cluster-config.yaml` is the image definition file, see [Chapter 3, Standalone clusters with Edge Image Builder](#) for more details.
- The base image when downloaded is `xz` compressed, which must be uncompressed with `unxz` and copied/moved under the `base-images` folder.
- The `network` folder is optional, see [Section 1.3.5.1.1, "Additional script for static network configuration"](#) for more details.
- The `custom/scripts` directory contains scripts to be run on first-boot; currently a `growfs.sh` script is required to resize the OS root partition on deployment

```
├─ downstream-cluster-config.yaml
├─ base-images/
```

```
|   └─ SLE-Micro.x86_64-5.5.0-Default-GM.raw
└─ network/
|   └─ configure-network.sh
└─ custom/
    └─ scripts/
        └─ growfs.sh
```

1.3.4.1.1 Downstream cluster image definition file

The `downstream-cluster-config.yaml` file is the main configuration file for the downstream cluster image. The following is a minimal example for deployment via Metal³:

```
apiVersion: 1.0
image:
  imageType: RAW
  arch: x86_64
  baseImage: SLE-Micro.x86_64-5.5.0-Default-GM.raw
  outputImageName: SLE-Micro-eib-output.raw
operatingSystem:
  kernelArgs:
    - ignition.platform.id=openstack
    - net.ifnames=1
  systemd:
    disable:
      - rebootmgr
  users:
    - username: root
      encryptedPassword: ${ROOT_PASSWORD}
      sshKeys:
        - ${USERKEY1}
```

`${ROOT_PASSWORD}` is the encrypted password for the root user, which can be useful for test/debugging. It can be generated with the `openssl passwd -6 PASSWORD` command

For the production environments, it is recommended to use the SSH keys that can be added to the users block replacing the `${USERKEY1}` with the real SSH keys.



Note

`net.ifnames=1` enables [Predictable Network Interface Naming](https://documentation.suse.com/smart/network/html/network-interface-predictable-naming/index.html) (<https://documentation.suse.com/smart/network/html/network-interface-predictable-naming/index.html>) [↗](#)

This matches the default configuration for the metal3 chart, but the setting must match the configured chart `predictableNicNames` value.

Also note `ignition.platform.id=openstack` is mandatory, without this argument SLEMicro configuration via ignition will fail in the Metal³ automated flow.

1.3.4.1.2 Growfs script

Currently is a custom script (`custom/scripts/growfs.sh`) which is required to grow the file system to the match the disk size on first-boot after provisioning. The `growfs.sh` script contains the following information:

```
#!/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 /
```



Note

Add your own custom scripts to be executed during the provisioning process using the same approach. For more information, see [Chapter 3, Standalone clusters with Edge Image Builder](#).

1.3.4.2 Image creation

Once the directory structure is prepared following the previous sections, run the following command to build the image:

```
podman run --rm --privileged -it -v $PWD:/eib \
registry.suse.com/edge/edge-image-builder:1.0.2 \
build --definition-file downstream-cluster-config.yaml
```

This creates the output image file named `SLE-Micro-eib-output.raw`, based on the definition described above.

The output image must then be made available via a webserver, either the media-server container enabled via the Metal3 chart (*Note*) or some other locally accessible server. In the examples below, we refer to this server as imagecache.local:8080

1.3.5 Adding BareMetalHost inventory

Registering bare-metal servers for automated deployment requires creating two resources: a Secret storing BMC access credentials and a Metal³ BareMetalHost resource defining the BMC connection and other details:

```
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
```

Note the following:

- The Secret username/password must be base64 encoded. Note this should not include any trailing newlines (for example, use `echo -n`, not just `echo`!)
- The `cluster-role` label may be set now or later on cluster creation. In the example below, we expect `control-plane` or `worker`
- `bootMACAddress` must be a valid MAC that matches the control plane NIC of the host

- The `bmc` address is the connection to the BMC management API, the following are supported:
 - `redfish-virtualmedia://<IP ADDRESS>/redfish/v1/Systems/<SYSTEM ID>`: Redfish virtual media, for example, SuperMicro
 - `idrac-virtualmedia://<IP ADDRESS>/redfish/v1/Systems/System.Embedded.1`: Dell iDRAC
- See the [Upstream API docs \(https://github.com/metal3-io/baremetal-operator/blob/main/docs/api.md\)](https://github.com/metal3-io/baremetal-operator/blob/main/docs/api.md) for more details on the BareMetalHost API

1.3.5.1 Configuring Static IPs

The BareMetalHost example above assumes DHCP provides the controlplane network configuration, but for scenarios where manual configuration is needed such as static IPs it is possible to provide additional configuration, as described below.

1.3.5.1.1 Additional script for static network configuration

When creating the base image with Edge Image Builder, in the `network` folder, create the following `configure-network.sh` file.

This consumes configuration drive data on first-boot, and configures the host networking using the [NM Configurator tool \(https://github.com/suse-edge/nm-configurator\)](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 Additional secret with host network configuration

An additional secret containing data in the [nmstate \(https://nmstate.io/\)](https://nmstate.io/) format supported by NM Configurator (*Chapter 10, Edge Networking*) can be defined for each host.

The secret is then referenced in the `BareMetalHost` resource via the `preprovisioningNetworkDataName` spec field.

```

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:
  provisioningNetworkDataName: controlplane-0-networkdata
# Remaining content as in previous example

```



Note

In some circumstances the `mac-address` may be omitted but the `configure-network.sh` script must use the `_all.yaml` filename described above to enable Unified node configuration ([Section 10.5.8, “Unified node configurations”](#)) in `nm-configurator`.

1.3.5.2 BareMetalHost preparation

After creating the `BareMetalHost` resource and associated secrets as described above, a host preparation workflow is triggered:

- A ramdisk image is booted by `virtualmedia` attachment to the target host BMC
- The ramdisk inspects hardware details, and prepares the host for provisioning (for example by cleaning disks of previous data)
- On completion of this process, hardware details in the `BareMetalHost` `status.hardware` field are updated and can be verified

This process can take several minutes, but when completed you should see the `BareMetalHost` state become `available`:

```

% kubectl get baremetalhost
NAME           STATE      CONSUMER  ONLINE  ERROR  AGE
controlplane-0  available             true     9m44s

```

worker-0	available	true	9m44s
----------	-----------	------	-------

1.3.6 Creating downstream clusters

We now create Cluster API resources which define the downstream cluster, and Machine resources which will cause the BareMetalHost resources to be provisioned, then bootstrapped to form an RKE2 cluster.

1.3.7 Control plane deployment

To deploy the controlplane we define a yaml manifest similar to the one below, which contains the following resources:

- Cluster resource defines the cluster name, networks, and type of controlplane/infrastructure provider (in this case RKE2/Metal3)
- Metal3Cluster defines the controlplane endpoint (host IP for single-node, LoadBalancer endpoint for multi-node, this example assumes single-node)
- RKE2ControlPlane defines the RKE2 version and any additional configuration needed during cluster bootstrapping
- Metal3MachineTemplate defines the OS Image to be applied to the BareMetalHost resources, and the hostSelector defines which BareMetalHosts to consume
- Metal3DataTemplate defines additional metaData to be passed to the BareMetalHost (note networkData is not currently supported in the Edge solution)

Note for simplicity this example assumes a single-node controlplane, where the BareMetalHost is configured with an IP of `192.168.125.200` - for more advanced multi-node examples please see the ATIP documentation ([Chapter 31, Fully automated directed network provisioning](#))

```
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/v1alpha1
      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/v1alpha1
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
  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
    version: v1.28.13+rke2r1
  ---
  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

```

When the example above has been copied and adapted to suit your environment, it can be applied via `kubectl` then the cluster status can be monitored with `clusterctl`

```

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

# Wait for the cluster to be provisioned - status can be checked via clusterctl
% clusterctl describe cluster sample-cluster

```

NAME	READY	SEVERITY	REASON	SINCE
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 Worker/Compute deployment

Similar to the controlplane we define a yaml manifest, which contains the following resources:

- MachineDeployment defines the number of replicas (hosts) and the bootstrap/infrastructure provider (in this case RKE2/Metal3)
- RKE2ConfigTemplate describes the RKE2 version and first-boot configuration for agent host bootstrapping
- Metal3MachineTemplate defines the OS Image to be applied to the BareMetalHost resources, and the hostSelector defines which BareMetalHosts to consume
- Metal3DataTemplate defines additional metaData to be passed to the BareMetalHost (note networkData is not currently supported in the Edge solution)

```

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: v1.28.13+rke2r1
---
apiVersion: bootstrap.cluster.x-k8s.io/v1alpha1
kind: RKE2ConfigTemplate
metadata:
  name: sample-cluster-workers
  namespace: default
spec:
  template:
    spec:
      agentConfig:
        format: ignition
        version: v1.28.13+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
```

When the example above has been copied and adapted to suit your environment, it can be applied via `kubectl` then the cluster status can be monitored with `clusterctl`

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

# Wait some time for the compute/agent hosts to be provisioned
% clusterctl describe cluster sample-cluster
```

NAME	READY	SEVERITY	REASON	SINCE
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 Cluster deprovisioning

The downstream cluster may be deprovisioned by deleting the resources applied in the creation steps above:

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

This triggers deprovisioning of the BareMetalHost resources, which may take several minutes, after which they should be in available state again:

```
% kubectl get bmh
```

NAME	STATE	CONSUMER	ONLINE	ERROR
controlplane-0	deprovisioning	sample-cluster-controlplane-vlrt6	false	
worker-0	deprovisioning	sample-cluster-workers-785x5	false	
...				

```
% kubectl get bmh
```

NAME	STATE	CONSUMER	ONLINE	ERROR	AGE
------	-------	----------	--------	-------	-----

controlplane-0	available	false	15m
worker-0	available	false	15m

1.4 Known issues

- The [upstream IP Address Management controller \(https://github.com/metal3-io/ip-address-manager\)](https://github.com/metal3-io/ip-address-manager) is currently not supported, because it's not yet compatible with our choice of network configuration tooling and first-boot toolchain in SLEMicro.
- Relatedly, the IPAM resources and Metal3DataTemplate networkData fields are not currently supported.
- Only deployment via redfish-virtualmedia is currently supported.
- Deployed clusters are not currently imported into Rancher
- Due to disabling the Rancher embedded CAPI controller, a management cluster configured for Metal³ as described above cannot also be used for other cluster provisioning methods such as Elemental (*Chapter 11, Elemental*)

1.5 Planned changes

- Deployed clusters imported into Rancher, this is planned via [Rancher Turtles \(https://turtles.docs.rancher.com/\)](https://turtles.docs.rancher.com/) in future
- Aligning with Rancher Turtles is also expected to remove the requirement to disable the Rancher embedded CAPI, so other cluster methods should be possible via the management cluster.
- Enable support of the IPAM resources and configuration via networkData fields

1.6 Additional resources

The ATIP Documentation (*Chapter 26, SUSE Adaptive Telco Infrastructure Platform (ATIP)*) has examples of more advanced usage of Metal³ for telco use-cases.

1.6.1 Single-node configuration

For test/PoC environments where the management cluster is a single node, it is possible to avoid the requirement for an additional floating IP managed via MetalLB.

In this mode, the endpoint for the management cluster APIs is the IP of the management cluster, therefore it should be reserved when using DHCP or statically configured to ensure the management cluster IP does not change - referred to as `<MANAGEMENT_CLUSTER_IP>` below.

To enable this scenario the metal3 chart values required are as follows:

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

1.6.2 Disabling TLS for virtualmedia ISO attachment

Some server vendors verify the SSL connection when attaching virtual-media ISO images to the BMC, which can cause a problem because the generated certificates for the Metal3 deployment are self-signed, to work around this issue it's possible to disable TLS only for the virtualmedia disk attachment with metal3 chart values as follows:

```
global:
  enable_vmedia_tls: false
```

An alternative solution is to configure the BMCs with the CA cert - in this case you can read the certificates from the cluster using `kubectl`:

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

The certificate can then be configured on the server BMC console, although the process for that is vendor specific (and not possible for all vendors, in which case the `enable_vmedia_tls` flag may be required).

2 Remote host onboarding with Elemental

This section documents the "phone home network provisioning" solution as part of SUSE Edge, where we use Elemental to assist with node onboarding. Elemental is a software stack enabling remote host registration and centralized full cloud-native OS management with Kubernetes. In the SUSE Edge stack we use the registration feature of Elemental to enable remote host onboarding into Rancher so that hosts can be integrated into a centralized management platform and from there, deploy and manage Kubernetes clusters along with layered components, applications, and their lifecycle, all from a common place.

This approach can be useful in scenarios where the devices that you want to control are not on the same network as the upstream cluster or do not have a out-of-band management controller onboard to allow more direct control, and where you're booting many different "unknown" systems at the edge, and need to securely onboard and manage them at scale. This is a common scenario for use cases in retail, industrial IoT, or other spaces where you have little control over the network your devices are being installed in.

2.1

Edge Location

Edge Server

Installs

rancher-system-agent

Installation/Upgrade

K3s/RKE2

SLE

2.2 Resources needed

The following describes the minimum system and environmental requirements to run through this quickstart:

- A host for the centralized management cluster (the one hosting Rancher and Elemental):
 - Minimum 8 GB RAM and 20 GB disk space for development or testing (see [here](https://ranchermanager.docs.rancher.com/pages-for-subheaders/installation-requirements#hardware-requirements) (<https://ranchermanager.docs.rancher.com/pages-for-subheaders/installation-requirements#hardware-requirements>) [↗] for production use)
- A target node to be provisioned, i.e. the edge device (a virtual machine can be used for demoing or testing purposes)
 - Minimum 4GB RAM, 2 CPU cores, and 20 GB disk
- A resolvable host name for the management cluster or a static IP address to use with a service like `sslip.io`
- A host to build the installation media via Edge Image Builder
 - Running SLES 15 SP5, openSUSE Leap 15.5, or another compatible operating system that supports Podman.
 - With `Kubectl` (<https://kubernetes.io/docs/reference/kubectl/kubectl/>) [↗], `Podman` (<https://podman.io>) [↗], and `Helm` (<https://helm.sh>) [↗] installed
- A USB flash drive to boot from (if using physical hardware)



Note

Existing data found on target machines will be overwritten as part of the process, please make sure you backup any data on any USB storage devices and disks attached to target deployment nodes.

This guide is created using a Digital Ocean droplet to host the upstream cluster and an Intel NUC as the downstream device. For building the installation media, SUSE Linux Enterprise Server is used.

2.3 How to use Elemental

The basic steps to install and use Elemental are:

- *Section 2.3.1, "Build bootstrap cluster"*
- *Section 2.3.2, "Install Rancher"*
- *Section 2.3.3, "Install Elemental"*
- *Section 2.3.4, "Build the installation media"*
- *Section 2.3.5, "Boot the downstream nodes"*
- *Section 2.3.6, "Create downstream clusters"*

2.3.1 Build bootstrap cluster

Start by creating a cluster capable of hosting Rancher and Elemental. This cluster needs to be routable from the network that the downstream nodes are connected to.

2.3.1.1 Create Kubernetes cluster

If you are using a hyperscaler (such as Azure, AWS or Google Cloud), the easiest way to set up a cluster is using their built-in tools. For the sake of conciseness in this guide, we do not detail the process of each of these options.

If you are installing onto bare-metal or another hosting service where you need to also provide the Kubernetes distribution itself, we recommend using [RKE2 \(https://docs.rke2.io/install/quick-start\)](https://docs.rke2.io/install/quick-start).

2.3.1.2 Set up DNS

Before continuing, you need to set up access to your cluster. As with the setup of the cluster itself, how you configure DNS will be different depending on where it is being hosted.



Tip

If you do not want to handle setting up DNS records (for example, this is just an ephemeral test server), you can use a service like sslip.io (<https://sslip.io>) instead. With this service, you can resolve any IP address with `<address>.sslip.io`.

2.3.2 Install Rancher

To install Rancher, you need to get access to the Kubernetes API of the cluster you just created. This looks differently depending on what distribution of Kubernetes is being used.

For RKE2, the kubeconfig file will have been written to `/etc/rancher/rke2/rke2.yaml`. Save this file as `~/.kube/config` on your local system. You may need to edit the file to include the correct externally routable IP address or host name.

Install Rancher easily with the commands from the [Rancher Documentation](https://ranchermanager.docs.rancher.com/pages-for-subheaders/install-upgrade-on-a-kubernetes-cluster) (<https://ranchermanager.docs.rancher.com/pages-for-subheaders/install-upgrade-on-a-kubernetes-cluster>):

1. Install `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
```

2. Then install Rancher itself:

```
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 2.8.8
```



Note

If this is intended to be a production system, please use cert-manager to configure a real certificate (such as one from Let's Encrypt).

Browse to the host name you set up and log in to Rancher with the bootstrapPassword you used. You will be guided through a short setup process.

2.3.3 Install Elemental

With Rancher installed, you can now install the Elemental operator and required CRD's. The Helm chart for Elemental is published as an OCI artifact so the installation is a little simpler than other charts. It can be installed from either the same shell you used to install Rancher or in the browser from within Rancher's shell.

```
helm install --create-namespace -n cattle-elemental-system \
  elemental-operator-crds \
  oci://registry.suse.com/rancher/elemental-operator-crds-chart \
  --version 1.4.4

helm install -n cattle-elemental-system \
  elemental-operator \
  oci://registry.suse.com/rancher/elemental-operator-chart \
  --version 1.4.4
```


2.3.3.1 (Optionally) Install the Elemental UI extension

3. Confirm that you want to install the extension:

4. After it installs, you will be prompted to reload the page.



Extensions

Installed

Available

Updates

All




Elemental
OS Management extension
1.2.0


Uninstall

5. Once you reload, you can access the Elemental extension through the "OS Management" global app.




 Home

EXPLORE CLUSTER


 local

GLOBAL APPS

 Continuous Delivery


 Cluster Management


 OS Management

 Virtualization Management

CONFIGURATION

 Users & Authentication

 Extensions

 Global Settings

updates

All

ion

Uninstall

2.3.3.2 Configure Elemental

For simplicity, we recommend setting the variable `$ELEM` to the full path of where you want the configuration directory:

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

To allow machines to register to Elemental, we need to create a `MachineRegistration` object in the `fleet-default` namespace.

Let us create a basic version of this object:

```
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
```



Note

The `cat` command escapes each `$` with a backslash (`\`) so that Bash does not template them. Remove the backslashes if copying manually.

Once the object is created, find and note the endpoint that gets assigned:

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

Alternatively, this can also be done from the UI.

Registration Endpoints

There are currently no

[Create a new](#)

2. Give this configuration a name.

v2.7.6



Dashboard

Registration Endpoints 0

Inventory of Machines 0

Advanced

Registration Endpoint

Configuration

Name*

ele-quickstart-nodes

Cloud Configuration

```
1 config:
2   cloud-config:
3     users:
4       - name: root
5         passwd: root
6   elemental:
7     install:
8       poweroff: true
9       device: /dev/n
```

Read from File

Labels And Annotations

Inventory of Machines

Regist

Install Elemental

Labels and annotations to be added to Machines when creating cluster



Note

You can ignore the Cloud Configuration field as the data here is overridden by the following steps with Edge Image Builder.

3. Next, scroll down and click "Add Label" for each label you want to be on the resource that gets created when a machine registers. This is useful for distinguishing machines.



Dashboard

Registration Endpoints 0

Inventory of Machines 0

Advanced

Read from File

Labels And Annotations

Inventory of Machines

Registration Endpoints

Labels and annotations to be added to the **Machines** when creating cluster. For reference on SMBIOS data c

Labels

Key

manufacturer

productName

Add Label

Annotations

Add Annotation

4. Lastly, click "Create" to save the configuration.



Dashboard

Registration Endpoints 0

Inventory of Machines 0

Advanced

Read from File

Labels And Annotations

Inventory of Machines

Registration

Labels and annotations to be added to the **Machines** when creating cluster. For reference on SMBIOS data c

Labels

Key

manufacturer

productName

Add Label

Annotations

Add Annotation

Registration URL (ends with regis

Registration URL

`https://rancher.gracey.dev/e1mq7vvf4cw29q6dgxw27r7h`

Build ISO image

OS Version

Elemental Teal ISO x86_64 v1....

Setting up an OS image

Download the Registration Endpoint Conf

Download Configuration File

Cloud Configuration

```

▼ config:
  ▼ cloud-config:
    ▼ users:
      - name: root
        passwd: root
    ▼ elemental:
    ▼ install:
      device: /dev/nvme0n1
      poweroff: true
  ▼ machineInventoryLabels:
    manufacturer: ${System Info
    productName: ${System Infor

```



Tip

If you clicked away from that screen, you can click "Registration Endpoints" in the left menu, then click the name of the endpoint you just created.

This URL is used in the next step.

2.3.4 Build the installation media

While the current version of Elemental has a way to build its own installation media, in SUSE Edge 3.0 we do this with the Edge Image Builder instead, so the resulting system is built with [SLE Micro \(https://www.suse.com/products/micro/\)](https://www.suse.com/products/micro/) as the base Operating System.



Tip

For more details on the Edge Image Builder, check out the Getting Started Guide for it ([Chapter 3, Standalone clusters with Edge Image Builder](#)) and also the Component Documentation ([Chapter 9, Edge Image Builder](#)).

From a Linux system with Podman installed, run:

```
mkdir -p $ELEM/eib_quickstart/base-images
mkdir -p $ELEM/eib_quickstart/elemental
```

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

```
cat << EOF > $ELEM/eib_quickstart/eib-config.yaml
apiVersion: 1.0
image:
  imageType: iso
  arch: x86_64
  baseImage: SLE-Micro.x86_64-5.5.0-Default-SelfInstall-GM2.install.iso
  outputImageName: elemental-image.iso
operatingSystem:
  isoConfiguration:
    installDevice: /dev/vda
  users:
    - username: root
      encryptedPassword: \$$\jHugJNnd3HElGsUZ\
$eodjVe4te5ps44SVcWshdfWizrP.xAyd71CVEXazBJ/.v799/WRCBXxfYmunlB02yp1hm/zb4r8EmnrrNCF.P/
```



Note

- The unencoded password is `eib`.
- The `cat` command escapes each `$` with a backslash (`\`) so that Bash does not template them. Remove the backslashes if copying manually.
- The installation device will be wiped during the installation.

```
podman run --privileged --rm -it -v $ELEM/eib_quickstart:/eib \
registry.suse.com/edge/edge-image-builder:1.0.2 \
build --definition-file eib-config.yaml
```

If you are booting a physical device, we need to burn the image to a USB flash drive. This can be done with:

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

2.3.5 Boot the downstream nodes

Now that we have created the installation media, we can boot our downstream nodes with it.

For each of the systems that you want to control with Elemental, add the installation media and boot the device. After installation, it will reboot and register itself.

If you are using the UI extension, you should see your node appear in the "Inventory of Machines."



Note

Do not remove the installation medium until you've seen the login prompt; during first-boot files are still accessed on the USB stick.

2.3.6 Create downstream clusters

There are two objects we need to create when provisioning a new cluster using Elemental.

Linux

The first is the `MachineInventorySelectorTemplate`. This object allows us to specify a mapping between clusters and the machines in the inventory.

1. Create a selector which will match any machine in the inventory with a label:

```
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. Apply the resource to the cluster:

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

3. Obtain the name of the machine and add the matching label:

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

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

4. Create a simple single-node K3s cluster resource and apply it to the cluster:

```
cat << EOF > $ELEM/cluster.yaml
apiVersion: provisioning.cattle.io/v1
kind: Cluster
metadata:
  name: location-123
  namespace: fleet-default
spec:
  kubernetesVersion: v1.28.13+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
```

UI Extension

The UI extension allows for a few shortcuts to be taken. Note that managing multiple locations may involve too much manual work.

1. As before, open the left three-dot menu and select "OS Management." This brings you back to the main screen for managing your Elemental systems.
2. On the left sidebar, click "Inventory of Machines." This opens the inventory of machines that have registered.
3. To create a cluster from these machines, select the systems you want, click the "Actions" drop-down list, then "Create Elemental Cluster." This opens the Cluster Creation dialog while also creating a MachineSelectorTemplate to use in the background.
4. On this screen, configure the cluster you want to be built. For this quick start, K3s v1.28.13 + k3s1 is selected and the rest of the options are left as is.



Tip

You may need to scroll down to see more options.

After creating these objects, you should see a new Kubernetes cluster spin up using the new node you just installed with.



Tip

To allow for easier grouping of systems, you could add a startup script that finds something in the environment that is known to be unique to that location.

For example, if you know that each location will have a unique subnet, you can write a script that finds the network prefix and adds a label to the corresponding MachineInventory.

This would typically be custom to your system's design but could look like:

```
INET=`ip addr show dev eth0 | grep "inet\ "`
elemental-register --label "network=$INET" \
  --label "network=$INET" /oem/registration
```

2.4 Node Reset

SUSE Rancher Elemental supports the ability to perform a "node reset" which can optionally trigger when either a whole cluster is deleted from Rancher, a single node is deleted from a cluster, or a node is manually deleted from the machine inventory. This is useful when you want to reset and clean-up any orphaned resources and want to automatically bring the cleaned node back into the machine inventory so it can be reused. This is not enabled by default, and thus any system that is removed, will not be cleaned up (i.e. data will not be removed, and any Kubernetes cluster resources will continue to operate on the downstream clusters) and it will require manual intervention to wipe data and re-register the machine to Rancher via Elemental. If you wish for this functionality to be enabled by default, you need to make sure that your `MachineRegistration` explicitly enables this by adding `config.elemental.reset.enabled: true`, for example:

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

Then, all systems registered with this `MachineRegistration` will automatically receive the `elemental.cattle.io/resettable: 'true'` annotation in their configuration. If you wish to do this manually on individual nodes, e.g. because you've got an existing `MachineInventory` that doesn't have this annotation, or you have already deployed nodes, you can modify the `MachineInventory` and add the `resettable` configuration, for example:

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

In SUSE Edge 3.0, the Elemental Operator puts down a marker on the operating system that will trigger the cleanup process automatically; it will stop all Kubernetes services, remove all persistent data, uninstall all Kubernetes services, cleanup any remaining Kubernetes/Rancher directories, and force a re-registration to Rancher via the original Elemental `MachineRegistration` configuration. This happens automatically, there is no need for any manual intervention. The script that gets called can be found in `/opt/edge/elemental_node_cleanup.sh` and is triggered via `systemd.path` upon the placement of the marker, so its execution is immediate.



Warning

Using the `resettable` functionality assumes that the desired behavior when removing a node/cluster from Rancher is to wipe data and force a re-registration. Data loss is guaranteed in this situation, so only use this if you're sure that you want automatic reset to be performed.

2.5 Next steps

Here are some recommended resources to research after using this guide:

- End-to-end automation in [Chapter 6, Fleet](#)
- Additional network configuration options in [Chapter 10, Edge Networking](#)

3 Standalone clusters with Edge Image Builder

Edge Image Builder (EIB) is a tool that streamlines the process of generating Customized, Ready-to-Boot (CRB) disk images for bootstrapping machines, even in fully air-gapped scenarios. EIB is used to create deployment images for use in all three of the SUSE Edge deployment footprints, as it's flexible enough to offer the smallest customizations, e.g. adding a user or setting the timezone, through offering a comprehensively configured image that sets up, for example, complex networking configurations, deploys multi-node Kubernetes clusters, deploys customer workloads, and registers to the centralized management platform via Rancher/Elemental and SUSE Manager. EIB runs as in a container image, making it incredibly portable across platforms and ensuring that all of the required dependencies are self-contained, having a very minimal impact on the installed packages of the system that's being used to operate the tool.

For more information, read the Edge Image Builder Introduction ([Chapter 9, Edge Image Builder](#)).

3.1 Prerequisites

- An x86_64 physical host (or virtual machine) running SLES 15 SP5, openSUSE Leap 15.5, or openSUSE Tumbleweed.
- An available container runtime (e.g. Podman)
- A downloaded copy of the latest SLE Micro 5.5 SelfInstall "GM2" ISO image found [here](https://www.suse.com/download/sle-micro/) (<https://www.suse.com/download/sle-micro/>) ↗.



Note

Other operating systems may function so long as a compatible container runtime is available, but testing on other platforms has not been extensive. The documentation focuses on Podman, but the same functionality should be able to be achieved with Docker.

3.1.1 Getting the EIB Image

The EIB container image is publicly available and can be downloaded from the SUSE Edge registry by running the following command on your image build host:

```
podman pull registry.suse.com/edge/edge-image-builder:1.0.2
```

3.2 Creating the image configuration directory

As EIB runs within a container, we need to mount a configuration directory from the host, enabling you to specify your desired configuration, and during the build process EIB has access to any required input files and supporting artifacts. This directory must follow a specific structure. Let's create it, assuming that this directory will exist in your home directory, and called "eib":

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

In the previous step we created a "base-images" directory that will host the SLE Micro 5.5 input image, let's ensure that the downloaded image is copied over to the configuration directory:

```
cp /path/to/downloads/SLE-Micro.x86_64-5.5.0-Default-SelfInstall-GM2.install.iso
$CONFIG_DIR/base-images/slemicro.iso
```



Note

During the EIB run, the original base image is **not** modified; a new and customized version is created with the desired configuration in the root of the EIB config directory.

The configuration directory at this point should look like the following:

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

3.3 Creating the image definition file

The definition file describes the majority of configurable options that the Edge Image Builder supports, a full example of options can be found [here \(https://github.com/suse-edge/edge-image-builder/blob/release-1.0/pkg/image/testdata/full-valid-example.yaml\)](https://github.com/suse-edge/edge-image-builder/blob/release-1.0/pkg/image/testdata/full-valid-example.yaml), and we would recommend that you take a look at the [upstream building images guide \(https://github.com/suse-edge/edge-image-builder/blob/release-1.0/docs/building-images.md\)](https://github.com/suse-edge/edge-image-builder/blob/release-1.0/docs/building-images.md) for more comprehensive examples than the one we're going to run through below. Let's start with a very basic definition file for our OS image:

```
cat << EOF > $CONFIG_DIR/iso-definition.yaml
apiVersion: 1.0
image:
```

```
imageType: iso
arch: x86_64
baseImage: slemicro.iso
outputImageName: eib-image.iso
EOF
```

This definition specifies that we're generating an output image for an `x86_64` based system. The image that will be used as the base for further modification is an `iso` image named `slemicro.iso`, expected to be located at `$CONFIG_DIR/base-images/slemicro.iso`. It also outlines that after EIB finishes modifying the image, the output image will be named `eib-image.iso`, and by default will reside in `$CONFIG_DIR`.

Now our directory structure should look like:

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

In the following sections we'll walk through a few examples of common operations:

3.3.1 Configuring OS Users

EIB allows you to preconfigure users with login information, such as passwords or SSH keys, including setting a fixed root password. As part of this example we're going to fix the root password, and the first step is to use `OpenSSL` to create a one-way encrypted password:

```
openssl passwd -6 SecurePassword
```

This will output something similar to:

```
$6$G392FCbxVgnLaFw1$Ujt00mdpJ3tDHxEg1snBU3GjujQf6f8kvopu7jiCBIhRbRvMmKUqwcmXAKggaSSKeUU0EtCP3ZUoZQY7zTX
```

We can then add a section in the definition file called `operatingSystem` with a `users` array inside it. The resulting file should look like:

```
apiVersion: 1.0
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
```

```
- username: root
  encryptedPassword:
$6$G392FCbxVgnLaFw1$Ujt00mdpJ3tDHxEg1snBU3GjujQf6f8kvopu7jiCBIhRbRvMmKUqwcmXAKggaSSKeUU0EtCP3ZUoZQY7zT
```



Note

It's also possible to add additional users, create the home directories, set user-id's, add ssh-key authentication, and modify group information. Please refer to the [upstream building images guide \(https://github.com/suse-edge/edge-image-builder/blob/release-1.0/docs/building-images.md\)](https://github.com/suse-edge/edge-image-builder/blob/release-1.0/docs/building-images.md) for further examples.

3.3.2 Configuring RPM packages

One of the major features of EIB is to provide a mechanism to add additional software packages to the image, so when the installation completes the system is able to leverage the installed packages right away. EIB permits users to specify the following:

- Packages by their name within a list in the image definition
- Network repositories to search for these packages in
- SUSE Customer Center (SCC) credentials to search official SUSE repositories for the listed packages
- Via an `$CONFIG_DIR/rpms` directory, side-load custom RPM's that don't exist in network repositories
- Via the same directory (`$CONFIG_DIR/rpms/gpg-keys`), GPG-keys to enable validation of third party packages

EIB will then run through a package resolution process at image build time, taking the base image as the input, and attempts to pull and install all supplied packages, either specified via the list or provided locally. EIB downloads all of the packages, including any dependencies into a repository that exists within the output image and instructs the system to install these during the first boot process. Doing this process during the image build guarantees that the packages will successfully install during first-boot on the desired platform, e.g. the node at the edge. This is also advantageous in environments where you want to bake the additional packages into the image rather than pull them over the network when in operation, e.g. for air-gapped or restricted network environments.

As a simple example to demonstrate this, we are going to install the `nvidia-container-toolkit` RPM package found in the third party vendor-supported NVIDIA repository:

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

The resulting definition file looks like:

```
apiVersion: 1.0
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword:
$6$G392FCbxVgnLaFw1$Ujt00mdpJ3tDHxEg1snBU3GjujQf6f8kvopu7jiCBIhRbRvMmKUqwcmXAKggaSSKeUU0EtCP3ZUoZQY7zT
  packages:
    packageList:
      - nvidia-container-toolkit
    additionalRepos:
      - url: https://nvidia.github.io/libnvidia-container/stable/rpm/x86_64
```

The above is a simple example, but for completeness, download the NVIDIA package signing key before running the image generation:

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



Warning

Adding in additional RPM's via this method is meant for the addition of supported third party components or user-supplied (and maintained) packages; this mechanism should not be used to add packages that would not usually be supported on SLE Micro. If this mechanism is used to add components from openSUSE repositories (which are not supported), including from newer releases or service packs, you may end up with an unsupported configuration, especially when dependency resolution results in core parts of the

operating system being replaced, even though the resulting system may appear to function as expected. If you're unsure, contact your SUSE representative for assistance in determining the supportability of your desired configuration.



Note

A more comprehensive guide with additional examples can be found in the [upstream installing packages guide \(https://github.com/suse-edge/edge-image-builder/blob/release-1.0/docs/installing-packages.md\)](https://github.com/suse-edge/edge-image-builder/blob/release-1.0/docs/installing-packages.md).

3.3.3 Configuring Kubernetes cluster and user workloads

Another feature of EIB is the ability to use it to automate the deployment of both single-node and multi-node highly-available Kubernetes clusters that "bootstrap in place", i.e. don't require any form of centralized management infrastructure to coordinate. The primary driver behind this approach is for air-gapped deployments, or network restricted environments, but it also serves as a way of quickly bootstrapping standalone clusters, even if full and unrestricted network access is available.

This method enables not only the deployment of the customized operating system, but also the ability to specify Kubernetes configuration, any additional layered components via Helm charts, and any user workloads via supplied Kubernetes manifests. However, the design principle behind using this method is that we default to assuming that the user is wanting to air-gap and therefore any items specified in the image definition will be pulled into the image, which includes user-supplied workloads, where EIB will make sure that any discovered images that are required by definitions supplied are copied locally, and are served by the embedded image registry in the resulting deployed system.

In this next example, we're going to take our existing image definition and will specify a Kubernetes configuration (in this example it doesn't list the systems and their roles, so we default to assuming single-node), which will instruct EIB to provision a single-node RKE2 Kubernetes cluster. To show the automation of both the deployment of both user-supplied workloads (via manifest) and layered components (via Helm), we are going to install KubeVirt via the SUSE Edge Helm chart, as well as NGINX via a Kubernetes manifest. The additional configuration we need to append to the existing image definition is as follows:

```
kubernetes:
```



```

version: v1.28.13+rke2r1
manifests:
  urls:
    - https://k8s.io/examples/application/nginx-app.yaml
helm:
  charts:
    - name: kubevirt-chart
      version: 0.2.4
      repositoryName: suse-edge
  repositories:
    - name: suse-edge
      url: oci://registry.suse.com/edge

```

The resulting full definition file should now look like:

```

apiVersion: 1.0
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword:
$6$G392FCbxVgnLaFw1$Ujt00mdpJ3tDHxEg1snBU3GjujQf6f8kvopu7jiCBIhRbRvMmKUqwcMxAKggaSSKeUU0EtCP3ZUoZQY7zT
  packages:
    packageList:
      - nvidia-container-toolkit
    additionalRepos:
      - url: https://nvidia.github.io/libnvidia-container/stable/rpm/x86_64
kubernetes:
  version: v1.28.13+rke2r1
  manifests:
    urls:
      - https://k8s.io/examples/application/nginx-app.yaml
  helm:
    charts:
      - name: kubevirt-chart
        version: 0.2.4
        repositoryName: suse-edge
    repositories:
      - name: suse-edge
        url: oci://registry.suse.com/edge

```



Note

Further examples of options such as multi-node deployments, custom networking, and Helm chart options/values can be found in the [upstream documentation \(https://github.com/suse-edge/edge-image-builder/blob/release-1.0/docs/building-images.md#kubernetes\)](https://github.com/suse-edge/edge-image-builder/blob/release-1.0/docs/building-images.md#kubernetes).

3.3.4 Configuring the network

In the last example in this quickstart, let's configure the network that will be brought up when a system is provisioned with the image generated by EIB. It's important to understand that unless a network configuration is supplied, the default model is that DHCP will be used on all interfaces discovered at boot time. However, this is not always a desirable configuration, especially if DHCP is not available and you need to provide static configurations, or you need to set up more complex networking constructs, e.g. bonds, LACP, and VLAN's, or need to override certain parameters, e.g. hostnames, DNS servers, and routes.

EIB provides the ability to provide either per-node configurations (where the system in question is uniquely identified by its MAC address), or an override for supplying an identical configuration to each machine, which is more useful when the system MAC addresses aren't known. An additional tool is used by EIB called Network Manager Configurator, or `nmc` for short, which is a tool built by the SUSE Edge team to allow custom networking configurations to be applied based on the [nmstate.io \(https://nmstate.io/\)](https://nmstate.io/) declarative network schema, and at boot time will identify the node it's booting on and will apply the desired network configuration prior to any services coming up.

We'll now apply a static network configuration for a system with a single interface by describing the desired network state in a node-specific file (based on the desired hostname) in the required `network` directory:

```
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
```



Warning

The above example is set up for the default `192.168.122.0/24` subnet assuming that testing is being executed on a virtual machine, please adapt to suit your environment, not forgetting the MAC address. As the same image can be used to provision multiple nodes, networking configured by EIB (via `nmc`) is dependent on it being able to uniquely identify the node by its MAC address, and hence during boot `nmc` will apply the correct networking configuration to each machine. This means that you'll need to know the MAC addresses of the systems you want to install onto. Alternatively, the default behavior is to rely on DHCP, but you can utilize the `configure-network.sh` hook to apply a common configuration to all nodes - see the networking guide ([Chapter 10, Edge Networking](#)) for further details.

The resulting file structure should look like:

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

```
└─ network/
   └─ host1.local.yaml
```

The network configuration we just created will be parsed and the necessary NetworkManager connection files will be automatically generated and inserted into the new installation image that EIB will create. These files will be applied during the provisioning of the host, resulting in a complete network configuration.



Note

Please refer to the Edge Networking component ([Chapter 10, Edge Networking](#)) for a more comprehensive explanation of the above configuration and examples of this feature.

3.4 Building the image

Now that we've got a base image and an image definition for EIB to consume, let's go ahead and build the image. For this, we simply use `podman` to call the EIB container with the "build" command, specifying the definition file:

```
podman run --rm -it --privileged -v $CONFIG_DIR:/eib \
registry.suse.com/edge/edge-image-builder:1.0.2 \
build --definition-file iso-definition.yaml
```

The output of the command should be similar to:

```
Setting up Podman API listener...
Generating image customization components...
Identifier ..... [SUCCESS]
Custom Files ..... [SKIPPED]
Time ..... [SKIPPED]
Network ..... [SUCCESS]
Groups ..... [SKIPPED]
Users ..... [SUCCESS]
Proxy ..... [SKIPPED]
Resolving package dependencies...
Rpm ..... [SUCCESS]
Systemd ..... [SKIPPED]
Elemental ..... [SKIPPED]
Suma ..... [SKIPPED]
Downloading file: dl-manifest-1.yaml 100% (498/498 B, 5.9 MB/s)
Populating Embedded Artifact Registry... 100% (3/3, 11 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% (782/782 MB, 98 MB/s)
Downloading file: rke2-images-cilium.linux-amd64.tar.zst 100% (367/367 MB, 100 MB/s)
Downloading file: rke2.linux-amd64.tar.gz 100% (34/34 MB, 101 MB/s)
Downloading file: sha256sum-amd64.txt 100% (3.9/3.9 kB, 1.5 MB/s)
Downloading file: dl-manifest-1.yaml 100% (498/498 B, 7.2 MB/s)
Kubernetes ..... [SUCCESS]
Certificates ..... [SKIPPED]
Building ISO image...
Kernel Params ..... [SKIPPED]
Image build complete!

```

The built ISO image is stored at `$CONFIG_DIR/eib-image.iso`:

```

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

```

Each build creates a time-stamped folder in `$CONFIG_DIR/_build/` that includes the logs of the build, the artifacts used during the build, and the `combustion` and `artefacts` directories which contain all the scripts and artifacts that are added to the CRB image.

The contents of this directory should look like:

```

├─ 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:1.14.2-registry.tar.zst
| | | | |     └─ rancher_kubectrl:v1.28.7-registry.tar.zst
| | | | |     └─ registry.suse.com_suse_sles_15.5_virt-operator:1.1.1-150500.8.12.1-
registry.tar.zst
| | | | |   └─ rpms/
| | | | |     └─ rpm-repo
| | | | |       └─ addrepo0
| | | | |         └─ x86_64
| | | | |           └─ nvidia-container-toolkit-1.15.0-1.x86_64.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-chart.yaml
| | | | |   └─ createrepo.log
| | | | |   └─ eib-build.log
| | | | |   └─ embedded-registry.log
| | | | |   └─ helm
| | | | |     └─ kubevirt-chart
| | | | |       └─ kubevirt-0.2.4.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
   └─ ...
```

If the build fails, `eib-build.log` is the first log that contains information. From there, it will direct you to the component that failed for debugging.

At this point, you should have a ready-to-use image that will:

1. Deploy SLE Micro 5.5
2. Configure the root password
3. Install the `nvidia-container-toolkit` package
4. Configure an embedded container registry to serve content locally
5. Install single-node RKE2
6. Configure static networking
7. Install KubeVirt
8. Deploy a user-supplied manifest

3.5 Debugging the image build process

If the image build process fails, refer to the [upstream debugging guide \(https://github.com/suse-edge/edge-image-builder/blob/release-1.0/docs/debugging.md\)](https://github.com/suse-edge/edge-image-builder/blob/release-1.0/docs/debugging.md).

3.6 Testing your newly built image

For instructions on how to test the newly built CRB image, refer to the [upstream image testing guide \(https://github.com/suse-edge/edge-image-builder/blob/release-1.0/docs/testing-guide.md\)](https://github.com/suse-edge/edge-image-builder/blob/release-1.0/docs/testing-guide.md).

II Components Used

- 4 Rancher **62**
- 5 Rancher Dashboard Extensions **65**
- 6 Fleet **70**
- 7 SLE Micro **81**
- 8 Metal³ **83**
- 9 Edge Image Builder **84**
- 10 Edge Networking **86**
- 11 Elemental **109**
- 12 Akri **111**
- 13 K3s **120**
- 14 RKE2 **122**
- 15 Longhorn **125**
- 16 NeuVector **133**
- 17 MetalLB **135**
- 18 Edge Virtualization **137**

List of components for Edge

4 Rancher

See Rancher upstream documentation at <https://ranchermanager.docs.rancher.com> .

Rancher is a powerful open-source Kubernetes management platform that streamlines the deployment, operations and monitoring of Kubernetes clusters across multiple environments. Whether you manage clusters on premises, in the cloud, or at the edge, Rancher provides a unified and centralized platform for all your Kubernetes needs.

4.1 Key Features of Rancher

- **Multi-cluster management:** Rancher’s intuitive interface lets you manage Kubernetes clusters from anywhere—public clouds, private data centers and edge locations.
- **Security and compliance:** Rancher enforces security policies, role-based access control (RBAC), and compliance standards across your Kubernetes landscape.
- **Simplified cluster operations:** Rancher automates cluster provisioning, upgrades and troubleshooting, simplifying Kubernetes operations for teams of all sizes.
- **Centralized application catalog:** The Rancher application catalog offers a diverse range of Helm charts and Kubernetes Operators, making it easy to deploy and manage containerized applications.
- **Continuous delivery:** Rancher supports GitOps and CI/CD pipelines, enabling automated and streamlined application delivery processes.

4.2 Rancher’s use in SUSE Edge

Rancher provides several core functionalities to the SUSE Edge stack:

4.2.1 Centralized Kubernetes management

In typical edge deployments with numerous distributed clusters, Rancher acts as a central control plane for managing these Kubernetes clusters. It offers a unified interface for provisioning, upgrading, monitoring, and troubleshooting, simplifying operations, and ensuring consistency.

4.2.2 Simplified cluster deployment

Rancher streamlines Kubernetes cluster creation on the lightweight SLE Micro (SUSE Linux Enterprise Micro) operating system, easing the rollout of edge infrastructure with robust Kubernetes capabilities.

4.2.3 Application deployment and management

The integrated Rancher application catalog can simplify deploying and managing containerized applications across SUSE Edge clusters, enabling seamless edge workload deployment.

4.2.4 Security and policy enforcement

Rancher provides policy-based governance tools, role-based access control (RBAC), and integration with external authentication providers. This helps SUSE Edge deployments maintain security and compliance, critical in distributed environments.

4.3 Best practices

4.3.1 GitOps

Rancher includes Fleet as a built-in component to allow manage cluster configurations and application deployments with code stored in git.

4.3.2 Observability

Rancher includes built-in monitoring and logging tools like Prometheus and Grafana for comprehensive insights into your cluster health and performance.

4.4 Installing with Edge Image Builder

SUSE Edge is using [Chapter 9, Edge Image Builder](#) in order to customize base SLE Micro OS images. Follow [Section 21.6, "Rancher Installation"](#) for an air-gapped installation of Rancher on top of Kubernetes clusters provisioned by EIB.

4.5 Additional Resources

- [Rancher Documentation \(https://rancher.com/docs/\)](https://rancher.com/docs/) ↗
- [Rancher Academy \(https://www.rancher.academy/\)](https://www.rancher.academy/) ↗
- [Rancher Community \(https://rancher.com/community/\)](https://rancher.com/community/) ↗
- [Helm Charts \(https://helm.sh/\)](https://helm.sh/) ↗
- [Kubernetes Operators \(https://operatorhub.io/\)](https://operatorhub.io/) ↗

5 Rancher Dashboard Extensions

Extensions allow users, developers, partners, and customers to extend and enhance the Rancher UI. SUSE Edge 3.0 provides KubeVirt and Akri dashboard extensions.

See [Rancher documentation](#) for general information about Rancher Dashboard Extensions.

5.1 Prerequisites

To enable extensions Rancher requires ui-plugin operator to be installed. When using the Rancher Dashboard UI, navigate to **Extensions** in the left navigation **Configuration** section. If the ui-plugin operator is not installed you'll get a prompt asking to enable the extensions support as described [here](#).

The operator can be also installed using Helm:

```
helm repo add rancher-charts https://charts.rancher.io/
helm upgrade --create-namespace -n cattle-ui-plugin-system \
  --install ui-plugin-operator rancher-charts/ui-plugin-operator
helm upgrade --create-namespace -n cattle-ui-plugin-system \
  --install ui-plugin-operator-crd rancher-charts/ui-plugin-operator-crd
```

Or with Fleet by creating a dedicated GitRepo resource. For more information see Fleet ([Chapter 6, Fleet](#)) section and [fleet-examples](#) repository.

5.2 Installation

All SUSE Edge 3.0 components including dashboard extensions are distributed as OCI artifacts. Rancher Dashboard Apps/Marketplace does not support OCI based Helm repositories [yet](#). Therefore, to install SUSE Edge Extensions you can use Helm or Fleet:

5.2.1 Installing with Helm

```
# KubeVirt extension
helm install kubvirt-dashboard-extension oci://registry.suse.com/edge/kubvirt-
dashboard-extension-chart --version 1.0.0 --namespace cattle-ui-plugin-system
```

```
# Akri extension
helm install akri-dashboard-extension oci://registry.suse.com/edge/akri-dashboard-
extension-chart --version 1.0.0 --namespace cattle-ui-plugin-system
```



Note

The extensions need to be installed in `cattle-ui-plugin-system` namespace.



Note

After an extension is installed, Rancher Dashboard UI needs to be reloaded.

5.2.2 Installing with Fleet

Installing Dashboard Extensions with Fleet requires defining a `gitRepo` resource which points to a Git repository with custom `fleet.yaml` bundle configuration file(s).

```
# KubeVirt extension fleet.yaml
defaultNamespace: cattle-ui-plugin-system
helm:
  releaseName: kubvirt-dashboard-extension
  chart: oci://registry.suse.com/edge/akri-dashboard-extension-chart
  version: "1.0.0"
```

```
# Akri extension fleet.yaml
defaultNamespace: cattle-ui-plugin-system
helm:
  releaseName: akri-dashboard-extension
  chart: oci://registry.suse.com/edge/akri-dashboard-extension-chart
  version: "1.0.0"
```



Note

The `releaseName` property is required and needs to match the extension name to get the extension correctly installed by `ui-plugin-operator`.

```
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
```

For more information see Fleet ([Chapter 6, Fleet](#)) section and [fleet-examples](#) repository.

Once the Extensions are installed they are listed in **Extensions** section under **Installed** tabs. Since they are not installed via Apps/Marketplace, they are marked with Third-Party label.



Extensions

Installed

Available

Updates

All



Akri

SUSE Edge: Akri extension for Rancher Dashboard

1.0.0

Third-Party

Uninstall



KubeVirt

SUSE Edge: KubeVirt extension for Rancher Dashboard

1.0.0

Third-Party

Uninstall

5.3 KubeVirt Dashboard Extension

KubeVirt Extension provides basic virtual machine management for Rancher dashboard UI. Its capabilities are described in Using KubeVirt Rancher Dashboard Extension ([Section 18.7.2, “Using KubeVirt Rancher Dashboard Extension”](#)).

5.4 Akri Dashboard Extension

Akri is a Kubernetes Resource Interface that lets you easily expose heterogeneous leaf devices (such as IP cameras and USB devices) as resources in a Kubernetes cluster, while also supporting the exposure of embedded hardware resources such as GPUs and FPGAs. Akri continually detects nodes that have access to these devices and schedules workloads based on them.

Akri Dashboard Extension allows you to use Rancher Dashboard user interface to manage and monitor leaf devices and run workloads once these devices are discovered.

Extension capabilities are further described in Akri section ([Section 12.1.4, “Akri Rancher Dashboard Extension”](#)).

6 Fleet

Fleet (<https://fleet.rancher.io>) is a container management and deployment engine designed to offer users more control on the local cluster and constant monitoring through GitOps. Fleet focuses not only on the ability to scale, but it also gives users a high degree of control and visibility to monitor exactly what is installed on the cluster.

Fleet can manage deployments from Git of raw Kubernetes YAML, Helm charts, Kustomize, or any combination of the three. Regardless of the source, all resources are dynamically turned into Helm charts, and Helm is used as the engine to deploy all resources in the cluster. As a result, users can enjoy a high degree of control, consistency and auditability of their clusters.

For information about how Fleet works, see [this page \(https://ranchermanager.docs.rancher.com/integrations-in-rancher/fleet/architecture\)](https://ranchermanager.docs.rancher.com/integrations-in-rancher/fleet/architecture).

6.1 Installing Fleet with Helm

Fleet comes built-in to Rancher, but it can be also [installed \(https://fleet.rancher.io/installation\)](https://fleet.rancher.io/installation) as a standalone application on any Kubernetes cluster using Helm.

6.2 Using Fleet with Rancher

Rancher uses Fleet to deploy applications across managed clusters. Continuous delivery with Fleet introduces GitOps at scale, designed to manage applications running on large numbers of clusters.

Fleet shines as an integrated part of Rancher. Clusters managed with Rancher automatically get the Fleet agent deployed as part of the installation/import process and the cluster is immediately available to be managed by Fleet.

6.3 Accessing Fleet in the Rancher UI

Fleet comes preinstalled in Rancher and is managed by the **Continuous Delivery** option in the Rancher UI. For additional information on Continuous Delivery and other Fleet troubleshooting tips, refer [here \(https://fleet.rancher.io/troubleshooting\)](https://fleet.rancher.io/troubleshooting).

The screenshot shows a web interface for Continuous Delivery. On the left is a sidebar menu with a hamburger icon at the top. Below it are several items: a home icon, a 'DS1' button, a 'DS0' button, another 'DS1' button, a 'DS2' button, a 'DS4' button, and another 'DS2' button. At the bottom of the sidebar is a blue bar with a sailboat icon and a tooltip that says 'Continuous Delivery'. Below the blue bar is a house icon. The main content area is titled 'Continuous Delivery' and contains a list of items:

- Dashboard
- Git Repos** (1) - This item is highlighted in blue.
- Clusters (8)
- Cluster Groups (1)
- Advanced (dropdown arrow)
- Workspaces (3)
- BundleNamespaceMappings (0)
- Bundles (9)
- Cluster Registration Tokens (0)
- GitRepoRestrictions (0)

To the right of the sidebar, the 'Git Repos' section is visible. It has a 'Pause' button and a 'Force' button. Below these are two columns: 'State' and 'Name'. The 'State' column has a dropdown arrow and a green pill labeled 'Active'. The 'Name' column has a dropdown arrow and the text 'test-repo'.

Continuous Delivery section consists of following items:

6.3.1 Dashboard

An overview page of all GitOps repositories across all workspaces. Only the workspaces with repositories are displayed.

6.3.2 Git repos

A list of GitOps repositories in the selected workspace. Select the active workspace using the drop-down list at the top of the page.

6.3.3 Clusters

A list of managed clusters. By default, all Rancher-managed clusters are added to the `fleet-default` workspace. `fleet-local` workspace includes the local (management) cluster. From here, it is possible to `Pause` or `Force update` the clusters or move the cluster into another workspace. Editing the cluster allows to update labels and annotations used for grouping the clusters.

6.3.4 Cluster groups

This section allows custom grouping of the clusters within the workspace using selectors.

6.3.5 Advanced

The "Advanced" section allows to manage workspaces and other related Fleet resources.

6.4 Example of installing KubeVirt with Rancher and Fleet using Rancher dashboard

1. Create a Git repository containing the `fleet.yaml` file:

```
defaultNamespace: kubevirt
helm:
  chart: "oci://registry.suse.com/edge/kubevirt-chart"
  version: "0.2.4"
  # kubevirt namespace is created by kubevirt as well, we need to take ownership of
  it
  takeOwnership: true
```

2. In the Rancher dashboard, navigate to `# > Continuous Delivery > Git Repos` and click `Add Repository`.

3. The Repository creation wizard guides through creation of the Git repo. Provide **Name**, **Repository URL** (referencing the Git repository created in the previous step) and select the appropriate branch or revision. In the case of a more complex repository, specify **Paths** to use multiple directories in a single repository.

☰
Continuous Delivery

- Dashboard
- Git Repos** (⇒) 1
- Clusters (⇒) 7
- Cluster Groups (⇒) 0
- Advanced >

DS4

DS5

DS4

DS6

DS7

DS8

DS5

Git Repo: Create

Create: Step 1

Define repository details

Name*
kubevirt

Enter a valid HTTPS or SSH URL to

Repository URL
https://github.com/suse-edge/fleet

Git Authentication
None

Helm Authentication
None

TLS Certificate Verification
Require a valid certificate

Resource Handling

Enable Self-Healing

When enabled, Fleet will ensure th


Always Keep Resources

4. Click Next.
5. In the next step, you can define where the workloads will get deployed. Cluster selection offers several basic options: you can select no clusters, all clusters, or directly choose a specific managed cluster or cluster group (if defined). The "Advanced" option allows to directly edit the selectors via YAML.

The screenshot shows the Rancher dashboard interface. On the left is a navigation sidebar with a home icon, a hamburger menu, and several cluster selection buttons labeled DS5, DS4, DS6, DS7, DS8, a cluster icon, and another DS5 button. The main content area is titled 'Continuous Delivery' and contains a menu with 'Dashboard', 'Git Repos' (selected, showing 0), 'Clusters' (6), 'Cluster Groups' (0), and 'Advanced' (>). The right-hand panel displays 'Git Repo: Create' with 'Create: Step 2' (Define target details). Below this is the 'Deploy To' section, where 'No Clusters' is selected, and a list of clusters is visible: ds15, ds4, ds5, ds6, ds7, and ds8.

6. Click Create. The repository gets created. From now on, the workloads are installed and kept in sync on the clusters matching the repository definition.

6.5 Debugging and troubleshooting

The "Advanced" navigation section provides overviews of lower-level Fleet resources. A [bundle \(https://fleet.rancher.io/ref-bundle-stages\)](https://fleet.rancher.io/ref-bundle-stages)  is an internal resource used for the orchestration of resources from Git. When a Git repo is scanned, it produces one or more bundles.

To find bundles relevant to a specific repository, go to the Git repo detail page and click the Bundles tab.

The screenshot displays the Argo CD Continuous Delivery interface. On the left, a navigation sidebar includes a home icon, a hamburger menu, and several cluster selection buttons labeled DS4, DS5, DS4, DS6, DS7, DS8, a Kubernetes icon, and another DS5 button. The main navigation menu is titled 'Continuous Delivery' and lists 'Dashboard', 'Git Repos' (selected, with a count of 1), 'Clusters' (7), 'Cluster Groups' (0), and 'Advanced'. The main content area shows 'Git Repo: kubevirt' in the workspace 'fleet-default'. A 'Bundles' section is visible with a count of 1 and a green progress bar. Below this, there are tabs for 'Bundles' and 'Resources', a 'Download YAML' button, and a table with columns for 'State' and 'Name'. One entry is shown with the state 'Active' and the name 'kubevirt'.

For each cluster, the bundle is applied to a `BundleDeployment` resource that is created. To view `BundleDeployment` details, click the [Graph](#) button in the upper right of the Git repo detail page. A graph of **Repo** > **Bundles** > **BundleDeployments** is loaded. Click the `BundleDeployment` in the graph to see its details and click the [Id](#) to view the `BundleDeployment` YAML.



Continuous Delivery



Dashboard

Git Repos {=} 1

DS4

Clusters {=} 7

DS5

Cluster Groups {=} 0

DS4

Advanced ▾

Workspaces 2

DS6

BundleNamespaceMappings {=} 0

DS7

Bundles {=} 8

DS8

Cluster Registration Tokens {=} 1



GitRepoRestrictions {=} 0

DS5

Git Repo: kubevirt

Workspace: [fleet-default](#) Ag

For additional information on Fleet troubleshooting tips, refer [here \(https://fleet.rancher.io/troubleshooting\)](https://fleet.rancher.io/troubleshooting).

6.6 Fleet examples

The Edge team maintains a [repository \(https://github.com/suse-edge/fleet-examples\)](https://github.com/suse-edge/fleet-examples) with examples of installing Edge projects with Fleet.

The Fleet project includes a [fleet-examples \(https://github.com/rancher/fleet-examples\)](https://github.com/rancher/fleet-examples) repository that covers all use cases for [Git repository structure \(https://fleet.rancher.io/gitrepo-content\)](https://fleet.rancher.io/gitrepo-content).

7 SLE Micro

See [SLE Micro official documentation \(https://documentation.suse.com/sle-micro/5.5/\)](https://documentation.suse.com/sle-micro/5.5/) ↗

SUSE Linux Enterprise Micro is a lightweight and secure operating system for the edge. It merges the enterprise-hardened components of SUSE Linux Enterprise with the features that developers want in a modern, immutable operating system. As a result, you get a reliable infrastructure platform with best-in-class compliance that is also simple to use.

7.1 How does SUSE Edge use SLE Micro?

We use SLE Micro as the base operating system for our platform stack. This provides us with a secure, stable and minimal base for building upon.

SLE Micro is unique in its use of file system (Btrfs) snapshots to allow for easy rollbacks in case something goes wrong with an upgrade. This allows for secure remote upgrades for the entire platform even without physical access in case of issues.

7.2 Best practices

7.2.1 Installation media

SUSE Edge uses the Edge Image Builder ([Chapter 9, Edge Image Builder](#)) to preconfigure the SLE Micro self-install installation image.

7.2.2 Local administration

SLE Micro comes with Cockpit to allow the local management of the host through a Web application.

This service is disabled by default but can be started by enabling the systemd service `cockpit.socket`.

7.3 Known issues

- There is no desktop environment available in SLE Micro at the moment but a containerized solution is in development.

8 Metal³

Metal³ (<https://metal3.io/>) is a CNCF project which provides bare-metal infrastructure management capabilities for Kubernetes.

Metal³ provides Kubernetes-native resources to manage the lifecycle of bare-metal servers which support management via out-of-band protocols such as Redfish (<https://www.dmtf.org/standards/redfish>).

It also has mature support for Cluster API (CAPI) (<https://cluster-api.sigs.k8s.io/>) which enables management of infrastructure resources across multiple infrastructure providers via broadly adopted vendor-neutral APIs.

8.1 How does SUSE Edge use Metal3?

This method is useful for scenarios where the target hardware supports out-of-band management, and a fully automated infrastructure management flow is desired.

This method provides declarative APIs that enable inventory and state management of bare-metal servers, including automated inspection, cleaning and provisioning/deprovisioning.

8.2 Known issues

- The upstream IP Address Management controller (<https://github.com/metal3-io/ip-address-manager>) is currently not supported, because it is not yet compatible with our choice of network configuration tooling.
- Relatedly, the IPAM resources and Metal3DataTemplate networkData fields are not supported.
- Only deployment via redfish-virtualmedia is currently supported.

9 Edge Image Builder

See the [Official Repository \(https://github.com/suse-edge/edge-image-builder\)](https://github.com/suse-edge/edge-image-builder) ↗.

Edge Image Builder (EIB) is a tool that streamlines the generation of Customized, Ready-to-Boot (CRB) disk images for bootstrapping machines. These images enable the end-to-end deployment of the entire SUSE software stack with a single image.

Whilst EIB can create CRB images for all provisioning scenarios, EIB demonstrates a tremendous value in air-gapped deployments with limited or completely isolated networks.

9.1 How does SUSE Edge use Edge Image Builder?

SUSE Edge uses EIB for the simplified and quick configuration of customized SLE Micro images for a variety of scenarios. These scenarios include the bootstrapping of virtual and bare-metal machines with:

- Fully air-gapped deployments of K3s/RKE2 Kubernetes (single & multi-node)
- Fully air-gapped Helm chart and Kubernetes manifest deployments
- Registration to Rancher via Elemental API
- Metal³
- Customized networking (for example, static IP, host name, VLAN's, bonding, etc.)
- Customized operating system configurations (for example, users, groups, passwords, SSH keys, proxies, NTP, custom SSL certificates, etc.)
- Air-gapped installation of host-level and side-loaded RPM packages (including dependency resolution)
- Registration to SUSE Manager for OS management
- Embedded container images
- Kernel command-line arguments
- Systemd units to be enabled/disabled at boot time
- Custom scripts and files for any manual tasks

9.2 Getting started

Comprehensive documentation for the usage and testing of Edge Image Builder can be found [here \(https://github.com/suse-edge/edge-image-builder/tree/release-1.0/docs\)](https://github.com/suse-edge/edge-image-builder/tree/release-1.0/docs).

Additionally, here is a quick start guide (*Chapter 3, Standalone clusters with Edge Image Builder*) for Edge Image Builder covering a basic deployment scenario.

9.3 Known issues

- EIB air-gaps Helm charts through templating the Helm charts and parsing all the images within the template. If a Helm chart does not keep all of its images within the template and instead side-loads the images, EIB will not be able to air-gap those images automatically. The solution to this is to manually add any undetected images to the `embeddedArtifactRegistry` section of the definition file.

10 Edge Networking

This section describes the approach to network configuration in the SUSE Edge solution. We will show how to configure NetworkManager on SLE Micro in a declarative manner, and explain how the related tools are integrated.

10.1 Overview of NetworkManager

NetworkManager is a tool that manages the primary network connection and other connection interfaces.

NetworkManager stores network configurations as connection files that contain the desired state. These connections are stored as files in the `/etc/NetworkManager/system-connections/` directory.

Details about NetworkManager can be found in the [upstream SLE Micro documentation \(https://documentation.suse.com/sle-micro/5.5/html/SLE-Micro-all/cha-nm-configuration.html\)](https://documentation.suse.com/sle-micro/5.5/html/SLE-Micro-all/cha-nm-configuration.html).

10.2 Overview of nmstate

nmstate is a widely adopted library (with an accompanying CLI tool) which offers a declarative API for network configurations via a predefined schema.

Details about nmstate can be found in the [upstream documentation \(https://nmstate.io/\)](https://nmstate.io/).

10.3 Enter: NetworkManager Configurator (nmc)

The network customization options available in SUSE Edge are achieved via a CLI tool called NetworkManager Configurator or *nmc* for short. It is leveraging the functionality provided by the nmstate library and, as such, it is fully capable of configuring static IP addresses, DNS servers, VLANs, bonding, bridges, etc. This tool allows us to generate network configurations from predefined desired states and to apply those across many different nodes in an automated fashion.

Details about the NetworkManager Configurator (nmc) can be found in the [upstream repository \(https://github.com/suse-edge/nm-configurator\)](https://github.com/suse-edge/nm-configurator).

10.4 How does SUSE Edge use NetworkManager Configurator?

SUSE Edge utilizes *nmc* for the network customizations in the various different provisioning models:


- Custom network configurations in the Directed Network Provisioning scenarios (*Chapter 1, BMC automated deployments with Metal³*)
- Declarative static configurations in the Image Based Provisioning scenarios (*Chapter 3, Standalone clusters with Edge Image Builder*)

10.5 Configuring with Edge Image Builder

Edge Image Builder (EIB) is a tool which enables configuring multiple hosts with a single OS image. In this section we'll show how you can use a declarative approach to describe the desired network states, how those are converted to the respective NetworkManager connections, and are then applied during the provisioning process.

10.5.1 Prerequisites

If you're following this guide, it's assumed that you've got the following already available:

- An x86_64 physical host (or virtual machine) running SLES 15 SP5 or openSUSE Leap 15.5
- An available container runtime (e.g. Podman)
- A copy of the SLE Micro 5.5 RAW image found [here \(https://www.suse.com/download/sle-micro/\)](https://www.suse.com/download/sle-micro/) 

10.5.2 Getting the Edge Image Builder container image

The EIB container image is publicly available and can be downloaded from the SUSE Edge registry by running:

```
podman pull registry.suse.com/edge/edge-image-builder:1.0.2
```

10.5.3 Creating the image configuration directory

Let's start with creating the configuration directory:

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

We will now ensure that the downloaded base image copy is moved over to the configuration directory:

```
mv /path/to/downloads/SLE-Micro.x86_64-5.5.0-Default-GM.raw $CONFIG_DIR/base-images/
```



Note

EIB is never going to modify the base image input.

The configuration directory at this point should look like the following:

```
├─ base-images/
  └─ SLE-Micro.x86_64-5.5.0-Default-GM.raw
```

10.5.4 Creating the image definition file

The definition file describes the majority of configurable options that the Edge Image Builder supports.

Let's start with a very basic definition file for our OS image:

```
cat << EOF > $CONFIG_DIR/definition.yaml
apiVersion: 1.0
image:
  arch: x86_64
  imageType: raw
  baseImage: SLE-Micro.x86_64-5.5.0-Default-GM.raw
  outputImageName: modified-image.raw
operatingSystem:
  users:
    - username: root
      encryptedPassword: $6$jHugJNNd3HElGsUZ
$eodjVe4te5ps44SVcWshdfWizrP.xAyd71CVEXazBJ/.v799/WRCBXxfYmunlB02yp1hm/zb4r8EmnrrNCF.P/
EOF
```

The `image` section is required, and it specifies the input image, its architecture and type, as well as what the output image will be called. The `operatingSystem` section is optional, and contains configuration to enable login on the provisioned systems with the `root/eib` username/password.



Note

Feel free to use your own encrypted password by running `openssl passwd -6 <password>`.

The configuration directory at this point should look like the following:

```
├─ definition.yaml
├─ base-images/
│  └─ SLE-Micro.x86_64-5.5.0-Default-GM.raw
```

10.5.5 Defining the network configurations

The desired network configurations are not part of the image definition file that we just created. We'll now populate those under the special `network/` directory. Let's create it:

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

As previously mentioned, the NetworkManager Configurator (*nmc*) tool expects an input in the form of predefined schema. You can find how to set up a wide variety of different networking options in the [upstream NMState examples documentation \(https://nmstate.io/examples.html\)](https://nmstate.io/examples.html).

This guide will explain how to configure the networking on three different nodes:

- A node which uses two Ethernet interfaces
- A node which uses network bonding
- A node which uses a network bridge



Warning

Using completely different network setups is not recommended in production builds, especially if configuring Kubernetes clusters. Networking configurations should generally be homogeneous amongst nodes or at least amongst roles within a given cluster. This guide is including various different options only to serve as an example reference.



Note

The following assumes a default `libvirt` network with an IP address range `192.168.122.1/24`. Adjust accordingly if this differs in your environment.

Let's create the desired states for the first node which we will call `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
```

In this example we define a desired state of two Ethernet interfaces (eth0 and eth3), their requested IP addresses, routing, and DNS resolution.



Warning

You must ensure that the MAC addresses of all Ethernet interfaces are listed. Those are used during the provisioning process as the identifiers of the nodes and serve to determine which configurations should be applied. This is how we are able to configure multiple nodes using a single ISO or RAW image.

Next up is the second node which we will call node2.suse.com and which will use network bonding:

```
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

In this example we define a desired state of two Ethernet interfaces (eth0 and eth1) which are not enabling IP addressing, as well as a bond with a round-robin policy and its respective address which is going to be used to forward the network traffic.

Lastly, we'll create the third and final desired state file which will be utilizing a network bridge and which we'll call `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

```

The configuration directory at this point should look like the following:

```

├─ definition.yaml
├─ network/
│  └─ node1.suse.com.yaml
│  └─ node2.suse.com.yaml
│  └─ node3.suse.com.yaml

```



```
└─ base-images/
   └─ SLE-Micro.x86_64-5.5.0-Default-GM.raw
```



Note

The names of the files under the `network/` directory are intentional. They correspond to the hostnames which will be set during the provisioning process.

10.5.6 Building the OS image

Now that all the necessary configurations are in place, we can build the image by simply running:

```
podman run --rm -it -v $CONFIG_DIR:/eib registry.suse.com/edge/edge-image-builder:1.0.2
build --definition-file definition.yaml
```

The output should be similar to the following:

```
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!
```

The snippet above tells us that the `Network` component has successfully been configured, and we can proceed with provisioning our edge nodes.



Note

A log file (`network-config.log`) and the respective NetworkManager connection files can be inspected in the resulting `_build` directory under a timestamped directory for the image run.

10.5.7 Provisioning the edge nodes

Let's copy the resulting RAW image:

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

You will notice that we copied the built image four times but only specified the network configurations for three nodes. This is because we also want to showcase what will happen if we provision a node which does not match any of the desired configurations.



Note

This guide will use virtualization for the node provisioning examples. Ensure the necessary extensions are enabled in the BIOS (see [here \(https://documentation.suse.com/sles/15-SP5/html/SLES-all/cha-virt-support.html#sec-kvm-requires-hardware\)](https://documentation.suse.com/sles/15-SP5/html/SLES-all/cha-virt-support.html#sec-kvm-requires-hardware) for details).

We will be using `virt-install` to create virtual machines using the copied raw disks. Each virtual machine will be using 10 GB of RAM and 6 vCPUs.

10.5.7.1 Provisioning the first node

Let's create the virtual machine:

```
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
```



Note

It is important that we create the network interfaces with the same MAC addresses as the ones in the desired state we described above.

Once the operation is complete, we will see something similar to the following:

```
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 Enterprise Micro 5.5 (x86_64) - Kernel 5.14.21-150500.55.19-
default (ttyS0).

SSH host key: SHA256:YN/R5Tw43reG+Qs0w480LxCnhkc/1uqMdwLI6KUBY70 (RSA)
SSH host key: SHA256:/96yGrPGKlhn04f1rb9cXv/2WJt4TtrIN5yEcN66r3s (DSA)
SSH host key: SHA256:Dy/YjBQ7LwjZGaaVcMhTWZNS0stxXBsPsvgJTJq5t00 (ECDSA)
SSH host key: SHA256:TNGqY1LRddpxD/jn/8dkT/9YmVl9hiwulqmayP+w0WQ (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:
```

We're now able to log in with the `root:eib` credentials pair. We're also able to SSH into the host if we prefer that over the `virsh console` we're presented with here.

Once logged in, let's confirm that all the settings are in place.

Verify that the hostname is properly set:

```
node1:~ # hostnamectl
Static hostname: node1.suse.com
...
```

Verify that the routing is properly configured:

```
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
```

Verify that Internet connection is available:

```
nodel:~ # 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
```

Verify that exactly two Ethernet interfaces are configured and only one of those is active:

```
nodel:~ # 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

nodel:~ # 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
```

You'll notice that the second interface is eth1 instead of the predefined eth3 in our desired networking state. This is the case because the NetworkManager Configurator (*nmc*) is able to detect that the OS has given a different name for the NIC with MAC address 34:8a:b1:4b:16:e2 and it adjusts its settings accordingly.

Verify this has indeed happened by inspecting the Combustion phase of the provisioning:

```
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
```

We will now provision the rest of the nodes, but we will only show the differences in the final configuration. Feel free to apply any or all of the above checks for all nodes you are about to provision.

10.5.7.2 Provisioning the second node

Let's create the virtual machine:

```
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
```

Once the virtual machine is up and running, we can confirm that this node is using bonded interfaces:

```
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

```

Confirm that the routing is using the bond:

```

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

```

Ensure that the static connection files are properly utilized:

```

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

```

10.5.7.3 Provisioning the third node

Let's create the virtual machine:

```

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

```

Once the virtual machine is up and running, we can confirm that this node is using a network bridge:

```

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

```

Confirm that the routing is using the bridge:

```

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

```

Ensure that the static connection files are properly utilized:

```

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

```

10.5.7.4 Provisioning the fourth node

Lastly, we will provision a node which will not match any of the predefined configurations by a MAC address. In these cases, we will default to DHCP to configure the network interfaces.

Let's create the virtual machine:

```

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

```

Once the virtual machine is up and running, we can confirm that this node is using a random IP address for its network interface:

```

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

```

Verify that nmc failed to apply static configurations for this node:

```

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

```

Verify that the Ethernet interface was configured via 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

```

10.5.8 Unified node configurations

There are occasions where relying on known MAC addresses is not an option. In these cases we can opt for the so-called *unified configuration* which allows us to specify settings in an `_all.yaml` file which will then be applied across all provisioned nodes.

We will build and provision an edge node using different configuration structure. Follow all steps starting from [Section 10.5.3, "Creating the image configuration directory"](#) up until [Section 10.5.5, "Defining the network configurations"](#).

In this example we define a desired state of two Ethernet interfaces (eth0 and eth1) - one using DHCP, and one assigned a static IP address.

```

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
```

Let's build the image:

```
podman run --rm -it -v $CONFIG_DIR:/eib registry.suse.com/edge/edge-image-builder:1.0.2
build --definition-file definition.yaml
```

Once the image is successfully built, let's create a virtual machine using it:

```
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
```

The provisioning process might take a few minutes. Once it's finished, log in to the system with the provided credentials.

Verify that the routing is properly configured:

```
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
```

Verify that Internet connection is available:

```
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
```

Verify that the Ethernet interfaces are configured and active:

```
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
```

10.5.9 Custom network configurations

We have already covered the default network configuration for Edge Image Builder which relies on the NetworkManager Configurator. However, there is also the option to modify it via a custom script. Whilst this option is very flexible and is also not MAC address dependant, its limitation stems from the fact that using it is much less convenient when bootstrapping multiple nodes with a single image.



Note

It is recommended to use the default network configuration via files describing the desired network states under the `/network` directory. Only opt for custom scripting when that behaviour is not applicable to your use case.

We will build and provision an edge node using different configuration structure. Follow all steps starting from [Section 10.5.3, “Creating the image configuration directory”](#) up until [Section 10.5.5, “Defining the network configurations”](#).

In this example, we will create a custom script which applies static configuration for the `eth0` interface on all provisioned nodes, as well as removing and disabling the automatically created wired connections by NetworkManager. This is beneficial in situations where you want to make sure that every node in your cluster has an identical networking configuration, and as such you do not need to be concerned with the MAC address of each node prior to image creation.

Let’s start by storing the connection file in the `/custom/files` directory:

```
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

```

Now that the static configuration is created, we will also create our custom network script:

```

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

```



Note

The `nmc` binary will still be included by default, so it can also be used in the `configure-network.sh` script if necessary.



Warning

The custom script must always be provided under `/network/configure-network.sh` in the configuration directory. If present, all other files will be ignored. It is NOT possible to configure a network by working with both static configurations in YAML format and a custom script simultaneously.

The configuration directory at this point should look like the following:

```
├─ definition.yaml
├─ custom/
│   └─ files/
│       └─ eth0.nmconnection
├─ network/
│   └─ configure-network.sh
└─ base-images/
    └─ SLE-Micro.x86_64-5.5.0-Default-GM.raw
```

Let's build the image:

```
podman run --rm -it -v $CONFIG_DIR:/eib registry.suse.com/edge/edge-image-builder:1.0.2
build --definition-file definition.yaml
```

Once the image is successfully built, let's create a virtual machine using it:

```
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
```

The provisioning process might take a few minutes. Once it's finished, log in to the system with the provided credentials.

Verify that the routing is properly configured:

```
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
```

Verify that Internet connection is available:

```
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
```

Verify that an Ethernet interface is statically configured using our connection file and is active:

```
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
```

11 Elemental

Elemental is a software stack enabling centralized and full cloud-native OS management with Kubernetes. The Elemental stack consists of a number of components that either reside on Rancher itself, or on the edge nodes. The core components are:

- **elemental-operator** - The core operator that resides on Rancher and handles registration requests from clients.
- **elemental-register** - The client that runs on the edge nodes allowing registration via the `elemental-operator`.
- **elemental-system-agent** - An agent that resides on the edge nodes; its configuration is fed from `elemental-register` and it receives a `plan` for configuring the `rancher-system-agent`
- **rancher-system-agent** - Once the edge node has fully registered, this takes over from `elemental-system-agent` and waits for further `plans` from Rancher Manager (e.g. for Kubernetes installation).

See [Elemental upstream documentation \(https://elemental.docs.rancher.com/\)](https://elemental.docs.rancher.com/) for full information about Elemental and its relationship to Rancher.

11.1 How does SUSE Edge use Elemental?

We use portions of Elemental for managing remote devices where Metal³ is not an option (for example, there is no BMC, or the device is behind a NAT gateway). This tooling allows for an operator to bootstrap their devices in a lab before knowing when or where they will be shipped to. Namely, we leverage the `elemental-register` and `elemental-system-agent` components to enable the onboarding of SLE Micro hosts to Rancher for "phone home" network provisioning use-cases. When using Edge Image Builder (EIB) to create deployment images, the automatic registration through Rancher via Elemental can be achieved by specifying the registration configuration in the configuration directory for EIB.



Note

In SUSE Edge 3.0 we do **not** leverage the operating system management aspects of Elemental, and therefore it's not possible to manage your operating system patching via Rancher. Instead of using the Elemental tools to build deployment images, SUSE Edge uses the Edge Image Builder tooling, which consumes the registration configuration.

11.2 Best practices

11.2.1 Installation media

The SUSE Edge recommended way of building deployments image that can leverage Elemental for registration to Rancher in the "phone home network provisioning" deployment footprint is to follow the instructions detailed in the remote host onboarding with Elemental ([Chapter 2, Remote host onboarding with Elemental](#)) quickstart.

11.2.2 Labels


Elemental tracks its inventory with the `MachineInventory` CRD and provides a way to select inventory, e.g. for selecting machines to deploy Kubernetes clusters to, based on labels. This provides a way for users to predefine most (if not all) of their infrastructure needs prior to hardware even being purchased. Also, since nodes can add/remove labels on their respective inventory object (by re-running `elemental-register` with the additional flag `--label "FOO=BAR"`), we can write scripts that will discover and let Rancher know where a node is booted.

11.3 Known issues

- The Elemental UI does not currently know how to build installation media or update non-"Elemental Teal" operating systems. This should be addressed in future releases.

12 Akri

Akri is a CNCF-Sandbox project that aims to discover leaf devices to present those as Kubernetes native resource. It also allows scheduling a pod or a job for each discovered device. Devices can be node-local or networked, and can use a wide variety of protocols.

Akri's upstream documentation is available at: <https://docs.akri.sh> 

12.1 How does SUSE Edge use Akri?



Warning

Akri is currently tech-preview in the SUSE Edge stack.

Akri is available as part of the Edge Stack whenever there is a need to discover and schedule workload against leaf devices.

12.1.1 Installing Akri

Akri is available as a Helm chart within the Edge Helm repository. The recommended way of configuring Akri is by using the given Helm chart to deploy the different components (agent, controller, discovery-handlers), and then use your preferred deployment mechanism to deploy Akri's Configuration CRDs.

12.1.2 Configuring Akri

Akri is configured using a `akri.sh/Configuration` object, this object takes in all information about how to discover the devices, as well as what to do when a matching one is discovered.

Here is an example configuration breakdown with all fields explained:

```
apiVersion: akri.sh/v0
kind: Configuration
metadata:
  name: sample-configuration
spec:
```

This part describes the configuration of the discovery handler, you have to specify its name (the handlers available as part of Akri's chart are [udev](#), [opcu](#), [onvif](#)). The [discoveryDetails](#) is handler specific, refer to the handler's documentation on how to configure it.

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

This section defines the workload to be deployed for every discovered device. The example shows a minimal version of a [Pod](#) configuration in [brokerPodSpec](#), all usual fields of a Pod's spec can be used here. It also shows the Akri specific syntax to request the device in the [resources](#) section.

You can alternatively use a Job instead of a Pod, using the [brokerJobSpec](#) key instead, and providing the spec part of a Job to it.

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

These two sections show how to configure Akri to deploy a service per broker ([instanceService](#)), or pointing to all brokers ([configurationService](#)). These are containing all elements pertaining to a usual Service.

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

```
port: 443
protocol: tcp
targetPort: 443
```

The `brokerProperties` field is a key/value store that will be exposed as additional environment variables to any pod requesting a discovered device.

The `capacity` is the allowed number of concurrent users of a discovered device.

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

12.1.3 Writing and deploying additional Discovery Handlers

In case the protocol used by your device isn't covered by an existing discovery handler, you can write your own using [this guide \(https://docs.akri.sh/development/handler-development\)](https://docs.akri.sh/development/handler-development) ↗


12.1.4 Akri Rancher Dashboard Extension


Akri Dashboard Extension allows you to use Rancher Dashboard user interface to manage and monitor leaf devices and run workloads once these devices are discovered.


Once the extension is installed you can navigate to any Akri-enabled managed cluster using cluster explorer. Under **Akri** navigation group you can see Configurations and Instances sections.


The screenshot displays the Akri Rancher Dashboard Extension interface. On the left, a navigation sidebar is visible with a menu icon at the top. The sidebar items include Cluster, Workloads, Apps, Service Discovery, Storage, Policy, Akri, Configurations (highlighted in blue), Instances, and More Resources. The main content area is titled 'Configurations' and features a 'Download YAML' button. Below this, there is a table with columns for 'State' and 'Name'. A single configuration is listed with the state 'Active' and the name 'akri-debu'.

The configurations list provides information about Configuration Discovery Handler and number of instances. Clicking the name opens a configuration detail page.




local





- Cluster >
- Workloads >
- Apps >
- Service Discovery >
- Storage >
- Policy >
- Akri >
- Configurations** {=} 1
- Instances {=} 2
- More Resources >

Configuration: akri

Namespace: akri Age: 1.4 hours

Labels: app.kubernetes.io/managed-by=akri

Annotations: [Show 2 annotations](#)

[Instances](#)

[Recent Events](#)

↓ Download YAML

<input type="checkbox"/>	State ⌵	Name
<input type="checkbox"/>	Active	akri-d echo- 57dde
<input type="checkbox"/>	Active	akri-d echo- a24f2

You can also edit or create a new Configuration. Extension allows you to select discovery handler, set up Broker Pod or Job, configure Configuration and Instance services and set the Configuration capacity.

☰ local

- Cluster >
- Workloads >
- Apps >
- Service Discovery >
- Storage >
- Policy >
- Akri ▾
- Configurations** {=} 1
- Instances {=} 2
- More Resources >

Configuration: akri

Namespace: akri Age: 1.4 hours

Namespace*
akri

- Discovery handler
- Broker pod
- Broker job
- Instance service
- Configuration service
- Capacity

Discovered devices are listed in the **Instances** list.

The screenshot displays the Akri Rancher Dashboard Extension interface. On the left, a navigation sidebar is visible with a hamburger menu icon at the top. Below it, there are three icons: a home icon, a bull icon, and a bull icon inside a square. The main content area is titled 'local' with a bull icon. A list of navigation items is shown: Cluster, Workloads, Apps, Service Discovery, Storage, Policy, Akri (expanded), Configurations (1), **Instances** (2), and More Resources. The 'Instances' item is highlighted in blue. On the right, the 'Instances' section is titled 'Instances' with a star icon. Below the title is a 'Download YAML' button. A table lists instances with columns for 'State' and 'Name'. The table shows two instances, both with a state of 'Active' and names starting with 'akri-debu'.

State	Name
Active	akri-debu
Active	akri-debu

Clicking the Instance name opens a detail page allowing to view the workloads and instance service.



 local



- Cluster >
- Workloads >
- Apps >
- Service Discovery >
- Storage >
- Policy >
- Akri >
 - Instances {=} 2
 - Configurations {=} 1
- More Resources >

Akri instance: akri-

Namespace: [akri](#) Age: 48 min

[Details](#)

[Broker jobs](#)

Referred To By

State	Type
Active	Configurat

Refers To

State	Type
Running	Pod
Active	Service

13 K3s

K3s (<https://k3s.io/>) is a highly available, certified Kubernetes distribution designed for production workloads in unattended, resource-constrained, remote locations or inside IoT appliances. It is packaged as a single and small binary, so installations and updates are fast and easy.

13.1 How does SUSE Edge use K3s

K3s can be used as the Kubernetes distribution backing the SUSE Edge stack. It is meant to be installed on a SLE Micro operating system.

Using K3s as the SUSE Edge stack Kubernetes distribution is only recommended when etcd as a backend does not fit your constraints. If etcd as a backend is possible, it is better to use RKE2 (*Chapter 14, RKE2*).

13.2 Best practices

13.2.1 Installation

The recommended way of installing K3s as part of the SUSE Edge stack is by using Edge Image Builder (EIB). See its documentation (*Chapter 9, Edge Image Builder*) for more details on how to configure it to deploy K3s.

It automatically supports HA setup, as well as Elemental setup.

13.2.2 Fleet for GitOps workflow

The SUSE Edge stack uses Fleet as its preferred GitOps tool. For more information around its installation and use, refer to the Fleet section (*Chapter 6, Fleet*) in this documentation.

13.2.3 Storage management

K3s comes with local-path storage preconfigured, which is suitable for single-node clusters. For clusters spanning over multiple nodes, we recommend using Longhorn (*Chapter 15, Longhorn*).

13.2.4 Load balancing and HA

If you installed K3s using EIB, this part is already covered by the EIB documentation in the HA section.

Otherwise, you need to install and configure MetalLB as per our MetalLB documentation ([Chapter 19, MetalLB on K3s \(using L2\)](#)).

14 RKE2

See [RKE2 official documentation \(https://docs.rke2.io/\)](https://docs.rke2.io/).

RKE2 is a fully conformant Kubernetes distribution that focuses on security and compliance by:

- Providing defaults and configuration options that allow clusters to pass the CIS Kubernetes Benchmark v1.6 or v1.23 with minimal operator intervention
- Enabling FIPS 140-2 compliance
- Regularly scanning components for CVEs using [trivy \(https://trivy.dev\)](https://trivy.dev) in the RKE2 build pipeline

RKE2 launches control plane components as static pods, managed by kubelet. The embedded container runtime is containerd.

Note: RKE2 is also known as RKE Government in order to convey another use case and sector it currently targets.

14.1 RKE2 vs K3s

K3s is a fully compliant and lightweight Kubernetes distribution focused on Edge, IoT, ARM - optimized for ease of use and resource constrained environments.

RKE2 combines the best of both worlds from the 1.x version of RKE (hereafter referred to as RKE1) and K3s.

From K3s, it inherits the usability, ease of operation and deployment model.

From RKE1, it inherits close alignment with upstream Kubernetes. In places, K3s has diverged from upstream Kubernetes in order to optimize for edge deployments, but RKE1 and RKE2 can stay closely aligned with upstream.

14.2 How does SUSE Edge use RKE2?

RKE2 is a fundamental piece of the SUSE Edge stack. It sits on top of SUSE Linux Micro ([Chapter 7, SLE Micro](#)), providing a standard Kubernetes interface required to deploy Edge workloads.

14.3 Best practices

14.3.1 Installation

The recommended way of installing RKE2 as part of the SUSE Edge stack is by using Edge Image Builder (EIB). See the EIB documentation ([Chapter 9, Edge Image Builder](#)) for more details on how to configure it to deploy RKE2.

EIB is flexible enough to support any parameter required by RKE2, such as specifying the RKE2 version, the [servers](https://docs.rke2.io/reference/server_config) or the [agents](https://docs.rke2.io/reference/linux_agent_config) configuration, covering all the Edge use cases.

For other use cases involving Metal³, RKE2 is also being used and installed. In those particular cases, the [Cluster API Provider RKE2](https://github.com/rancher-sandbox/cluster-api-provider-rke2) automatically deploys RKE2 on clusters being provisioned with Metal³ using the Edge Stack.

In those cases, the RKE2 configuration must be applied on the different CRDs involved. An example of how to provide a different CNI using the `RKE2ControlPlane` CRD looks like:

```
apiVersion: controlplane.cluster.x-k8s.io/v1alpha1
kind: RKE2ControlPlane
metadata:
  name: single-node-cluster
  namespace: default
spec:
  serverConfig:
    cni: calico
    cniMultusEnable: true
  ...
```

For more information about the Metal³ use cases, see [Chapter 8, Metal³](#).

14.3.2 High availability

For HA deployments, EIB automatically deploys and configures MetalLB ([Chapter 17, MetalLB](#)) and the [Endpoint Copier Operator](https://github.com/suse-edge/endpoint-copier-operator) to expose the RKE2 API endpoint externally.

14.3.3 Networking

The supported CNI for the Edge Stack is [Cilium](https://docs.cilium.io/en/stable/) with optionally adding the meta-plugin [Multus](https://github.com/k8snetworkplumbingwg/multus-cni), but RKE2 supports [a few others](https://docs.rke2.io/install/network_options) as well.

14.3.4 Storage

RKE2 does not provide any kind of persistent storage class or operators. For clusters spanning over multiple nodes, it is recommended to use Longhorn ([Chapter 15, Longhorn](#)).

15 Longhorn

Longhorn is a lightweight, reliable and user-friendly distributed block storage system designed for Kubernetes. As an open source project, Longhorn was initially developed by Rancher Labs and is currently incubated under the CNCF.

15.1 Prerequisites

If you are following this guide, it assumes that you have the following already available:

- At least one host with SLE Micro 5.5 installed; this can be physical or virtual
- A Kubernetes cluster installed; either K3s or RKE2
- Helm

15.2 Manual installation of Longhorn

15.2.1 Installing Open-iSCSI

A core requirement of deploying and using Longhorn is the installation of the `open-iscsi` package and the `iscsid` daemon running on all Kubernetes nodes. This is necessary, since Longhorn relies on `iscsiadm` on the host to provide persistent volumes to Kubernetes.

Let's install it:

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

It is important to note that once the operation is completed, the package is only installed into a new snapshot as SLE Micro is an immutable operating system. In order to load it and for the `iscsid` daemon to start running, we must reboot into that new snapshot that we just created. Issue the `reboot` command when you are ready:

```
reboot
```




Tip

For additional help installing open-iscsi, refer to the [official Longhorn documentation \(https://longhorn.io/docs/1.6.1/deploy/install/#installing-open-iscsi\)](https://longhorn.io/docs/1.6.1/deploy/install/#installing-open-iscsi).

15.2.2 Installing Longhorn

There are several ways to install Longhorn on your Kubernetes clusters. This guide will follow through the Helm installation, however feel free to follow the [official documentation \(https://longhorn.io/docs/1.6.1/deploy/install/\)](https://longhorn.io/docs/1.6.1/deploy/install/) if another approach is desired.

1. Add the Longhorn Helm repository:

```
helm repo add longhorn https://charts.longhorn.io
```

2. Fetch the latest charts from the repository:

```
helm repo update
```

3. Install Longhorn in the longhorn-system namespace:

```
helm install longhorn longhorn/longhorn --namespace longhorn-system --create-namespace --version 1.6.1
```

4. Confirm that the deployment succeeded:

```
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	

15.3 Creating Longhorn volumes

Longhorn utilizes Kubernetes resources called StorageClass in order to automatically provision PersistentVolume objects for pods. Think of StorageClass as a way for administrators to describe the *classes* or *profiles* of storage they offer.

Let's create a StorageClass with some default options:

```
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
```

Now that we have our StorageClass in place, we need a PersistentVolumeClaim referencing it. A PersistentVolumeClaim (PVC) is a request for storage by a user. PVCs consume PersistentVolume resources. Claims can request specific sizes and access modes (e.g., they can be mounted once read/write or many times read-only).

Let's create a 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
```

That's it! Once we have the PersistentVolumeClaim created, we can proceed with attaching it to a Pod. When the Pod is deployed, Kubernetes creates the Longhorn volume and binds it to the Pod if storage is available.

```
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

```



Tip

The concept of storage in Kubernetes is a complex, but important topic. We briefly mentioned some of the most common Kubernetes resources, however, we suggest to familiarize yourself with the [terminology documentation \(https://longhorn.io/docs/1.6.1/terminology/\)](https://longhorn.io/docs/1.6.1/terminology/) that Longhorn offers.

In this example, the result should look something like this:

```

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        RW0
                                           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

15.4 Accessing the UI

If you installed Longhorn with `kubectl` or Helm, you need to set up an Ingress controller to allow external traffic into the cluster. Authentication is not enabled by default. If the Rancher catalog app was used, Rancher automatically created an Ingress controller with access control (the `rancher-proxy`).

1. Get the Longhorn's external service IP address:

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

2. Once you have retrieved the `longhorn-frontend` IP address, you can start using the UI by navigating to it in your browser.

15.5 Installing with Edge Image Builder

SUSE Edge is using [Chapter 9, Edge Image Builder](#) in order to customize base SLE Micro OS images. We are going to demonstrate how to do so for provisioning an RKE2 cluster with Longhorn on top of it.

Let's create the definition file:

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

cat << EOF > $CONFIG_DIR/iso-definition.yaml
apiVersion: 1.0
image:
  imageType: iso
  baseImage: SLE-Micro.x86_64-5.5.0-Default-SelfInstall-GM2.install.iso
  arch: x86_64
  outputImageName: eib-image.iso
kubernetes:
  version: v1.28.13+rke2r1
helm:
```

```

charts:
  - name: longhorn
    version: 1.6.1
    repositoryName: longhorn
    targetNamespace: longhorn-system
    createNamespace: true
    installationNamespace: kube-system
repositories:
  - name: longhorn
    url: https://charts.longhorn.io
operatingSystem:
  packages:
    sccRegistrationCode: <reg-code>
    packageList:
      - open-iscsi
  users:
    - username: root
      encryptedPassword: \$6\$jHugJNNd3HELGsUZ\
\$eodjVe4te5ps44SVcWshdfWizrP.xAyd71CVEXazBJ/.v799/WRCBXxfYmunlB02yp1hm/zb4r8EmnrrNCF.P/
EOF

```



Note

Customizing any of the Helm chart values is possible via a separate file provided under `helm.charts[].valuesFile`. Refer to the [upstream documentation](https://github.com/suse-edge/edge-image-builder/blob/release-1.0/docs/building-images.md#kubernetes) (<https://github.com/suse-edge/edge-image-builder/blob/release-1.0/docs/building-images.md#kubernetes>) for details.

Let's build the image:

```
podman run --rm --privileged -it -v $CONFIG_DIR:/eib registry.suse.com/edge/edge-image-builder:1.0.2 build --definition-file $CONFIG_DIR/iso-definition.yaml
```

After the image is built, you can use it to install your OS on a physical or virtual host. Once the provisioning is complete, you are able to log in to the system using the `root:eib` credentials pair.

Ensure that Longhorn has been successfully deployed:

```
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 103s	1/1	Running	0
csi-provisioner-667796df57-fmz2d 103s	1/1	Running	0
csi-provisioner-667796df57-p7rjr 103s	1/1	Running	0
csi-provisioner-667796df57-w9fdq 103s	1/1	Running	0
csi-resizer-694f8f5f64-2rb8v 103s	1/1	Running	0
csi-resizer-694f8f5f64-z9v9x 103s	1/1	Running	0
csi-resizer-694f8f5f64-zlncz 103s	1/1	Running	0
csi-snapshotter-959b69d4b-5dpvj 103s	1/1	Running	0
csi-snapshotter-959b69d4b-lwwkv 103s	1/1	Running	0
csi-snapshotter-959b69d4b-tzhwc 103s	1/1	Running	0
engine-image-ei-5cefaf2b-hvdv5 109s	1/1	Running	0
instance-manager-0ee452a2e9583753e35ad00602250c5b 109s	1/1	Running	0
longhorn-csi-plugin-gd2jx 103s	3/3	Running	0
longhorn-driver-deployer-9f4fc86-j6h2b 2m28s	1/1	Running	0
longhorn-manager-z4lnl 2m28s	1/1	Running	0
longhorn-ui-5f4b7bbf69-bl7h 2m28s	1/1	Running	3 (2m7s ago)
longhorn-ui-5f4b7bbf69-lh97n 2m28s	1/1	Running	3 (2m10s ago)



Note


This installation will not work for completely air-gapped environments. In those cases, please refer to [Section 21.8, “Longhorn Installation”](#).

16 NeuVector

NeuVector is a security solution for Kubernetes that provides L7 network security, runtime security, supply chain security, and compliance checks in a cohesive package.

NeuVector is deployed as a platform of several containers that communicate with each other on various ports and interfaces. The following are the different containers deployed:

- **Manager.** A stateless container which presents the Web-based console. Typically, only one is needed and this can run anywhere. Failure of the Manager does not affect any of the operations of the controller or enforcer. However, certain notifications (events) and recent connection data are cached in memory by the Manager so viewing of these would be affected.
- **Controller.** The ‘control plane’ for NeuVector must be deployed in an HA configuration, so configuration is not lost in a node failure. These can run anywhere, although customers often choose to place these on ‘management’, master or infra nodes because of their criticality.
- **Enforcer.** This container is deployed as a DaemonSet so one Enforcer is on every node to be protected. Typically deploys to every worker node but scheduling can be enabled for master and infra nodes to deploy there as well. Note: If the Enforcer is not on a cluster node and connections come from a pod on that node, NeuVector labels them as ‘unmanaged’ workloads.
- **Scanner.** Performs the vulnerability scanning using the built-in CVE database, as directed by the Controller. Multiple scanners can be deployed to increase scanning capacity. Scanners can run anywhere but are often run on the nodes where the controllers run. See below for sizing considerations of scanner nodes. A scanner can also be invoked independently when used for build-phase scanning, for example, within a pipeline that triggers a scan, retrieves the results, and stops the scanner. The scanner contains the latest CVE database so should be updated daily.
- **Updater.** The updater triggers an update of the scanner through a Kubernetes cron job when an update of the CVE database is desired. Please be sure to configure this for your environment.

A more in-depth NeuVector onboarding and best practices documentation can be found [here](https://open-docs.neuvector.com/deploying/production/NV_Onboarding_5.0.pdf) (https://open-docs.neuvector.com/deploying/production/NV_Onboarding_5.0.pdf) .

16.1 How does SUSE Edge use NeuVector?

SUSE Edge provides a leaner configuration of NeuVector as a starting point for edge deployments.

Find the NeuVector configuration changes [here](https://github.com/suse-edge/charts/blob/main/packages/neuvector-core/generated-changes/patch/values.yaml.patch) (<https://github.com/suse-edge/charts/blob/main/packages/neuvector-core/generated-changes/patch/values.yaml.patch>) ↗.

16.2 Important notes

- The Scanner container must have enough memory to pull the image to be scanned into memory and expand it. To scan images exceeding 1 GB, increase the scanner's memory to slightly above the largest expected image size.
- High network connections expected in Protect mode. The Enforcer requires CPU and memory when in Protect (inline firewall blocking) mode to hold and inspect connections and possible payload (DLP). Increasing memory and dedicating a CPU core to the Enforcer can ensure adequate packet filtering capacity.

16.3 Installing with Edge Image Builder

SUSE Edge is using [Chapter 9, Edge Image Builder](#) in order to customize base SLE Micro OS images. Follow [Section 21.7, "NeuVector Installation"](#) for an air-gapped installation of NeuVector on top of Kubernetes clusters provisioned by EIB.

17 MetalLB

See [MetalLB official documentation \(https://metallb.universe.tf/\)](https://metallb.universe.tf/).

MetalLB is a load-balancer implementation for bare-metal Kubernetes clusters, using standard routing protocols.

In bare-metal environments, setting up network load balancers is notably more complex than in cloud environments. Unlike the straightforward API calls in cloud setups, bare-metal requires either dedicated network appliances or a combination of load balancers and Virtual IP (VIP) configurations to manage High Availability (HA) or address the potential Single Point of Failure (SPOF) inherent in a single node load balancer. These configurations are not easily automated, posing challenges in Kubernetes deployments where components dynamically scale up and down.

MetalLB addresses these challenges by harnessing the Kubernetes model to create LoadBalancer type services as if they were operating in a cloud environment, even on bare-metal setups.

There are two different approaches, via [L2 mode \(https://metallb.universe.tf/concepts/layer2/\)](https://metallb.universe.tf/concepts/layer2/) (using *ARP tricks*) or via [BGP \(https://metallb.universe.tf/concepts/bgp/\)](https://metallb.universe.tf/concepts/bgp/). Mainly L2 does not need any special network gear but BGP is in general better. It depends on the use cases.

17.1 How does SUSE Edge use MetalLB?

SUSE Edge uses MetalLB in two key ways:

- As a Load Balancer Solution: MetalLB serves as the Load Balancer solution for bare-metal machines.
- For an HA K3s/RKE2 Setup: MetalLB allows for load balancing the Kubernetes API using a Virtual IP address.



Note

In order to be able to expose the API, the `endpoint-copier-operator` is used to keep in sync the K8s API endpoints from the 'kubernetes' service to a 'kubernetes-vip' LoadBalancer service.

17.2 Best practices

Installation of MetalLB in L2 mode is detailed in the MetalLB guide ([Chapter 19, MetalLB on K3s \(using L2\)](#)).

A guide on installing MetalLB in front of the kube-api-server to achieve HA setups can be found in the MetalLB in front of the Kubernetes API server ([Chapter 20, MetalLB in front of the Kubernetes API server](#)) tutorial.

17.3 Known issues

- K3S LoadBalancer Solution: K3S comes with its Load Balancer solution, [Klipper](#). To use MetalLB, Klipper must be disabled. This can be done by starting the K3s server with the `--disable servicelb` option, as described in the [K3s documentation \(https://docs.k3s.io/networking\)](https://docs.k3s.io/networking) [↗](#).

18 Edge Virtualization

This section describes how you can use Edge Virtualization to run virtual machines on your edge nodes. It is important to point out that Edge Virtualization is not a comprehensive solution and has limited features; it attempts to solve requirements for lightweight virtualization where basic virtual machine capabilities are required. SUSE provides a more comprehensive virtualization (and hyperconverged infrastructure) solution with [Harvester](https://harvesterhci.io/) (<https://harvesterhci.io/>) [↗](#).

SUSE Edge Virtualization supports two methods of running virtual machines:

1. Deploying the virtual machines manually via libvirt + qemu-kvm at the host level
2. Deploying the KubeVirt operator for Kubernetes-based management of virtual machines

Both options are valid, but only the second one is covered below. If you want to use the standard out-of-the box virtualization mechanisms provided by SLE Micro, a comprehensive guide can be found [here](https://documentation.suse.com/sles/15-SP5/html/SLES-all/chap-virtualization-introduction.html) (<https://documentation.suse.com/sles/15-SP5/html/SLES-all/chap-virtualization-introduction.html>) [↗](#), and whilst it was primarily written for SUSE Linux Enterprise Server, the concepts are almost identical.

This guide initially explains how to deploy the additional virtualization components onto a system that has already been pre-deployed, but follows with a section that describes how to embed this configuration in the initial deployment via Edge Image Builder. If you do not want to run through the basics and set things up manually, skip right ahead to that section.

18.1 KubeVirt overview

KubeVirt allows for managing Virtual Machines with Kubernetes alongside the rest of your containerized workloads. It does this by running the user space portion of the Linux virtualization stack in a container. This minimizes the requirements on the host system, allowing for easier setup and management.

Details about KubeVirt's architecture can be found in [the upstream documentation](https://kubevirt.io/user-guide/architecture/). (<https://kubevirt.io/user-guide/architecture/>) [↗](#)

18.2 Prerequisites

If you are following this guide, we assume you have the following already available:

- At least one physical host with SLE Micro 5.5+ installed, and with virtualization extensions enabled in the BIOS (see [here \(https://documentation.suse.com/sles/15-SP5/html/SLES-all/cha-virt-support.html#sec-kvm-requires-hardware\)](https://documentation.suse.com/sles/15-SP5/html/SLES-all/cha-virt-support.html#sec-kvm-requires-hardware) for details).
- Across your nodes, a K3s/RKE2 Kubernetes cluster already deployed and with an appropriate `kubeconfig` that enables superuser access to the cluster.
- Access to the root user — these instructions assume you are the root user, and *not* escalating your privileges via `sudo`.
- You have Helm (<https://helm.sh/docs/intro/install/>) available locally with an adequate network connection to be able to push configurations to your Kubernetes cluster and download the required images.

18.3 Manual installation of Edge Virtualization

This guide will not walk you through the deployment of Kubernetes, but it assumes that you have either installed the SUSE Edge-appropriate version of K3s (<https://k3s.io/>) or RKE2 (<https://docs.rke2.io/install/quickstart>) and that you have your `kubeconfig` configured accordingly so that standard `kubectl` commands can be executed as the superuser. We assume your node forms a single-node cluster, although there are no significant differences expected for multi-node deployments.

SUSE Edge Virtualization is deployed via three separate Helm charts, specifically:

- **KubeVirt:** The core virtualization components, that is, Kubernetes CRDs, operators and other components required for enabling Kubernetes to deploy and manage virtual machines.
- **KubeVirt Dashboard Extension:** An optional Rancher UI extension that allows basic virtual machine management, for example, starting/stopping of virtual machines as well as accessing the console.
- **Containerized Data Importer (CDI):** An additional component that enables persistent-storage integration for KubeVirt, providing capabilities for virtual machines to use existing Kubernetes storage back-ends for data, but also allowing users to import or clone data volumes for virtual machines.

Each of these Helm charts is versioned according to the SUSE Edge release you are currently using. For production/supported usage, employ the artifacts that can be found in the SUSE Registry.

First, ensure that your `kubectl` access is working:

```
$ kubectl get nodes
```

This should show something similar to the following:

NAME	STATUS	ROLES	AGE	VERSION
node1.edge.rdo.wales	Ready	control-plane,etcd,master	4h20m	v1.28.13+rke2r1
node2.edge.rdo.wales	Ready	control-plane,etcd,master	4h15m	v1.28.13+rke2r1
node3.edge.rdo.wales	Ready	control-plane,etcd,master	4h15m	v1.28.13+rke2r1

Now you can proceed to install the **KubeVirt** and **Containerized Data Importer (CDI)** Helm charts:

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

In a few minutes, you should have all KubeVirt and CDI components deployed. You can validate this by checking all the deployed resources in the `kubevirt-system` and `cdi-system` namespace.

Verify KubeVirt resources:

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

This should show something similar to the following:

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)
service/kubevirt-prometheus-metrics	ClusterIP	None	<none>	443/TCP
service/virt-api	ClusterIP	10.43.56.140	<none>	443/TCP
service/kubevirt-operator-webhook	ClusterIP	10.43.201.121	<none>	443/TCP
service/virt-exportproxy	ClusterIP	10.43.83.23	<none>	443/TCP

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

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

Verify CDI resources:

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

This should show something similar to the following:

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

```

To verify that the `VirtualMachine` custom resource definitions (CRDs) are deployed, you can validate with:

```
$ kubectl explain virtualmachine
```

This should print out the definition of the `VirtualMachine` object, which should print as follows:

```

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)

```


18.4 Deploying virtual machines

Now that KubeVirt and CDI are deployed, let us define a simple virtual machine based on [openSUSE Tumbleweed](https://get.opensuse.org/tumbleweed/) (<https://get.opensuse.org/tumbleweed/>) [↗](#). This virtual machine has the most simple of configurations, using standard "pod networking" for a networking configuration identical to any other pod. It also employs non-persistent storage, ensuring the storage is ephemeral, just like in any container that does not have a [PVC](https://kubernetes.io/docs/concepts/storage/persistent-volumes/) (<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:
I2Nsb3VklWNvbmZpZwpkaXNhYmXlX3Jvb3Q6IGZhbHNlCnNzaF9wd2F1dGg6IFRydWUKdXNlcnM6CiAgLSBkZWZhdWx0CiAgLSBuY
          name: cloudinitdisk
EOF
```

This should print that a `VirtualMachine` was created:

```
virtualmachine.kubevirt.io/tumbleweed created
```

This `VirtualMachine` definition is minimal, specifying little about the configuration. It simply outlines that it is a machine type "q35" (<https://wiki.qemu.org/Features/Q35>) [↗](#) with 2 GB of memory that uses a disk image based on an ephemeral `containerDisk` (that is, a disk image that is

stored in a container image from a remote image repository), and specifies a base64 encoded cloudInit disk, which we only use for user creation and password enforcement at boot time (use `base64 -d` to decode it).



Note

This virtual machine image is only for testing. The image is not officially supported and is only meant as a documentation example.

This machine takes a few minutes to boot as it needs to download the openSUSE Tumbleweed disk image, but once it has done so, you can view further details about the virtual machine by checking the virtual machine information:

```
$ kubectl get vmi
```

This should print the node that the virtual machine was started on, and the IP address of the virtual machine. Remember, since it uses pod networking, the reported IP address will be just like any other pod, and routable as such:

NAME	AGE	PHASE	IP	NODENAME	READY
tumbleweed	4m24s	Running	10.42.2.98	node3.edge.rdo.wales	True

When running these commands on the Kubernetes cluster nodes themselves, with a CNI that routes traffic directly to pods (for example, Cilium), you should be able to `ssh` directly to the machine itself. Substitute the following IP address with the one that was assigned to your virtual machine:

```
$ ssh suse@10.42.2.98
(password is "suse")
```

Once you are in this virtual machine, you can play around, but remember that it is limited in terms of resources, and only has 1 GB disk space. When you are finished, `Ctrl-D` or `exit` to disconnect from the SSH session.

The virtual machine process is still wrapped in a standard Kubernetes pod. The `VirtualMachine` CRD is a representation of the desired virtual machine, but the process in which the virtual machine is actually started is via the `virt-launcher` pod, a standard Kubernetes pod, just like any other application. For every virtual machine started, you can see there is a `virt-launcher` pod:

```
$ kubectl get pods
```

This should then show the one `virt-launcher` pod for the Tumbleweed machine that we have defined:

NAME	READY	STATUS	RESTARTS	AGE
virt-launcher-tumbleweed-8gcn4	3/3	Running	0	10m

If we take a look into this `virt-launcher` pod, you see it is executing `libvirt` and `qemu-kvm` processes. We can enter the pod itself and have a look under the covers, noting that you need to adapt the following command for your pod name:

```
$ kubectl exec -it virt-launcher-tumbleweed-8gcn4 -- bash
```

Once you are in the pod, try running `virsh` commands along with looking at the processes. You will see the `qemu-system-x86_64` binary running, along with certain processes for monitoring the virtual machine. You will also see the location of the disk image and how the networking is plugged (as a tap device):

```
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

```

Finally, let us delete this virtual machine to clean up:

```

$ kubectl delete vm/tumbleweed
virtualmachine.kubevirt.io "tumbleweed" deleted

```

18.5 Using virtctl

Along with the standard Kubernetes CLI tooling, that is, `kubectl`, KubeVirt comes with an accompanying CLI utility that allows you to interface with your cluster in a way that bridges some gaps between the virtualization world and the world that Kubernetes was designed for. For example, the `virtctl` tool provides the capability of managing the lifecycle of virtual machines (starting, stopping, restarting, etc.), providing access to the virtual consoles, uploading virtual machine images, as well as interfacing with Kubernetes constructs such as services, without using the API or CRDs directly.

Let us download the latest stable version of the `virtctl` tool:

```

$ export VERSION=v1.1.0
$ wget https://github.com/kubevirt/kubevirt/releases/download/${VERSION}/virtctl-
${VERSION}-linux-amd64

```

If you are using a different architecture or a non-Linux machine, you can find other releases [here \(https://github.com/kubevirt/kubevirt/releases\)](https://github.com/kubevirt/kubevirt/releases). You need to make this executable before proceeding, and it may be useful to move it to a location within your `$PATH`:

```

$ mv virtctl-${VERSION}-linux-amd64 /usr/local/bin/virtctl
$ chmod a+x /usr/local/bin/virtctl

```

You can then use the `virtctl` command-line tool to create virtual machines. Let us replicate our previous virtual machine, noting that we are piping the output directly into `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
"I2Nsb3VklWNvbmZpZwpkaXNhYmxlX3Jvb3Q6IGZhbHNLcNzaF9wd2F1dGg6IFRydWUKdXNlcnM6CiAgLSBkZWZhdWx0CiAgLSBuY
| kubectl apply -f -
```

This should then show the virtual machine running (it should start a lot quicker this time given that the container image will be cached):

```
$ kubectl get vmi
NAME                AGE   PHASE     IP              NODENAME                READY
virtctl-example    52s   Running   10.42.2.29     node3.edge.rdo.wales    True
```

Now we can use `virtctl` to connect directly to the virtual machine:

```
$ virtctl ssh suse@virtctl-example
(password is "suse" - Ctrl-D to exit)
```

There are plenty of other commands that can be used by `virtctl`. For example, `virtctl console` can give you access to the serial console if networking is not working, and you can use `virtctl guestosinfo` to get comprehensive OS information, subject to the guest having the `gemu-guest-agent` installed and running.

Finally, let us pause and resume the virtual machine:

```
$ virtctl pause vm virtctl-example
VMI virtctl-example was scheduled to pause
```

You find that the `VirtualMachine` object shows as **Paused** and the `VirtualMachineInstance` object shows as **Running** but **READY = False**:

```
$ 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
```

You also find that you can no longer connect to the virtual machine:

```
$ virtctl ssh suse@virtctl-example
can't access VMI virtctl-example: Operation cannot be fulfilled on
virtualmachineinstance.kubevirt.io "virtctl-example": VMI is paused
```

Let us resume the virtual machine and try again:

```
$ virtctl unpause vm virtctl-example
```

```
VMI virtctl-example was scheduled to unpause
```

Now we should be able to re-establish a connection:

```
$ virtctl ssh suse@virtctl-example
suse@vmi/virtctl-example.default's password:
suse@virtctl-example:~> exit
logout
```

Finally, let us remove the virtual machine:

```
$ kubectl delete vm/virtctl-example
virtualmachine.kubevirt.io "virtctl-example" deleted
```

18.6 Simple ingress networking

In this section, we show how you can expose virtual machines as standard Kubernetes services and make them available via the Kubernetes ingress service, for example, [NGINX with RKE2 \(https://docs.rke2.io/networking/networking_services#nginx-ingress-controller\)](https://docs.rke2.io/networking/networking_services#nginx-ingress-controller) or [Traefik with K3s \(https://docs.k3s.io/networking/networking-services#traefik-ingress-controller\)](https://docs.k3s.io/networking/networking-services#traefik-ingress-controller). This document assumes that these components are already configured appropriately and that you have an appropriate DNS pointer, for example, via a wild card, to point at your Kubernetes server nodes or your ingress virtual IP for proper ingress resolution.



Note

In SUSE Edge 3.0 + , if you are using K3s in a multi-server node configuration, you might have needed to configure a MetalLB-based VIP for Ingress; this is not required for RKE2.

In the example environment, another openSUSE Tumbleweed virtual machine is deployed, cloud-init is used to install NGINX as a simple Web server at boot time, and a simple message is configured to be returned to verify that it works as expected when a call is made. To see how this is done, simply `base64 -d` the cloud-init section in the output below.

Let us create this virtual machine now:

```
$ 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:
I2Nsb3VklWNvbmZpZwpkaXNhYmxlX3Jvb3Q6IGZhbHNLcnNzaF9wd2F1dGg6IFRydWUKdXNlcuM6CiAgLSBkZWZhdWx0CiAgLSBuY
          name: cloudinitdisk
EOF

```

When this virtual machine has successfully started, we can use the `virtctl` command to expose the `VirtualMachineInstance` with an external port of `8080` and a target port of `80` (where NGINX listens by default). We use the `virtctl` command here as it understands the mapping between the virtual machine object and the pod. This creates a new service for us:

```

$ virtctl expose vmi ingress-example --port=8080 --target-port=80 --name=ingress-example
Service ingress-example successfully exposed for vmi ingress-example

```

We will then have an appropriate service automatically created:

```

$ kubectl get svc/ingress-example

```

NAME	AGE	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
ingress-example	9s	ClusterIP	10.43.217.19	<none>	8080/TCP

Next, if you then use `kubectl create ingress`, we can create an ingress object that points to this service. Adapt the URL (known as the "host" in the [ingress \(https://kubernetes.io/docs/reference/kubectl/generated/kubectl_create/kubectl_create_ingress/\)](https://kubernetes.io/docs/reference/kubectl/generated/kubectl_create/kubectl_create_ingress/) object) here to match your DNS configuration and ensure that you point it to port `8080`:

```
$ kubectl create ingress ingress-example --rule=ingress-example.suse.local/=ingress-example:8080
```

With DNS being configured correctly, you should be able to curl the URL immediately:

```
$ curl ingress-example.suse.local
It works!
```

Let us clean up by removing this virtual machine and its service and ingress resources:

```
$ 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
```

18.7 Using the Rancher UI extension

SUSE Edge Virtualization provides a UI extension for Rancher Manager, enabling basic virtual machine management using the Rancher dashboard UI.

18.7.1 Installation

See Rancher Dashboard Extensions ([Chapter 5, Rancher Dashboard Extensions](#)) for installation guidance.

18.7.2 Using KubeVirt Rancher Dashboard Extension

The extension introduces a new **KubeVirt** section to the Cluster Explorer. This section is added to any managed cluster which has KubeVirt installed.

The extension allows you to directly interact with two KubeVirt resources:

1. Virtual Machine instances — A resource representing a single running virtual machine instance.
2. Virtual Machines — A resource used to manage virtual machines lifecycle.

18.7.2.1 Creating a virtual machine

1. Navigate to **Cluster Explorer** clicking KubeVirt-enabled managed cluster in the left navigation.
2. Navigate to **KubeVirt > Virtual Machines** page and click Create from YAML in the upper right of the screen.
3. Fill in or paste a virtual machine definition and press Create. Use virtual machine definition from Deploying Virtual Machines section as an inspiration.

The screenshot displays the Rancher Dashboard interface for the 'local' cluster. The left-hand navigation sidebar is visible, with the 'Virtual Machines' option under the 'KubeVirt' section highlighted in blue. The main content area is titled 'Virtual Machines' and features two buttons: 'Start' (with a play icon) and 'Stop' (with an 'X' icon). Below these buttons is a table with two columns: 'State' and 'Name'. The table lists two virtual machines: 'testvm' with a state of 'Off' and 'tumblevm' with a state of 'Running'. The 'testvm' entry also includes the text 'VMI does not exist' below its state indicator.

18.7.2.2 Starting and stopping virtual machines




Start and stop virtual machines using the action menu accessed from the # drop-down list to the right of each virtual machine or use group actions at the top of the list by selecting virtual machines to perform the action on.

It is possible to run start and stop actions only on the virtual machines which have `spec.running` property defined. In case when `spec.runStrategy` is used, it is not possible to directly start and stop such a machine. For more information, see [KubeVirt documentation \(https://kubevirt.io/user-guide/virtual_machines/run_strategies/#run-strategies\)](https://kubevirt.io/user-guide/virtual_machines/run_strategies/#run-strategies).

18.7.2.3 Accessing virtual machine console

The "Virtual machines" list provides a Console drop-down list that allows to connect to the machine using **VNC or Serial Console**. This action is only available to running machines.

In some cases, it takes a short while before the console is accessible on a freshly started virtual machine.

☰	 local
	Cluster >
	Workloads >
	Apps >
	Service Discovery >
	Storage >
	Policy >
	KubeVirt ▾
	Virtual Machine Instances {=} 1
	Virtual Machines {=} 2
	More Resources >

Virtual Machines

Not Secure

Shortcut Keys

```

04:33:18 +0
ci-info: no
ci-info: no
<14>Mar 15
<14>Mar 15
<14>Mar 15
eed (DSA)
<14>Mar 15
eed (ECDSA)
<14>Mar 15
eed (ED25519)
<14>Mar 15
eed (RSA)
<14>Mar 15
<14>Mar 15
-----BEGIN
ecdsa-sha2-
gkmZ5iBXiyx
ssh-ed25519
ssh-rsa AAA
SdkCVi0ox+M
Kz7tFQ1vDIS
EEIGIU9qrN/
0xyFi2KCb/g
2sh/jvJRIbR
-----END SS
[ 27.1844
Datasource
[ OK ] Fi
[ OK ] Re
Welcome to
  
```

18.8 Installing with Edge Image Builder

SUSE Edge is using *Chapter 9, Edge Image Builder* in order to customize base SLE Micro OS images. Follow *Section 21.9, “KubeVirt and CDI Installation”* for an air-gapped installation of both KubeVirt and CDI on top of Kubernetes clusters provisioned by EIB.

III How-To Guides

- 19 MetalLB on K3s (using L2) 155
- 20 MetalLB in front of the Kubernetes API server 164
- 21 Air-gapped deployments with Edge Image Builder 171

How-to guides and best practices

19 MetalLB on K3s (using L2)

MetalLB is a load-balancer implementation for bare-metal Kubernetes clusters, using standard routing protocols.

In this guide, we demonstrate how to deploy MetalLB in layer 2 mode.

19.1 Why use this method

MetalLB is a compelling choice for load balancing in bare-metal Kubernetes clusters for several reasons:

1. **Native Integration with Kubernetes:** MetalLB seamlessly integrates with Kubernetes, making it easy to deploy and manage using familiar Kubernetes tools and practices.
2. **Bare-Metal Compatibility:** Unlike cloud-based load balancers, MetalLB is designed specifically for on-premises deployments where traditional load balancers might not be available or feasible.
3. **Supports Multiple Protocols:** MetalLB supports both Layer 2 and BGP (Border Gateway Protocol) modes, providing flexibility for different network architectures and requirements.
4. **High Availability:** By distributing load-balancing responsibilities across multiple nodes, MetalLB ensures high availability and reliability for your services.
5. **Scalability:** MetalLB can handle large-scale deployments, scaling alongside your Kubernetes cluster to meet increasing demand.

In layer 2 mode, one node assumes the responsibility of advertising a service to the local network. From the network's perspective, it simply looks like that machine has multiple IP addresses assigned to its network interface.

The major advantage of the layer 2 mode is its universality: it works on any Ethernet network, with no special hardware required, not even fancy routers.

19.2 MetalLB on K3s (using L2)

In this quick start, L2 mode will be used, so it means we do not need any special network gear but just a couple of free IPs in our network range, ideally outside of the DHCP pool so they are not assigned.


In this example, our DHCP pool is 192.168.122.100-192.168.122.200 (yes, three IPs, see Traefik and MetalLB ([Section 19.3.3, "Traefik and MetalLB"](#)) for the reason of the extra IP) for a 192.168.122.0/24 network, so anything outside this range is OK (besides the gateway and other hosts that can be already running!)

19.3 Prerequisites

- A K3s cluster where MetalLB is going to be deployed.



Warning

K3S comes with its own service load balancer named Klipper. You **need to disable it to run MetalLB** (<https://metallb.universe.tf/configuration/k3s/>) . To disable Klipper, K3s needs to be installed using the `--disable=service-lb` flag.

- Helm
- A couple of free IPs in our network range. In this case, 192.168.122.10-192.168.122.12

19.3.1 Deployment

MetalLB leverages Helm (and other methods as well), so:

```
helm install \
  metallb oci://registry.suse.com/edge/metallb-chart \
  --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
```

19.3.2 Configuration

At this point, the installation is completed. Now it is time to [configure \(https://metallb.universe.tf/configuration/\)](https://metallb.universe.tf/configuration/) using our example values:

```
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
```

Now, it is ready to be used. You can customize many things for L2 mode, such as:

- [IPv6 And Dual Stack Services \(https://metallb.universe.tf/usage/#ipv6-and-dual-stack-services\)](https://metallb.universe.tf/usage/#ipv6-and-dual-stack-services)
- [Control automatic address allocation \(https://metallb.universe.tf/configuration/_advanced_ipaddresspool_configuration/#controlling-automatic-address-allocation\)](https://metallb.universe.tf/configuration/_advanced_ipaddresspool_configuration/#controlling-automatic-address-allocation)
- [Reduce the scope of address allocation to specific namespaces and services \(https://metallb.universe.tf/configuration/_advanced_ipaddresspool_configuration/#reduce-scope-of-address-allocation-to-specific-namespace-and-service\)](https://metallb.universe.tf/configuration/_advanced_ipaddresspool_configuration/#reduce-scope-of-address-allocation-to-specific-namespace-and-service)

- Limiting the set of nodes where the service can be announced from (https://metallb.universe.tf/configuration/_advanced_l2_configuration/#limiting-the-set-of-nodes-where-the-service-can-be-announced-from)
- Specify network interfaces that LB IP can be announced from (https://metallb.universe.tf/configuration/_advanced_l2_configuration/#specify-network-interfaces-that-lb-ip-can-be-announced-from)

And a lot more for BGP (https://metallb.universe.tf/configuration/_advanced_bgp_configuration/).

19.3.3 Traefik and MetalLB

Traefik is deployed by default with K3s (it can be disabled (<https://docs.k3s.io/networking#traefik-ingress-controller>) with `--disable=traefik`) and it is by default exposed as `LoadBalancer` (to be used with Klipper). However, as Klipper needs to be disabled, Traefik service for ingress is still a `LoadBalancer` type. So at the moment of deploying MetalLB, the first IP will be assigned automatically to Traefik Ingress.

```
# 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
```

This will be applied later (*Section 19.4, "Ingress with MetalLB"*) in the process.

19.3.4 Usage

Let us create an example deployment:

```
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: "paulbouver/hello-kubernetes:1.10"

```

EOF

And finally, the service:

```

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

```

Let us see it in action:

```

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">
  
  <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>
</body>
</html>

```

19.4 Ingress with MetalLB

As Traefik is already serving as an ingress controller, we can expose any HTTP/HTTPS traffic via an Ingress object such as:

```

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

```

And then:

```

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">
    
    <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>
```

Also, to verify that MetalLB works correctly, arping can be used as:

arping hellok3s.\${IP}.sslip.io

Expected result:

```
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
```

In the example above, the traffic flows as follows:

1. hellok3s.\${IP}.sslip.io is resolved to the actual IP.
2. Then the traffic is handled by the metallb-speaker pod.
3. metallb-speaker redirects the traffic to the traefik controller.
4. Finally, Traefik forwards the request to the hello-kubernetes service.

20 MetalLB in front of the Kubernetes API server

This guide demonstrates using a MetalLB service to expose the RKE2/K3s API externally on an HA cluster with three control-plane nodes. To achieve this, a Kubernetes Service of type Load-Balancer and Endpoints will be manually created. The Endpoints keep the IPs of all control plane nodes available in the cluster. For the Endpoint to be continuously synchronized with the events occurring in the cluster (adding/removing a node or a node goes offline), the Endpoint Copier Operator (<https://github.com/suse-edge/endpoint-copier-operator>) will be deployed. The operator monitors the events happening in the default kubernetes Endpoint and updates the managed one automatically to keep them in sync. Since the managed Service is of type Load-Balancer, MetalLB assigns it a static ExternalIP. This ExternalIP will be used to communicate with the API Server.

20.1 Prerequisites

- Three hosts to deploy RKE2/K3s on top.
 - Ensure the hosts have different host names.
 - For testing, these could be virtual machines
- At least 2 available IPs in the network (one for the Traefik/Nginx and one for the managed service).
- Helm

20.2 Installing RKE2/K3s



Note

If you do not want to use a fresh cluster but want to use an existing one, skip this step and proceed to the next one.

First, a free IP in the network must be reserved that will be used later for ExternalIP of the managed Service.

SSH to the first host and install the wanted distribution in cluster mode.

For 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)
```

For 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 -
```



Note

Make sure that `--disable=serviceLB` flag is provided in the `k3s server` command.



Important

From now on, the commands should be run on the local machine.

To access the API server from outside, the IP of the RKE2/K3s VM will be used.

```
# 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
```


20.3 Configuring an existing cluster



Note

This step is valid only if you intend to use an existing RKE2/K3s cluster.

To use an existing cluster the `tls-san` flags should be modified and also, `serviceLB` LB should be disabled for K3s.

To change the flags for RKE2 or K3s servers, you need to modify either the `/etc/systemd/system/rke2.service` or `/etc/systemd/system/k3s.service` file on all the VMs in the cluster, depending on the distribution.

The flags should be inserted in the `ExecStart`. For example:

For 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' \
```

For 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' \
```

Then the following commands should be executed to load the new configurations:

```
systemctl daemon-reload
systemctl restart ${KUBE_DISTRIBUTION}
```

20.4 Installing MetalLB

To deploy `MetalLB`, the `MetalLB on K3s` (<https://suse-edge.github.io/docs/quickstart/metallb>)  guide can be used.

NOTE: Ensure that the IP addresses of the `ip-pool` `IPAddressPool` do not overlap with the IP addresses previously selected for the `LoadBalancer` service.

Create a separate `IPAddressPool` that will be used only for the managed Service.

```
# 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
```

20.5 Installing the Endpoint Copier Operator

```
helm install \
endpoint-copier-operator oci://registry.suse.com/edge/endpoint-copier-operator-chart \
--namespace endpoint-copier-operator \
--create-namespace
```

The command above will deploy the `endpoint-copier-operator` operator Deployment with two replicas. One will be the leader and the other will take over the leader role if needed.

Now, the `kubernetes-vip` Service should be deployed, which will be reconciled by the operator and an Endpoint with the configured ports and IP will be created.

For 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
```

For 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: 443
    protocol: TCP
    targetPort: 6443
  sessionAffinity: None
  type: LoadBalancer
```

```
EOF
```

Verify that the `kubernetes-vip` Service has the correct IP address:

```
kubectl get service kubernetes-vip -n default \
-o=jsonpath='{.status.loadBalancer.ingress[0].ip}'
```

Ensure that the `kubernetes-vip` and `kubernetes` Endpoints resources in the `default` namespace point to the same IPs.

```
kubectl get endpoints kubernetes kubernetes-vip
```

If everything is correct, the last thing left is to use the `VIP_SERVICE_IP` in our `Kubeconfig`.

```
sed -i '' "s/${NODE_IP}/${VIP_SERVICE_IP}/g" ~/.kube/config
```

From now on, all the `kubectl` will go through the `kubernetes-vip` service.

20.6 Adding control-plane nodes

To monitor the entire process, two more terminal tabs can be opened.

First terminal:

```
watch kubectl get nodes
```

Second terminal:

```
watch kubectl get endpoints
```

Now execute the commands below on the second and third nodes.

For 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
```

For 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=service\
--write-kubeconfig-mode=644" K3S_TOKEN=foobar sh -
```

21 Air-gapped deployments with Edge Image Builder

21.1 Intro

This guide will show how to deploy several of the SUSE Edge components completely air-gapped on SLE Micro 5.5 utilizing Edge Image Builder(EIB) ([Chapter 9, Edge Image Builder](#)). With this, you'll be able to boot into a customized, ready to boot (CRB) image created by EIB and have the specified components deployed on either a RKE2 or K3s cluster without an Internet connection or any manual steps. This configuration is highly desirable for customers that want to pre-bake all artifacts required for deployment into their OS image, so they are immediately available on boot.

We will cover an air-gapped installation of:

- [Chapter 4, Rancher](#)
- [Chapter 16, NeuVector](#)
- [Chapter 15, Longhorn](#)
- [Chapter 18, Edge Virtualization](#)



Warning

EIB will parse and pre-download all images referenced in the provided Helm charts and Kubernetes manifests. However, some of those may be attempting to pull container images and create Kubernetes resources based on those at runtime. In these cases we have to manually specify the necessary images in the definition file if we want to set up a completely air-gapped environment.

21.2 Prerequisites

If you're following this guide, it's assumed that you are already familiar with EIB ([Chapter 9, Edge Image Builder](#)). If not, please follow the quick start guide ([Chapter 3, Standalone clusters with Edge Image Builder](#)) to better understand the concepts shown in practice below.

21.3 Libvirt Network Configuration



Note

To demo the air-gapped deployment, this guide will be done using a simulated air-gapped libvirt network and the following configuration will be tailored to that. For your own deployments, you may have to modify the host1.local.yaml configuration that will be introduced in the next step.

If you would like to use the same libvirt network configuration, follow along. If not, skip to [Section 21.4, "Base Directory Configuration"](#).

Let's create an isolated network configuration with an IP address range 192.168.100.2/24 for 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
```

Now, the only thing left is to create the network and start it:

```
virsh net-define isolatednetwork.xml
virsh net-start isolatednetwork
```

21.4 Base Directory Configuration

The base directory configuration is the same across all different components, so we will set it up here.

We will first create the necessary subdirectories:

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

```
mkdir -p $CONFIG_DIR/network
mkdir -p $CONFIG_DIR/kubernetes/helm/values
```

Make sure to add whichever base image you plan to use into the `base-images` directory. This guide will focus on the Self Install ISO found [here \(https://www.suse.com/download/sle-micro/\)](https://www.suse.com/download/sle-micro/).

Let's copy the downloaded image:

```
cp SLE-Micro.x86_64-5.5.0-Default-SelfInstall-GM2.install.iso $CONFIG_DIR/base-images/slemicro.iso
```



Note

EIB is never going to modify the base image input.

Let's create a file containing the desired network configuration:

```
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
```

This configuration ensures the following are present on the provisioned systems (using the specified MAC address):

- an Ethernet interface with a static IP address
- routing
- DNS
- hostname (host1.local)

The resulting file structure should now look like:

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

21.5 Base Definition File

Edge Image Builder is using *definition files* to modify the SLE Micro images. These files contain the majority of configurable options. Many of these options will be repeated across the different component sections, so we will list and explain those here.



Tip

Full list of customization options in the definition file can be found in the [upstream documentation \(https://github.com/suse-edge/edge-image-builder/blob/release-1.0/docs/building-images.md#image-definition-file\)](https://github.com/suse-edge/edge-image-builder/blob/release-1.0/docs/building-images.md#image-definition-file)

We will take a look at the following fields which will be present in all definition files:

```
apiVersion: 1.0
image:
  imageType: iso
```

```
arch: x86_64
baseImage: slemicro.iso
outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword: $6$jHugJNNd3HElGsUZ
$eodjVe4te5ps44SVcWshdfWizrP.xAyd71CVEXazBJ/.v799/WRCBXxfYmunlB02yp1hm/zb4r8EmnrrNCF.P/
kubernetes:
  version: v1.28.13+rke2r1
embeddedArtifactRegistry:
  images:
    - ...
```

The `image` section is required, and it specifies the input image, its architecture and type, as well as what the output image will be called.

The `operatingSystem` section is optional, and contains configuration to enable login on the provisioned systems with the `root/eib` username/password.

The `kubernetes` section is optional, and it defines the Kubernetes type and version. We are going to use Kubernetes 1.28.13 and RKE2 by default. Use `kubernetes.version: v1.28.13+k3s1` if K3s is desired instead. Unless explicitly configured via the `kubernetes.nodes` field, all clusters we bootstrap in this guide will be single-node ones.

The `embeddedArtifactRegistry` section will include all images which are only referenced and pulled at runtime for the specific component.

21.6 Rancher Installation



Note

The Rancher ([Chapter 4, Rancher](#)) deployment that will be demonstrated will be highly slimmed down for demonstration purposes. For your actual deployments, additional artifacts may be necessary depending on your configuration.

The [Rancher v2.8.8](https://prime.ribs.rancher.io/rancher/v2.8.8/rancher-images.txt) (<https://prime.ribs.rancher.io/rancher/v2.8.8/rancher-images.txt>) [↗](#) container images file lists all the images required for an air-gapped installation.

There are over 600 container images in total which means that the resulting CRB image would be roughly 30GB. For our Rancher installation, we will strip down that list to the smallest working configuration. From there, you can add back any images you may need for your deployments.

We will create the definition file and include the stripped down image list:

```
apiVersion: 1.0
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword: $6$jHugJNNd3HElGsUZ
$eodjVe4te5ps44SVcWshdfWizrP.xAyd71CVEXazBJ/.v799/WRCBXxfYmunlB02yp1hm/zb4r8EmnrrNCF.P/
kubernetes:
  version: v1.28.13+rke2r1
  network:
    apiVIP: 192.168.100.151
  manifests:
    urls:
      - https://github.com/cert-manager/cert-manager/releases/download/v1.14.2/cert-
manager.crd.yaml
  helm:
    charts:
      - name: rancher
        version: 2.8.8
        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: 1.14.2
    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:v4.0.3
    - name: registry.rancher.com/rancher/calico-cni:v3.27.4-rancher1
    - name: registry.rancher.com/rancher/cis-operator:v1.0.15
    - name: registry.rancher.com/rancher/coreos-kube-state-metrics:v1.9.7
```

- name: registry.rancher.com/rancher/coreos-prometheus-config-reloader:v0.38.1
- name: registry.rancher.com/rancher/coreos-prometheus-operator:v0.38.1
- name: registry.rancher.com/rancher/flannel-cni:v0.3.0-rancher9
- name: registry.rancher.com/rancher/fleet-agent:v0.9.9
- name: registry.rancher.com/rancher/fleet:v0.9.9
- name: registry.rancher.com/rancher/gitjob:v0.9.13
- name: registry.rancher.com/rancher/grafana-grafana:7.1.5
- name: registry.rancher.com/rancher/hardened-addon-resizer:1.8.20-build20240410
- name: registry.rancher.com/rancher/hardened-calico:v3.28.1-build20240806
- name: registry.rancher.com/rancher/hardened-cluster-autoscaler:v1.8.10-build20240124
- name: registry.rancher.com/rancher/hardened-cni-plugins:v1.5.1-build20240805
- name: registry.rancher.com/rancher/hardened-coredns:v1.11.1-build20240305
- name: registry.rancher.com/rancher/hardened-dns-node-cache:1.22.28-build20240125
- name: registry.rancher.com/rancher/hardened-etcd:v3.5.13-k3s1-build20240531
- name: registry.rancher.com/rancher/hardened-flannel:v0.25.5-build20240801
- name: registry.rancher.com/rancher/hardened-k8s-metrics-server:v0.7.1-build20240401
- name: registry.rancher.com/rancher/hardened-kubernetes:v1.28.13-rke2r1-build20240815
- name: registry.rancher.com/rancher/hardened-multus-cni:v4.0.2-build20240612
- name: registry.rancher.com/rancher/hardened-node-feature-discovery:v0.15.4-build20240513
- name: registry.rancher.com/rancher/hardened-whereabouts:v0.7.0-build20240429
- name: registry.rancher.com/rancher/helm-project-operator:v0.2.1
- name: registry.rancher.com/rancher/istio-kubectll:1.5.10
- name: registry.rancher.com/rancher/jimmydyson-configmap-reload:v0.3.0
- name: registry.rancher.com/rancher/k3s-upgrade:v1.28.13-k3s1
- name: registry.rancher.com/rancher/klipper-helm:v0.8.4-build20240523
- name: registry.rancher.com/rancher/klipper-lb:v0.4.9
- name: registry.rancher.com/rancher/kube-api-auth:v0.2.1
- name: registry.rancher.com/rancher/kubectll:v1.28.12
- name: registry.rancher.com/rancher/library-nginx:1.19.2-alpine
- name: registry.rancher.com/rancher/local-path-provisioner:v0.0.28
- name: registry.rancher.com/rancher/machine:v0.15.0-rancher116
- name: registry.rancher.com/rancher/mirrored-cluster-api-controller:v1.4.4
- name: registry.rancher.com/rancher/nginx-ingress-controller:v1.10.4-hardened2
- name: registry.rancher.com/rancher/pause:3.6
- name: registry.rancher.com/rancher/prom-alertmanager:v0.21.0
- name: registry.rancher.com/rancher/prom-node-exporter:v1.0.1
- name: registry.rancher.com/rancher/prom-prometheus:v2.18.2
- name: registry.rancher.com/rancher/prometheus-auth:v0.2.2
- name: registry.rancher.com/rancher/prometheus-federator:v0.3.4
- name: registry.rancher.com/rancher/pushprox-client:v0.1.3-rancher2-client
- name: registry.rancher.com/rancher/pushprox-proxy:v0.1.3-rancher2-proxy
- name: registry.rancher.com/rancher/rancher-agent:v2.8.8
- name: registry.rancher.com/rancher/rancher-csp-adapter:v3.0.1
- name: registry.rancher.com/rancher/rancher-webhook:v0.4.11

```

- name: registry.rancher.com/rancher/rancher:v2.8.8
- name: registry.rancher.com/rancher/rke-tools:v0.1.102
- name: registry.rancher.com/rancher/rke2-cloud-provider:v1.29.3-build20240515
- name: registry.rancher.com/rancher/rke2-runtime:v1.28.13-rke2r1
- name: registry.rancher.com/rancher/rke2-upgrade:v1.28.13-rke2r1
- name: registry.rancher.com/rancher/security-scan:v0.2.17
- name: registry.rancher.com/rancher/shell:v0.1.26
- name: registry.rancher.com/rancher/system-agent-installer-k3s:v1.28.13-k3s1
- name: registry.rancher.com/rancher/system-agent-installer-rke2:v1.28.13-rke2r1
- name: registry.rancher.com/rancher/system-agent:v0.3.9-suc
- name: registry.rancher.com/rancher/system-upgrade-controller:v0.13.4
- name: registry.rancher.com/rancher/ui-plugin-catalog:2.1.0
- name: registry.rancher.com/rancher/ui-plugin-operator:v0.1.1
- name: registry.rancher.com/rancher/webhook-receiver:v0.2.5
- name: registry.rancher.com/rancher/kubect1:v1.20.2
- name: registry.rancher.com/rancher/shell:v0.1.24
- name: registry.rancher.com/rancher/mirrored-ingress-nginx-kube-webhook-
certgen:v1.4.1
  - name: registry.rancher.com/rancher/mirrored-ingress-nginx-kube-webhook-
certgen:v20221220-controller-v1.5.1-58-g787ea74b6
  - name: registry.rancher.com/rancher/mirrored-ingress-nginx-kube-webhook-
certgen:v20230312-helm-chart-4.5.2-28-g66a760794
  - name: registry.rancher.com/rancher/mirrored-ingress-nginx-kube-webhook-
certgen:v20231011-8b53cabe0
  - name: registry.rancher.com/rancher/mirrored-ingress-nginx-kube-webhook-
certgen:v20231226-1a7112e06

```

As compared to the full list of 602 container images, this slimmed down version only contains 62 which makes the new CRB image only about 7GB.

We also need to create a Helm values file for 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

```



Warning

Setting the `systemDefaultRegistry` to `registry.rancher.com` allows Rancher to automatically look for images in the embedded artifact registry started within the CRB image at boot. Omitting this field may result in failure to find the container images on the node.

Let's build the image:

```
podman run --rm -it --privileged -v $CONFIG_DIR:/eib \
registry.suse.com/edge/edge-image-builder:1.0.2 \
build --definition-file eib-iso-definition.yaml
```

The output should be similar to the following:

```
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]
Downloading file: dl-manifest-1.yaml 100% (437/437 kB, 17 MB/s)
Populating Embedded Artifact Registry... 100% (69/69, 26 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% (780/780 MB, 115 MB/s)
Downloading file: rke2-images-cilium.linux-amd64.tar.zst 100% (367/367 MB, 108 MB/s)
Downloading file: rke2.linux-amd64.tar.gz 100% (34/34 MB, 117 MB/s)
Downloading file: sha256sum-amd64.txt 100% (3.9/3.9 kB, 34 MB/s)
Downloading file: dl-manifest-1.yaml 100% (437/437 kB, 106 MB/s)
Kubernetes ..... [SUCCESS]
Certificates ..... [SKIPPED]
Building ISO image...
Kernel Params ..... [SKIPPED]
Image build complete!
```

Once a node using the built image is provisioned, we can verify the Rancher installation:

```
/var/lib/rancher/rke2/bin/kubectl get all -A --kubeconfig /etc/rancher/rke2/rke2.yaml
```

The output should be similar to the following, showing that everything has been successfully deployed:

NAMESPACE	READY	STATUS	RESTARTS	NAME	AGE
-----------	-------	--------	----------	------	-----

cattle-fleet-local-system	1/1	Running	0	pod/fleet-agent-68f4d5d5f7-tdlk7
				34s
cattle-fleet-system	1/1	Running	0	pod/fleet-controller-85564cc978-pbtvk
				5m51s
cattle-fleet-system	1/1	Running	0	pod/gitjob-9dc58fb5b-7csw
				5m51s
cattle-provisioning-capi-system	1/1	Running	0	pod/capi-controller-manager-5c57b4b8f7-wlp5k
				4m52s
cattle-system	0/2	Completed	0	pod/helm-operation-4fk5c
				37s
cattle-system	0/2	Completed	0	pod/helm-operation-6zgbq
				4m54s
cattle-system	0/2	Completed	0	pod/helm-operation-cjds5
				5m37s
cattle-system	0/2	Completed	0	pod/helm-operation-kt5c2
				5m21s
cattle-system	0/2	Completed	0	pod/helm-operation-ppgtw
				5m30s
cattle-system	0/2	Completed	0	pod/helm-operation-tvcwk
				5m54s
cattle-system	0/2	Completed	0	pod/helm-operation-wpxd4
				53s
cattle-system	1/1	Running	0	pod/rancher-58575f9575-svrg2
				6m34s
cattle-system	1/1	Running	0	pod/rancher-webhook-5c6556f7ff-vgmkt
				5m19s
cert-manager	1/1	Running	0	pod/cert-manager-6c69f9f796-fkm8f
				7m14s
cert-manager	1/1	Running	0	pod/cert-manager-cainjector-584f44558c-wg7p6
				7m14s
cert-manager	1/1	Running	0	pod/cert-manager-webhook-76f9945d6f-lv2nv
				7m14s
endpoint-copier-operator	1/1	Running	0	pod/endpoint-copier-operator-58964b659b-l64dk
				7m16s
endpoint-copier-operator	1/1	Running	0	pod/endpoint-copier-operator-58964b659b-z9t9d
				7m16s
kube-system	1/1	Running	0	pod/cilium-fht55
				7m32s
kube-system	1/1	Running	0	pod/cilium-operator-558bbf6cfd-gwfwf
				7m32s
kube-system	0/1	Pending	0	pod/cilium-operator-558bbf6cfd-qsb5
				7m32s
kube-system	1/1	Running	0	pod/cloud-controller-manager-host1.local
				7m21s
kube-system	1/1	Running	0	pod/etcd-host1.local
				7m8s

kube-system	0/1	Completed	0	pod/helm-install-cert-manager-fvbtt
				8m12s
kube-system	0/1	Completed	0	pod/helm-install-endpoint-copier-operator-5kkgw
				8m12s
kube-system	0/1	Completed	0	pod/helm-install-metallb-zfphb
				8m12s
kube-system	0/1	Completed	2	pod/helm-install-rancher-nc4nt
				8m12s
kube-system	0/1	Completed	0	pod/helm-install-rke2-cilium-7wq87
				8m12s
kube-system	0/1	Completed	0	pod/helm-install-rke2-coredns-nl4gc
				8m12s
kube-system	0/1	Completed	0	pod/helm-install-rke2-ingress-nginx-svjqd
				8m12s
kube-system	0/1	Completed	0	pod/helm-install-rke2-metrics-server-gqqgz
				8m12s
kube-system	0/1	Completed	0	pod/helm-install-rke2-snapshot-controller-crd-r6b5p
				8m12s
kube-system	0/1	Completed	1	pod/helm-install-rke2-snapshot-controller-ss9v4
				8m12s
kube-system	0/1	Completed	0	pod/helm-install-rke2-snapshot-validation-webhook-vlkpn
				8m12s
kube-system	1/1	Running	0	pod/kube-apiserver-host1.local
				7m29s
kube-system	1/1	Running	0	pod/kube-controller-manager-host1.local
				7m30s
kube-system	1/1	Running	0	pod/kube-proxy-host1.local
				7m30s
kube-system	1/1	Running	0	pod/kube-scheduler-host1.local
				7m42s
kube-system	1/1	Running	0	pod/rke2-coredns-rke2-coredns-6c8d9bb6d-qlwc8
				7m31s
kube-system	1/1	Running	0	pod/rke2-coredns-rke2-coredns-autoscaler-55fb4bbbcf-j5r2z
				7m31s
kube-system	1/1	Running	0	pod/rke2-ingress-nginx-controller-4h2mm
				7m3s
kube-system	1/1	Running	0	pod/rke2-metrics-server-544c8c66fc-lsrc6
				7m15s
kube-system	1/1	Running	0	pod/rke2-snapshot-controller-59cc9cd8f4-4wx75
				7m14s
kube-system	1/1	Running	0	pod/rke2-snapshot-validation-webhook-54c5989b65-5kp2x
				7m15s
metallb-system	1/1	Running	0	pod/metallb-controller-5895d8446d-z54lm
				7m15s
metallb-system	1/1	Running	0	pod/metallb-speaker-fxwgk
				7m15s

NAMESPACE	NAME	EXTERNAL-IP	PORT(S)	TYPE
				AGE
cattle-fleet-system	service/gitjob			
ClusterIP	10.43.30.8	<none>	80/TCP	5m51s
cattle-provisioning-capi-system	service/capi-webhook-service			
ClusterIP	10.43.7.100	<none>	443/TCP	4m52s
cattle-system	service/rancher			
ClusterIP	10.43.100.229	<none>	80/TCP,443/TCP	6m34s
cattle-system	service/rancher-webhook			
ClusterIP	10.43.121.133	<none>	443/TCP	5m19s
cert-manager	service/cert-manager			
ClusterIP	10.43.140.65	<none>	9402/TCP	7m14s
cert-manager	service/cert-manager-webhook			
ClusterIP	10.43.108.158	<none>	443/TCP	7m14s
default	service/kubernetes			
ClusterIP	10.43.0.1	<none>	443/TCP	8m26s
default	service/kubernetes-vip			
LoadBalancer	10.43.138.138	192.168.100.151	9345:31006/TCP,6443:31599/TCP	8m21s
kube-system	service/cilium-agent			
ClusterIP	None	<none>	9964/TCP	7m32s
kube-system	service/rke2-coredns-rke2-coredns			
ClusterIP	10.43.0.10	<none>	53/UDP,53/TCP	7m31s
kube-system	service/rke2-ingress-nginx-controller-admission			
ClusterIP	10.43.157.19	<none>	443/TCP	7m3s
kube-system	service/rke2-metrics-server			
ClusterIP	10.43.4.123	<none>	443/TCP	7m15s
kube-system	service/rke2-snapshot-validation-webhook			
ClusterIP	10.43.91.161	<none>	443/TCP	7m16s
metallb-system	service/metallb-webhook-service			
ClusterIP	10.43.71.192	<none>	443/TCP	7m15s
NAMESPACE	NAME	DESIRED	CURRENT	READY
UP-TO-DATE	AVAILABLE	NODE SELECTOR	AGE	

kube-system	daemonset.apps/cilium			1	1	1
1	1	kubernetes.io/os=linux	7m32s			
kube-system	daemonset.apps/rke2-ingress-nginx-controller			1	1	1
1	1	kubernetes.io/os=linux	7m3s			
metallb-system	daemonset.apps/metallb-speaker			1	1	1
1	1	kubernetes.io/os=linux	7m15s			
NAMESPACE		NAME				
READY	UP-TO-DATE	AVAILABLE	AGE			
cattle-fleet-local-system		deployment.apps/fleet-agent				
1/1	1	1	34s			
cattle-fleet-system		deployment.apps/fleet-controller				
1/1	1	1	5m51s			
cattle-fleet-system		deployment.apps/gitjob				
1/1	1	1	5m51s			
cattle-provisioning-capi-system		deployment.apps/capi-controller-manager				
1/1	1	1	4m52s			
cattle-system		deployment.apps/rancher				
1/1	1	1	6m34s			
cattle-system		deployment.apps/rancher-webhook				
1/1	1	1	5m19s			
cert-manager		deployment.apps/cert-manager				
1/1	1	1	7m14s			
cert-manager		deployment.apps/cert-manager-cainjector				
1/1	1	1	7m14s			
cert-manager		deployment.apps/cert-manager-webhook				
1/1	1	1	7m14s			
endpoint-copier-operator		deployment.apps/endpoint-copier-operator				
2/2	2	2	7m16s			
kube-system		deployment.apps/cilium-operator				
1/2	2	1	7m32s			
kube-system		deployment.apps/rke2-coredns-rke2-coredns				
1/1	1	1	7m31s			
kube-system		deployment.apps/rke2-coredns-rke2-coredns-autoscaler				
1/1	1	1	7m31s			
kube-system		deployment.apps/rke2-metrics-server				
1/1	1	1	7m15s			
kube-system		deployment.apps/rke2-snapshot-controller				
1/1	1	1	7m14s			
kube-system		deployment.apps/rke2-snapshot-validation-webhook				
1/1	1	1	7m15s			
metallb-system		deployment.apps/metallb-controller				
1/1	1	1	7m15s			
NAMESPACE		NAME				
	DESIRED	CURRENT	READY	AGE		

cattle-fleet-local-system			replicaset.apps/fleet-agent-68f4d5d5f7
1	1	1	34s
cattle-fleet-system			replicaset.apps/fleet-controller-85564cc978
1	1	1	5m51s
cattle-fleet-system			replicaset.apps/gitjob-9dc58fb5b
1	1	1	5m51s
cattle-provisioning-capi-system			replicaset.apps/capi-controller-manager-5c57b4b8f7
1	1	1	4m52s
cattle-system			replicaset.apps/rancher-58575f9575
1	1	1	6m34s
cattle-system			replicaset.apps/rancher-webhook-5c6556f7ff
1	1	1	5m19s
cert-manager			replicaset.apps/cert-manager-6c69f9f796
1	1	1	7m14s
cert-manager			replicaset.apps/cert-manager-cainjector-584f44558c
1	1	1	7m14s
cert-manager			replicaset.apps/cert-manager-webhook-76f9945d6f
1	1	1	7m14s
endpoint-copier-operator			replicaset.apps/endpoint-copier-operator-58964b659b
2	2	2	7m16s
kube-system			replicaset.apps/cilium-operator-558bbf6cfd
2	2	1	7m32s
kube-system			replicaset.apps/rke2-coredns-rke2-coredns-6c8d9bb6d
1	1	1	7m31s
kube-system			replicaset.apps/rke2-coredns-rke2-coredns-
autoscaler-55fb4bbbcf	1	1	1 7m31s
kube-system			replicaset.apps/rke2-metrics-server-544c8c66fc
1	1	1	7m15s
kube-system			replicaset.apps/rke2-snapshot-controller-59cc9cd8f4
1	1	1	7m14s
kube-system			replicaset.apps/rke2-snapshot-validation-
webhook-54c5989b65	1	1	1 7m15s
metallb-system			replicaset.apps/metallb-controller-5895d8446d
1	1	1	7m15s

NAMESPACE	NAME	COMPLETIONS
kube-system	job.batch/helm-install-cert-manager	1/1 85s
kube-system	job.batch/helm-install-endpoint-copier-operator	1/1 59s
kube-system	job.batch/helm-install-metallb	1/1 60s
kube-system	job.batch/helm-install-rancher	1/1 100s 8m21s
kube-system	job.batch/helm-install-rke2-cilium	1/1 44s 8m18s

kube-system	job.batch/helm-install-rke2-coredns	1/1	45s
8m18s			
kube-system	job.batch/helm-install-rke2-ingress-nginx	1/1	76s
8m16s			
kube-system	job.batch/helm-install-rke2-metrics-server	1/1	60s
8m16s			
kube-system	job.batch/helm-install-rke2-snapshot-controller	1/1	61s
8m15s			
kube-system	job.batch/helm-install-rke2-snapshot-controller-crd	1/1	60s
8m16s			
kube-system	job.batch/helm-install-rke2-snapshot-validation-webhook	1/1	60s
8m14s			



Learn more about the improvements and new capabilities in this version

You can change what you see when you login via preferences

Clusters 1

State	Name	Provider	Kubern
Active	local	Local RKE2	v1.28.7



About

21.7 NeuVector Installation

Unlike the Rancher installation, the NeuVector installation does not require any special handling in EIB. EIB will automatically air-gap every image required by NeuVector.

We will create the definition file:

```
apiVersion: 1.0
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword: $6$jHugJNNd3HElGsUZ
$eodjVe4te5ps44SVcWshdfWizrP.xAyd71CVEXazBJ/.v799/WRCBXxfYmunlB02yp1hm/zb4r8EmnrrNCF.P/
kubernetes:
  version: v1.28.13+rke2r1
  helm:
    charts:
      - name: neuvector-crd
        version: 103.0.3+up2.7.6
        repositoryName: rancher-charts
        targetNamespace: neuvector
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: neuvector-values.yaml
      - name: neuvector
        version: 103.0.3+up2.7.6
        repositoryName: rancher-charts
        targetNamespace: neuvector
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: neuvector-values.yaml
    repositories:
      - name: rancher-charts
        url: https://charts.rancher.io/
```

We will also create a Helm values file for 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
```

Let's build the image:

```
podman run --rm -it --privileged -v $CONFIG_DIR:/eib \
registry.suse.com/edge/edge-image-builder:1.0.2 \
build --definition-file eib-iso-definition.yaml
```

The output should be similar to the following:

```
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]
Populating Embedded Artifact Registry... 100% (6/6, 20 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]
Building ISO image...
Kernel Params ..... [SKIPPED]
Image build complete!
```

Once a node using the built image is provisioned, we can verify the NeuVector installation:

```
/var/lib/rancher/rke2/bin/kubectl get all -n neuvector --kubeconfig /etc/rancher/rke2/
rke2.yaml
```

The output should be similar to the following, showing that everything has been successfully deployed:

NAME	READY	STATUS	RESTARTS	AGE
pod/neuvector-controller-pod-bc74745cf-x9fsc	1/1	Running	0	13m
pod/neuvector-enforcer-pod-vzw7t	1/1	Running	0	13m

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP
service/neuvector-svc-admission-webhook	ClusterIP	10.43.240.25	<none>
service/neuvector-svc-controller	ClusterIP	None	<none>

NAME	DESIRED	CURRENT	READY	UP-TO-DATE
daemonset.apps/neuvector-enforcer-pod	1	1	1	1

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/neuvector-controller-pod	1/1	1	1	13m

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/neuvector-controller-pod-bc74745cf	1	1	1	13m

NAME	SCHEDULE	SUSPEND	ACTIVE	LAST SCHEDULE	AGE
cronjob.batch/neuvector-updater-pod	0 0 * * *	False	0	<none>	13m

21.8 Longhorn Installation

The [official documentation \(https://longhorn.io/docs/1.6.1/deploy/install/airgap/\)](https://longhorn.io/docs/1.6.1/deploy/install/airgap/) for Longhorn contains a `longhorn-images.txt` file which lists all the images required for an air-gapped installation. We will be including them in our definition file. Let's create it:

```
apiVersion: 1.0
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
```



```

    encryptedPassword: $6$jHugJNNd3HElGsUZ
$eodjVe4te5ps44SVcWshdfWizrP.xAyd71CVEXazBJ/.v799/WRCBXxfYmunlB02yp1hm/zb4r8EmnrNCF.P/
kubernetes:
  version: v1.28.13+rke2r1
  helm:
    charts:
      - name: longhorn
        repositoryName: longhorn
        targetNamespace: longhorn-system
        createNamespace: true
        version: 1.6.1
    repositories:
      - name: longhorn
        url: https://charts.longhorn.io
embeddedArtifactRegistry:
  images:
    - name: longhornio/csi-attacher:v4.4.2
    - name: longhornio/csi-provisioner:v3.6.2
    - name: longhornio/csi-resizer:v1.9.2
    - name: longhornio/csi-snapshotter:v6.3.2
    - name: longhornio/csi-node-driver-registrar:v2.9.2
    - name: longhornio/livenessprobe:v2.12.0
    - name: longhornio/backing-image-manager:v1.6.1
    - name: longhornio/longhorn-engine:v1.6.1
    - name: longhornio/longhorn-instance-manager:v1.6.1
    - name: longhornio/longhorn-manager:v1.6.1
    - name: longhornio/longhorn-share-manager:v1.6.1
    - name: longhornio/longhorn-ui:v1.6.1
    - name: longhornio/support-bundle-kit:v0.0.36

```

Let's build the image:

```

podman run --rm -it --privileged -v $CONFIG_DIR:/eib \
registry.suse.com/edge/edge-image-builder:1.0.2 \
build --definition-file eib-iso-definition.yaml

```

The output should be similar to the following:

```

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]
Populating Embedded Artifact Registry... 100% (13/13, 20 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% (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]
Building ISO image...
Kernel Params ..... [SKIPPED]
Image build complete!

```

Once a node using the built image is provisioned, we can verify the Longhorn installation:

```

/var/lib/rancher/rke2/bin/kubectl get all -n longhorn-system --kubeconfig /etc/rancher/
rke2/rke2.yaml

```

The output should be similar to the following, showing that everything has been successfully deployed:

NAME	READY	STATUS	RESTARTS
AGE			
pod/csi-attacher-5c4bfdcf59-9hgvv	1/1	Running	0
35s			
pod/csi-attacher-5c4bfdcf59-dt6jl	1/1	Running	0
35s			
pod/csi-attacher-5c4bfdcf59-swpwq	1/1	Running	0
35s			
pod/csi-provisioner-667796df57-dfrzw	1/1	Running	0
35s			
pod/csi-provisioner-667796df57-tvsrt	1/1	Running	0
35s			
pod/csi-provisioner-667796df57-xszsj	1/1	Running	0
35s			
pod/csi-resizer-694f8f5f64-6khlb	1/1	Running	0
35s			
pod/csi-resizer-694f8f5f64-gnr45	1/1	Running	0
35s			
pod/csi-resizer-694f8f5f64-sbl4k	1/1	Running	0
35s			
pod/csi-snapshotter-959b69d4b-2k4v8	1/1	Running	0
35s			

pod/csi-snapshotter-959b69d4b-9d8wl 35s	1/1	Running	0
pod/csi-snapshotter-959b69d4b-l2w95 35s	1/1	Running	0
pod/engine-image-ei-5cefaf2b-cwd8f 43s	1/1	Running	0
pod/instance-manager-f0d17f96bc92f3cc44787a2a347f6a98 43s	1/1	Running	0
pod/longhorn-csi-plugin-szv7t 35s	3/3	Running	0
pod/longhorn-driver-deployer-9f4fc86-q8fz2 83s	1/1	Running	0
pod/longhorn-manager-zp66l 83s	1/1	Running	0
pod/longhorn-ui-5f4b7bbf69-k645d 83s	1/1	Running	3 (65s ago)
pod/longhorn-ui-5f4b7bbf69-t7xt4 83s	1/1	Running	3 (62s ago)

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
AGE				
service/longhorn-admission-webhook 83s	ClusterIP	10.43.74.59	<none>	9502/TCP
service/longhorn-backend 83s	ClusterIP	10.43.45.206	<none>	9500/TCP
service/longhorn-conversion-webhook 83s	ClusterIP	10.43.83.108	<none>	9501/TCP
service/longhorn-engine-manager 83s	ClusterIP	None	<none>	<none>
service/longhorn-frontend 83s	ClusterIP	10.43.84.55	<none>	80/TCP
service/longhorn-recovery-backend 83s	ClusterIP	10.43.75.200	<none>	9503/TCP
service/longhorn-replica-manager 83s	ClusterIP	None	<none>	<none>

NAME	DESIRED	CURRENT	READY	UP-TO-DATE
AVAILABLE				
NODE SELECTOR				
AGE				
daemonset.apps/engine-image-ei-5cefaf2b <none> 43s	1	1	1	1
daemonset.apps/longhorn-csi-plugin <none> 35s	1	1	1	1
daemonset.apps/longhorn-manager <none> 83s	1	1	1	1

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/csi-attacher	3/3	3	3	35s

deployment.apps/csi-provisioner	3/3	3	3	35s
deployment.apps/csi-resizer	3/3	3	3	35s
deployment.apps/csi-snapshotter	3/3	3	3	35s
deployment.apps/longhorn-driver-deployer	1/1	1	1	83s
deployment.apps/longhorn-ui	2/2	2	2	83s
NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/csi-attacher-5c4bfdcf59	3	3	3	35s
replicaset.apps/csi-provisioner-667796df57	3	3	3	35s
replicaset.apps/csi-resizer-694f8f5f64	3	3	3	35s
replicaset.apps/csi-snapshotter-959b69d4b	3	3	3	35s
replicaset.apps/longhorn-driver-deployer-9f4fc86	1	1	1	83s
replicaset.apps/longhorn-ui-5f4b7bbf69	2	2	2	83s

21.9 KubeVirt and CDI Installation

The Helm charts for both KubeVirt and CDI are only installing their respective operators. It is up to the operators to deploy the rest of the systems which means we will have to include all necessary container images in our definition file. Let's create it:

```

apiVersion: 1.0
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword: $6$jHugJNNd3HElGsUZ
$eodjVe4te5ps44SVcWshdfWizrP.xAyd71CVEXazBJ/.v799/WRCBXxfYmunlB02yp1hm/zb4r8EmnrrNCF.P/
kubernetes:
  version: v1.28.13+rke2r1
helm:
  charts:
    - name: kubevirt-chart
      repositoryName: suse-edge
      version: 0.3.0
      targetNamespace: kubevirt-system
      createNamespace: true
      installationNamespace: kube-system
    - name: cdi-chart
      repositoryName: suse-edge
      version: 0.3.0

```

```

    targetNamespace: cdi-system
    createNamespace: true
    installationNamespace: kube-system
  repositories:
    - name: suse-edge
      url: oci://registry.suse.com/edge
  embeddedArtifactRegistry:
    images:
      - name: registry.suse.com/suse/sles/15.5/cdi-uploadproxy:1.59.0-150500.6.18.1
      - name: registry.suse.com/suse/sles/15.5/cdi-uploadserver:1.59.0-150500.6.18.1
      - name: registry.suse.com/suse/sles/15.5/cdi-apiserver:1.59.0-150500.6.18.1
      - name: registry.suse.com/suse/sles/15.5/cdi-controller:1.59.0-150500.6.18.1
      - name: registry.suse.com/suse/sles/15.5/cdi-importer:1.59.0-150500.6.18.1
      - name: registry.suse.com/suse/sles/15.5/cdi-cloner:1.59.0-150500.6.18.1
      - name: registry.suse.com/suse/sles/15.5/virt-api:1.2.2-150500.8.21.1
      - name: registry.suse.com/suse/sles/15.5/virt-controller:1.2.2-150500.8.21.1
      - name: registry.suse.com/suse/sles/15.5/virt-launcher:1.2.2-150500.8.21.1
      - name: registry.suse.com/suse/sles/15.5/virt-handler:1.2.2-150500.8.21.1
      - name: registry.suse.com/suse/sles/15.5/virt-exportproxy:1.2.2-150500.8.21.1
      - name: registry.suse.com/suse/sles/15.5/virt-exportserver:1.2.2-150500.8.21.1

```

Let's build the image:

```

podman run --rm -it --privileged -v $CONFIG_DIR:/eib \
registry.suse.com/edge/edge-image-builder:1.0.2 \
build --definition-file eib-iso-definition.yaml

```

The output should be similar to the following:

```

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]
Populating Embedded Artifact Registry... 100% (13/13, 6 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]
Building ISO image...
Kernel Params ..... [SKIPPED]
Image build complete!
```

Once a node using the built image is provisioned, we can verify the installation of both KubeVirt and CDI.

Verify KubeVirt:

```
/var/lib/rancher/rke2/bin/kubectl get all -n kubevirt-system --kubeconfig /etc/rancher/rke2/rke2.yaml
```

The output should be similar to the following, showing that everything has been successfully deployed:

```
NAME                                READY   STATUS    RESTARTS   AGE
pod/virt-api-75dd5896c-ck24g        1/1     Running   0           2m11s
pod/virt-controller-54b46dffbc-8j8x9 1/1     Running   0           106s
pod/virt-controller-54b46dffbc-qhpkc 1/1     Running   0           106s
pod/virt-handler-qbbcq              1/1     Running   0           106s
pod/virt-operator-b599bcd7b-mq87d    1/1     Running   0           2m38s
pod/virt-operator-b599bcd7b-q7hkg    1/1     Running   0           2m38s

NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)
AGE
service/kubevirt-operator-webhook    ClusterIP           10.43.60.25     <none>           443/TCP
2m14s
service/kubevirt-prometheus-metrics ClusterIP           None            <none>           443/TCP
2m14s
service/virt-api                     ClusterIP           10.43.70.57     <none>           443/TCP
2m14s
service/virt-exportproxy             ClusterIP           10.43.255.129   <none>           443/TCP
2m14s

NAME                                DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR                            AGE
daemonset.apps/virt-handler         1         1         1       1             1
kubernetes.io/os=linux             106s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/virt-api             1/1     1             1           2m11s
deployment.apps/virt-controller     2/2     2             2           106s
deployment.apps/virt-operator       2/2     2             2           2m38s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/virt-api-75dd5896c  1         1         1       2m11s
replicaset.apps/virt-controller-54b46dffbc 2         2         2       106s
```

replicaset.apps/virt-operator-b599bcd7b	2	2	2	2m38s
NAME	AGE	PHASE		
kubevirt.kubevirt.io/kubevirt	2m38s	Deployed		

Verify CDI:

```
/var/lib/rancher/rke2/bin/kubectl get all -n cdi-system --kubeconfig /etc/rancher/rke2/rke2.yaml
```

The output should be similar to the following, showing that everything has been successfully deployed:

NAME	READY	STATUS	RESTARTS	AGE
pod/cdi-apiserver-85dff89756-7j97k	1/1	Running	0	2m56s
pod/cdi-deployment-66b96bf79f-6whvj	1/1	Running	0	2m56s
pod/cdi-operator-8f5f4654d-786rc	1/1	Running	0	3m
pod/cdi-uploadproxy-77db4ccd8-mzjz5	1/1	Running	0	2m56s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/cdi-api	ClusterIP	10.43.66.178	<none>	443/TCP	2m56s
service/cdi-prometheus-metrics	ClusterIP	10.43.99.119	<none>	8080/TCP	2m56s
service/cdi-uploadproxy	ClusterIP	10.43.207.154	<none>	443/TCP	2m56s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/cdi-apiserver	1/1	1	1	2m56s
deployment.apps/cdi-deployment	1/1	1	1	2m56s
deployment.apps/cdi-operator	1/1	1	1	3m
deployment.apps/cdi-uploadproxy	1/1	1	1	2m56s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/cdi-apiserver-85dff89756	1	1	1	2m56s
replicaset.apps/cdi-deployment-66b96bf79f	1	1	1	2m56s
replicaset.apps/cdi-operator-8f5f4654d	1	1	1	3m
replicaset.apps/cdi-uploadproxy-77db4ccd8	1	1	1	2m56s

21.10 Troubleshooting

If you run into any issues while building the images or are looking to further test and debug the process, please refer to the [upstream documentation \(https://github.com/suse-edge/edge-image-builder/tree/release-1.0/docs\)](https://github.com/suse-edge/edge-image-builder/tree/release-1.0/docs).

IV Third-Party Integration

22 NATS **198**

23 NVIDIA GPUs on SLE Micro **203**

How to integrate third-party tools

22 NATS

NATS (<https://nats.io/>)⁷ is a connective technology built for the ever-increasingly hyper-connected world. It is a single technology that enables applications to securely communicate across any combination of cloud vendors, on-premises, edge, Web and mobile devices. NATS consists of a family of open-source products that are tightly integrated but can be deployed easily and independently. NATS is being used globally by thousands of companies, spanning use cases including microservices, edge computing, mobile and IoT, and can be used to augment or replace traditional messaging.

22.1 Architecture

NATS is an infrastructure that allows data exchange between applications in the form of messages.

22.1.1 NATS client applications

NATS client libraries can be used to allow the applications to publish, subscribe, request and reply between different instances. These applications are generally referred to as client applications.

22.1.2 NATS service infrastructure

The NATS services are provided by one or more NATS server processes that are configured to interconnect with each other and provide a NATS service infrastructure. The NATS service infrastructure can scale from a single NATS server process running on an end device to a public global super-cluster of many clusters spanning all major cloud providers and all regions of the world.

22.1.3 Simple messaging design

NATS makes it easy for applications to communicate by sending and receiving messages. These messages are addressed and identified by subject strings and do not depend on network location. Data is encoded and framed as a message and sent by a publisher. The message is received, decoded and processed by one or more subscribers.

22.1.4 NATS JetStream

NATS has a built-in distributed persistence system called JetStream. JetStream was created to solve the problems identified with streaming in technology today — complexity, fragility and a lack of scalability. JetStream also solves the problem with the coupling of the publisher and the subscriber (the subscribers need to be up and running to receive the message when it is published). More information about NATS JetStream can be found [here \(https://docs.nats.io/nats-concepts/jetstream\)](https://docs.nats.io/nats-concepts/jetstream).

22.2 Installation

22.2.1 Installing NATS on top of K3s

NATS is built for multiple architectures so it can easily be installed on K3s. ([Chapter 13, K3s](#))

Let us create a values file to overwrite the default values of 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
```

Now let us install NATS via 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
```

With the `values.yaml` file above, the following components will be in the `nats` namespace:

1. HA version of NATS Statefulset containing three containers: NATS server + Config re-loader and Metrics sidecars.
2. NATS box container, which comes with a set of `NATS` utilities that can be used to verify the setup.
3. JetStream also leverages its Key-Value back-end that comes with `PVCs` bounded to the pods.

22.2.1.1 Testing the setup

```
kubectl exec -n nats -it deployment/nats-box -- /bin/sh -l
```

1. Create a subscription for the test subject:

```
nats sub test &
```

2. Send a message to the test subject:

```
nats pub test hi
```

22.2.1.2 Cleaning up

```
helm -n nats uninstall nats
rm values.yaml
```

22.2.2 NATS as a back-end for K3s

One component K3s leverages is [KINE \(https://github.com/k3s-io/kine\)](https://github.com/k3s-io/kine), which is a shim enabling the replacement of etcd with alternate storage back-ends originally targeting relational databases. As JetStream provides a Key Value API, this makes it possible to have NATS as a back-end for the K3s cluster.

There is an already merged PR which makes the built-in NATS in K3s straightforward, but the change is still [not included \(https://github.com/k3s-io/k3s/issues/7410#issue-1692989394\)](https://github.com/k3s-io/k3s/issues/7410#issue-1692989394) in the K3s releases.

For this reason, the K3s binary should be built manually.

In this tutorial, [SLE Micro on OSX on Apple Silicon \(UTM\) \(https://suse-edge.github.io/docs/quick-start/slemicro-utm-aarch64\)](https://suse-edge.github.io/docs/quick-start/slemicro-utm-aarch64) VM is used.



Note

Run the commands below on the OSX PC.

22.2.2.1 Building K3s

```
git clone --depth 1 https://github.com/k3s-io/k3s.git && cd k3s
```

The following command adds `nats` in the build tags to enable the NATS built-in feature in K3s:

```
sed -i '' 's/TAGS="ctrd/TAGS="nats ctrd/g' scripts/build  
make local
```

Replace `<node-ip>` with the actual IP of the node where the K3s will be started:

```
export NODE_IP=<node-ip>  
sudo scp dist/artifacts/k3s-arm64 ${NODE_IP}:/usr/local/bin/k3s
```



Note

Locally building K3s requires the `buildx` Docker CLI plugin. It can be [manually installed \(https://github.com/docker/buildx#manual-download\)](https://github.com/docker/buildx#manual-download) if `$ make local` fails.

22.2.2.2 Installing NATS CLI

```
TMPDIR=$(mktemp -d)
```

```
nats_version="nats-0.0.35-linux-arm64"
curl -o "${TMPDIR}/nats.zip" -sL https://github.com/nats-io/natscli/releases/download/
v0.0.35/${nats_version}.zip
unzip "${TMPDIR}/nats.zip" -d "${TMPDIR}"

sudo scp ${TMPDIR}/${nats_version}/nats ${NODE_IP}:/usr/local/bin/nats
rm -rf ${TMPDIR}
```

22.2.2.3 Running NATS as K3s back-end

Let us ssh on the node and run the K3s with the --datastore-endpoint flag pointing to nats.



Note

The command below starts K3s as a foreground process, so the logs can be easily followed to see if there are any issues. To not block the current terminal, a & flag could be added before the command to start it as a background process.

```
k3s server --datastore-endpoint=nats://
```



Note

For making the K3s server with the NATS back-end permanent on your slemicro VM, the script below can be run, which creates a systemd service with the needed configurations.

```
export INSTALL_K3S_SKIP_START=false
export INSTALL_K3S_SKIP_DOWNLOAD=true

curl -sL https://get.k3s.io | INSTALL_K3S_EXEC="server \
--datastore-endpoint=nats://" sh -
```

22.2.2.4 Troubleshooting

The following commands can be run on the node to verify that everything with the stream works properly:

```
nats str report -a
nats str view -a
```

23 NVIDIA GPUs on SLE Micro

23.1 Intro

This guide demonstrates how to implement host-level NVIDIA GPU support via the pre-built [open-source drivers \(https://github.com/NVIDIA/open-gpu-kernel-modules\)](https://github.com/NVIDIA/open-gpu-kernel-modules) on SLE Micro 5.5. These are drivers that are baked into the operating system rather than dynamically loaded by NVIDIA's [GPU Operator \(https://github.com/NVIDIA/gpu-operator\)](https://github.com/NVIDIA/gpu-operator). This configuration is highly desirable for customers that want to pre-bake all artifacts required for deployment into the image, and where the dynamic selection of the driver version, that is, the user selecting the version of the driver via Kubernetes, is not a requirement. This guide initially explains how to deploy the additional components onto a system that has already been pre-deployed, but follows with a section that describes how to embed this configuration into the initial deployment via Edge Image Builder. If you do not want to run through the basics and set things up manually, skip right ahead to that section.

It is important to call out that the support for these drivers is provided by both SUSE and NVIDIA in tight collaboration, where the driver is built and shipped by SUSE as part of the package repositories. However, if you have any concerns or questions about the combination in which you use the drivers, ask your SUSE or NVIDIA account managers for further assistance. If you plan to use [NVIDIA AI Enterprise \(https://www.nvidia.com/en-gb/data-center/products/ai-enterprise/\)](https://www.nvidia.com/en-gb/data-center/products/ai-enterprise/) (NVAIE), ensure that you are using an [NVAIE certified GPU \(https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/platform-support.html#supported-nvidia-gpus-and-systems\)](https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/platform-support.html#supported-nvidia-gpus-and-systems), which *may* require the use of proprietary NVIDIA drivers. If you are unsure, speak with your NVIDIA representative.

Further information about NVIDIA GPU operator integration is *not* covered in this guide. While integrating the NVIDIA GPU Operator for Kubernetes is not covered here, you can still follow most of the steps in this guide to set up the underlying operating system and simply enable the GPU operator to use the *pre-installed* drivers via the `driver.enabled=false` flag in the NVIDIA GPU Operator Helm chart, where it will simply pick up the installed drivers on the host. More comprehensive instructions are available from NVIDIA [here \(https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/install-gpu-operator.html#chart-customization-options\)](https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/install-gpu-operator.html#chart-customization-options). SUSE recently also made a [Technical Reference Doc-](#)

ument (https://documentation.suse.com/trd/kubernetes/single-html/gs_rke2-slebc_i_nvidia-gpu-operator/) (TRD) available that discusses how to use the GPU operator and the NVIDIA proprietary drivers, should this be a requirement for your use case.

23.2 Prerequisites

If you are following this guide, it assumes that you have the following already available:

- At least one host with SLE Micro 5.5 installed; this can be physical or virtual.
- Your hosts are attached to a subscription as this is required for package access — an evaluation is available [here \(https://www.suse.com/download/sle-micro/\)](https://www.suse.com/download/sle-micro/).
- A [compatible NVIDIA GPU \(https://github.com/NVIDIA/open-gpu-kernel-modules#compatible-gpus\)](https://github.com/NVIDIA/open-gpu-kernel-modules#compatible-gpus) installed (or *fully* passed through to the virtual machine in which SLE Micro is running).
- Access to the root user — these instructions assume you are the root user, and *not* escalating your privileges via `sudo`.

23.3 Manual installation

In this section, you are going to install the NVIDIA drivers directly onto the SLE Micro operating system as the NVIDIA open-driver is now part of the core SLE Micro package repositories, which makes it as easy as installing the required RPM packages. There is no compilation or downloading of executable packages required. Below we walk through deploying the "G06" generation of driver, which supports the latest GPUs (see [here \(https://en.opensuse.org/SDB:NVIDIA_drivers#Install\)](https://en.opensuse.org/SDB:NVIDIA_drivers#Install) for further information), so select an appropriate driver generation for the NVIDIA GPU that your system has. For modern GPUs, the "G06" driver is the most common choice.

Before we begin, it is important to recognize that besides the NVIDIA open-driver that SUSE ships as part of SLE Micro, you might also need additional NVIDIA components for your setup. These could include OpenGL libraries, CUDA toolkits, command-line utilities such as `nvidia-smi`, and container-integration components such as `nvidia-container-toolkit`. Many of these components are not shipped by SUSE as they are proprietary NVIDIA software, or it makes no sense for us to ship them instead of NVIDIA. Therefore, as part of the instructions, we are going to configure additional repositories that give us access to said components and walk through

certain examples of how to use these tools, resulting in a fully functional system. It is important to distinguish between SUSE repositories and NVIDIA repositories, as occasionally there can be a mismatch between the package versions that NVIDIA makes available versus what SUSE has built. This usually arises when SUSE makes a new version of the open-driver available, and it takes a couple of days before the equivalent packages are made available in NVIDIA repositories to match.

We recommend that you ensure that the driver version that you are selecting is compatible with your GPU and meets any CUDA requirements that you may have by checking:

- The [CUDA release notes](https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/) (<https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/>) ↗
- The driver version that you plan on deploying has a matching version in the [NVIDIA SLE15-SP5 repository](http://download.nvidia.com/suse/sle15sp5/x86_64/) (http://download.nvidia.com/suse/sle15sp5/x86_64/) ↗ and ensuring that you have equivalent package versions for the supporting components available



Tip

To find the NVIDIA open-driver versions, either run `zypper se -s nvidia-open-driver` on the target machine *or* search the SUSE Customer Center for the "nvidia-open-driver" in SLE Micro 5.5 for x86_64 (https://scc.suse.com/packages?name=SUSE%20Linux%20Enterprise%20Micro&version=5.5&arch=x86_64) ↗.

Here, you will see *four* versions available, with *545.29.06* being the newest:

When you have confirmed that an equivalent version is available in the NVIDIA repos, you are ready to install the packages on the host operating system. For this, we need to open up a `transactional-update` session, which creates a new read/write snapshot of the underlying operating system so we can make changes to the immutable platform (for further instructions on `transactional-update`, see [here \(https://documentation.suse.com/sle-micro/5.4/html/SLE-Micro-all/sec-transactional-update.html\)](https://documentation.suse.com/sle-micro/5.4/html/SLE-Micro-all/sec-transactional-update.html)):

```
transactional-update shell
```

When you are in your `transactional-update` shell, add an additional package repository from NVIDIA. This allows us to pull in additional utilities, for example, `nvidia-smi`:

```
zypper ar https://download.nvidia.com/suse/sle15sp5/ nvidia-sle15sp5-main
zypper --gpg-auto-import-keys refresh
```

You can then install the driver and `nvidia-compute-utils` for additional utilities. If you do not need the utilities, you can omit it, but for testing purposes, it is worth installing at this stage:

```
zypper install -y --auto-agree-with-licenses nvidia-open-driver-G06-signed-kmp nvidia-compute-utils-G06
```



Note

If the installation fails, this might indicate a dependency mismatch between the selected driver version and what NVIDIA ships in their repositories. Refer to the previous section to verify that your versions match. Attempt to install a different driver version. For example, if the NVIDIA repositories have an earlier version, you can try specifying `nvidia-open-driver-G06-signed-kmp=545.29.06` on your install command to specify a version that aligns.

Next, if you are *not* using a supported GPU (remembering that the list can be found [here \(https://github.com/NVIDIA/open-gpu-kernel-modules#compatible-gpus\)](https://github.com/NVIDIA/open-gpu-kernel-modules#compatible-gpus)), you can see if the driver works by enabling support at the module level, but your mileage may vary — skip this step if you are using a *supported* GPU:

```
sed -i '/NVreg_OpenRmEnableUnsupportedGpus/s/^#//g' /etc/modprobe.d/50-nvidia-default.conf
```

Now that you have installed these packages, it is time to exit the `transactional-update` session:

```
exit
```



Note

Make sure that you have exited the `transactional-update` session before proceeding.

Now that you have installed the drivers, it is time to reboot. As SLE Micro is an immutable operating system, it needs to reboot into the new snapshot that you created in a previous step. The drivers are only installed into this new snapshot, hence it is not possible to load the drivers without rebooting into this new snapshot, which happens automatically. Issue the reboot command when you are ready:

```
reboot
```

Once the system has rebooted successfully, log back in and use the `nvidia-smi` tool to verify that the driver is loaded successfully and that it can both access and enumerate your GPUs:

```
nvidia-smi
```

The output of this command should show you something similar to the following output, noting that in the example below, we have two GPUs:

```
Wed Feb 28 12:31:06 2024
+-----+
| 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                |
+-----+
```

This concludes the installation and verification process for the NVIDIA drivers on your SLE Micro system.

23.4 Further validation of the manual installation

At this stage, all we have been able to verify is that, at the host level, the NVIDIA device can be accessed and that the drivers are loading successfully. However, if we want to be sure that it is functioning, a simple test would be to validate that the GPU can take instructions from a user-space application, ideally via a container, and through the CUDA library, as that is typically what a real workload would use. For this, we can make a further modification to the host OS by installing the `nvidia-container-toolkit` ([NVIDIA Container Toolkit \(https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html#installing-with-zypper\)](https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html#installing-with-zypper)). First, open another `transactional-update` shell, noting that we could have done this in a single transaction in the previous step, and see how to do this fully automated in a later section:

```
transactional-update shell
```

Next, install the `nvidia-container-toolkit` package from the NVIDIA Container Toolkit repo:

- The `nvidia-container-toolkit.repo` below contains a stable (`nvidia-container-toolkit`) and an experimental (`nvidia-container-toolkit-experimental`) repository. The stable repository is recommended for production use. The experimental repository is disabled by default.

```
zypper ar https://nvidia.github.io/libnvidia-container/stable/rpm/nvidia-container-toolkit.repo
zypper --gpg-auto-import-keys install -y nvidia-container-toolkit
```

When you are ready, you can exit the `transactional-update` shell:

```
exit
```

...and reboot the machine into the new snapshot:

```
reboot
```



Note

As before, you need to ensure that you have exited the `transactional-shell` and rebooted the machine for your changes to be enacted.

With the machine rebooted, you can verify that the system can successfully enumerate the devices using the NVIDIA Container Toolkit. The output should be verbose, with INFO and WARN messages, but no ERROR messages:

```
nvidia-ctk cdi generate --output=/etc/cdi/nvidia.yaml
```

This ensures that any container started on the machine can employ NVIDIA GPU devices that have been discovered. When ready, you can then run a podman-based container. Doing this via `podman` gives us a good way of validating access to the NVIDIA device from within a container, which should give confidence for doing the same with Kubernetes at a later stage. Give `podman` access to the labeled NVIDIA devices that were taken care of by the previous command, based on [SLE BCI \(https://registry.suse.com/bci/bci-base-15sp5/index.html\)](https://registry.suse.com/bci/bci-base-15sp5/index.html), and simply run the Bash command:

```
podman run --rm --device nvidia.com/gpu=all --security-opt=label=disable -it registry.suse.com/bci/bci-base:latest bash
```

You will now execute commands from within a temporary podman container. It does not have access to your underlying system and is ephemeral, so whatever we do here will not persist, and you should not be able to break anything on the underlying host. As we are now in a container, we can install the required CUDA libraries, again checking the correct CUDA version for your driver [here \(https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/\)](https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/), although the previous output of `nvidia-smi` should show the required CUDA version. In the example below, we are installing *CUDA 12.3* and pulling many examples, demos and development kits so you can fully validate the GPU:

```
zypper ar http://developer.download.nvidia.com/compute/cuda/repos/sles15/x86_64/ cuda-sle15-sp5
zypper in -y cuda-libraries-devel-12-3 cuda-minimal-build-12-3 cuda-demo-suite-12-3
```

Once this has been installed successfully, do not exit the container. We will run the `deviceQuery` CUDA example, which comprehensively validates GPU access via CUDA, and from within the container itself:

```
/usr/local/cuda-12/extras/demo_suite/deviceQuery
```

If successful, you should see output that shows similar to the following, noting the `Result = PASS` message at the end of the command, and noting that in the output below, the system correctly identifies two GPUs, whereas your environment may only have one:

```
/usr/local/cuda-12/extras/demo_suite/deviceQuery Starting...

CUDA Device Query (Runtime API) version (CUDA 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
```

From here, you can continue to run any other CUDA workload — use compilers and any other aspect of the CUDA ecosystem to run further tests. When done, you can exit from the container, noting that whatever you have installed in there is ephemeral (so will be lost!), and has not impacted the underlying operating system:

```
exit
```

23.5 Implementation with Kubernetes

Now that we have proven the installation and use of the NVIDIA open-driver on SLE Micro, let us explore configuring Kubernetes on the same machine. This guide does not walk you through deploying Kubernetes, but it assumes that you have installed [K3s](https://k3s.io/) (<https://k3s.io/>) or [RKE2](https://docs.rke2.io/install/quickstart) (<https://docs.rke2.io/install/quickstart>) and that your kubeconfig is configured accordingly, so that standard `kubectl` commands can be executed as the superuser. We assume that your node forms a single-node cluster, although the core steps should be similar for multi-node clusters. First, ensure that your `kubectl` access is working:

```
kubectl get nodes
```

This should show something similar to the following:

NAME	STATUS	ROLES	AGE	VERSION
node0001	Ready	control-plane,etcd,master	13d	v1.28.13+rke2r1

What you should find is that your `k3s/rke2` installation has detected the NVIDIA Container Toolkit on the host and auto-configured the NVIDIA runtime integration into `containerd` (the Container Runtime Interface that `k3s/rke2` use). Confirm this by checking the `containerd config.toml` file:

```
tail -n8 /var/lib/rancher/rke2/agent/etc/containerd/config.toml
```

This must show something akin to the following. The equivalent K3s location is `/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"
```



Note

If these entries are not present, the detection might have failed. This could be due to the machine or the Kubernetes services not being restarted. Add these manually as above, if required.

Next, we need to configure the NVIDIA `RuntimeClass` as an additional Kubernetes runtime to the default, ensuring that any user requests for pods that need access to the GPU can use the NVIDIA Container Toolkit to do so, via the `nvidia-container-runtime`, as configured in the `containerd` configuration:

```
kubectl apply -f - <<EOF
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: nvidia
handler: nvidia
EOF
```

The next step is to configure the [NVIDIA Device Plugin \(https://github.com/NVIDIA/k8s-device-plugin\)](https://github.com/NVIDIA/k8s-device-plugin), which configures Kubernetes to leverage the NVIDIA GPUs as resources within the cluster that can be used, working in combination with the NVIDIA Container Toolkit. This tool initially detects all capabilities on the underlying host, including GPUs, drivers and other capabilities (such as GL) and then allows you to request GPU resources and consume them as part of your applications.

First, you need to add and update the Helm repository for the NVIDIA Device Plugin:

```
helm repo add nvdp https://nvidia.github.io/k8s-device-plugin
helm repo update
```

Now you can install the NVIDIA Device Plugin:

```
helm upgrade -i nvdp nvdp/nvidia-device-plugin --namespace nvidia-device-plugin --create-namespace --version 0.14.5 --set runtimeClassName=nvidia
```

After a few minutes, you see a new pod running that will complete the detection on your available nodes and tag them with the number of GPUs that have been detected:

```
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"
}

```

Now you are ready to create an NVIDIA pod that attempts to use this GPU. Let us try with the CUDA Benchmark container:

```

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

```

If all went well, you can look at the logs and see the benchmark information:

```

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
```

Finally, if your applications require OpenGL, you can install the required NVIDIA OpenGL libraries at the host level, and the NVIDIA Device Plugin and NVIDIA Container Toolkit can make them available to containers. To do this, install the package as follows:

```
transactional-update pkg install nvidia-gl-G06
```



Note

You need to reboot to make this package available to your applications. The NVIDIA Device Plugin should automatically redetect this via the NVIDIA Container Toolkit.

23.6 Bringing it together via Edge Image Builder

Okay, so you have demonstrated full functionality of your applications and GPUs on SLE Micro and you now want to use [Chapter 9, Edge Image Builder](#) to provide it all together via a deployable/consumable ISO or RAW disk image. This guide does not explain how to use Edge Image Builder, but it provides the necessary configurations to build such image. Below you can find an example of an image definition, along with the necessary Kubernetes configuration files, to ensure that all the required components are deployed out of the box. Here is the directory structure of the Edge Image Builder directory for the example shown below:

```
.
```

```

├─ base-images
│   └─ SLE-Micro.x86_64-5.5.0-Default-SelfInstall-GM2.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

```

Let us explore those files. First, here is a sample image definition for a single-node cluster running K3s that deploys the utilities and OpenGL packages, too ([eib-config-iso.yaml](#)):

```

apiVersion: 1.0
image:
  arch: x86_64
  imageType: iso
  baseImage: SLE-Micro.x86_64-5.5.0-Default-SelfInstall-GM2.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/JPeFk9jzQ06eapESYZesZH09Es1D1
  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/sle15sp5/
      - url: https://nvidia.github.io/libnvidia-container/stable/rpm/x86_64
    sccRegistrationCode: <snip>
kubernetes:

```

```
version: v1.28.13+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
```



Note

This is just an example. You may need to customize it to fit your requirements and expectations. Additionally, if using SLE Micro, you need to provide your own `sccRegistrationCode` to resolve package dependencies and pull the NVIDIA drivers.

Besides this, we need to add additional components, so they get loaded by Kubernetes at boot time. The EIB directory needs a `kubernetes` directory first, with subdirectories for the configuration, Helm chart values and any additional manifests required:

```
mkdir -p kubernetes/config kubernetes/helm/values kubernetes/manifests
```

Let us now set up the (optional) Kubernetes configuration by choosing a CNI (which defaults to Cilium if unselected) and enabling SELinux:

```
cat << EOF > kubernetes/config/server.yaml
cni: cilium
selinux: true
EOF
```

Now ensure that the NVIDIA RuntimeClass is created on the Kubernetes cluster:

```
cat << EOF > kubernetes/manifests/nvidia-runtime-class.yaml
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: nvidia
handler: nvidia
EOF
```

We use the built-in Helm Controller to deploy the NVIDIA Device Plugin through Kubernetes itself. Let's provide the runtime class in the values file for the chart:

```
cat << EOF > kubernetes/helm/values/nvidia-device-plugin.yaml
runtimeClassName: nvidia
EOF
```

We need to grab the NVIDIA Container Toolkit RPM public key before proceeding:

```
mkdir -p rpms/gpg-keys
curl -o rpms/gpg-keys/nvidia-container-toolkit.key https://nvidia.github.io/libnvidia-
container/gpgkey
```

All the required artifacts, including Kubernetes binary, container images, Helm charts (and any referenced images), will be automatically air-gapped, meaning that the systems at deploy time should require no Internet connectivity by default. Now you need only to grab the SLE Micro ISO from the [SUSE Downloads Page \(https://www.suse.com/download/sle-micro/\)](https://www.suse.com/download/sle-micro/) (and place it in the `base-images` directory), and you can call the Edge Image Builder tool to generate the ISO for you. To complete the example, here is the command that was used to build the image:

```
podman run --rm --privileged -it -v /path/to/eib-files:/eib \
registry.suse.com/edge/edge-image-builder:1.0.2 \
build --definition-file eib-config-iso.yaml
```

For further instructions, please see the [documentation \(https://github.com/suse-edge/edge-image-builder/blob/release-1.0/docs/building-images.md\)](https://github.com/suse-edge/edge-image-builder/blob/release-1.0/docs/building-images.md) for Edge Image Builder.

23.7 Resolving issues

23.7.1 nvidia-smi does not find the GPU

Check the kernel messages using `dmesg`. If this indicates that it cannot allocate `NvKMSKapDevice`, apply the unsupported GPU workaround:

```
sed -i '/NVreg_OpenRmEnableUnsupportedGpus/s/^#//g' /etc/modprobe.d/50-nvidia-
default.conf
```

NOTE: You will need to reload the kernel module, or reboot, if you change the kernel module configuration in the above step for it to take effect.

V Day 2 Operations

24 Management Cluster **220**

25 Downstream clusters **232**

This section explains how administrators can handle different "Day Two" operation tasks both on the management and on the downstream clusters.

24 Management Cluster

This section covers how to do various Day 2 operations on a management cluster.

24.1 RKE2 upgrade



Note

To ensure **disaster recovery**, we advise to do a backup of the RKE2 cluster data. For information on how to do this, check [here \(https://docs.rke2.io/backup_restore\)](https://docs.rke2.io/backup_restore). The default location for the rke2 binary is /opt/rke2/bin.

You can upgrade the RKE2 version using the RKE2 installation script as follows:

```
curl -sfl https://get.rke2.io | INSTALL_RKE2_VERSION=vX.Y.Z+rke2rN sh -
```

Remember to restart the rke2 process after installing:

```
# For server nodes:
systemctl restart rke2-server

# For agent nodes:
systemctl restart rke2-agent
```



Important

To avoid any unforeseen upgrade problems, use the following node upgrade order:

1. *Server nodes* - should be upgraded **one** node at a time.
2. *Agent nodes* - should be upgraded after **all** server node upgrades have finished. Can be upgraded in parallel.

For further information, see the RKE2 upgrade documentation (https://docs.rke2.io/upgrade/manual_upgrade#upgrade-rke2-using-the-installation-script).

24.2 OS upgrade



Note

This section assumes that you have registered your system to <https://scc.suse.com>.

SUSE regularly releases new SLE Micro package updates. To retrieve the updated package versions SLE Micro uses transactional-upgrade.

transactional-upgrade provides an application and library to update a Linux operating system in a transactional way, i.e. the update will be performed in the background while the system continues running as it is. Only after you **reboot** the system will the update take effect. For further information, see the transactional-update [GitHub](https://github.com/openSUSE/transactional-update) (<https://github.com/openSUSE/transactional-update>) [GitHub page](#).

To update all packages in the system, execute:

```
transactional-update
```

Since **rebooting** the node will result in it being unavailable for some time, if you are running a multi-node cluster, you can [cordon](https://kubernetes.io/docs/reference/kubectl/generated/kubectl_cordon/) (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_cordon/) and [drain](https://kubernetes.io/docs/reference/kubectl/generated/kubectl_drain/) (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_drain/) the node before the **reboot**.

To cordon a node, execute:

```
kubectl cordon <node>
```

This will result in the node being taken out of the default scheduling mechanism, ensuring that no pods will be assigned to it by mistake.

To drain a node, execute:

```
kubectl drain <node>
```

This will ensure that all workloads on the node will be transferred to other available nodes.




Note

Depending on what workloads you are running on the node, you might also need to provide additional flags (e.g. --delete-emptydir-data, --ignore-daemonsets) to the command.

Reboot node:

```
sudo reboot
```

After a successful reboot, the packages on your node will be updated. The only thing left is to bring the node back to the default scheduling mechanism with the `uncordon` (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_uncordon/)  command.

Uncordon node:

```
kubectl uncordon <node>
```



Note

In case you want to revert the update, use the above steps with the following `transactional-update` command:

```
transactional-update rollback last
```

24.3 Helm upgrade




Note

This section assumes you have installed `helm` on your system. For `helm` installation instructions, check [here](https://helm.sh/docs/intro/install) (<https://helm.sh/docs/intro/install>) .

This section covers how to upgrade both an EIB ([Section 24.3.1, “EIB deployed helm chart”](#)) and non-EIB ([Section 24.3.2, “Non-EIB deployed helm chart”](#)) deployed helm chart.

24.3.1 EIB deployed helm chart

EIB deploys helm charts defined in its image definition file ([Section 3.3, “Creating the image definition file”](#)) by using RKE2’s manifest `auto-deploy` (<https://docs.rke2.io/advanced#auto-deploying-manifests>)  functionality.

In order to upgrade a chart that is deployed in such a manner, you need to upgrade the chart manifest file that EIB will create under the `/var/lib/rancher/rke2/server/manifests` directory on your `initializer` node.



Note

To ensure disaster recovery, we advise that you always backup your chart manifest file as well as follow any documentation related to disaster recovery that your chart offers.

To upgrade the chart manifest file, follow these steps:

1. Locate the initializer node

- For multi-node clusters - in your EIB image definition file, you should have specified the initializer: true property for one of your nodes. If you have not specified this property, the initializer node will be the first **server** node in your node list.
- For single-node clusters - the initializer is the currently running node.

2. SSH to the initializer node:

```
ssh root@<node_ip>
```

3. Pull (https://helm.sh/docs/helm/helm_pull/) the helm chart:

- For helm charts hosted in a helm chart repository:

```
helm repo add <chart_repo_name> <chart_repo_urls>
helm pull <chart_repo_name>/<chart_name>

# Alternatively if you want to pull a specific verison
helm pull <chart_repo_name>/<chart_name> --version=X.Y.Z
```

- For OCI-based helm charts:

```
helm pull oci://<chart_oci_url>

# Alternatively if you want to pull a specific verison
helm pull oci://<chart_oci_url> --version=X.Y.Z
```

4. Encode the pulled .tgz archive so that it can be passed to a HelmChart CR config:

```
base64 -w 0 <chart_name>-X.Y.Z.tgz > <chart_name>-X.Y.Z.txt
```

5. Make a copy of the chart manifest file that we will edit:

```
cp /var/lib/rancher/rke2/server/manifests/<chart_name>.yaml ./<chart_name>.yaml
```

6. Change the `chartContent` and `version` configurations of the `bar.yaml` file:

```
sed -i -e "s|chartContent:.*|chartContent: $(<chart-name-X.Y.Z.txt)|" -e "s|version:.*|version: X.Y.Z|" <chart_name>.yaml
```




Note

If you need to do any additional upgrade changes to the chart (e.g. adding **new** custom chart values), you need to manually edit the chart manifest file.

7. Replace the original chart manifest file:

```
cp <chart_name>.yaml /var/lib/rancher/rke2/server/manifests/
```

The above commands will trigger an upgrade of the helm chart. The upgrade will be handled by the `helm-controller` (<https://github.com/k3s-io/helm-controller#helm-controller>) .

To track the helm chart upgrade you need to view the logs of the pod that the `helm-controller` creates for the chart upgrade. Refer to the Examples ([Section 24.3.1.1, "Examples"](#)) section for more information.

24.3.1.1 Examples



Note

The examples in this section assume that you have already located and connected to your initializer node.

This section offer examples on how to upgrade a:

- Rancher ([Section 24.3.1.1.1, "Rancher upgrade"](#)) helm chart
- Metal3 ([Section 24.3.1.1.2, "Metal³ upgrade"](#)) helm chart



Note

To ensure disaster recovery, we advise to do a Rancher backup. For information on how to do this, check [here \(https://ranchermanager.docs.rancher.com/how-to-guides/new-user-guides/backup-restore-and-disaster-recovery/back-up-rancher\)](https://ranchermanager.docs.rancher.com/how-to-guides/new-user-guides/backup-restore-and-disaster-recovery/back-up-rancher).

This example shows how to upgrade Rancher to the 2.8.8 version.

1. Add the Rancher Prime Helm repository:

```
helm repo add rancher-prime https://charts.rancher.com/server-charts/prime
```

2. Pull the latest Rancher Prime helm chart version:

```
helm pull rancher-prime/rancher --version=2.8.8
```

3. Encode .tgz archive so that it can be passed to a HelmChart CR config:

```
base64 -w 0 rancher-2.8.8.tgz > rancher-2.8.8-encoded.txt
```

4. Make a copy of the rancher.yaml file that we will edit:

```
cp /var/lib/rancher/rke2/server/manifests/rancher.yaml ./rancher.yaml
```

5. Change the chartContent and version configurations of the rancher.yaml file:

```
sed -i -e "s|chartContent:.*|chartContent: $(<rancher-2.8.8-encoded.txt)|" -e "s|version:.*|version: 2.8.8|" rancher.yaml
```



Note

If you need to do any additional upgrade changes to the chart (e.g. adding **new** custom chart values), you need to manually edit the rancher.yaml file.

6. Replace the original rancher.yaml file:

```
cp rancher.yaml /var/lib/rancher/rke2/server/manifests/
```

To verify the update:

1. List pods in default namespace:

```
kubectl get pods -n default

# Example output
NAME                                READY   STATUS    RESTARTS   AGE
helm-install-cert-manager-7v7nm     0/1     Completed 0           88m
helm-install-rancher-p99k5          0/1     Completed 0           3m21s
```

2. Look at the logs of the helm-install-rancher-* pod:

```
kubectl logs <helm_install_rancher_pod> -n default

# Example
kubectl logs helm-install-rancher-p99k5 -n default
```

3. Verify Rancher pods are running:

```
kubectl get pods -n cattle-system

# Example output
NAME                                READY   STATUS    RESTARTS   AGE
helm-operation-mccvd                0/2     Completed 0           3m52s
helm-operation-np8kn                 0/2     Completed 0           106s
helm-operation-q8lf7                 0/2     Completed 0           2m53s
rancher-648d4fbc6c-qxfpj             1/1     Running   0           5m27s
rancher-648d4fbc6c-trdnf             1/1     Running   0           9m57s
rancher-648d4fbc6c-wvhbf             1/1     Running   0           9m57s
rancher-webhook-649dcc48b4-zqjs7     1/1     Running   0           100s
```

4. Verify Rancher version upgrade:

```
kubectl get settings.management.cattle.io server-version

# Example output
NAME           VALUE
server-version v2.8.8
```

24.3.1.1.2 **Metal³ upgrade**

This example shows how to upgrade Metal³ to the 0.7.4 version.

1. Pull the latest Meta13 helm chart version:

```
helm pull oci://registry.suse.com/edge/metal3-chart --version 0.7.4
```

2. Encode .tgz archive so that it can be passed to a HelmChart CR config:

```
base64 -w 0 metal3-chart-0.7.4.tgz > metal3-chart-0.7.4-encoded.txt
```

3. Make a copy of the Meta13 manifest file that we will edit:

```
cp /var/lib/rancher/rke2/server/manifests/metal3.yaml ./metal3.yaml
```

4. Change the chartContent and version configurations of the Meta13 manifest file:

```
sed -i -e "s|chartContent:.*|chartContent: $(<metal3-chart-0.7.4-encoded.txt)|" -e "s|version:.*|version: 0.7.4|" metal3.yaml
```



Note

If you need to do any additional upgrade changes to the chart (e.g. adding **new** custom chart values), you need to manually edit the metal3.yaml file.

5. Replace the original Meta13 manifest file:

```
cp metal3.yaml /var/lib/rancher/rke2/server/manifests/
```

To verify the update:

1. List pods in default namespace:

```
kubectl get pods -n default
```

```
# Example output
```

NAME	READY	STATUS	RESTARTS	AGE
helm-install-metal3-7p7bl	0/1	Completed	0	27s

2. Look at the logs of the helm-install-rancher-* pod:

```
kubectl logs <helm_install_rancher_pod> -n default
```

```
# Example
```

```
kubectl logs helm-install-metal3-7p7bl -n default
```

3. Verify Metal3 pods are running:

```
kubectl get pods -n metal3-system
```

Example output

NAME	READY	STATUS	RESTARTS
baremetal-operator-controller-manager-785f99c884-9z87p ago) 36m	2/2	Running	2 (25m ago)
metal3-metal3-ironic-96fb66cdd-lkss2 3m54s	4/4	Running	0
metal3-metal3-mariadb-55fd44b648-q6zhk 36m	1/1	Running	0

4. Verify the HelmChart resource version is upgraded:

```
kubectl get helmchart metal3 -n default
```

Example output

NAME	JOB	CHART	TARGETNAMESPACE	VERSION	REPO
metal3	helm-install-metal3		metal3-system	0.7.4	

24.3.2 Non-EIB deployed helm chart

1. Get the values for the currently running helm chart .yaml file and make any changes to them **if necessary**:

```
helm get values <chart_name> -n <chart_namespace> -o yaml > <chart_name>-values.yaml
```

2. Update the helm chart:

```
# For charts using a chart repository
helm upgrade <chart_name> <chart_repo_name>/<chart_name> \
  --namespace <chart_namespace> \
  -f <chart_name>-values.yaml \
  --version=X.Y.Z

# For OCI based charts
helm upgrade <chart_name> oci://<oci_registry_url>/<chart_name> \
  --namespace <chart_namespace> \
  -f <chart_name>-values.yaml \
```

```
--version=X.Y.Z
```

3. Verify the chart upgrade. Depending on the chart you may need to verify different resources. For examples of chart upgrades, see the Examples ([Section 24.3.2.1, “Examples”](#)) section.

24.3.2.1 Examples

This section offer examples on how to upgrade a:

- Rancher ([Section 24.3.2.1.1, “Rancher”](#)) helm chart
- Metal3 ([Section 24.3.2.1.2, “Metal³”](#)) helm chart

24.3.2.1.1 Rancher



Note

To ensure disaster recovery, we advise to do a Rancher backup. For information on how to do this, check [here \(https://ranchermanager.docs.rancher.com/how-to-guides/new-user-guides/backup-restore-and-disaster-recovery/back-up-rancher\)](https://ranchermanager.docs.rancher.com/how-to-guides/new-user-guides/backup-restore-and-disaster-recovery/back-up-rancher).

This example shows how to upgrade Rancher to the 2.8.8 version.

1. Get the values for the current Rancher release and print them to a rancher-values.yaml file:

```
helm get values rancher -n cattle-system -o yaml > rancher-values.yaml
```

2. Update the helm chart:

```
helm upgrade rancher rancher-prime/rancher \  
  --namespace cattle-system \  
  -f rancher-values.yaml \  
  --version=2.8.8
```

3. Verify Rancher version upgrade:

```
kubectl get settings.management.cattle.io server-version
```



```
# Example output
NAME          VALUE
server-version v2.8.8
```

For additional information on the Rancher helm chart upgrade, check [here \(https://ranchermanager.docs.rancher.com/getting-started/installation-and-upgrade/install-upgrade-on-a-kubernetes-cluster/upgrades\)](https://ranchermanager.docs.rancher.com/getting-started/installation-and-upgrade/install-upgrade-on-a-kubernetes-cluster/upgrades).

24.3.2.1.2 Metal³

This example shows how to upgrade Metal³ to the 0.7.4 version.

1. Get the values for the current Rancher release and print them to a rancher-values.yaml file:

```
helm get values metal3 -n metal3-system -o yaml > metal3-values.yaml
```

2. Update the helm chart:

```
helm upgrade metal3 oci://registry.suse.com/edge/metal3-chart \
  --namespace metal3-system \
  -f metal3-values.yaml \
  --version=0.7.4
```

3. Verify Metal3 pods are running:

```
kubectl get pods -n metal3-system

# Example output
NAME                                                                 READY   STATUS    RESTARTS
  AGE
baremetal-operator-controller-manager-785f99c884-fvsx4             2/2     Running   0
  12m
metal3-metal3-ironic-96fb66cdd-j9mgf                             4/4     Running   0
  2m41s
metal3-metal3-mariadb-55fd44b648-7fmvk                            1/1     Running   0
  12m
```

4. Verify Metal3 helm release version change:

```
helm ls -n metal3-system

# Expected output
```

NAME	NAMESPACE	REVISION	UPDATED	STATUS
CHART	APP VERSION			
metal3	metal3-system	2	2024-06-17 12:43:06.774802846 +0000 UTC	deployed
metal3-0.7.4	1.16.0			

24.4 Cluster API upgrade

The Cluster API (CAPI) controllers on a Metal³ management cluster are not currently managed via Helm, this section describes the upgrade process.



Note

This section assumes you have installed `clusterctl` and configured on your system as described in the Metal3 quickstart ([Chapter 1, BMC automated deployments with Metal³](#))

When upgrading to Edge 3.0.2 from any previous version it will be necessary to upgrade the RKE2 providers:

```
clusterctl upgrade apply --bootstrap "rke2:v0.4.1" --control-plane "rke2:v0.4.1"
```



Warning

Please ensure the versions selected align with those described in the Release Notes ([Chapter 33, Release Notes](#)), usage of other upstream releases is not supported.

25 Downstream clusters

This section covers how to do various Day 2 operations for different parts of your downstream cluster using your management cluster.

25.1 Introduction

This section is meant to be a **starting point** for the Day 2 operations documentation. You can find the following information.

1. The default components ([Section 25.1.1, “Components”](#)) used to achieve Day 2 operations over multiple downstream clusters.
2. Determining which Day 2 resources should you use for your specific use-case ([Section 25.1.2, “Determine your use-case”](#)).
3. The suggested workflow sequence ([Section 25.1.3, “Day 2 workflow”](#)) for Day 2 operations.

25.1.1 Components

Below you can find a description of the default components that should be setup on either your management cluster or your downstream clusters so that you can successfully perform Day 2 operations.

25.1.1.1 Rancher



Note

For use-cases where you want to utilise Fleet ([Chapter 6, Fleet](#)) without Rancher, you can skip the Rancher component all together.

Responsible for the management of your downstream clusters. Should be deployed on your management cluster.

For more information, see [Chapter 4, Rancher](#).

25.1.1.2 Fleet

Responsible for multi-cluster resource deployment.

Typically offered by the Rancher component. For use-cases where Rancher is not used, can be deployed as a standalone component.

For more information on installing Fleet as a standalone component, see Fleet's [Installation Details \(https://fleet.rancher.io/installation\)](https://fleet.rancher.io/installation).

For more information regarding the Fleet component, see [Chapter 6, Fleet](#).



Important

This documentation heavily relies on Fleet and more specifically on the GitRepo and Bundle resources (more on this in [Section 25.1.2, "Determine your use-case"](#)) for establishing a GitOps way of automating the deployment of resources related to Day 2 operations.

For use-cases, where a third party GitOps tool usage is desired, see:

1. For OS package updates - [Section 25.3.4.3, "SUC Plan deployment - third-party GitOps workflow"](#)
2. For Kubernetes distribution upgrades - [Section 25.4.4.3, "SUC Plan deployment - third-party GitOps workflow"](#)
3. For Helm chart upgrades - retrieve the chart version supported by the desired Edge release from the [Section 33.1, "Abstract"](#) page and populate the chart version and URL in your third party GitOps tool

25.1.1.3 System-upgrade-controller (SUC)

The **system-upgrade-controller (SUC)** is responsible for executing tasks on specified nodes based on configuration data provided through a custom resource, called a Plan. Should be located on each downstream cluster that requires some sort of a Day 2 operation.

For more information regarding **SUC**, see the upstream [repository \(https://github.com/rancher/system-upgrade-controller\)](https://github.com/rancher/system-upgrade-controller).

For information on how to deploy **SUC** on your downstream clusters, first determine your use-case ([Section 25.1.2, “Determine your use-case”](#)) and then refer to either [Section 25.2.1.1, “SUC deployment using a GitRepo resource”](#), or [Section 25.2.1.2, “SUC deployment using a Bundle resource”](#) for SUC deployment information.

25.1.2 Determine your use-case

As mentioned previously, resources related to Day 2 operations are propagated to downstream clusters using Fleet’s GitRepo and Bundle resources.

Below you can find more information regarding what these resources do and for which use-cases should they be used for Day 2 operations.

25.1.2.1 GitRepo

A GitRepo is a Fleet ([Chapter 6, Fleet](#)) resource that represents a Git repository from which Fleet can create Bundles. Each Bundle is created based on configuration paths defined inside of the GitRepo resource. For more information, see the [GitRepo \(https://fleet.rancher.io/gitrepo-add\)](https://fleet.rancher.io/gitrepo-add) [↗](#) documentation.

In terms of Day 2 operations GitRepo resources are normally used to deploy SUC or SUC Plans on **non air-gapped** environments that utilise a *Fleet GitOps* approach.

Alternatively, GitRepo resources can also be used to deploy SUC or SUC Plans on **air-gapped** environments, **if you mirror your repository setup through a local git server**.

25.1.2.2 Bundle

Bundles hold **raw** Kubernetes resources that will be deployed on the targeted cluster. Usually they are created from a GitRepo resource, but there are use-cases where they can be deployed manually. For more information refer to the [Bundle \(https://fleet.rancher.io/bundle-add\)](https://fleet.rancher.io/bundle-add) [↗](#) documentation.

In terms of Day 2 operations Bundle resources are normally used to deploy SUC or SUC Plans on **air-gapped** environments that do not use some form of *local GitOps* procedure (e.g. a **local git server**).


Alternatively, if your use-case does not allow for a *GitOps* workflow (e.g. using a Git repository), **Bundle** resources could also be used to deploy SUC or SUC Plans on **non air-gapped** environments.

25.1.3 Day 2 workflow

The following is a Day 2 workflow that should be followed when upgrading a downstream cluster to a specific Edge release.

1. OS package update (*Section 25.3, "OS package update"*)
2. Kubernetes version upgrade (*Section 25.4, "Kubernetes version upgrade"*)
3. Helm chart upgrade (*Section 25.5, "Helm chart upgrade"*)

25.2 System upgrade controller deployment guide

The **system-upgrade-controller (SUC)** is responsible for deploying resources on specific nodes of a cluster based on configurations defined in a custom resource called a **Plan**. For more information, see the [upstream \(https://github.com/rancher/system-upgrade-controller\)](https://github.com/rancher/system-upgrade-controller)  documentation.



Note

This section focuses solely on deploying the system-upgrade-controller. **Plan** resources should be deployed from the following documentations:

1. OS package update (*Section 25.3, "OS package update"*)
2. Kubernetes version upgrade (*Section 25.4, "Kubernetes version upgrade"*)
3. Helm chart upgrade (*Section 25.5, "Helm chart upgrade"*)

25.2.1 Deployment



Note

This section assumes that you are going to use Fleet (*Chapter 6, Fleet*) to orchestrate the SUC deployment. Users using a third-party GitOps workflow should see *Section 25.2.1.3, “Deploying system-upgrade-controller when using a third-party GitOps workflow”* for information on what resources they need to setup in their workflow.

To determine the resource to use, refer to *Section 25.1.2, “Determine your use-case”*.

25.2.1.1 SUC deployment using a GitRepo resource

This section covers how to create a GitRepo resource that will ship the needed SUC Plans for a successful SUC deployment to your **target** downstream clusters.

The Edge team maintains a ready to use GitRepo resource for SUC in each of our suse-edge/fleet-examples releases (<https://github.com/suse-edge/fleet-examples/releases>)  under gitrepos/day2/system-upgrade-controller-gitrepo.yaml.



Important

If using the suse-edge/fleet-examples repository, make sure you are using the resources from a dedicated release (<https://github.com/suse-edge/fleet-examples/releases>)  tag.

GitRepo creation can be done in one of the following ways:

- Through the Rancher UI (*Section 25.2.1.1.1, “GitRepo deployment - Rancher UI”*) (when Rancher is available)
- By manually deploying (*Section 25.2.1.1.2, “GitRepo creation - manual”*) the resources to your management cluster

Once created, Fleet will be responsible for picking up the resource and deploying the SUC resources to all your **target** clusters. For information on how to track the deployment process, see *Section 25.2.2.1, “Monitor SUC deployment”*.

25.2.1.1.1 GitRepo deployment - Rancher UI

1. In the upper left corner, # → **Continuous Delivery**
2. Go to **Git Repos** → **Add Repository**

If you use the [suse-edge/fleet-examples](#) repository:

1. **Repository URL** - <https://github.com/suse-edge/fleet-examples.git>
2. **Watch** → **Revision** - choose a [release \(https://github.com/suse-edge/fleet-examples/releases\)](https://github.com/suse-edge/fleet-examples/releases) tag for the [suse-edge/fleet-examples](#) repository that you wish to use, e.g. [re-release-3.0.1](#).
3. Under **Paths** add the path to the **system-upgrade-controller** as seen in the release tag - [fleets/day2/system-upgrade-controller](#)
4. Select **Next** to move to the **target** configuration section
5. **Only select clusters for which you wish to deploy the** [system-upgrade-controller](#).
When you are satisfied with your configurations, click **Create**

Alternatively, if you decide to use your own repository to host these files, you would need to provide your repo data above.

25.2.1.1.2 GitRepo creation - manual

1. Choose the desired Edge [release \(https://github.com/suse-edge/fleet-examples/releases\)](https://github.com/suse-edge/fleet-examples/releases) tag that you wish to deploy the **SUC GitRepo** from (referenced below as `_${REVISION}_`).
2. Pull the **GitRepo** resource:

```
curl -o system-upgrade-controller-gitrepo.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/${REVISION}/gitrepos/day2/system-upgrade-controller-gitrepo.yaml
```

3. Edit the **GitRepo** configurations, under `spec.targets` specify your desired target list. By default, the **GitRepo** resources from the [suse-edge/fleet-examples](#) are **NOT** mapped to any down stream clusters.

- To match all clusters, change the default **GitRepo target** to:

```
spec:
```



```
targets:
- clusterSelector: {}
```

- Alternatively, if you want a more granular cluster selection, see [Mapping to Downstream Clusters \(https://fleet.rancher.io/gitrepo-targets\)](https://fleet.rancher.io/gitrepo-targets) ↗

4. Apply the **GitRepo** resource to your management cluster:

```
kubectl apply -f system-upgrade-controller-gitrepo.yaml
```

5. View the created **GitRepo** resource under the fleet-default namespace:

```
kubectl get gitrepo system-upgrade-controller -n fleet-default

# Example output
NAME                                REPO
COMMIT      BUNDLEDEPLOYMENTS-READY  STATUS
system-upgrade-controller  https://github.com/suse-edge/fleet-examples.git
release-3.0.1    0/0
```

25.2.1.2 SUC deployment using a Bundle resource

This section covers how to create a Bundle resource that will ship the needed SUC Plans for a successful **SUC** deployment to your **target** downstream clusters.

The Edge team maintains a ready to use Bundle resources for **SUC** in each of our suse-edge/fleet-examples releases (<https://github.com/suse-edge/fleet-examples/releases>) ↗ under bundles/day2/system-upgrade-controller/controller-bundle.yaml.

Important

If using the suse-edge/fleet-examples repository, make sure you are using the resources from a dedicated release (<https://github.com/suse-edge/fleet-examples/releases>) ↗ tag.

Bundle creation can be done in one of the following ways:

- Through the Rancher UI ([Section 25.2.1.2.1, "Bundle creation - Rancher UI"](#)) (when Rancher is available)
- By manually deploying ([Section 25.2.1.2.2, "Bundle creation - manual"](#)) the resources to your management cluster

Once created, **Fleet** will be responsible for pickuping the resource and deploying the **SUC** resources to all your **target** clusters. For information on how to track the deployment process, see [Section 25.2.2.1, "Monitor SUC deployment"](#).

25.2.1.2.1 Bundle creation - Rancher UI

1. In the upper left corner, # → **Continuous Delivery**

2. Go to **Advanced > Bundles**

3. Select **Create from YAML**

4. From here you can create the Bundle in one of the following ways:

- By manually copying the file content to the **Create from YAML** page. File content can be retrieved from this url - [https://raw.githubusercontent.com/suse-edge/fleet-examples/\\${REVISION}/bundles/day2/system-upgrade-controller/controller-bundle.yaml](https://raw.githubusercontent.com/suse-edge/fleet-examples/${REVISION}/bundles/day2/system-upgrade-controller/controller-bundle.yaml). Where `${REVISION}` is the Edge release (<https://github.com/suse-edge/fleet-examples/releases>) tag that you desire (e.g. `release-3.0.1`).
- By cloning the `suse-edge/fleet-examples` repository to the desired release (<https://github.com/suse-edge/fleet-examples/releases>) tag and selecting the **Read from File** option in the **Create from YAML** page. From there, navigate to `bundles/day2/system-upgrade-controller` directory and select `controller-bundle.yaml`. This will auto-populate the **Create from YAML** page with the Bundle content.

5. Change the **target** clusters for the Bundle:

- To match all downstream clusters change the default Bundle `.spec.targets` to:

```
spec:
  targets:
  - clusterSelector: {}
```

- For a more granular downstream cluster mappings, see [Mapping to Downstream Clusters \(https://fleet.rancher.io/gitrepo-targets\)](https://fleet.rancher.io/gitrepo-targets).

6. **Create**

25.2.1.2.2 Bundle creation - manual

1. Choose the desired Edge [release](https://github.com/suse-edge/fleet-examples/releases) (<https://github.com/suse-edge/fleet-examples/releases>) tag that you wish to deploy the **SUC Bundle** from (referenced below as `${REVISION}`).
2. Pull the **Bundle** resource:

```
curl -o controller-bundle.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/${REVISION}/bundles/day2/system-upgrade-controller/controller-bundle.yaml
```

3. Edit the **Bundle target** configurations, under `spec.targets` provide your desired target list. By default the **Bundle** resources from the `suse-edge/fleet-examples` are **NOT** mapped to any down stream clusters.

- To match all clusters change the default **Bundle target** to:

```
spec:
  targets:
  - clusterSelector: {}
```

- Alternatively, if you want a more granular cluster selection, see [Mapping to Down-stream Clusters](https://fleet.rancher.io/gitrepo-targets) (<https://fleet.rancher.io/gitrepo-targets>)

4. Apply the **Bundle** resource to your management cluster:

```
kubectl apply -f controller-bundle.yaml
```

5. View the created **Bundle** resource under the fleet-default namespace:

```
kubectl get bundles system-upgrade-controller -n fleet-default

# Example output
NAME                                BUNDLEDEPLOYMENTS-READY  STATUS
system-upgrade-controller          0/0
```

25.2.1.3 Deploying system-upgrade-controller when using a third-party GitOps workflow

To deploy the `system-upgrade-controller` using a third-party GitOps tool, depending on the tool, you might need information for the `system-upgrade-controller` Helm chart or Kubernetes resources, or both.

Choose a specific Edge [release \(https://github.com/suse-edge/fleet-examples/releases\)](https://github.com/suse-edge/fleet-examples/releases) from which you wish to use the **SUC** from.

From there, the **SUC** Helm chart data can be found under the `helm` configuration section of the `fleets/day2/system-upgrade-controller/fleet.yaml` file.

The **SUC** Kubernetes resources can be found under the **SUC Bundle** configuration under `.spec.resources.content`. The location for the bundle is `bundles/day2/system-upgrade-controller/controller-bundle.yaml`.

Use the above mentioned resources to populate the data that your third-party GitOps workflow needs in order to deploy **SUC**.

25.2.2 Monitor SUC resources using Rancher

This section covers how to monitor the lifecycle of the **SUC** deployment and any deployed **SUC** Plans using the Rancher UI.

25.2.2.1 Monitor SUC deployment

To check the **SUC** pod logs for a specific cluster:

1. In the upper left corner, # → `<your-cluster-name>`
2. Select **Workloads** → **Pods**
3. Under the namespace drop down menu select the `cattle-system` namespace

The screenshot shows the Rancher UI navigation menu for a cluster named "k3s-selinux". The menu is open, displaying various Kubernetes resource types. The "Pods" option is highlighted in blue, and a red arrow points to it from the right. Another red arrow points to the "Workloads" header. On the right side of the screen, the "Pods" section is visible with a star icon and a "State" dropdown menu.

Resource Type	Count
Cluster	>
Workloads	∨
CronJobs	{=} 0
DaemonSets	{=} 0
Deployments	{=} 0
Jobs	{=} 0
StatefulSets	{=} 0
Pods	{=} 0
Apps	>
Service Discovery	>
Storage	>
Policy	>
More Resources	>


4. In the Pod filter bar, write the **SUC** name - system-upgrade-controller
5. On the right of the pod select # → **View Logs**

The screenshot shows the Rancher management interface for a cluster named 'k3s-selinux'. The left sidebar contains navigation icons: a hamburger menu, a home icon, five 'K3S' labels, a blue bull icon (selected), a sailboat icon, a house icon, a network icon, a person icon, a puzzle piece icon, and a globe icon. The main navigation menu is open, listing various Kubernetes resources. The 'Pods' resource is highlighted in blue and shows a count of 3. Other resources include Cluster, Workloads (with a dropdown arrow), CronJobs (0), DaemonSets (0), Deployments (3), Jobs (0), StatefulSets (0), Apps, Service Discovery, Storage, Policy, and More Resources. On the right, the 'Pods' section is partially visible, showing a 'Down' button and a 'State' dropdown menu with 'Running' selected.

Resource	Count
Cluster	>
Workloads	▼
CronJobs	{=} 0
DaemonSets	{=} 0
Deployments	{=} 3
Jobs	{=} 0
StatefulSets	{=} 0
Pods	{=} 3
Apps	>
Service Discovery	>
Storage	>
Policy	>
More Resources	>


25.2.2.2 Monitor SUC Plans

Important

The **SUC Plan** Pods are kept alive for **15** minutes. After that they are removed by the corresponding Job that created them. To have access to the **SUC Plan** Pod logs, you should enable logging for your cluster. For information on how to do this in Rancher, see [Rancher Integration with Logging Services \(https://ranchermanager.docs.rancher.com/v2.8/integrations-in-rancher/logging\)](https://ranchermanager.docs.rancher.com/v2.8/integrations-in-rancher/logging) .

To check **Pod** logs for the specific **SUC** plan:

1. In the upper left corner, # → <**your-cluster-name**>
2. Select **Workloads** → **Pods**
3. Under the namespace drop down menu select the cattle-system namespace

☰  k3s-selinux


🏠

K3S

K3S

K3S

K3S



⏏

🚢

🏠

🔗

👤

🧩

🌐

Cluster >

Workloads ▾

CronJobs {=} 0

DaemonSets {=} 0

Deployments {=} 0

Jobs {=} 0

StatefulSets {=} 0

Pods {=} 0

Apps >

Service Discovery >

Storage >

Policy >

More Resources >

Pods ☆

State ⌵

4. In the Pod filter bar, write the name for your **SUC Plan** Pod. The name will be in the following template format: apply-<plan_name>-on-<node_name>

Pods ☆

Download YAML

Delete

State ◇ Name ◇

Namespace: cattle-system

<input type="checkbox"/>	Completed	apply-k3s-plan-control-plane-on-k3s-noselinu sd5k2
<input type="checkbox"/>	Unknown	apply-k3s-plan-control-plane-on-k3s-noselinu w4pbm

Note how in *Figure 1*, we have one Pod in **Completed** and one in **Unknown** state. This is expected and has happened due to the Kubernetes version upgrade on the node.

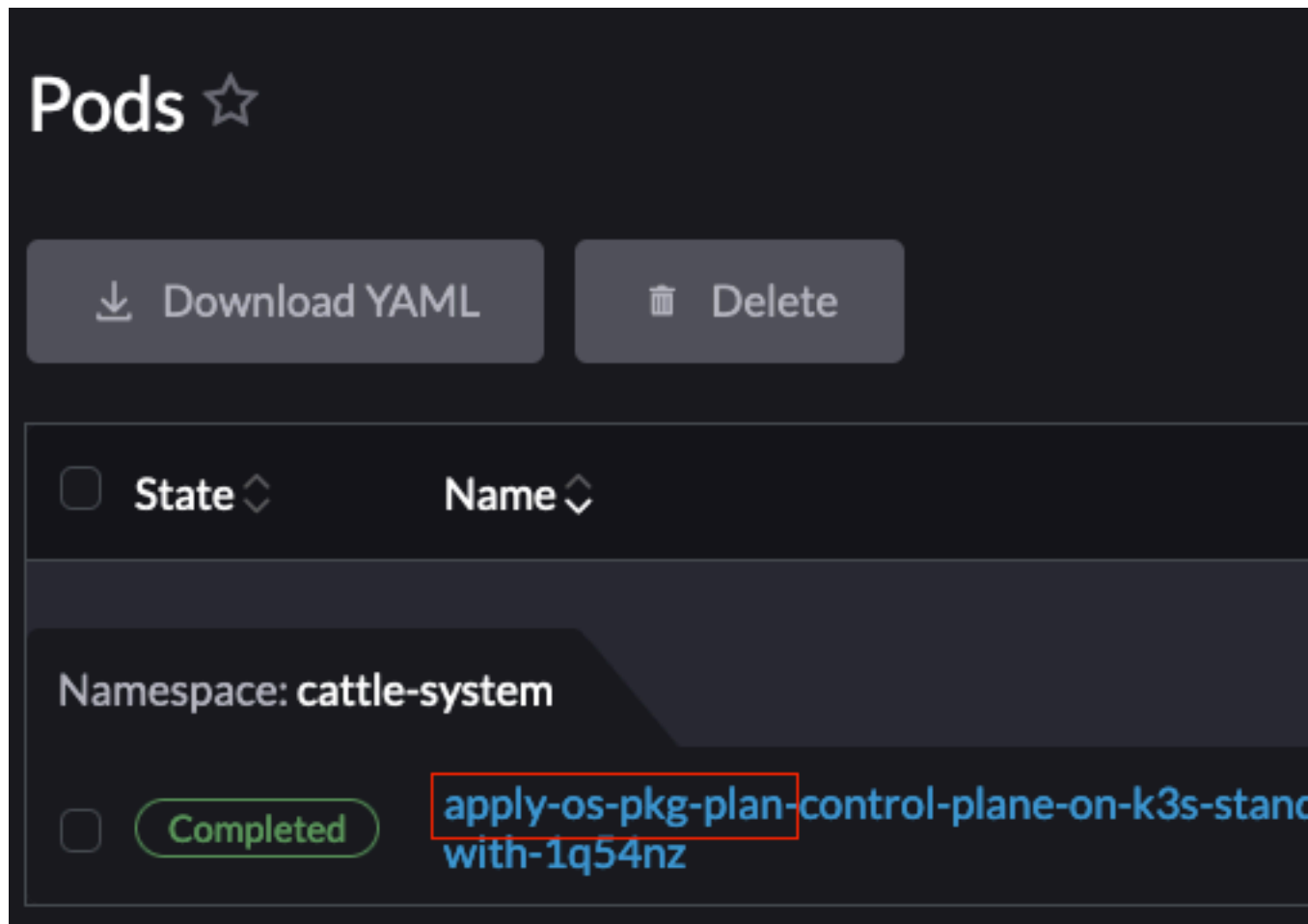


FIGURE 25.2: EXAMPLE OS PACKAGE UPDATE PLAN PODS

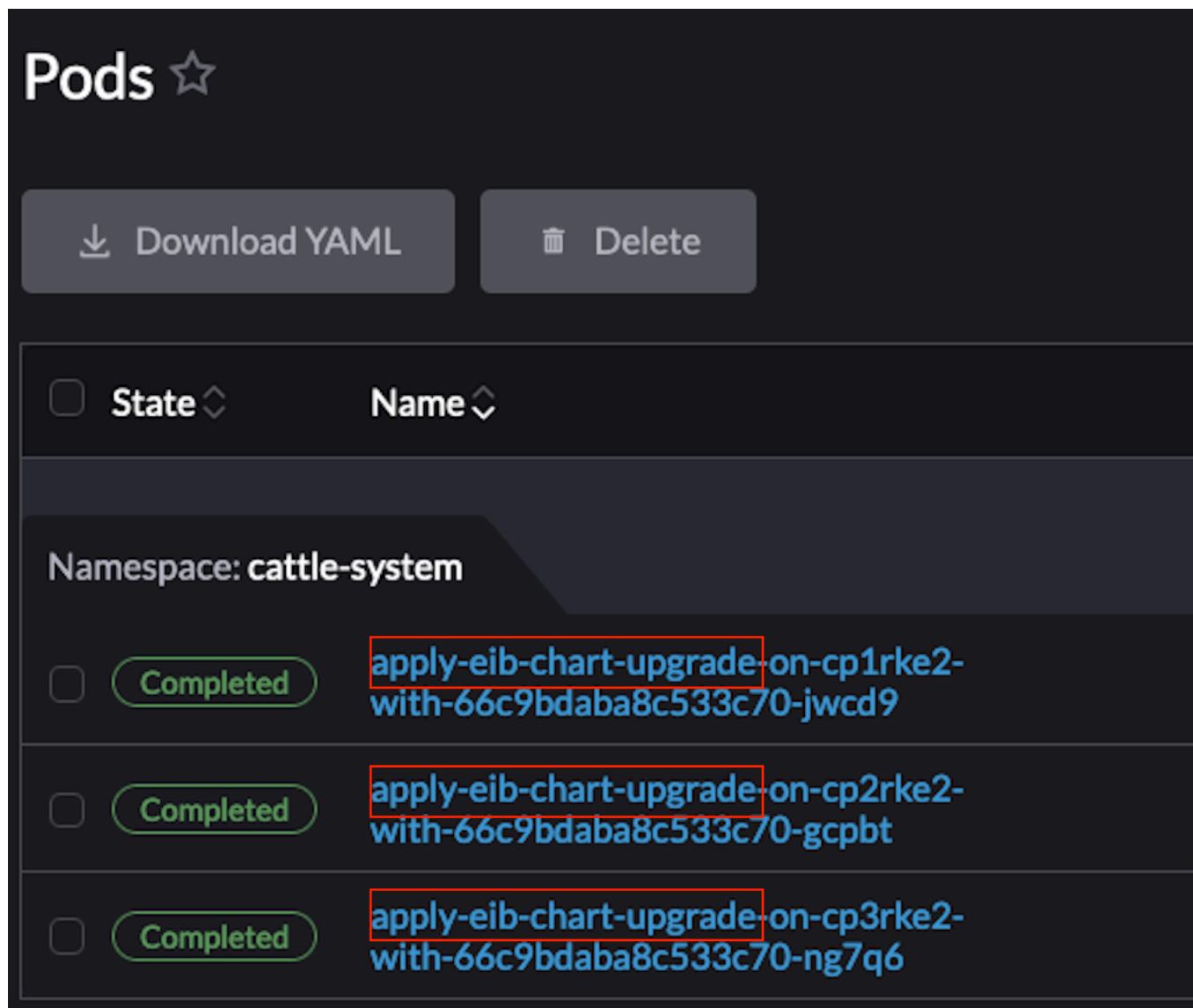


FIGURE 25.3: EXAMPLE OF UPGRADE PLAN PODS FOR EIB DEPLOYED HELM CHARTS ON AN HA CLUSTER



5. Select the pod that you want to review the logs of and navigate to # → **View Logs**

25.3 OS package update

25.3.1 Components


This section covers the custom components that the `OS package update` process uses over the default `Day 2` components (*Section 25.1.1, "Components"*).

25.3.1.1 `edge-update.service`

Systemd service responsible for performing the `OS package update`. Uses the `transactional-update` (<https://kubic.opensuse.org/documentation/man-pages/transactional-update.8.html>)  command to perform a `distribution upgrade` (https://en.opensuse.org/SDB:Zypper_usage#Distribution_upgrade)  (`dup`).




Note

If you wish to use a `normal upgrade` (https://en.opensuse.org/SDB:Zypper_usage#Updating_packages)  method, create a `edge-update.conf` file under `/etc/edge/` on each node. Inside this file, add the `UPDATE_METHOD=up` variable.

Shipped through a **SUC plan**, which should be located on each **downstream cluster** that is in need of a OS package update.

25.3.2 Requirements

General:

1. **SCC registered machine** - All downstream cluster nodes should be registered to `https://scc.suse.com/`. This is needed so that the `edge-update.service` can successfully connect to the needed OS RPM repositories.
2. **Make sure that SUC Plan tolerations match node tolerations** - If your Kubernetes cluster nodes have custom **taints**, make sure to add `tolerations` (<https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/>)  for those taints in the **SUC Plans**. By default **SUC Plans** have tolerations only for **control-plane** nodes. Default tolerations include:
 - `CriticalAddonsOnly=true:NoExecute`
 - `node-role.kubernetes.io/control-plane:NoSchedule`
 - `node-role.kubernetes.io/etcd:NoExecute`



Note

Any additional tolerations must be added under the `.spec.tolerations` section of each Plan. **SUC Plans** related to the OS package update can be found in the [suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples) (<https://github.com/suse-edge/fleet-examples>) repository under `fleets/day2/system-upgrade-controller-plans/os-pkg-update`. **Make sure you use the Plans from a valid repository release** (<https://github.com/suse-edge/fleet-examples/releases>) tag.

An example of defining custom tolerations for the **control-plane** SUC Plan, would look like this:

```
apiVersion: upgrade.cattle.io/v1
kind: Plan
metadata:
  name: os-pkg-plan-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. **Mirror SUSE RPM repositories** - OS RPM repositories should be locally mirrored so that `edge-update.service` can have access to them. This can be achieved using RMT (<https://github.com/SUSE/rmt>).

25.3.3 Update procedure



Note

This section assumes you will be deploying the `OS package update SUC Plan` using Fleet ([Chapter 6, Fleet](#)). If you intend to deploy the **SUC Plan** using a different approach, refer to [Section 25.3.4.3, "SUC Plan deployment - third-party GitOps workflow"](#).

The `OS package update procedure` revolves around deploying **SUC Plans** to downstream clusters. These plans then hold information about how and on which nodes to deploy the `edge-update.service systemd.service`. For information regarding the structure of a **SUC Plan**, refer to the [upstream \(https://github.com/rancher/system-upgrade-controller?tab=readme-ov-file#example-plans\)](https://github.com/rancher/system-upgrade-controller?tab=readme-ov-file#example-plans) documentation.

`OS package update SUC Plans` are shipped in the following ways:

- Through a `GitRepo` resources - [Section 25.3.4.1, "SUC Plan deployment - GitRepo resource"](#)
- Through a `Bundle` resource - [Section 25.3.4.2, "SUC Plan deployment - Bundle resource"](#)

To determine which resource you should use, refer to [Section 25.1.2, "Determine your use-case"](#).

For a full overview of what happens during the *update procedure*, refer to the [Section 25.3.3.1, "Overview"](#) section.

25.3.3.1 Overview

This section aims to describe the full workflow that the **OS package update process** goes through from start to finish.

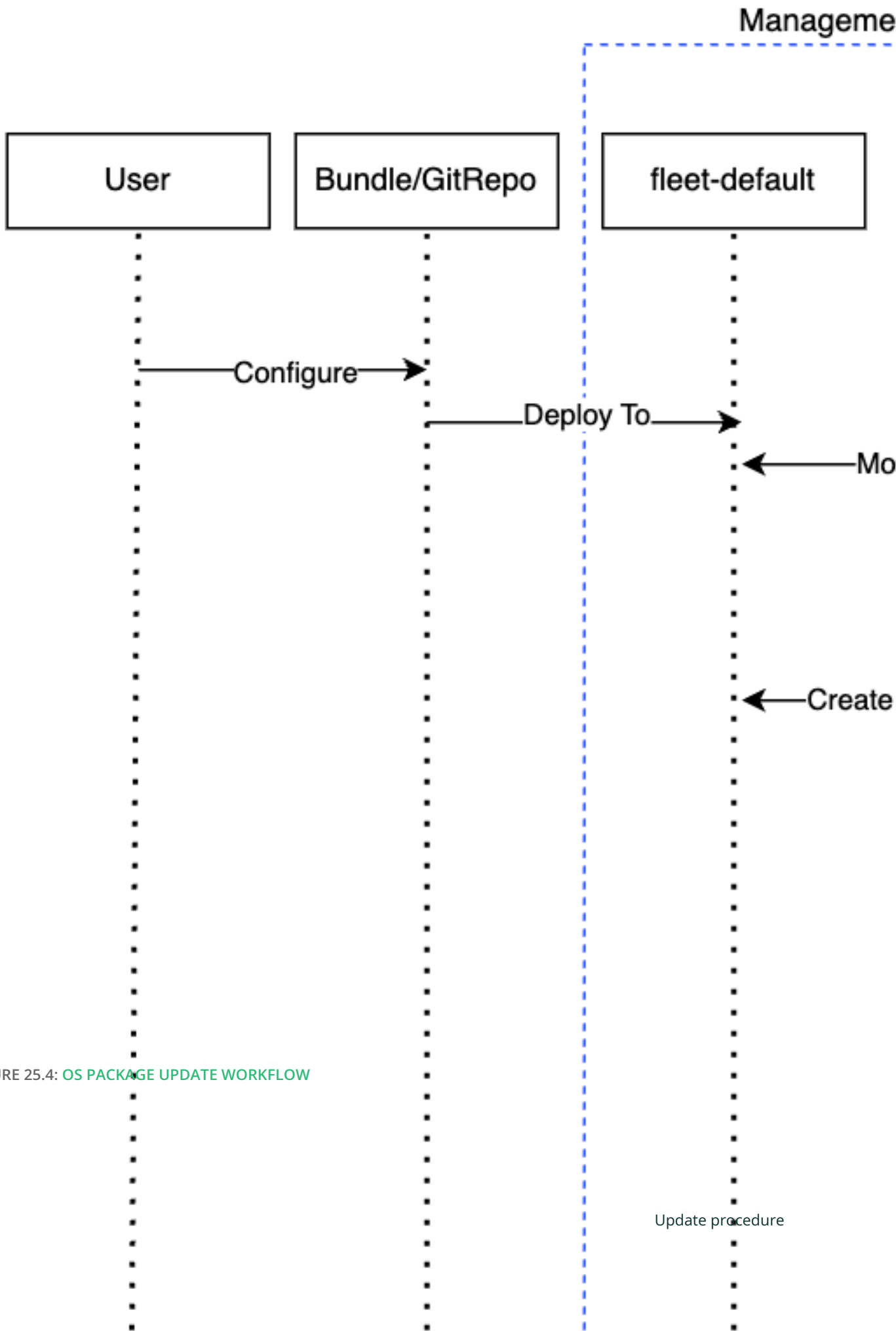


FIGURE 25.4: OS PACKAGE UPDATE WORKFLOW

OS package update steps:

1. Based on his use-case, the user determines whether to use a **GitRepo** or a **Bundle** resource for the deployment of the OS package update SUC Plans to the desired downstream clusters. For information on how to map a **GitRepo/Bundle** to a specific set of downstream clusters, see [Mapping to Downstream Clusters \(https://fleet.rancher.io/gitrepo-targets\)](https://fleet.rancher.io/gitrepo-targets) ⁷.
 - a. If you are unsure whether you should use a **GitRepo** or a **Bundle** resource for the **SUC Plan** deployment, refer to [Section 25.1.2, "Determine your use-case"](#).
 - b. For **GitRepo/Bundle** configuration options, refer to [Section 25.3.4.1, "SUC Plan deployment - GitRepo resource"](#) or [Section 25.3.4.2, "SUC Plan deployment - Bundle resource"](#).
2. The user deploys the configured **GitRepo/Bundle** resource to the fleet-default namespace in his management cluster. This is done either **manually** or through the **Rancher UI** if such is available.
3. Fleet ([Chapter 6, Fleet](#)) constantly monitors the fleet-default namespace and immediately detects the newly deployed **GitRepo/Bundle** resource. For more information regarding what namespaces does Fleet monitor, refer to Fleet's [Namespaces \(https://fleet.rancher.io/namespaces\)](https://fleet.rancher.io/namespaces) ⁷ documentation.
4. If the user has deployed a **GitRepo** resource, Fleet will reconcile the **GitRepo** and based on its **paths** and **fleet.yaml** configurations it will deploy a **Bundle** resource in the fleet-default namespace. For more information, refer to Fleet's [GitRepo Contents \(https://fleet.rancher.io/gitrepo-content\)](https://fleet.rancher.io/gitrepo-content) ⁷ documentation.
5. Fleet then proceeds to deploy the Kubernetes resources from this **Bundle** to all the targeted downstream clusters. In the context of OS package updates, Fleet deploys the following resources from the **Bundle**:
 - a. os-pkg-plan-agent SUC Plan - instructs **SUC** on how to do a package update on cluster **agent** nodes. Will **not** be interpreted if the cluster consists only from *control-plane* nodes.
 - b. os-pkg-plan-control-plane SUC Plan - instructs **SUC** on how to do a package update on cluster **control-plane** nodes.
 - c. os-pkg-update Secret - referenced in each **SUC Plan**; ships an update.sh script responsible for creating the edge-update.service systemd.service which will do the actual package update.



Note

The above resources will be deployed in the `cattle-system` namespace of each downstream cluster.

6. On the downstream cluster, **SUC** picks up the newly deployed **SUC Plans** and deploys an **Update Pod** on each node that matches the **node selector** defined in the **SUC Plan**. For information how to monitor the **SUC Plan Pod**, refer to [Section 25.2.2.2, "Monitor SUC Plans"](#).
7. The **Update Pod** (deployed on each node) **mounts** the `os-pkg-update` Secret and **executes** the `update.sh` script that the Secret ships.
8. The `update.sh` proceeds to do the following:
 - a. Create the `edge-update.service` - the service created will be of type **oneshot** and will adopt the following workflow:
 - i. Update all package versions on the node OS, by executing:

```
transactional-update cleanup dup
```

- ii. After a successful `transactional-update`, schedule a system **reboot** so that the package version updates can take effect



Note

System reboot will be scheduled for **1 minute** after a successful `transactional-update` execution.

- b. Start the `edge-update.service` and wait for it to complete
 - c. Cleanup the `edge-update.service` - after the **systemd.service** has done its job, it is removed from the system in order to ensure that no accidental executions/reboots happen in the future.

The OS package update procedure finishes with the **system reboot**. After the reboot all OS package versions should be updated to their respective latest version as seen in the available OS RPM repositories.

25.3.4 OS package update - SUC Plan deployment

This section describes how to orchestrate the deployment of **SUC Plans** related OS package updates using Fleet's **GitRepo** and **Bundle** resources.

25.3.4.1 SUC Plan deployment - GitRepo resource

A **GitRepo** resource, that ships the needed OS package update SUC Plans, can be deployed in one of the following ways:

1. Through the Rancher UI - *Section 25.3.4.1.1, "GitRepo creation - Rancher UI"* (when Rancher is available).
2. By manually deploying (*Section 25.3.4.1.2, "GitRepo creation - manual"*) the resource to your management cluster.

Once deployed, to monitor the OS package update process of the nodes of your targeted cluster, refer to the *Section 25.2.2.2, "Monitor SUC Plans"* documentation.

25.3.4.1.1 GitRepo creation - Rancher UI

1. In the upper left corner, # → **Continuous Delivery**
2. Go to **Git Repos** → **Add Repository**

If you use the suse-edge/fleet-examples repository:

1. **Repository URL** - <https://github.com/suse-edge/fleet-examples.git>
2. **Watch** → **Revision** - choose a [release \(https://github.com/suse-edge/fleet-examples/releases\)](https://github.com/suse-edge/fleet-examples/releases) tag for the suse-edge/fleet-examples repository that you wish to use
3. Under **Paths** add the path to the OS package update Fleets that you wish to use - fleets/day2/system-upgrade-controller-plans/os-pkg-update
4. Select **Next** to move to the **target** configuration section. **Only select clusters whose node's packages you wish to upgrade**
5. **Create**

Alternatively, if you decide to use your own repository to host these files, you would need to provide your repo data above.

25.3.4.1.2 GitRepo creation - manual

1. Choose the desired Edge [release](https://github.com/suse-edge/fleet-examples/releases) (https://github.com/suse-edge/fleet-examples/releases) tag that you wish to apply the OS **SUC update Plans** from (referenced below as `${REVISION}`).

2. Pull the **GitRepo** resource:

```
curl -o os-pkg-update-gitrepo.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/${REVISION}/gitrepos/day2/os-pkg-update-gitrepo.yaml
```

3. Edit the **GitRepo** configuration, under `spec.targets` specify your desired target list. By default the `GitRepo` resources from the `suse-edge/fleet-examples` are **NOT** mapped to any down stream clusters.

- To match all clusters change the default `GitRepo target` to:

```
spec:
  targets:
  - clusterSelector: {}
```

- Alternatively, if you want a more granular cluster selection see [Mapping to Down-stream Clusters](https://fleet.rancher.io/gitrepo-targets) (https://fleet.rancher.io/gitrepo-targets)

4. Apply the **GitRepo** resources to your `management cluster`:

```
kubectl apply -f os-pkg-update-gitrepo.yaml
```

5. View the created **GitRepo** resource under the `fleet-default` namespace:

```
kubectl get gitrepo os-pkg-update -n fleet-default

# Example output
NAME                                REPO                                COMMIT
BUNDLEDEPLOYMENTS-READY  STATUS
os-pkg-update             https://github.com/suse-edge/fleet-examples.git  release-3.0.1  0/0
```

25.3.4.2 SUC Plan deployment - Bundle resource

A **Bundle** resource, that ships the needed OS package update SUC Plans, can be deployed in one of the following ways:

1. Through the Rancher UI - *Section 25.3.4.2.1, "Bundle creation - Rancher UI"* (when Rancher is available).
2. By manually deploying (*Section 25.3.4.2.2, "Bundle creation - manual"*) the resource to your management cluster.

Once deployed, to monitor the OS package update process of the nodes of your targeted cluster, refer to the *Section 25.2.2.2, "Monitor SUC Plans"* documentation.

25.3.4.2.1 Bundle creation - Rancher UI

1. In the upper left corner, click # → **Continuous Delivery**
2. Go to **Advanced > Bundles**
3. Select **Create from YAML**
4. From here you can create the Bundle in one of the following ways:
 - a. By manually copying the **Bundle** content to the **Create from YAML** page. Content can be retrieved from [https://raw.githubusercontent.com/suse-edge/fleet-examples/\\${REVISION}/bundles/day2/system-upgrade-controller-plans/os-pkg-update/pkg-update-bundle.yaml](https://raw.githubusercontent.com/suse-edge/fleet-examples/${REVISION}/bundles/day2/system-upgrade-controller-plans/os-pkg-update/pkg-update-bundle.yaml), where `${REVISION}` is the Edge release (<https://github.com/suse-edge/fleet-examples/releases>) that you are using
 - b. By cloning the [suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples.git) repository to the desired [release](https://github.com/suse-edge/fleet-examples/releases) tag and selecting the **Read from File** option in the **Create from YAML** page. From there, navigate to `bundles/day2/system-upgrade-controller-plans/os-pkg-update` directory and select `pkg-update-bundle.yaml`. This will auto-populate the **Create from YAML** page with the Bundle content.

5. Change the **target** clusters for the Bundle:

- To match all downstream clusters change the default Bundle `.spec.targets` to:

```
spec:
  targets:
  - clusterSelector: {}
```

- For a more granular downstream cluster mappings, see [Mapping to Downstream Clusters \(https://fleet.rancher.io/gitrepo-targets\)](https://fleet.rancher.io/gitrepo-targets).

6. Select **Create**

25.3.4.2.2 Bundle creation - manual

1. Choose the desired Edge [release \(https://github.com/suse-edge/fleet-examples/releases\)](https://github.com/suse-edge/fleet-examples/releases) tag that you wish to apply the OS package update **SUC Plans** from (referenced below as `_${REVISION}_`).

2. Pull the **Bundle** resource:

```
curl -o pkg-update-bundle.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/$_REVISION_/bundles/day2/system-upgrade-controller-plans/os-pkg-update/pkg-update-bundle.yaml
```

3. Edit the Bundle **target** configurations, under `spec.targets` provide your desired target list. By default the Bundle resources from the [suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples) are **NOT** mapped to any down stream clusters.

- To match all clusters change the default Bundle **target** to:

```
spec:
  targets:
  - clusterSelector: {}
```

- Alternatively, if you want a more granular cluster selection see [Mapping to Downstream Clusters \(https://fleet.rancher.io/gitrepo-targets\)](https://fleet.rancher.io/gitrepo-targets).

4. Apply the **Bundle** resources to your management cluster:

```
kubectl apply -f pkg-update-bundle.yaml
```

5. View the created **Bundle** resource under the fleet-default namespace:

```
kubectl get bundles os-pkg-update -n fleet-default

# Example output
NAME                BUNDLEDEPLOYMENTS-READY  STATUS
os-pkg-update       0/0
```

25.3.4.3 SUC Plan deployment - third-party GitOps workflow

There might be use-cases where users would like to incorporate the OS package update **SUC Plans** to their own third-party GitOps workflow (e.g. Flux).

To get the OS package update resources that you need, first determine the Edge release (<https://github.com/suse-edge/fleet-examples/releases>) tag of the suse-edge/fleet-examples (<https://github.com/suse-edge/fleet-examples.git>) repository that you would like to use.

After that, resources can be found at fleets/day2/system-upgrade-controller-plans/os-pkg-update, where:

- plan-control-plane.yaml - system-upgrade-controller Plan resource for **control-plane** nodes
- plan-agent.yaml - system-upgrade-controller Plan resource for **agent** nodes
- secret.yaml - secret that ships a script that creates the edge-update.service systemd.service (<https://www.freedesktop.org/software/systemd/man/latest/systemd.service.html>)




Important

These Plan resources are interpreted by the system-upgrade-controller and should be deployed on each downstream cluster that you wish to upgrade. For information on how to deploy the system-upgrade-controller, see *Section 25.2.1.3, "Deploying system-upgrade-controller when using a third-party GitOps workflow"*.

To better understand how your GitOps workflow can be used to deploy the **SUC Plans** for OS package update, it can be beneficial to take a look at the overview ([Section 25.3.3.1, “Overview”](#)) of the update procedure using Fleet.

25.4 Kubernetes version upgrade

Important

This section covers Kubernetes upgrades for downstream clusters that have **NOT** been created through a Rancher ([Chapter 4, Rancher](#)) instance. For information on how to upgrade the Kubernetes version of Rancher created clusters, see [Upgrading and Rolling Back Kubernetes](https://ranchermanager.docs.rancher.com/v2.8/getting-started/installation-and-upgrade/upgrade-and-roll-back-kubernetes#upgrading-the-kubernetes-version) (<https://ranchermanager.docs.rancher.com/v2.8/getting-started/installation-and-upgrade/upgrade-and-roll-back-kubernetes#upgrading-the-kubernetes-version>) .


25.4.1 Components

This section covers the custom components that the Kubernetes upgrade process uses over the default Day 2 components ([Section 25.1.1, “Components”](#)).

25.4.1.1 rke2-upgrade

Image responsible for upgrading the RKE2 version of a specific node.


Shipped through a Pod created by **SUC** based on a **SUC Plan**. The Plan should be located on each **downstream cluster** that is in need of a RKE2 upgrade.

For more information regarding how the rke2-upgrade image performs the upgrade, see the upstream (<https://github.com/rancher/rke2-upgrade/tree/master>)  documentation.

25.4.1.2 k3s-upgrade



Image responsible for upgrading the K3s version of a specific node.

Shipped through a Pod created by **SUC** based on a **SUC Plan**. The Plan should be located on each **downstream cluster** that is in need of a K3s upgrade.


For more information regarding how the `k3s-upgrade` image performs the upgrade, see the upstream (<https://github.com/k3s-io/k3s-upgrade>)  documentation.

25.4.2 Requirements

1. Backup your Kubernetes distribution:

- a. For **imported RKE2 clusters**, see the [RKE2 Backup and Restore \(https://docs.rke2.io/backup_restore\)](https://docs.rke2.io/backup_restore)  documentation.
- b. For **imported K3s clusters**, see the [K3s Backup and Restore \(https://docs.k3s.io/datastore/backup-restore\)](https://docs.k3s.io/datastore/backup-restore)  documentation.


2. Make sure that SUC Plan tolerations match node tolerations

- If your Kubernetes cluster nodes have custom **taints**, make sure to add [tolerations \(https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/\)](https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/)  for those taints in the **SUC Plans**. By default **SUC Plans** have tolerations only for **control-plane** nodes. Default tolerations include:

- `CriticalAddonsOnly = true:NoExecute`
- `node-role.kubernetes.io/control-plane:NoSchedule`
- `node-role.kubernetes.io/etcd:NoExecute`



Note

Any additional tolerations must be added under the `.spec.tolerations` section of each Plan. **SUC Plans** related to the Kubernetes version upgrade can be found in the [suse-edge/fleet-examples \(https://github.com/suse-edge/fleet-examples\)](https://github.com/suse-edge/fleet-examples)  repository under:

- For **RKE2** - [fleets/day2/system-upgrade-controller-plans/rke2-upgrade](https://github.com/suse-edge/fleet-examples/tree/main/fleets/day2/system-upgrade-controller-plans/rke2-upgrade)
- For **K3s** - [fleets/day2/system-upgrade-controller-plans/k3s-upgrade](https://github.com/suse-edge/fleet-examples/tree/main/fleets/day2/system-upgrade-controller-plans/k3s-upgrade)

Make sure you use the Plans from a valid repository release (<https://github.com/suse-edge/fleet-examples/releases>)  tag.

An example of defining custom tolerations for the RKE2 **control-plane** SUC Plan, would look like this:

```
apiVersion: upgrade.cattle.io/v1
kind: Plan
metadata:
  name: rke2-plan-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"
  ...
```

25.4.3 Upgrade procedure



Note

This section assumes you will be deploying **SUC Plans** using Fleet ([Chapter 6, Fleet](#)). If you intend to deploy the **SUC Plan** using a different approach, refer to [Section 25.4.4.3, "SUC Plan deployment - third-party GitOps workflow"](#).

The [Kubernetes version upgrade procedure](#) revolves around deploying **SUC Plans** to downstream clusters. These plans hold information that instructs the **SUC** on which nodes to create Pods which run the `rke2/k3s - upgrade` images. For information regarding the structure of a **SUC Plan**, refer to the [upstream \(https://github.com/rancher/system-upgrade-controller?tab=readme-ov-file#example-plans\)](https://github.com/rancher/system-upgrade-controller?tab=readme-ov-file#example-plans) [↗](#) documentation.

[Kubernetes upgrade Plans](#) are shipped in the following ways:

- Through a [GitRepo](#) resources - [Section 25.4.4.1, "SUC Plan deployment - GitRepo resource"](#)
- Through a [Bundle](#) resource - [Section 25.4.4.2, "SUC Plan deployment - Bundle resource"](#)

To determine which resource you should use, refer to [Section 25.1.2, "Determine your use-case"](#).

For a full overview of what happens during the *update procedure*, refer to the [Section 25.4.3.1, "Overview"](#) section.

25.4.3.1 Overview

This section aims to describe the full workflow that the ***Kubernetes version upgrade process*** goes through from start to finish.

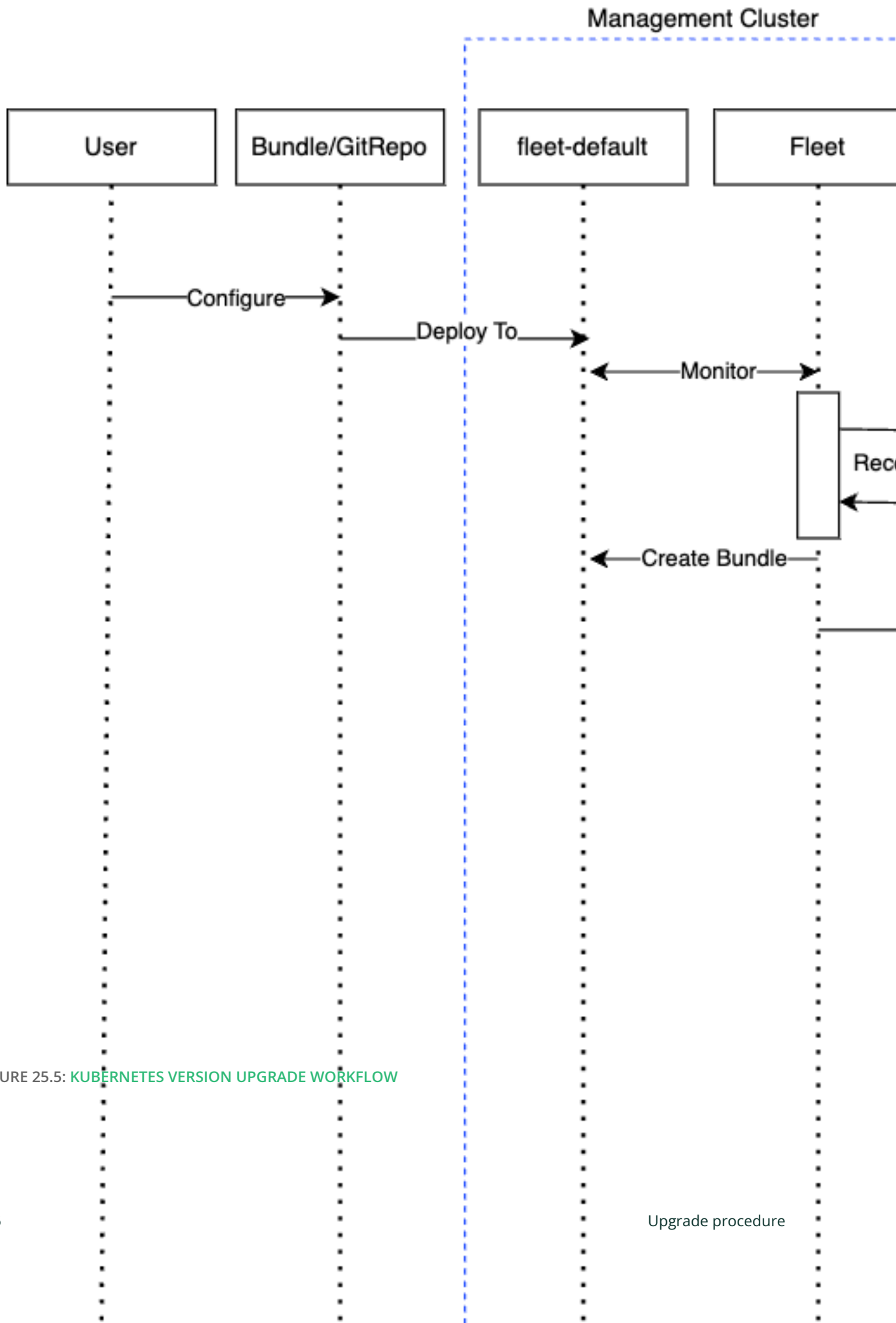





FIGURE 25.5: KUBERNETES VERSION UPGRADE WORKFLOW

Kubernetes version upgrade steps:

1. Based on his use-case, the user determines whether to use a **GitRepo** or a **Bundle** resource for the deployment of the Kubernetes upgrade SUC Plans to the desired downstream clusters. For information on how to map a **GitRepo/Bundle** to a specific set of downstream clusters, see [Mapping to Downstream Clusters \(https://fleet.rancher.io/gitrepo-targets\)](https://fleet.rancher.io/gitrepo-targets) .
 - a. If you are unsure whether you should use a **GitRepo** or a **Bundle** resource for the **SUC Plan** deployment, refer to [Section 25.1.2, "Determine your use-case"](#).
 - b. For **GitRepo/Bundle** configuration options, refer to [Section 25.4.4.1, "SUC Plan deployment - GitRepo resource"](#) or [Section 25.4.4.2, "SUC Plan deployment - Bundle resource"](#).
2. The user deploys the configured **GitRepo/Bundle** resource to the fleet-default namespace in his management cluster. This is done either **manually** or through the **Rancher UI** if such is available.
3. Fleet ([Chapter 6, Fleet](#)) constantly monitors the fleet-default namespace and immediately detects the newly deployed **GitRepo/Bundle** resource. For more information regarding what namespaces does Fleet monitor, refer to Fleet's [Namespaces \(https://fleet.rancher.io/namespaces\)](https://fleet.rancher.io/namespaces)  documentation.
4. If the user has deployed a **GitRepo** resource, Fleet will reconcile the **GitRepo** and based on its **paths** and **fleet.yaml** configurations it will deploy a **Bundle** resource in the fleet-default namespace. For more information, refer to Fleet's [GitRepo Contents \(https://fleet.rancher.io/gitrepo-content\)](https://fleet.rancher.io/gitrepo-content)  documentation.
5. Fleet then proceeds to deploy the Kubernetes resources from this **Bundle** to all the targeted downstream clusters. In the context of the Kubernetes version upgrade, Fleet deploys the following resources from the **Bundle** (depending on the Kubernetes distribution):
 - a. rke2-plan-agent/k3s-plan-agent - instructs **SUC** on how to do a Kubernetes upgrade on cluster **agent** nodes. Will **not** be interpreted if the cluster consists only from *control-plane* nodes.
 - b. rke2-plan-control-plane/k3s-plan-control-plane - instructs **SUC** on how to do a Kubernetes upgrade on cluster **control-plane** nodes.



Note

The above **SUC Plans** will be deployed in the `cattle-system` namespace of each downstream cluster.

6. On the downstream cluster, **SUC** picks up the newly deployed **SUC Plans** and deploys an **Update Pod** on each node that matches the **node selector** defined in the **SUC Plan**. For information how to monitor the **SUC Plan Pod**, refer to [Section 25.2.2.2, "Monitor SUC Plans"](#).
7. Depending on which **SUC Plans** you have deployed, the **Update Pod** will run either a `rke2-upgrade` (<https://hub.docker.com/r/rancher/rke2-upgrade/tags>) or a `k3s-upgrade` (<https://hub.docker.com/r/rancher/k3s-upgrade/tags>) image and will execute the following workflow on **each** cluster node:
 - a. `Cordon` (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_cordon/) cluster node - to ensure that no pods are scheduled accidentally on this node while it is being upgraded, we mark it as `unschedulable`.
 - b. Replace the `rke2/k3s` binary that is installed on the node OS with the binary shipped by the `rke2-upgrade/k3s-upgrade` image that the Pod is currently running.
 - c. Kill the `rke2/k3s` process that is running on the node OS - this instructs the **supervisor** to automatically restart the `rke2/k3s` process using the new version.
 - d. `Uncordon` (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_uncordon/) cluster node - after the successful Kubernetes distribution upgrade, the node is again marked as `schedulable`.



Note

For further information regarding how the `rke2-upgrade` and `k3s-upgrade` images work, see the `rke2-upgrade` (<https://github.com/rancher/rke2-upgrade>) and `k3s-upgrade` (<https://github.com/k3s-io/k3s-upgrade>) upstream projects.

With the above steps executed, the Kubernetes version of each cluster node should have been upgraded to the desired Edge compatible [release](https://github.com/suse-edge/fleet-examples/releases).

25.4.4 Kubernetes version upgrade - SUC Plan deployment

25.4.4.1 SUC Plan deployment - GitRepo resource

A **GitRepo** resource, that ships the needed Kubernetes upgrade SUC Plans, can be deployed in one of the following ways:

1. Through the Rancher UI - *Section 25.4.4.1.1, "GitRepo creation - Rancher UI"* (when Rancher is available).
2. By manually deploying (*Section 25.4.4.1.2, "GitRepo creation - manual"*) the resource to your management cluster.

Once deployed, to monitor the Kubernetes upgrade process of the nodes of your targeted cluster, refer to the *Section 25.2.2.2, "Monitor SUC Plans"* documentation.

25.4.4.1.1 GitRepo creation - Rancher UI

1. In the upper left corner, # → **Continuous Delivery**
2. Go to **Git Repos** → **Add Repository**

If you use the suse-edge/fleet-examples repository:

1. **Repository URL** - <https://github.com/suse-edge/fleet-examples.git>
2. **Watch** → **Revision** - choose a [release \(https://github.com/suse-edge/fleet-examples/releases\)](https://github.com/suse-edge/fleet-examples/releases) tag for the suse-edge/fleet-examples repository that you wish to use
3. Under **Paths** add the path to the Kubernetes distribution upgrade Fleets as seen in the release tag:
 - a. For RKE2 - [fleets/day2/system-upgrade-controller-plans/rke2-upgrade](#)
 - b. For K3s - [fleets/day2/system-upgrade-controller-plans/k3s-upgrade](#)
4. Select **Next** to move to the **target** configuration section. **Only select clusters for which you wish to upgrade the desired Kubernetes distribution**
5. **Create**

Alternatively, if you decide to use your own repository to host these files, you would need to provide your repo data above.

25.4.4.1.2 GitRepo creation - manual

1. Choose the desired Edge [release](https://github.com/suse-edge/fleet-examples/releases) (<https://github.com/suse-edge/fleet-examples/releases>) tag that you wish to apply the Kubernetes **SUC upgrade Plans** from (referenced below as `#{REVISION}`).

2. Pull the **GitRepo** resource:

- For **RKE2** clusters:

```
curl -o rke2-upgrade-gitrepo.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/#{REVISION}/gitrepos/day2/rke2-upgrade-gitrepo.yaml
```

- For **K3s** clusters:

```
curl -o k3s-upgrade-gitrepo.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/#{REVISION}/gitrepos/day2/k3s-upgrade-gitrepo.yaml
```

3. Edit the **GitRepo** configuration, under `spec.targets` specify your desired target list. By default the **GitRepo** resources from the [suse-edge/fleet-examples](#) are **NOT** mapped to any down stream clusters.

- To match all clusters change the default **GitRepo target** to:

```
spec:
  targets:
  - clusterSelector: {}
```

- Alternatively, if you want a more granular cluster selection see [Mapping to Downstream Clusters](https://fleet.rancher.io/gitrepo-targets) (<https://fleet.rancher.io/gitrepo-targets>)

4. Apply the **GitRepo** resources to your management cluster:

```
# RKE2
kubectl apply -f rke2-upgrade-gitrepo.yaml

# K3s
```

```
kubectl apply -f k3s-upgrade-gitrepo.yaml
```

5. View the created **GitRepo** resource under the `fleet-default` namespace:

```
# 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  release-3.0.1  0/0
rke2-upgrade        https://github.com/suse-edge/fleet-examples.git  release-3.0.1  0/0
```

25.4.4.2 SUC Plan deployment - Bundle resource

A **Bundle** resource, that ships the needed Kubernetes upgrade SUC Plans, can be deployed in one of the following ways:

1. Through the Rancher UI - *Section 25.4.4.2.1, "Bundle creation - Rancher UI"* (when Rancher is available).
2. By manually deploying (*Section 25.4.4.2.2, "Bundle creation - manual"*) the resource to your management cluster.

Once deployed, to monitor the Kubernetes upgrade process of the nodes of your targeted cluster, refer to the *Section 25.2.2.2, "Monitor SUC Plans"* documentation.

25.4.4.2.1 Bundle creation - Rancher UI

1. In the upper left corner, click # → **Continuous Delivery**
2. Go to **Advanced > Bundles**
3. Select **Create from YAML**

4. From here you can create the Bundle in one of the following ways:

a. By manually copying the **Bundle** content to the **Create from YAML** page. Content can be retrieved:

i. For RKE2 - <https://raw.githubusercontent.com/suse-edge/fleet-examples/{REVISION}/bundles/day2/system-upgrade-controller-plans/rke2-upgrade/plan-bundle.yaml>

ii. For K3s - <https://raw.githubusercontent.com/suse-edge/fleet-examples/{REVISION}/bundles/day2/system-upgrade-controller-plans/k3s-upgrade/plan-bundle.yaml>

b. By cloning the [suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples) (<https://github.com/suse-edge/fleet-examples>) repository to the desired [release](https://github.com/suse-edge/fleet-examples/releases) (<https://github.com/suse-edge/fleet-examples/releases>) tag and selecting the **Read from File** option in the **Create from YAML** page. From there, navigate to the bundle that you need (</bundles/day2/system-upgrade-controller-plans/rke2-upgrade/plan-bundle.yaml> for RKE2 and </bundles/day2/system-upgrade-controller-plans/k3s-upgrade/plan-bundle.yaml> for K3s). This will auto-populate the **Create from YAML** page with the Bundle content

5. Change the **target** clusters for the Bundle:

- To match all downstream clusters change the default Bundle `.spec.targets` to:

```
spec:
  targets:
  - clusterSelector: {}
```

- For a more granular downstream cluster mappings, see [Mapping to Downstream Clusters](https://fleet.rancher.io/gitrepo-targets) (<https://fleet.rancher.io/gitrepo-targets>).

6. **Create**

25.4.4.2.2 Bundle creation - manual

1. Choose the desired Edge [release](https://github.com/suse-edge/fleet-examples/releases) (<https://github.com/suse-edge/fleet-examples/releases>)[↗] tag that you wish to apply the Kubernetes **SUC upgrade Plans** from (referenced below as `${REVISION}`).

2. Pull the **Bundle** resources:

- For **RKE2** clusters:

```
curl -o rke2-plan-bundle.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/${REVISION}/bundles/day2/system-upgrade-controller-plans/rke2-upgrade/plan-bundle.yaml
```

- For **K3s** clusters:

```
curl -o k3s-plan-bundle.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/${REVISION}/bundles/day2/system-upgrade-controller-plans/k3s-upgrade/plan-bundle.yaml
```

3. Edit the **Bundle target** configurations, under `spec.targets` provide your desired target list. By default the **Bundle** resources from the `suse-edge/fleet-examples` are **NOT** mapped to any down stream clusters.

- To match all clusters change the default **Bundle target** to:

```
spec:
  targets:
  - clusterSelector: {}
```

- Alternatively, if you want a more granular cluster selection see [Mapping to Downstream Clusters](https://fleet.rancher.io/gitrepo-targets) (<https://fleet.rancher.io/gitrepo-targets>)[↗]

4. Apply the **Bundle** resources to your management cluster:

```
# For RKE2
kubectl apply -f rke2-plan-bundle.yaml

# For K3s
kubectl apply -f k3s-plan-bundle.yaml
```

5. View the created **Bundle** resource under the `fleet-default` namespace:

```
# 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
```

25.4.4.3 SUC Plan deployment - third-party GitOps workflow

There might be use-cases where users would like to incorporate the Kubernetes upgrade resources to their own third-party GitOps workflow (e.g. [Flux](#)).

To get the upgrade resources that you need, first determine the Edge [release](https://github.com/suse-edge/fleet-examples/releases) (<https://github.com/suse-edge/fleet-examples/releases>) tag of the [suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples) (<https://github.com/suse-edge/fleet-examples>) repository that you would like to use.

After that, the resources can be found at:

- For a RKE2 cluster upgrade:
 - For control-plane nodes - [fleets/day2/system-upgrade-controller-plans/rke2-upgrade/plan-control-plane.yaml](#)
 - For agent nodes - [fleets/day2/system-upgrade-controller-plans/rke2-upgrade/plan-agent.yaml](#)
- For a K3s cluster upgrade:
 - For control-plane nodes - [fleets/day2/system-upgrade-controller-plans/k3s-upgrade/plan-control-plane.yaml](#)
 - For agent nodes - [fleets/day2/system-upgrade-controller-plans/k3s-upgrade/plan-agent.yaml](#)



Important

These Plan resources are interpreted by the system-upgrade-controller and should be deployed on each downstream cluster that you wish to upgrade. For information on how to deploy the system-upgrade-controller, see [Section 25.2.1.3, "Deploying system-upgrade-controller when using a third-party GitOps workflow"](#).

To better understand how your GitOps workflow can be used to deploy the **SUC Plans** for Kubernetes version upgrade, it can be beneficial to take a look at the overview ([Section 25.4.3.1, "Overview"](#)) of the update procedure using Fleet.

25.5 Helm chart upgrade



Note

The below sections focus on using Fleet functionalities to achieve a Helm chart update. Users adopting a third-party GitOps workflow, should take the configurations for their desired helm chart from its `fleet.yaml` located at `fleets/day2/chart-templates/<chart-name>`. **Make sure you are retrieving the chart data from a valid "Day 2" Edge release (<https://github.com/suse-edge/fleet-examples/releases>)**.

25.5.1 Components

Apart from the default Day 2 components ([Section 25.1.1, "Components"](#)), no other custom components are needed for this operation.

25.5.2 Preparation for air-gapped environments

25.5.2.1 Ensure that you have access to your Helm chart's upgrade `fleet.yaml` file

Host the needed resources on a local git server that is accessible by your management cluster.

25.5.2.2 Find the required assets for your Edge release version

1. Go to the Day 2 release (<https://github.com/suse-edge/fleet-examples/releases>) page and find the Edge 3.X.Y release that you want to upgrade your chart to and click **Assets**.
2. From the release's **Assets** section, download the following files, which are required for an air-gapped upgrade of a SUSE supported helm chart:

Release File	Description
<i>edge-save-images.sh</i>	This script pulls the images in the <code>edge-release-images.txt</code> file and saves them to a '.tar.gz' archive that can then be used in your air-gapped environment.
<i>edge-save-oci-artefacts.sh</i>	This script pulls the SUSE OCI chart artefacts in the <code>edge-release-helm-oci-artefacts.txt</code> file and creates a '.tar.gz' archive of a directory containing all other chart OCI archives.
<i>edge-load-images.sh</i>	This script loads the images in the '.tar.gz' archive generated by <code>edge-save-images.sh</code> , retags them and pushes them to your private registry.
<i>edge-load-oci-artefacts.sh</i>	This script takes a directory containing '.tgz' SUSE OCI charts and loads all OCI charts to your private registry. The directory is retrieved from the '.tar.gz' archive that the <code>edge-save-oci-artefacts.sh</code> script has generated.
<i>edge-release-helm-oci-artefacts.txt</i>	This file contains a list of OCI artefacts for the SUSE Edge release Helm charts.
<i>edge-release-images.txt</i>	This file contains a list of images needed by the Edge release Helm charts.

25.5.2.3 Create the SUSE Edge release images archive

On a machine with internet access:

1. Make `edge-save-images.sh` executable:

```
chmod +x edge-save-images.sh
```

2. Use `edge-save-images.sh` script to create a *Docker* importable `.tar.gz` archive:

```
./edge-save-images.sh --source-registry registry.suse.com
```

3. This will create a ready to load `edge-images.tar.gz` (unless you have specified the `-i|--images` option) archive with the needed images.
4. Copy this archive to your **air-gapped** machine

```
scp edge-images.tar.gz <user>@<machine_ip>:/path
```

25.5.2.4 Create a SUSE Edge Helm chart OCI images archive

On a machine with internet access:

1. Make `edge-save-oci-artefacts.sh` executable:

```
chmod +x edge-save-oci-artefacts.sh
```

2. Use `edge-save-oci-artefacts.sh` script to create a `.tar.gz` archive of all SUSE Edge Helm chart OCI images:

```
./edge-save-oci-artefacts.sh --source-registry registry.suse.com
```

3. This will create a `oci-artefacts.tar.gz` archive containing all SUSE Edge Helm chart OCI images
4. Copy this archive to your **air-gapped** machine

```
scp oci-artefacts.tar.gz <user>@<machine_ip>:/path
```


25.5.2.5 Load SUSE Edge release images to your air-gapped machine

On your air-gapped machine:

1. Log into your private registry (if required):

```
podman login <REGISTRY.YOURDOMAIN.COM:PORT>
```

2. Make `edge-load-images.sh` executable:

```
chmod +x edge-load-images.sh
```

3. Use `edge-load-images.sh` to load the images from the **copied** `edge-images.tar.gz` archive, retag them and push them to your private registry:

```
./edge-load-images.sh --source-registry registry.suse.com --registry  
<REGISTRY.YOURDOMAIN.COM:PORT> --images edge-images.tar.gz
```

25.5.2.6 Load SUSE Edge Helm chart OCI images to your air-gapped machine

On your air-gapped machine:

1. Log into your private registry (if required):

```
podman login <REGISTRY.YOURDOMAIN.COM:PORT>
```

2. Make `edge-load-oci-artefacts.sh` executable:

```
chmod +x edge-load-oci-artefacts.sh
```

3. Untar the copied `oci-artefacts.tar.gz` archive:

```
tar -xvf oci-artefacts.tar.gz
```

4. This will produce a directory with the naming template `edge-release-oci-tgz-<date>`
5. Pass this directory to the `edge-load-oci-artefacts.sh` script to load the SUSE Edge helm chart OCI images to your private registry:



Note

This script assumes the `helm` CLI has been pre-installed on your environment. For Helm installation instructions, see [Installing Helm \(https://helm.sh/docs/intro/install/\)](https://helm.sh/docs/intro/install/).

```
./edge-load-oci-artefacts.sh --archive-directory edge-release-oci-tgz-<date> --registry <REGISTRY.YOURDOMAIN.COM:PORT> --source-registry registry.suse.com
```

25.5.2.7 Create registry mirrors pointing to your private registry for your Kubernetes distribution

For RKE2, see [Containerd Registry Configuration \(https://docs.rke2.io/install/containerd_registry_configuration\)](https://docs.rke2.io/install/containerd_registry_configuration)

For K3s, see [Embedded Registry Mirror \(https://docs.k3s.io/installation/registry-mirror\)](https://docs.k3s.io/installation/registry-mirror)

25.5.3 Upgrade procedure



Note

The below upgrade procedure utilises Rancher's Fleet ([Chapter 6, Fleet](#)) functionality. Users using a third-party GitOps workflow should retrieve the chart versions supported by each Edge release from the [Section 33.1, "Abstract"](#) and populate these versions to their third-party GitOps workflow.

This section focuses on the following Helm upgrade procedure use-cases:

1. I have a new cluster and would like to deploy and manage a SUSE Helm chart (*Section 25.5.3.1, "I have a new cluster and would like to deploy and manage a SUSE Helm chart"*)
2. I would like to upgrade a Fleet managed Helm chart (*Section 25.5.3.2, "I would like to upgrade a Fleet managed Helm chart"*)
3. I would like to upgrade an EIB created Helm chart (*Section 25.5.3.3, "I would like to upgrade an EIB created Helm chart"*)


Important

Manually deployed Helm charts cannot be reliably upgraded. We suggest to redeploy the helm chart using the *Section 25.5.3.1, "I have a new cluster and would like to deploy and manage a SUSE Helm chart"* method.

25.5.3.1 I have a new cluster and would like to deploy and manage a SUSE Helm chart

For users that want to manage their Helm chart lifecycle through Fleet.

25.5.3.1.1 Prepare your Fleet resources

1. Acquire the Chart's Fleet resources from the Edge [release \(https://github.com/suse-edge/fleet-examples/releases\)](https://github.com/suse-edge/fleet-examples/releases)  tag that you wish to use
 - a. From the selected Edge release tag revision, navigate to the Helm chart fleet - [fleets/day2/chart-templates/<chart>](#)
 - b. Copy the chart Fleet directory to the Git repository that you will be using for your GitOps workflow

- c. **Optionally**, if the Helm chart requires configurations to its **values**, edit the `.helm.values` configuration inside the `fleet.yaml` file of the copied directory
- d. **Optionally**, there may be use-cases where you need to add additional resources to your chart's fleet so that it can better fit your environment. For information on how to enhance your Fleet directory, see [Git Repository Contents \(https://fleet.rancher.io/gitrepo-content\)](https://fleet.rancher.io/gitrepo-content) ↗

An **example** for the `longhorn` helm chart would look like:

- User Git repository structure:

```
<user_repository_root>
├── longhorn
│   └── fleet.yaml
```

- `fleet.yaml` content populated with user `longhorn` data:

```
defaultNamespace: longhorn-system

helm:
  releaseName: "longhorn"
  chart: "longhorn"
  repo: "https://charts.longhorn.io"
  version: "1.6.1"
  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"}

```



Note

These are just example values that are used to illustrate custom configurations over the [longhorn chart](#). They should **NOT** be treated as deployment guidelines for the [longhorn chart](#).

25.5.3.1.2 Create the GitRepo

After populating your repository with the chart's Fleet resources, you must create a [GitRepo](https://fleet.rancher.io/ref-gitrepo) resource. This resource will hold information on how to access your chart's Fleet resources and to which clusters it needs to apply those resources.

The [GitRepo](#) resource can be created through the Rancher UI, or by manually deploying the resource to the [management cluster](#).

For information on how to create and deploy the GitRepo resource **manually**, see [Creating a Deployment](https://fleet.rancher.io/tut-deployment).

To create a [GitRepo](#) resource through the **Rancher UI**, see [Accessing Fleet in the Rancher UI](https://ranchermanager.docs.rancher.com/v2.8/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui).

*Example **longhorn** [GitRepo](#) resource for **manual** deployment:*

```

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
repo: <user_repository_url>
targets:
# Match all clusters
- clusterSelector: {}
```

25.5.3.1.3 Managing the deployed Helm chart

Once deployed with Fleet, for Helm chart upgrades, see [Section 25.5.3.2, “I would like to upgrade a Fleet managed Helm chart”](#).

25.5.3.2 I would like to upgrade a Fleet managed Helm chart

1. Determine the version to which you need to upgrade your chart so that it is compatible with an Edge 3.X.Y release. Helm chart version per Edge release can be viewed from the [Section 33.1, “Abstract”](#).
2. In your Fleet monitored Git repository, edit the Helm chart’s `fleet.yaml` file with the correct chart **version** and **repository** from the [Section 33.1, “Abstract”](#).
3. After committing and pushing the changes to your repository, this will trigger an upgrade of the desired Helm chart

25.5.3.3 I would like to upgrade an EIB created Helm chart



Note

This section assumes that you have deployed the system-upgrade-controller (SUC) beforehand, if you have not done so, or are unsure why you need it, see the default Day 2 components ([Section 25.1.1, “Components”](#)) list.

EIB deploys Helm charts by utilizing the auto-deploy manifests functionality of rke2 (<https://docs.rke2.io/advanced#auto-deploying-manifests>)[↗]/k3s (<https://docs.k3s.io/installation/packaged-components#auto-deploying-manifests-addons>)[↗]. It creates a HelmChart ([283](https://</p></div><div data-bbox=)

github.com/k3s-io/helm-controller#helm-controller resource definition manifest under the `/var/lib/rancher/<rke2/k3s>/server/manifests` location of the initialiser node and lets `rke2/k3s` pick it up and auto-deploy it in the cluster.

From a Day 2 point of view this would mean that any upgrades of the Helm chart need to happen by editing the `HelMChart` manifest file of the specific chart. To automate this process for multiple clusters, this section uses **SUC Plans**.

Below you can find information on:

- The general overview ([Section 25.5.3.3.1, “Overview”](#)) of the helm chart upgrade workflow.
- The necessary upgrade steps ([Section 25.5.3.3.2, “Upgrade Steps”](#)) needed for a successful helm chart upgrade.
- An example ([Section 25.5.3.3.3, “Example”](#)) showcasing a [Longhorn](https://longhorn.io) chart upgrade using the explained method.
- How to use the upgrade process with a different GitOps tool ([Section 25.5.3.3.4, “Helm chart upgrade using a third-party GitOps tool”](#)).

25.5.3.3.1 Overview

This section is meant to give a high overview of the workflow that the user goes through in order to upgrade one or multiple Helm charts. For a detailed explanation of the steps needed for a Helm chart upgrade, see [Section 25.5.3.3.2, “Upgrade Steps”](#).

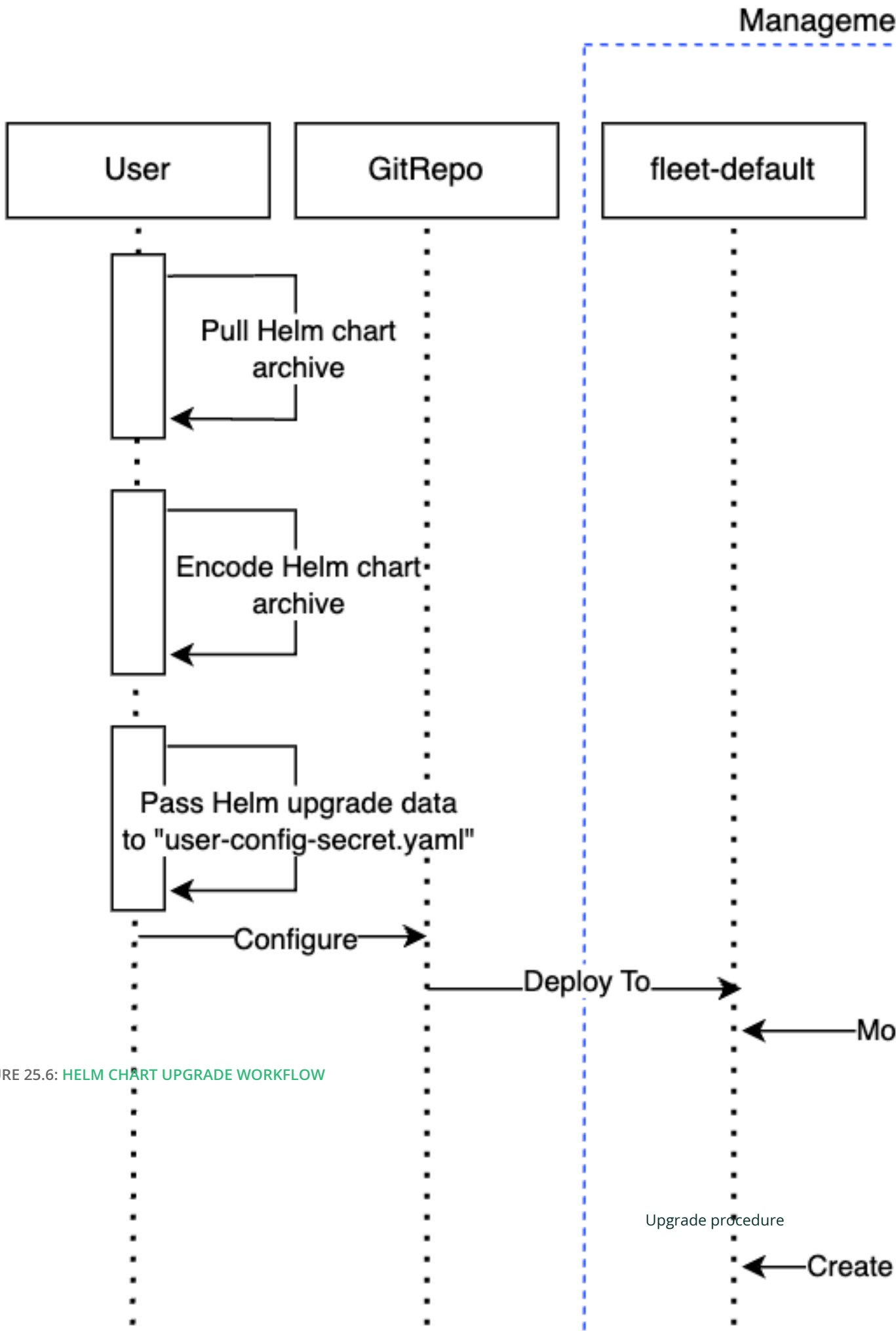


FIGURE 25.6: HELM CHART UPGRADE WORKFLOW

1. The workflow begins with the user [pulling \(https://helm.sh/docs/helm/helm_pull/\)](https://helm.sh/docs/helm/helm_pull/) the new Helm chart archive(s) that he wishes to upgrade his chart(s) to.
2. The archive(s) should then be *encoded* and passed as configuration to the `eib-chart-upgrade-user-data.yaml` file that is located under the fleet directory for the related SUC Plan. This is further explained in the upgrade steps ([Section 25.5.3.3.2, "Upgrade Steps"](#)) section.
3. The user then proceeds to configure and deploy a `GitRepo` resource that will ship all the needed resources (SUC Plan, secrets, etc.) to the downstream clusters.
 - a. The resource is deployed on the `management cluster` under the `fleet-default` namespace.
4. Fleet ([Chapter 6, Fleet](#)) detects the deployed resource and deploys all the configured resources to the specified downstream clusters. Deployed resources include:
 - a. The `eib-chart-upgrade` SUC Plan that will be used by SUC to create an **Upgrade Pod** on each node.
 - b. The `eib-chart-upgrade-script` Secret that ships the `upgrade script` that the **Upgrade Pod** will use to upgrade the `HelmChart` manifests on the initialiser node.
 - c. The `eib-chart-upgrade-user-data` Secret that ships the chart data that the `upgrade script` will use in order to understand which chart manifests it needs to upgrade.
5. Once the `eib-chart-upgrade` SUC Plan has been deployed, the SUC picks it up and creates a Job which deploys the **Upgrade Pod**.
6. Once deployed, the **Upgrade Pod** mounts the `eib-chart-upgrade-script` and `eib-chart-upgrade-user-data` Secrets and executes the `upgrade script` that is shipped by the `eib-chart-upgrade-script` Secret.
7. The `upgrade script` does the following:
 - a. Determine whether the Pod that the script is running on has been deployed on the `initialiser` node. The `initialiser` node is the node that is hosting the `Helm-Chart` manifests. For a single-node cluster it is the single control-plane node. For HA clusters it is the node that you have marked as `initializer` when creating the cluster in EIB. If you have not specified the `initializer` property, then the

first node from the `nodes` list is marked as `initializer`. For more information, see the [upstream \(https://github.com/suse-edge/edge-image-builder/blob/main/docs/building-images.md#kubernetes\)](https://github.com/suse-edge/edge-image-builder/blob/main/docs/building-images.md#kubernetes) documentation for EIB.



Note

If the `upgrade script` is running on a non-initialiser node, it immediately finishes its execution and does not go through the steps below.

- b. Backup the manifests that will be edited in order to ensure disaster recover.




Note

By default backups of the manifests are stored under the `/tmp/eib-helm-chart-upgrade-<date>` directory. If you wish to use a custom location you can pass the `MANIFEST_BACKUP_DIR` environment variable to the Helm chart upgrade SUC Plan (example in the Plan).

- c. Edit the `HelmChart` manifests. As of this version, the following properties are changed in order to trigger a chart upgrade:
 - i. The content of the `chartContent` property is replaced with the encoded archive provided in the `eib-chart-upgrade-user-data` Secret.
 - ii. The value of the `version` property is replaced with the version provided in the `eib-chart-upgrade-user-data` Secret.
8. After the successful execution of the `upgrade script`, the Helm integration for [RKE2 \(https://docs.rke2.io/helm\)](https://docs.rke2.io/helm) / [K3s \(https://docs.k3s.io/helm\)](https://docs.k3s.io/helm) will pickup the change and automatically trigger an upgrade on the Helm chart.

25.5.3.3.2 Upgrade Steps

1. Determine an Edge [release tag \(https://github.com/suse-edge/fleet-examples/releases\)](https://github.com/suse-edge/fleet-examples/releases) from which you wish to copy the Helm chart upgrade logic.
2. Copy the `fleets/day2/system-upgrade-controller-plans/eib-chart-upgrade` fleet to the repository that your Fleet will be using to do GitOps from.

3. Pull (https://helm.sh/docs/helm/helm_pull/)  the Helm chart archive that you wish to upgrade to:

```
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. Encode the chart archive that you pulled:

```
# Encode the archive and disable line wrapping
base64 -w 0 <chart-archive>.tgz
```

5. Configure the `eib-chart-upgrade-user-data.yaml` secret located under the `eib-chart-upgrade` fleet that you copied from step (2):

- a. The secret ships a file called `chart_upgrade_data.txt`. This file holds the chart upgrade data that the `upgrade` script will use to know which charts need to be upgraded. The file expects one-line per chart entries in the following format "`<name> | <version> | <base64_encoded_archive>`":
 - i. `name` - is the name of the helm chart as seen in the `kubernetes.helm.chart-s.name[]` property of the EIB definition file.
 - ii. `version` - should hold the new version of the Helm chart. During the upgrade this value will be used to replace the old `version` value of the `HelmChart` manifest.
 - iii. `base64_encoded_archive` - pass the output of the `base64 -w 0 <chart-archive>.tgz` here. During upgrade this value will be used to replace the old `chartContent` value of the `HelmChart` manifest.



Note

The `<name> | <version> | <base64_encoded_archive>` line should be removed from the file before you start adding your data. It serves as an example of where and how you should configure your chart data.

6. Configure a `GitRepo` resource that will be shipping your chart upgrade fleet. For more information on what a `GitRepo` is, see [GitRepo Resource \(https://fleet.rancher.io/ref-gitrepo\)](https://fleet.rancher.io/ref-gitrepo) .

- a. Configure GitRepo through the Rancher UI:
 - i. In the upper left corner, # → **Continuous Delivery**
 - ii. Go to **Git Repos** → **Add Repository**
 - iii. Here pass your **repository data** and **path** to your chart upgrade fleet
 - iv. Select **Next** and specify the **target** clusters of which you want to upgrade the configured charts
 - v. **Create**
- b. If Rancher is not available on your setup, you can configure a GitRepo manually on your management cluster:
 - i. Populate the following template with your data:

```
apiVersion: fleet.cattle.io/v1alpha1
kind: GitRepo
metadata:
  name: CHANGE_ME
  namespace: fleet-default
spec:
  # if running from a tag
  # revision: CHANGE_ME
  # if running from a branch
  # branch: CHANGE_ME
  paths:
    # path to your chart upgrade fleet relative to your repository
    - CHANGE_ME
  # your repository URL
  repo: CHANGE_ME
  targets:
    # Select target clusters
    - clusterSelector: CHANGE_ME
  # To match all clusters:
  # - clusterSelector: {}
```

For more information on how to setup and deploy a GitRepo resource, see [GitRepo Resource \(https://fleet.rancher.io/ref-gitrepo\)](https://fleet.rancher.io/ref-gitrepo) and [Create a GitRepo Resource \(https://fleet.rancher.io/gitrepo-add\)](https://fleet.rancher.io/gitrepo-add).

For information on how to match **target** clusters on a more granular level, see [Mapping to Downstream Clusters \(https://fleet.rancher.io/gitrepo-targets\)](https://fleet.rancher.io/gitrepo-targets).

- ii. Deploy the configured `GitRepo` resource to the `fleet-default` namespace of the `management` cluster.

Executing this steps should result in a successfully created `GitRepo` resource. It will then be picked up by Fleet and a Bundle will be created. This Bundle will hold the **raw** Kubernetes resources that the `GitRepo` has configured under its fleet directory.

Fleet will then deploy all the Kubernetes resources from the Bundle to the specified downstream clusters. One of this resources will be a SUC Plan that will trigger the chart upgrade. For a full list of the resources that will be deployed and the workflow of the upgrade process, refer to the overview ([Section 25.5.3.3.1, "Overview"](#)) section.

To track the upgrade process itself, refer to the Monitor SUC Plans ([Section 25.2.2.2, "Monitor SUC Plans"](#)) section.

25.5.3.3.3 Example

The following section serves to provide a real life example to the [Section 25.5.3.3.2, "Upgrade Steps"](#) section.

I have the following two EIB deployed clusters:

- `longhorn-single-k3s` - single node K3s cluster
- `longhorn-ha-rke2` - HA RKE2 cluster

Both clusters are running [Longhorn \(https://longhorn.io\)](https://longhorn.io) and have been deployed through EIB, using the following image definition *snippet*:

```
kubernetes:
  # HA RKE2 cluster specific snippet
  # nodes:
  # - hostname: cp1rke2.example.com
  #   initializer: true
  #   type: server
  # - hostname: cp2rke2.example.com
  #   type: server
  # - hostname: cp3rke2.example.com
  #   type: server
  # - hostname: agent1rke2.example.com
  #   type: agent
```

```
# - hostname: agent2rke2.example.com
#   type: agent
# version depending on the distribution
version: v1.28.13+k3s1/v1.28.13+rke2r1
helm:
  charts:
    - name: longhorn
      repositoryName: longhorn
      targetNamespace: longhorn-system
      createNamespace: true
      version: 1.5.5
  repositories:
    - name: longhorn
      url: https://charts.longhorn.io
...
```

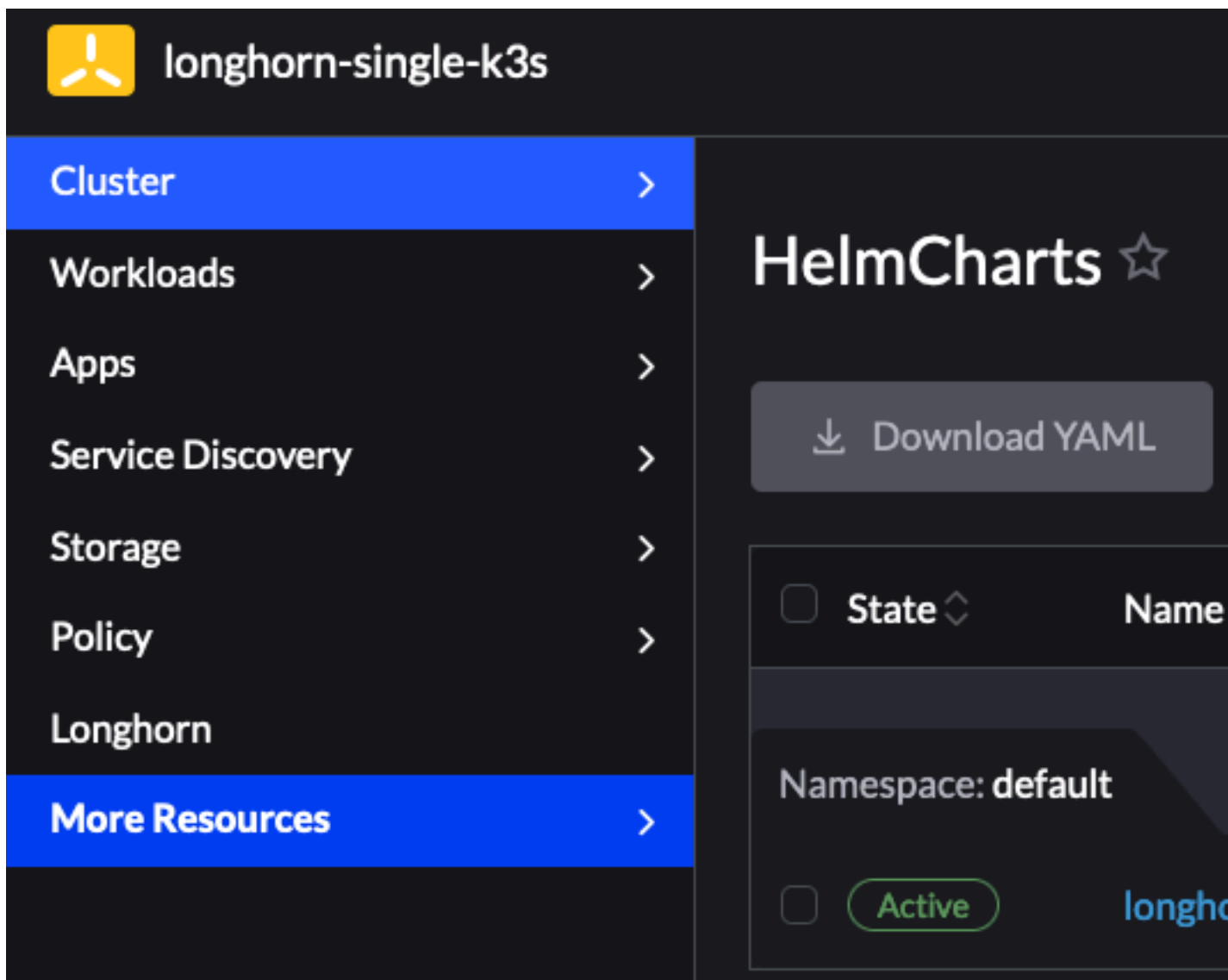


FIGURE 25.7: LONGHORN-SINGLE-K3S INSTALLED LONGHORN VERSION

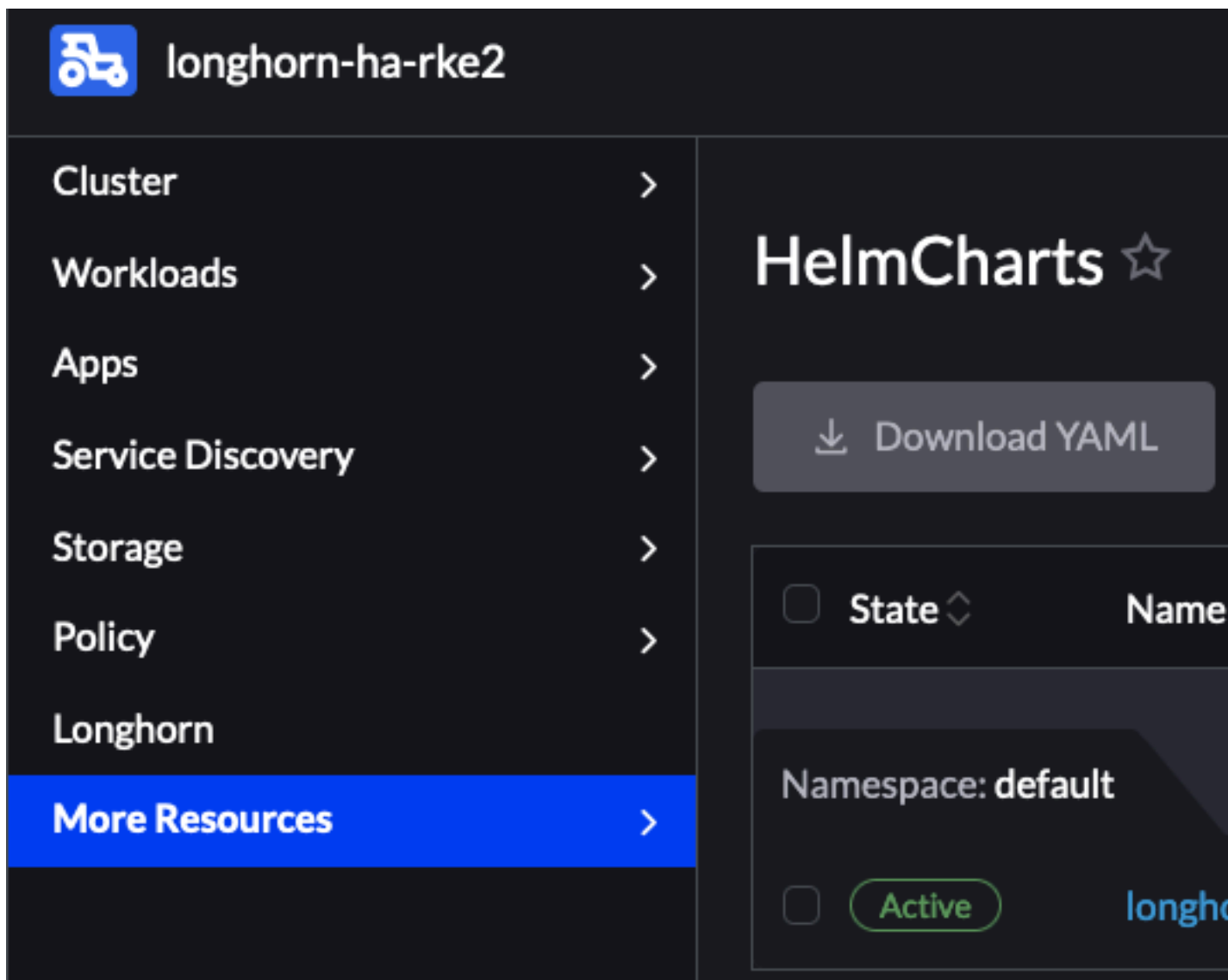


FIGURE 25.8: LONGHORN-HA-RKE2 INSTALLED LONGHORN VERSION

The problem with this is that currently `longhorn-single-k3s` and `longhorn-ha-rke2` are running with a Longhorn version that is not compatible with any Edge release.

We need to upgrade the chart on both clusters to a Edge supported Longhorn version.

To do this we follow these steps:

1. Determine the Edge [release tag](https://github.com/suse-edge/fleet-examples/releases) (<https://github.com/suse-edge/fleet-examples/releases>) from which we want to take the upgrade logic. For example, this example will use the `release-3.0.1` release tag for which the supported Longhorn version is `1.6.1`.
2. Clone the `release-3.0.1` release tag and copy the `fleets/day2/system-upgrade-controller-plans/eib-chart-upgrade` directory to our own repository.

For simplicity this section works from a branch of the [suse-edge/fleet-examples](#) repository, so the directory structure is the same, but you can place the [eib-chart-upgrade](#) fleet anywhere in your repository.

Directory structure example.

```
.
...
|-- fleets
|   |-- day2
|       |-- system-upgrade-controller-plans
|           |-- eib-chart-upgrade
|               |-- eib-chart-upgrade-script.yaml
|               |-- eib-chart-upgrade-user-data.yaml
|               |-- fleet.yaml
|               |-- plan.yaml
...
```

3. Add the Longhorn chart repository:

```
helm repo add longhorn https://charts.longhorn.io
```

4. Pull the Longhorn chart version [1.6.1](#):

```
helm pull longhorn/longhorn --version 1.6.1
```

This will pull the longhorn as an archive named [longhorn-1.6.1.tgz](#).

5. Encode the Longhorn archive:

```
base64 -w 0 longhorn-1.6.1.tgz
```

This will output a long one-line base64 encoded string of the archive.

6. Now we have all the needed data to configure the [eib-chart-upgrade-user-data.yaml](#) file. The file configuration should look like this:

```
apiVersion: v1
kind: Secret
metadata:
  name: eib-chart-upgrade-user-data
type: Opaque
stringData:
  # <name>|<version>|<base64_encoded_archive>
  chart_upgrade_data.txt: |
```

```
longhorn|1.6.1|H4sIFAAAAAAAA/ykAK2FIUjBjSE02THk5NWIzV...
```

- a. longhorn is the name of the chart as seen in my EIB definition file
- b. 1.6.1 is the version to which I want to upgrade the version property of the Longhorn HelmChart manifest
- c. H4sIFAAAAAAAA/ykAK2FIUjBjSE02THk5NWIzV... is a snippet of the encoded Longhorn 1.6.1 archive. **A snippet has been added here for better readability. You should always provide the full base64 encoded archive string here.**



Note

This example shows configuration for a single chart upgrade, but if your use-case requires to upgrade multiple charts on multiple clusters, you can append the additional chart data as seen below:


```
apiVersion: v1
kind: Secret
metadata:
  name: eib-chart-upgrade-user-data
type: Opaque
stringData:
  # <name>|<version>|<base64_encoded_archive>
  chart_upgrade_data.txt: |
    chartA|0.0.0|<chartA_base64_archive>
    chartB|0.0.0|<chartB_base64_archive>
    chartC|0.0.0|<chartC_base64_archive>
    ...
```

7. We also decided that we do not want to keep manifest backups at /tmp, so the following configuration was added to the plan.yaml file:

```
apiVersion: upgrade.cattle.io/v1
kind: Plan
metadata:
  name: eib-chart-upgrade
spec:
  ...
  upgrade:
    ...
    # For when you want to backup your chart
    # manifest data under a specific directory
```

```
#
envs:
- name: MANIFEST_BACKUP_DIR
  value: "/root"
```

This will ensure that manifest backups will be saved under the /root directory instead of /tmp.

8. Now that we have made all the needed configurations, what is left is to create the GitRepo resource. This example creates the GitRepo resource through the Rancher UI.
9. Following the steps described in the Upgrade Steps (*Section 25.5.3.3.2, "Upgrade Steps"*), we:
 - a. Named the GitRepo "longhorn-upgrade".
 - b. Passed the URL to the repository that will be used - <https://github.com/suse-edge/fleet-examples.git> 
 - c. Passed the branch of the repository - "doc-example"
 - d. Passed the path to the eib-chart-upgrade fleet as seen in the repo - fleets/day2/system-upgrade-controller-plans/eib-chart-upgrade
 - e. Selected the target clusters and created the resource

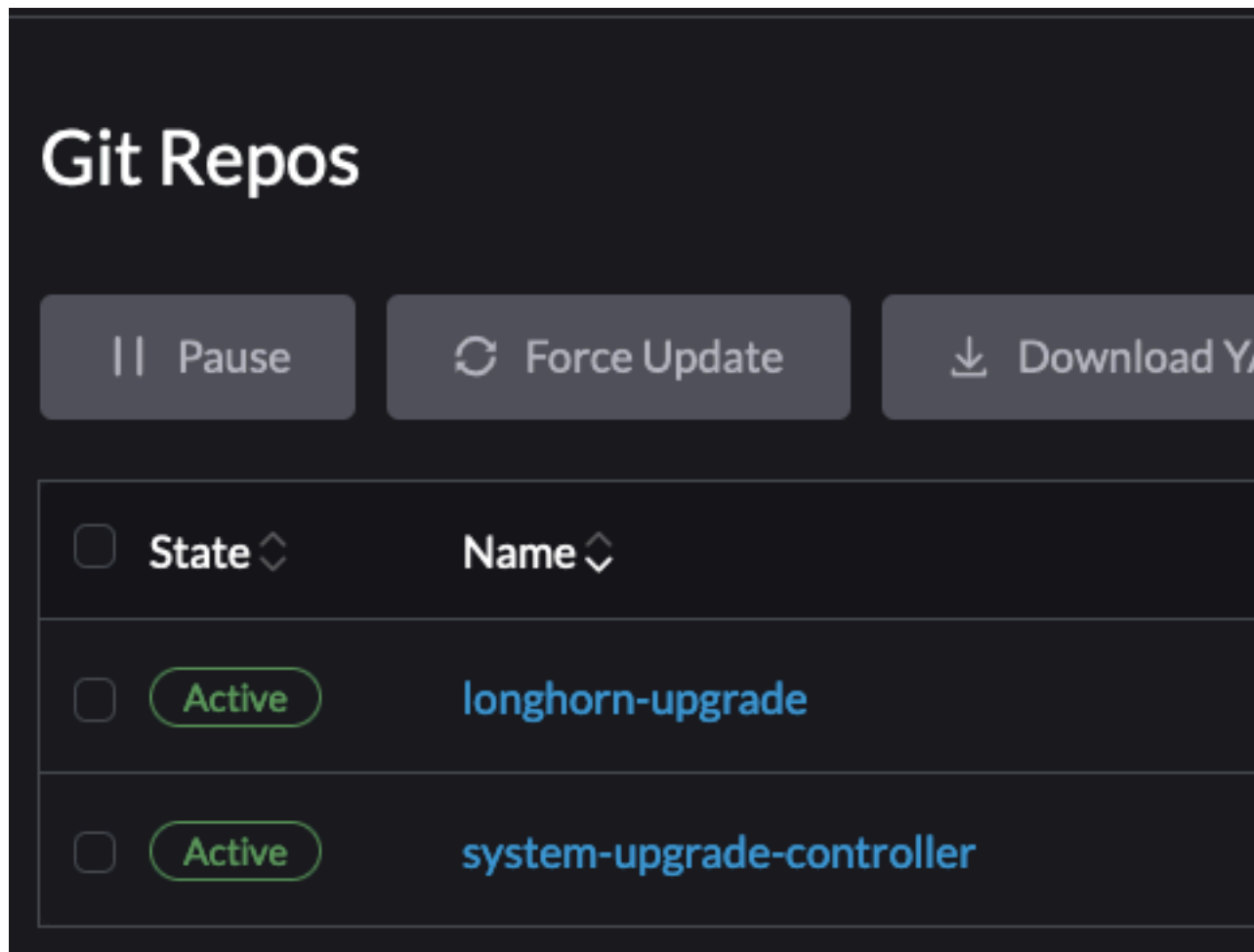


FIGURE 25.9: SUCCESSFULLY DEPLOYED SUC AND LONGHORN GITREPOS

Now we need to monitor the upgrade procedures on the clusters:

1. Check the status of the **Upgrade Pods**, following the directions from the SUC plan monitor ([Section 25.2.2.2, "Monitor SUC Plans"](#)) section.

- a. A successfully completed **Upgrade Pod** that has been working on an initialiser node should hold logs similar to:

```
apply-eib-chart-upgrade-on-cp1rke2-with-66c9bdaba8c533c
Fri, Jun 7 2024 5:34:26 pm Determining whether
2024-06-07T14:34:26
2024-06-07T14:34:26
2024-06-07T14:34:26
Fri, Jun 7 2024 5:34:26 pm Modifying the 'char
2024-06-07T14:34:26
2024-06-07T14:34:26
```

FIGURE 25.10: UPGRADE POD RUNNING ON AN INITIALISER NODE

- b. A successfully completed **Upgrade Pod** that has been working on a non-initialiser node should hold logs similar to:

```
apply-eib-chart-upgrade-on-cp2rke2-with-66c9bdaba8c533c
Fri, Jun 7 2024 5:34:21 pm Determining whether
2024-06-07T14:34:21
```

FIGURE 25.11: UPGRADE POD RUNNING ON A NON-INITIALISER NODE

2. After a successful **Upgrade Pod** completion, we would also need to wait and monitor for the pods that will be created by the helm controller. These pods will do the actual upgrade based on the file changes that the **Upgrade Pod** has done to the HelmChart manifest file.
 - a. In your cluster, go to **Workloads** → **Pods** and search for a pod that contains the longhorn string in the default namespace. This should produce a pod with the naming template helm-install-longhorn-*, view the logs of this pod.



longhorn-ha-rke2

Cluster >

Workloads 

CronJobs (⇌) 0

DaemonSets (⇌) 0

Deployments (⇌) 0

Jobs (⇌) 3

StatefulSets (⇌) 0

Pods (⇌) 3

Apps >

Service Discovery >

Storage >

Policy >

Longhorn

More Resources >

Pods

 Download YAML

State 

Namespace: default

Completed

Now that we have ensured that everything has completed successfully, we need to verify the version change:

1. On the clusters we need to go to **More Resources** → **Helm** → **HelmCharts** and in the default namespace search for the longhorn HelmChart resource:

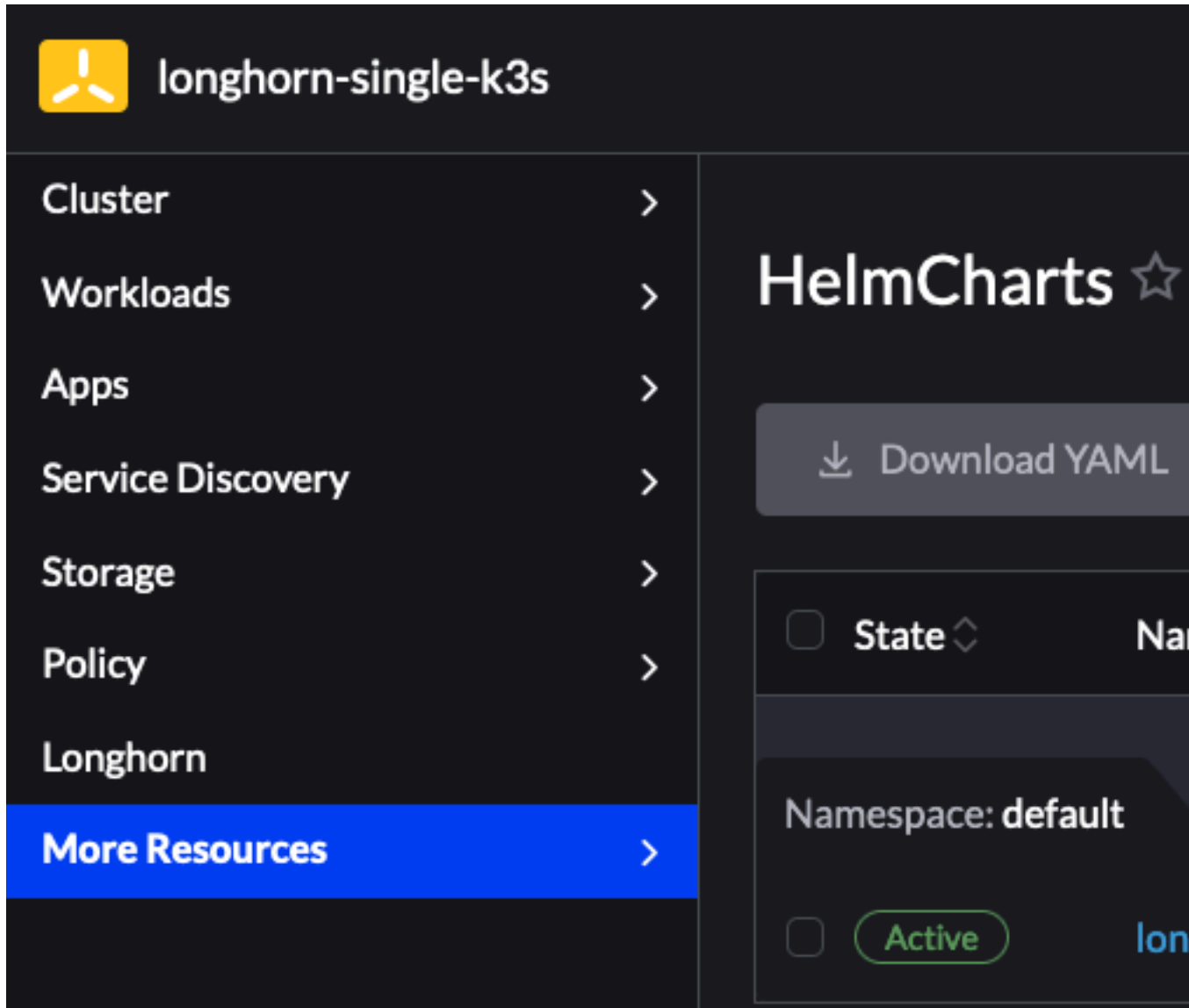


FIGURE 25.14: LONGHORN-SINGLE-K3S UPGRADED LONGHORN VERSION

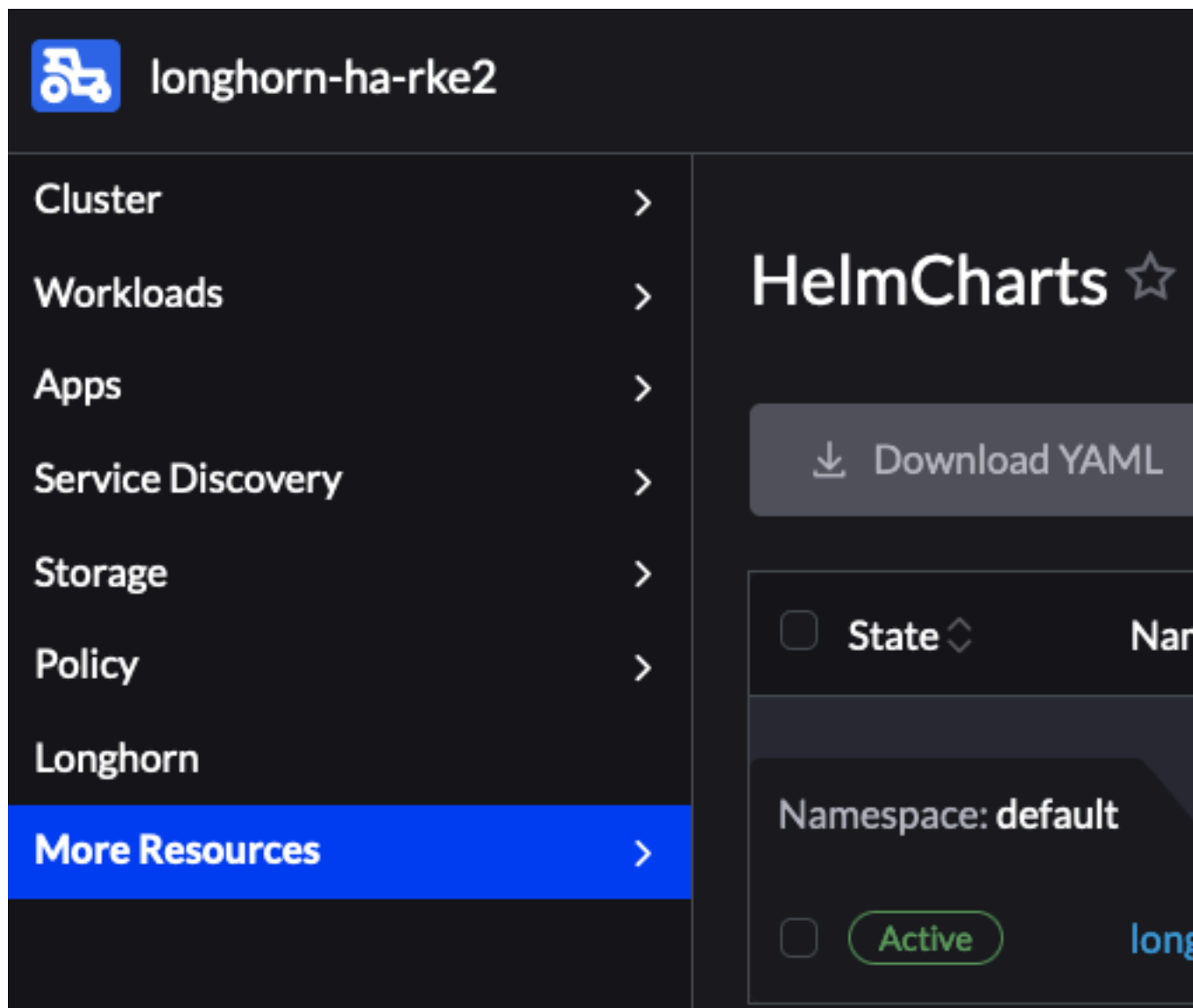


FIGURE 25.15: LONGHORN-HA-RKE2 UPGRADED LONGHORN VERSION

This ensures that the Longhorn helm chart has been successfully upgraded and concludes this example.

If for some reason we would like to revert to the previous chart version of Longhorn, the previous Longhorn manifest will be located under `/root/longhorn.yaml` on the initialiser node. This is true, because we have specified the MANIFEST_BACKUP_DIR in the SUC Plan.

25.5.3.3.4 Helm chart upgrade using a third-party GitOps tool

There might be use-cases where users would like to use this upgrade procedure with a GitOps workflow other than Fleet (e.g. [Flux](#)).

To get the resources related to EIB deployed Helm chart upgrades you need to first determine the Edge [release](https://github.com/suse-edge/fleet-examples/releases) (<https://github.com/suse-edge/fleet-examples/releases>) tag of the [suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples) (<https://github.com/suse-edge/fleet-examples.git>) repository that you would like to use.

After that, resources can be found at [fleets/day2/system-upgrade-controller-plans/eib-chart-upgrade](#), where:

- [plan.yaml](#) - system-upgrade-controller Plan related to the upgrade procedure.
- [eib-chart-upgrade-script.yaml](#) - Secret holding the [upgrade script](#) that is responsible for editing and upgrade the [HelmChart](#) manifest files.
- [eib-chart-upgrade-user-data.yaml](#) - Secret holding a file that is utilised by the [upgrade script](#); populated by the user with relevant chart upgrade data beforehand.

Important

These [Plan](#) resources are interpreted by the [system-upgrade-controller](#) and should be deployed on each downstream cluster that holds charts in need of an upgrade. For information on how to deploy the [system-upgrade-controller](#), see [Section 25.2.1.3, “Deploying system-upgrade-controller when using a third-party GitOps workflow”](#).

To better understand how your GitOps workflow can be used to deploy the **SUC Plans** for the upgrade process, it can be beneficial to take a look at the overview ([Section 25.5.3.3.1, “Overview”](#)) of the process using [Fleet](#).

VI Product Documentation

- 26 SUSE Adaptive Telco Infrastructure Platform (ATIP) **304**
- 27 Concept & Architecture **305**
- 28 Requirements & Assumptions **313**
- 29 Setting up the management cluster **317**
- 30 Telco features configuration **355**
- 31 Fully automated directed network provisioning **383**
- 32 Lifecycle actions **435**

Find the ATIP documentation [here](#)

26 SUSE Adaptive Telco Infrastructure Platform (ATIP)

SUSE Adaptive Telco Infrastructure Platform (ATIP) is a Telco-optimized edge computing platform that enables telecom companies to innovate and accelerate the modernization of their networks.

ATIP is a complete Telco cloud stack for hosting CNFs such as 5G Packet Core and Cloud RAN.

- Automates zero-touch rollout and lifecycle management of complex edge stack configurations at Telco scale.
- Continuously assures quality on Telco-grade hardware, using Telco-specific configurations and workloads.
- Consists of components that are purpose-built for the edge and hence have smaller footprint and higher performance per Watt.
- Maintains a flexible platform strategy with vendor-neutral APIs and 100% open source.

27 Concept & Architecture

SUSE ATIP is a platform designed for hosting modern, cloud native, Telco applications at scale from core to edge.

This page explains the architecture and components used in ATIP. Knowledge of this helps deploy and use ATIP.

Management Stack

GitOps Engine

Git Repo



Kubernetes Management
Platform

Multi-cluster, Multi-cloud

GitOps-Ready

CNCF Cluster
API

OS, I
Mar

Public/Private Cloud

Bare M
x86-

27.2 Components

There are two different blocks, the management stack and the runtime stack:

- **Management stack:** This is the part of ATIP that is used to manage the provision and lifecycle of the runtime stacks. It includes the following components:
 - Multi-cluster management in public and private cloud environments with Rancher ([Chapter 4, Rancher](#))
 - Bare-metal support with Metal3 ([Chapter 8, Metal³](#)), MetalLB ([Chapter 17, MetalLB](#)) and CAPI (Cluster API) infrastructure providers
 - Comprehensive tenant isolation and IDP (Identity Provider) integrations
 - Large marketplace of third-party integrations and extensions
 - Vendor-neutral API and rich ecosystem of providers
 - Control the SLE Micro transactional updates
 - GitOps Engine for managing the lifecycle of the clusters using Git repositories with Fleet ([Chapter 6, Fleet](#))
- **Runtime stack:** This is the part of ATIP that is used to run the workloads.
 - Kubernetes with secure and lightweight distributions like K3s ([Chapter 13, K3s](#)) and RKE2 ([Chapter 14, RKE2](#)) (RKE2 is hardened, certified and optimized for government use and regulated industries).
 - NeuVector ([Chapter 16, NeuVector](#)) to enable security features like image vulnerability scanning, deep packet inspection and automatic intra-cluster traffic control.
 - Block Storage with Longhorn ([Chapter 15, Longhorn](#)) to enable a simple and easy way to use a cloud native storage solution.
 - Optimized Operating System with SLE Micro ([Chapter 7, SLE Micro](#)) to enable a secure, lightweight and immutable (transactional file system) OS for running containers. SLE Micro is available on aarch64 and x86_64 architectures, and it also supports Real-Time Kernel for Telco and edge use cases.

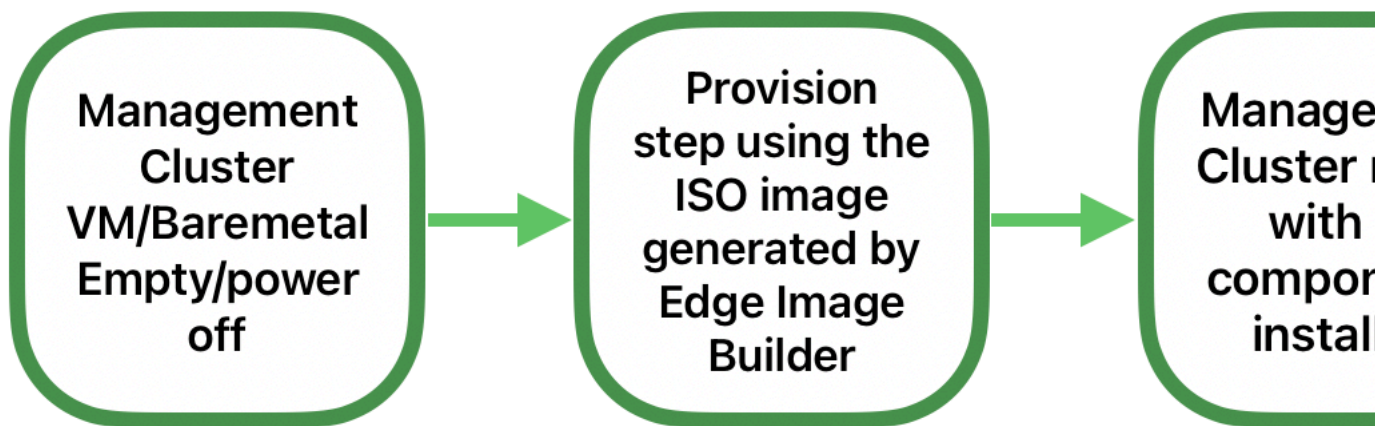
27.3 Example deployment flows

The following are high-level examples of workflows to understand the relationship between the management and the runtime components.

Directed network provisioning is the workflow that enables the deployment of a new downstream cluster with all the components preconfigured and ready to run workloads with no manual intervention.

27.3.1 Example 1: Deploying a new management cluster with all components installed

Using the Edge Image Builder ([Chapter 9, Edge Image Builder](#)) to create a new ISO image with the management stack included. You can then use this ISO image to install a new management cluster on VMs or bare-metal.



Note

For more information about how to deploy a new management cluster, see the ATIP Management Cluster guide ([Chapter 29, Setting up the management cluster](#)).



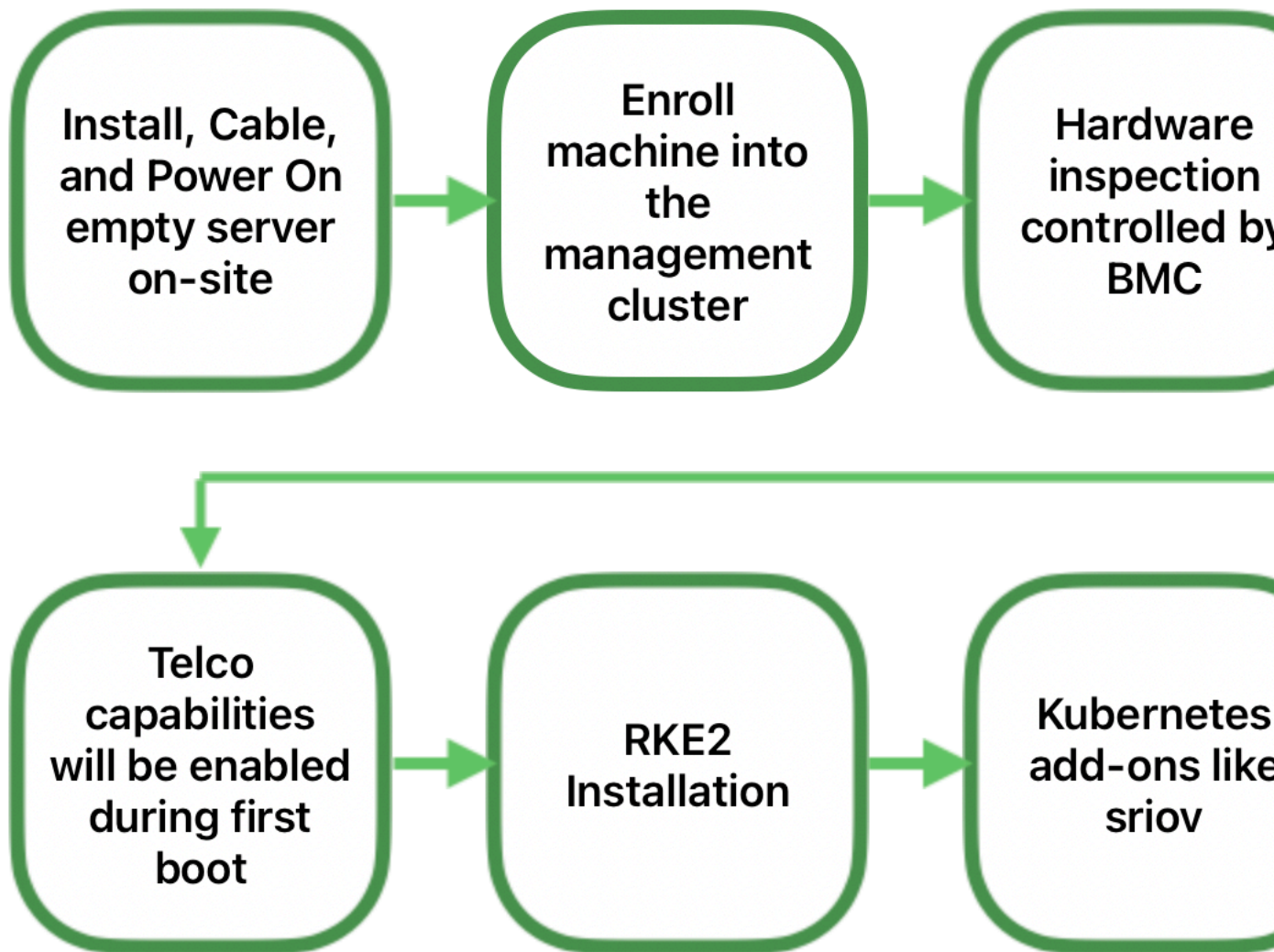
Note

For more information about how to use the Edge Image Builder, see the Edge Image Builder guide ([Chapter 3, Standalone clusters with Edge Image Builder](#)).

27.3.2 Example 2: Deploying a single-node downstream cluster with Telco profiles to enable it to run Telco workloads

Once we have the management cluster up and running, we can use it to deploy a single-node downstream cluster with all Telco capabilities enabled and configured using the directed network provisioning workflow.

The following diagram shows the high-level workflow to deploy it:





Note

For more information about how to deploy a downstream cluster, see the ATIP Automated Provisioning guide. ([Chapter 31, Fully automated directed network provisioning](#))



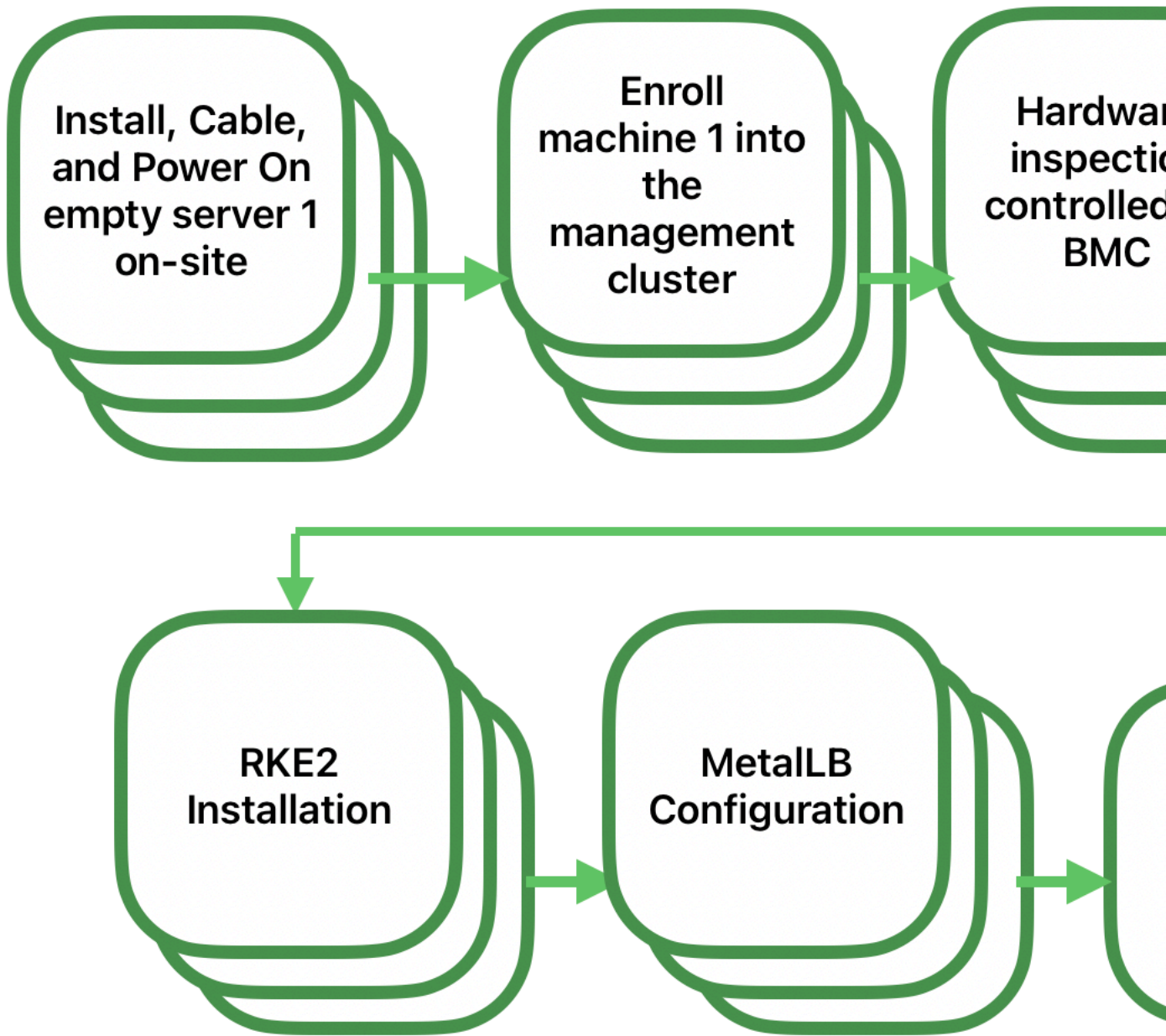
Note

For more information about Telco features, see the ATIP Telco Features guide. ([Chapter 30, Telco features configuration](#))

27.3.3 Example 3: Deploying a high availability downstream cluster using MetalLB as a Load Balancer

Once we have the management cluster up and running, we can use it to deploy a high availability downstream cluster with MetaLLB as a load balancer using the directed network provisioning workflow.

The following diagram shows the high-level workflow to deploy it:



Note

For more information about how to deploy a downstream cluster, see the ATIP Automated Provisioning guide. ([Chapter 31, Fully automated directed network provisioning](#))



Note

For more information about MetaLLB, see here: ([Chapter 17, MetalLB](#))

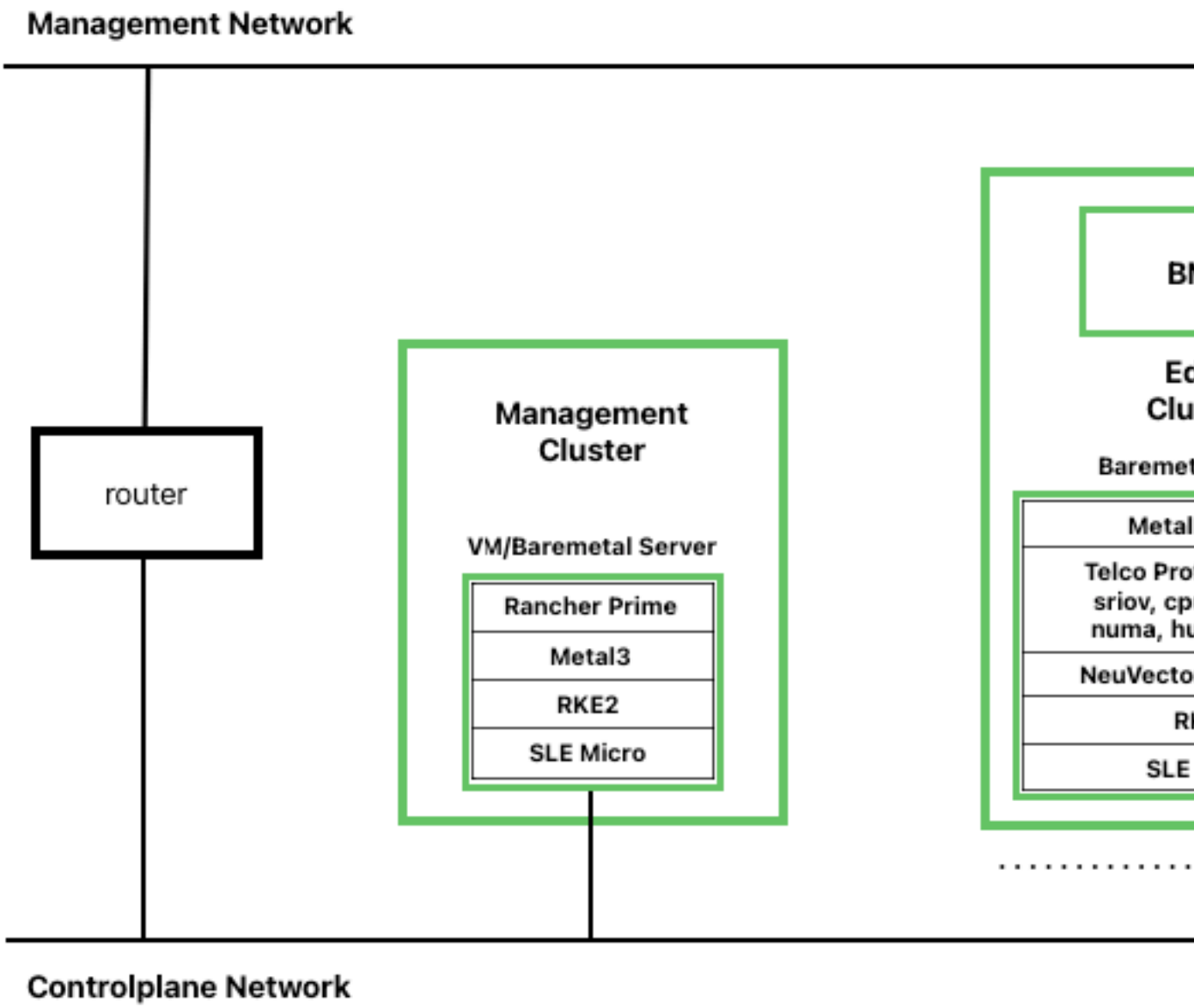
28 Requirements & Assumptions

28.1 Hardware

The hardware requirements for the ATIP nodes are based on the following components:

- **Management cluster:** The management cluster contains components like [SLE Micro](#), [RKE2](#), [Rancher Prime](#), [Metal3](#), and it is used to manage several downstream clusters. Depending on the number of downstream clusters to be managed, the hardware requirements for the server could vary.
 - Minimum requirements for the server ([VM](#) or [bare-metal](#)) are:
 - RAM: 8 GB Minimum (we recommend at least 16 GB)
 - CPU: 2 Minimum (we recommend at least 4 CPU)
- **Downstream clusters:** The downstream clusters are the clusters deployed on the ATIP nodes to run Telco workloads. Specific requirements are needed to enable certain Telco capabilities like [SR-IOV](#), [CPU Performance Optimization](#), etc.
 - SR-IOV: to attach VFs (Virtual Functions) in pass-through mode to CNFs/VNFs, the NIC must support SR-IOV and VT-d/AMD-Vi be enabled in the BIOS.
 - CPU Processors: To run specific Telco workloads, the CPU Processor model should be adapted to enable most of the features available in this reference table ([Chapter 30, Telco features configuration](#)).
 - Firmware requirements for installing with virtual media:

Server Hardware	BMC Model	Management
Dell hardware	15th Generation	iDRAC9
Supermicro hardware	01.00.25	Supermicro SMC - redfish
HPE hardware	1.50	iLO6



The network architecture is based on the following components:

- **Management network:** This network is used for the management of the ATIP nodes. It is used for the out-of-band management. Usually, this network is also connected to a separate management switch, but it can be connected to the same service switch using VLANs to isolate the traffic.
- **Control-plane network:** This network is used for the communication between the ATIP nodes and the services that are running on them. This network is also used for the communication between the ATIP nodes and the external services, like the DHCP or DNS servers. In some cases, for connected environments, the switch/router can handle traffic through the Internet.
- **Other networks:** In some cases, the ATIP nodes could be connected to other networks for specific customer purposes.



Note

To use the directed network provisioning workflow, the management cluster must have network connectivity to the downstream cluster server Baseboard Management Controller (BMC) so that host preparation and provisioning can be automated.

28.3 Services (DHCP, DNS, etc.)

Some external services like DHCP, DNS, etc. could be required depending on the kind of environment where they are deployed:

- **Connected environment:** In this case, the ATIP nodes will be connected to the Internet (via routing L3 protocols) and the external services will be provided by the customer.
- **Disconnected / air-gap environment:** In this case, the ATIP nodes will not have Internet IP connectivity and additional services will be required to locally mirror content required by the ATIP directed network provisioning workflow.
- **File server:** A file server is used to store the ISO images to be provisioned on the ATIP nodes during the directed network provisioning workflow. The meta13 Helm chart can deploy a media server to store the ISO images — check the following section (*Note*), but it is also possible to use an existing local webserver.

28.4 Disabling rebootmgr

`rebootmgr` is a service which allows to configure a strategy for reboot when the system has pending updates. For Telco workloads, it is really important to disable or configure properly the `rebootmgr` service to avoid the reboot of the nodes in case of updates scheduled by the system, to avoid any impact on the services running on the nodes.



Note

For more information about `rebootmgr`, see [rebootmgr GitHub repository \(https://github.com/SUSE/rebootmgr\)](https://github.com/SUSE/rebootmgr).

Verify the strategy being used by running:

```
cat /etc/rebootmgr.conf
[rebootmgr]
window-start=03:30
window-duration=1h30m
strategy=best-effort
lock-group=default
```

and you could disable it by running:

```
sed -i 's/strategy=best-effort/strategy=off/g' /etc/rebootmgr.conf
```

or using the `rebootmgrctl` command:

```
rebootmgrctl strategy off
```



Note

This configuration to set the `rebootmgr` strategy can be automated using the directed network provisioning workflow. For more information, check the ATIP Automated Provisioning documentation ([Chapter 31, Fully automated directed network provisioning](#)).

29 Setting up the management cluster

29.1 Introduction

The management cluster is the part of ATIP that is used to manage the provision and lifecycle of the runtime stacks. From a technical point of view, the management cluster contains the following components:

- [SUSE Linux Enterprise Micro](#) as the OS. Depending on the use case, some configurations like networking, storage, users and kernel arguments can be customized.
- [RKE2](#) as the Kubernetes cluster. Depending on the use case, it can be configured to use specific CNI plugins, such as [Multus](#), [Cilium](#), etc.
- [Rancher](#) as the management platform to manage the lifecycle of the clusters.
- [Metal3](#) as the component to manage the lifecycle of the bare-metal nodes.
- [CAPI](#) as the component to manage the lifecycle of the Kubernetes clusters (downstream clusters). With ATIP, also the [RKE2 CAPI Provider](#) is used to manage the lifecycle of the RKE2 clusters (downstream clusters).

With all components mentioned above, the management cluster can manage the lifecycle of downstream clusters, using a declarative approach to manage the infrastructure and applications.



Note

For more information about [SUSE Linux Enterprise Micro](#), see: SLE Micro ([Chapter 7, SLE Micro](#))

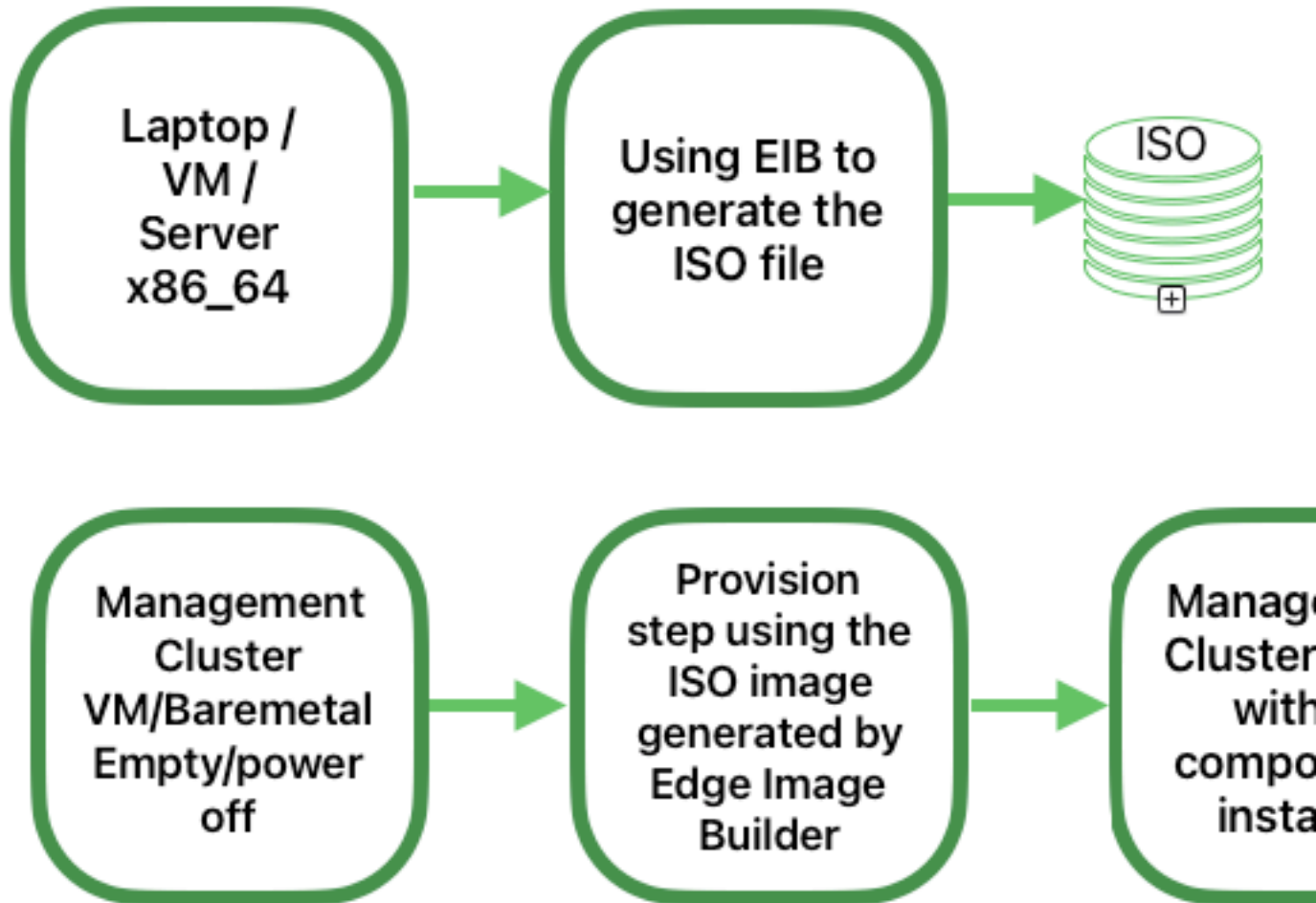
For more information about [RKE2](#), see: RKE2 ([Chapter 14, RKE2](#))

For more information about [Rancher](#), see: Rancher ([Chapter 4, Rancher](#))

For more information about [Metal3](#), see: Metal3 ([Chapter 8, Metal³](#))

29.2 Steps to set up the management cluster

The following steps are necessary to set up the management cluster (using a single node):



The following are the main steps to set up the management cluster using a declarative approach:

1. **Image preparation for connected environments** ([Section 29.3, “Image preparation for connected environments”](#)): The first step is to prepare the manifests and files with all the necessary configurations to be used in connected environments.

- Directory structure for connected environments (*Section 29.3.1, "Directory structure"*): This step creates a directory structure to be used by Edge Image Builder to store the configuration files and the image itself.
 - Management cluster definition file (*Section 29.3.2, "Management cluster definition file"*): The `mgmt-cluster.yaml` file is the main definition file for the management cluster. It contains the following information about the image to be created:
 - Image Information: The information related to the image to be created using the base image.
 - Operating system: The operating system configurations to be used in the image.
 - Kubernetes: Helm charts and repositories, kubernetes version, network configuration, and the nodes to be used in the cluster.
 - Custom folder (*Section 29.3.3, "Custom folder"*): The `custom` folder contains the configuration files and scripts to be used by Edge Image Builder to deploy a fully functional management cluster.
 - Files: Contains the configuration files to be used by the management cluster.
 - Scripts: Contains the scripts to be used by the management cluster.
 - Kubernetes folder (*Section 29.3.4, "Kubernetes folder"*): The `kubernetes` folder contains the configuration files to be used by the management cluster.
 - Manifests: Contains the manifests to be used by the management cluster.
 - Helm: Contains the Helm charts to be used by the management cluster.
 - Config: Contains the configuration files to be used by the management cluster.
 - Network folder (*Section 29.3.5, "Networking folder"*): The `network` folder contains the network configuration files to be used by the management cluster nodes.
2. **Image preparation for air-gap environments** (*Section 29.4, "Image preparation for air-gap environments"*): The step is to show the differences to prepare the manifests and files to be used in an air-gap scenario.

- Directory structure for air-gap environments (*Section 29.4.1, “Directory structure for air-gap environments”*): The directory structure must be modified to include the resources needed to run the management cluster in an air-gap environment.
 - Modifications in the definition file (*Section 29.4.2, “Modifications in the definition file”*): The `mgmt-cluster.yaml` file must be modified to include the `embeddedArtifactRegistry` section with the `images` field set to all container images to be included into the EIB output image.
 - Modifications in the custom folder (*Section 29.4.3, “Modifications in the custom folder”*): The `custom` folder must be modified to include the resources needed to run the management cluster in an air-gap environment.
 - Register script: The `custom/scripts/99-register.sh` script must be removed when you use an air-gap environment.
 - Air-gap resources: The `custom/files/airgap-resources.tar.gz` file must be included in the `custom/files` folder with all the resources needed to run the management cluster in an air-gap environment.
 - Scripts: The `custom/scripts/99-mgmt-setup.sh` script must be modified to extract and copy the `airgap-resources.tar.gz` file to the final location. The `custom/files/meta13.sh` script must be modified to use the local resources included in the `airgap-resources.tar.gz` file instead of downloading them from the internet.
3. **Image creation** (*Section 29.5, “Image creation”*): This step covers the creation of the image using the Edge Image Builder tool (for both, connected and air-gap scenarios). Check the prerequisites (*Chapter 9, Edge Image Builder*) to run the Edge Image Builder tool on your system.
 4. **Management Cluster Provision** (*Section 29.6, “Provision the management cluster”*): This step covers the provisioning of the management cluster using the image created in the previous step (for both, connected and air-gap scenarios). This step can be done using a laptop, server, VM or any other x86_64 system with a USB port.



Note

For more information about Edge Image Builder, see Edge Image Builder ([Chapter 9, Edge Image Builder](#)) and Edge Image Builder Quick Start ([Chapter 3, Standalone clusters with Edge Image Builder](#)).

29.3 Image preparation for connected environments

Using Edge Image Builder to create the image for the management cluster, a lot of configurations can be customized, but in this document, we cover the minimal configurations necessary to set up the management cluster. Edge Image Builder is typically run from inside a container so, if you do not already have a way to run containers, we need to start by installing a container runtime such as [Podman \(https://podman.io\)](https://podman.io) or [Rancher Desktop \(https://rancherdesktop.io\)](https://rancherdesktop.io). For this guide, we assume you already have a container runtime available.

Also, as a prerequisite to deploy a highly available management cluster, you need to reserve three IPs in your network: - [apiVIP](#) for the API VIP Address (used to access the Kubernetes API server). - [ingressVIP](#) for the Ingress VIP Address (consumed, for example, by the Rancher UI). - [metal3VIP](#) for the Metal3 VIP Address.

29.3.1 Directory structure

When running EIB, a directory is mounted from the host, so the first thing to do is to create a directory structure to be used by EIB to store the configuration files and the image itself. This directory has the following structure:

```
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

```



Note

The image `SLE-Micro.x86_64-5.5.0-Default-SelfInstall-GM2.install.iso` must be downloaded from the [SUSE Customer Center \(https://scc.suse.com/\)](https://scc.suse.com/) or the [SUSE Download page \(https://www.suse.com/download/sle-micro/\)](https://www.suse.com/download/sle-micro/), and it must be located under the `base-images` folder.

You should check the SHA256 checksum of the image to ensure it has not been tampered with. The checksum can be found in the same location where the image was downloaded.

An example of the directory structure can be found in the [SUSE Edge GitHub repository](https://github.com/suse-edge/atip) under the "telco-examples" folder (<https://github.com/suse-edge/atip>).

29.3.2 Management cluster definition file

The `mgmt-cluster.yaml` file is the main definition file for the management cluster. It contains the following information:

```

apiVersion: 1.0
image:
  imageType: iso
  arch: x86_64
  baseImage: SLE-Micro.x86_64-5.5.0-Default-SelfInstall-GM2.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: ${KUBERNETES_VERSION}
  helm:
    charts:
      - name: cert-manager
        repositoryName: jetstack
        version: 1.14.2
        targetNamespace: cert-manager
        valuesFile: certmanager.yaml
        createNamespace: true
        installationNamespace: kube-system
      - name: longhorn-crd
        version: 103.3.0+up1.6.1
        repositoryName: rancher-charts
        targetNamespace: longhorn-system
        createNamespace: true
        installationNamespace: kube-system
      - name: longhorn
        version: 103.3.0+up1.6.1
        repositoryName: rancher-charts
        targetNamespace: longhorn-system
        createNamespace: true
        installationNamespace: kube-system
      - name: metal3-chart
        version: 0.7.4
        repositoryName: suse-edge-charts
        targetNamespace: metal3-system
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: metal3.yaml
      - name: neuvector-crd
        version: 103.0.3+up2.7.6
        repositoryName: rancher-charts
        targetNamespace: neuvector
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: neuvector.yaml
      - name: neuvector
```

```

    version: 103.0.3+up2.7.6
    repositoryName: rancher-charts
    targetNamespace: neuvector
    createNamespace: true
    installationNamespace: kube-system
    valuesFile: neuvector.yaml
  - name: rancher
    version: 2.8.8
    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
  - 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

```

To explain the fields and values in the `mgmt-cluster.yaml` definition file, we have divided it into the following sections.

- Image section (definition file):

```

image:
  imageType: iso
  arch: x86_64
  baseImage: SLE-Micro.x86_64-5.5.0-Default-SelfInstall-GM2.install.iso
  outputImageName: eib-mgmt-cluster-image.iso

```

where the `baseImage` is the original image you downloaded from the SUSE Customer Center or the SUSE Download page. `outputImageName` is the name of the new image that will be used to provision the management cluster.

- Operating system section (definition file):

```
operatingSystem:
  isoConfiguration:
    installDevice: /dev/sda
  users:
  - username: root
    encryptedPassword: ${ROOT_PASSWORD}
  packages:
    packageList:
    - jq
    sccRegistrationCode: ${SCC_REGISTRATION_CODE}
```

where the `installDevice` is the device to be used to install the operating system, the `username` and `encryptedPassword` are the credentials to be used to access the system, the `packageList` is the list of packages to be installed (`jq` is required internally during the installation process), and the `sccRegistrationCode` is the registration code used to get the packages and dependencies at build time and can be obtained from the SUSE Customer Center. The encrypted password can be generated using the `openssl` command as follows:

```
openssl passwd -6 MyPassword!123
```

This outputs something similar to:

```
$6$UrXB1sAGs46D0iSq$HSwi9GFJLCorm0J53nF2Sq8YEoyINhHc0bHzX2R8h13mswUIsMwzx4eUzn/
rRx0QPv4JIb0eWCoNrxGiKH4R31
```

- Kubernetes section (definition file):

```
kubernetes:
  version: ${KUBERNETES_VERSION}
  helm:
    charts:
    - name: cert-manager
      repositoryName: jetstack
      version: 1.14.2
      targetNamespace: cert-manager
      valuesFile: certmanager.yaml
      createNamespace: true
      installationNamespace: kube-system
    - name: longhorn-crd
```



```
version: 103.3.0+up1.6.1
repositoryName: rancher-charts
targetNamespace: longhorn-system
createNamespace: true
installationNamespace: kube-system
- name: longhorn
  version: 103.3.0+up1.6.1
  repositoryName: rancher-charts
  targetNamespace: longhorn-system
  createNamespace: true
  installationNamespace: kube-system
- name: metal3-chart
  version: 0.7.4
  repositoryName: suse-edge-charts
  targetNamespace: metal3-system
  createNamespace: true
  installationNamespace: kube-system
  valuesFile: metal3.yaml
- name: neuvector-crd
  version: 103.0.3+up2.7.6
  repositoryName: rancher-charts
  targetNamespace: neuvector
  createNamespace: true
  installationNamespace: kube-system
  valuesFile: neuvector.yaml
- name: neuvector
  version: 103.0.3+up2.7.6
  repositoryName: rancher-charts
  targetNamespace: neuvector
  createNamespace: true
  installationNamespace: kube-system
  valuesFile: neuvector.yaml
- name: rancher
  version: 2.8.8
  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
- 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
```

where version is the version of Kubernetes to be installed. In our case, we are using an RKE2 cluster, so the version must be minor less than 1.29 to be compatible with Rancher (for example, v1.28.13+rke2r1).

The helm section contains the list of Helm charts to be installed, the repositories to be used, and the version configuration for all of them.

The network section contains the configuration for the network, like the apiHost and apiVIP to be used by the RKE2 component. The apiVIP should be an IP address that is not used in the network and should not be part of the DHCP pool (in case we use DHCP). Also, when we use the apiVIP in a multi-node cluster, it is used to access the Kubernetes API server. The apiHost is the name resolution to apiVIP to be used by the RKE2 component.

The nodes section contains the list of nodes to be used in the cluster. The nodes section contains the list of nodes to be used in the cluster. In this example, a single-node cluster is being used, but it can be extended to a multi-node cluster by adding more nodes to the list (by uncommenting the lines).



Note

- The names of the nodes must be unique in the cluster.
- Optionally, use the initializer field to specify the bootstrap host, otherwise it will be the first node in the list.
- The names of the nodes must be the same as the host names defined in the Network Folder (*Section 29.3.5, "Networking folder"*) when network configuration is required.

29.3.3 Custom folder

The `custom` folder contains the following subfolders:

```
...
├─ custom
│  └─ scripts
│     └─ 99-register.sh
│     └─ 99-mgmt-setup.sh
│     └─ 99-alias.sh
│  └─ files
│     └─ rancher.sh
│     └─ mgmt-stack-setup.service
│     └─ metal3.sh
│     └─ basic-setup.sh
...
```

- The `custom/files` folder contains the configuration files to be used by the management cluster.
- The `custom/scripts` folder contains the scripts to be used by the management cluster.

The `custom/files` folder contains the following files:

- `basic-setup.sh`: contains the configuration parameters about the `Metal3` version to be used, as well as the `Rancher` and `MetalLB` basic parameters. Only modify this file if you want to change the versions of the components or the namespaces to be used.

```
#!/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"
export METAL3_CLUSTERCTLVERSION="1.6.2"
export METAL3_CAPICOREVERSION="1.6.2"
export METAL3_CAPIMETAL3VERSION="1.6.0"
export METAL3_CAPIRKE2VERSION="0.4.1"
export METAL3_CAPIPROVIDER="rke2"
export METAL3_CAPISYSTEMNAMESPACE="capi-system"
export METAL3_RKE2BOOTSTRAPNAMESPACE="rke2-bootstrap-system"
export METAL3_CAPM3NAMESPACE="capm3-system"
```

```

export METAL3_RKE2CONTROLPLANENAMESPACE="rke2-control-plane-system"
export METAL3_CAPI_IMAGES="registry.suse.com/edge"
# Or registry.opensuse.org/isv/suse/edge/clusterapi/containerfile/suse for the
  upstream ones

#####
# METALLB #
#####
export METALLB_NAMESPACE="metallb-system"

#####
# RANCHER #
#####
export RANCHER_CHART_TARGET_NAMESPACE="cattle-system"
export RANCHER_FINAL_PASSWORD="adminadminadmin"

die(){
  echo ${1} 1>&2
  exit ${2}
}

```

- `metal3.sh`: contains the configuration for the `Metal3` component to be used (no modifications needed). In future versions, this script will be replaced to use instead `Rancher Turtles` to make it easy.

```

#!/bin/bash
set -euo pipefail

BASEDIR="$(dirname "$0")"
source ${BASEDIR}/basic-setup.sh

METAL3_LOCK_NAMESPACE="default"
METAL3_LOCK_CMNAME="metal3-lock"

trap 'catch $? $LINENO' EXIT

catch() {
  if [ "$1" != "0" ]; then
    echo "Error $1 occurred on $2"
    ${KUBECTL} delete configmap ${METAL3_LOCK_CMNAME} -n ${METAL3_LOCK_NAMESPACE}
  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: ${METALLB_NAMESPACE}
spec:
  ipAddressPools:
    - ironic-ip-pool
EOF
fi

# If clusterctl is not installed, install it
if ! command -v clusterctl > /dev/null 2>&1; then
  LINUXARCH=$(uname -m)
  case $(uname -m) in
    "x86_64")
      export GOARCH="amd64" ;;
    "aarch64")
      export GOARCH="arm64" ;;
    *)
      echo "Arch not found, assuming amd64"
      export GOARCH="amd64" ;;
  esac

  # Clusterctl bin
  # Maybe just use the binary from hauler if available
  curl -L https://github.com/kubernetes-sigs/cluster-api/releases/download/v
  ${METAL3_CLUSTERCTL_VERSION}/clusterctl-linux-${GOARCH} -o /usr/local/bin/clusterctl
  chmod +x /usr/local/bin/clusterctl
fi

# If rancher is deployed
if [ $((${KUBECTL} get pods -n ${RANCHER_CHART_TARGET_NAMESPACE} -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 mutatingwebhookconfiguration.admissionregistration.k8s.io
  mutating-webhook-configuration
  ${KUBECTL} delete validatingwebhookconfigurations.admissionregistration.k8s.io
  validating-webhook-configuration

```

```

    ${KUBECTL} wait --for=delete namespace/cattle-provisioning-capi-system --
timeout=300s
fi

# Deploy CAPI
if [ $((${KUBECTL} get pods -n ${METAL3_CAPISYSTEMNAMESPACE} -o name | wc -l) -lt
1 ]; then

    # https://github.com/rancher-sandbox/cluster-api-provider-rke2#setting-up-
clusterctl
    mkdir -p ~/.cluster-api
    cat <<-EOF > ~/.cluster-api/clusterctl.yaml
images:
  all:
    repository: ${METAL3_CAPI_IMAGES}
EOF

    # Try this command 3 times just in case, stolen from https://stackoverflow.com/
a/33354419
    if ! (r=3; while ! clusterctl init \
--core "cluster-api:v${METAL3_CAPICOREVERSION}"\
--infrastructure "metal3:v${METAL3_CAPIMETAL3VERSION}"\
--bootstrap "${METAL3_CAPIPROVIDER}:v${METAL3_CAPIRKE2VERSION}"\
--control-plane "${METAL3_CAPIPROVIDER}:v${METAL3_CAPIRKE2VERSION}" ; do
((--r))||exit
echo "Something went wrong, let's wait 10 seconds and retry"
sleep 10;done) ; then
echo "clusterctl failed"
exit 1
fi

    # Wait for capi-controller-manager
    while ! ${KUBECTL} wait --for condition=ready -n ${METAL3_CAPISYSTEMNAMESPACE}
${(${KUBECTL} get pods -n ${METAL3_CAPISYSTEMNAMESPACE} -l cluster.x-k8s.io/
provider=cluster-api -o name) --timeout=10s; do sleep 2 ; done

    # Wait for capm3-controller-manager, there are two pods, the ipam and the capm3
one, just wait for the first one
    while ! ${KUBECTL} wait --for condition=ready -n ${METAL3_CAPM3NAMESPACE}
${(${KUBECTL} get pods -n ${METAL3_CAPM3NAMESPACE} -l cluster.x-k8s.io/
provider=infrastructure-metal3 -o name | head -n1 ) --timeout=10s; do sleep 2 ; done

    # Wait for rke2-bootstrap-controller-manager
    while ! ${KUBECTL} wait --for condition=ready -n ${METAL3_RKE2BOOTSTRAPNAMESPACE}
${(${KUBECTL} get pods -n ${METAL3_RKE2BOOTSTRAPNAMESPACE} -l cluster.x-k8s.io/
provider=bootstrap-rke2 -o name) --timeout=10s; do sleep 2 ; done

```

```

# Wait for rke2-control-plane-controller-manager
while ! ${KUBECTL} wait --for condition=ready -n
${METAL3_RKE2CONTROLPLANENAMESPACE} ${KUBECTL} get pods -n
${METAL3_RKE2CONTROLPLANENAMESPACE} -l cluster.x-k8s.io/provider=control-plane-rke2
-o name) --timeout=10s; do sleep 2 ; done

fi

# Clean up the lock cm

${KUBECTL} delete configmap ${METAL3LOCKCMNAME} -n ${METAL3LOCKNAMESPACE}

```

- `rancher.sh`: contains the configuration for the Rancher component to be used (no modifications needed).

```

#!/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: contains the configuration to create the systemd service to run the scripts during the first boot (no modifications needed).

```

[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 on
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

```

The custom/scripts folder contains the following files:

- 99-alias.sh script: contains the alias to be used by the management cluster to load the kubeconfig file at first boot (no modifications needed).

```
#!/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
```

- 99-mgmt-setup.sh script: contains the configuration to copy the scripts during the first boot (no modifications needed).

```
#!/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
```

- 99-register.sh script: contains the configuration to register the system using the SCC registration code. The `${SCC_ACCOUNT_EMAIL}` and `${SCC_REGISTRATION_CODE}` have to be set properly to register the system with your account.

```
#!/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}"
```

29.3.4 Kubernetes folder

The kubernetes folder contains the following subfolders:

```
...
├─ 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
| ...
```

The `kubernetes/config` folder contains the following files:

- `server.yaml`: By default, the CNI plug-in installed by default is `Cilium`, so you do not need to create this folder and file. Just in case you need to customize the CNI plug-in, you can use the `server.yaml` file under the `kubernetes/config` folder. It contains the following information:

```
cni:
- multus
- cilium
```



Note

This is an optional file to define certain Kubernetes customization, like the CNI plug-ins to be used or many options you can check in the [official documentation \(https://docs.rke2.io/install/configuration\)](https://docs.rke2.io/install/configuration).

The `kubernetes/manifests` folder contains the following files:

- `rke2-ingress-config.yaml`: contains the configuration to create the `Ingress` service for the management cluster (no modifications needed).

```
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: contains the configuration to create the NeuVector namespace (no modifications needed).

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    pod-security.kubernetes.io/enforce: privileged
  name: neuvector
```

- ingress-l2-adv.yaml: contains the configuration to create the L2Advertisement for the MetallB component (no modifications needed).

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: ingress-l2-adv
  namespace: metallb-system
spec:
  ipAddressPools:
    - ingress-ippool
```

- ingress-ippool.yaml: contains the configuration to create the IPAddressPool for the rke2-ingress-nginx component. The `${INGRESS_VIP}` has to be set properly to define the IP address reserved to be used by the rke2-ingress-nginx component.

```
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]}
```

The `kubernetes/helm/values` folder contains the following files:

- `rancher.yaml`: contains the configuration to create the Rancher component. The `${INGRESS_VIP}` must be set properly to define the IP address to be consumed by the Rancher component. The URL to access the Rancher component will be `https://rancher-${INGRESS_VIP}.sslip.io`.

```
hostname: rancher-${INGRESS_VIP}.sslip.io
bootstrapPassword: "foobar"
replicas: 1
global.cattle.psp.enabled: "false"
```

- `neuvevector.yaml`: contains the configuration to create the NeuVector component (no modifications needed).

```
controller:
  replicas: 1
  ranchersso:
    enabled: true
manager:
  enabled: false
cve:
  scanner:
    enabled: false
    replicas: 1
k3s:
  enabled: true
crdwebhook:
  enabled: false
```

- `metal3.yaml`: contains the configuration to create the Metal3 component. The `${METAL3_VIP}` must be set properly to define the IP address to be consumed by the Metal3 component.

```
global:
  ironicIP: ${METAL3_VIP}
  enable_vmedia_tls: false
  additionalTrustedCAs: false
metal3-ironic:
  global:
    predictableNicNames: "true"
  persistence:
```

```
ironic:
  size: "5Gi"
```



Note

The Media Server is an optional feature included in Metal³ (by default is disabled). To use the Metal3 feature, you need to configure it on the previous manifest. To use the Metal³ media server, specify the following variable:

- add the `enable_metal3_media_server` to `true` to enable the media server feature in the global section.
- include the following configuration about the media server where `${MEDIA_VOLUME_PATH}` is the path to the media volume in the media (e.g `/home/metal3/bmh-image-cache`)

```
metal3-media:
  mediaVolume:
    hostPath: ${MEDIA_VOLUME_PATH}
```

An external media server can be used to store the images, and in the case you want to use it with TLS, you will need to modify the following configurations:

- set to `true` the `additionalTrustedCAs` in the previous `metal3.yaml` file to enable the additional trusted CAs from the external media server.
- include the following secret configuration in the folder `kubernetes/manifests/metal3-cacert-secret.yaml` to store the CA certificate of the external media server.

```
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 }}
```

The `additional_ca_cert` is the base64-encoded CA certificate of the external media server. You can use the following command to encode the certificate and generate the secret doing manually:

```
kubectl -n meta3-system create secret generic tls-ca-additional --from-file=ca-additional.crt=./ca-additional.crt
```

- `certmanager.yaml`: contains the configuration to create the `Cert-Manager` component (no modifications needed).

```
installCRDs: "true"
```

29.3.5 Networking folder

The `network` folder contains as many files as nodes in the management cluster. In our case, we have only one node, so we have only one file called `mgmt-cluster-node1.yaml`. The name of the file must match the host name defined in the `mgmt-cluster.yaml` definition file into the `network/node` section described above.

If you need to customize the networking configuration, for example, to use a specific static IP address (DHCP-less scenario), you can use the `mgmt-cluster-node1.yaml` file under the `network` folder. It contains the following information:

- `_${MGMT_GATEWAY}`: The gateway IP address.
- `_${MGMT_DNS}`: The DNS server IP address.
- `_${MGMT_MAC}`: The MAC address of the network interface.
- `_${MGMT_NODE_IP}`: The IP address of the management cluster.

```
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

```

If you want to use DHCP to get the IP address, you can use the following configuration (the MAC address must be set properly using the `${MGMT_MAC}` variable):

```

## 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

```



Note

- Depending on the number of nodes in the management cluster, you can create more files like `mgmt-cluster-node2.yaml`, `mgmt-cluster-node3.yaml`, etc. to configure the rest of the nodes.
- The `routes` section is used to define the routing table for the management cluster.

29.4 Image preparation for air-gap environments

This section describes how to prepare the image for air-gap environments showing only the differences from the previous sections. The following changes to the previous section (Image preparation for connected environments ([Section 29.3, “Image preparation for connected environments”](#))) are required to prepare the image for air-gap environments:

- The `mgmt-cluster.yaml` file must be modified to include the `embeddedArtifactRegistry` section with the `images` field set to all container images to be included into the EIB output image.
- The `custom/scripts/99-register.sh` script must be removed when use an air-gap environment.
- The `custom/files/airgap-resources.tar.gz` file must be included in the `custom/files` folder with all the resources needed to run the management cluster in an air-gap environment.
- The `custom/scripts/99-mgmt-setup.sh` script must be modified to extract and copy the `airgap-resources.tar.gz` file to the final location.
- The `custom/files/metal3.sh` script must be modified to use the local resources included in the `airgap-resources.tar.gz` file instead of downloading them from the internet.

29.4.1 Directory structure for air-gap environments

The directory structure for air-gap environments is the same as for connected environments, with the differences explained as follows:

```
eib
|-- base-images
|   |-- SLE-Micro.x86_64-5.5.0-Default-SelfInstall-GM2.install.iso
|-- custom
|   |-- files
|   |   |-- airgap-resources.tar.gz
|   |   |-- basic-setup.sh
|   |   |-- metal3.sh
|   |   |-- mgmt-stack-setup.service
|   |   |-- rancher.sh
|   |-- scripts
|       |-- 99-alias.sh
|       |-- 99-mgmt-setup.sh
```

```

|-- kubernetes
|   |-- config
|   |   |-- server.yaml
|   |-- helm
|   |   |-- values
|   |       |-- certmanager.yaml
|   |       |-- metal3.yaml
|   |       |-- neuvector.yaml
|   |       |-- rancher.yaml
|   |-- manifests
|       |-- neuvector-namespace.yaml
|-- mgmt-cluster.yaml
|-- network
    |-- mgmt-cluster-network.yaml

```



Note

The image `SLE-Micro.x86_64-5.5.0-Default-SelfInstall-GM2.install.iso` must be downloaded from the [SUSE Customer Center \(https://scc.suse.com/\)](https://scc.suse.com/) or the [SUSE Download page \(https://www.suse.com/download/sle-micro/\)](https://www.suse.com/download/sle-micro/), and it must be located under the `base-images` folder before starting with the process.

You should check the SHA256 checksum of the image to ensure it has not been tampered with. The checksum can be found in the same location where the image was downloaded.

An example of the directory structure can be found in the [SUSE Edge GitHub repository](https://github.com/suse-edge/atip) under the "telco-examples" folder (<https://github.com/suse-edge/atip>).

29.4.2 Modifications in the definition file

The `mgmt-cluster.yaml` file must be modified to include the `embeddedArtifactRegistry` section with the `images` field set to all container images to be included into the EIB output image. The `images` field must contain the list of all container images to be included in the output image. The following is an example of the `mgmt-cluster.yaml` file with the `embeddedArtifactRegistry` section included:

```

apiVersion: 1.0
image:
  imageType: iso
  arch: x86_64
  baseImage: SLE-Micro.x86_64-5.5.0-Default-SelfInstall-GM2.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: ${KUBERNETES_VERSION}
  helm:
    charts:
      - name: cert-manager
        repositoryName: jetstack
        version: 1.14.2
        targetNamespace: cert-manager
        valuesFile: certmanager.yaml
        createNamespace: true
        installationNamespace: kube-system
      - name: longhorn-crd
        version: 103.3.0+up1.6.1
        repositoryName: rancher-charts
        targetNamespace: longhorn-system
        createNamespace: true
        installationNamespace: kube-system
      - name: longhorn
        version: 103.3.0+up1.6.1
        repositoryName: rancher-charts
        targetNamespace: longhorn-system
        createNamespace: true
        installationNamespace: kube-system
      - name: metal3-chart
        version: 0.7.4
        repositoryName: suse-edge-charts
        targetNamespace: metal3-system
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: metal3.yaml
      - name: neuvector-crd
        version: 103.0.3+up2.7.6
        repositoryName: rancher-charts
        targetNamespace: neuvector
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: neuvector.yaml
```

```

- name: neuvector
  version: 103.0.3+up2.7.6
  repositoryName: rancher-charts
  targetNamespace: neuvector
  createNamespace: true
  installationNamespace: kube-system
  valuesFile: neuvector.yaml
- name: rancher
  version: 2.8.8
  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
- 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.rancher.com/rancher/backup-restore-operator:v4.0.3
- name: registry.rancher.com/rancher/calico-cni:v3.27.4-rancher1
- name: registry.rancher.com/rancher/cis-operator:v1.0.15
- name: registry.rancher.com/rancher/coreos-kube-state-metrics:v1.9.7
- name: registry.rancher.com/rancher/coreos-prometheus-config-reloader:v0.38.1
- name: registry.rancher.com/rancher/coreos-prometheus-operator:v0.38.1
- name: registry.rancher.com/rancher/flannel-cni:v0.3.0-rancher9
- name: registry.rancher.com/rancher/fleet-agent:v0.9.9
- name: registry.rancher.com/rancher/fleet:v0.9.9
- name: registry.rancher.com/rancher/gitjob:v0.9.13

```

```

- name: registry.rancher.com/rancher/grafana-grafana:7.1.5
- name: registry.rancher.com/rancher/hardened-addon-resizer:1.8.20-build20240410
- name: registry.rancher.com/rancher/hardened-calico:v3.28.1-build20240806
- name: registry.rancher.com/rancher/hardened-cluster-autoscaler:v1.8.10-
build20240124
- name: registry.rancher.com/rancher/hardened-cni-plugins:v1.5.1-build20240805
- name: registry.rancher.com/rancher/hardened-coredns:v1.11.1-build20240305
- name: registry.rancher.com/rancher/hardened-dns-node-cache:1.22.28-build20240125
- name: registry.rancher.com/rancher/hardened-etcd:v3.5.13-k3s1-build20240531
- name: registry.rancher.com/rancher/hardened-flannel:v0.25.5-build20240801
- name: registry.rancher.com/rancher/hardened-k8s-metrics-server:v0.7.1-build20240401
- name: registry.rancher.com/rancher/hardened-kubernetes:v1.28.13-rke2r1-
build20240815
- name: registry.rancher.com/rancher/hardened-multus-cni:v4.0.2-build20240612
- name: registry.rancher.com/rancher/hardened-node-feature-discovery:v0.15.4-
build20240513
- name: registry.rancher.com/rancher/hardened-whereabouts:v0.7.0-build20240429
- name: registry.rancher.com/rancher/helm-project-operator:v0.2.1
- name: registry.rancher.com/rancher/istio-kubectrl:1.5.10
- name: registry.rancher.com/rancher/jimmydyson-configmap-reload:v0.3.0
- name: registry.rancher.com/rancher/k3s-upgrade:v1.28.13-k3s1
- name: registry.rancher.com/rancher/klipper-helm:v0.8.4-build20240523
- name: registry.rancher.com/rancher/klipper-lb:v0.4.9
- name: registry.rancher.com/rancher/kube-api-auth:v0.2.1
- name: registry.rancher.com/rancher/kubectrl:v1.28.12
- name: registry.rancher.com/rancher/library-nginx:1.19.2-alpine
- name: registry.rancher.com/rancher/local-path-provisioner:v0.0.28
- name: registry.rancher.com/rancher/machine:v0.15.0-rancher116
- name: registry.rancher.com/rancher/mirrored-cluster-api-controller:v1.4.4
- name: registry.rancher.com/rancher/nginx-ingress-controller:v1.10.4-hardened2
- name: registry.rancher.com/rancher/pause:3.6
- name: registry.rancher.com/rancher/prom-alertmanager:v0.21.0
- name: registry.rancher.com/rancher/prom-node-exporter:v1.0.1
- name: registry.rancher.com/rancher/prom-prometheus:v2.18.2
- name: registry.rancher.com/rancher/prometheus-auth:v0.2.2
- name: registry.rancher.com/rancher/prometheus-federator:v0.3.4
- name: registry.rancher.com/rancher/pushprox-client:v0.1.3-rancher2-client
- name: registry.rancher.com/rancher/pushprox-proxy:v0.1.3-rancher2-proxy
- name: registry.rancher.com/rancher/rancher-agent:v2.8.8
- name: registry.rancher.com/rancher/rancher-csp-adapter:v3.0.1
- name: registry.rancher.com/rancher/rancher-webhook:v0.4.11
- name: registry.rancher.com/rancher/rancher:v2.8.8
- name: registry.rancher.com/rancher/rke-tools:v0.1.102
- name: registry.rancher.com/rancher/rke2-cloud-provider:v1.29.3-build20240515
- name: registry.rancher.com/rancher/rke2-runtime:v1.28.13-rke2r1
- name: registry.rancher.com/rancher/rke2-upgrade:v1.28.13-rke2r1
- name: registry.rancher.com/rancher/security-scan:v0.2.17

```

- name: registry.rancher.com/rancher/shell:v0.1.26
- name: registry.rancher.com/rancher/system-agent-installer-k3s:v1.28.13-k3s1
- name: registry.rancher.com/rancher/system-agent-installer-rke2:v1.28.13-rke2r1
- name: registry.rancher.com/rancher/system-agent:v0.3.9-suc
- name: registry.rancher.com/rancher/system-upgrade-controller:v0.13.4
- name: registry.rancher.com/rancher/ui-plugin-catalog:2.1.0
- name: registry.rancher.com/rancher/ui-plugin-operator:v0.1.1
- name: registry.rancher.com/rancher/webhook-receiver:v0.2.5
- name: registry.rancher.com/rancher/kubectrl:v1.20.2
- name: registry.rancher.com/rancher/shell:v0.1.24
- name: registry.rancher.com/rancher/mirrored-ingress-nginx-kube-webhook-
certgen:v1.4.1
 - name: registry.rancher.com/rancher/mirrored-ingress-nginx-kube-webhook-
certgen:v20221220-controller-v1.5.1-58-g787ea74b6
 - name: registry.rancher.com/rancher/mirrored-ingress-nginx-kube-webhook-
certgen:v20230312-helm-chart-4.5.2-28-g66a760794
 - name: registry.rancher.com/rancher/mirrored-ingress-nginx-kube-webhook-
certgen:v20231011-8b53cabe0
 - name: registry.rancher.com/rancher/mirrored-ingress-nginx-kube-webhook-
certgen:v20231226-1a7112e06
 - name: registry.rancher.com/rancher/mirrored-longhornio-csi-attacher:v4.4.2
 - name: registry.rancher.com/rancher/mirrored-longhornio-csi-provisioner:v3.6.2
 - name: registry.rancher.com/rancher/mirrored-longhornio-csi-resizer:v1.9.2
 - name: registry.rancher.com/rancher/mirrored-longhornio-csi-snapshotter:v6.3.2
 - name: registry.rancher.com/rancher/mirrored-longhornio-csi-node-driver-
registrar:v2.9.2
 - name: registry.rancher.com/rancher/mirrored-longhornio-livenessprobe:v2.12.0
 - name: registry.rancher.com/rancher/mirrored-longhornio-backing-image-manager:v1.6.1
 - name: registry.rancher.com/rancher/mirrored-longhornio-longhorn-engine:v1.6.1
 - name: registry.rancher.com/rancher/mirrored-longhornio-longhorn-instance-
manager:v1.6.1
 - name: registry.rancher.com/rancher/mirrored-longhornio-longhorn-manager:v1.6.1
 - name: registry.rancher.com/rancher/mirrored-longhornio-longhorn-share-
manager:v1.6.1
 - name: registry.rancher.com/rancher/mirrored-longhornio-longhorn-ui:v1.6.1
 - name: registry.rancher.com/rancher/mirrored-longhornio-support-bundle-kit:v0.0.36
 - name: registry.suse.com/edge/cluster-api-provider-rke2-bootstrap:v0.4.1
 - name: registry.suse.com/edge/cluster-api-provider-rke2-controlplane:v0.4.1
 - name: registry.suse.com/edge/cluster-api-controller:v1.6.2
 - name: registry.suse.com/edge/cluster-api-provider-metal3:v1.6.0
 - name: registry.suse.com/edge/ip-address-manager:v1.6.0

29.4.3 Modifications in the custom folder

- The `custom/scripts/99-register.sh` script must be removed when using an air-gap environment. As you can see in the directory structure, the `99-register.sh` script is not included in the `custom/scripts` folder.
- The `custom/scripts/99-mgmt-setup.sh` script must be modified to extract and copy the `airgap-resources.tar.gz` file to the final location. The following is an example of the `99-mgmt-setup.sh` script with the modifications to extract and copy the `airgap-resources.tar.gz` file:

```
#!/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

# Extract the airgap resources
tar xzf airgap-resources.tar.gz

# Copy the clusterctl binary to the final location
cp airgap-resources/clusterctl /opt/mgmt/bin/ && chmod +x /opt/mgmt/bin/clusterctl

# Copy the clusterctl.yaml and override
mkdir -p /root/cluster-api
cp -r airgap-resources/clusterctl.yaml airgap-resources/overrides /root/cluster-api/
```

- The `custom/files/metal3.sh` script must be modified to use the local resources included in the `airgap-resources.tar.gz` file instead of downloading them from the internet. The following is an example of the `metal3.sh` script with the modifications to use the local resources:

```
#!/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

# 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 mutatingwebhookconfiguration.admissionregistration.k8s.io
mutating-webhook-configuration
  ${KUBECTL} delete validatingwebhookconfigurations.admissionregistration.k8s.io
validating-webhook-configuration
  ${KUBECTL} wait --for=delete namespace/cattle-provisioning-capi-system --
timeout=300s

```

```

fi

# Deploy CAPI
if [ ${KUBECTL} get pods -n ${METAL3_CAPISYSTEMNAMESPACE} -o name | wc -l) -lt
  1 ]; then

  # Try this command 3 times just in case, stolen from https://stackoverflow.com/
  a/33354419
  if ! (r=3; while ! /opt/mgmt/bin/clusterctl init \
    --core "cluster-api:v${METAL3_CAPICOREVERSION}"\
    --infrastructure "metal3:v${METAL3_CAPIMETAL3VERSION}"\
    --bootstrap "${METAL3_CAPIPROVIDER}:v${METAL3_CAPIRKE2VERSION}"\
    --control-plane "${METAL3_CAPIPROVIDER}:v${METAL3_CAPIRKE2VERSION}"\
    --config /root/cluster-api/clusterctl.yaml ; do
    ((--r))||exit
    echo "Something went wrong, let's wait 10 seconds and retry"
    sleep 10;done) ; then
    echo "clusterctl failed"
    exit 1
  fi

  # Wait for capi-controller-manager
  while ! ${KUBECTL} wait --for condition=ready -n ${METAL3_CAPISYSTEMNAMESPACE}
    ${KUBECTL} get pods -n ${METAL3_CAPISYSTEMNAMESPACE} -l cluster.x-k8s.io/
    provider=cluster-api -o name) --timeout=10s; do sleep 2 ; done

  # Wait for capm3-controller-manager, there are two pods, the ipam and the capm3
  one, just wait for the first one
  while ! ${KUBECTL} wait --for condition=ready -n ${METAL3_CAPM3NAMESPACE}
    ${KUBECTL} get pods -n ${METAL3_CAPM3NAMESPACE} -l cluster.x-k8s.io/
    provider=infrastructure-metal3 -o name | head -n1 ) --timeout=10s; do sleep 2 ; done

  # Wait for rke2-bootstrap-controller-manager
  while ! ${KUBECTL} wait --for condition=ready -n ${METAL3_RKE2BOOTSTRAPNAMESPACE}
    ${KUBECTL} get pods -n ${METAL3_RKE2BOOTSTRAPNAMESPACE} -l cluster.x-k8s.io/
    provider=bootstrap-rke2 -o name) --timeout=10s; do sleep 2 ; done

  # Wait for rke2-control-plane-controller-manager
  while ! ${KUBECTL} wait --for condition=ready -n
    ${METAL3_RKE2CONTROLPLANENAMESPACE} ${KUBECTL} get pods -n
    ${METAL3_RKE2CONTROLPLANENAMESPACE} -l cluster.x-k8s.io/provider=control-plane-rke2
    -o name) --timeout=10s; do sleep 2 ; done

fi

# Clean up the lock cm

```

```
`${KUBECTL}` delete configmap `${METAL3LOCKCMNAME}` -n `${METAL3LOCKNAMESPACE}`
```

- The `custom/files/airgap-resources.tar.gz` file must be included in the `custom/files` folder with all the resources needed to run the management cluster in an airgap environment. This file must be prepared manually downloading all resources and compressing them into this single file. The `airgap-resources.tar.gz` file contains the following resources:

```
|-- clusterctl
|-- clusterctl.yaml
|-- overrides
    |-- bootstrap-rke2
        |-- v0.4.1
            |-- bootstrap-components.yaml
            |-- metadata.yaml
    |-- cluster-api
        |-- v1.6.2
            |-- core-components.yaml
            |-- metadata.yaml
    |-- control-plane-rke2
        |-- v0.4.1
            |-- control-plane-components.yaml
            |-- metadata.yaml
    |-- infrastructure-metal3
        |-- v1.6.0
            |-- cluster-template.yaml
            |-- infrastructure-components.yaml
            |-- metadata.yaml
```

The `clusterctl.yaml` file contains the configuration to specify the images location and the overrides to be used by the `clusterctl` tool. The `overrides` folder contains `yaml` file manifests to be used instead of downloading them from the internet.

```
providers:
  # override a pre-defined provider
  - name: "cluster-api"
    url: "/root/cluster-api/overrides/cluster-api/v1.6.2/core-components.yaml"
    type: "CoreProvider"
  - name: "metal3"
    url: "/root/cluster-api/overrides/infrastructure-metal3/v1.6.0/infrastructure-
components.yaml"
    type: "InfrastructureProvider"
  - name: "rke2"
    url: "/root/cluster-api/overrides/bootstrap-rke2/v0.4.1/bootstrap-components.yaml"
    type: "BootstrapProvider"
```

```
- name: "rke2"
  url: "/root/cluster-api/overrides/control-plane-rke2/v0.4.1/control-plane-
components.yaml"
  type: "ControlPlaneProvider"
images:
  all:
    repository: registry.suse.com/edge
```

The `clusterctl` and the rest of the files included in the `overrides` folder can be downloaded using the following `curls` commands:

```
# clusterctl binary
curl -L https://github.com/kubernetes-sigs/cluster-api/releases/download/v1.6.2/
clusterctl-linux- $\{GOARCH\}$  -o /usr/local/bin/clusterct

# bootstrap-components (bootstrap-rke2)
curl -L https://github.com/rancher-sandbox/cluster-api-provider-rke2/releases/download/
v0.4.1/bootstrap-components.yaml
curl -L https://github.com/rancher-sandbox/cluster-api-provider-rke2/releases/download/
v0.4.1/metadata.yaml

# control-plane-components (control-plane-rke2)
curl -L https://github.com/rancher-sandbox/cluster-api-provider-rke2/releases/download/
v0.4.1/control-plane-components.yaml
curl -L https://github.com/rancher-sandbox/cluster-api-provider-rke2/releases/download/
v0.4.1/metadata.yaml

# cluster-api components
curl -L https://github.com/kubernetes-sigs/cluster-api/releases/download/v1.6.2/core-
components.yaml
curl -L https://github.com/kubernetes-sigs/cluster-api/releases/download/v1.6.2/
metadata.yaml

# infrastructure-components (infrastructure-metal3)
curl -L https://github.com/metal3-io/cluster-api-provider-metal3/releases/download/
v1.6.0/infrastructure-components.yaml
curl -L https://github.com/metal3-io/cluster-api-provider-metal3/releases/download/
v1.6.0/metadata.yaml
```



Note

If you want to use different versions of the components, you can change the version in the URL to download the specific version of the components.

With the previous resources downloaded, you can compress them into a single file using the following command:

```
tar -czvf airgap-resources.tar.gz clusterctl clusterctl.yaml overrides
```

29.5 Image creation

Once the directory structure is prepared following the previous sections (for both, connected and air-gap scenarios), run the following command to build the image:

```
podman run --rm --privileged -it -v $PWD:/eib \
registry.suse.com/edge/edge-image-builder:1.0.2 \
build --definition-file mgmt-cluster.yaml
```

This creates the ISO output image file that, in our case, based on the image definition described above, is eib-mgmt-cluster-image.iso.

29.6 Provision the management cluster

The previous image contains all components explained above, and it can be used to provision the management cluster using a virtual machine or a bare-metal server (using the virtual-media feature).

30 Telco features configuration

This section documents and explains the configuration of Telco-specific features on ATIP-deployed clusters.

The directed network provisioning deployment method is used, as described in the ATIP Automated Provision ([Chapter 31, Fully automated directed network provisioning](#)) section.

The following topics are covered in this section:

- Kernel image for real time ([Section 30.1, "Kernel image for real time"](#)): Kernel image to be used by the real-time kernel.
- CPU tuned configuration ([Section 30.2, "CPU tuned configuration"](#)): Tuned configuration to be used by the real-time kernel.
- CNI configuration ([Section 30.3, "CNI Configuration"](#)): CNI configuration to be used by the Kubernetes cluster.
- SR-IOV configuration ([Section 30.4, "SR-IOV"](#)): SR-IOV configuration to be used by the Kubernetes workloads.
- DPDK configuration ([Section 30.5, "DPDK"](#)): DPDK configuration to be used by the system.
- vRAN acceleration card ([Section 30.6, "vRAN acceleration \(Intel ACC100/ACC200\)"](#)): Acceleration card configuration to be used by the Kubernetes workloads.
- Huge pages ([Section 30.7, "Huge pages"](#)): Huge pages configuration to be used by the Kubernetes workloads.
- CPU pinning configuration ([Section 30.8, "CPU pinning configuration"](#)): CPU pinning configuration to be used by the Kubernetes workloads.
- NUMA-aware scheduling configuration ([Section 30.9, "NUMA-aware scheduling"](#)): NUMA-aware scheduling configuration to be used by the Kubernetes workloads.
- Metal LB configuration ([Section 30.10, "Metal LB"](#)): Metal LB configuration to be used by the Kubernetes workloads.
- Private registry configuration ([Section 30.11, "Private registry configuration"](#)): Private registry configuration to be used by the Kubernetes workloads.

30.1 Kernel image for real time

The real-time kernel image is not necessarily better than a standard kernel. It is a different kernel tuned to a specific use case. The real-time kernel is tuned for lower latency at the cost of throughput. The real-time kernel is not recommended for general purpose use, but in our case, this is the recommended kernel for Telco Workloads where latency is a key factor.

There are four top features:

- **Deterministic execution:**
Get greater predictability — ensure critical business processes complete in time, every time and deliver high-quality service, even under heavy system loads. By shielding key system resources for high-priority processes, you can ensure greater predictability for time-sensitive applications.
- **Low jitter:**
The low jitter built upon the highly deterministic technology helps to keep applications synchronized with the real world. This helps services that need ongoing and repeated calculation.
- **Priority inheritance:**
Priority inheritance refers to the ability of a lower priority process to assume a higher priority when there is a higher priority process that requires the lower priority process to finish before it can accomplish its task. SUSE Linux Enterprise Real Time solves these priority inversion problems for mission-critical processes.
- **Thread interrupts:**
Processes running in interrupt mode in a general-purpose operating system are not preemptible. With SUSE Linux Enterprise Real Time, these interrupts have been encapsulated by kernel threads, which are interruptible, and allow the hard and soft interrupts to be preempted by user-defined higher priority processes.
In our case, if you have installed a real-time image like [SLE Micro RT](#), kernel real time is already installed. From the [SUSE Customer Center \(https://scc.suse.com/\)](https://scc.suse.com/), you can download the real-time kernel image.



Note

For more information about the real-time kernel, visit [SUSE Real Time \(https://www.suse.com/products/realtime/\)](https://www.suse.com/products/realtime/).

30.2 CPU tuned configuration

The CPU Tuned configuration allows the possibility to isolate the CPU cores to be used by the real-time kernel. It is important to prevent the OS from using the same cores as the real-time kernel, because the OS could use the cores and increase the latency in the real-time kernel.

To enable and configure this feature, the first thing is to create a profile for the CPU cores we want to isolate. In this case, we are isolating the cores 1-30 and 33-62.

```
$ 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.
```

Then we need to modify the GRUB option to isolate CPU cores and other important parameters for CPU usage. The following options are important to be customized with your current hardware specifications:

parameter	value	description
isolcpus	1-30,33-62	Isolate the cores 1-30 and 33-62
skew_tick	1	This option allows the kernel to skew the timer interrupts across the isolated CPUs.
nohz	on	This option allows the kernel to run the timer tick on a single CPU when the system is idle.
nohz_full	1-30,33-62	kernel boot parameter is the current main interface to configure full dynticks along with CPU Isolation.

parameter	value	description
rcu_nocbs	1-30,33-62	This option allows the kernel to run the RCU callbacks on a single CPU when the system is idle.
kthread_cpus	0,31,32,63	This option allows the kernel to run the kthreads on a single CPU when the system is idle.
irqaffinity	0,31,32,63	This option allows the kernel to run the interrupts on a single CPU when the system is idle.
processor.max_cstate	1	This option prevents the CPU from dropping into a sleep state when idle
intel_idle.max_cstate	0	This option disables the intel_idle driver and allows acpi_idle to be used

With the values shown above, we are isolating 60 cores, and we are using four cores for the OS. The following commands modify the GRUB configuration and apply the changes mentioned above to be present on the next boot:

Edit the `/etc/default/grub` file and add the parameters mentioned above:

```
GRUB_CMDLINE_LINUX="intel_iommu=on intel_pstate=passive processor.max_cstate=1
intel_idle.max_cstate=0 iommu=pt usbcore.autosuspend=-1 selinux=0 enforcing=0
nmi_watchdog=0 crashkernel=auto softlockup_panic=0 audit=0 mce=off hugepagesz=1G
hugepages=40 hugepagesz=2M hugepages=0 default_hugepagesz=1G kthread_cpus=0,31,32,63
irqaffinity=0,31,32,63 isolcpus=1-30,33-62 skew_tick=1 nohz_full=1-30,33-62
rcu_nocbs=1-30,33-62 rcu_nocb_poll"
```

Update the GRUB configuration:

```
$ transactional-update grub.cfg
```

```
$ reboot
```

To validate that the parameters are applied after the reboot, the following command can be used to check the kernel command line:

```
$ cat /proc/cmdline
```

30.3 CNI Configuration

30.3.1 Cilium

Cilium is the default CNI plug-in for ATIP. To enable Cilium on RKE2 cluster as the default plug-in, the following configurations are required in the `/etc/rancher/rke2/config.yaml` file:

```
cni:  
- cilium
```

This can also be specified with command-line arguments, that is, `--cni=cilium` into the server line in `/etc/systemd/system/rke2-server` file.

To use the SR-IOV network operator described in the next section ([Section 30.4, “SR-IOV”](#) (page 365)), use Multus with another CNI plug-in, like Cilium or Calico, as a secondary plug-in.

```
cni:  
- multus  
- cilium
```



Note

For more information about CNI plug-ins, visit [Network Options \(https://docs.rke2.io/install/network_options\)](https://docs.rke2.io/install/network_options).

30.4 SR-IOV

SR-IOV allows a device, such as a network adapter, to separate access to its resources among various PCIe hardware functions. There are different ways to deploy SR-IOV, and here, we show two different options:

- Option 1: using the SR-IOV CNI device plug-ins and a config map to configure it properly.
- Option 2 (recommended): using the SR-IOV Helm chart from Rancher Prime to make this deployment easy.

Option 1 - Installation of SR-IOV CNI device plug-ins and a config map to configure it properly

- Prepare the config map for the device plug-in

Get the information to fill the config map from the lspci command:

```
$ 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)
```

The config map consists of a JSON file that describes devices using filters to discover, and creates groups for the interfaces. The key is understanding filters and groups. The filters are used to discover the devices and the groups are used to create the interfaces.

It could be possible to set filters:

- vendorID: 8086 (Intel)
- deviceID: 0d5c (Accelerator card)
- driver: vfio-pci (driver)
- pfNames: p2p1 (physical interface name)

It could be possible to also set filters to match more complex interface syntax, for example:

- pfNames: ["eth1#1,2,3,4,5,6"] or [eth1#1-6] (physical interface name)

Related to the groups, we could create a group for the FEC card and another group for the Intel card, even creating a prefix depending on our use case:

- resourceName: pci_sriov_net_bh_dpdk
- resourcePrefix: Rancher.io

There are a lot of combinations to discover and create the resource group to allocate some VFs to the pods.



Note

For more information about the filters and groups, visit [sr-ioV network device plug-in \(https://github.com/k8snetworkplumbingwg/sriov-network-device-plugin\)](https://github.com/k8snetworkplumbingwg/sriov-network-device-plugin).

After setting the filters and groups to match the interfaces depending on the hardware and the use case, the following config map shows an example to be used:

```
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"]
          }
        }
      ]
    }

```

- Prepare the `daemonset` file to deploy the device plug-in.

The device plug-in supports several architectures (`arm`, `amd`, `ppc64le`), so the same file can be used for different architectures deploying several `daemonset` for each architecture.

```

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:v3.5.1-build20231009-amd64
          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

```

- After applying the config map and the daemonset, the device plug-in will be deployed and the interfaces will be discovered and available for the pods.

```

$ kubectl get pods -n kube-system | grep sriov
kube-system kube-sriov-device-plugin-amd64-twjfl 1/1 Running 0 2m

```

- Check the interfaces discovered and available in the nodes to be used by the 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"
}

```

- The FEC is intel.com/intel_fec_5g and the value is 1.
- The VF is intel.com/intel_sriov_odu or intel.com/intel_sriov_oru if you deploy it with a device plug-in and the config map without Helm charts.

Important

If there are no interfaces here, it makes little sense to continue because the interface will not be available for pods. Review the config map and filters to solve the issue first.

Option 2 (recommended) - Installation using Rancher using Helm chart for SR-IOV CNI and device plug-ins

- Get Helm if not present:

```
$ curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
```

- Install SR-IOV.

This part could be done in two ways, using the [CLI](#) or using the [Rancher UI](#).

Install Operator from CLI

```
helm repo add suse-edge https://suse-edge.github.io/charts
helm install sriov-crd suse-edge/sriov-crd -n sriov-network-operator
helm install install sriov-network-operator suse-edge/sriov-network-operator -n
sriov-network-operator
```

Install Operator from Rancher UI

Once your cluster is installed, and you have access to the [Rancher UI](#), you can install the [SR-IOV Operator](#) from the [Rancher UI](#) from the apps tab:

Note

Make sure you select the right namespace to install the operator, for example, [sriov-network-operator](#).

+ image::features_sriov.png[sriov.png]

- Check the deployed resources crd and pods:

```
$ kubectl get crd
$ kubectl -n sriov-network-operator get pods
```


- Check the label in the nodes.

With all resources running, the label appears automatically in your node:

```
$ kubectl get nodes -oyaml | grep feature.node.kubernetes.io/network-sriov.capable
feature.node.kubernetes.io/network-sriov.capable: "true"
```

- Review the [daemonset](#) to see the new [sriov-network-config-daemon](#) and [sriov-rancher-nfd-worker](#) as active and ready:

```
$ 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
```

In a few minutes (can take up to 10 min to be updated), the nodes are detected and configured with the [SR-IOV](#) capabilities:

```
$ kubectl get sriovnetworknodestates.sriovnetwork.openshift.io -A
NAMESPACE          NAME      AGE
sriov-network-operator  xr11-2   83s
```

- Check the interfaces detected.

The interfaces discovered should be the PCI address of the network device. Check this information with the [lspci](#) command in the 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: ""

```



Note

If your interface is not detected here, ensure that it is present in the next config map:

```
$ kubectl get cm supported-nic-ids -oyaml -n sriov-network-operator
```

If your device is not there, edit the config map, adding the right values to be discovered (should be necessary to restart the `sriov-network-config-daemon` daemonset).

- Create the `NetworkNode Policy` to configure the `VFs`.

Some `VFs` (`numVfs`) from the device (`rootDevices`) will be created, and it will be configured with the driver `deviceType` and the `MTU`:



Note

The `resourceName` field must not contain any special characters and must be unique across the cluster. The example uses the `deviceType: vfio-pci` because `dpdk` will be used in combination with `sr-iov`. If you don't use `dpdk`, the `deviceType` should be `deviceType: netdevice` (default value).

```
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
```

- Validate configurations:

```
$ 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/intelnicDpdk": "8"
}

```

- Create the sr-iov network (optional, just in case a different network is needed):

```

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: intelnicDpdk

```

- Check the network created:

```

$ 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/intelnicDpdk
    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": "0.3.1", "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: ""
```

30.5 DPDK

DPDK (Data Plane Development Kit) is a set of libraries and drivers for fast packet processing. It is used to accelerate packet processing workloads running on a wide variety of CPU architectures. The DPDK includes data plane libraries and optimized network interface controller (NIC) drivers for the following:

1. A queue manager implements lockless queues.
2. A buffer manager pre-allocates fixed size buffers.
3. A memory manager allocates pools of objects in memory and uses a ring to store free objects; ensures that objects are spread equally on all DRAM channels.
4. Poll mode drivers (PMD) are designed to work without asynchronous notifications, reducing overhead.
5. A packet framework as a set of libraries that are helpers to develop packet processing.

The following steps will show how to enable DPDK and how to create VFs from the NICs to be used by the DPDK interfaces:

- Install the DPDK package:

```
$ transactional-update pkg install dpdk22 dpdk22-tools libdpdk-23  
$ reboot
```

- Kernel parameters:

To use DPDK, employ some drivers to enable certain parameters in the kernel:

parameter	value	description
iommu	pt	This option enables the use of the <u>vfio</u> driver for the DPDK interfaces.
intel_iommu	on	This option enables the use of <u>vfio</u> for <u>VFs</u> .

To enable the parameters, add them to the /etc/default/grub file:

```
GRUB_CMDLINE_LINUX="intel_iommu=on intel_pstate=passive processor.max_cstate=1
intel_idle.max_cstate=0 iommu=pt usbcore.autosuspend=-1 selinux=0 enforcing=0
nmi_watchdog=0 crashkernel=auto softlockup_panic=0 audit=0 mce=off hugepagesz=1G
hugepages=40 hugepagesz=2M hugepages=0 default_hugepagesz=1G kthread_cpus=0,31,32,63
irqaffinity=0,31,32,63 isolcpus=1-30,33-62 skew_tick=1 nohz_full=1-30,33-62
rcu_nocbs=1-30,33-62 rcu_nocb_poll"
```

Update the GRUB configuration and reboot the system to apply the changes:

```
$ transactional-update grub.cfg
$ reboot
```

- Load vfio-pci kernel module and enable SR-IOV on the NICs:

```
$ modprobe vfio-pci enable_sriov=1 disable_idle_d3=1
```

- Create some virtual functions (VFs) from the NICs.

To create for VFs, for example, for two different NICs, the following commands are required:

```
$ echo 4 > /sys/bus/pci/devices/0000:51:00.0/sriov_numvfs
$ echo 4 > /sys/bus/pci/devices/0000:51:00.1/sriov_numvfs
```

- Bind the new VFs with the vfio-pci driver:

```
$ 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
```

- Review the configuration is correctly applied:

```

$ 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

```

30.6 vRAN acceleration (Intel ACC100/ACC200)

As communications service providers move from 4 G to 5 G networks, many are adopting virtualized radio access network (vRAN) architectures for higher channel capacity and easier deployment of edge-based services and applications. vRAN solutions are ideally located to deliver low-latency services with the flexibility to increase or decrease capacity based on the volume of real-time traffic and demand on the network.

One of the most compute-intensive 4 G and 5 G workloads is RAN layer 1 (L1) FEC, which resolves data transmission errors over unreliable or noisy communication channels. FEC technology detects and corrects a limited number of errors in 4 G or 5 G data, eliminating the need for retransmission. Since the FEC acceleration transaction does not contain cell state information, it can be easily virtualized, enabling pooling benefits and easy cell migration.

- Kernel parameters

To enable the vRAN acceleration, we need to enable the following kernel parameters (if not present yet):

parameter	value	description
iommu	pt	This option enables the use of vfio for the DPDK interfaces.
intel_iommu	on	This option enables the use of vfio for VFs.

Modify the GRUB file /etc/default/grub to add them to the kernel command line:

```
GRUB_CMDLINE_LINUX="intel_iommu=on intel_pstate=passive processor.max_cstate=1
intel_idle.max_cstate=0 iommu=pt usbcore.autosuspend=-1 selinux=0 enforcing=0
nmi_watchdog=0 crashkernel=auto softlockup_panic=0 audit=0 mce=off hugepagesz=1G
hugepages=40 hugepagesz=2M hugepages=0 default_hugepagesz=1G kthread_cpus=0,31,32,63
irqaffinity=0,31,32,63 isolcpus=1-30,33-62 skew_tick=1 nohz_full=1-30,33-62
rcu_nocbs=1-30,33-62 rcu_nocb_poll"
```

Update the GRUB configuration and reboot the system to apply the changes:

```
$ transactional-update grub.cfg
$ reboot
```

To verify that the parameters are applied after the reboot, check the command line:

```
$ cat /proc/cmdline
```

- Load vfio-pci kernel modules to enable the vRAN acceleration:

```
$ modprobe vfio-pci enable_sriov=1 disable_idle_d3=1
```

- Get interface information Acc100:

```
$ lspci | grep -i acc
8a:00.0 Processing accelerators: Intel Corporation Device 0d5c
```

- Bind the physical interface (PF) with vfio-pci driver:

```
$ dpdk-devbind.py -b vfio-pci 0000:8a:00.0
```


- Create the virtual functions (VFs) from the physical interface (PF).

Create 2 VFs from the PF and bind with `vfio-pci` following the next steps:

```
$ echo 2 > /sys/bus/pci/devices/0000:8a:00.0/sriov_numvfs
$ dpdk-devbind.py -b vfio-pci 0000:8b:00.0
```

- Configure acc100 with the proposed configuration file:

```
$ 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!
```

- Check the new VFs created from the FEC PF:

```
$ 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=
```

30.7 Huge pages

When a process uses RAM, the CPU marks it as used by that process. For efficiency, the CPU allocates RAM in chunks 4K bytes is the default value on many platforms. Those chunks are named pages. Pages can be swapped to disk, etc.

Since the process address space is virtual, the CPU and the operating system need to remember which pages belong to which process, and where each page is stored. The greater the number of pages, the longer the search for memory mapping. When a process uses 1 GB of memory, that is 262144 entries to look up (1 GB / 4 K). If a page table entry consumes 8 bytes, that is 2 MB (262144 * 8) to look up.

Most current CPU architectures support larger-than-default pages, which give the CPU/OS fewer entries to look up.

- Kernel parameters

To enable the huge pages, we should add the next kernel parameters:

parameter	value	description
hugepagesz	1G	This option allows to set the size of huge pages to 1 G
hugepages	40	This is the number of huge pages defined before
default_hugepagesz	1G	This is the default value to get the huge pages

Modify the GRUB file `/etc/default/grub` to add them to the kernel command line:

```
GRUB_CMDLINE_LINUX="intel_iommu=on intel_pstate=passive processor.max_cstate=1
intel_idle.max_cstate=0 iommu=pt usbcore.autosuspend=-1 selinux=0 enforcing=0
nmi_watchdog=0 crashkernel=auto softlockup_panic=0 audit=0 mce=off hugepagesz=1G
hugepages=40 hugepagesz=2M hugepages=0 default_hugepagesz=1G kthread_cpus=0,31,32,63
irqaffinity=0,31,32,63 isolcpus=1-30,33-62 skew_tick=1 nohz_full=1-30,33-62
rcu_nocbs=1-30,33-62 rcu_nocb_poll"
```

Update the GRUB configuration and reboot the system to apply the changes:

```
$ transactional-update grub.cfg
$ reboot
```

To validate that the parameters are applied after the reboot, you can check the command line:

```
$ cat /proc/cmdline
```

- Using huge pages

To use the huge pages, we need to mount them:

```
$ mkdir -p /hugepages
$ mount -t hugetlbfs nodev /hugepages
```

Deploy a Kubernetes workload, creating the resources and the 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
...

```

30.8 CPU pinning configuration

- Requirements

1. Must have the CPU tuned to the performance profile covered in this section ([Section 30.2, "CPU tuned configuration"](#)).
2. Must have the RKE2 cluster kubelet configured with the CPU management arguments adding the following block (as an example) to the /etc/rancher/rke2/config.yaml file:

```

kubelet-arg:
  - "cpu-manager=true"
  - "cpu-manager-policy=static"
  - "cpu-manager-policy-options=full-pcpus-only=true"
  - "cpu-manager-reconcile-period=0s"
  - "kubelet-reserved=cpu=1"
  - "system-reserved=cpu=1"

```

- Using CPU pinning on Kubernetes

There are three ways to use that feature using the Static Policy defined in kubelet depending on the requests and limits you define on your workload:

1. BestEffort QoS Class: If you do not define any request or limit for CPU, the pod is scheduled on the first CPU available on the system.

An example of using the BestEffort QoS Class could be:

```
spec:
  containers:
  - name: nginx
    image: nginx
```

2. Burstable QoS Class: If you define a request for CPU, which is not equal to the limits, or there is no CPU request.

Examples of using the Burstable QoS Class could be:

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
      requests:
        memory: "100Mi"
```

or

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
        cpu: "2"
      requests:
        memory: "100Mi"
        cpu: "1"
```

3. Guaranteed QoS Class: If you define a request for CPU, which is equal to the limits.

An example of using the Guaranteed QoS Class could be:

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
        cpu: "2"
      requests:
        memory: "200Mi"
        cpu: "2"
```

30.9 NUMA-aware scheduling

Non-Uniform Memory Access or Non-Uniform Memory Architecture (NUMA) is a physical memory design used in SMP (multiprocessors) architecture, where the memory access time depends on the memory location relative to a processor. Under NUMA, a processor can access its own local memory faster than non-local memory, that is, memory local to another processor or memory shared between processors.

30.9.1 Identifying NUMA nodes

To identify the NUMA nodes, on your system use the following command:

```
$ lscpu | grep NUMA
NUMA node(s):                1
NUMA node0 CPU(s):           0-63
```



Note

For this example, we have only one NUMA node showing 64 CPUs.

NUMA needs to be enabled in the BIOS. If `dmesg` does not have records of NUMA initialization during the bootup, then NUMA-related messages in the kernel ring buffer might have been overwritten.

30.10 Metal LB

MetaLB is a load-balancer implementation for bare-metal Kubernetes clusters, using standard routing protocols like L2 and BGP as advertisement protocols. It is a network load balancer that can be used to expose services in a Kubernetes cluster to the outside world due to the need to use Kubernetes Services type LoadBalancer with bare-metal.

To enable MetaLB in the RKE2 cluster, the following steps are required:

- Install MetaLB using the following command:

```
$ kubectl apply <<EOF -f
apiVersion: helm.cattle.io/v1
kind: HelmChart
metadata:
  name: metallb
  namespace: kube-system
spec:
  repo: https://metallb.github.io/metallb/
  chart: metallb
  targetNamespace: metallb-system
---
apiVersion: helm.cattle.io/v1
kind: HelmChart
metadata:
  name: endpoint-copier-operator
  namespace: kube-system
spec:
  repo: https://suse-edge.github.io/endpoint-copier-operator
  chart: endpoint-copier-operator
  targetNamespace: endpoint-copier-operator
EOF
```

- Create the IpAddressPool and the L2advertisement configuration:

```
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
```

- Create the endpoint service to expose the 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
```

- Check the VIP is created and the MetaLB pods are running:

```
$ kubectl get svc -n default
$ kubectl get pods -n default
```

30.11 Private registry configuration

Containerd can be configured to connect to private registries and use them to pull private images on each node.

Upon startup, RKE2 checks if a `registries.yaml` file exists at `/etc/rancher/rke2/` and instructs `containerd` to use any registries defined in the file. If you wish to use a private registry, create this file as root on each node that will use the registry.

To add the private registry, create the file `/etc/rancher/rke2/registries.yaml` with the following content:

```
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
```

or without authentication:

```
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
```

For the registry changes to take effect, you need to either configure this file before starting RKE2 on the node, or restart RKE2 on each configured node.



Note

For more information about this, please check [containerd registry configuration rke2](https://docs.rke2.io/install/containerd_registry_configuration_rke2) (https://docs.rke2.io/install/containerd_registry_configuration#registries-configuration-file) [↗](#).

31 Fully automated directed network provisioning

31.1 Introduction

Directed network provisioning is a feature that allows you to automate the provisioning of downstream clusters. This feature is useful when you have many downstream clusters to provision, and you want to automate the process.

A management cluster ([Chapter 29, Setting up the management cluster](#)) automates deployment of the following components:

- [SUSE Linux Enterprise Micro RT](#) as the OS. Depending on the use case, configurations like networking, storage, users and kernel arguments can be customized.
- [RKE2](#) as the Kubernetes cluster. The default [CNI plug-in](#) is [Cilium](#). Depending on the use case, certain [CNI plug-ins](#) can be used, such as [Cilium+Multus](#).
- [Longhorn](#) as the storage solution.
- [NeuVector](#) as the security solution.
- [MetalLB](#) can be used as the load balancer for highly available multi-node clusters.



Note

For more information about [SUSE Linux Enterprise Micro](#), see [Chapter 7, SLE Micro](#) For more information about [RKE2](#), see [Chapter 14, RKE2](#) For more information about [Longhorn](#), see [Chapter 15, Longhorn](#) For more information about [NeuVector](#), see [Chapter 16, NeuVector](#)

The following sections describe the different directed network provisioning workflows and some additional features that can be added to the provisioning process:

- [Section 31.2, "Prepare downstream cluster image for connected scenarios"](#)
- [Section 31.3, "Prepare downstream cluster image for air-gap scenarios"](#)
- [Section 31.4, "Downstream cluster provisioning with Directed network provisioning \(single-node\)"](#)
- [Section 31.5, "Downstream cluster provisioning with Directed network provisioning \(multi-node\)"](#)
- [Section 31.6, "Advanced Network Configuration"](#)
- [Section 31.7, "Telco features \(DPDK, SR-IOV, CPU isolation, huge pages, NUMA, etc.\)"](#)

- [Section 31.8, “Private registry”](#)
- [Section 31.9, “Downstream cluster provisioning in air-gapped scenarios”](#)

31.2 Prepare downstream cluster image for connected scenarios

Edge Image Builder ([Chapter 9, Edge Image Builder](#)) is used to prepare a modified SLEMicro base image which is provisioned on downstream cluster hosts.

Much of the configuration via Edge Image Builder is possible, but in this guide, we cover the minimal configurations necessary to set up the downstream cluster.

31.2.1 Prerequisites for connected scenarios

- A container runtime such as [Podman \(https://podman.io\)](https://podman.io) or [Rancher Desktop \(https://rancherdesktop.io\)](https://rancherdesktop.io) is required to run Edge Image Builder.
- The base image [SLE-Micro.x86_64-5.5.0-Default-RT-GM.raw](#) must be downloaded from the [SUSE Customer Center \(https://scc.suse.com/\)](https://scc.suse.com/) or the [SUSE Download page \(https://www.suse.com/download/sle-micro/\)](https://www.suse.com/download/sle-micro/).

31.2.2 Image configuration for connected scenarios

When running Edge Image Builder, a directory is mounted from the host, so it is necessary to create a directory structure to store the configuration files used to define the target image.

- [downstream-cluster-config.yaml](#) is the image definition file, see [Chapter 3, Standalone clusters with Edge Image Builder](#) for more details.
- The base image when downloaded is `xz` compressed, which must be uncompressed with `unxz` and copied/moved under the [base-images](#) folder.
- The [network](#) folder is optional, see [Section 31.2.2.4, “Additional script for Advanced Network Configuration”](#) for more details.
- The [custom/scripts](#) directory contains scripts to be run on first-boot; currently a [01-fix-growfs.sh](#) script is required to resize the OS root partition on deployment

```
├─ downstream-cluster-config.yaml
├─ base-images/
│   └─ SLE-Micro.x86_64-5.5.0-Default-RT-GM.raw
├─ network/
│   └─ configure-network.sh
└─ custom/
    └─ scripts/
        └─ 01-fix-growfs.sh
```

31.2.2.1 Downstream cluster image definition file

The `downstream-cluster-config.yaml` file is the main configuration file for the downstream cluster image. The following is a minimal example for deployment via Metal³:

```
apiVersion: 1.0
image:
  imageType: RAW
  arch: x86_64
  baseImage: SLE-Micro.x86_64-5.5.0-Default-RT-GM.raw
  outputImageName: eibimage-slemicro55rt-telco.raw
operatingSystem:
  kernelArgs:
    - ignition.platform.id=openstack
    - net.ifnames=1
  systemd:
    disable:
      - rebootmgr
  users:
    - username: root
      encryptedPassword: ${ROOT_PASSWORD}
      sshKeys:
        - ${USERKEY1}
```

`${ROOT_PASSWORD}` is the encrypted password for the root user, which can be useful for test/debugging. It can be generated with the `openssl passwd -6 PASSWORD` command

For the production environments, it is recommended to use the SSH keys that can be added to the users block replacing the `${USERKEY1}` with the real SSH keys.



Note

`net.ifnames=1` enables [Predictable Network Interface Naming](https://documentation.suse.com/smart/network/html/network-interface-predictable-naming/index.html) (<https://documentation.suse.com/smart/network/html/network-interface-predictable-naming/index.html>) 

This matches the default configuration for the metal3 chart, but the setting must match the configured chart `predictableNicNames` value.

Also note `ignition.platform.id=openstack` is mandatory, without this argument SLEMicro configuration via ignition will fail in the Metal³ automated flow.

31.2.2.2 Growfs script

Currently, a custom script (`custom/scripts/01-fix-growfs.sh`) is required to grow the file system to match the disk size on first-boot after provisioning. The `01-fix-growfs.sh` script contains the following information:

```
#!/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 /
```



Note

Add your own custom scripts to be executed during the provisioning process using the same approach. For more information, see [Chapter 3, Standalone clusters with Edge Image Builder](#).

31.2.2.3 Additional configuration for Telco workloads

To enable Telco features like `dpdk`, `sr-io` or `FEC`, additional packages may be required as shown in the following example.

```
apiVersion: 1.0
```

```

image:
  imageType: RAW
  arch: x86_64
  baseImage: SLE-Micro.x86_64-5.5.0-Default-RT-GM.raw
  outputImageName: eibimage-slemicro55rt-telco.raw
operatingSystem:
  kernelArgs:
    - ignition.platform.id=openstack
    - net.ifnames=1
  systemd:
    disable:
      - rebootmgr
  users:
    - username: root
      encryptedPassword: ${ROOT_PASSWORD}
      sshKeys:
        - ${user1Key1}
  packages:
    packageList:
      - jq
      - dpdk22
      - dpdk22-tools
      - libdpdk-23
      - pf-bb-config
    additionalRepos:
      - url: https://download.opensuse.org/repositories/isv:/SUSE:/Edge:/Telco/
SLEMicro5.5/
  sccRegistrationCode: ${SCC_REGISTRATION_CODE}

```

Where `${SCC_REGISTRATION_CODE}` is the registration code copied from [SUSE Customer Center \(https://scc.suse.com/\)](https://scc.suse.com/), and the package list contains the minimum packages to be used for the Telco profiles. To use the `pf-bb-config` package (to enable the `FEC` feature and binding with drivers), the `additionalRepos` block must be included to add the `SUSE Edge Telco` repository.

31.2.2.4 Additional script for Advanced Network Configuration

If you need to configure static IPs or more advanced networking scenarios as described in [Section 31.6, “Advanced Network Configuration”](#), the following additional configuration is required.

In the `network` folder, create the following `configure-network.sh` file - this consumes configuration drive data on first-boot, and configures the host networking using the [NM Configurator tool \(https://github.com/suse-edge/nm-configurator\)](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

```

31.2.3 Image creation

Once the directory structure is prepared following the previous sections, run the following command to build the image:

```

podman run --rm --privileged -it -v $PWD:/eib \
registry.suse.com/edge/edge-image-builder:1.0.2 \
build --definition-file downstream-cluster-config.yaml

```

This creates the output ISO image file named eibimage-slemicro55rt-telco.raw, based on the definition described above.

The output image must then be made available via a webserver, either the media-server container enabled via the Management Cluster Documentation (*Note*) or some other locally accessible server. In the examples below, we refer to this server as `imagecache.local:8080`

31.3 Prepare downstream cluster image for air-gap scenarios

Edge Image Builder (*Chapter 9, Edge Image Builder*) is used to prepare a modified SLEMicro base image which is provisioned on downstream cluster hosts.

Much of the configuration is possible with Edge Image Builder, but in this guide, we cover the minimal configurations necessary to set up the downstream cluster for air-gap scenarios.

31.3.1 Prerequisites for air-gap scenarios

- A container runtime such as Podman (<https://podman.io>) or Rancher Desktop (<https://rancherdesktop.io>) is required to run Edge Image Builder.
- The base image `SLE-Micro.x86_64-5.5.0-Default-RT-GM.raw` must be downloaded from the SUSE Customer Center (<https://scc.suse.com/>) or the SUSE Download page (<https://www.suse.com/download/sle-micro/>).
- If you want to use SR-IOV or any other workload which require a container image, a local private registry must be deployed and already configured (with/without TLS and/or authentication). This registry will be used to store the images and the helm chart OCI images.

31.3.2 Image configuration for air-gap scenarios

When running Edge Image Builder, a directory is mounted from the host, so it is necessary to create a directory structure to store the configuration files used to define the target image.

- `downstream-cluster-airgap-config.yaml` is the image definition file, see *Chapter 3, Standalone clusters with Edge Image Builder* for more details.
- The base image when downloaded is `xz` compressed, which must be uncompressed with `unxz` and copied/moved under the `base-images` folder.

- The `network` folder is optional, see [Section 31.2.2.4, “Additional script for Advanced Network Configuration”](#) for more details.
- The `custom/scripts` directory contains scripts to be run on first-boot; currently a `01-fix-growfs.sh` script is required to resize the OS root partition on deployment. For air-gap scenarios, a script `02-airgap.sh` is required to copy the images to the right place during the image creation process.
- The `custom/files` directory contains the `rke2` and the `cni` images to be copied to the image during the image creation process.

```

├─ downstream-cluster-airgap-config.yaml
├─ base-images/
│   └─ SLE-Micro.x86_64-5.5.0-Default-RT-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
    └─ scripts/
        ├── 01-fix-growfs.sh
        └─ 02-airgap.sh

```

31.3.2.1 Downstream cluster image definition file

The `downstream-cluster-airgap-config.yaml` file is the main configuration file for the downstream cluster image and the content has been described in the previous section ([Section 31.2.2.3, “Additional configuration for Telco workloads”](#)).

31.3.2.2 Growfs script

Currently, a custom script (`custom/scripts/01-fix-growfs.sh`) is required to grow the file system to match the disk size on first-boot after provisioning. The `01-fix-growfs.sh` script contains the following information:

```
#!/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 /

```

31.3.2.3 Air-gap script

The following script (`custom/scripts/02-airgap.sh`) is required to copy the images to the right place during the image creation process:

```

#!/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/

```

31.3.2.4 Custom files for air-gap scenarios

The `custom/files` directory contains the `rke2` and the `cni` images to be copied to the image during the image creation process. To easily generate the images, prepare them locally using following script (<https://github.com/suse-edge/fleet-examples/blob/release-3.0/scripts/day2/edge-save-rke2-images.sh>) and the list of images [here](https://github.com/suse-edge/fleet-examples/blob/release-3.0/scripts/day2/edge-release-rke2-images.txt) to generate the artifacts required to be included in `custom/files`. Also, you can download the latest `rke2-install` script from [here](https://get.rke2.io/).

```

$ ./edge-save-rke2-images.sh -o custom/files -l ~/edge-release-rke2-images.txt

```

After downloading the images, the directory structure should look like this:

```
└─ 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
```

31.3.2.5 Preload your private registry with images required for air-gap scenarios and SR-IOV (optional)

If you want to use SR-IOV in your air-gap scenario or any other workload images, you must preload your local private registry with the images following the next steps:

- Download, extract, and push the helm-chart OCI images to the private registry
- Download, extract, and push the rest of images required to the private registry

The following scripts can be used to download, extract, and push the images to the private registry. We will show an example to preload the SR-IOV images, but you can also use the same approach to preload any other custom images:

1. Preload with helm-chart OCI images for SR-IOV:

a. You must create a list with the helm-chart OCI images required:

```
$ cat > edge-release-helm-oci-artifacts.txt <<EOF
edge/sriov-network-operator-chart:1.2.2
edge/sriov-crd-chart:1.2.2
EOF
```

b. Generate a local tarball file using the following script (<https://github.com/suse-edge/fleet-examples/blob/release-3.0/scripts/day2/edge-save-oci-artefacts.sh>) and the list created above:

```
$ ./edge-save-oci-artefacts.sh -al ./edge-release-helm-oci-artifacts.txt -s
registry.suse.com
Pulled: registry.suse.com/edge/sriov-network-operator-chart:1.2.2
```

```
Pulled: registry.suse.com/edge/sriov-crd-chart:1.2.2
a edge-release-oci-tgz-20240705
a edge-release-oci-tgz-20240705/sriov-network-operator-chart-1.2.2.tgz
a edge-release-oci-tgz-20240705/sriov-crd-chart-1.2.2.tgz
```

- c. Upload your tarball file to your private registry (e.g. `myregistry:5000`) using the following script (<https://github.com/suse-edge/fleet-examples/blob/release-3.0/scripts/day2/edge-load-oci-artefacts.sh>) to preload your registry with the helm chart OCI images downloaded in the previous step:

```
$ tar zxvf edge-release-oci-tgz-20240705.tgz
$ ./edge-load-oci-artefacts.sh -ad edge-release-oci-tgz-20240705 -r
myregistry:5000
```

2. Preload with the rest of the images required for SR-IOV:

- a. In this case, we must include the ``sr-iov`` container images for telco workloads (e.g. as a reference, you could get them from [helm-chart values](https://github.com/suse-edge/charts/blob/release-3.0/charts/sriov-network-operator/1.2.2%2Bup0.1.0/values.yaml) (<https://github.com/suse-edge/charts/blob/release-3.0/charts/sriov-network-operator/1.2.2%2Bup0.1.0/values.yaml>))

```
$ cat > edge-release-images.txt <<EOF
rancher/hardened-sriov-network-operator:v1.2.0-build20240327
rancher/hardened-sriov-network-config-daemon:v1.2.0-build20240327
rancher/hardened-sriov-cni:v2.7.0-build20240327
rancher/hardened-ib-sriov-cni:v1.0.3-build20240327
rancher/hardened-sriov-network-device-plugin:v3.6.2-build20240327
rancher/hardened-sriov-network-resources-injector:v1.5-build20240327
rancher/hardened-sriov-network-webhook:v1.2.0-build20240327
EOF
```

- b. Using the following script (<https://github.com/suse-edge/fleet-examples/blob/release-3.0/scripts/day2/edge-save-images.sh>) and the list created above, you must generate locally the tarball file with the images required:

```
$ ./edge-save-images.sh -l ./edge-release-images.txt -s registry.suse.com
Image pull success: registry.suse.com/rancher/hardened-sriov-network-
operator:v1.2.0-build20240327
Image pull success: registry.suse.com/rancher/hardened-sriov-network-config-
daemon:v1.2.0-build20240327
Image pull success: registry.suse.com/rancher/hardened-sriov-cni:v2.7.0-
build20240327
Image pull success: registry.suse.com/rancher/hardened-ib-sriov-cni:v1.0.3-
build20240327
```

```
Image pull success: registry.suse.com/rancher/hardened-sriov-network-device-
plugin:v3.6.2-build20240327
Image pull success: registry.suse.com/rancher/hardened-sriov-network-resources-
injector:v1.5-build20240327
Image pull success: registry.suse.com/rancher/hardened-sriov-network-
webhook:v1.2.0-build20240327
Creating edge-images.tar.gz with 7 images
```

- c. Upload your tarball file to your private registry (e.g. `myregistry:5000`) using the following [script \(https://github.com/suse-edge/fleet-examples/blob/release-3.0/scripts/day2/edge-load-images.sh\)](https://github.com/suse-edge/fleet-examples/blob/release-3.0/scripts/day2/edge-load-images.sh) to preload your private registry with the images downloaded in the previous step:

```
$ tar zxvf edge-release-images-tgz-20240705.tgz
$ ./edge-load-images.sh -ad edge-release-images-tgz-20240705 -r myregistry:5000
```

31.3.3 Image creation for air-gap scenarios

Once the directory structure is prepared following the previous sections, run the following command to build the image:

```
podman run --rm --privileged -it -v $PWD:/eib \
registry.suse.com/edge/edge-image-builder:1.0.2 \
build --definition-file downstream-cluster-airgap-config.yaml
```

This creates the output ISO image file named `eibimage-slemicro55rt-telco.raw`, based on the definition described above.

The output image must then be made available via a webserver, either the `media-server` container enabled via the [Management Cluster Documentation](#) (*Note*) or some other locally accessible server. In the examples below, we refer to this server as `imagecache.local:8080`.

31.4 Downstream cluster provisioning with Directed network provisioning (single-node)

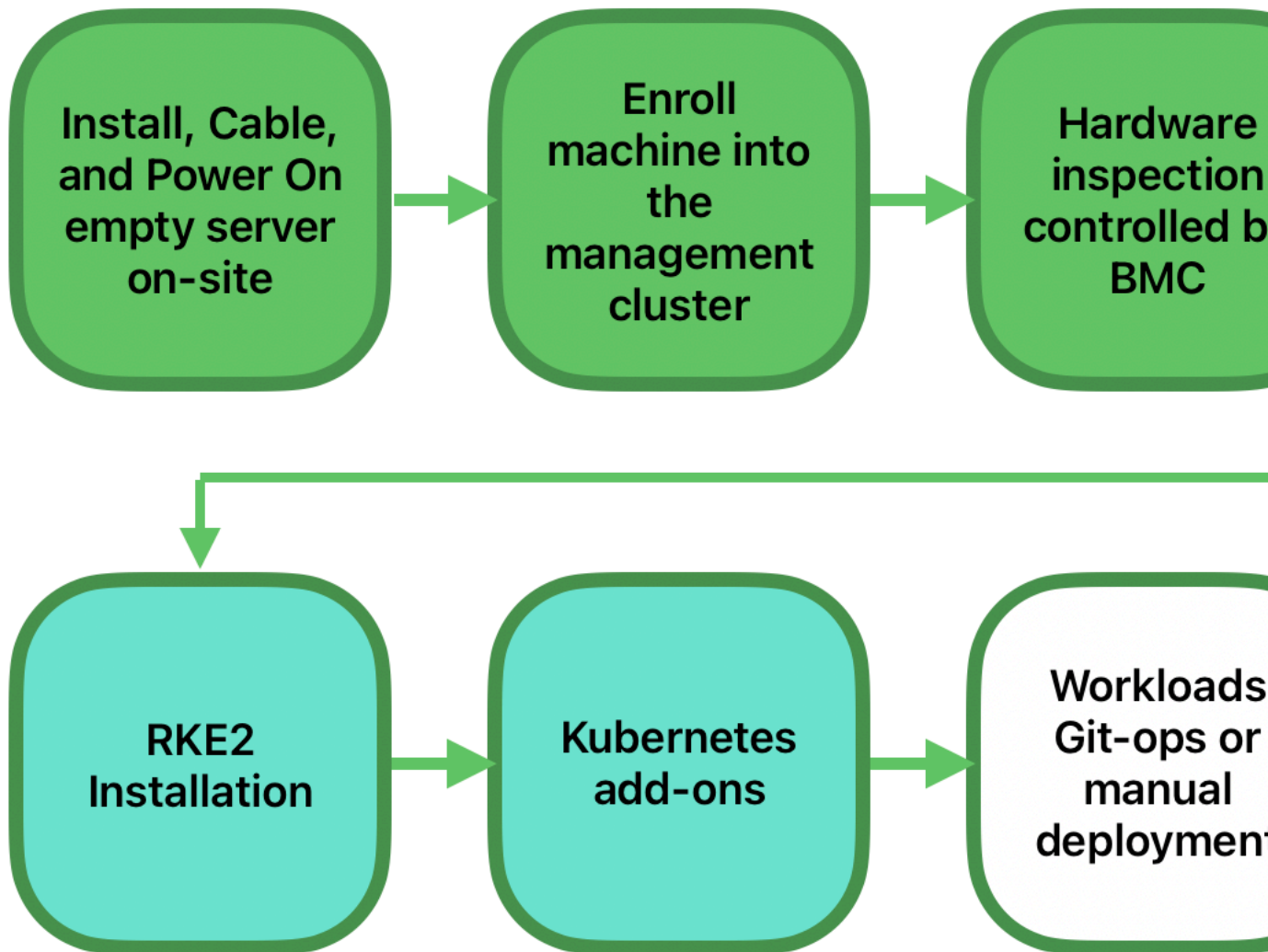
This section describes the workflow used to automate the provisioning of a single-node downstream cluster using directed network provisioning. This is the simplest way to automate the provisioning of a downstream cluster.

Requirements

- The image generated using [EIB](#), as described in the previous section ([Section 31.2, “Prepare downstream cluster image for connected scenarios”](#)), with the minimal configuration to set up the downstream cluster has to be located in the management cluster exactly on the path you configured on this section (*Note*).
- The management server created and available to be used on the following sections. For more information, refer to the Management Cluster section [Chapter 29, Setting up the management cluster](#).

Workflow

The following diagram shows the workflow used to automate the provisioning of a single-node downstream cluster using directed network provisioning:



There are two different steps to automate the provisioning of a single-node downstream cluster using directed network provisioning:

1. Enroll the bare-metal host to make it available for the provisioning process.
2. Provision the bare-metal host to install and configure the operating system and the Kubernetes cluster.

Enroll the bare-metal host

The first step is to enroll the new bare-metal host in the management cluster to make it available to be provisioned. To do that, the following file (`bmh-example.yaml`) has to be created in the management cluster, to specify the BMC credentials to be used and the BaremetalHost object to be enrolled:

```
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: flexran-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
```

where:

- `${BMC_USERNAME}` — The user name for the BMC of the new bare-metal host.
- `${BMC_PASSWORD}` — The password for the BMC of the new bare-metal host.

- `${BMC_MAC}` — The MAC address of the new bare-metal host to be used.
- `${BMC_ADDRESS}` — The URL for the bare-metal host BMC (for example, `redfish-virtual-media://192.168.200.75/redfish/v1/Systems/1/`). To learn more about the different options available depending on your hardware provider, check the following [link \(https://github.com/metal3-io/baremetal-operator/blob/main/docs/api.md\)](https://github.com/metal3-io/baremetal-operator/blob/main/docs/api.md).

Once the file is created, the following command has to be executed in the management cluster to start enrolling the new bare-metal host in the management cluster:

```
$ kubectl apply -f bmh-example.yaml
```

The new bare-metal host object will be enrolled, changing its state from `registering` to `inspecting` and `available`. The changes can be checked using the following command:

```
$ kubectl get bmh
```



Note

The `BaremetalHost` object is in the `registering` state until the BMC credentials are validated. Once the credentials are validated, the `BaremetalHost` object changes its state to `inspecting`, and this step could take some time depending on the hardware (up to 20 minutes). During the `inspecting` phase, the hardware information is retrieved and the Kubernetes object is updated. Check the information using the following command: `kubectl get bmh -o yaml`.

Provision step

Once the bare-metal host is enrolled and available, the next step is to provision the bare-metal host to install and configure the operating system and the Kubernetes cluster. To do that, the following file (`capi-provisioning-example.yaml`) has to be created in the management-cluster with the following information (the `capi-provisioning-example.yaml` can be generated by joining the following blocks).



Note

Only values between `${...}` must be replaced with the real values.

The following block is the cluster definition, where the networking can be configured using the `Pods` and the `Services` blocks. Also, it contains the references to the control plane and the infrastructure (using the `Metal3` provider) objects to be used.

```
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/v1alpha1
    kind: RKE2ControlPlane
    name: single-node-cluster
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3Cluster
    name: single-node-cluster
```

The `Metal3Cluster` object specifies the control-plane endpoint (replacing the `DOWNSTREAM_CONTROL_PLANE_IP`) to be configured and the `noCloudProvider` because a bare-metal node is used.

```
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
```

The `RKE2ControlPlane` object specifies the control-plane configuration to be used and the `Metal3MachineTemplate` object specifies the control-plane image to be used. Also, it contains the information about the number of replicas to be used (in this case, one) and the `CNI` plug-in to be used (in this case, `Cilium`). The `agentConfig` block contains the `Ignition` format to be used

and the `additionalUserData` to be used to configure the `RKE2` node with information like a `systemd` named `rke2-preinstall.service` to replace automatically the `BAREMETALHOST_UUID` and `node-name` during the provisioning process using the `Ironic` information. The last block of information contains the `Kubernetes` version to be used. `${RKE2_VERSION}` is the version of `RKE2` to be used replacing this value (for example, `v1.28.13+rke2r1`).

```
apiVersion: controlplane.cluster.x-k8s.io/v1alpha1
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
  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
```

```

kubelet:
  extraArgs:
    - provider-id=metal3://BAREMETALHOST_UUID
  version: ${RKE2_VERSION}
  nodeName: "localhost.localdomain"

```

The `Metal3MachineTemplate` object specifies the following information:

- The `dataTemplate` to be used as a reference to the template.
- The `hostSelector` to be used matching with the label created during the enrollment process.
- The `image` to be used as a reference to the image generated using `EIB` on the previous section (*Section 31.2, "Prepare downstream cluster image for connected scenarios"*), and the `checksum` and `checksumType` to be used to validate the image.

```

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-slemicro55rt-telco.raw.sha256
        checksumType: sha256
        format: raw
        url: http://imagecache.local:8080/eibimage-slemicro55rt-telco.raw

```

The `Metal3DataTemplate` object specifies the `metaData` for the downstream cluster.

```

apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3DataTemplate
metadata:
  name: multinode-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
```

Once the file is created by joining the previous blocks, the following command must be executed in the management cluster to start provisioning the new bare-metal host:

```
$ kubectl apply -f capi-provisioning-example.yaml
```

31.5 Downstream cluster provisioning with Directed network provisioning (multi-node)

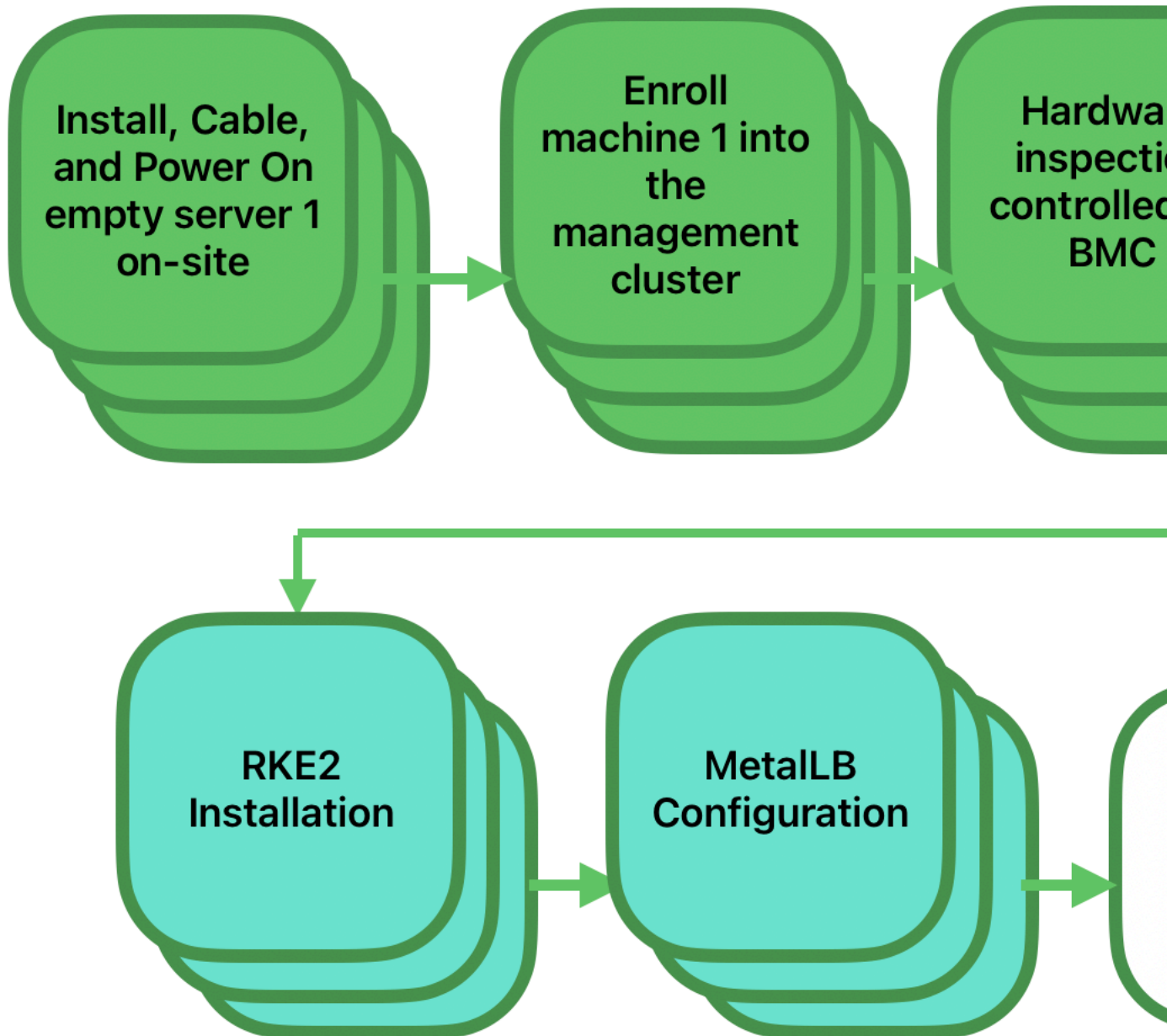
This section describes the workflow used to automate the provisioning of a multi-node downstream cluster using directed network provisioning and [MetalLB](#) as a load-balancer strategy. This is the simplest way to automate the provisioning of a downstream cluster. The following diagram shows the workflow used to automate the provisioning of a multi-node downstream cluster using directed network provisioning and [MetalLB](#).

Requirements

- The image generated using [EIB](#), as described in the previous section ([Section 31.2, “Prepare downstream cluster image for connected scenarios”](#)), with the minimal configuration to set up the downstream cluster has to be located in the management cluster exactly on the path you configured on this section ([Note](#)).
- The management server created and available to be used on the following sections. For more information, refer to the Management Cluster section: [Chapter 29, Setting up the management cluster](#).

Workflow

The following diagram shows the workflow used to automate the provisioning of a multi-node downstream cluster using directed network provisioning:



1. Enroll the three bare-metal hosts to make them available for the provisioning process.
2. Provision the three bare-metal hosts to install and configure the operating system and the Kubernetes cluster using MetaLB.

Enroll the bare-metal hosts

The first step is to enroll the three bare-metal hosts in the management cluster to make them available to be provisioned. To do that, the following files (`bmh-example-node1.yaml`, `bmh-example-node2.yaml` and `bmh-example-node3.yaml`) must be created in the management cluster, to specify the BMC credentials to be used and the `BareMetalHost` object to be enrolled in the management cluster.



Note

- Only the values between ``${...}`` have to be replaced with the real values.
- We will walk you through the process for only one host. The same steps apply to the other two nodes.

```
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
```

Where:

- ``${BMC_NODE1_USERNAME}`` — The username for the BMC of the first bare-metal host.
- ``${BMC_NODE1_PASSWORD}`` — The password for the BMC of the first bare-metal host.

- `${BMC_NODE1_MAC}` — The MAC address of the first bare-metal host to be used.
- `${BMC_NODE1_ADDRESS}` — The URL for the first bare-metal host BMC (for example, `redfish-virtualmedia://192.168.200.75/redfish/v1/Systems/1/`). To learn more about the different options available depending on your hardware provider, check the following [link \(https://github.com/metal3-io/baremetal-operator/blob/main/docs/api.md\)](https://github.com/metal3-io/baremetal-operator/blob/main/docs/api.md).

Once the file is created, the following command must be executed in the management cluster to start enrolling the bare-metal hosts in the management cluster:

```
$ kubectl apply -f bmh-example-node1.yaml
$ kubectl apply -f bmh-example-node2.yaml
$ kubectl apply -f bmh-example-node3.yaml
```

The new bare-metal host objects are enrolled, changing their state from `registering` to `inspecting` and `available`. The changes can be checked using the following command:

```
$ kubectl get bmh -o wide
```



Note

The `BaremetalHost` object is in the `registering` state until the `BMC` credentials are validated. Once the credentials are validated, the `BaremetalHost` object changes its state to `inspecting`, and this step could take some time depending on the hardware (up to 20 minutes). During the `inspecting` phase, the hardware information is retrieved and the Kubernetes object is updated. Check the information using the following command: `kubectl get bmh -o yaml`.

Provision step

Once the three bare-metal hosts are enrolled and available, the next step is to provision the bare-metal hosts to install and configure the operating system and the Kubernetes cluster, creating a load balancer to manage them. To do that, the following file (`capi-provisioning-example.yaml`) must be created in the management cluster with the following information (the `capi-provisioning-example.yaml` can be generated by joining the following blocks).



Note

- Only values between `_${...}_` must be replaced with the real values.
- The `VIP` address is a reserved IP address that is not assigned to any node and is used to configure the load balancer.

Below is the cluster definition, where the cluster network can be configured using the `Pods` and the `services` blocks. Also, it contains the references to the control plane and the infrastructure (using the `Metal3` provider) objects to be used.

```
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/v1alpha1
    kind: RKE2ControlPlane
    name: multinode-cluster
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3Cluster
    name: multinode-cluster
```

The `Metal3Cluster` object specifies the control-plane endpoint that uses the `VIP` address already reserved (replacing the `_${DOWNSTREAM_VIP_ADDRESS}_`) to be configured and the `noCloudProvider` because the three bare-metal nodes are used.

```
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
```

The `RKE2ControlPlane` object specifies the control-plane configuration to be used, and the `Metal3MachineTemplate` object specifies the control-plane image to be used.

- The number of replicas to be used (in this case, three).
- The advertisement mode to be used by the Load Balancer (`address` uses the L2 implementation), as well as the address to be used (replacing the `${EDGE_VIP_ADDRESS}` with the `VIP` address).
- The `serverConfig` with the `CNI` plug-in to be used (in this case, `Cilium`), and the `tlsSan` to be used to configure the `VIP` address.
- The `agentConfig` block contains the `Ignition` format to be used and the `additionalUserData` to be used to configure the `RKE2` node with information like:
 - The `systemd` service named `rke2-preinstall.service` to replace automatically the `BAREMETALHOST_UUID` and `node-name` during the provisioning process using the `Iron` information.
 - The `storage` block which contains the Helm charts to be used to install the `MetalLB` and the `endpoint-copier-operator`.
 - The `metalLB` custom resource file with the `IPaddressPool` and the `L2Advertisement` to be used (replacing `${EDGE_VIP_ADDRESS}` with the `VIP` address).
 - The `endpoint-svc.yaml` file to be used to configure the `kubernetes-vip` service to be used by the `MetalLB` to manage the `VIP` address.
- The last block of information contains the `Kubernetes` version to be used. The `${RKE2_VERSION}` is the version of `RKE2` to be used replacing this value (for example, `v1.28.13+rke2r1`).

```
apiVersion: controlplane.cluster.x-k8s.io/v1alpha1
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
registrationMethod: "address"
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:
            - 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/endpoint-copier-operator-chart
  targetNamespace: endpoint-copier-operator
  version: 0.2.0
  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/metallb-chart
        targetNamespace: metallb-system
        version: 0.14.3
        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
kubenet:
  extraArgs:
    - provider-id=metal3://BAREMETALHOST_UUID
version: ${RKE2_VERSION}
nodeName: "Node-multinode-cluster"

```

The `Metal3MachineTemplate` object specifies the following information:

- The `dataTemplate` to be used as a reference to the template.
- The `hostSelector` to be used matching with the label created during the enrollment process.
- The `image` to be used as a reference to the image generated using `EIB` on the previous section (*Section 31.2, "Prepare downstream cluster image for connected scenarios"*), and `checksum` and `checksumType` to be used to validate the image.

```

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-slemicro55rt-telco.raw.sha256
      checksumType: sha256
      format: raw
      url: http://imagecache.local:8080/eibimage-slemicro55rt-telco.raw

```

The `Metal3DataTemplate` object specifies the `metaData` for the downstream cluster.

```

apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3DataTemplate
metadata:
  name: multinode-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

```

Once the file is created by joining the previous blocks, the following command has to be executed in the management cluster to start provisioning the new three bare-metal hosts:

```
$ kubectl apply -f capi-provisioning-example.yaml
```

31.6 Advanced Network Configuration

The directed network provisioning workflow allows downstream clusters network configurations such as static IPs, bonding, VLAN's, etc.

The following sections describe the additional steps required to enable provisioning downstream clusters using advanced network configuration.

Requirements

- The image generated using [EIB](#) has to include the network folder and the script following this section ([Section 31.2.2.4, "Additional script for Advanced Network Configuration"](#)).

Configuration

Use the following two sections as the base to enroll and provision the hosts:

- Downstream cluster provisioning with Directed network provisioning (single-node) ([Section 31.4, "Downstream cluster provisioning with Directed network provisioning \(single-node\)"](#))
- Downstream cluster provisioning with Directed network provisioning (multi-node) ([Section 31.5, "Downstream cluster provisioning with Directed network provisioning \(multi-node\)"](#))

The changes required to enable the advanced network configuration are the following:

- Enrollment step: The following new example file with a secret containing the information about the `networkData` to be used to configure, for example, the static `IPs` and `VLAN` for the downstream cluster

```
apiVersion: v1
kind: Secret
metadata:
  name: controlplane-0-networkdata
type: Opaque
stringData:
  networkData: |
    interfaces:
      - name: ${CONTROLPLANE_INTERFACE}
        type: ethernet
        state: up
        mtu: 1500
        mac-address: "${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}

```

This file contains the `networkData` in a `nmstate` format to be used to configure the advance network configuration (for example, static IPs and VLAN) for the downstream cluster. As you can see, the example shows the configuration to enable the interface with static IPs, as well as the configuration to enable the VLAN using the base interface. Any other `nmstate` example could be defined to be used to configure the network for the downstream cluster to adapt to the specific requirements, where the following variables have to be replaced with real values:

- `${CONTROLPLANE1_INTERFACE}` — The control-plane interface to be used for the edge cluster (for example, `eth0`).
- `${CONTROLPLANE1_IP}` — The IP address to be used as an endpoint for the edge cluster (must match with the `kubeapi-server` endpoint).
- `${CONTROLPLANE1_PREFIX}` — The CIDR to be used for the edge cluster (for example, `24` if you want `/24` or `255.255.255.0`).
- `${CONTROLPLANE1_GATEWAY}` — The gateway to be used for the edge cluster (for example, `192.168.100.1`).
- `${CONTROLPLANE1_MAC}` — The MAC address to be used for the control-plane interface (for example, `00:0c:29:3e:3e:3e`).
- `${DNS_SERVER}` — The DNS to be used for the edge cluster (for example, `192.168.100.2`).
- `${VLAN_ID}` — The VLAN ID to be used for the edge cluster (for example, `100`).

Also, the reference to that secret using `preprovisioningNetworkDataName` is needed in the `BaremetalHost` object at the end of the file to be enrolled in the management cluster.

```

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: flexran-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

```



Note

If you need to deploy a multi-node cluster, the same process must be done for the other nodes.

- Provision step: The block of information related to the network data has to be removed because the platform includes the network data configuration into the secret `controlplane-0-networkdata`.

```

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

```




Note

The `Metal3DataTemplate`, `networkData` and `Metal3 IPAM` are currently not supported; only the configuration via static secrets is fully supported.

31.7 Telco features (DPDK, SR-IOV, CPU isolation, huge pages, NUMA, etc.)

The directed network provisioning workflow allows to automate the Telco features to be used in the downstream clusters to run Telco workloads on top of those servers.

Requirements

- The image generated using `EIB` has to include the specific Telco packages following this section (*Section 31.2.2.3, "Additional configuration for Telco workloads"*).
- The image generated using `EIB`, as described in the previous section (*Section 31.2, "Prepare downstream cluster image for connected scenarios"*), has to be located in the management cluster exactly on the path you configured on this section (*Note*).
- The management server created and available to be used on the following sections. For more information, refer to the Management Cluster section: *Chapter 29, Setting up the management cluster*.

Configuration

Use the following two sections as the base to enroll and provision the hosts:

- Downstream cluster provisioning with Directed network provisioning (single-node) (*Section 31.4, "Downstream cluster provisioning with Directed network provisioning (single-node)"*)
- Downstream cluster provisioning with Directed network provisioning (multi-node) (*Section 31.5, "Downstream cluster provisioning with Directed network provisioning (multi-node)"*)

The Telco features covered in this section are the following:

- DPDK and VFs creation
- SR-IOV and VFs allocation to be used by the workloads

- CPU isolation and performance tuning
- Huge pages configuration
- Kernel parameters tuning



Note

For more information about the Telco features, see [Chapter 30, Telco features configuration](#).

The changes required to enable the Telco features shown above are all inside the `RKE2ControlPlane` block in the provision file `capi-provisioning-example.yaml`. The rest of the information inside the file `capi-provisioning-example.yaml` is the same as the information provided in the provisioning section ([Section 31.4, “Downstream cluster provisioning with Directed network provisioning \(single-node\)”](#) (page 397)).

To make the process clear, the changes required on that block (`RKE2ControlPlane`) to enable the Telco features are the following:

- The `preRKE2Commands` to be used to execute the commands before the `RKE2` installation process. In this case, use the `modprobe` command to enable the `vfio-pci` and the `SR-IOV` kernel modules.
- The ignition file `/var/lib/rancher/rke2/server/manifests/configmap-sriov-custom-auto.yaml` to be used to define the interfaces, drivers and the number of `VFs` to be created and exposed to the workloads.
 - The values inside the config map `sriov-custom-auto-config` are the only values to be replaced with real values.
 - `${RESOURCE_NAME1}` — The resource name to be used for the first `PF` interface (for example, `sriov-resource-du1`). It is added to the prefix `rancher.io` to be used as a label to be used by the workloads (for example, `rancher.io/sriov-resource-du1`).
 - `${SRIOV-NIC-NAME1}` — The name of the first `PF` interface to be used (for example, `eth0`).
 - `${PF_NAME1}` — The name of the first physical function `PF` to be used. Generate more complex filters using this (for example, `eth0#2-5`).

- `${DRIVER_NAME1}` — The driver name to be used for the first VF interface (for example, `vfio-pci`).
- `${NUM_VFS1}` — The number of VFs to be created for the first PF interface (for example, `8`).
- The `/var/sriov-auto-filler.sh` to be used as a translator between the high-level config map `sriov-custom-auto-config` and the `sriovnetworknodepolicy` which contains the low-level hardware information. This script has been created to abstract the user from the complexity to know in advance the hardware information. No changes are required in this file, but it should be present if we need to enable `sr-iov` and create VFs.
- The kernel arguments to be used to enable the following features:

Parameter	Value	Description
<code>isolcpus</code>	<code>1-30,33-62</code>	Isolate the cores 1-30 and 33-62.
<code>skew_tick</code>	<code>1</code>	Allows the kernel to skew the timer interrupts across the isolated CPUs.
<code>nohz</code>	<code>on</code>	Allows the kernel to run the timer tick on a single CPU when the system is idle.
<code>nohz_full</code>	<code>1-30,33-62</code>	kernel boot parameter is the current main interface to configure full dynticks along with CPU Isolation.
<code>rcu_nocbs</code>	<code>1-30,33-62</code>	Allows the kernel to run the RCU callbacks on a single CPU when the system is idle.
<code>kthread_cpus</code>	<code>0,31,32,63</code>	Allows the kernel to run the kthreads on a single CPU when the system is idle.

<code>irqaffinity</code>	0,31,32,63	Allows the kernel to run the interrupts on a single CPU when the system is idle.
<code>processor.max_cstate</code>	1	Prevents the CPU from dropping into a sleep state when idle.
<code>intel_idle.max_cstate</code>	0	Disables the <code>intel_idle</code> driver and allows <code>acpi_idle</code> to be used.
<code>iommu</code>	pt	Allows to use <code>vfio</code> for the <code>dpdk</code> interfaces.
<code>intel_iommu</code>	on	Enables the use of <code>vfio</code> for VFs.
<code>hugepagesz</code>	1G	Allows to set the size of huge pages to 1 G.
<code>hugepages</code>	40	Number of huge pages defined before.
<code>default_hugepagesz</code>	1G	Default value to enable huge pages.

- The following `systemd` services are used to enable the following:
 - `rke2-preinstall.service` to replace automatically the `BAREMETALHOST_UUID` and `node-name` during the provisioning process using the Ironic information.
 - `cpu-performance.service` to enable the CPU performance tuning. The `#{CPU_FREQUENCY}` has to be replaced with the real values (for example, `2500000` to set the CPU frequency to `2.5GHz`).

- `cpu-partitioning.service` to enable the isolation cores of the CPU (for example, 1-30,33-62).
- `sriov-custom-auto-vfs.service` to install the sriov Helm chart, wait until custom resources are created and run the `/var/sriov-auto-filler.sh` to replace the values in the config map `sriov-custom-auto-config` and create the `sriovnetwork-knodepolicy` to be used by the workloads.
- The `${RKE2_VERSION}` is the version of RKE2 to be used replacing this value (for example, `v1.28.13+rke2r1`).

With all these changes mentioned, the `RKE2ControlPlane` block in the `capi-provisioning-example.yaml` will look like the following:

```
apiVersion: controlplane.cluster.x-k8s.io/v1alpha1
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
  serverConfig:
    cni: cilium
    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/sriov-crd-chart
        targetNamespace: sriov-network-operator
        version: 1.2.2
        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/sriov-network-operator-chart
        targetNamespace: sriov-network-operator
        version: 1.2.2
        createNamespace: true
- path: /var/sriov-auto-filler.sh
  overwrite: true
  contents:
    inline: |
        #!/bin/bash
        cat <<- EOF > /var/sriov-networkpolicy-template.yaml
        apiVersion: sriovnetwork.openshift.io/v1
        kind: SriovNetworkNodePolicy
        metadata:
            name: atip-RESOURCE_NAME
            namespace: sriov-network-operator
        spec:
            nodeSelector:
                feature.node.kubernetes.io/network-sriov.capable: "true"
            resourceName: RESOURCE_NAME
            deviceType: DRIVER
            numVfs: NUMVF
            mtu: 1500
            nicSelector:
                pfNames: ["PFNAMES"]
                deviceID: "DEVICEID"
                vendor: "VENDOR"
                rootDevices:
                    - PCIADDRESS
        EOF

        export KUBECONFIG=/etc/rancher/rke2/rke2.yaml; export KUBECTL=/var/lib/
rancher/rke2/bin/kubectl
        while [ ${KUBECTL} --kubeconfig=${KUBECONFIG} get
sriovnetworknodestates.sriovnetwork.openshift.io -n sriov-network-operator -ojson | jq -
r '.items[].status.syncStatus') != "Succeeded" ]; do sleep 1; done
        input=${KUBECTL} --kubeconfig=${KUBECONFIG} get cm sriov-custom-auto-
config -n kube-system -ojson | jq -r '.data."config.json"'
        jq -c '.[[]] <<< $input | while read i; do
            interface=$(echo $i | jq -r '.interface')
            pfname=$(echo $i | jq -r '.pfname')
            pciaddress=${KUBECTL} --kubeconfig=${KUBECONFIG} get
sriovnetworknodestates.sriovnetwork.openshift.io -n sriov-network-operator -ojson | jq -
r ".items[].status.interfaces[]|select(.name==\"$interface\")|.pciAddress")

```

```

        vendor=$((${KUBECTL} --kubeconfig=${KUBECONFIG} get
sriovnetworknodestates.sriovnetwork.openshift.io -n sriov-network-operator -ojson | jq -
r ".items[].status.interfaces[]|select(.name==\"$interface\")|.vendor")
        deviceid=$((${KUBECTL} --kubeconfig=${KUBECONFIG} get
sriovnetworknodestates.sriovnetwork.openshift.io -n sriov-network-operator -ojson | jq -
r ".items[].status.interfaces[]|select(.name==\"$interface\")|.deviceID")
        resourceName=$(echo $i | jq -r '.resourceName')
        driver=$(echo $i | jq -r '.driver')
        sed -e "s/RESOURCENAME/$resourceName/g" \
            -e "s/DRIVER/$driver/g" \
            -e "s/PFNAMES/$pfname/g" \
            -e "s/VENDOR/$vendor/g" \
            -e "s/DEVICEID/$deviceid/g" \
            -e "s/PCIADDRESS/$pciaddress/g" \
            -e "s/NUMVF/$(echo $i | jq -r '.numVFsToCreate')/g" /var/sriov-
networkpolicy-template.yaml > /var/lib/rancher/rke2/server/manifests/$resourceName.yaml
    done
    mode: 0755
    user:
      name: root
    group:
      name: root
  kernel_arguments:
    should_exist:
      - intel_iommu=on
      - intel_pstate=passive
      - processor.max_cstate=1
      - intel_idle.max_cstate=0
      - iommu=pt
      - mce=off
      - hugepagesz=1G hugepages=40
      - hugepagesz=2M hugepages=0
      - default_hugepagesz=1G
      - kthread_cpus=${NON-ISOLATED_CPU_CORES}
      - irqaffinity=${NON-ISOLATED_CPU_CORES}
      - isolcpus=${ISOLATED_CPU_CORES}
      - nohz_full=${ISOLATED_CPU_CORES}
      - rcu_nocbs=${ISOLATED_CPU_CORES}
      - rcu_nocb_poll
      - nosoftlockup
      - nohz=on
  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-performance.service
enabled: true
contents: |
[Unit]
Description=CPU perfomance
Wants=network-online.target
After=network.target network-online.target
[Service]
User=root
Type=forking
TimeoutStartSec=900
ExecStart=/bin/sh -c "cpupower frequency-set -g performance; cpupower
frequency-set -u ${CPU_FREQUENCY}; cpupower frequency-set -d ${CPU_FREQUENCY}"
RemainAfterExit=yes
KillMode=process
[Install]
WantedBy=multi-user.target
- name: cpu-partitioning.service
enabled: true
contents: |
[Unit]
Description=cpu-partitioning
Wants=network-online.target
After=network.target network-online.target
[Service]
Type=oneshot
User=root
ExecStart=/bin/sh -c "echo isolated_cores=${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: 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 --
kubecfg=/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 --kubecfg=/etc/rancher/rke2/rke2.yaml get
sriovnetworkstates.sriovnetwork.openshift.io --ignore-not-found --no-headers -A | wc
-l) -eq 0 ]; do sleep 1; done"
    ExecStartPost=/bin/sh -c "/var/sriov-auto-filler.sh"
    RemainAfterExit=yes
    KillMode=process
    [Install]
    WantedBy=multi-user.target

  kubelet:
    extraArgs:
      - provider-id=metal3://BAREMETALHOST_UUID
    version: ${RKE2_VERSION}
    nodeName: "localhost.localdomain"

```

Once the file is created by joining the previous blocks, the following command must be executed in the management cluster to start provisioning the new downstream cluster using the Telco features:

```
$ kubectl apply -f capi-provisioning-example.yaml
```

31.8 Private registry

It is possible to configure a private registry as a mirror for images used by workloads.

To do this we create the secret containing the information about the private registry to be used by the downstream cluster.

```

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}

```

The `tls.crt`, `tls.key` and `ca.crt` are the certificates to be used to authenticate the private registry. The `username` and `password` are the credentials to be used to authenticate the private registry.



Note

The `tls.crt`, `tls.key`, `ca.crt`, `username` and `password` have to be encoded in base64 format before to be used in the secret.

With all these changes mentioned, the `RKE2ControlPlane` block in the `capi-provisioning-example.yaml` will look like the following:

```

apiVersion: controlplane.cluster.x-k8s.io/v1alpha1
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
  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: 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
    kubelet:
      extraArgs:
        - provider-id=metal3://BAREMETALHOST_UUID
      version: ${RKE2_VERSION}

```

```
nodeName: "localhost.localdomain"
```

Where the `registry.example.com` is the example name of the private registry to be used by the downstream cluster, and it should be replaced with the real values.

31.9 Downstream cluster provisioning in air-gapped scenarios

The directed network provisioning workflow allows to automate the provisioning of downstream clusters in air-gapped scenarios.

31.9.1 Requirements for air-gapped scenarios

1. The raw image generated using EIB must include the specific container images (helm-chart OCI and container images) required to run the downstream cluster in an air-gapped scenario. For more information, refer to this section (*Section 31.3, "Prepare downstream cluster image for air-gap scenarios"*).
2. In case of using SR-IOV or any other custom workload, the images required to run the workloads must be preloaded in your private registry following the preload private registry section (*Section 31.3.2.5, "Preload your private registry with images required for air-gap scenarios and SR-IOV (optional)"*).

31.9.2 Enroll the bare-metal hosts in air-gap scenarios

The process to enroll the bare-metal hosts in the management cluster is the same as described in the previous section (*Section 31.4, "Downstream cluster provisioning with Directed network provisioning (single-node)"* (page 396)).

31.9.3 Provision the downstream cluster in air-gap scenarios

There are some important changes required to provision the downstream cluster in air-gapped scenarios:

1. The `RKE2ControlPlane` block in the `capi-provisioning-example.yaml` file must include the `spec.agentConfig.airGapped: true` directive.
2. The private registry configuration must be included in the `RKE2ControlPlane` block in the `capi-provisioning-airgap-example.yaml` file following the private registry section ([Section 31.8, "Private registry"](#)).
3. If you are using SR-IOV or any other `AdditionalUserData` configuration (combustion script) which requires the helm-chart installation, you must modify the content to reference the private registry instead of using the public registry.

The following example shows the SR-IOV configuration in the `AdditionalUserData` block in the `capi-provisioning-airgap-example.yaml` file with the modifications required to reference the private registry

- Private Registry secrets references
- Helm-Chart definition using the private registry instead of the public OCI images.

```
# 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/v1alpha1
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
  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: cilium
    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: ${SRIOV_CRD_VERSION}
  ---
apiVersion: helm.cattle.io/v1
kind: HelmChart
metadata:
  name: sriov-network-operator
  namespace: kube-system
spec:
  chart: oci://${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: ${SRIOV_OPERATOR_VERSION}
mode: 0644
user:
  name: root
group:
  name: root
- path: /var/sriov-auto-filler.sh
  overwrite: true
  contents:
    inline: |
      #!/bin/bash
      cat <<- EOF > /var/sriov-networkpolicy-template.yaml
      apiVersion: sriovnetwork.openshift.io/v1
      kind: SriovNetworkNodePolicy
      metadata:
        name: atip-RESOURCE_NAME
        namespace: sriov-network-operator
      spec:
        nodeSelector:
          feature.node.kubernetes.io/network-sriov.capable: "true"
        resourceName: RESOURCE_NAME
        deviceType: DRIVER
        numVfs: NUMVF
        mtu: 1500
        nicSelector:
          pfNames: ["PFNAMES"]
          deviceID: "DEVICEID"
          vendor: "VENDOR"
          rootDevices:
            - PCIADDRESS
      EOF

      export KUBECONFIG=/etc/rancher/rke2/rke2.yaml; export KUBECTL=/var/lib/
rancher/rke2/bin/kubectl
      while [ ${KUBECTL} --kubeconfig=${KUBECONFIG} get
sriovnetworknodestates.sriovnetwork.openshift.io -n sriov-network-operator -ojson | jq -
r '.items[].status.syncStatus') != "Succeeded" ]; do sleep 1; done
      input=${KUBECTL} --kubeconfig=${KUBECONFIG} get cm sriov-custom-auto-
config -n sriov-network-operator -ojson | jq -r '.data."config.json"'
      jq -c '.[[]] <<< $input | while read i; do
interface=$(echo $i | jq -r '.interface')
pfname=$(echo $i | jq -r '.pfname')

```

```

        pciaddress=$((${KUBECTL} --kubeconfig=${KUBECONFIG} get
sriovnetworknodestates.sriovnetwork.openshift.io -n sriov-network-operator -ojson | jq -
r ".items[].status.interfaces[]|select(.name==\"$interface\")|.pciAddress")
        vendor=$((${KUBECTL} --kubeconfig=${KUBECONFIG} get
sriovnetworknodestates.sriovnetwork.openshift.io -n sriov-network-operator -ojson | jq -
r ".items[].status.interfaces[]|select(.name==\"$interface\")|.vendor")
        deviceid=$((${KUBECTL} --kubeconfig=${KUBECONFIG} get
sriovnetworknodestates.sriovnetwork.openshift.io -n sriov-network-operator -ojson | jq -
r ".items[].status.interfaces[]|select(.name==\"$interface\")|.deviceID")
        resourceName=$(echo $i | jq -r '.resourceName')
        driver=$(echo $i | jq -r '.driver')
        sed -e "s/RESOURCENAME/$resourceName/g" \
            -e "s/DRIVER/$driver/g" \
            -e "s/PFNAME/$pfname/g" \
            -e "s/VENDOR/$vendor/g" \
            -e "s/DEVICEID/$deviceid/g" \
            -e "s/PCIADDRESS/$pciaddress/g" \
            -e "s/NUMVF/$(echo $i | jq -r '.numVFsToCreate')/g" /var/sriov-
networkpolicy-template.yaml > /var/lib/rancher/rke2/server/manifests/$resourceName.yaml
    done
    mode: 0755
    user:
      name: root
    group:
      name: root
  kernel_arguments:
    should_exist:
      - intel_iommu=on
      - intel_pstate=passive
      - processor.max_cstate=1
      - intel_idle.max_cstate=0
      - iommu=pt
      - mce=off
      - hugepagesz=1G hugepages=40
      - hugepagesz=2M hugepages=0
      - default_hugepagesz=1G
      - kthread_cpus=${NON-ISOLATED_CPU_CORES}
      - irqaffinity=${NON-ISOLATED_CPU_CORES}
      - isolcpus=${ISOLATED_CPU_CORES}
      - nohz_full=${ISOLATED_CPU_CORES}
      - rcu_nocbs=${ISOLATED_CPU_CORES}
      - rcu_nocb_poll
      - nosoftlockup
      - nohz=on
  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: 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 --
kubecfg=/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 --kubecfg=/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 "/var/sriov-auto-filler.sh"
        RemainAfterExit=yes
        KillMode=process
        [Install]
        WantedBy=multi-user.target
kubelet:
  extraArgs:
    - provider-id=metal3://BAREMETALHOST_UUID
  version: ${RKE2_VERSION}
  nodeName: "localhost.localdomain"

```

32 Lifecycle actions

This section covers the lifecycle management actions of deployed ATIP clusters.

32.1 Management cluster upgrades

The upgrade of the management cluster involves several components. For a list of the general components that require an upgrade, see the [Day 2 management cluster](#) (*Chapter 24, Management Cluster*) documentation.

The upgrade procedure for components specific to this setup can be seen below.

Upgrading Metal³

To upgrade Metal³, use the following command to update the Helm repository cache and fetch the latest chart to install Metal³ from the Helm chart repository:

```
helm repo update
helm fetch suse-edge/metal3
```

After that, the easy way to upgrade is to export your current configurations to a file, and then upgrade the Metal³ version using that previous file. If any change is required in the new version, the file can be edited before the 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=0.7.4
```

32.2 Downstream cluster upgrades

Upgrading downstream clusters involves updating several components. The following sections cover the upgrade process for each of the components.

Upgrading the operating system

For this process, check the following reference ([Section 31.2, "Prepare downstream cluster image for connected scenarios"](#)) to build the new image with a new operating system version. With this new image generated by EIB, the next provision phase uses the new operating version provided. In the following step, the new image is used to upgrade the nodes.

Upgrading the RKE2 cluster

The changes required to upgrade the RKE2 cluster using the automated workflow are the following:

- Change the block RKE2ControlPlane in the capi-provisioning-example.yaml shown in the following section (*Section 31.4, “Downstream cluster provisioning with Directed network provisioning (single-node)”* (page 397)):
 - Add the rollout strategy in the spec file.
 - Change the version of the RKE2 cluster to the new version replacing \${RKE2_NEW_VERSION}.

```
apiVersion: controlplane.cluster.x-k8s.io/v1alpha1
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
  serverConfig:
    cni: cilium
  rolloutStrategy:
    rollingUpdate:
      maxSurge: 0
  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
  version: ${RKE2_NEW_VERSION}
  nodeName: "localhost.localdomain"

```

- Change the block `Metal3MachineTemplate` in the `capi-provisioning-example.yaml` shown in the following section ([Section 31.4, “Downstream cluster provisioning with Directed network provisioning \(single-node\)”](#) (page 397)):
 - Change the image name and checksum to the new version generated in the previous step.
 - Add the directive `nodeReuse` to `true` to avoid creating a new node.
 - Add the directive `automatedCleaningMode` to `metadata` to enable the automated cleaning for the node.

```

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
```

After making these changes, the `capi-provisioning-example.yaml` file can be applied to the cluster using the following command:

```
kubectl apply -f capi-provisioning-example.yaml
```

VII Appendix

33 Release Notes 440

33 Release Notes

33.1 Abstract

SUSE Edge 3.0 is a tightly integrated and comprehensively validated end-to-end solution for addressing the unique challenges of the deployment of infrastructure and cloud-native applications at the edge. Its driving focus is to provide an opinionated, yet highly flexible, highly scalable, and secure platform that spans initial deployment image building, node provisioning and onboarding, application deployment, observability, and lifecycle management.

The solution is designed with the notion that there is no "one-size-fits-all" edge platform due to our customers' widely varying requirements and expectations. Edge deployments push us to solve, and continually evolve, some of the most challenging problems, including massive scalability, restricted network availability, physical space constraints, new security threats and attack vectors, variations in hardware architecture and system resources, the requirement to deploy and interface with legacy infrastructure and applications, and customer solutions that have extended lifespans.


SUSE Edge is built on best-of-breed open source software from the ground up, consistent with both our 30-year history in delivering secure, stable, and certified SUSE Linux platforms and our experience in providing highly scalable and feature-rich Kubernetes management with our Rancher portfolio. SUSE Edge builds on-top of these capabilities to deliver functionality that can address a wide number of market segments, including retail, medical, transportation, logistics, telecommunications, smart manufacturing, and Industrial IoT.



Note

SUSE Adaptive Telco Infrastructure Platform (ATIP) is a derivative (or downstream product) of SUSE Edge, with additional optimizations and components that enable the platform to address the requirements found in telecommunications use-cases. Unless explicitly stated, all of the release notes are applicable for both SUSE Edge 3.0, and SUSE ATIP 3.0.

33.2 About

These Release Notes are, unless explicitly specified and explained, identical across all architectures, and the most recent version, along with the release notes of all other SUSE products are always available online at <https://www.suse.com/releasenotes> .

Entries are only listed once, but they can be referenced in several places if they are important and belong to more than one section. Release notes usually only list changes that happened between two subsequent releases. Certain important entries from the release notes of previous product versions may be repeated. To make these entries easier to identify, they contain a note to that effect.

However, repeated entries are provided as a courtesy only. Therefore, if you are skipping one or releases, check the release notes of the skipped releases also. If you are only reading the release notes of the current release, you could miss important changes that may affect system behavior. SUSE Edge versions are defined as x.y.z, where 'x' denotes the major version, 'y' denotes the minor, and 'z' denotes the patch version, also known as the "z-stream". SUSE Edge product lifecycles are defined based around a given minor release, e.g. "3.0", but ship with subsequent patch updates through its lifecycle, e.g. "3.0.1".



Note

SUSE Edge z-stream releases are tightly integrated and thoroughly tested as a versioned stack. Upgrade of any individual components to a different versions to those listed above is likely to result in system downtime. While it's possible to run Edge clusters in untested configurations, it is not recommended, and it may take longer to provide resolution through the support channels.

33.3 Release 3.0.3

Availability Date: 15th November 2024

Summary: SUSE Edge 3.0.3 is the third z-stream release in the SUSE Edge 3.0 portfolio.

33.3.1 Bug & Security Fixes

- The Rancher version is updated to **2.8.8**: [Release Notes \(https://github.com/rancher/rancher/releases/tag/v2.8.8\)](https://github.com/rancher/rancher/releases/tag/v2.8.8) ↗
- The RKE2 version is updated to **1.28.13**: [Release Notes \(https://docs.rke2.io/release-notes/v1.28.X#release-v12813rke2r1\)](https://docs.rke2.io/release-notes/v1.28.X#release-v12813rke2r1) ↗
- The K3s version is updated to **1.28.13**: [Release Notes \(https://docs.k3s.io/release-notes/v1.28.X#release-v12813k3s1\)](https://docs.k3s.io/release-notes/v1.28.X#release-v12813k3s1) ↗
- The Metal³ chart fixes an issue with the handling of the `predictableNicNames` parameter: [SUSE Edge issue #163 \(https://github.com/suse-edge/charts/pull/163\)](https://github.com/suse-edge/charts/pull/163) ↗
- The Metal³ chart resolves security issues identified in [CVE-2024-43803 \(https://www.cve.org/CVERecord?id=CVE-2024-43803:\)](https://www.cve.org/CVERecord?id=CVE-2024-43803) ↗: [SUSE Edge issue #163 \(https://github.com/suse-edge/charts/pull/163\)](https://github.com/suse-edge/charts/pull/163) ↗
- The Metal³ chart resolves security issues identified in [CVE-2024-44082 \(https://www.cve.org/CVERecord?id=CVE-2024-44082:\)](https://www.cve.org/CVERecord?id=CVE-2024-44082) ↗: [SUSE Edge issue #163 \(https://github.com/suse-edge/charts/pull/163\)](https://github.com/suse-edge/charts/pull/163) ↗

33.3.2 Components Versions

The following table describes the individual components that make up the 3.0.3 release, including the version, the Helm chart version (if applicable), and from where the released artifact can be pulled in the binary format. Please follow the associated documentation for usage and deployment examples. Note that items in bold are highlighted changes from the previous z-stream release.

Name	Version	Helm Chart Version	Artifact Location (URL/Image)
SLE Micro	5.5 (latest)	N/A	SLE Micro Download Page (https://www.suse.com/download/sle-micro/) ↗ SLE-Micro.x86_64-5.5.0-Default-SelfIn-

			<p>stall-GM2.in-stall.iso (sha256 4f672a4a0f8ec421e7c25797def05598037c56b7f306283566a9f921dce904a)</p> <p>SLE-Micro.x86_64-5.5.0-Default-RT-SelfInstall-GM2.in-stall.iso (sha256 527a5a7cdbf11e3e6238e386533755SLE-Micro.x86_64-5.5.0-Default-GM.raw.xz (sha256 13243a737ca219bad6a7aa41fa7470f10a756cf4d575f4493ed68b)</p> <p>SLE-Micro.x86_64-5.5.0-Default-RT-GM.raw.xz (sha256 6c2af94e7ac785c8f6a276032c8e6a</p>
SUSE Manager	4.3.11	N/A	<p>SUSE Manager Download Page (https://www.suse.com/download/suse-manager/)</p>
K3s	1.28.13	N/A	<p>Upstream K3s Release (https://github.com/k3s-io/k3s/releases/tag/v1.28.13%2Bk3s1)</p>

RKE2	1.28.13	N/A	Upstream RKE2 Release (https://github.com/rancher/rke2/releases/tag/v1.28.13%2Brke2r1) ↗
Rancher Prime	2.8.8	2.8.8	Rancher 2.8.8 Images (https://prime.rub-s.rancher.io/rancher/v2.8.8/rancher-images.txt) ↗ Rancher Prime Helm Repo (https://charts.rancher.com/server-charts/prime) ↗
Longhorn	1.6.1	103.3.0	Longhorn 1.6.1 Images (https://raw.githubusercontent.com/longhorn/longhorn/v1.6.1/deploy/longhorn-images.txt) ↗ Longhorn Helm Repo (https://charts.longhorn.io) ↗
NM Configurator	0.3.0	N/A	NMConfigurator Upstream Release (https://github.com/suse-edge/nm-configurator/releases/tag/v0.3.0) ↗

NeuVector	5.3.2	103.0.3	reg- istry.suse.com/ranch- er/mirrored-neuvec- tor-controller:5.3.2 reg- istry.suse.com/ranch- er/mirrored-neuvec- tor-enforcer:5.3.2 reg- istry.suse.com/ranch- er/mirrored-neuvec- tor-manager:5.3.2 reg- istry.suse.com/ranch- er/mirrored-neuvec- tor-prometheus-ex- porter:5.3.2 reg- istry.suse.com/ranch- er mirrored-neu- vector-reg- istry-adapter:0.1.1-s1 reg- istry.suse.com/ranch- er/mirrored-neuvec- tor-scanner:latest reg- istry.suse.com/ranch- er/mirrored-neuvec- tor-updater:latest
Cluster API (CAPI)	1.6.2	N/A	reg- istry.suse.com/edge/ cluster-api-con- troller:1.6.2

			<p>reg-istry.suse.com/edge/cluster-api-provider-metal3:1.6.0</p> <p>reg-istry.suse.com/edge/cluster-api-provider-rke2-bootstrap:0.4.1</p> <p>reg-istry.suse.com/edge/cluster-api-provider-rke2-control-plane:0.4.1</p>
Metal³	0.7.4	0.7.4	<p>reg-istry.suse.com/edge/metal3-chart:0.7.4</p> <p>reg-istry.suse.com/edge/baremetal-operator:0.5.2</p> <p>reg-istry.suse.com/edge/ip-address-manager:1.6.0</p> <p>reg-istry.suse.com/edge/ironic:23.0.3.1</p> <p>reg-istry.suse.com/edge/ironic-ipa-downloader:1.3.5</p> <p>reg-istry.suse.com/edge/kube-rbac-prox-</p>

			registry:v0.14.2 +.1 registry.suse.com/edge/mariadb:10.6.15.1
MetalLB	0.14.3	0.14.3	registry.suse.com/edge/metallb-chart:0.14.3 registry.suse.com/edge/metallb-controller:v0.14.3 registry.suse.com/edge/metallb-speaker:v0.14.3 registry.suse.com/edge/frr:8.4 registry.suse.com/edge/frr-k8s:v0.0.8
Elemental	1.4.4	1.4.4	registry.suse.com/rancher/elemental-operator-chart:1.4.4 registry.suse.com/rancher/elemental-operator-crds-chart:1.4.4 registry.suse.com/rancher/elemental-operator:1.4.4

Edge Image Builder	1.0.2	N/A	reg-istry.suse.com/edge/edge-image-builder:1.0.2
KubeVirt	1.2.2	0.3.0	reg-istry.suse.com/edge/kubevirt-chart:0.3.0 reg-istry.suse.com/suse/sles/15.5/virt-operator:1.2.2 reg-istry.suse.com/suse/sles/15.5/virt-api:1.2.2 reg-istry.suse.com/suse/sles/15.5/virt-controller:1.2.2 reg-istry.suse.com/suse/sles/15.5/virt-export-proxy:1.2.2 reg-istry.suse.com/suse/sles/15.5/virt-export-server:1.2.2 reg-istry.suse.com/suse/sles/15.5/virt-handler:1.2.2

			reg- istry.suse.com/suse/ sles/15.5/virt- launcher:1.2.2
KubeVirt Dashboard Extension	1.0.0	1.0.0	reg- istry.suse.com/edge/ kubevirt-dash- board-exten- sion-chart:1.0.0
Containerized Data Importer	1.59.0	0.3.0	reg- istry.suse.com/edge/ cdi-chart:0.3.0 reg- istry.suse.com/suse/ sles/15.5/cdi-opera- tor:1.59.0 reg- istry.suse.com/suse/ sles/15.5/cdi-con- troller:1.59.0 reg- istry.suse.com/suse/ sles/15.5/cdi-im- porter:1.59.0 reg- istry.suse.com/suse/ sles/15.5/cdi-clon- er:1.59.0 reg- istry.suse.com/suse/ sles/15.5/cdi-apis- erver:1.59.0

			<p>reg-istry.suse.com/suse/sles/15.5/cdi-upload-server:1.59.0</p> <p>reg-istry.suse.com/suse/sles/15.5/cdi-upload-proxy:1.59.0</p>
Endpoint Copier Operator	0.2.0	0.2.0	<p>reg-istry.suse.com/edge/endpoint-copier-operator:v0.2.0</p> <p>reg-istry.suse.com/edge/endpoint-copier-operator-chart:0.2.0</p>
Akri (Tech Preview)	0.12.20	0.12.20	<p>reg-istry.suse.com/edge/akri-chart:0.12.20</p> <p>reg-istry.suse.com/edge/akri-dashboard-extension-chart:1.0.0</p> <p>reg-istry.suse.com/edge/akri-agent:v0.12.20</p> <p>reg-istry.suse.com/edge/akri-controller:v0.12.20</p>

			reg-istry.suse.com/edge/akri-debug-echo-discovery-handler:v0.12.20 reg-istry.suse.com/edge/akri-onvif-discovery-handler:v0.12.20 reg-istry.suse.com/edge/akri-opcua-discovery-handler:v0.12.20 reg-istry.suse.com/edge/akri-udev-discovery-handler:v0.12.20 reg-istry.suse.com/edge/akri-webhook-configuration:v0.12.20
SR-IOV Network Operator	1.2.2	1.2.2 + up0.1.0	reg-istry.suse.com/edge/sriov-network-operator-chart:1.2.2 reg-istry.suse.com/edge/sriov-crd-chart:1.2.2

33.4 Release 3.0.2

Availability Date: 16th August 2024

Summary: SUSE Edge 3.0.2 is the second z-stream release in the SUSE Edge 3.0 portfolio.

33.4.1 New Features

- The Metal³ chart now supports static network configuration without any `mac-address`: [SUSE Edge issue #134 \(https://github.com/suse-edge/charts/pull/134\)](https://github.com/suse-edge/charts/pull/134) ↗
- KubeVirt is updated from `1.1.1` to `1.2.2` for details of new features refer to the: [Upstream Release Notes \(https://github.com/kubevirt/kubevirt/releases\)](https://github.com/kubevirt/kubevirt/releases) ↗

33.4.2 Bug & Security Fixes

- The Metal³ chart fixes an issue where host reprovisioning may use stale static network configuration: [SUSE Edge issue #133 \(https://github.com/suse-edge/charts/pull/133\)](https://github.com/suse-edge/charts/pull/133) ↗
- The RKE2 Cluster API provider fixes an issue when specifying TLS configuration for a local registry: [RKE2 Provider issue #357 \(https://github.com/rancher/cluster-api-provider-rke2/pull/357\)](https://github.com/rancher/cluster-api-provider-rke2/pull/357) ↗
- The RKE2 Cluster API provider fixes an issue causing `rke2-install` to error after system reboot: [RKE2 Provider issue #346 \(https://github.com/rancher/cluster-api-provider-rke2/pull/346\)](https://github.com/rancher/cluster-api-provider-rke2/pull/346) ↗
- KubeVirt is updated to include several security fixes: [Kubevirt Update \(https://www.suse.com/support/update/announcement/2024/suse-su-20242669-1\)](https://www.suse.com/support/update/announcement/2024/suse-su-20242669-1) ↗


33.4.3 Components Versions

The following table describes the individual components that make up the 3.0.2 release, including the version, the Helm chart version (if applicable), and from where the released artifact can be pulled in the binary format. Please follow the associated documentation for usage and deployment examples. Note that items in bold are highlighted changes from the previous z-stream release.

Name	Version	Helm Chart Version	Artifact Location (URL/Image)
SLE Micro	5.5 (latest)	N/A	SLE Micro Download Page (https://www.suse.com/download/sle-micro/) ↗

			<p>SLE-Mi-cro.x86_64-5.5.0-Default-SelfInstall-GM2.install.iso (sha256 4f672a4a0f8ec421e7c25797def05598037c56b7f306283566a9f921dce904a)</p> <p>SLE-Mi-cro.x86_64-5.5.0-Default-RT-SelfInstall-GM2.install.iso (sha256 527a5a7cdbf11e3e6238e386533755)</p> <p>SLE-Mi-cro.x86_64-5.5.0-Default-GM.raw.xz (sha256 13243a737ca219bad6a7aa41fa7470f10a756cf4d575f4493ed68b)</p> <p>SLE-Mi-cro.x86_64-5.5.0-Default-RT-GM.raw.xz (sha256 6c2af94e7ac785c8f6a276032c8e6a)</p>
SUSE Manager	4.3.11	N/A	<p>SUSE Manager Download Page (https://www.suse.com/download/suse-manager/) ↗</p>

K3s	1.28.10	N/A	Upstream K3s Release (https://github.com/k3s-io/k3s/releases/tag/v1.28.10%2Bk3s1) ↗
RKE2	1.28.10	N/A	Upstream RKE2 Release (https://github.com/rancher/rke2/releases/tag/v1.28.10%2Brke2r1) ↗
Rancher Prime	2.8.5	2.8.5	Rancher 2.8.5 Images (https://github.com/rancher/rancher/releases/download/v2.8.5/rancher-images.txt) ↗ Rancher Prime Helm Repo (https://charts.rancher.com/server-charts/prime) ↗
Longhorn	1.6.1	103.3.0	Longhorn 1.6.1 Images (https://raw.githubusercontent.com/longhorn/longhorn/v1.6.1/deploy/longhorn-images.txt) ↗ Longhorn Helm Repo (https://charts.longhorn.io) ↗
NM Configurator	0.3.0	N/A	NMConfigurator Upstream Release (https://github.com/)

			suse-edge/nm-configurator/releases/tag/v0.3.0 
NeuVector	5.3.2	103.0.3	reg- istry.suse.com/ranch- er/mirrored-neuvec- tor-controller:5.3.2 reg- istry.suse.com/ranch- er/mirrored-neuvec- tor-enforcer:5.3.2 reg- istry.suse.com/ranch- er/mirrored-neuvec- tor-manager:5.3.2 reg- istry.suse.com/ranch- er/mirrored-neuvec- tor-prometheus-ex- porter:5.3.2 reg- istry.suse.com/ranch- er mirrored-neu- vector-reg- istry-adapter:0.1.1-s1 reg- istry.suse.com/ranch- er/mirrored-neuvec- tor-scanner:latest reg- istry.suse.com/ranch- er/mirrored-neuvec- tor-updater:latest

Cluster API (CAPI)	1.6.2	N/A	<p>reg-istry.suse.com/edge/cluster-api-controller:1.6.2</p> <p>reg-istry.suse.com/edge/cluster-api-provider-metal3:1.6.0</p> <p>reg-istry.suse.com/edge/cluster-api-provider-rke2-bootstrap:0.4.1</p> <p>reg-istry.suse.com/edge/cluster-api-provider-rke2-controlplane:0.4.1</p>
Metal³	0.7.3	0.7.3	<p>reg-istry.suse.com/edge/metal3-chart:0.7.3</p> <p>reg-istry.suse.com/edge/baremetal-operator:0.5.1</p> <p>reg-istry.suse.com/edge/ip-address-manager:1.6.0</p> <p>reg-istry.suse.com/edge/ironic:23.0.2.1</p>

			<p>registry.suse.com/edge/ironic-ipa-downloader:1.3.4</p> <p>registry.suse.com/edge/kube-rbac-proxy:v0.14.2 +.1 registry.suse.com/edge/mariadb:10.6.15.1</p>
MetalLB	0.14.3	0.14.3	<p>registry.suse.com/edge/metallb-chart:0.14.3</p> <p>registry.suse.com/edge/metallb-controller:v0.14.3</p> <p>registry.suse.com/edge/metallb-speaker:v0.14.3</p> <p>registry.suse.com/edge/frr:8.4</p> <p>registry.suse.com/edge/frr-k8s:v0.0.8</p>
Elemental	1.4.4	1.4.4	<p>registry.suse.com/rancher/elemental-operator-chart:1.4.4</p>

			<p>reg-istry.suse.com/rancher/elemental-operator-crds-chart:1.4.4</p> <p>reg-istry.suse.com/rancher/elemental-operator:1.4.4</p>
Edge Image Builder	1.0.2	N/A	reg-istry.suse.com/edge/edge-image-builder:1.0.2
KubeVirt	1.2.2	0.3.0	<p>reg-istry.suse.com/edge/kubevirt-chart:0.3.0</p> <p>reg-istry.suse.com/suse/sles/15.5/virt-operator:1.2.2</p> <p>reg-istry.suse.com/suse/sles/15.5/virt-api:1.2.2</p> <p>reg-istry.suse.com/suse/sles/15.5/virt-controller:1.2.2</p> <p>reg-istry.suse.com/suse/sles/15.5/virt-exportproxy:1.2.2</p>

			<p>reg- istry.suse.com/suse/ sles/15.5/virt-ex- portserver:1.2.2</p> <p>reg- istry.suse.com/suse/ sles/15.5/virt-han- dler:1.2.2</p> <p>reg- istry.suse.com/suse/ sles/15.5/virt- launcher:1.2.2</p>
KubeVirt Dashboard Extension	1.0.0	1.0.0	reg- istry.suse.com/edge/ kubevirt-dash- board-exten- sion-chart:1.0.0
Containerized Data Importer	1.59.0	0.3.0	<p>reg- istry.suse.com/edge/ cdi-chart:0.3.0</p> <p>reg- istry.suse.com/suse/ sles/15.5/cdi-opera- tor:1.59.0</p> <p>reg- istry.suse.com/suse/ sles/15.5/cdi-con- troller:1.59.0</p> <p>reg- istry.suse.com/suse/ sles/15.5/cdi-im- porter:1.59.0</p>

			<p>reg-istry.suse.com/suse/sles/15.5/cdi-cloner:1.59.0</p> <p>reg-istry.suse.com/suse/sles/15.5/cdi-apiserver:1.59.0</p> <p>reg-istry.suse.com/suse/sles/15.5/cdi-uploadserver:1.59.0</p> <p>reg-istry.suse.com/suse/sles/15.5/cdi-uploadproxy:1.59.0</p>
Endpoint Copier Operator	0.2.0	0.2.0	<p>reg-istry.suse.com/edge/endpoint-copier-operator:v0.2.0</p> <p>reg-istry.suse.com/edge/endpoint-copier-operator-chart:0.2.0</p>
Akri (Tech Preview)	0.12.20	0.12.20	<p>reg-istry.suse.com/edge/akri-chart:0.12.20</p> <p>reg-istry.suse.com/edge/akri-dashboard-extension-chart:1.0.0</p> <p>reg-istry.suse.com/edge/akri-agent:v0.12.20</p>

			<p>reg-istry.suse.com/edge/akri-controller:v0.12.20</p> <p>reg-istry.suse.com/edge/akri-debug-echo-discovery-handler:v0.12.20</p> <p>reg-istry.suse.com/edge/akri-onvif-discovery-handler:v0.12.20</p> <p>reg-istry.suse.com/edge/akri-opcua-discovery-handler:v0.12.20</p> <p>reg-istry.suse.com/edge/akri-udev-discovery-handler:v0.12.20</p> <p>reg-istry.suse.com/edge/akri-webhook-configuration:v0.12.20</p>
SR-IOV Network Operator	1.2.2	1.2.2 + up0.1.0	<p>reg-istry.suse.com/edge/sriov-network-operator-chart:1.2.2</p> <p>reg-istry.suse.com/edge/sriov-crd-chart:1.2.2</p>

33.5 Release 3.0.1

Availability Date: 14th June 2024

Summary: SUSE Edge 3.0.1 is the first z-stream release in the SUSE Edge 3.0 portfolio.

33.5.1 New Features


- Elemental and EIB now support node reset for unmanaged hosts
- SR-IOV Network Operator chart is now included
- The Metal³ chart now supports providing additional trusted CA certificates
- NM Configurator now supports applying unified configurations without any MAC specification
- Added `version` subcommand to EIB; the version will also automatically be included in each image built by EIB

33.5.2 Bug & Security Fixes

- EIB now automatically sets the execute bit on custom scripts: [SUSE Edge issue #429 \(https://github.com/suse-edge/edge-image-builder/issues/429\)](https://github.com/suse-edge/edge-image-builder/issues/429) ↗
- EIB now supports disks which are > 512 byte sector size: [SUSE Edge issue #447 \(https://github.com/suse-edge/edge-image-builder/issues/447\)](https://github.com/suse-edge/edge-image-builder/issues/447) ↗
- Enhance EIB's ability to detect container images in Helm charts: [SUSE Edge issue #442 \(https://github.com/suse-edge/edge-image-builder/issues/442\)](https://github.com/suse-edge/edge-image-builder/issues/442) ↗

33.5.3 Components Versions

The following table describes the individual components that make up the 3.0.1 release, including the version, the Helm chart version (if applicable), and where the released artifact can be pulled from in binary format. Please follow the associated documentation for usage and deployment examples. Note that items in bold are highlighted changes from the previous z-stream release.

Name	Version	Helm Chart Version	Artifact Location (URL/Image)
SLE Micro	5.5 (latest)	N/A	<p>SLE Micro Download Page (https://www.suse.com/download/sle-micro/) </p> <p>SLE-Mi-cro.x86_64-5.5.0-Default-SelfInstall-GM2.iso (sha256 4f672a4a0f8ec421e7c25797def05598037c56b7f306283566a9f921dce904a)</p> <p>SLE-Mi-cro.x86_64-5.5.0-Default-RT-SelfInstall-GM2.iso (sha256 527a5a7cdbf11e3e6238e386533755)</p> <p>SLE-Mi-cro.x86_64-5.5.0-Default-GM.raw.xz (sha256 13243a737ca219bad6a7aa41fa747cf10a756cf4d575f4493ed68b)</p> <p>SLE-Mi-cro.x86_64-5.5.0-Default-RT-GM.raw.xz (sha256 6c2af94e7ac785c8f6a276032c8e6a)</p>

SUSE Manager	4.3.11	N/A	SUSE Manager Download Page (https://www.suse.com/download/suse-manager/)
K3s	1.28.9	N/A	Upstream K3s Release (https://github.com/k3s-io/k3s/releases/tag/v1.28.9%2Bk3s1)
RKE2	1.28.9	N/A	Upstream RKE2 Release (https://github.com/rancher/rke2/releases/tag/v1.28.9%2Brke2r1)
Rancher Prime	2.8.4	2.8.4	Rancher 2.8.4 Images (https://github.com/rancher/rancher/releases/download/v2.8.4/rancher-images.txt) Rancher Prime Helm Repo (https://charts.rancher.com/server-charts/prime)
Longhorn	1.6.1	103.3.0	Longhorn 1.6.1 Images (https://raw.githubusercontent.com/longhorn/longhorn/v1.6.1/deploy/longhorn-images.txt)

			Longhorn Helm Repo (https://charts.longhorn.io) ↗
NM Configurator	0.3.0	N/A	NMConfigurator Upstream Release (https://github.com/suse-edge/nm-configurator/releases/tag/v0.3.0) ↗
NeuVector	5.3.2	103.0.3	reg-istry.suse.com/rancher/mirrored-neuvector-controller:5.3.2 reg-istry.suse.com/rancher/mirrored-neuvector-enforcer:5.3.2 reg-istry.suse.com/rancher/mirrored-neuvector-manager:5.3.2 reg-istry.suse.com/rancher/mirrored-neuvector-prometheus-exporter:5.3.2 reg-istry.suse.com/rancher mirrored-neuvector-reg-istry-adapter:0.1.1-s1

			<p>reg-istry.suse.com/rancher/mirrored-neuvector-scanner:latest</p> <p>reg-istry.suse.com/rancher/mirrored-neuvector-updater:latest</p>
Cluster API (CAPI)	1.6.2	N/A	<p>reg-istry.suse.com/edge/cluster-api-controller:1.6.2</p> <p>reg-istry.suse.com/edge/cluster-api-provider-metal3:1.6.0</p> <p>reg-istry.suse.com/edge/cluster-api-provider-rke2-bootstrap:0.2.6</p> <p>reg-istry.suse.com/edge/cluster-api-provider-rke2-control-plane:0.2.6</p>
Metal³	0.7.1	0.7.1	<p>reg-istry.suse.com/edge/metal3-chart:0.7.1</p> <p>reg-istry.suse.com/edge/baremetal-operator:0.5.1</p>

			<p>reg-istry.suse.com/edge/ip-address-manager:1.6.0</p> <p>reg-istry.suse.com/edge/ironic:23.0.2.1</p> <p>reg-istry.suse.com/edge/ironic-ipa-downloader:1.3.2</p> <p>reg-istry.suse.com/edge/kube-rbac-proxy:v0.14.2 +.1 reg-istry.suse.com/edge/mariadb:10.6.15.1</p>
MetalLB	0.14.3	0.14.3	<p>reg-istry.suse.com/edge/metallb-chart:0.14.3</p> <p>reg-istry.suse.com/edge/metallb-controller:v0.14.3</p> <p>reg-istry.suse.com/edge/metallb-speaker:v0.14.3</p> <p>reg-istry.suse.com/edge/frr:8.4</p> <p>reg-istry.suse.com/edge/frr-k8s:v0.0.8</p>

Elemental	1.4.4	1.4.4	reg- istry.suse.com/ranch- er/elemental-opera- tor-chart:1.4.4 reg- istry.suse.com/ranch- er/elemental-opera- tor-crds-chart:1.4.4 reg- istry.suse.com/ranch- er/elemental-opera- tor:1.4.4
Edge Image Builder	1.0.2	N/A	reg- istry.suse.com/edge/ edge-im- age-builder:1.0.2
KubeVirt	1.1.1	0.2.4	reg- istry.suse.com/edge/ kubevirt-chart:0.2.4 reg- istry.suse.com/suse/ sles/15.5/virt-opera- tor:1.1.1 reg- istry.suse.com/suse/ sles/15.5/virt- api:1.1.1 reg- istry.suse.com/suse/ sles/15.5/virt-con- troller:1.1.1

			<p>reg-istry.suse.com/suse/sles/15.5/virt-export-proxy:1.1.1</p> <p>reg-istry.suse.com/suse/sles/15.5/virt-export-server:1.1.1</p> <p>reg-istry.suse.com/suse/sles/15.5/virt-handler:1.1.1</p> <p>reg-istry.suse.com/suse/sles/15.5/virt-launcher:1.1.1</p>
KubeVirt Dashboard Extension	1.0.0	1.0.0	reg-istry.suse.com/edge/kubevirt-dashboard-extension-chart:1.0.0
Containerized Data Importer	1.58.0	0.2.3	<p>reg-istry.suse.com/edge/cdi-chart:0.2.3</p> <p>reg-istry.suse.com/suse/sles/15.5/cdi-operator:1.58.0</p> <p>reg-istry.suse.com/suse/sles/15.5/cdi-controller:1.58.0</p>

			<p>reg-istry.suse.com/suse/sles/15.5/cdi-importer:1.58.0</p> <p>reg-istry.suse.com/suse/sles/15.5/cdi-cloner:1.58.0</p> <p>reg-istry.suse.com/suse/sles/15.5/cdi-api-server:1.58.0</p> <p>reg-istry.suse.com/suse/sles/15.5/cdi-upload-server:1.58.0</p> <p>reg-istry.suse.com/suse/sles/15.5/cdi-upload-proxy:1.58.0</p>
Endpoint Copier Operator	0.2.0	0.2.0	<p>reg-istry.suse.com/edge/endpoint-copier-operator:v0.2.0</p> <p>reg-istry.suse.com/edge/endpoint-copier-operator-chart:0.2.0</p>
Akri (Tech Preview)	0.12.20	0.12.20	<p>reg-istry.suse.com/edge/akri-chart:0.12.20</p>

		reg-istry.suse.com/edge/akri-dashboard-extension-chart:1.0.0
		reg-istry.suse.com/edge/akri-agent:v0.12.20
		reg-istry.suse.com/edge/akri-controller:v0.12.20
		reg-istry.suse.com/edge/akri-debug-echo-discovery-handler:v0.12.20
		reg-istry.suse.com/edge/akri-onvif-discovery-handler:v0.12.20
		reg-istry.suse.com/edge/akri-opcua-discovery-handler:v0.12.20
		reg-istry.suse.com/edge/akri-udev-discovery-handler:v0.12.20
		reg-istry.suse.com/edge/akri-webhook-configuration:v0.12.20

SR-IOV Network Operator	1.2.2	1.2.2 + up0.1.0	reg- istry.suse.com/edge/ sriov-network-opera- tor-chart:1.2.2 reg- istry.suse.com/edge/ sriov-crd-chart:1.2.2
--------------------------------	--------------	------------------------	--

33.6 Release 3.0.0

Availability Date: 26th April 2024

Summary: SUSE Edge 3.0.0 is the first release in the SUSE Edge 3.0 portfolio.

33.6.1 New Features

- Not Applicable - this is the first release shipped in 3.0.z.

33.6.2 Bug & Security Fixes

- Not Applicable - this is the first release shipped in 3.0.z.


33.6.3 Components Versions

The following table describes the individual components that make up the 3.0.0 release, including the version, the Helm chart version (if applicable), and where the released artifact can be pulled from in binary format. Please follow the associated documentation for usage and deployment examples.

Name	Version	Helm Chart Version	Artifact Location (URL/Image)

SLE Micro	5.5 (latest)	N/A	<p>SLE Micro Download Page (https://www.suse.com/download/sle-micro/)</p> <p>SLE-Micro.x86_64-5.5.0-Default-SelfInstall-GM2.iso (sha256 4f672a4a0f8ec421e7c25797def05598037c56b7f306283566a9f921dce904a)</p> <p>SLE-Micro.x86_64-5.5.0-Default-RT-SelfInstall-GM2.iso (sha256 527a5a7cdbf11e3e6238e386533755)</p> <p>SLE-Micro.x86_64-5.5.0-Default-GM.raw.xz (sha256 13243a737ca219bad6a7aa41fa747cf10a756cf4d575f4493ed68b)</p> <p>SLE-Micro.x86_64-5.5.0-Default-RT-GM.raw.xz (sha256 6c2af94e7ac785c8f6a276032c8e6a)</p>
SUSE Manager	4.3.11	N/A	<p>SUSE Manager Download Page (https://www.suse.com/download/suse-manager/)</p>

K3s	1.28.8	N/A	Upstream K3s Release (https://github.com/k3s-io/k3s/releases/tag/v1.28.8%2Bk3s1) ↗
RKE2	1.28.8	N/A	Upstream RKE2 Release (https://github.com/rancher/rke2/releases/tag/v1.28.8%2Brke2r1) ↗
Rancher Prime	2.8.3	2.8.3	Rancher 2.8.3 Images (https://github.com/rancher/rancher/releases/download/v2.8.3/rancher-images.txt) ↗ Rancher Prime Helm Repo (https://charts.rancher.com/server-charts/prime) ↗
Longhorn	1.6.1	103.3.0	Longhorn 1.6.1 Images (https://raw.githubusercontent.com/longhorn/longhorn/v1.6.1/deploy/longhorn-images.txt) ↗ Longhorn Helm Repo (https://charts.longhorn.io) ↗
NM Configurator	0.2.3	N/A	NMConfigurator Upstream Release (https://github.com/)

			suse-edge/nm-configurator/releases/tag/v0.2.3 
NeuVector	5.3.2	103.0.3	reg- istry.suse.com/ranch- er/mirrored-neuvec- tor-controller:5.3.2 reg- istry.suse.com/ranch- er/mirrored-neuvec- tor-enforcer:5.3.2 reg- istry.suse.com/ranch- er/mirrored-neuvec- tor-manager:5.3.2 reg- istry.suse.com/ranch- er/mirrored-neuvec- tor-prometheus-ex- porter:5.3.2 reg- istry.suse.com/ranch- er mirrored-neu- vector-reg- istry-adapter:0.1.1-s1 reg- istry.suse.com/ranch- er/mirrored-neuvec- tor-scanner:latest reg- istry.suse.com/ranch- er/mirrored-neuvec- tor-updater:latest

Cluster API (CAPI)	1.6.2	N/A	<p>reg-istry.suse.com/edge/cluster-api-controller:1.6.2</p> <p>reg-istry.suse.com/edge/cluster-api-provider-metal3:1.6.0</p> <p>reg-istry.suse.com/edge/cluster-api-provider-rke2-bootstrap:0.2.6</p> <p>reg-istry.suse.com/edge/cluster-api-provider-rke2-control-plane:0.2.6</p>
Metal ³	0.6.5	0.6.5	<p>reg-istry.suse.com/edge/metal3-chart:0.6.5</p> <p>reg-istry.suse.com/edge/baremetal-operator:0.5.1</p> <p>reg-istry.suse.com/edge/ip-address-manager:1.6.0</p> <p>reg-istry.suse.com/edge/ironic:23.0.1.2</p> <p>reg-istry.suse.com/edge/ironic-ipa-downloader:1.3.1</p>

			reg-istry.suse.com/edge/kube-rbac-proxy:v0.14.2 reg-istry.suse.com/edge/mariadb:10.6.15.1
MetalLB	0.14.3	0.14.3	reg-istry.suse.com/edge/metallb-chart:0.14.3 reg-istry.suse.com/edge/metallb-controller:v0.14.3 reg-istry.suse.com/edge/metallb-speaker:v0.14.3 reg-istry.suse.com/edge/frr:8.4 reg-istry.suse.com/edge/frr-k8s:v0.0.8
Elemental	1.4.3	1.4.3	reg-istry.suse.com/rancher/elemental-operator-chart:1.4.3 reg-istry.suse.com/rancher/elemental-operator-crds-chart:1.4.3

			reg- istry.suse.com/ranch- er/elemental-opera- tor:1.4.3
Edge Image Builder	1.0.1	N/A	reg- istry.suse.com/edge/ edge-im- age-builder:1.0.1
KubeVirt	1.1.1	0.2.4	reg- istry.suse.com/edge/ kubevirt-chart:0.2.4 reg- istry.suse.com/suse/ sles/15.5/virt-opera- tor:1.1.1 reg- istry.suse.com/suse/ sles/15.5/virt- api:1.1.1 reg- istry.suse.com/suse/ sles/15.5/virt-con- troller:1.1.1 reg- istry.suse.com/suse/ sles/15.5/virt-export- proxy:1.1.1 reg- istry.suse.com/suse/ sles/15.5/virt-export- server:1.1.1

			<p>reg- istry.suse.com/suse/ sles/15.5/virt-han- dler:1.1.1</p> <p>reg- istry.suse.com/suse/ sles/15.5/virt- launcher:1.1.1</p>
KubeVirt Dashboard Extension	1.0.0	1.0.0	<p>reg- istry.suse.com/edge/ kubevirt-dash- board-exten- sion-chart:1.0.0</p>
Containerized Data Importer	1.58.0	0.2.3	<p>reg- istry.suse.com/edge/ cdi-chart:0.2.3</p> <p>reg- istry.suse.com/suse/ sles/15.5/cdi-opera- tor:1.58.0</p> <p>reg- istry.suse.com/suse/ sles/15.5/cdi-con- troller:1.58.0</p> <p>reg- istry.suse.com/suse/ sles/15.5/cdi-im- porter:1.58.0</p> <p>reg- istry.suse.com/suse/ sles/15.5/cdi-clon- er:1.58.0</p>

			<p>reg-istry.suse.com/suse/sles/15.5/cdi-apiserver:1.58.0</p> <p>reg-istry.suse.com/suse/sles/15.5/cdi-upload-server:1.58.0</p> <p>reg-istry.suse.com/suse/sles/15.5/cdi-upload-proxy:1.58.0</p>
Endpoint Copier Operator	0.2.0	0.2.0	<p>reg-istry.suse.com/edge/endpoint-copier-operator:v0.2.0</p> <p>reg-istry.suse.com/edge/endpoint-copier-operator-chart:0.2.0</p>
Akri (Tech Preview)	0.12.20	0.12.20	<p>reg-istry.suse.com/edge/akri-chart:0.12.20</p> <p>reg-istry.suse.com/edge/akri-dashboard-extension-chart:1.0.0</p> <p>reg-istry.suse.com/edge/akri-agent:v0.12.20</p> <p>reg-istry.suse.com/edge/akri-controller:v0.12.20</p>

		reg-istry.suse.com/edge/akri-debug-echo-discovery-handler:v0.12.20
		reg-istry.suse.com/edge/akri-onvif-discovery-handler:v0.12.20
		reg-istry.suse.com/edge/akri-opcua-discovery-handler:v0.12.20
		reg-istry.suse.com/edge/akri-udev-discovery-handler:v0.12.20
		reg-istry.suse.com/edge/akri-webhook-configuration:v0.12.20

33.7 Components Verification

The components mentioned above may be verified using the Software Bill Of Materials (SBOM) data - for example using `cosign` as outlined below:

Download the SUSE Edge Container public key from the [SUSE Signing Keys source \(https://www.suse.com/support/security/keys/\)](https://www.suse.com/support/security/keys/):

```
> cat key.pem
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAA0CAg8AMIICGKCAgEA7N0S2d8LFW4WU43bq7Z
IZT537x1Ke170QEpYjNrdtqnSwA0/jLtK83m7bTzfYRK4wty/so0g3BGo+x6yDFt
SVXTPBqnYvabU/j7UKaybJtX3jc4SjaezeBqdi96h6yEs1vg4VTZDpy6TFP5ZHxZ
A0fX6m5kU2/RYhGXIt0eUmL5hZ+APYgYG4/455NBaZT2y0ywJ6+1zRgpR0cRAekI
OZX151k0ebsGV6ui/NGEC06MB5e3arAhszf8eHDE02FeNJw5cimXkgDh/1Lg3Kp0
dvUNm0EPWvknkNYeMCKR+687QG0bXqSVyCbY6+HG/HLkeBWkv6Hn41oeTSLrjYVGa
```

```
T3zxPVQM726sami6pgZ5vULy0leQuKBZrLFhFLbFyXqv1/DokUqEppm2Y3xZQv77
fMNogapp0qYz+nE3wSK4UHPd9z+2bq5WEkQSalYxadyuq0zxqZgSoCNoX5iIuWte
Zf1RmHjiEndg/2UgxKUysVnyCpiWoGbaLM4dnWE24102050Gj6M4B5fe73hbaRlf
NBqP+97uznnRlS18FizhXzdzJiVPcRav1tDdRUyDE2XkNRXmGfD3aCmILhB27SOA
Lppkouw849PWBt9kDMvzeLUYLpINyPHRi2+/eyhHNlufeyJ7e7d6N9VcvjR/6qWG
64iSkcF2DTW61CN5TrCe0k0CAwEAAQ==
-----END PUBLIC KEY-----
```

Verify the container image hash, for example using [crane](#):

```
> crane digest registry.suse.com/edge/baremetal-operator:0.5.1
sha256:13e8b2c59aeb503f8adaac095495007071559c9d6d8ef5a7cb1ce6fd1430c782
```

Verify with [cosign](#):

```
> cosign verify-attestation --type spdxjson --key key.pem registry.suse.com/edge/
baremetal-
operator@sha256:13e8b2c59aeb503f8adaac095495007071559c9d6d8ef5a7cb1ce6fd1430c782 > /dev/
null
#
Verification for registry.suse.com/edge/baremetal-
operator@sha256:13e8b2c59aeb503f8adaac095495007071559c9d6d8ef5a7cb1ce6fd1430c782 --
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
```

Extract SBOM data as described at the [upstream documentation \(https://www.suse.com/support/security/sbom/\)](https://www.suse.com/support/security/sbom/):

```
> cosign verify-attestation --type spdxjson --key key.pem registry.suse.com/edge/
baremetal-
operator@sha256:13e8b2c59aeb503f8adaac095495007071559c9d6d8ef5a7cb1ce6fd1430c782 | jq
'.payload | @base64d | fromjson | .predicate'
```

33.8 Upgrade Steps

Refer to the Day 2 Documentation for details around how to upgrade to a new z-stream release.

33.9 Known Limitations

Unless otherwise stated these apply to the 3.0.0 release and all subsequent z-stream versions.

- Akri is released for the first time as a Technology Preview offering, and is not subject to the standard scope of support.
- Rancher UI Extensions used in SUSE Edge cannot currently be deployed via the Rancher Marketplace and must be deployed manually. [Rancher issue #29105 \(https://github.com/rancher/rancher/issues/29105\)](https://github.com/rancher/rancher/issues/29105)
- If you're using NVIDIA GPU's, SELinux cannot be enabled at the containerd layer due to a missing SELinux policy. [Bugzilla #1222725 \(https://bugzilla.suse.com/show_bug.cgi?id=1222725\)](https://bugzilla.suse.com/show_bug.cgi?id=1222725)
- If deploying with Metal³ and Cluster API (CAPI), clusters aren't automatically imported into Rancher post-installation. It will be addressed in future releases.
- Due to certain limitations, Elemental and Metal³ components cannot be deployed on the same management cluster. It will be addressed in future releases.

33.10 Product Support Lifecycle

SUSE Edge is backed by award-winning support from SUSE, an established technology leader with a proven history of delivering enterprise-quality support services. For more information, see <https://www.suse.com/lifecycle> and the Support Policy page at <https://www.suse.com/support/policy.html>. If you have any questions about raising a support case, how SUSE classifies severity levels, or the scope of support, please see the Technical Support Handbook at <https://www.suse.com/support/handbook/>.

At the time of publication, each minor version of SUSE Edge, e.g. "3.0" is supported for 12-months of production support, with an initial 6-months of "full support", followed by 6-months of "maintenance support". In the "full support" coverage period, SUSE may introduce new features (that do not break existing functionality), introduce bug fixes, and deliver security patches. During the "maintenance support" window, only critical security and bug fixes will be introduced, with other fixes delivered at our discretion.

Unless explicitly stated, all components listed are considered Generally Available (GA), and are covered by SUSE's standard scope of support. Some components may be listed as "Technology Preview", where SUSE is providing customers with access to early pre-GA features and functionality for evaluation, but are not subject to the standard support policies and are not recommended for production use-cases. SUSE very much welcomes feedback and suggestions on the

improvements that can be made to Technology Preview components, but SUSE reserves the right to deprecate a Technology Preview feature before it becomes Generally Available if it doesn't meet the needs of our customers or doesn't reach a state of maturity that we require.

Please note that SUSE must occasionally deprecate features or change API specifications. Reasons for feature deprecation or API change could include a feature being updated or replaced by a new implementation, a new feature set, upstream technology is no longer available, or the upstream community has introduced incompatible changes. It is not intended that this will ever happen within a given minor release (x.z), and so all z-stream releases will maintain API compatibility and feature functionality. SUSE will endeavor to provide deprecation warnings with plenty of notice within the release notes, along with workarounds, suggestions, and mitigations to minimize service disruption.

The SUSE Edge team also welcomes community feedback, where issues can be raised within the respective code repository within <https://www.github.com/suse-edge>.

33.11 Obtaining source code

This SUSE product includes materials licensed to SUSE under the GNU General Public License (GPL) and various other open source licenses. The GPL requires SUSE to provide the source code that corresponds to the GPL-licensed material, and SUSE conforms to all other open-source license requirements. As such, SUSE makes all source code available, and can generally be found in the SUSE Edge GitHub repository (<https://www.github.com/suse-edge>), the SUSE Rancher GitHub repository (<https://www.github.com/rancher>) for dependent components, and specifically for SLE Micro, the source code is available for download at <https://www.suse.com/download/sle-micro> (<https://www.suse.com/download/sle-micro/>) on "Medium 2".

33.12 Legal notices

SUSE makes no representations or warranties with regard to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, SUSE reserves the right to revise this publication and to make changes to its content, at any time, without the obligation to notify any person or entity of such revisions or changes.

Further, SUSE makes no representations or warranties with regard to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, SUSE reserves the right to make changes to any and all parts of SUSE software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classifications to export, re-export, or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical/biological weaponry end uses. Refer to <https://www.suse.com/company/legal/> for more information on exporting SUSE software. SUSE assumes no responsibility for your failure to obtain any necessary export approvals.

Copyright © 2024 SUSE LLC.

This release notes document is licensed under a Creative Commons Attribution-NoDerivatives 4.0 International License (CC-BY-ND-4.0). You should have received a copy of the license along with this document. If not, see <https://creativecommons.org/licenses/by-nd/4.0/>.

SUSE has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <https://www.suse.com/company/legal/> and one or more additional patents or pending patent applications in the U.S. and other countries.

For SUSE trademarks, see the SUSE Trademark and Service Mark list (<https://www.suse.com/company/legal/>). All third-party trademarks are the property of their respective owners. For SUSE brand information and usage requirements, please see the guidelines published at <https://brand.suse.com/>.