



SUSE Edge ドキュメント

SUSE Edgeドキュメント

発行日: 2025年9月26日

<https://documentation.suse.com> 

目次

SUSE Edge 3.4ドキュメント xix

- 1 SUSE Edgeとは xix
- 2 設計理念 xix
- 3 高レベルアーキテクチャ xx
 - SUSE Edgeで使用するコンポーネント xx • 接続 xxiv
- 4 一般的なEdgeデプロイメントパターン xxv
 - ダイレクトネットワークプロビジョニング xxv • 「Phone Home」 ネットワークプロビジョニング xxvi • イメージベースのプロビジョニング xxvii
- 5 SUSE Edge Stack Validation xxvii
- 6 コンポーネントの全リスト xxviii

I クイックスタート 1

1 Metal³を使用したBMCの自動デプロイメント 2

- 1.1 この方法を使用する理由 2
- 1.2 高レベルアーキテクチャ 3
- 1.3 前提条件 4
- 1.4 デプロイメント 5
 - 管理クラスタのセットアップ 5 • Metal³の依存関係のインストール 5 • Cluster APIの依存関係のインストール 7 • ダウンストリームクラスタイムージの準備 8 • BareMetalHostインベントリの追加 11 • ダウンストリームクラスタの作成 15 • コントロールプレーンのデプロイメント 15 • ワーカー/コンピュータのデプロイメント 19 • クラスタのプロビジョニング解除 22
- 1.5 既知の問題 22
- 1.6 予定されている変更 23

1.7 追加のリソース 23

シングルノード設定 23 • 仮想メディアISOをアタッチするためのTLSの無効化 24 • ストレージ設定 24

2 Elementalを使用したリモートホストのオンボーディング 25

2.1 高レベルアーキテクチャ 26

2.2 必要なリソース 26

2.3 ブートストラップクラスタの構築 28

Kubernetesクラスタの作成 28 • DNSの設定 28

2.4 Rancherのインストール 28

2.5 Elementalのインストール 29

(オプション) Elemental UI拡張機能のインストール 30

2.6 Elementalの設定 32

2.7 イメージの構築 36

2.8 ダウンストリームノードのブート 38

2.9 ダウンストリームクラスタの作成 38

2.10 ノードリセット(オプション) 40

2.11 次の手順 42

3 Edge Image Builderを使用したスタンドアロンクラスタ 43

3.1 前提条件 43

EIBイメージの取得 44

3.2 イメージ設定ディレクトリの作成 44

3.3 イメージ定義ファイルの作成 45

OSユーザの設定 45 • OSの時刻の設定 46 • 証明書の追加 47 • RPMパッケージの設定 47 • Kubernetesクラスタとユーザーワークロードの設定 49 • ネットワークの設定 51

3.4 イメージの構築 53

3.5 イメージ構築プロセスのデバッグ 56

3.6 新しく構築されたイメージのテスト 57

4 SUSE Multi-Linux Manager 58

4.1 SUSE Multi-Linux Manager Serverのデプロイ 58

4.2 SUSE Multi-Linux Managerの設定 60

4.3 Edge Image Builderを使用したカスタマイズされたインストールイメージの作成 62

venv-salt-minionパッケージのダウンロード 64

4.4 SUSE Multi-Linux Manager CA証明書のダウンロード 65

II コンポーネント 67

5 Rancher 68

5.1 Rancherの主な機能 68

5.2 SUSE EdgeでのRancherの使用 68

Kubernetesの集中管理 69 • クラスタデプロイメントのシンプル化 69 • アプリケーションのデプロイメントと管理 69 • セキュリティとポリシーの適用 69

5.3 ベストプラクティス 69

GitOps 69 • 可観測性 70

5.4 Edge Image Builderを使用したインストール 70

5.5 追加リソース 70

6 Rancher Dashboard拡張機能 71

6.1 インストール 71

Rancher Dashboard UIを使用したインストール 71 • Helmを使用したインストール 73 • Fleetを使用したインストール 74

6.2 KubeVirtダッシュボード拡張機能 75

7 Rancher Turtles 76

7.1 Rancher Turtlesの主な機能 76

7.2 SUSE EdgeでのRancher Turtlesの使用 76

7.3 Rancher Turtlesのインストール 76

7.4 追加リソース 77

8 Fleet 78

8.1 Helmを使用したFleetのインストール 78

8.2 RancherでのFleetの使用 78

8.3 Rancher UIでのFleetへのアクセス 78

Dashboard (ダッシュボード) 79 • Git repos (Gitリポジトリ) 79 • Clusters (クラスタ) 79 • Cluster groups (クラスタグループ) 79 • Advanced (詳細) 80

8.4 Rancher Dashboardを使用してRancherおよびFleetとともにKubeVirtをインストールする例 80

8.5 デバッグとトラブルシューティング 82

8.6 Fleetの例 83

9 SUSE Linux Micro 84

9.1 SUSE EdgeでのSUSE Linux Microの用途 84

9.2 ベストプラクティス 84

インストールメディア 84 • ローカル管理 84

9.3 既知の問題 85

10 Metal³ 86

10.1 SUSE EdgeでのMetal³の用途 86

10.2 既知の問題 86

11 Edge Image Builder 88

11.1 SUSE EdgeでのEdge Image Builderの用途 88

11.2 はじめに 89

11.3 既知の問題 89

12 Edgeネットワーキング 90

- 12.1 NetworkManagerの概要 90
- 12.2 nmstateの概要 90
- 12.3 NetworkManager Configurator (nmc)の概要 90
- 12.4 SUSE EdgeでのNetworkManager Configuratorの用途 91
- 12.5 Edge Image Builderを使用した設定 91
 - 前提条件 91 • Edge Image Builderのコンテナイメージの取得 91 • イメージ設定ディレクトリの作成 92 • イメージ定義ファイルの作成 92 • ネットワーク設定の定義 93 • OSイメージの構築 98 • エッジノードのプロビジョニング 98 • 統合されたノード設定 105 • カスタムネットワーク設定 108

13 Elemental 113

- 13.1 SUSE EdgeでのElementalの用途 113
- 13.2 ベストプラクティス 114
 - インストールメディア 114 • ラベル 114
- 13.3 既知の問題 114

14 K3s 115

- 14.1 SUSE EdgeでのK3sの用途 115
- 14.2 ベストプラクティス 115
 - インストール 115 • GitOpsワークフローでのFleet 115 • ストレージ管理 116 • 負荷分散とHA 116

15 RKE2 117

- 15.1 RKE2とK3s 117
- 15.2 SUSE EdgeでのRKE2の用途 117
- 15.3 ベストプラクティス 118
 - インストール 118 • 高可用性 118 • ネットワーキング 119 • ストレージ 119

16 SUSE Storage 120

- 16.1 前提条件 120
- 16.2 SUSE Storageの手動インストール 120
 - Open-iSCSIのインストール 120 • SUSE Storageのインストール 121
- 16.3 SUSE Storageボリュームの作成 122
- 16.4 UIへのアクセス 125
- 16.5 Edge Image Builderを使用したインストール 125

17 SUSE Security 128

- 17.1 SUSE EdgeでのSUSE Securityの用途 129
- 17.2 重要なメモ 129
- 17.3 Edge Image Builderを使用したインストール 129

18 MetalLB 130

- 18.1 SUSE EdgeでのMetalLBの用途 130
- 18.2 ベストプラクティス 131
- 18.3 既知の問題 131

19 Endpoint Copier Operator 132

- 19.1 SUSE EdgeでのEndpoint Copier Operatorの用途 132
- 19.2 ベストプラクティス 132
- 19.3 既知の問題 132

20 Edge Virtualization 133

- 20.1 KubeVirtの概要 133
- 20.2 前提条件 133
- 20.3 Edge Virtualizationの手動インストール 134
- 20.4 仮想マシンのデプロイ 137
- 20.5 virtctlの使用 140

- 20.6 シンプルなIngressネットワーキング 142
- 20.7 Rancher UI拡張機能の使用 144
 - インストール 144 • KubeVirt Rancher Dashboard拡張機能の使用 144
- 20.8 Edge Image Builderを使用したインストール 146
- 21 System Upgrade Controller 147**
 - 21.1 SUSE EdgeでのSystem Upgrade Controllerの用途 147
 - 21.2 System Upgrade Controllerのインストール 147
 - System Upgrade Controller Fleetのインストール 148 • System Upgrade Controller Helmのインストール 153
 - 21.3 System Upgrade Controller Planのモニタリング 154
 - System Upgrade Controller Planのモニタリング - Rancher UI 154 • System Upgrade Controller Planのモニタリング - 手動 155
- 22 Upgrade Controller 156**
 - 22.1 SUSE EdgeでのUpgrade Controllerの用途 156
 - 22.2 Upgrade ControllerとSystem Upgrade Controller 157
 - 22.3 Upgrade Controllerのインストール 157
 - 前提条件 157 • 手順 157
 - 22.4 Upgrade Controllerの仕組み 158
 - オペレーティングシステムのアップグレード 159 • Kubernetesのアップグレード 160 • 追加のコンポーネントのアップグレード 160
 - 22.5 Kubernetes API拡張機能 161
 - UpgradePlan 161 • ReleaseManifest 162
 - 22.6 アップグレードプロセスの追跡 163
 - 一般 163 • Helm Controller 168
 - 22.7 既知の制限事項 168

23 SUSE Multi-Linux Manager 170

III ハウツーガイド 171

24 K3s上のMetalLB (レイヤ2モードを使用) 172

- 24.1 MetalLBを使用する理由 172
- 24.2 K3s上のMetalLB (L2を使用) 172
- 24.3 前提条件 173
- 24.4 デプロイメント 173
- 24.5 設定 174
 - TraefikとMetalLB 175
- 24.6 使用法 175
 - IngressとMetalLB 178

25 K3s上のMetalLB (レイヤ3モードを使用) 181

- 25.1 MetalLBを使用する理由 181
- 25.2 K3s上のMetalLB (L3を使用) 182
- 25.3 前提条件 182
- 25.4 サービスIPアドレスをアドバタイズするための設定 182
- 25.5 デプロイメント 182
- 25.6 設定 183
- 25.7 使用法 184

26 Kubernetes APIサーバの前面のMetalLB 187

- 26.1 前提条件 187
- 26.2 RKE2/K3sのインストール 187
- 26.3 既存のクラスタの設定 189
- 26.4 MetalLBのインストール 189
- 26.5 Endpoint Copier Operatorのインストール 190

26.6	コントロールプレーンノードの追加	192
27	Edge Image Builderを使用したエアギャップデプロイメント	194
27.1	概要	194
27.2	前提条件	194
27.3	Libvirtのネットワーク設定	195
27.4	ベースディレクトリの設定	195
27.5	ベース定義ファイル	197
27.6	Rancherのインストール	198
27.7	SUSE Securityのインストール	203
27.8	SUSE Storageのインストール	206
27.9	KubeVirtとCDIのインストール	210
27.10	トラブルシューティング	213
28	Kiwiを使用したSUSE Linux Microの更新イメージの構築	214
28.1	前提条件	215
28.2	はじめに	215
28.3	デフォルトイメージの構築	216
28.4	他のプロファイルを使用したイメージの構築	217
28.5	大きいセクタサイズを使用したイメージの構築	217
28.6	カスタムのKiwiイメージ定義ファイルの使用	218
29	ClusterClassを使用したダウンストリームクラスタのデプロイ	219
29.1	はじめに	219
29.2	ClusterClassとは	219

- 29.3 現在のCAPIプロビジョニングファイルの例 220
- 29.4 CAPIプロビジョニングファイルのClusterClassへの変換 225
 - ClusterClass定義 225 • クラスティンスタンス定義 231

IV ヒントとコツ 234

30 Edge Image Builder 235

- 30.1 一般 235
- 30.2 Kubernetes 235

31 Elemental 237

- 31.1 一般 237
 - Rancherサービスの公開 237
- 31.2 ハードウェア固有 239
 - Trusted Platform Module 239

V サードパーティの統合 241

32 NATS 242

- 32.1 アーキテクチャ 242
 - NATSクライアントアプリケーション 242 • NATSサービスインフラストラクチャ 242 • シンプルなメッセージングデザイン 243 • NATS JetStream 243
- 32.2 インストール 243
 - K3s上へのNATSのインストール 243 • K3sのバックエンドとしてのNATS 245

33 SUSE Linux Micro上のNVIDIA GPU 247

- 33.1 概要 247
- 33.2 前提条件 248
- 33.3 手動インストール 248
- 33.4 手動インストールの追加検証 252
- 33.5 Kubernetesを使用した実装 255

33.6 Edge Image Builderを使用した統合 258

33.7 問題の解決 261
nvidia-smiでGPUが検出されない 261

VI DAY 2操作 262

34 Edge 3.4のマイグレーション 263

34.1 管理クラスタ 263
前提条件 263 • Upgrade Controller 264 • Fleet 265

34.2 ダウンストリームクラスタ 266
Fleet 266

35 管理クラスタ 268

35.1 Upgrade Controller 268
前提条件 268 • アップグレード 269

35.2 Fleet 269
コンポーネント 270 • ユースケースの決定 271 • Day 2ワークフロー 272 • OSアップグレード 272 • Kubernetesバージョンアップグレード 284 • Helmチャートのアップグレード 298

36 ダウンストリームクラスタ 322

36.1 Fleet 322
コンポーネント 323 • ユースケースの決定 323 • Day 2ワークフロー 324 • OSアップグレード 325 • Kubernetesバージョンアップグレード 336 • Helmチャートのアップグレード 349

VII SUSE TELCO CLOUDのドキュメント 372

37 SUSE Telco Cloud 373

38 コンセプトとアーキテクチャ 374

38.1 SUSE Telco Cloudアーキテクチャ 374

38.2 コンポーネント 374

- 38.3 デプロイメントフローの例 375
 - 例1: すべてのコンポーネントがインストールされた新しい管理クラスタをデプロイする 376
 - 例2: 通信事業者プロファイルを使用してシングルノードのダウストリームクラスタをデプロイして通信ワークロードを実行可能にする 376
 - 例3: MetalLBをロードバランサとして使用して高可用性ダウストリームクラスタをデプロイする 377

39 要件と前提 379

- 39.1 ハードウェア 379
- 39.2 ネットワーク 380
- 39.3 ポート要件 381
 - 管理ノード 381
 - ダウストリームノード 384
 - CNI固有のポート要件 387
- 39.4 サービス(DHCP、DNSなど) 388
- 39.5 systemdサービスの無効化 388

40 管理クラスタの設定 390

- 40.1 はじめに 390
- 40.2 管理クラスタの設定手順 390
- 40.3 接続環境用のイメージの準備 393
 - ディレクトリ構造 393
 - 管理クラスタ定義ファイル 395
 - Customフォルダ 400
 - Kubernetesフォルダ 406
 - ネットワーキングフォルダ 411
- 40.4 エアギャップ環境のイメージの準備 413
 - 定義ファイルの変更 413
 - カスタムフォルダの変更 417
 - Helm値フォルダの変更 417
- 40.5 イメージの作成 418
- 40.6 管理クラスタのプロビジョニング 418
- 40.7 デュアルスタックの考慮事項と設定 418

41 通信機能の設定 423

- 41.1 リアルタイム用のカーネルイメージ 424

- 41.2 低レイテンシとハイパフォーマンスのためのカーネル引数 425
- 41.3 Tunedとカーネル引数によるCPUピニング 429
- 41.4 CNI設定 432
 - Cilium 432 • Calico 433 • Bond CNI 433
- 41.5 SR-IOV 439
- 41.6 DPDK 449
- 41.7 vRANアクセラレーション(Intel ACC100/ACC200) 451
- 41.8 Huge Page 453
- 41.9 KubernetesでのCPUピニング 455
 - 前提条件 455 • CPUピニング用のKubernetesの設定 455 • ワークロードにピニングされたCPUを活用する 455
- 41.10 NUMA対応のスケジューリング 457
 - NUMAノードの特定 457
- 41.11 MetalLB 457
- 41.12 プライベートレジストリ設定 459
- 41.13 Precision Time Protocol 460
 - PTPソフトウェアコンポーネントのインストール 461 • 通信事業者のデプロイメント向けPTPの設定 463 • Cluster APIの統合 466

42 完全に自動化されたダイレクトネットワークプロビジョニング 470

- 42.1 はじめに 470
- 42.2 接続シナリオのダウンストリームクラスタイメージの準備 471
 - 接続シナリオの前提条件 471 • 接続シナリオのイメージの設定 472 • イメージの作成 477
- 42.3 エアギャップシナリオ用のダウンストリームクラスタイメージの準備 478
 - エアギャップシナリオの前提条件 478 • エアギャップシナリオのイメージの設定 479 • エアギャップシナリオのイメージの作成 484

- 42.4 ダイレクトネットワークプロビジョニングを使用したダウンストリームクラスタのプロビジョニング(シングルノード) 485
- 42.5 ダイレクトネットワークプロビジョニングを使用したダウンストリームクラスタのプロビジョニング(マルチノード) 492
- 42.6 高度なネットワーク設定 504
- 42.7 通信機能(DPDK、SR-IOV、CPUの分離、Huge Page、NUMAなど) 509
- 42.8 プライベートレジストリ 517
- 42.9 エアギャップシナリオでのダウンストリームクラスタのプロビジョニング 519
 - エアギャップシナリオの要件 520
 - エアギャップシナリオでのベアメタルホストの登録 520
 - エアギャップシナリオでのダウンストリームクラスタのプロビジョニング 520

43 ライフサイクルのアクション 527

- 43.1 管理クラスタのアップグレード 527
- 43.2 ダウンストリームクラスタのアップグレード 527

VIII **トラブルシューティング 531**

44 一般的なトラブルシューティングの原則 532

45 Kiwiのトラブルシューティング 534

46 Edge Image Builder (EIB)のトラブルシューティング 536

47 Edgeネットワーキング(NMC)のトラブルシューティング 538

48 Phone-Homeシナリオのトラブルシューティング 540

49 ダイレクトネットワークプロビジョニングのトラブルシューティング 541

50 その他のコンポーネントのトラブルシューティング 547

51 サポートのための診断情報の収集 548

IX **付録 550**

52 リリースノート 551

52.1 要約 551

52.2 概要 552

52.3 リリース3.4.0 552

新機能 553 • バグおよびセキュリティの修正 553 • 既知の問題 554 • コンポーネントバージョン 556

52.4 廃止された機能 566

52.5 技術プレビュー 567

52.6 コンポーネントの検証 567

52.7 アップグレード手順 568

52.8	製品サポートライフサイクル	568
52.9	ソースコードの取得	570
52.10	法的通知	570

SUSE Edge 3.4ドキュメント

『SUSE Edgeドキュメント』をお読みいただきありがとうございます。このドキュメントには、高レベルアーキテクチャの概要、クイックスタートガイド、検証済みの設計、コンポーネントの使用に関するガイダンス、サードパーティ統合、エッジコンピューティングインフラストラクチャとワークロードを管理するためのベストプラクティスが記載されています。

1 SUSE Edgeとは

SUSE Edgeは、インフラストラクチャとクラウドネイティブなアプリケーションをエッジにデプロイするという独自の課題に対処することに特化した、緊密に統合されて包括的に検証されたエンドツーエンドのソリューションです。SUSE Edgeが重点を置いているのは、独創的でありながら高い柔軟性とスケーラビリティを持つセキュアなプラットフォームを提供し、初期デプロイメントイメージの構築からノードのプロビジョニングとオンボーディング、アプリケーションのデプロイメント、可観測性、ライフサイクル全体の運用にまで対応することです。このプラットフォームは、最良のオープンソースソフトウェアに基づいてゼロから構築されており、SUSEが持つ、30年にわたってセキュアで安定した定評あるSUSE Linuxプラットフォームを提供してきた歴史と、Rancherポートフォリオによって拡張性に優れ機能豊富なKubernetes管理を提供してきた経験の両方に合致するものです。SUSE Edgeは、これらの機能の上に構築されており、小売、医療、輸送、物流、通信、スマート製造、産業用IoTなど、さまざまな市場セグメントに対応できる機能を提供します。

2 設計理念

このソリューションは、顧客の要件や期待はさまざまであるため「万能」なエッジプラットフォームは存在しないという考え方に基いて設計されています。エッジデプロイメントにより、実に困難な問題を解決し、継続的に進化させることが要求されます。たとえば、大規模なスケーラビリティ、ネットワークの可用性の制限、物理的なスペースの制約、新たなセキュリティの脅威と攻撃ベクトル、ハードウェアアーキテクチャとシステムリソースのバリエーション、レガシインフラストラクチャやレガシアプリケーションのデプロイとインタフェースの要件、耐用年数を延長している顧客ソリューションといった課題があります。こうした課題の多くは、従来の考え方(たとえば、データセンター内やパブリッククラウドへのインフラストラクチャやアプリケーションのデプロイメント)とは異なるため、はるかに細かく設計を検討し、一般的な前提の多くを再検討する必要があります。

たとえば、SUSEはミニマリズム、モジュール性、操作のしやすさに価値を見出しています。システムは複雑化するほど故障しやすくなるため、エッジ環境ではミニマリズムが重要です。数百、数十万カ所に及ぶとなると、複雑なシステムは複雑な故障が発生します。また、SUSEのソリューションはモジュール性を備えているため、ユーザの選択肢を増やししながら、デプロイしたプラットフォームが不必要に複雑になることを解消できます。さらに、ミニマリズムおよびモジュール性と、操作のしやすさとのバランスを取ることも必要です。人間はプロセスを何千回も繰り返すとミスを犯す可能性があるため、プラットフォーム側で潜在的なミスを確実に回復し、技術者が現場に出向かなくても済むようにすると同時に、一貫性と標準化を実現するよう努める必要もあります。

3 高レベルアーキテクチャ

SUSE Edgeの高レベルシステムアーキテクチャは、「管理」クラスタと「ダウンストリーム」クラスタの2つのコアカテゴリに分けられます。管理クラスタは1つまたは複数のダウンストリームクラスタのリモート管理を担当しますが、特定の状況下では、ダウンストリームクラスタはリモート管理なしで動作する必要があります。たとえば、エッジサイトに外部接続がない場合や独立して動作する必要がある場合などです。SUSE Edgeでは、管理クラスタとダウンストリームクラスタの両方の動作に利用される技術コンポーネントは大部分が共通していますが、システム仕様と最上位に位置するアプリケーションが異なる場合があります。すなわち管理クラスタはシステム管理とライフサイクル操作を有効にするアプリケーションを実行しますが、ダウンストリームクラスタはユーザアプリケーションを提供するための要件を満たします。

3.1 SUSE Edgeで使用されるコンポーネント

SUSE Edgeは、既存のSUSEとRancherのコンポーネントと、エッジコンピューティングに必要な制約や複雑さに対応できるようにEdgeチームが構築した追加機能とコンポーネントで構成されています。管理クラスタとダウンストリームクラスタの両方で使用されるコンポーネントは、高レベルなアーキテクチャ図とともに以下に説明しますが、これは網羅的なリストではないことに注意してください。

3.1.1 管理クラスタ



- **管理:** これは、接続されたダウストリームクラスタのプロビジョニングとライフサイクルの管理に使用されるSUSE Edgeの中核です。管理クラスタには通常、以下のコンポーネントが含まれます。
 - Rancher Prime (第5章「[Rancher](#)」)によるマルチクラスタ管理により、ダウストリームクラスタのオンボーディングとインフラストラクチャおよびアプリケーションの継続的なライフサイクル管理のための共通ダッシュボードが可能になり、包括的なテナント分離とIDP (アイデンティティプロバイダ)統合、サードパーティの統合および拡張のための大規模なマーケットプレイス、ベンダニュートラルなAPIも提供されます。
 - SUSE Multi-Linux Managerを使用したLinuxシステム管理により、ダウストリームクラスタ上で実行される基礎となるLinuxオペレーティングシステム(*SUSE Linux Micro (第9章「[SUSE Linux Micro](#)」))の自動的なLinuxパッチおよび設定管理が可能になります。このコンポーネントはコンテナ化されていますが、現時点では他の管理コンポーネントとは別のシステムで実行する必要があるため、上の図では「Linux管理」とラベル付けされています。
 - 特定のSUSE Edgeリリースへの管理クラスタコンポーネントのアップグレードを処理する専用のライフサイクル管理(第22章「[Upgrade Controller](#)」)コントローラ。
 - Elemental (第13章「[Elemental](#)」)を使用したRancher Primeへのリモートシステムのオンボーディングにより、接続されたエッジノードを目的のKubernetesクラスタに遅延バインディングしたり、GitOps経由でアプリケーションをデプロイメントしたりできます。

- Metal3 (第10章 「Metal³」)、MetalLB (第18章 「MetalLB」)、およびCAPI (Cluster API)インフラストラクチャプロバイダによるオプションの完全なベアメタルライフサイクルおよび管理サポートにより、リモート管理機能を備えたベアメタルシステムの完全なエンドツーエンドのプロビジョニングが可能になります。
- ダウンストリームクラスタとそれらに存在するアプリケーションのプロビジョニングとライフサイクルの管理のためのFleet (第8章 「Fleet」)と呼ばれるオプションのGitOpsエンジン。
- 管理クラスタ自体を支えるのは、ベースオペレーティングシステムとしてのSUSE Linux Micro (第9章 「SUSE Linux Micro」)と、管理クラスタアプリケーションをサポートするKubernetesディストリビューションとしてのRKE2 (第15章 「RKE2」)です。

3.1.2 ダウンストリームクラスタ



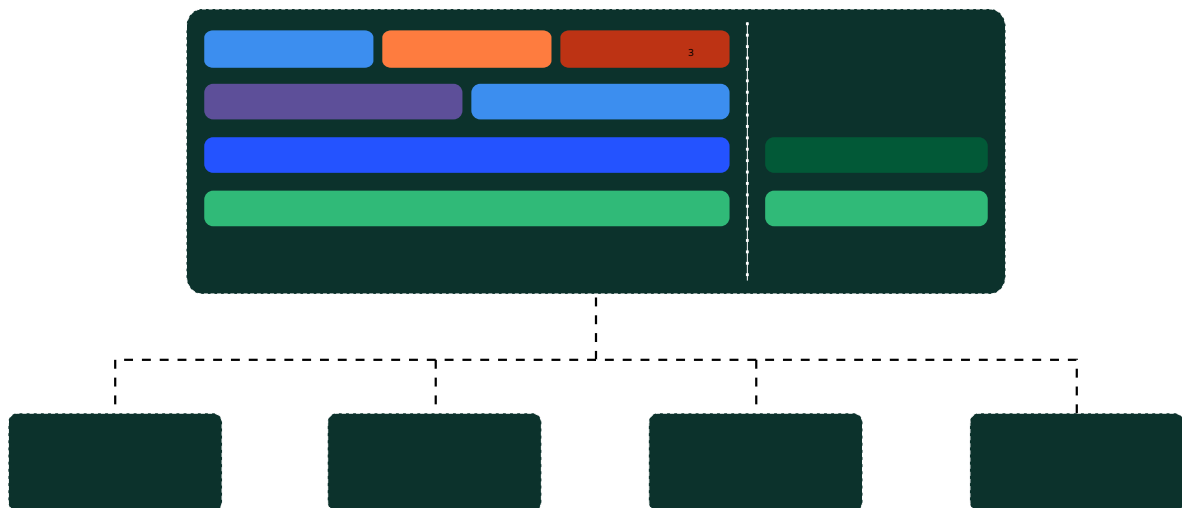
- **ダウンストリーム:** これは、エッジでユーザワークロードを実行するために使用されるSUSE Edgeの分散部分です。つまり、エッジの場所自体で実行されるソフトウェアであり、通常は次のコンポーネントで構成されます。

- K3s (第14章 「K3s」)やRKE2 (第15章 「RKE2」)などのセキュアで軽量なディストリビューションを含む、Kubernetesディストリビューションの選択肢(RKE2は、政府機関や規制産業での使用に耐えるように強化、認定、最適化されています)。
- SUSE Security (第17章 「SUSE Security」)を使用するとイメージ脆弱性スキャン、ディープパケットインスペクション、リアルタイム脅威および脆弱性保護のようなセキュリティ機能が有効になります。
- SUSE Storage (第16章 「SUSE Storage」)によるソフトウェアブロックストレージにより、軽量で永続的、弾力性があり、拡張可能なブロックストレージが可能になります。
- SUSE Linux Micro (第9章 「SUSE Linux Micro」)を搭載した、軽量でコンテナに最適化された堅牢なLinuxオペレーティングシステムで、エッジでのコンテナや仮想マシンの実行に不変で耐障害性に優れたOSを提供します。SUSE Linux Micro

は、AArch64およびAMD64/Intel 64アーキテクチャの両方で使用でき、レイテンシの影響を受けやすいアプリケーション(通信事業者のユースケースなど)向けのリアルタイムカーネルもサポートしています。

- 接続されたクラスタ(つまり、管理クラスタに接続しているクラスタ)には、Rancher Primeへの接続を管理するためのRancher System Agentと、SUSE Multi-Linux Managerからの指示を受けてLinuxソフトウェアの更新を適用するためのvenv-salt-minionの2つのエージェントがデプロイされます。これらのエージェントは、切断されたクラスタの管理には必要ありません。

3.2 接続



上記のイメージは、**接続された**ダウストリームクラスタと、それらの管理クラスタへの接続に関する高レベルアーキテクチャの概要を示しています。管理クラスタは、ダウストリームクラスタとターゲット管理クラスタとの間のネットワーキングの可用性に応じて、オンプレミスとクラウドの両方の容量で、さまざまな基礎となるインフラストラクチャプラットフォーム上にデプロイできます。これが機能するための唯一の要件は、ダウストリームクラスタノードを管理インフラストラクチャに接続するネットワークでアクセス可能なAPIとコールバックURLです。

この接続が確立されるメカニズムは、ダウストリームクラスタのデプロイメントのメカニズムとは異なるものであることを認識することが重要です。この詳細については、次のセクションでさらに詳しく説明しますが、基本的な理解を深めるために、接続されたダウストリームクラスタが「管理」クラスタとして確立される主なメカニズムは3つあります。

1. ダウンストリームクラスタはまず「切断された」容量でデプロイされ(Edge Image Builder (第11章「Edge Image Builder」)経由)、接続が許可されると、管理クラスタにインポートされます。
2. ダウンストリームクラスタは、組み込みオンボーディングメカニズム(たとえば Elemental (第13章「Elemental」)経由)を使用するように設定され、初回ブート時に管理クラスタに自動的に登録されるため、クラスタ設定の遅延バインディングが許可されます。
3. ダウンストリームクラスタにはベアメタル管理機能(CAPI + Metal³)がプロビジョニングされており、クラスタがデプロイされ、設定されると(Rancher Turtlesオペレータ経由)管理クラスタに自動的にインポートされます。



注記

大規模なデプロイメントの規模に対応し、地理的に分散した環境における帯域幅とレイテンシの問題を最適化し、停止時や管理クラスタのアップグレード時の混乱を最小限に抑えるために、複数の管理クラスタを実装することが推奨されます。現在の管理クラスタのスケーラビリティの限界とシステム要件については、[こちら \(https://ranchermanager.docs.rancher.com/v2.12/getting-started/installation-and-upgrade/installation-requirements\)](https://ranchermanager.docs.rancher.com/v2.12/getting-started/installation-and-upgrade/installation-requirements) をご覧ください。

4 一般的なEdgeデプロイメントパターン

動作環境とライフサイクル要件はさまざまであるため、SUSEでは、SUSE Edgeを運用する市場セグメントやユースケースに大まかに一致する別個のデプロイメントパターンを多数サポートしています。また、これらの各デプロイメントパターンに対応するクイックスタートガイドを作成し、ユーザのニーズに基づいてSUSE Edgeプラットフォームに習熟できるようにしています。以下に、現在サポートされている3つのデプロイメントパターンを、各クイックスタートページへのリンクとともに説明します。

4.1 ダイレクトネットワークプロビジョニング

ダイレクトネットワークプロビジョニングでは、デプロイ先のハードウェアの詳細がわかっている場合に、アウトオブバンド管理インタフェースに直接アクセスして、プロビジョニングプロセス全体をオーケストレーションして自動化します。このシナリオで顧客が期待するソリューションとは、エッジサイトを一元的な場所から完全に自動化してプロビジョニングす

ることができ、ブートイメージの作成をはるかに上回る機能を備えていて、エッジロケーションでの手動操作を最小限に抑えられるソリューションです。つまり、ラックに搭載して電源をオンにし、必要なネットワークを物理ハードウェアに接続するだけで、自動化プロセスによってアウトオブバンド管理(Redfish APIなど)を介してマシンの電源が投入され、ユーザの介入なしにインフラストラクチャのプロビジョニング、オンボーディング、デプロイメントが処理されるソリューションです。これが機能するための鍵は、管理者がシステムを把握している、つまりどのハードウェアがどこにあるかを管理者が把握していることと、デプロイメントが中央で処理されることが想定されていることです。

このソリューションは最も堅牢です。管理者がハードウェアの管理インタフェースを直接操作して既知のハードウェアを扱うことに加え、ネットワークの利用可否に対する制約が少ないためです。機能面では、このソリューションは、Cluster APIとMetal³を広範に使用して、ベアメタルからオペレーティングシステム、Kubernetes、階層化アプリケーションまでを自動プロビジョニングし、デプロイメント後にSUSE Edgeの他の一般的なライフサイクル管理機能にリンクする機能を提供します。このソリューションのクイックスタートについては、[第1章「Metal³を使用したBMCの自動デプロイメント」](#)を参照してください。

4.2 「Phone Home」 ネットワークプロビジョニング

場合によっては、中央管理クラスタでハードウェアを直接管理できない環境で運用することがあります(たとえば、リモートネットワークがファイアウォールの背後にある場合や、アウトオブバンド管理インタフェースがない場合などがあり、エッジでよく見られる「PC」タイプのハードウェアで一般的です)。このシナリオの場合のために、SUSEでは、ハードウェアのブートストラップ時にその配置先がわかっていなくても、クラスタとそのワークロードをリモートでプロビジョニングできるツールを提供しています。エッジコンピューティングについて考える場合、ほとんどの人はこう考えます。エッジコンピューティングとは、不明な部分がある数千あるいは数万台のシステムがエッジロケーションで起動し、安全にPhone Home通信を行い、そのシステムの身元を検証し、実行すべき処理についての指示を受信することです。ここで要件として期待されるのは、工場でマシンを事前イメージングしたり、USBなどでブートイメージをアタッチしたりする以外には、ユーザがほとんど介入しなくてもプロビジョニングとライフサイクル管理ができることです。この領域での主な課題は、こうしたデバイスの規模、一貫性、セキュリティ、ライフサイクルに対処することです。

このソリューションでは、非常に柔軟で一貫性のある方法でシステムをプロビジョニングおよびオンボーディングできます。システムの種類、タイプや仕様、初回電源投入日時などは問いません。SUSE Edgeでは、Edge Image Builderを使用してシステムを非常に柔軟にカスタマイズできます。また、ノードのオンボーディングとKubernetesのプロビジョニングにはRancherのElementalが提供する登録機能を活用するとともに、オペレーティングシステ

ムへのパッチの適用にはSUSE Multi-Linux Managerを活用します。このソリューションのクイックスタートについては、第2章「Elementalを使用したリモートホストのオンボーディング」を参照してください。

4.3 イメージベースのプロビジョニング

スタンドアロン環境、エアギャップ環境、またはネットワークが制限された環境で運用する必要があるお客様向けに、SUSE Edgeでは、必要なデプロイメントアーティファクトがすべて含まれる、完全にカスタマイズされたインストールメディアを生成できるソリューションを提供しています。これにより、シングルノードとマルチノード両方の高可用性Kubernetesクラスタを、必要なワークロードと追加の階層化コンポーネントを含めてエッジに設定できます。これはすべて、外部とのネットワーク接続や集中管理プラットフォームの介入なしに行うことができます。ユーザエクスペリエンスは、インストールメディアをターゲットシステムに提供するという点では「Phone Home」ソリューションによく似ていますが、このソリューションは「インプレースでブートストラップ」する点異なります。このシナリオでは、生成されたクラスタをRancherに接続して継続的に管理する(つまり、大幅な再設定や再デプロイメントなしに、「非接続」動作モードから「接続」動作モードに移行する)ことも、分離した状態のまま動作を続行することもできます。どちらの場合も、一貫した同じメカニズムを適用してライフサイクル操作を自動化できることに注意してください。

さらに、このソリューションを使用すると、「ダイレクトネットワークプロビジョニング」モデルと「Phone Homeネットワークプロビジョニング」モデルの両方をサポートする集中型インフラストラクチャをホストできる管理クラスタを迅速に作成することもできます。この方法では、あらゆるタイプのエッジインフラストラクチャを最も迅速・簡単にプロビジョニングできます。このソリューションでは、SUSE Edge Image Builderの機能を多用して、完全にカスタマイズされた無人インストールメディアを作成します。クイックスタートについては、第3章「Edge Image Builderを使用したスタンドアロンクラスタ」を参照してください。

5 SUSE Edge Stack Validation

すべてのSUSE Edgeリリースは、緊密に統合され、徹底的に検証されたコンポーネントで構成されており、1つのバージョンとして管理されています。コンポーネント間の統合をテストするだけでなく、強制的な障害シナリオ下でシステムが期待どおりに動作することを保証する継続的な統合とスタック検証の一環として、SUSE Edgeチームはすべてのテスト実行と結果を公開しています。結果とすべての入力パラメータは[ci.edge.suse.com \(https://ci.edge.suse.com\)](https://ci.edge.suse.com)でご確認いただけます。

6 コンポーネントの全リスト

コンポーネントの全リストと、各コンポーネントの概要説明へのリンク、およびSUSE Edgeでの使用方法については、以下をご覧ください。

- Rancher (第5章 「Rancher」)
- Rancher Dashboard拡張機能(第6章 「Rancher Dashboard拡張機能」)
- Rancher Turtles (第7章 「Rancher Turtles」)
- SUSE Multi-Linux Manager
- Fleet (第8章 「Fleet」)
- SUSE Linux Micro (第9章 「SUSE Linux Micro」)
- Metal³ (第10章 「Metal³」)
- Edge Image Builder (第11章 「Edge Image Builder」)
- NetworkManager Configurator (第12章 「Edgeネットワーキング」)
- Elemental (第13章 「Elemental」)
- K3s (第14章 「K3s」)
- RKE2 (第15章 「RKE2」)
- SUSE Storage (第16章 「SUSE Storage」)
- SUSE Security (第17章 「SUSE Security」)
- MetalLB (第18章 「MetalLB」)
- KubeVirt (第20章 「Edge Virtualization」)
- System Upgrade Controller (第21章 「System Upgrade Controller」)
- Upgrade Controller (第22章 「Upgrade Controller」)

| クイックスタート

- 1 Metal³を使用したBMCの自動デプロイメント 2
- 2 Elementalを使用したリモートホストのオンボーディング 25
- 3 Edge Image Builderを使用したスタンドアロンクラスタ 43
- 4 SUSE Multi-Linux Manager 58

クイックスタートはこちら

1 Metal³を使用したBMCの自動デプロイメント

Metal³は、Kubernetesにベアメタルインフラストラクチャ管理機能を提供するCNCFプロジェクト (<https://metal3.io/>)  です。

Metal³は、Redfish (<https://www.dmtf.org/standards/redfish>)  などのアウトオブバンドプロトコルを介した管理をサポートするベアメタルサーバのライフサイクルを管理するためのKubernetesネイティブリソースを提供します。

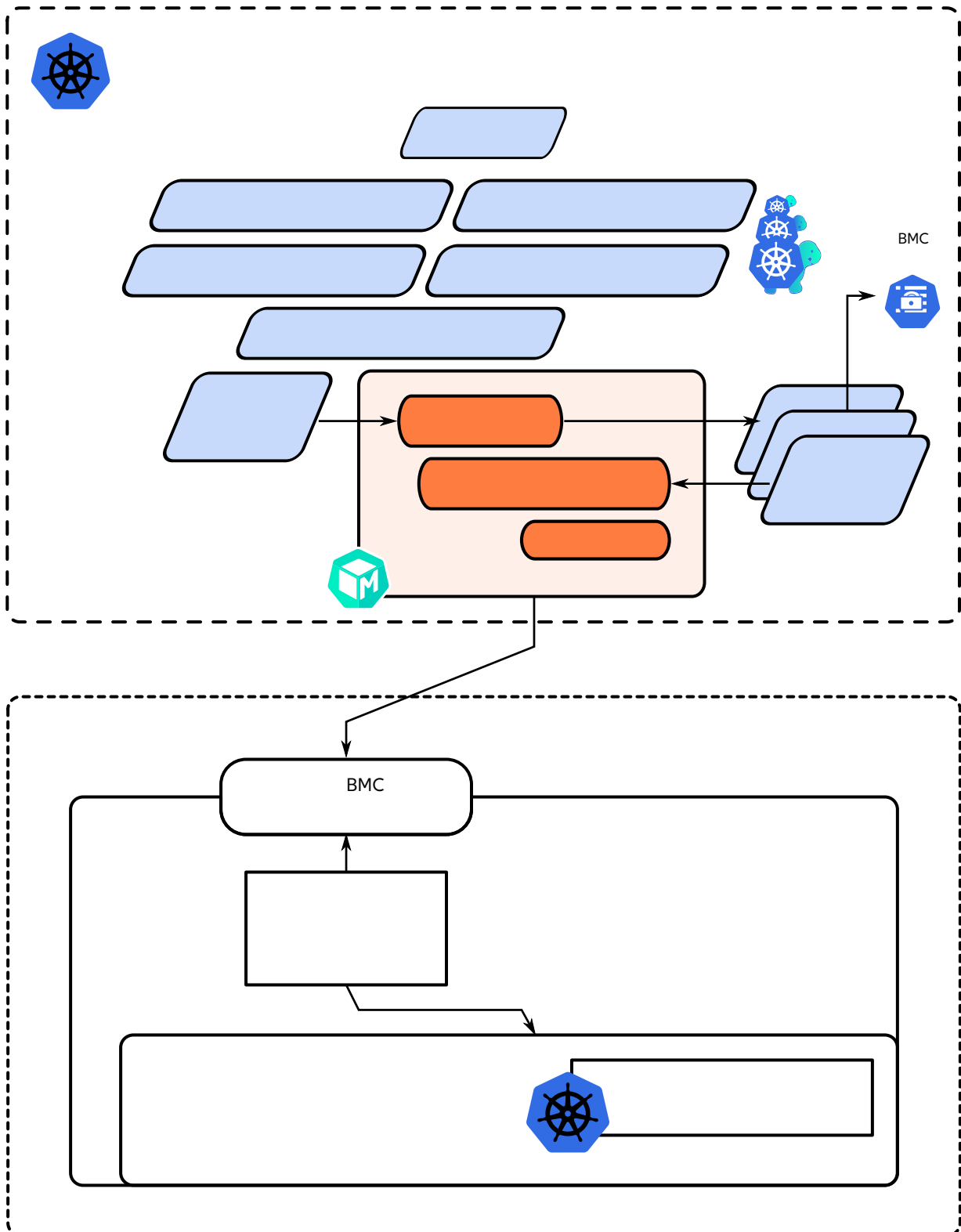
また、Cluster API (CAPI) (<https://cluster-api.sigs.k8s.io/>)  も十分にサポートされており、広く採用されているベンダニュートラルなAPIを使用して、複数のインフラストラクチャプロバイダにわたってインフラストラクチャリソースを管理できます。

1.1 この方法を使用する理由

この方法は、ターゲットハードウェアがアウトオブバンド管理をサポートしていて、完全に自動化されたインフラストラクチャ管理フローが望まれるシナリオで役立ちます。

管理クラスタは宣言型APIを提供するように設定されており、このAPIによってダウンストリームクラスタのベアメタルサーバのインベントリと状態を管理できます。これには、自動検査、クリーニング、プロビジョニング/プロビジョニング解除も含まれます。

1.2 高レベルアーキテクチャ



1.3 前提条件

ダウストリームクラスタのサーバハードウェアとネットワーキングに関連する固有の制約がいくつかあります。

- 管理クラスタ
 - ターゲットサーバ管理/BMC APIへのネットワーク接続が必要
 - ターゲットサーバのコントロールプレーンネットワークへのネットワーク接続が必要
 - マルチノード管理クラスタの場合、追加の予約済みIPアドレスが必要
- 制御対象ホスト
 - Redfish、iDRAC、またはiLOのインタフェースを介したアウトオブバンド管理のサポートが必要
 - 仮想メディアを使用したデプロイメントのサポートが必要(PXEは現在サポートされていない)
 - Metal³プロビジョニングAPIにアクセスするために管理クラスタへのネットワーク接続が必要

ツールがいくつか必要です。ツールは管理クラスタにインストールするか、管理クラスタにアクセス可能なホストにインストールできます。

- Kubectl (<https://kubernetes.io/docs/reference/kubectl/kubectl/>) 、Helm (<https://helm.sh>) 、およびClusterctl (<https://cluster-api.sigs.k8s.io/user/quick-start.html#install-clusterctl>) 
- Podman (<https://podman.io>)  や Rancher Desktop (<https://rancherdesktop.io>)  などのコンテナランタイム

SL-Micro.x86_64-6.1-Base-GM.raw OSイメージファイルはSUSE Customer Center (<https://scc.suse.com/>)  またはSUSEダウンロードページ (<https://www.suse.com/download/sle-micro/>)  からダウンロードする必要があります。

1.4 デプロイメント

1.4.1 管理クラスタのセットアップ

管理クラスタをインストールし、Metal³を使用する基本的な手順は次のとおりです。

1. RKE2管理クラスタをインストールします。
2. Rancherのインストール
3. ストレージプロバイダをインストールします(オプション)。
4. Metal³の依存関係をインストールします。
5. Rancher Turtles経由でCAPIの依存関係をインストールします。
6. ダウンストリームクラスタホスト用のSLEMicro OSイメージを構築します。
7. BareMetalHost CRを登録し、ベアメタルのインベントリを定義します。
8. CAPIリソースを定義して、ダウンストリームクラスタを作成します。

このガイドでは、既存のRKE2クラスタとRancher (cert-managerを含む)が、たとえばEdge Image Builder (第11章「Edge Image Builder」)を使用してインストールされていることを前提としています。



ヒント

ここでの手順は、管理クラスタのドキュメント(第40章「管理クラスタの設定」)で説明されているように、完全に自動化することもできます。

1.4.2 Metal³の依存関係のインストール

cert-managerがまだRancherのインストールの一部としてインストールされていない場合は、cert-managerをインストールして実行する必要があります。

永続ストレージプロバイダをインストールする必要があります。SUSE Storageを推奨しますが、開発/PoC環境ではlocal-path-provisionerを使用することもできます。以下の手順は、StorageClassがデフォルトとしてマーク (<https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/>) されていることを前提としています。マークされていない場合は、Metal³チャートに追加の設定が必要です。

追加のIPが必要です。このIPはMetalLB (<https://metallb.universe.tf/>)  によって管理され、Metal³管理サービスに一貫したエンドポイントを提供します。このIPは、コントロールプレーンサブネットに属していて、静的設定用に予約されている必要があります(どのDHCPプールにも属してはなりません)。



ヒント

管理クラスタがシングルノードである場合、MetalLBを介して管理されるフローティングIPを追加する必要はありません。1.7.1項「[シングルノード設定](#)」を参照してください。

1. まず、MetalLBをインストールします。

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

2. 続いて、次のように、`STATIC_IRONIC_IP`として定義された予約済みIPを使用して、`IPAddressPool`と `L2Advertisement`を定義します。

```
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. これでMetal³をインストールできます。

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

4. initコンテナがこのデプロイメントで実行されるまで約2分かかる場合があるため、続行する前にPodがすべて実行されていることを確認します。

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

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



警告

`metal3-system`ネームスペースのすべてのPodが実行されるまで、以下の手順に進まな
いでください。

1.4.3 Cluster APIの依存関係のインストール

Cluster APIの依存関係は、Rancher Turtles Helmチャートで管理されます。

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

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

```
--create-namespace \  
-f values.yaml
```

しばらくすると、コントローラPodがcapi-system、capm3-system、rke2-bootstrap-system、およびrke2-control-plane-systemの各ネームスペースで実行されているはずです。

1.4.4 ダウンストリームクラスタイメージの準備

Kiwi (第28章「Kiwiを使用したSUSE Linux Microの更新イメージの構築」)とEdge Image Builder (第11章「Edge Image Builder」)を使用して、ダウンストリームクラスタホスト上にプロビジョニングされる、変更されたSLEMicroゴールデンイメージを準備します。

このガイドではダウンストリームクラスタをデプロイするために必要な最小限の設定について説明します。

1.4.4.1 イメージの設定



注記

クラスタの作成に必要な最初の手順として、まず第28章「Kiwiを使用したSUSE Linux Microの更新イメージの構築」に従って、新しいイメージを構築してください。

Edge Image Builderを実行すると、そのホストからディレクトリがマウントされるため、ターゲットイメージの定義に使用する設定ファイルを保存するディレクトリ構造を作成する必要があります。

- `downstream-cluster-config.yaml`はイメージ定義ファイルです。詳細については、第3章「Edge Image Builderを使用したスタンドアロンクラスタ」を参照してください。
- ダウンロードされたゴールデンイメージはxzで圧縮されているので、`unxz`で展開し、`base-images`フォルダの下にコピー/移動する必要があります。
- `network`フォルダはオプションです。詳細については、1.4.5.1.1項「静的ネットワーク設定用の追加スクリプト」を参照してください。
- `custom/scripts`ディレクトリには、初回ブート時に実行するスクリプトが含まれます。現在、デプロイメントのOSルートパーティションのサイズを変更するには、`01-fix-growfs.sh`スクリプトが必要です。


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

1.4.4.1.1 ダウンストリームクラスタイメージ定義ファイル

downstream-cluster-config.yamlファイルは、ダウンストリームクラスタイメージの主要な設定ファイルです。次に、Metal³を介したデプロイメントの最小例を示します。

```
apiVersion: 1.3
image:
  imageType: raw
  arch: x86_64
  baseImage: SL-Micro.x86_64-6.1-Base-GM.raw
  outputImageName: SLE-Micro-eib-output.raw
operatingSystem:
  time:
    timezone: Europe/London
    ntp:
      forceWait: true
      pools:
        - 2.suse.pool.ntp.org
      servers:
        - 10.0.0.1
        - 10.0.0.2
  kernelArgs:
    - ignition.platform.id=openstack
    - net.ifnames=1
  systemd:
    disable:
      - rebootmgr
      - transactional-update.timer
      - transactional-update-cleanup.timer
  users:
    - username: root
      encryptedPassword: $ROOT_PASSWORD
      sshKeys:
        - $USERKEY1
  packages:
    packageList:
      - jq
sccRegistrationCode: $SCC_REGISTRATION_CODE
```

ここで、`$SCC_REGISTRATION_CODE`は[SUSE Customer Center \(https://scc.suse.com/\)](https://scc.suse.com/) からコピーした登録コードで、パッケージリストには必要なjqが含まれています。

`$ROOT_PASSWORD`はルートユーザの暗号化パスワードで、テスト/デバッグに役立ちます。このパスワードは、`openssl passwd -6 PASSWORD`コマンドで生成できます。

運用環境では、`$USERKEY1`を実際のSSHキーに置き換えて、`users`ブロックに追加できるSSHキーを使用することをお勧めします。



注記

`net.ifnames=1`は、[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) を有効にします。

これはMetal³チャートのデフォルト設定と一致しますが、この設定は、設定されたチャートの`predictableNicNames`の値と一致する必要があります。

また、`ignition.platform.id=openstack`は必須であり、この引数がないと、Metal³の自動化フローでIgnitionによるSUSE Linux Microの設定が失敗することにも注意してください。

`time` セクションはオプションですが、証明書とクロックスキューに関する潜在的な問題を回避するように設定することを強くお勧めします。この例で指定されている値は説明のみを目的としています。ご自身の特定の要件に合わせて調整してください。

1.4.4.1.2 Growfsスクリプト

現在、プロビジョニング後の初回ブート時にディスクサイズに合わせてファイルシステムを拡張するには、カスタムスクリプト(`custom/scripts/01-fix-growfs.sh`)が必要です。`01-fix-growfs.sh`スクリプトには次の情報が含まれます。

```
#!/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\}\([0-9\+\)\$\/1/\)')")"
    ret=0
    growpart "$parent_dev" "$partnum" || ret=$?
    [ $ret -eq 0 ] || [ $ret -eq 1 ] || exit 1
    /usr/lib/systemd/systemd-growfs "$mnt"
}
```



注記

同じアプローチを使用して、プロビジョニングプロセス中に実行する独自のカスタムスクリプトを追加します。詳細については、[第3章「Edge Image Builderを使用したスタンドアロンクラスタ」](#)を参照してください。

1.4.4.2 イメージの作成

これまでのセクションに従ってディレクトリ構造を準備したら、次のコマンドを実行してイメージを構築します。

```
podman run --rm --privileged -it -v $PWD:/eib \
registry.suse.com/edge/3.4/edge-image-builder:1.3.0 \
build --definition-file downstream-cluster-config.yaml
```

これにより、上記の定義に基づいて、SLE-Micro-eib-output.rawという名前の出力イメージファイルが作成されます。

その後、この出力イメージをWebサーバ経由で利用できるようにする必要があります。その際、Metal3チャートを使用して有効にしたメディアサーバコンテナ([注記](#))か、ローカルにアクセス可能な他のサーバのいずれかを使用します。以下の例では、このサーバをimagecache.local:8080として参照します。



注記

EIBイメージをダウンストリームクラスタにデプロイする際は、Metal3MachineTemplateオブジェクトにイメージのSHA256ハッシュ値を含める必要があります。これは次のように生成できます：

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

1.4.5 BareMetalHostインベントリの追加

自動デプロイメント用にベアメタルサーバを登録するには、リソースを2つ作成する必要があります。BMCアクセス資格情報を保存するシークレットと、BMC接続とその他の詳細を定義するMetal³ BareMetalHostリソースです。

```

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:
  architecture: x86_64
  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

```

次の点に注意してください。

- シークレットのユーザ名/パスワードはbase64でエンコードされている必要があります。また、末尾に改行を含めないでください(たとえば、単なるechoではなく、echo -nを使用してください)。
- cluster-roleラベルは、この時点で設定することも、後でクラスタの作成時に設定することもできます。以下の例では、control-planeまたはworkerを想定しています。
- bootMACAddressは、ホストのコントロールプレーンNICに一致する有効なMACである必要があります。
- bmcのアドレスはBMC管理APIへの接続です。次のアドレスがサポートされています。
 - redfish-virtualmedia://<IP ADDRESS>/redfish/v1/Systems/<SYSTEM ID>: Redfish仮想メディア(たとえば、SuperMicro)
 - idrac-virtualmedia://<IP ADDRESS>/redfish/v1/Systems/System.Embedded.1: Dell iDRAC
- BareMetalHost APIの詳細については、アップストリームのAPIドキュメント (<https://github.com/metal3-io/baremetal-operator/blob/main/docs/api.md>)  を参照してください。

1.4.5.1 静的IPの設定

上記のBareMetalHostの例では、DHCPでコントロールプレーンネットワークの設定を提供することを想定していますが、静的IPなどの手動設定が必要なシナリオでは、以下に説明するように追加の設定を指定できます。

1.4.5.1.1 静的ネットワーク設定用の追加スクリプト

Edge Image Builderでゴールデンイメージを作成する際には、`network`フォルダ内に次の`configure-network.sh`ファイルを作成します。

このファイルにより、初回ブート時に設定ドライブのデータを使用して、**NM Configurator ツール** (<https://github.com/suse-edge/nm-configurator>) を使ってホストネットワーキングを設定します。

```
#!/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
```

1.4.5.1.2 ホストネットワーク設定の追加シークレット

NM Configurator (第12章「Edgeネットワーキング」)でサポートされているnmstate (<https://nmstate.io/>)形式のデータを含む追加シークレットをホストごとに定義できます。

その後、このシークレットは、BareMetalHostリソースでpreprovisioningNetworkDataNameの指定フィールドを使用して参照されます。

```
apiVersion: v1
kind: Secret
metadata:
  name: controlplane-0-networkdata
type: Opaque
stringData:
  networkData: |
    interfaces:
      - name: enpls0
        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: enpls0
    ---
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: controlplane-0
  labels:
    cluster-role: control-plane
spec:
  preprovisioningNetworkDataName: controlplane-0-networkdata
# Remaining content as in previous example
```



注記

状況によっては、MACアドレスが省略される場合があります。詳細については、[12.5.8 項「統合されたノード設定」](#)を参照してください。

1.4.5.2 BareMetalHostの準備

上記の説明に従ってBareMetalHostリソースと関連するシークレットを作成すると、次のようにホスト準備ワークフローがトリガされます。

- ターゲットホストのBMCに接続された仮想メディアによってramdiskイメージがブートされる
- ramdiskがハードウェア詳細を検査し、ホストをプロビジョニング用に準備する(たとえば、ディスクから以前のデータを消去する)
- このプロセスが完了すると、BareMetalHostの`status.hardware`フィールドのハードウェア詳細が更新され、検証可能になる

このプロセスには数分かかる場合がありますが、完了すると、BareMetalHostの状態が`available`になります。

```
% kubectl get baremetalhost
NAME                STATE      CONSUMER  ONLINE  ERROR  AGE
controlplane-0      available
worker-0            available
```

1.4.6 ダウンストリームクラスタの作成

続いて、ダウンストリームクラスタを定義するCluster APIリソースと、BareMetalHostリソースをプロビジョニングしてからブートストラップを実行してRKE2クラスタを形成するマシンリソースを作成します。

1.4.7 コントロールプレーンのデプロイメント

コントロールプレーンをデプロイするために、以下のリソースを含む次のようなyamlマニフェストを定義します。

- クラスタリソースでは、クラスタ名、ネットワーク、およびコントロールプレーン/インフラストラクチャプロバイダのタイプ(この場合はRKE2/Metal3)を定義します。
- Metal3Clusterでは、コントロールプレーンのエンドポイント(シングルノードの場合はホストIP、マルチノードの場合はLoadBalancerエンドポイント。この例ではシングルノードを想定)を定義します。
- RKE2ControlPlaneでは、RKE2のバージョンと、クラスタのブートストラップ時に必要な追加設定を定義します。
- Metal3MachineTemplateではBareMetalHostリソースに適用するOSイメージを定義し、hostSelectorでは使用するBareMetalHostを定義します。
- Metal3DataTemplateでは、BareMetalHostに渡す追加のメタデータを定義します(networkDataは現在のところEdgeソリューションではサポートされていないことに注意してください)。



注記

簡潔にするため、この例では、BareMetalHostのIPアドレスが192.168.125.200で設定されている単一ノードのコントロールプレーンを想定しています。より高度なマルチノードの例については、[第42章「完全に自動化されたダイレクトネットワークプロビジョニング」](#)を参照してください。

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: sample-cluster
  namespace: default
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - 192.168.0.0/18
    services:
      cidrBlocks:
        - 10.96.0.0/12
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: RKE2ControlPlane
    name: sample-cluster
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3Cluster
    name: sample-cluster
---
```



```

apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3Cluster
metadata:
  name: sample-cluster
  namespace: default
spec:
  controlPlaneEndpoint:
    host: 192.168.125.200
    port: 6443
  noCloudProvider: true
---
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
metadata:
  name: sample-cluster
  namespace: default
spec:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3MachineTemplate
    name: sample-cluster-controlplane
  replicas: 1
  version: v1.33.3+rke2r1
  rolloutStrategy:
    type: "RollingUpdate"
    rollingUpdate:
      maxSurge: 0
  agentConfig:
    format: ignition
    kubelet:
      extraArgs:
        - provider-id=metal3://BAREMETALHOST_UUID
  additionalUserData:
    config: |
      variant: fcos
      version: 1.4.0
      systemd:
        units:
          - name: rke2-preinstall.service
            enabled: true
            contents: |
              [Unit]
              Description=rke2-preinstall
              Wants=network-online.target
              Before=rke2-install.service
              ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
              [Service]
              Type=oneshot
              User=root
              ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"

```

```

        ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/${jq -r .uuid /mnt/
openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
        ExecStart=/bin/sh -c "echo \"node-name: ${jq -r .name /mnt/openstack/
latest/meta_data.json}\" >> /etc/rancher/rke2/config.yaml"
        ExecStartPost=/bin/sh -c "umount /mnt"
        [Install]
        WantedBy=multi-user.target
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3MachineTemplate
metadata:
  name: sample-cluster-controlplane
  namespace: default
spec:
  template:
    spec:
      dataTemplate:
        name: sample-cluster-controlplane-template
      hostSelector:
        matchLabels:
          cluster-role: control-plane
      image:
        checksum: http://imagecache.local:8080/SLE-Micro-eib-output.raw.sha256
        checksumType: sha256
        format: raw
        url: http://imagecache.local:8080/SLE-Micro-eib-output.raw
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3DataTemplate
metadata:
  name: sample-cluster-controlplane-template
  namespace: default
spec:
  clusterName: sample-cluster
  metaData:
    objectNames:
      - key: name
        object: machine
      - key: local-hostname
        object: machine
      - key: local_hostname
        object: machine

```

自身の環境に合わせて調整したら、kubectlを使用して例を適用し、その後 clusterctlを使用してクラスタのステータスを監視できます。

```

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

# Wait for the cluster to be provisioned
% clusterctl describe cluster sample-cluster

```

NAME	READY	SEVERITY	REASON	SINCE
MESSAGE				
Cluster/sample-cluster	True			22m
└ClusterInfrastructure - Metal3Cluster/sample-cluster	True			27m
└ControlPlane - RKE2ControlPlane/sample-cluster	True			22m
└Machine/sample-cluster-chflc	True			23m

1.4.8 ワーカー/コンピュータのデプロイメント

コントロールプレーンのデプロイメントと同様に、次のリソースを含むYAMLマニフェストを定義します。

- MachineDeploymentでは、レプリカ(ホスト)の数とブートストラップ/インフラストラクチャプロバイダ(この場合はRKE2/Metal3)を定義します。
- RKE2ConfigTemplateでは、エージェントホストのブートストラップ用のRKE2のバージョンと初回ブート設定を記述します。
- Metal3MachineTemplateではBareMetalHostリソースに適用するOSイメージを定義し、hostSelectorでは使用するBareMetalHostを定義します。
- Metal3DataTemplateでは、BareMetalHostに渡す追加のメタデータを定義します(networkDataは現在サポートされていないことに注意してください)。

```
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: v1.33.3+rke2r1
  ---
apiVersion: bootstrap.cluster.x-k8s.io/v1alpha1
kind: RKE2ConfigTemplate
metadata:
  name: sample-cluster-workers
  namespace: default
spec:
  template:
    spec:
      agentConfig:
        format: ignition
        version: v1.33.3+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

```

上記の例をコピーし、自身の環境に合わせて調整したら、kubect1を使用して適用し、clusterctlでクラスタのステータスを監視できます。

```

% kubectl apply -f rke2-agent.yaml

# Wait for the worker nodes to be provisioned
% clusterctl describe cluster sample-cluster

```

NAME	READY	SEVERITY	REASON	SINCE
MESSAGE				
Cluster/sample-cluster	True			25m
└ClusterInfrastructure - Metal3Cluster/sample-cluster	True			30m
└ControlPlane - RKE2ControlPlane/sample-cluster	True			25m
└Machine/sample-cluster-chflc	True			27m
└Workers				
└MachineDeployment/sample-cluster	True			22m
└└Machine/sample-cluster-56df5b4499-zfljj	True			23m

1.4.9 クラスタのプロビジョニング解除

ダウンストリームクラスタをプロビジョニング解除するには、上記の作成手順で適用したリソースを削除します。

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

これにより、BareMetalHostリソースのプロビジョニング解除がトリガされます。これには数分かかることがあり、その後リソースは再び利用可能な状態になります。

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

% kubectl get bmh
NAME                STATE             CONSUMER                                ONLINE  ERROR  AGE
controlplane-0      available         sample-cluster-controlplane-vlrt6     false         15m
worker-0            available         sample-cluster-workers-785x5         false         15m
```

1.5 既知の問題

- 現在、アップストリームのIPアドレス管理コントローラ (<https://github.com/metal3-io/ip-address-manager>) はサポートされていません。このコントローラには、SLEMicroで選択されているネットワーク設定ツールと初回ブートツールチェーンとの互換性がまだないためです。
- 関連して、IPAMリソースと、Metal3DataTemplateのnetworkDataフィールドは現在のところサポートされていません。
- redfish-virtualmediaを介したデプロイメントのみが現在サポートされています。
- Ironic Python Agent (IPA)とターゲットオペレーティングシステム(SL Micro 6.0/6.1)の間で、特に、デバイスに予測可能な名前を設定しようとする際に、ネットワークデバイス名の不整合が発生する可能性があります。

これが発生するのは、Ironic Python Agent (IPA)のカーネルが現在、ターゲットオペレーティングシステム(SL Micro 6.0/6.1)のカーネルと整合していないためです。これにより、ネットワークドライバに不整合が生じ、SL Microが想定する命名パターンとは異なる命名パターンでIPAがネットワークデバイスを検出できるようになります。

当面の回避策として、以下の2つの異なるアプローチが利用可能です。* ネットワーク設定で2つの異なるシークレットを作成します。1つはIPAが検出するデバイス名を使用したIPAで使用されるシークレットで、`BareMetalHost`定義の`preprovisioningNetworkDataName`として使用します。もう1つはSL Microが検出するデバイス名を使用したシークレットで、`BareMetalHost`定義の`networkData.name`として使用します。* 代わりに、生成された`nmconnection`ファイル上の他のインタフェースを参照するためにUUIDを使用します。詳細は、[ヒントとコツ \(..tips/metal3.adoc\)](https://suse.github.io/tips/metal3.adoc)  セクションを参照してください。

1.6 予定されている変更

- `networkData`フィールドを使用した、IPAMリソースと設定のサポートの有効化

1.7 追加のリソース

SUSE Telco Cloudドキュメント(第37章「[SUSE Telco Cloud](#)」)には、通信事業者のユースケースにおけるMetal³のより高度な使用例が記載されています。

1.7.1 シングルノード設定

管理クラスタがシングルノードであるテスト/PoC環境では、MetalLBを介して管理されるフローティングIPを追加する必要はありません。

このモードでは、管理クラスタAPIのエンドポイントが管理クラスタのIPになるため、DHCPを使用している場合はそのIPを予約するか、管理クラスタのIPが変更されないように静的に設定する必要があります(以下では`<MANAGEMENT_CLUSTER_IP>`と表記しています)。

このシナリオを有効にするために必要なMetal³チャートの値は次のとおりです。

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

1.7.2 仮想メディアISOをアタッチするためのTLSの無効化

一部のサーバベンダは、仮想メディアISOイメージをBMCにアタッチする際にSSL接続を検証しますが、Metal³のデプロイメント用に生成された証明書は自己署名されているため、問題が発生する可能性があります。この問題を回避するには、次のようなMetal³チャートの値を使用して、仮想メディアディスクをアタッチする場合にのみTLSを無効にすることができます。

```
global:
  enable_vmedia_tls: false
```

別の解決策は、CA証明書を使用してBMCを設定することです。この場合、`kubectl`を使用してクラスタから証明書を読み込むことができます。

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

これにより、証明書をサーバのBMCコンソールで設定できますが、そのプロセスはベンダ固有です(すべてのベンダで可能というわけではなく、可能でない場合は`enable_vmedia_tls`フラグが必要なことがあります)。

1.7.3 ストレージ設定

管理クラスタがシングルノードであるテスト/PoC環境では、永続ストレージは不要ですが、運用ユースケースでは管理クラスタにSUSE Storage (Longhorn)をインストールすることをお勧めします。これによりPodの再起動/再スケジュール時にMetal³に関連するイメージを保持できます。

この永続ストレージを有効にするために必要なMetal³チャート値は次のとおりです。

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

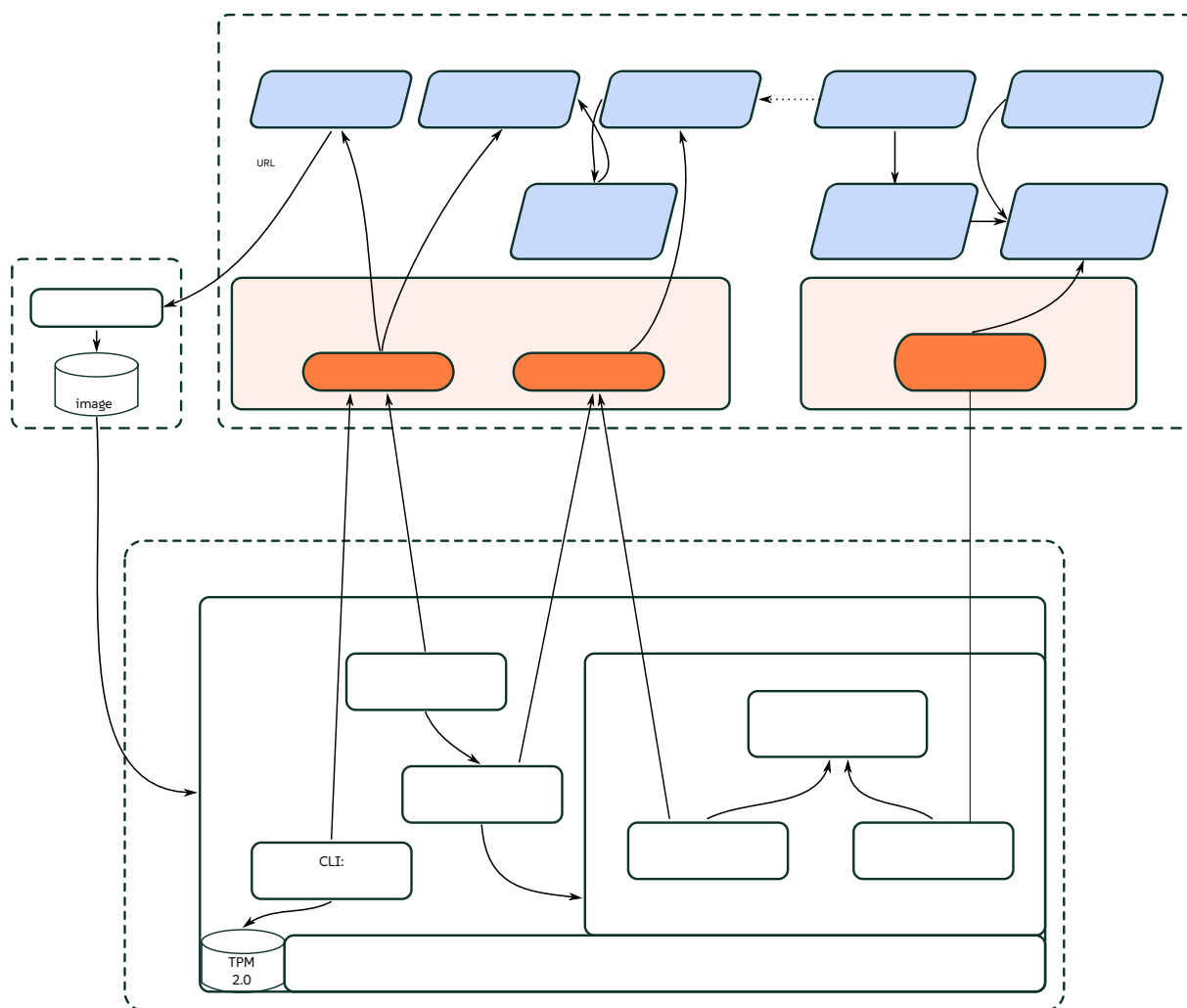
SUSE Telco Cloud管理クラスタのドキュメント(第40章「[管理クラスタの設定](#)」)には、永続ストレージを使用した管理クラスタの設定方法に関する詳細が記載されています。

2 Elementalを使用したリモートホストのオンボーディング

このセクションでは、SUSE Edgeの一部としての「Phone Homeネットワークプロビジョニング」ソリューションについて説明します。このソリューションは、Elementalを使用してノードのオンボーディングを支援します。Elementalは、Kubernetesを使用してリモートホスト登録と一元化された完全なクラウドネイティブOS管理を可能にするソフトウェアスタックです。SUSE Edgeスタックでは、Elementalの登録機能を使用して、リモートホストをRancherにオンボーディングできます。これにより、ホストを集中管理プラットフォームに統合し、そこからKubernetesクラスタに加えて、階層化コンポーネント、アプリケーション、およびそのライフサイクルをすべて共通の場所からデプロイおよび管理できるようになります。

このアプローチが役立つシナリオとしては、制御するデバイスが管理クラスタと同じネットワーク上にないか、アウトオブバンド管理コントローラが搭載されておらず、より直接的に制御できない場合や、さまざまな「不明」なシステムをエッジで多数ブートしており、それらを安全にオンボーディングして大規模に管理する必要がある場合が考えられます。これは、小売や産業用IoTなど、デバイスが設置されるネットワークをほとんど制御できない分野のユースケースによく見られるシナリオです。

2.1 高レベルアーキテクチャ



2.2 必要なリソース

このクイックスタートを実行するためのシステムと環境の最小要件を次に示します。

- 集中管理クラスタ(RancherとElementalをホストするクラスタ)用のホスト:
 - 開発またはテスト用の場合、最小8GBのRAMと20GBのディスク容量(運用環境での使用については[こちら \(https://ranchermanager.docs.rancher.com/v2.12/getting-started/installation-and-upgrade/installation-requirements#hardware-requirements\)](https://ranchermanager.docs.rancher.com/v2.12/getting-started/installation-and-upgrade/installation-requirements#hardware-requirements) を参照)
- プロビジョニングするターゲットノード、すなわちエッジデバイス(デモまたはテストの場合は仮想マシンを使用可能)
 - 最小4GBのRAM、2 CPUコア、20GBのディスク
- 管理クラスタの解決可能なホスト名、またはsslip.ioなどのサービスで使用する静的IPアドレス
- Edge Image Builderでインストールメディアを構築するためのホスト
 - SLES 15 SP6、openSUSE Leap 15.6、またはPodmanをサポートする他の互換性のあるオペレーティングシステムを実行している
 - Kubectl (<https://kubernetes.io/docs/reference/kubectl/kubectl/>)、Podman (<https://podman.io>)、およびHelm (<https://helm.sh>) がインストールされていること
- ブート用のUSBフラッシュドライブ(物理ハードウェアを使用する場合)
- 最新のSUSE Linux Micro 6.1 SelfInstall ISOイメージのダウンロードコピー([こちら \(https://www.suse.com/download/sle-micro/\)](https://www.suse.com/download/sle-micro/) から入手可能です)



注記

ターゲットマシンにある既存のデータはこのプロセスの一環として上書きされます。ターゲットデプロイメントノードに接続されているUSBストレージデバイスやディスク上のデータは、必ずバックアップしてください。

このガイドは、アップストリームクラスタをホストするためにDigital Oceanドロップレットを使用し、ダウンストリームデバイスとしてIntel NUCを使用して作成されています。インストールメディアの構築には、SUSE Linux Enterprise Serverを使用しています。

2.3 ブートストラップクラスタの構築

まず、RancherとElementalをホストできるクラスタを作成します。このクラスタは、ダウンストリームノードが接続されているネットワークからルーティングできる必要があります。

2.3.1 Kubernetesクラスタの作成

ハイパースケーラ(Azure、AWS、Google Cloudなど)を使用している場合、クラスタを設定する最も簡単な方法は、ハイパースケーラのビルトインツールを使用することです。このガイドでは、簡潔にするために、これらの各オプションのプロセスについては詳述しません。

ベアメタルや別のホスティングサービスにインストールしようとしていて、Kubernetesディストリビューションそのものも用意する必要がある場合は、[RKE2 \(https://docs.rke2.io/install/quickstart\)](https://docs.rke2.io/install/quickstart) を使用することをお勧めします。

2.3.2 DNSの設定

続行する前に、クラスタへのアクセスを設定する必要があります。クラスタ自体のセットアップと同様に、DNSの設定方法は、クラスタがホストされている場所によって異なります。



ヒント

DNSレコードの設定を扱わない場合(たとえば、これが一時的なテストサーバである場合)、代わりに[sslip.io \(https://sslip.io\)](https://sslip.io) などのサービスを使用できます。このサービスを使用すると、`<address>.sslip.io`を使用して任意のIPアドレスを解決できます。

2.4 Rancherのインストール

Rancherをインストールするには、作成したクラスタのKubernetes APIにアクセスする必要があります。これは、使用しているKubernetesのディストリビューションによって異なります。

RKE2の場合、kubeconfigファイルは`/etc/rancher/rke2/rke2.yaml`に書き込まれます。このファイルをローカルシステムに`~/.kube/config`として保存します。このファイルを編集して、外部にルーティング可能な正しいIPアドレスまたはホスト名を含めなければならない場合があります。

Rancherのドキュメント (<https://ranchermanager.docs.rancher.com/v2.12/getting-started/installation-and-upgrade/install-upgrade-on-a-kubernetes-cluster>) に記載されているコマンドを使用して、Rancherを簡単にインストールできます。

[cert-manager](https://cert-manager.io) (<https://cert-manager.io>) をインストールします。

```
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
```

次に、Rancher自体をインストールします。

```
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 2.12.1
```



注記

これを運用システムにする予定の場合は、cert-managerを使用して、実際の証明書 (Let's Encryptの証明書など)を設定してください。

設定したホスト名をブラウザし、使用した`bootstrapPassword`でRancherにログインします。ガイドに従って簡単なセットアッププロセスを完了します。

2.5 Elementalのインストール

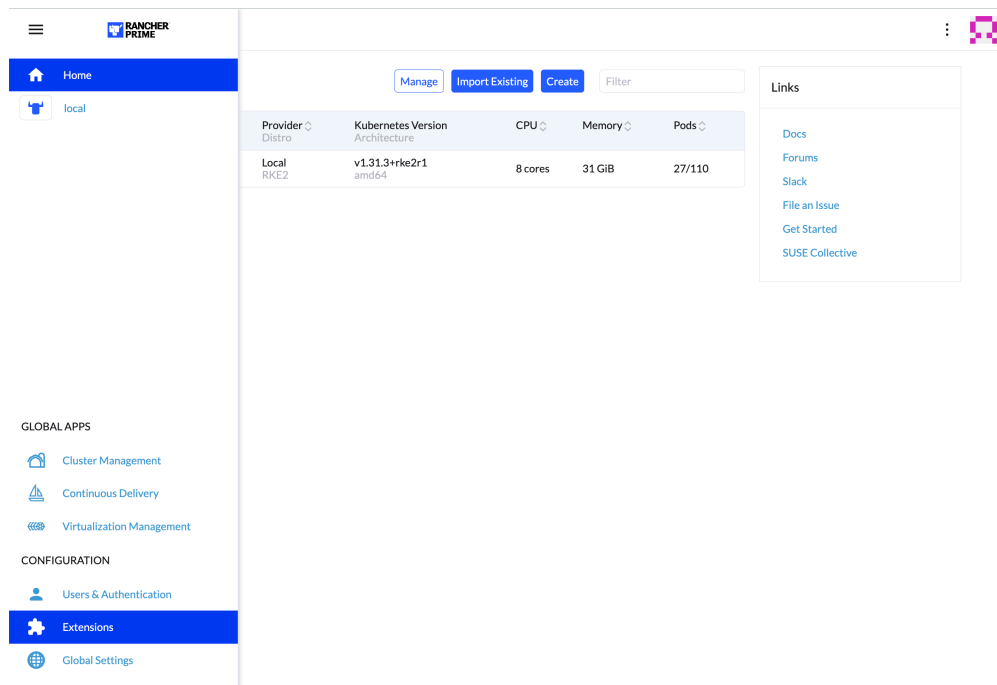
Rancherをインストールしたら、続いてElementalのオペレータと必要なCRDをインストールできます。Elemental用のHelmチャートはOCIアーティファクトとして公開されているため、インストールは他のチャートよりも若干シンプルです。Rancherのインストールに使用したものと同じシェルからインストールすることも、ブラウザでRancherのシェル内からインストールすることもできます。

```
helm install --create-namespace -n cattle-elemental-system \
```

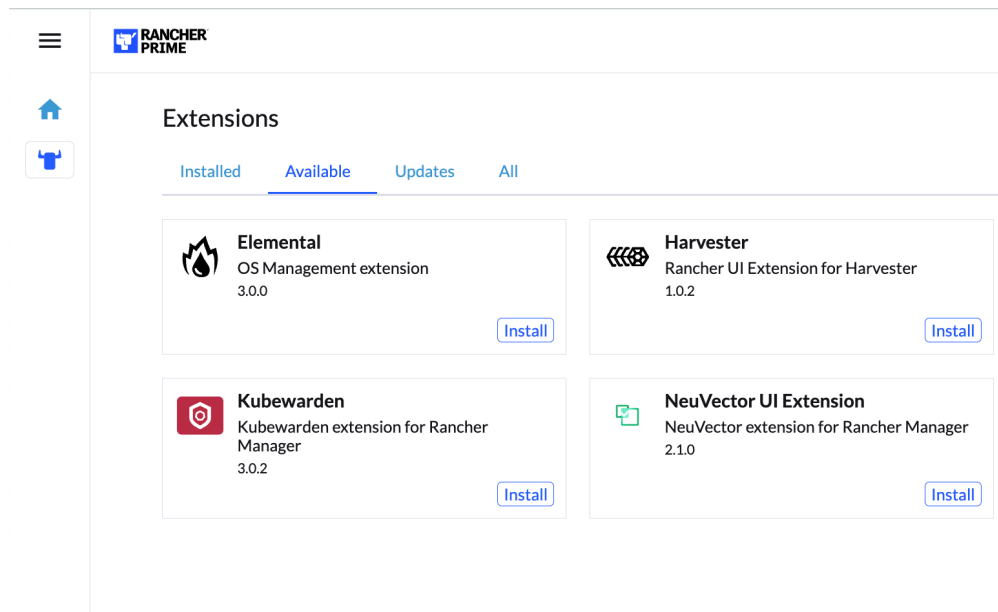
```
elemental-operator-crds \  
oci://registry.suse.com/rancher/elemental-operator-crds-chart \  
--version 1.7.3  
  
helm install -n cattle-elemental-system \  
elemental-operator \  
oci://registry.suse.com/rancher/elemental-operator-chart \  
--version 1.7.3
```

2.5.1 (オプション) Elemental UI拡張機能のインストール

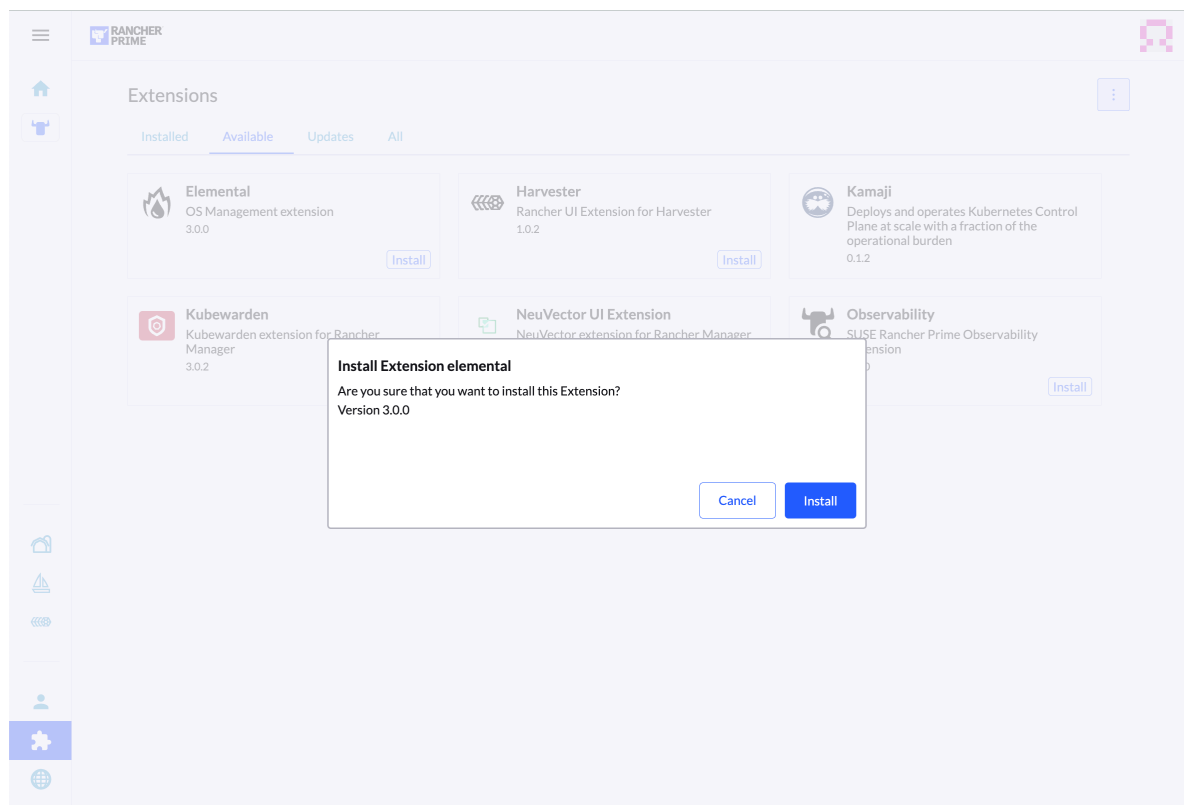
1. Elemental UIを使用するには、Rancherインスタンスにログインし、左上の三本線のメニューをクリックします。



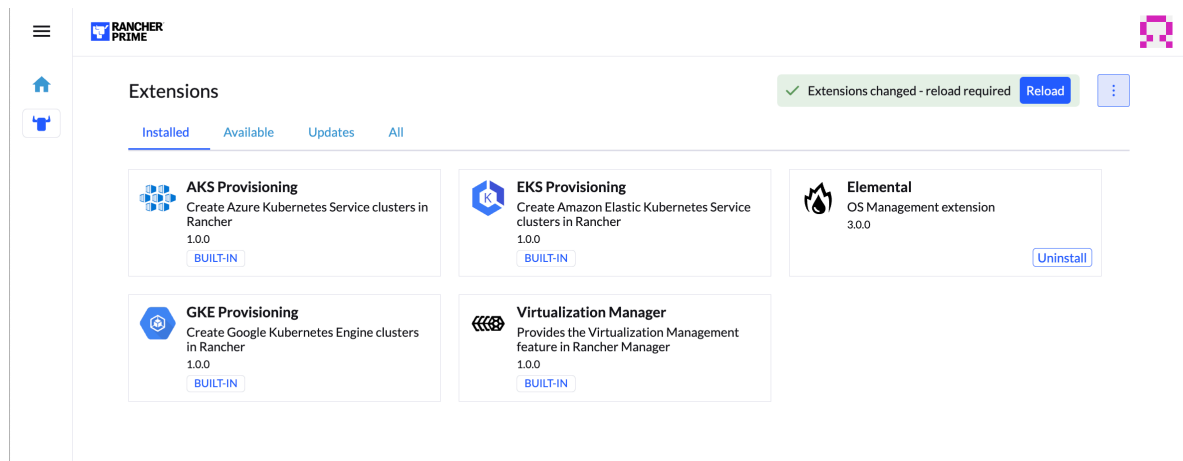
2. このページの「Available (使用可能)」タブから、Elementalカードの「Install (インストール)」をクリックします。



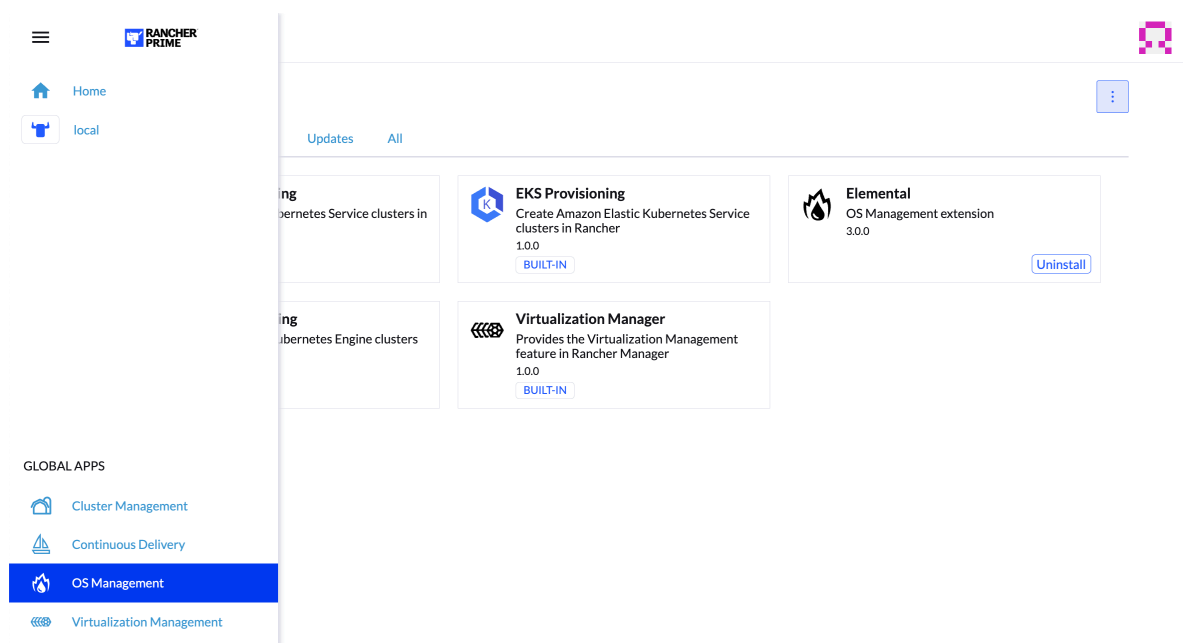
3. 拡張機能をインストールすることを確認します。



4. インストール後、ページを再ロードするよう求められます。



5. 再ロードすると、[OS Management (OS管理)] グローバルアプリからElemental拡張機能にアクセスできるようになります。



2.6 Elementalの設定

シンプルにするために、変数`$ELEM`を、設定ディレクトリを配置する場所のフルパスに設定することをお勧めします。

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


マシンがElementalに登録できるようにするために、`fleet-default`ネームスペースに`MachineRegistration`オブジェクトを作成する必要があります。

このオブジェクトの基本的なバージョンを作成してみましょう。

```
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
```



注記

この`cat`コマンドでは、各`$`をバックスラッシュ(`\`)でエスケープしています。このため、バッシュではテンプレート化されていません。手動でコピーする場合は、バックスラッシュを削除してください。

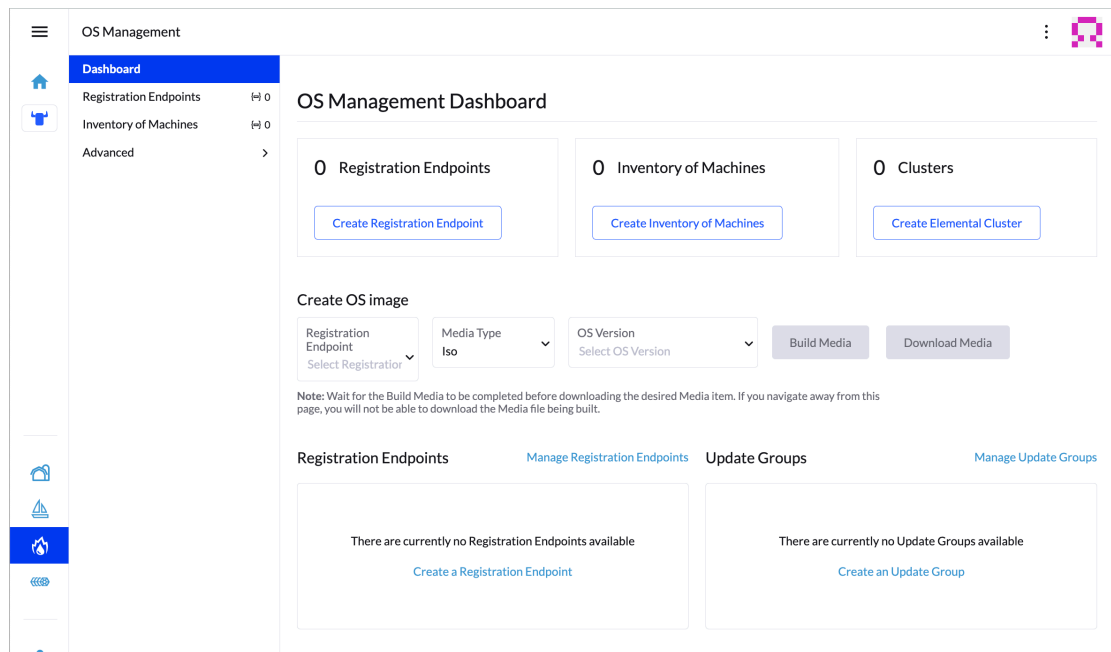
オブジェクトが作成されたら、割り当てられるエンドポイントを見つけてメモを取ります。

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

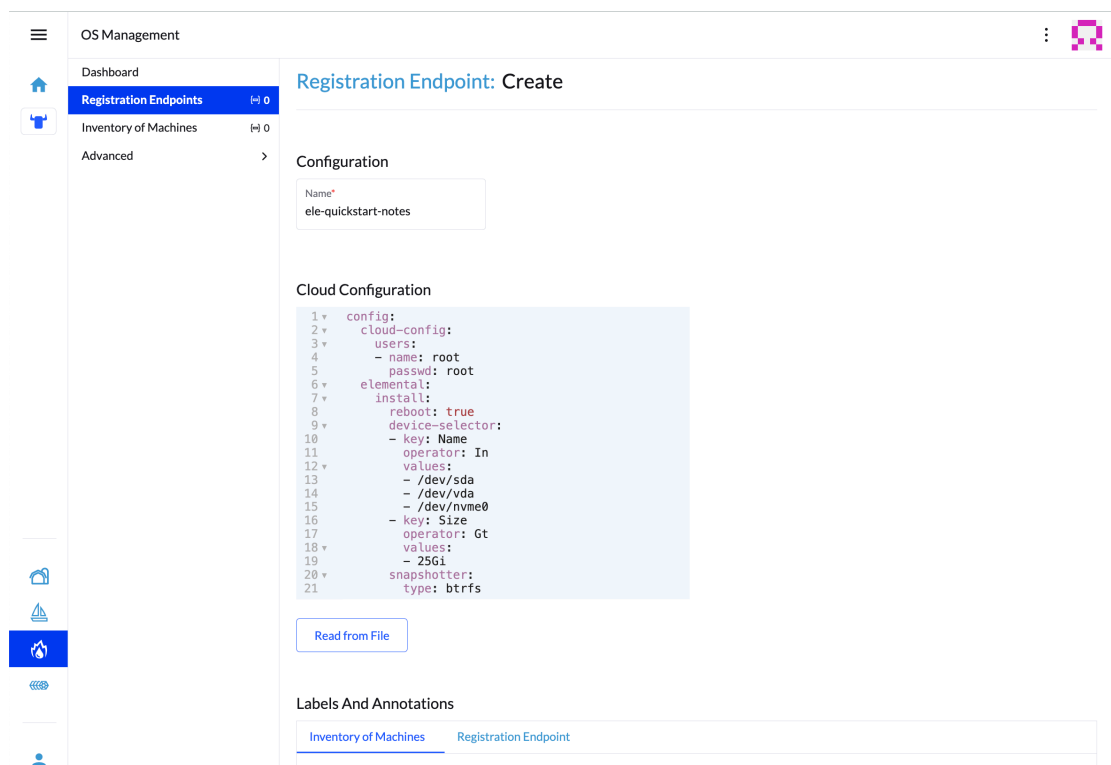
または、UIからこの操作を実行することもできます。

UI拡張機能

1. [OS Management extension (OS管理拡張機能)] から [Create Registration Endpoint (登録エンドポイントの作成)] をクリックします。



2. この設定に名前を付けます。





注記

〔Cloud Configuration (クラウドの設定)〕 フィールドは無視して構いません。このデータは、Edge Image Builderを使用した次の手順で上書きされるためです。

- 次に、下にスクロールして、マシンの登録時に作成されるリソースに付ける各ラベルに対して〔Add Label (ラベルの追加)〕 をクリックします。これはマシンを区別するのに役立ちます。

The screenshot shows the 'OS Management' interface. On the left is a sidebar with navigation links: Dashboard, Registration Endpoints (selected), Inventory of Machines, and Advanced. The main content area is titled 'Labels And Annotations' and has two tabs: 'Inventory of Machines' (selected) and 'Registration Endpoint'. A blue box contains text: 'Labels and annotations to be added to the Inventory of Machines resource when a new machine is registered. These can be used to select the correct Inventory of Machines when creating clusters and also can be used as templates using SMBIOS data. For reference on SMBIOS data check the official [documentation](#).' Below this is a 'Labels' section with a table. The table has two columns: 'Key' and 'Value'. There are two rows: one with 'manufacturer' as the key and '\$(System Information/Manufacturer)' as the value, and another with 'productName' as the key and '\$(System Information/Product Name)' as the value. Each row has a 'Remove' button to its right. Below the table is an 'Add Label' button. Underneath is an 'Annotations' section with an 'Add Annotation' button. At the bottom right of the main content area are three buttons: 'Cancel', 'Edit as YAML', and 'Create'.

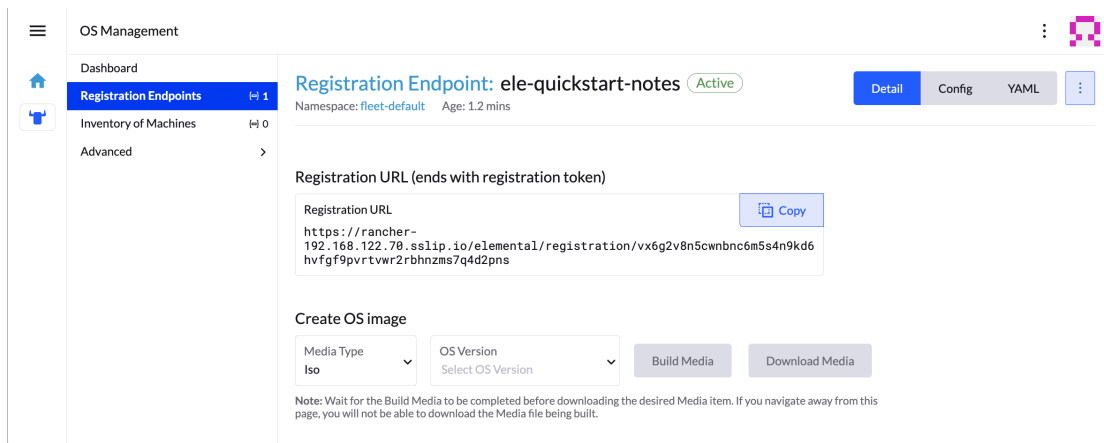
Key	Value	
manufacturer	\$(System Information/Manufacturer)	Remove
productName	\$(System Information/Product Name)	Remove

Labels

Annotations

Cancel Edit as YAML Create

- 〔Create (作成)〕 をクリックして、設定を保存します。
- 登録が作成されると、登録URLが表示され、〔Copy (コピー)〕 をクリックしてアドレスをコピーできます。



ヒント

クリックしてその画面から移動してしまった場合は、左側のメニューの「Registration Endpoints (登録エンドポイント)」をクリックし、先ほど作成したエンドポイント名をクリックできます。

このURLは次の手順で使用します。

2.7 イメージの構築

Elementalの現在のバージョンには独自のインストールメディアを構築する方法が用意されていますが、SUSE Edge 3.4では代わりにKiwiとEdge Image Builderでインストールメディアを構築します。したがって、生成されるシステムは、[SUSE Linux Micro \(https://www.suse.com/products/micro/\)](https://www.suse.com/products/micro/) をベースオペレーティングシステムとして構築されます。



ヒント

Kiwiの詳細については、まずKiwi Image Builderプロセス([第28章「Kiwiを使用したSUSE Linux Microの更新イメージの構築」](#))に従って、新しいイメージを構築してください。また、Edge Image Builderについては、Edge Image Builder導入ガイド([第3章「Edge Image Builderを使用したスタンドアロンクラスタ」](#))のほかに、コンポーネントのドキュメント([第11章「Edge Image Builder」](#))も参照してください。

PodmanをインストールしたLinuxシステムで、ディレクトリを作成し、Kiwiによって構築されるゴールデンイメージを配置します。

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

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

```
cat << EOF > $ELEM/eib_quickstart/eib-config.yaml
apiVersion: 1.3
image:
  imageType: iso
  arch: x86_64
  baseImage: SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso
  outputImageName: elemental-image.iso
operatingSystem:
  time:
    timezone: Europe/London
  ntp:
    forceWait: true
    pools:
      - 2.suse.pool.ntp.org
    servers:
      - 10.0.0.1
      - 10.0.0.2
  isoConfiguration:
    installDevice: /dev/vda
  users:
    - username: root
      encryptedPassword: \ $6\ $jHugJNNd3HElGsUZ\
$eodjVe4te5ps44SVcWshdfWizrP.xAyd71CVEXazBJ/.v799/WRCBXxfYmunlB02yp1hm/zb4r8EmnrnCf.P/
  packages:
    sccRegistrationCode: XXX
EOF
```



注記

- `time` セクションはオプションですが、証明書とクロックスキューに関する潜在的な問題を回避するように設定することを強くお勧めします。この例で指定されている値は説明のみを目的としています。ご自身の特定の要件に合わせて調整してください。
- エンコードされていないパスワードは `eib` です。
- `sccRegistrationCode` は、公式なソースから必要なRPMをダウンロードしてインストールするために必要です(または、代わりに `elemental-register` および `elemental-system-agent` のRPMを手動でサイドロードすることもできます)。

- この`cat`コマンドでは、各`$`をバックスラッシュ(`\`)でエスケープしています。このため、バッシュではテンプレート化されていません。手動でコピーする場合は、バックスラッシュを削除してください。
- インストールデバイスは、インストール中に消去されます。

```
podman run --privileged --rm -it -v $ELEM/eib_quickstart/:/eib \
registry.suse.com/edge/3.4/edge-image-builder:1.3.0 \
build --definition-file eib-config.yaml
```

物理デバイスをブートする場合は、イメージをUSBフラッシュドライブに書き込む必要があります。これは、次のコマンドで実行できます。

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

2.8 ダウンストリームノードのブート

インストールメディアを作成したので、それを使用してダウンストリームノードをブートできます。

Elementalで制御するシステムごとに、インストールメディアを追加してデバイスをブートします。インストールが完了すると、デバイスは再起動して自身を登録します。

UI拡張機能を使用している場合は、[Inventory of Machines (マシンのインベントリ)] にノードが表示されます。



注記

ログインプロンプトが表示されるまでインストールメディアを取り外さないでください。初回ブート時には、USBスティック上のファイルにアクセスしたままになります。

2.9 ダウンストリームクラスタの作成

Elementalを使用して新しいクラスタをプロビジョニングする際に作成する必要があるオブジェクトが2つあります。

Linux

最初のオブジェクトはMachineInventorySelectorTemplateです。このオブジェクトにより、クラスタとインベントリ内のマシン間のマッピングを指定できます。

1. インベントリ内のマシンをラベルに一致させるセレクトラを作成します。

```
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. リソースをクラスタに適用します。

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

3. マシンの名前を取得し、一致するラベルを追加します。

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

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

4. シンプルなシングルノードK3sクラスタリソースを作成し、クラスタに適用します。

```
cat << EOF > $ELEM/cluster.yaml
apiVersion: provisioning.cattle.io/v1
kind: Cluster
metadata:
  name: location-123
  namespace: fleet-default
spec:
  kubernetesVersion: v1.33.3+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拡張機能

UI拡張機能では、ショートカットをいくつか使用できます。複数の場所を管理する場合は、手動による作業が多くなりすぎる可能性があります。

1. 以前と同様に、左側の三本線メニューを開いて、[OS Management (OS管理)] を選択します。これで、Elementalシステムを管理するためのメイン画面に戻ります。
2. 左側のサイドバーで、[Inventory of Machines (マシンのインベントリ)] をクリックします。登録済みのマシンのインベントリが開きます。
3. これらのマシンからクラスタを作成するには、必要なシステムを選択し、[Actions (アクション)] ドロップダウンリストから [Create Elemental Cluster (Elementalクラスタの作成)] をクリックします。[Cluster Creation (クラスタの作成)] ダイアログが開き、それと同時に、使用するMachineSelectorTemplateがバックグラウンドで作成されます。
4. この画面では、構築するクラスタを設定します。このクイックスタートでは、K3s v1.30.5+k3s1が選択され、残りのオプションはそのままにしておきます。



ヒント

他のオプションを表示するには、下にスクロールする必要がある場合があります。

これらのオブジェクトを作成したら、先ほどインストールした新しいノードを使用して新しいKubernetesクラスタがスピンアップするはずです。

2.10 ノードリセット(オプション)

SUSE Rancher Elementalは、「ノードリセット」を実行する機能をサポートしています。ノードリセットは、Rancherからクラスタ全体が削除されたとき、クラスタからシングルノードが削除されたとき、またはマシンインベントリからノードが手動で削除されたときに任意で

トリガできます。これは、孤立したリソースをリセットしてクリーンアップし、クリーンアップされたノードを自動的にマシンインベントリに戻して再利用可能にする場合に役立ちます。ノードリセットはデフォルトでは有効になっていないため、削除されたシステムはクリーンアップされず(つまり、データは削除されず、Kubernetesクラスタリソースはダウンストリームクラスタで動作し続けます)、データを消去してマシンをElemental経由でRancherに再登録するには手動操作が必要となります。

この機能をデフォルトで有効にするには、MachineRegistrationに `config.elemental.reset.enabled: true`を追加して明示的に有効にする必要があります。例:

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

その後、このMachineRegistrationに登録されているすべてのシステムが自動的に `elemental.cattle.io/resettable: 'true'` のアノテーションを受け取って設定に反映します。既存のMachineInventoryにこのアノテーションがない場合や、すでにノードをデプロイ済みである場合などに、個々のノードで手動でこの操作を実行する場合は、MachineInventoryを変更し、resettable設定を追加します。例:

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

SUSE Edge 3.1では、Elemental Operatorによってオペレーティングシステム上にマーカが配置され、これによってクリーンアッププロセスが自動的にトリガされます。クリーンアッププロセスは、すべてのKubernetesサービスを停止して永続データをすべて削除し、すべてのKubernetesサービスをアンインストールして、残っているKubernetes/Rancherディレクトリをクリーンアップし、元のElemental MachineRegistration設定を使用して強制的にRancherに再登録します。これは自動的に行われるため、手動での操作は必要ありません。呼び出されるスクリプトは `/opt/edge/elemental_node_cleanup.sh` にあり、マーカが配置されるとすぐに `systemd.path` を介してトリガされるため、直ちに実行されます。



警告

`resettable`機能を使用する場合、Rancherからノード/クラスタを削除する際の望ましい動作は、データを消去して再登録を強制することであると想定されています。この状況ではデータが確実に失われるため、この機能は、自動リセットを実行することがわかっている場合にのみ使用してください。

2.11 次の手順

このガイドの使用後に調べるべき推奨リソースを次に示します。

- 第8章 「Fleet」のエンドツーエンドの自動化
- 第12章 「Edgeネットワーキング」の追加のネットワーク設定オプション

3 Edge Image Builderを使用したスタンドアロンクラスタ

Edge Image Builder (EIB)は、完全なエアギャップシナリオでもマシンをブートストラップできるCustomized, Ready-to-Boot (CRB)ディスクイメージの生成プロセスを効率化するツールです。EIBを使用すると、SUSE Edgeの3つのデプロイメントフットプリントすべてで使用するデプロイメントイメージを作成できます。これは、EIBが十分に柔軟であり、最小限のカスタマイズ(例: ユーザの追加やタイムゾーンの設定)から、あらゆる設定を網羅したイメージ(例: 複雑なネットワーク設定を行い、マルチノードKubernetesクラスタをデプロイして、顧客ワークロードをデプロイし、Rancher/ElementalとSUSE Multi-Linux Managerを介して集中管理プラットフォームに登録するイメージ)までを提供できるためです。EIBはコンテナイメージ内で動作するため、プラットフォーム間できわめて容易に移植可能です。さらに、必要な依存関係をすべて備えた自己完結型であるため、EIBツールの操作に使用するシステムにインストールされているパッケージに及ぼす影響が最小限に抑えられます。



注記

マルチノードシナリオの場合、同じ構築されたイメージを使用してプロビジョニングされたホストがKubernetesクラスタに自動的に参加できるように、EIBはMetalLBとEndpoint Copier Operatorを自動的にデプロイします。

詳細については、Edge Image Builderの紹介(第11章「Edge Image Builder」)を参照してください。



警告

Edge Image Builder 1.3.0は、SUSE Linux Micro 6.1イメージのカスタマイズをサポートしています。SUSE Linux Enterprise Micro 5.5や6.0などの古いバージョンはサポートされていません。

3.1 前提条件

- SLES 15 SP6を実行しているAMD64/Intel 64ビルドホストマシン(物理または仮想)
- Podmanコンテナエンジン
- Kiwi Builder手順を使用して作成されたSUSE Linux Micro 6.1 SelfInstall ISOイメージ(第28章「Kiwiを使用したSUSE Linux Microの更新イメージの構築」)



注記

非運用目的の場合は、openSUSE Leap 15.6、またはopenSUSE Tumbleweedをビルドホストマシンとして使用できます。互換性のあるコンテナランタイムが利用できれば、その他のオペレーティングシステムでも機能します。

3.1.1 EIBイメージの取得

EIBのコンテナイメージは一般に公開されており、イメージ構築ホストで次のコマンドを実行することでSUSE Edgeレジストリからダウンロードできます。

```
podman pull registry.suse.com/edge/3.4/edge-image-builder:1.3.0
```

3.2 イメージ設定ディレクトリの作成

EIBはコンテナ内で動作するため、ホストから設定ディレクトリをマウントして、必要な設定を指定できるようにする必要があります。ビルドプロセス中に、EIBは必要な入力ファイルやサポートアーティファクトすべてにアクセスできます。このディレクトリは、特定の構造に従う必要があります。このディレクトリがホームディレクトリに存在し、「eib」という名前であると仮定して、ディレクトリを作成してみましょう。

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

前の手順でSUSE Linux Micro 6.1入力イメージをホストする「base-images」ディレクトリを作成しました。このイメージが設定ディレクトリにコピーされていることを確認しましょう。

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



注記

EIBの実行中に元のゴールデンイメージは変更**されません**。EIBの設定ディレクトリのルートに、目的の設定でカスタマイズされた新しいバージョンが作成されます。

この時点では、設定ディレクトリは次のようになっているはずです。

```
└─ base-images/
```

3.3 イメージ定義ファイルの作成

定義ファイルには、Edge Image Builderがサポートする設定可能なオプションの大部分が記述されています。オプションの完全な例については、[こちら \(https://github.com/suse-edge/edge-image-builder/blob/release-1.3/pkg/image/testdata/full-valid-example.yaml\)](https://github.com/suse-edge/edge-image-builder/blob/release-1.3/pkg/image/testdata/full-valid-example.yaml) を参照してください。以下で説明する例よりも広範な例については、[アップストリームのイメージ構築ガイド \(https://github.com/suse-edge/edge-image-builder/blob/release-1.3/docs/building-images.md\)](https://github.com/suse-edge/edge-image-builder/blob/release-1.3/docs/building-images.md) を参照することをお勧めします。まずは、OSイメージの非常に基本的な定義ファイルから始めましょう。

```
cat << EOF > $CONFIG_DIR/iso-definition.yaml
apiVersion: 1.3
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
EOF
```

この定義では、AMD64/Intel 64ベースのシステム用の出力イメージを生成するように指定しています。さらに変更を加えるためのベースとして使用するイメージは、`slemicro.iso` という名前のisoイメージであり、`$CONFIG_DIR/base-images/slemicro.iso`にあることが想定されています。また、EIBがイメージの変更を完了すると、出力イメージは`eib-image.iso` という名前になり、デフォルトでは`$CONFIG_DIR`に存在することも記述されています。

これで、ディレクトリ構造は次のようになります。

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

以降のセクションでは、一般的な操作の例をいくつか紹介していきます。

3.3.1 OSユーザの設定

EIBを使用すると、パスワードやSSHキーなどのログイン情報を事前にユーザに設定できます(固定されたルートパスワードの設定も含む)。この例の一部として、ルートパスワードを修正します。最初の手順は、[OpenSSL](#)を使用して一方向暗号化パスワードを作成することです。

```
openssl passwd -6 SecurePassword
```

これは次のような出力になります。

```
$6$G392FCbxVgnLaFw1$Ujt00mdpJ3tDHxEg1snBU3GjujQf6f8kvopu7jiCBihRbRvMmKUqwcMxAKggaSSKeUU0EtCP3ZUoZQY7zTXr
```

次に、定義ファイルに`operatingSystem`というセクションを追加し、その中に`users`配列を含めます。作成したファイルは次のようになります。

```
apiVersion: 1.3
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword:
        $6$G392FCbxVgnLaFw1$Ujt00mdpJ3tDHxEg1snBU3GjujQf6f8kvopu7jiCBihRbRvMmKUqwcMxAKggaSSKeUU0EtCP3ZUoZQY7zTXr
```



注記

ユーザの追加、ホームディレクトリの作成、ユーザIDの設定、ssh-key認証の追加、グループ情報の変更も行うことができます。他の例については、[アップストリームのイメージ構築ガイド \(https://github.com/suse-edge/edge-image-builder/blob/release-1.3/docs/building-images.md\)](https://github.com/suse-edge/edge-image-builder/blob/release-1.3/docs/building-images.md) を参照してください。

3.3.2 OSの時刻の設定

`time`セクションはオプションですが、証明書とクロックスキューに関する潜在的な問題を避けるために、設定することを強くお勧めします。EIBは、`chronyd`と`/etc/localtime`をこのパラメータに応じて設定します。

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

- `timezone`では、タイムゾーンを「地域/都市」(例: 「Europe/London」)の形式で指定します。完全なリストを確認するには、Linuxシステム上で`timedatectl list-timezones`を実行します。
- `ntp` - NTPの設定(`chronyd`を使用)に関連する属性を定義します。
- `forceWait` - 他のサービスを開始する前に`chronyd`が時刻ソースの同期を試行するよう要求します。タイムアウトは180秒です。
- `pools` - `chronyd`がデータソースとして使用するプールのリストを指定します(`iburst`を使用すると、最初の同期にかかる時間を短縮できます)。
- `servers` - `chronyd`がデータソースとして使用するサーバのリストを指定します(`iburst`を使用すると、最初の同期にかかる時間を短縮できます)。



注記

この例で指定されている値は説明のみを目的としています。ご自身の特定の要件に合わせて調整してください。

3.3.3 証明書の追加

`certificates`ディレクトリに保存された拡張子「.pem」または「.crt」の証明書ファイルは、ノードのシステム全体の証明書ストアにインストールされます。

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

詳細については、「[Securing Communication with TLS Certificate \(TLS証明書による通信のセキュリティ保護\)](https://documentation.suse.com/smart/security/html/tls-certificates/index.html#tls-adding-new-certificates)」ガイド (<https://documentation.suse.com/smart/security/html/tls-certificates/index.html#tls-adding-new-certificates>) を参照してください。

3.3.4 RPMパッケージの設定

EIBの主な特徴の1つは、イメージにソフトウェアパッケージを追加するメカニズムを備えていることです。このため、インストールが完了した時点で、システムはインストールされたパッケージをすぐに利用できます。EIBでは、ユーザは以下を行うことができます。

- イメージ定義のリスト内の名前でパッケージを指定する
- これらのパッケージを検索するネットワークリポジトリを指定する
- 一覧にされたパッケージをSUSEの公式リポジトリで検索するためのSUSE Customer Center (SCC)資格情報を指定する
- `$CONFIG_DIR/rpms`ディレクトリ経由で、ネットワークリポジトリに存在しないカスタムRPMをサイドロードする
- 同じディレクトリ(`$CONFIG_DIR/rpms/gpg-keys`)経由で、サードパーティ製パッケージの検証を有効にするためにGPGキーを指定する

これにより、EIBは、イメージの構築時にパッケージ解決プロセスを実行し、ゴールデンイメージを入力として受け取り、提供されているパッケージ(リストで指定されているか、ローカルで提供されているパッケージ)をすべてプルしてインストールしようと試みます。EIBは、依存関係を含むすべてのパッケージを出力イメージ内に存在するリポジトリにダウンロードし、そのパッケージを初回ブートプロセス中にインストールするようにシステムに指示します。イメージの構築中にこのプロセスを実行することで、初回ブート時にパッケージが目的のプラットフォーム(エッジのノードなど)に正常にインストールされることが保証されます。これは、操作時に追加パッケージをネットワーク経由でプルするのではなく、イメージに事前書き込んでおきたい環境でも便利です。たとえば、エアギャップ環境や、ネットワークが制限された環境です。

これを示すシンプルな例として、サードパーティベンダがサポートするNVIDIAリポジトリにある `nvidia-container-toolkit` RPMパッケージをインストールします。

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

生成される定義ファイルは次のようになります。

```
apiVersion: 1.3
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword:
$6$G392FCbxVgnLaFw1$Ujt00mdpJ3tDHxEg1snBU3GjujQf6f8kvopu7jiCBIhRbRvMmKUqwcXAKggaSSKeUU0EtCP3ZUoZQY7zTX
```



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

上記はシンプルな例ですが、完全を期するために、イメージ生成を実行する前にNVIDIAパッケージ署名キーをダウンロードしてください。

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



警告

この方法でRPMを追加することは、サポートされているサードパーティコンポーネント、またはユーザが提供(および保守)するパッケージを追加することを目的としています。このメカニズムは、通常ではSUSE Linux Microでサポートされないパッケージを追加する目的では使用しないでください。このメカニズムを使用して、openSUSEのリポジトリから新しいリリースやサービスパックなどのコンポーネントを追加した場合(この操作はサポートされません)、最終的にサポート対象外の設定になるおそれがあります。特に、依存関係の解決によってオペレーティングシステムのコア部分が置き換えられる場合は、作成されたシステムが期待どおりに機能しているように見えても注意が必要です。確信が持てない場合は、SUSEの担当者に連絡してサポートを依頼し、目的の設定がサポート可能かどうかを判断してください。



注記

追加の例を含む総合的なガイドについては、[アップストリームのパッケージインストーラガイド \(https://github.com/suse-edge/edge-image-builder/blob/release-1.3/docs/installing-packages.md\)](https://github.com/suse-edge/edge-image-builder/blob/release-1.3/docs/installing-packages.md) を参照してください。

3.3.5 Kubernetesクラスタとユーザワークロードの設定

EIBのもう1つの特徴は、EIBを使用すると、「インプレースでブートストラップ」する(つまり調整のためにどのような形態の集中管理インフラストラクチャも必要としない)シングルノードとマルチノード両方の高可用性Kubernetesクラスタのデプロイメントを自動化できることです。このアプローチは主にエアギャップデプロイメント、すなわちネットワークが制限された環境のためのものですが、ネットワークに制限なく完全にアクセスできる場合であっても、スタンドアロンクラスタを迅速にブートストラップする方法として役立ちます。

この方法を使用すると、カスタマイズされたオペレーティングシステムをデプロイできるだけでなく、Kubernetesの設定を指定したり、Helmチャートを介して追加の階層化コンポーネントを指定したり、指定したKubernetesマニフェストを介してユーザワークロードを指定したりすることもできます。ただしこの方法を使用する場合、その背景にある設計理念として、ユーザがエアギャップ化を望んでいるとデフォルトで想定します。したがって、イメージ定義で指定されているすべての項目を、ユーザが指定したワークロードを含めてイメージにプルします。その際に、EIBは、定義で要求されている検出済みイメージをすべてローカルにコピーし、作成されたデプロイ済みシステムの組み込みのイメージレジストリで提供します。

次の例では、既存のイメージ定義を使用してKubernetesの設定を指定します(この例では、複数のシステムとその役割が一覧にされていないため、デフォルトでシングルノードを想定します)。この設定により、シングルノードのRKE2 KubernetesクラスタをプロビジョニングするようにEIBに指示します。ユーザが指定したワークロード(マニフェストを使用)と階層化コンポーネント(Helmを使用)の両方のデプロイメントの自動化を説明するために、SUSE EdgeのHelmチャートを使用してKubeVirtをインストールし、Kubernetesマニフェストを使用してNGINXをインストールします。既存のイメージ定義に追加する必要がある設定は次のとおりです。

```
kubernetes:
  version: v1.33.3+rke2r1
  manifests:
    urls:
      - https://k8s.io/examples/application/nginx-app.yaml
  helm:
    charts:
      - name: kubevirt
        version: 304.0.1+up0.6.0
        repositoryName: suse-edge
    repositories:
      - name: suse-edge
        url: oci://registry.suse.com/edge/charts
```

作成された完全な定義ファイルは次のようになります。

```
apiVersion: 1.3
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword:
        $6$G392FCbxVgnLaFw1$Ujt00mdpJ3tDHxEg1snBU3GjujQf6f8kvopu7jiCBiHrBvMmKUqwcMxAKGgaSSKeUU0EtCP3ZUoZQY7zTX
  packages:
    packageList:
```

```

- nvidia-container-toolkit
additionalRepos:
- url: https://nvidia.github.io/libnvidia-container/stable/rpm/x86_64
kubernetes:
version: v1.33.3+k3s1
manifests:
urls:
- https://k8s.io/examples/application/nginx-app.yaml
helm:
charts:
- name: kubevirt
version: 304.0.1+up0.6.0
repositoryName: suse-edge
repositories:
- name: suse-edge
url: oci://registry.suse.com/edge/charts

```



注記

マルチノードデプロイメント、カスタムネットワーキング、Helmチャートのオプション/値など、オプションの他の例については、[アップストリームドキュメント](https://github.com/suse-edge/edge-image-builder/blob/release-1.3/docs/building-images.md) (<https://github.com/suse-edge/edge-image-builder/blob/release-1.3/docs/building-images.md>) を参照してください。

3.3.6 ネットワークの設定

このクイックスタートの最後の例では、EIBで生成したイメージを使ってシステムをプロビジョニングした場合に作成されるネットワークを設定しましょう。ネットワーク設定を指定しない限り、ブート時に検出されたすべてのインタフェースでDHCPが使用されるのがデフォルトのモデルであることを理解することが重要です。ただし、これが常に望ましい設定であるとは限りません。これは特に、DHCPが利用できず静的な設定を指定する必要がある場合や、より複雑なネットワーキング構造(ボンド、LACP、VLANなど)を設定する必要がある場合、特定のパラメータ(ホスト名、DNSサーバ、ルートなど)を上書きする必要がある場合に該当します。

EIBでは、ノードごとの設定を指定することも(対象のシステムをMACアドレスで一意に識別します)、上書きによって各マシンに同一の設定を指定することもできます(システムのMACアドレスが不明な場合に便利です)。またEIBでは、Network Manager Configurator (nmc) というツールも追加で使用されます。Network Manager ConfiguratorはSUSE Edgeチームによって構築されたツールであり、nmstate.io (<https://nmstate.io/>) の宣言型ネットワークスキーマに基づいてカスタムネットワーキング設定を適用できるようにします。また、ブートしているノードをブート時に識別し、必要なネットワーク設定をサービス開始前に適用します。

ここでは、1つのインタフェースを持つシステムに静的ネットワーク設定を適用します。そのために、ネットワークの望ましい状態を、必要なnetworkディレクトリ内にある(目的のホスト名に基づく)ノード固有のファイルに記述します。

```
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
```



警告

上記の例は、テストを仮想マシン上で実行すると仮定して、デフォルトの192.168.122.0/24サブネット用に設定されています。ご使用の環境に合わせて、MACアドレスも忘れずに変更してください。同じイメージを使用して複数のノードをプロビジョニングできるため、EIBで(nmcを介して)設定されたネットワーキングは、各ノードをMACアドレスで一意に識別できるかどうか依存しており、その後、ブート中にnmcは正しいネットワーキング設定を各マシンに適用します。つまり、イン

ストール先のシステムのMACアドレスを把握する必要があります。また、デフォルトの動作ではDHCPに依存しますが、`configure-network.sh`フックを利用して、すべてのノードに共通の設定を適用することもできます。詳細については、ネットワーキングのガイド(第12章「Edgeネットワーキング」)を参照してください。

作成されるファイル構造は、次のようになります。

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

作成したネットワーク設定が解析され、必要なNetworkManager接続ファイルが自動的に生成されて、EIBが作成する新しいインストールイメージに挿入されます。これらのファイルはホストのプロビジョニング中に適用され、完全なネットワーク設定が生成されます。



注記

上記の設定のより包括的な説明と、この機能の例については、Edgeネットワーキングのコンポーネント(第12章「Edgeネットワーキング」)を参照してください。

3.4 イメージの構築

EIBで使用するゴールデンイメージとイメージ定義ができたので、イメージを構築してみましょう。これには、`podman`を使用し、定義ファイルを指定して「`build`」コマンドでEIBコンテナを呼び出すだけです。

```
podman run --rm -it --privileged -v $CONFIG_DIR:/eib \
registry.suse.com/edge/3.4/edge-image-builder:1.3.0 \
build --definition-file iso-definition.yaml
```

コマンドの出力は次のようになります。

```
Setting up Podman API listener...
Downloading file: dl-manifest-1.yaml 100% (498/498 B, 9.5 MB/s)
Pulling selected Helm charts... 100% (1/1, 43 it/min)
Generating image customization components...
Identifier ..... [SUCCESS]
Custom Files ..... [SKIPPED]
Time ..... [SKIPPED]
Network ..... [SUCCESS]
Groups ..... [SKIPPED]
Users ..... [SUCCESS]
```

```

Proxy ..... [SKIPPED]
Resolving package dependencies...
Rpm ..... [SUCCESS]
Os Files ..... [SKIPPED]
Systemd ..... [SKIPPED]
Fips ..... [SKIPPED]
Elemental ..... [SKIPPED]
Suma ..... [SKIPPED]
Populating Embedded Artifact Registry... 100% (3/3, 10 it/min)
Embedded Artifact Registry ... [SUCCESS]
Keymap ..... [SUCCESS]
Configuring Kubernetes component...
The Kubernetes CNI is not explicitly set, defaulting to 'cilium'.
Downloading file: rke2_installer.sh
Downloading file: rke2-images-core.linux-amd64.tar.zst 100% (657/657 MB, 48 MB/s)
Downloading file: rke2-images-cilium.linux-amd64.tar.zst 100% (368/368 MB, 48 MB/s)
Downloading file: rke2.linux-amd64.tar.gz 100% (35/35 MB, 50 MB/s)
Downloading file: sha256sum-amd64.txt 100% (4.3/4.3 kB, 6.2 MB/s)
Kubernetes ..... [SUCCESS]
Certificates ..... [SKIPPED]
Cleanup ..... [SKIPPED]
Building ISO image...
Kernel Params ..... [SKIPPED]
Build complete, the image can be found at: eib-image.iso

```

構築されたISOイメージは`$CONFIG_DIR/eib-image.iso`に保存されます。

```

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

```

ビルドごとに、`$CONFIG_DIR/_build/`内にタイムスタンプ付きのフォルダが作成されます。このフォルダには、ビルドのログ、ビルド中に使用されたアーティファクト、およびCRBイメージに追加されたすべてのスクリプトとアーティファクトを含む`combustion`ディレクトリと`artefacts`ディレクトリが含まれます。

このディレクトリの内容は次のようになります。

```

├─ build-<timestamp>/
│   └─ combustion/
│       └─ 05-configure-network.sh
│       └─ 10-rpm-install.sh
│       └─ 12-keymap-setup.sh

```

```

├─ 13b-add-users.sh
├─ 20-k8s-install.sh
├─ 26-embedded-registry.sh
├─ 48-message.sh
├─ network/
│   └─ host1.local/
│       └─ eth0.nmconnection
│       └─ host_config.yaml
├─ nmc
├─ script
├─ artefacts/
│   └─ registry/
│       ├── hauler
│       ├── nginx:<version>-registry.tar.zst
│       ├── rancher_kubectrl:<version>-registry.tar.zst
│       └─ registry.suse.com_suse_sles_15.6_virt-operator:<version>-registry.tar.zst
├─ rpms/
│   └─ rpm-repo
│       ├── addrepo0
│       │   ├── nvidia-container-toolkit-<version>.rpm
│       │   ├── nvidia-container-toolkit-base-<version>.rpm
│       │   ├── libnvidia-container1-<version>.rpm
│       │   └─ libnvidia-container-tools-<version>.rpm
│       ├── repodata
│       ├── ...
│       └─ zypper-success
├─ kubernetes/
│   ├── rke2_installer.sh
│   ├── registries.yaml
│   ├── server.yaml
│   ├── images/
│   │   ├── rke2-images-cilium.linux-amd64.tar.zst
│   │   └─ rke2-images-core.linux-amd64.tar.zst
│   ├── install/
│   │   ├── rke2.linux-amd64.tar.gz
│   │   └─ sha256sum-amd64.txt
│   └─ manifests/
│       ├── dl-manifest-1.yaml
│       └─ kubevirt.yaml
├─ createrepo.log
├─ eib-build.log
├─ embedded-registry.log
├─ helm
│   └─ kubevirt
│       └─ kubevirt-0.4.0.tgz
├─ helm-pull.log
├─ helm-template.log
├─ iso-build.log
├─ iso-build.sh
├─ iso-extract
├─ ...

```

```
| | | | | iso-extract.log
| | | | | iso-extract.sh
| | | | | modify-raw-image.sh
| | | | | network-config.log
| | | | | podman-image-build.log
| | | | | podman-system-service.log
| | | | | prepare-resolver-base-tarball-image.log
| | | | | prepare-resolver-base-tarball-image.sh
| | | | | raw-build.log
| | | | | raw-extract
| | | | |   |
| | | | |   | ...
| | | | |   |
| | | | |   | resolver-image-build
| | | | |   | ...
| | | | |
| | | | | cache
| | | | |   |
| | | | |   | ...
```

ビルドが失敗した場合、情報が含まれる最初のログは`eib-build.log`です。そこから、失敗したコンポーネントに移動してデバッグを行います。

この時点で、以下を行う、すぐに使用できるイメージができています。

1. SUSE Linux Micro 6.1をデプロイする
2. ルートパスワードを設定する
3. `nvidia-container-toolkit`パッケージをインストールする
4. コンテンツをローカルに提供する組み込みのコンテナレジストリを設定する
5. シングルノードRKE2をインストールする
6. 静的ネットワーキングを設定する
7. KubeVirtをインストールする
8. ユーザが指定したマニフェストをデプロイする

3.5 イメージ構築プロセスのデバッグ

イメージ構築プロセスが失敗する場合は、アップストリームのデバッグガイド (<https://github.com/suse-edge/edge-image-builder/blob/release-1.3/docs/debugging.md>) を参照してください。

3.6 新しく構築されたイメージのテスト

新しく構築されたCRBイメージをテストする方法については、アップストリームのイメージテストガイド (<https://github.com/suse-edge/edge-image-builder/blob/release-1.3/docs/testing-guide.md>)  を参照してください。

4 SUSE Multi-Linux Manager

SUSE Multi-Linux ManagerはSUSE Edgeに含まれており、エッジデプロイメントのすべてのノードでSUSE Linux Microを基盤となるオペレーティングシステムとして常に最新の状態に保つための自動化と制御を提供します。また、エッジノード上のKubernetesとKubernetes上にデプロイされたアプリケーションの管理にも使用できます。

このクイックスタートガイドは、エッジノードにオペレーティングシステムの更新を提供することを目的として、SUSE Multi-Linux Managerをできるだけ早く使いこなすためのものです。このクイックスタートガイドでは、ストレージのサイズ設定、ステージング目的の追加ソフトウェアチャンネルの作成と管理、大規模なデプロイメントのためのユーザ、システムグループ、組織の管理などのトピックについては説明しません。運用環境で使用する場合は、包括的な[SUSE Multi-Linux Manager のドキュメント \(https://documentation.suse.com/suma/5.0/en/suse-manager/index.html\)](https://documentation.suse.com/suma/5.0/en/suse-manager/index.html) を参照することを強くお勧めします。

SUSE Multi-Linux Managerを効果的に使用するためにSUSE Edgeを準備するには、次の手順が必要です。

- SUSE Multi-Linux Manager Serverのデプロイと設定
- SUSE Linux Microパッケージリポジトリの同期
- システムグループの作成
- アクティベーションキーの作成
- Edge Image Builderを使用したSUSE Multi-Linux Manager登録用インストールメディアの準備

4.1 SUSE Multi-Linux Manager Serverのデプロイ

SUSE Multi-Linux Manager 5.0.5のインスタンスをすでに実行している場合は、この手順をスキップできます。

SUSE Multi-Linux Manager Serverは、専用の物理サーバ、自身のハードウェア上の仮想マシン、またはクラウドで実行できます。SUSE Multi-Linux Serverの事前に設定された仮想マシンイメージはサポートされているパブリッククラウド用に提供されています。

このクイックスタートでは、AMD64/Intel 64用の「qcow2」イメージ `SUSE-Manager-Server.x86_64-5.0.4-Qcow-5.0-2025-04.qcow2` を使用しています。このイメージは、<https://www.suse.com/download/suse-manager/> またはSUSE Customer Centerで入手できます。このイメージは、KVMなどのハイパーバイザー上の仮想マシンとして機能します。イメージの最新バージョンを常に確認して、新しいインストールに使用してください。

SUSE Multi-Linux Manager Serverを他のサポートされているハードウェアアーキテクチャにもインストールできます。その場合は、ハードウェアアーキテクチャに合ったイメージを選択してください。

イメージをダウンロードしたら、次の最小ハードウェア仕様を満たす仮想マシンを作成します。

- 16GB RAM
- 4つの物理コアまたは仮想コア
- 最低100GBの追加のブロックデバイス

qcow2イメージでは、オペレーティングシステムをインストールする必要はありません。イメージをルートパーティションとして直接アタッチできます。

後でエッジノードが完全修飾ドメイン名(「FQDN」)を含むホスト名でSUSE Multi-Linux Manager Serverにアクセスできるように、ネットワークを設定する必要があります。

SUSE Multi-Linux Managerを初めてブートする際には、いくつかの初期設定を実行する必要があります。

- キーボードレイアウトの選択
- ライセンス契約の同意
- タイムゾーンの選択
- オペレーティングシステムのルートパスワードの入力

次の各手順は「ルート」ユーザとして実行する必要があります。

次の手順では、SUSE Customer Centerで入手可能なSUSE Multi-Linux Manager Extensionの登録コードが必要です。同じコードを使用して、SUSE Linux MicroとSUSE Multi-Linux Managerの両方を登録できます。

SUSE Linux Microを登録します。

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

SUSE Multi-Linux Managerを登録します。

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

製品文字列は、ハードウェアアーキテクチャによって異なります。たとえば、64ビットArmシステムでSUSE Multi-Linux Managerを使用している場合、文字列は「SUSE-Manager-Server/5.0/aarch64」になります。

再起動します。

システムを更新します。

変更がなければ、再起動して更新を適用します。

SUSE Multi-Linux Managerは、Podmanによって管理されるコンテナを介して提供されます。`mgradm`コマンドがセットアップと設定を自動的行います。



警告

SUSE Multi-Linux Manager Serverのホスト名が、管理対象のエッジノードがネットワークで適切に解決できる完全修飾ドメイン名(「FQDN」)で設定されていることが非常に重要です。

SUSE Multi-Linux Manager Serverコンテナをインストールして設定する前に、以前に追加した追加のブロックデバイスを準備しておく必要があります。そのためには、仮想マシンがそのデバイスに付けた名前を知っている必要があります。たとえば、ブロックデバイスが `/dev/vdb` の場合、次のコマンドを使用して、SUSE Multi-Linux Managerで使用するよう設定できます。

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

SUSE Multi-Linux Managerをデプロイします。

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

CA証明書のパスワードを入力します。このパスワードはログインパスワードとは異なる必要があります。通常は後で入力する必要はありませんが、メモしておいてください。

「admin」ユーザのパスワードを入力します。これは、SUSE Multi-Linux Managerにログインするための初期ユーザです。後で完全な権利または制限された権利を持つ追加のユーザを作成できます。

4.2 SUSE Multi-Linux Managerの設定

デプロイメントが完了したら、先ほど指定したホスト名を使用して、SUSE Multi-Linux Manager Web UIにログインできます。初期ユーザは「admin」です。前の手順で指定したパスワードを使用します。

次の手順では、SUSE Customer Centerの組織の「ユーザ」タブの2番目のサブタブにある組織の資格情報が必要です。これらの資格情報を使用して、SUSE Multi-Linux Managerはサブスクリプションを持つすべての製品を同期できます。

[管理] > [セットアップウィザード] を選択します。

組織の資格情報タブで、SUSE Customer Centerにあるユーザー名とパスワードを使用して新しい資格情報を作成します。

次のタブ [SUSE製品] に移動します。SUSE Customer Centerとの最初のデータ同期が完了するまで待つ必要があります。

リストが入力されたら、フィルタを使用して「Micro 6.1」のみを表示します。エッジノードが実行されるハードウェアアーキテクチャ(x86_64またはaarch64)に対応するSUSE Linux Micro 6.1のチェックボックスをオンにします。

[製品の追加] をクリックします。これにより、SUSE Linux Microのメインパッケージリポジトリ(「チャンネル」)が追加され、SUSE Managerクライアントツールのチャンネルがサブチャンネルとして自動的に追加されます。

インターネット接続によっては、最初の同期にしばらく時間がかかります。次の手順からすぐに開始できます。

[システム] > [システムグループ] で、システムがオンボード時に自動的に参加するグループを少なくとも1つ作成します。グループはシステムを分類する重要な方法であり、設定やアクションをシステムセット全体に一度に適用できます。概念的には、Kubernetesのラベルに似ています。

[+グループの作成] をクリックします。

短い名前(「エッジノード」など)と長い説明を入力します。

[システム] > [アクティベーションキー] で、少なくとも1つのアクティベーションキーを作成します。アクティベーションキーは、システムが SUSE Multi-Linux Managerにオンボードされたときに自動的に適用される設定プロファイルと考えることができます。特定のエッジノードを異なるグループに追加したり、異なる設定を使用したりする場合は、それら用に個別のアクティベーションキーを作成し、後でEdge Image Builderでカスタマイズしたインストールメディアを作成する際に使用できます。

アクティベーションキーの典型的な高度なユースケースは、テストクラスタを最新の更新を含むソフトウェアチャンネルに割り当て、運用クラスタを、テストクラスタでテストした後にのみ最新の更新を取得するソフトウェアチャンネルに割り当てることです。

[+キーの作成] をクリックします。

短い説明を選択します(「エッジノード」など)。キーを識別する一意の名前を指定します(AMD64/Intel 64ハードウェアアーキテクチャを備えたエッジノードの場合は「edge-x86_64」など)。番号のプレフィックスが自動的にキーに追加されます。デフォルトの組織の場合、番号は常に「1」です。SUSE Multi-Linux Managerで追加の組織を作成し、これらの組織のキーを作成する場合、この番号は異なる可能性があります。

クローンソフトウェアチャンネルを作成していない場合は、ベースチャンネルの設定を「SUSE Managerの既定値」のままにしておくことができます。これにより、エッジノードに正しいSUSE更新リポジトリが自動的に割り当てられます。

「子チャンネル」として、アクティベーションキーが使用されているハードウェアアーキテクチャの「推奨を含める」スライダを選択します。これにより、「SUSE-Manager-Tools-For-SL-Micro-6.1」チャンネルが追加されます。

[グループ] タブで、以前に作成したグループを追加します。このアクティベーションキーを使用してオンボードされたすべてのノードがそのグループに自動的に追加されます。

4.3 Edge Image Builderを使用したカスタマイズされたインストールイメージの作成

Edge Image Builderを使用するには、PodmanでLinuxベースのコンテナを起動できる環境のみが必要です。

最小限のラボのセットアップでは、SUSE Multi-Linux Manager Serverが実行しているのと同じ仮想マシンを実際に使用できます。仮想マシンに十分なディスク容量があることを確認してください。これは運用環境での使用には推奨されないセットアップです。Edge Image Builderをテストしたホストオペレーティングシステムについては、3.1項「前提条件」を参照してください。

SUSE Multi-Linux Managerサーバホストにrootとしてログインします。

Edge Image Builderコンテナをプルします。

```
podman pull registry.suse.com/edge/3.4/edge-image-builder:1.3.0
```

ディレクトリ/opt/eibとサブディレクトリbase-imagesを作成します。

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

このクイックスタートでは、SUSE Linux Microイメージの「セルフインストール」フレーバーを使用しています。このイメージは、後で物理USBメモリに書き込んで、物理サーバにインストールするために使用できます。ご使用のサーバにBMC（ベースボード管理コントローラ）経由でインストールISOをリモートアタッチするオプションがある場合は、その方法も使用できます。また、このイメージはほとんどの仮想化ツールでも使用できます。

イメージを物理ノードに直接プリロードする場合、またはVMから直接起動する場合は、「生」イメージフレーバーを使用することもできます。

これらのイメージはSUSE Customer Centerまたは<https://www.suse.com/download/sle-micro/>にあります。

イメージ `SL-Micro.x86_64-6.1-Default-SelfInstall-GM.install.iso` を `base-images` ディレクトリにダウンロードまたはコピーして、「`slemicro.iso`」という名前を付けます。

Armベースの構築ホスト上でのAArch64イメージの構築は、SUSE Edge 3.4の技術プレビュー対象です。おそらく動作しますが、まだサポートされていません。お試しいただく場合は、64ビットのArmマシンでPodmanを実行し、すべての例とコードスニペットの「`x86_64`」を「`aarch64`」に置き換える必要があります。

`/opt/eib` に、`iso-definition.yaml` というファイルを作成します。これはEdge Image Builderのビルド定義です。

ここでは、SL Micro 6.1をインストールし、ルートパスワードとキーマップを設定して、CockpitグラフィカルUIを起動し、ノードをSUSE Multi-Linux Managerに登録するシンプルな例を示します。

```
apiVersion: 1.3
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      createHomeDir: true
      encryptedPassword: $6$aaBTHyqDRUMY1HAp$pmBY7.qLtoVlCGj32XR/0gei4cngc3f40X7fwBD/gw7HWyuNB0KYbBwnJ4pvrYwH2WUtJLKMbinVtBhMDHQIY0
  keymap: de
  systemd:
    enable:
      - cockpit.socket
  packages:
    noGPGCheck: true
  suma:
    host: ${fully qualified hostname of your SUSE Multi-Linux Manager Server}
    activationKey: 1-edge-x86_64
```

Edge Image Builderは、ネットワークを設定し、ノードにKubernetesを自動的にインストールして、Helmチャート経由でアプリケーションをデプロイすることもできます。より包括的な例については、[第3章「Edge Image Builderを使用したスタンドアロンクラスタ」](#)を参照してください。

`baseImage` には、使用する `base-images` ディレクトリ内のISOの実際の名前を指定します。

この例では、ルートパスワードは「`root`」になります。使用する安全なパスワードのパスワードハッシュを作成する方法については、[3.3.1項「OSユーザの設定」](#)を参照してください。

キーマップを、インストール後にシステムが使用する実際のキーボードレイアウトに設定します。



注記

SUSEではRPMパッケージのチェックにGPGキーを提供しないため、オプション`noGPGCheck: true`を使用しています。運用環境での使用に推奨するより安全なセットアップに関する包括的なガイドについては、[アップストリームのパッケージインストールガイド \(https://github.com/suse-edge/edge-image-builder/blob/release-1.3/docs/installing-packages.md\)](https://github.com/suse-edge/edge-image-builder/blob/release-1.3/docs/installing-packages.md) を参照してください。

何度か言及したように、SUSE Multi-Linux Managerホストには、エッジノードがブートするネットワークで解決できる完全修飾ホスト名が必要です。

`activationKey`の値は、SUSE Multi-Linux Managerで作成したキーと一致する必要があります。

インストール後にエッジノードをSUSE Multi-Linux Managerに自動的に登録するインストールイメージを構築するには、次の2つのアーティファクトを準備する必要があります。

- SUSE Multi-Linux Managerの管理エージェントをインストールするSalt minionパッケージ
- SUSE Multi-Linux ManagerサーバのCA証明書

4.3.1 `venv-salt-minion`パッケージのダウンロード

`/opt/eib`に、サブディレクトリ`rpms`を作成します。

SUSE Multi-Linux Managerサーバからそのディレクトリにパッケージ`venv-salt-minion`をダウンロードします。Web UIから取得する場合は、[\[ソフトウェア\]](#) > [\[チャンネル一覧\]](#) にあるパッケージを見つけてSUSE-Manager-Tools…チャンネルからダウンロードするか、SUSE Multi-Linux Managerの「ブートストラップリポジトリ」から`curl`などのツールを使用してダウンロードします。

```
curl -O http://${HOSTNAME_OF_SUSE_MANAGER}/pub/repositories/slmicro/6/1/bootstrap/x86_64/venv-salt-minion-3006.0-8.1.x86_64.rpm
```

実際のパッケージ名は、新しいリリースがすでにリリースされている場合は異なる場合があります。複数のパッケージから選択できる場合は、必ず最新のパッケージを選択してください。

SUSE Multi-Linux Managerのリリースノート (https://www.suse.com/releasenotes/x86_64/multi-linux-manager/5.1/index.html#_bootstrapping_sl_micro_6_1_clients) に記載されている問題を回避するには、最新バージョンのビルドキーパッケージをrpmsディレクトリ(このドキュメントの作成時点ではsuse-build-key-12.0-slfo.1.1_3.1.noarch.rpm)に配置する必要があります。このパッケージは、SL MicroのPoolチャンネルのPackagesタブ経由で、SUSE Multi-Linux Managerの「ソフトウェア」セクションにあります。「詳細」ビューには「ダウンロード」ボタンがあります。

4.4 SUSE Multi-Linux Manager CA証明書のダウンロード

/opt/eib/に、サブディレクトリcertificatesを作成します。

SUSE Multi-Linux ManagerからそのディレクトリにCA証明書をダウンロードします。

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



警告

証明書の名前をRHN-ORG-TRUSTED-SSL-CERT.crtに変更する必要があります。その後、Edge Image Builderは、インストール時に証明書がエッジノードにインストールされ、アクティブ化されていることを確認します。

これでEdge Image Builderを実行できます。

```
cd /opt/eib
podman run --rm -it --privileged -v /opt/eib:/eib \
registry.suse.com/edge/3.4/edge-image-builder:1.3.0 \
build --definition-file iso-definition.yaml
```

YAML定義ファイルに別の名前を使用している場合や別のバージョンのEdge Image Builderを使用する場合は、それに応じてコマンドを調整する必要があります。

ビルドが完了したら、/opt/eibディレクトリにインストールISOがeib-image.isoとして保存されます。

このイメージは、SUSE Multi-Linux Managerに登録しようとするノードをデプロイするために使用できます。

ノードのインストールが完了すると、SUSE Multi-Linux Managerの [Salt/キー] セクションにノードのキーが保留中として表示されます。キーを受け入れると、ノードはSUSE Multi-Linux Managerに自動的にオンボードされ、そのプロセスが完了すると [システム] リストに表示されます。ノードには、アクティベーションキーで指定したシステムグループが割り当てられます。

その後、追加の設定を適用する前に再起動をスケジュールする必要があります。

キーの受け入れは、[こちら \(https://docs.saltproject.io/en/latest/topics/tutorials/autoaccept_grains.html\)](https://docs.saltproject.io/en/latest/topics/tutorials/autoaccept_grains.html)  で説明されているように、ホワइटリストを使用して完全に自動化できることに注意してください。

II コンポーネント

- 5 Rancher **68**
- 6 Rancher Dashboard拡張機能 **71**
- 7 Rancher Turtles **76**
- 8 Fleet **78**
- 9 SUSE Linux Micro **84**
- 10 Metal³ **86**
- 11 Edge Image Builder **88**
- 12 Edgeネットワーキング **90**
- 13 Elemental **113**
- 14 K3s **115**
- 15 RKE2 **117**
- 16 SUSE Storage **120**
- 17 SUSE Security **128**
- 18 MetalLB **130**
- 19 Endpoint Copier Operator **132**
- 20 Edge Virtualization **133**
- 21 System Upgrade Controller **147**
- 22 Upgrade Controller **156**
- 23 SUSE Multi-Linux Manager **170**

Edgeのコンポーネントのリスト

5 Rancher

<https://ranchermanager.docs.rancher.com/v2.12>にあるRancherのドキュメントを参照してください。

Rancherは、オープンソースの強力なKubernetes管理プラットフォームであり、複数の環境にまたがるKubernetesクラスタのデプロイメント、運用、および監視を効率化します。オンプレミス、クラウド、エッジのいずれのクラスタを管理する場合でも、Rancherは、Kubernetesに関するあらゆるニーズに対応する、統合された中央プラットフォームを提供します。

5.1 Rancherの主な機能

- **マルチクラスタ管理:** Rancherの直感的なインターフェースを使用して、パブリッククラウド、プライベートデータセンター、エッジロケーションのどこからでもKubernetesクラスタを管理できます。
- **セキュリティとコンプライアンス:** Rancherでは、Kubernetes環境全体にセキュリティポリシー、ロールベースのアクセス制御(RBAC)、およびコンプライアンス標準が適用されます。
- **クラスタ操作のシンプル化:** Rancherはクラスタのプロビジョニング、アップグレード、トラブルシューティングを自動化し、あらゆる規模のチームでKubernetesの操作をシンプル化します。
- **中央型のアプリケーションカタログ:** Rancherアプリケーションカタログは、多種多様なHelmチャートとKubernetes Operatorを提供し、コンテナ化アプリケーションのデプロイと管理を容易にします。
- **継続的デリバリ:** RancherはGitOpsと継続的インテグレーション/継続的デリバリパイプラインをサポートしており、自動化および効率化されたアプリケーションデリバリプロセスを実現します。

5.2 SUSE EdgeでのRancherの使用

Rancherは、SUSE Edgeスタックに複数のコア機能を提供します。

5.2.1 Kubernetesの集中管理

大量の分散クラスタが存在する一般的なエッジデプロイメントでは、Rancherは、これらのKubernetesクラスタを管理するための中央コントロールプレーンとして機能します。プロビジョニング、アップグレード、監視、およびトラブルシューティングのための統合インターフェースを提供し、操作をシンプル化し、一貫性を確保します。

5.2.2 クラスタデプロイメントのシンプル化

Rancherは、軽量なSUSE Linux Microオペレーティングシステム上でのKubernetesクラスタの作成を効率化し、Kubernetesの堅牢な機能を備えたエッジインフラストラクチャの展開を容易にします。

5.2.3 アプリケーションのデプロイメントと管理

統合されたRancherアプリケーションカタログは、SUSE Edgeクラスタ全体でのコンテナ化アプリケーションのデプロイと管理をシンプル化し、エッジワークロードのシームレスなデプロイメントを可能にします。

5.2.4 セキュリティとポリシーの適用

Rancherは、ポリシーベースのガバナンスツール、ロールベースのアクセス制御(RBAC)を備えているほか、外部の認証プロバイダと統合できます。これにより、SUSE Edgeのデプロイメントは、分散環境において重要なセキュリティとコンプライアンスを維持できます。

5.3 ベストプラクティス

5.3.1 GitOps

RancherにはビルトインコンポーネントとしてFleetが含まれており、Gitに保存されたコードでクラスタ設定やアプリケーションのデプロイメントを管理できます。

5.3.2 可観測性

Rancherには、PrometheusやGrafanaなどのビルトインのモニタリングおよびログツールが含まれており、クラスタのヘルスとパフォーマンスについて包括的な洞察を得ることができます。

5.4 Edge Image Builderを使用したインストール

SUSE Edgeは、SUSE Linux Micro OSのゴールデンイメージをカスタマイズするために第11章「Edge Image Builder」を使用しています。EIBでプロビジョニングしたKubernetesクラスタ上にRancherをエアギャップインストールするには、27.6項「Rancherのインストール」に従ってください。

5.5 追加リソース

- Rancherのドキュメント (<https://rancher.com/docs/>) 
- Rancher Academy (<https://www.rancher.academy/>) 
- Rancher Community (<https://rancher.com/community/>) 
- Helmチャート (<https://helm.sh/>) 
- Kubernetes Operator (<https://operatorhub.io/>) 

6 Rancher Dashboard拡張機能

拡張機能により、ユーザ、開発者、パートナー、および顧客はRancher UIを拡張および強化できます。SUSE Edgeでは、KubeVirtとAkriのダッシュボード拡張機能を提供しています。

Rancher Dashboard拡張機能の一般的な情報については、[Rancherのドキュメント](#)を参照してください。

6.1 インストール

ダッシュボード拡張機能を含むすべてのSUSE Edge 3.4コンポーネントは、OCIアーティファクトとして配布されます。SUSE Edge拡張機能をインストールするには、Rancher Dashboard UI、Helm、またはFleetを使用できます。

6.1.1 Rancher Dashboard UIを使用したインストール

1. ナビゲーションサイドバーの [**Configuration (設定)**] セクションで、 [**Extensions (拡張機能)**] をクリックします。
2. [Extensions (拡張機能)] ページで、右上にある3つのドットメニューをクリックし、 [**Manage Repositories (リポジトリの管理)**] を選択します。
各拡張機能は、独自のOCIアーティファクトで配布されます。これらはSUSE Edge Helmチャートリポジトリから入手できます。
3. [**Repositories (リポジトリ)**] ページで、 [Create (作成)] をクリックします。
4. フォームにリポジトリ名とURLを指定して、 [Create (作成)] をクリックします。
SUSE Edge HelmチャートリポジトリURL: <oci://registry.suse.com/edge/charts>

local

Only User Namespaces

Cluster

Workloads

Apps

Charts

Installed Apps

Repositories

Recent Operations

Service Discovery

Storage

Policy

More Resources

Repository: Create

Name *

suse-edge

Description

SUSE Edge Helm charts repository

Target

☐ http(s) URL to an index generated by Helm
 ☐ Git repository containing Helm chart or cluster template definitions
 ☒ OCI Repository

OCI URLs must ONLY target helm chart/s.

For large repositories containing many charts consider targeting a specific namespace or chart to improve performance, for example with oci://<registry-host>/<namespace> or oci://<registry-host>/<namespace>/<chart-name>.

OCI Repository Host URL *

oci://registry.suse.com/edge/charts

Refresh Interval

default: 24

hours

Authentication

None

CA Cert Bundle

☐ Skip TLS Verifications
 ☐ Insecure Plain Http

Exponential Back Off

Min Wait Time

default: 1

Seconds

Max Wait Time

default: 5

Seconds

Max Number of Retries

default: 5

Labels

Key/value pairs that are attached to objects which specify identifying attributes.

Add Label

Annotations

Add Annotation

Cancel

Create

5. 拡張機能リポジトリがリストに追加され、**Active (アクティブ)** 状態であることがわかります。

local

Only User Namespaces

Cluster

Workloads

Apps

Charts

Installed Apps

Repositories

Recent Operations

Service Discovery

Storage

Policy

More Resources

A chart repository is a Helm repository or Rancher Prime git based application catalog. It provides the list of available charts in the cluster. Cluster Templates are deployed via Helm charts.

Create

Repositories ☆

Refresh

Disable

Download YAML

Delete

Filter

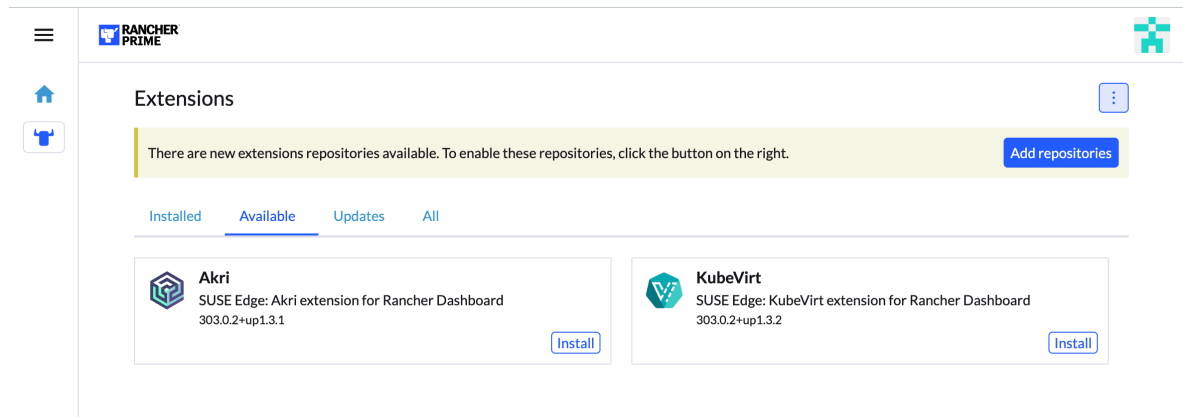
State	Name	Type	URL	Branch	Age
Active	Partners	git	https://git.rancher.io/partner-charts	main	33 mins
Active	Rancher Prime	git	https://git.rancher.io/charts	release-v2.11	33 mins
Active	RKE2	git	https://git.rancher.io/rke2-charts	main	33 mins
Active	suse-edge	oci	oci://registry.suse.com/edge/charts	—	4 mins

72

Rancher Dashboard UIを使用したインストール

6. ナビゲーションサイドバーの **【Configuration (設定)】** セクションの **【Extensions (拡張機能)】** に戻ります。

【Available (利用可能)】 タブで、インストール可能な拡張機能を確認できます。



7. 拡張機能カードで、 **【Install (インストール)】** をクリックし、インストールを確認します。

拡張機能がインストールされると、Rancher UIによってページの再ロードが促されます。拡張機能のインストールのRancherドキュメントページを参照してください。

6.1.2 Helmを使用したインストール

```
# KubeVirt extension
helm install kubevirt-dashboard-extension oci://registry.suse.com/edge/charts/kubevirt-
dashboard-extension --version 304.0.2+up1.3.2 --namespace cattle-ui-plugin-system
```



注記

拡張機能は `cattle-ui-plugin-system` ネームスペースにインストールする必要があります。



注記

拡張機能がインストールされたら、Rancher Dashboard UIを再ロードする必要があります。

6.1.3 Fleetを使用したインストール

Fleetを使用してダッシュボード拡張機能をインストールするには、カスタムのfleet.yamlバンドル設定ファイルを使用して、Gitリポジトリを指すgitRepoリソースを定義する必要があります。

```
# KubeVirt extension fleet.yaml
defaultNamespace: cattle-ui-plugin-system
helm:
  releaseName: kubevirt-dashboard-extension
  chart: oci://registry.suse.com/edge/charts/kubevirt-dashboard-extension
  version: "304.0.2+up1.3.2"
```



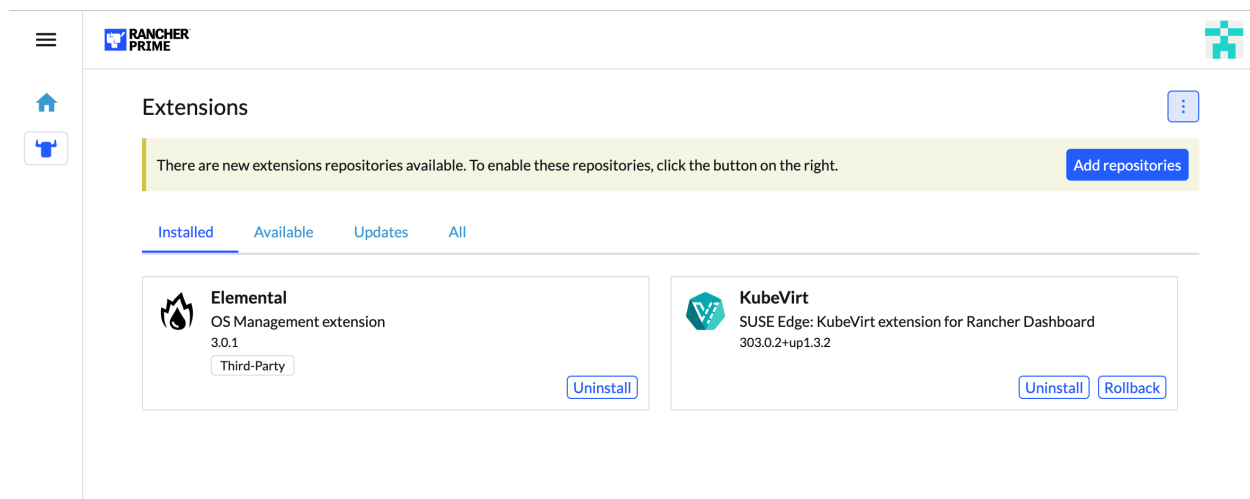
注記

拡張機能を正しくインストールするには、releaseNameプロパティが必須であり、拡張機能名と一致している必要があります。

```
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
```

詳細については、[第8章「Fleet」](#)とfleet-examplesリポジトリを参照してください。

拡張機能がインストールされると、その拡張機能が **[Extensions (拡張機能)]** セクションの **[Installed (インストール済み)]** タブに表示されます。拡張機能はApps/Marketplace経由でインストールされたものではないため、「Third-Party (サードパーティ)」というラベルが付きます。



6.2 KubeVirtダッシュボード拡張機能

KubeVirt拡張機能は、Rancher Dashboard UIに基本的な仮想マシン管理機能を提供します。その機能については、[20.7.2項「KubeVirt Rancher Dashboard拡張機能の使用」](#)を参照してください。

7 Rancher Turtles

<https://documentation.suse.com/cloudnative/cluster-api/> にあるRancher Turtlesのドキュメントを参照してください。

Rancher Turtlesは、Rancher ManagerとCluster API (CAPI)の統合を提供するKubernetes Operatorであり、RancherにCAPIの完全サポートをもたらすことを目的としています。

7.1 Rancher Turtlesの主な機能

- CAPIでプロビジョニングされたクラスタにRancher Cluster Agentをインストールして、CAPIクラスタをRancherに自動的にインポートします。
- CAPI Operator (<https://cluster-api-operator.sigs.k8s.io/>) を介してCAPIコントローラの依存関係をインストールして設定します。

7.2 SUSE EdgeでのRancher Turtlesの使用

SUSE Edgeスタックは、Rancher Turtlesを次のコンポーネントを有効にする特定の設定でインストールするhelmラッパーチャートを提供します。

- コアCAPIコントローラコンポーネント
- RKE2コントロールプレーンおよびブートストラッププロバイダコンポーネント
- Metal3 (第10章 「Metal³」)インフラストラクチャプロバイダコンポーネント

ラッパーチャートを介してインストールされるデフォルトプロバイダのみがサポートされます。代替コントロールプレーン、ブートストラップ、およびインフラストラクチャプロバイダは現在、SUSE Edgeスタックの一部としてサポートされていません。

7.3 Rancher Turtlesのインストール

Rancher Turtlesは、Metal3クイックスタート(第1章 「Metal³を使用したBMCの自動デプロイメント」)ガイド、または管理クラスタ(第40章 「管理クラスタの設定」)ドキュメントに従ってインストールできます。

7.4 追加リソース

- Rancherのドキュメント (<https://rancher.com/docs/>) 
- Cluster APIのドキュメント (<https://cluster-api.sigs.k8s.io/>) 

8 Fleet

Fleet (<https://fleet.rancher.io>) は、ユーザがローカルクラスタをより細かく制御できるようにするとともに、GitOpsを通じて常時監視を行えるようにすることを目的に設計されたコンテナ管理およびデプロイメントエンジンです。Fleetはスケール能力に重点を置いているだけでなく、クラスタに何がインストールされているかを正確に監視するための高度な制御と可視性もユーザに提供します。

Fleetは、生のKubernetes YAML、Helmチャート、Kustomize、またはこれら3つの組み合わせのGitからデプロイメントを管理できます。ソースにかかわらず、すべてのリソースは動的にHelmチャートに変換され、すべてのリソースをクラスタにデプロイするエンジンとしてHelmが使用されます。その結果、ユーザはクラスタの高度な制御、一貫性、監査能力を実現できます。

Fleetの仕組みについては、[Fleetアーキテクチャ \(https://ranchermanager.docs.rancher.com/v2.12/integrations-in-rancher/fleet/architecture\)](https://ranchermanager.docs.rancher.com/v2.12/integrations-in-rancher/fleet/architecture) を参照してください。

8.1 Helmを使用したFleetのインストール

FleetはRancherにビルトインされていますが、Helmを使用して、スタンドアロンアプリケーションとして任意のKubernetesクラスタにインストール (<https://fleet.rancher.io/installation>) することもできます。

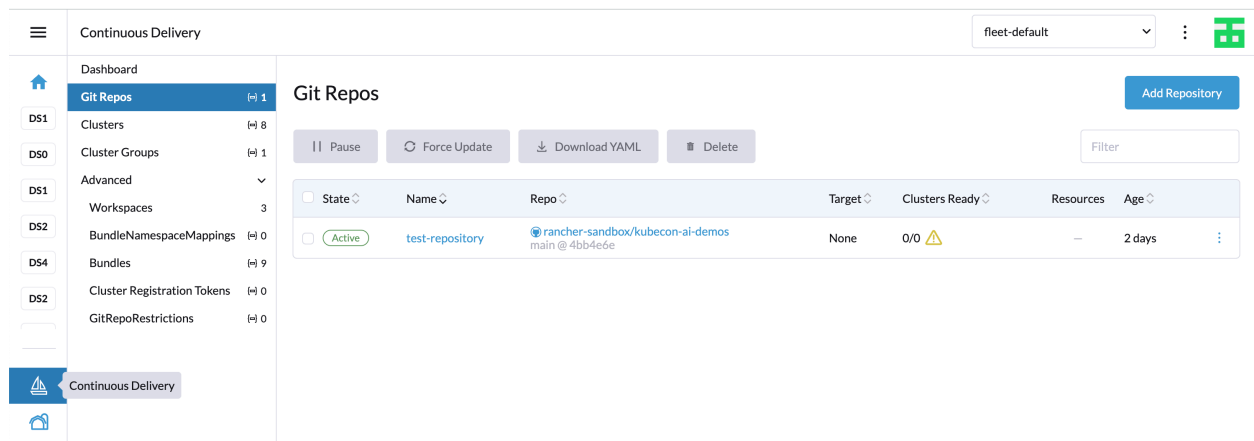
8.2 RancherでのFleetの使用

Rancherは、Fleetを使用してアプリケーションを管理対象クラスタ全体にデプロイします。Fleetを使用した継続的デリバリーにより、大量のクラスタで実行されるアプリケーションを管理するために設計された、大規模なGitOpsが導入されます。

FleetはRancherと統合してその一部として機能します。Rancherで管理されるクラスタには、インストール/インポートプロセスの一部としてFleetエージェントが自動的にデプロイされるため、クラスタはすぐにFleetで管理できるようになります。

8.3 Rancher UIでのFleetへのアクセス

FleetはRancherにプリインストールされており、Rancher UIの **[Continuous Delivery (継続的デリバリー)]** オプションで管理されます。



[Continuous Delivery (継続的デリバリ)] セクションは次の項目で構成されます。

8.3.1 Dashboard (ダッシュボード)

すべてのワークスペースにわたるすべてのGitOpsリポジトリの概要ページ。リポジトリのあるワークスペースのみが表示されます。

8.3.2 Git repos (Gitリポジトリ)

選択したワークスペース内のGitOpsリポジトリのリスト。ページ上部のドロップダウンリストを使用してアクティブなワークスペースを選択します。

8.3.3 Clusters (クラスタ)

管理対象クラスタのリスト。デフォルトでは、Rancherで管理されているすべてのクラスタが `fleet-default` ワークスペースに追加されます。 `fleet-local` ワークスペースにはローカル(管理)クラスタが含まれます。ここから、クラスタを `Pause` (一時停止) または `Force Update` (強制的に更新) したり、クラスタを別のワークスペースに移動したりすることができます。クラスタを編集すると、クラスタのグループ化に使用するラベルや注釈を更新できます。

8.3.4 Cluster groups (クラスタグループ)

このセクションでは、セレクトラを使用してワークスペース内のクラスタのカスタムグループを作成できます。

8.3.5 Advanced (詳細)

[Advanced (詳細)] セクションでは、ワークスペースやその他の関連するFleetリソースを管理できます。

8.4 Rancher Dashboardを使用してRancherおよびFleetとともにKubeVirtをインストールする例

1. `fleet.yaml` ファイルを含むGitリポジトリを作成します。

```
defaultNamespace: kubevirt
helm:
  chart: "oci://registry.suse.com/edge/charts/kubevirt"
  version: "304.0.1+up0.6.0"
  # kubevirt namespace is created by kubevirt as well, we need to take ownership of
  it
  takeOwnership: true
```

2. Rancher Dashboardで、**[#] > [Continuous Delivery (継続的デリバリー)] > [Git Repos (Gitリポジトリ)]** に移動して、**[Add Repository (リポジトリの追加)]** をクリックします。
3. リポジトリの作成ウィザードの指示に従ってGitリポジトリを作成します。 **[Name (名前)]**、**[Repository URL (リポジトリのURL)]** (前の手順で作成したGitリポジトリを参照)を指定し、適切なブランチまたはリビジョンを選択します。より複雑なリポジトリの場合は、**[Paths (パス)]** を指定して、1つのリポジトリで複数のディレクトリを使用します。

Continuous Delivery

fleet-default

Git Repo: Create

Create: Step 1
Define repository details

Repository Details

Name: kubevirt

Description: Any text you want that better describes this resource

Enter a valid HTTPS or SSH URL to a git repository.

Repository URL: https://github.com/loose-edges/fleet-examples.git

Branch: A Branch

Branch Name: main

Git Authentication: None

Helix Authentication: None

TLS Certificate Verification: Require a valid certificate

Resource Handling

Enable Self-Healing

When enabled, Fleet will ensure that the cluster resources are kept in sync with the git repository. All resource changes made on the cluster will be lost.

Always Keep Resources

When enabled above, resources will be kept when deleting a GitRepo or Bundle - only Helm release secrets will be deleted.

Paths

fleets/kubevirt

Add Path

Remove

Cancel Edit as YAML Next

4. [Next (次へ)] をクリックします。

5. 次の手順では、ワークロードをデプロイする場所を定義できます。クラスタの選択では複数の基本オプションがあります。クラスタをまったく選択しないことも、すべてのクラスタを選択することも、特定の管理対象クラスタやクラスタグループ(定義されている場合)を直接選択することもできます。[Advanced (詳細)] オプションを使用すると、YAML経由でセレクタを直接編集できます。

Continuous Delivery

Git Repo: Create

Create: Step 2
Define target details

Repository Details

Deploy To

Target

No Clusters

No Clusters

All Clusters in the Workspace

Advanced

Clusters

ds15

ds4

ds5

ds6

ds7

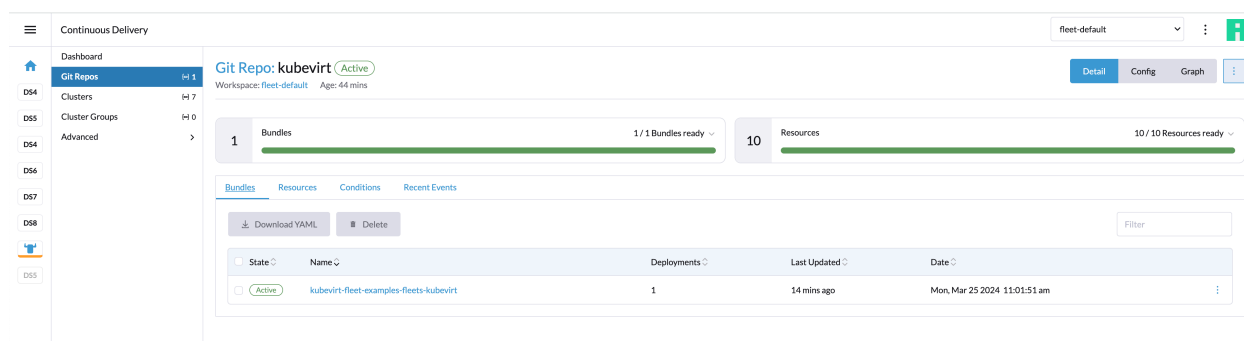
ds8

6. [Create (作成)] をクリックします。リポジトリが作成されます。今後、ワークロードはリポジトリ定義に一致するクラスタにインストールされ、同期が維持されます。

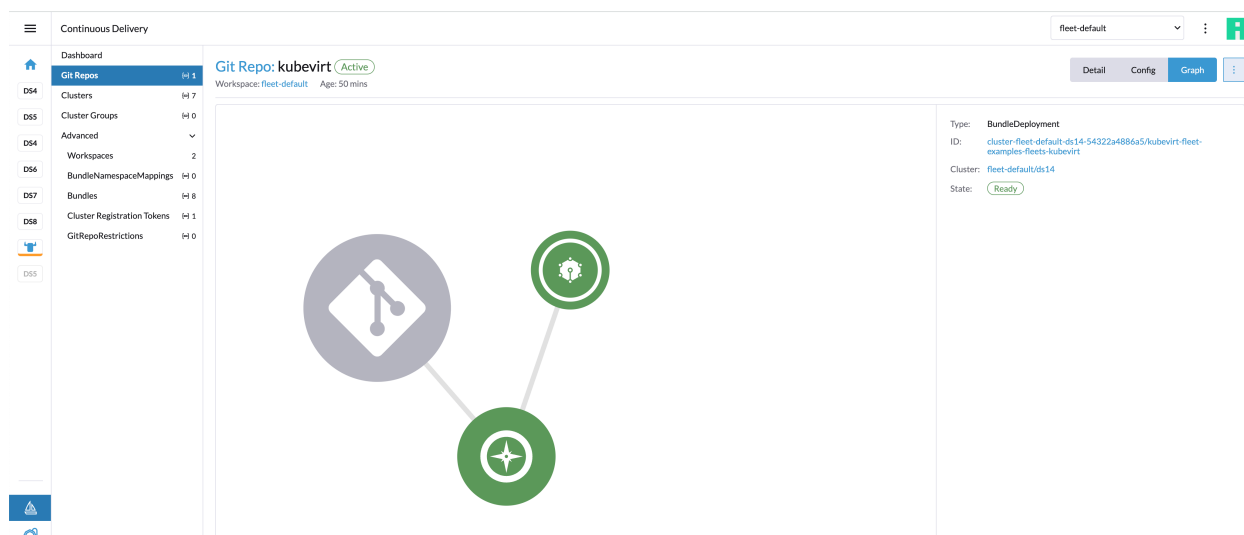
8.5 デバッグとトラブルシューティング

ナビゲーションの [Advanced (詳細)] セクションでは、下位レベルのFleetリソースの概要が表示されます。バンドル (<https://fleet.rancher.io/ref-bundle-stages>) は、Gitからのリソースのオーケストレーションに使用される内部リソースです。Gitリポジトリがスキャンされると、バンドルが1つ以上生成されます。

特定のリポジトリに関連するバンドルを見つけるには、[Git Repos (Gitリポジトリ)] の [Detail (詳細)] ページに移動し、[Bundles (バンドル)] タブをクリックします。



クラスタごとに、作成されたBundleDeploymentリソースにバンドルが適用されます。BundleDeploymentの詳細を表示するには、[Git Repos (Gitリポジトリ)] の [Detail (詳細)] ページの右上にある [Graph (グラフ)] ボタンをクリックします。[Repo (リポジトリ)] > [Bundles (バンドル)] > [BundleDeployments] のグラフがロードされます。グラフ内のBundleDeploymentをクリックすると詳細が表示され、[Id (ID)] をクリックすると BundleDeployment YAMLが表示されます。



Fleetのトラブルシューティングのヒントに関する追加情報については、[こちら \(https://fleet.rancher.io/troubleshooting\)](https://fleet.rancher.io/troubleshooting) を参照してください。

8.6 Fleetの例

Edgeチームは、Fleetを使用してEdgeプロジェクトをインストールする例を含むリポジトリ (<https://github.com/suse-edge/fleet-examples>)  を維持しています。

Fleetプロジェクトには、Gitリポジトリ構造 (<https://fleet.rancher.io/gitrepo-content>)  のすべてのユースケースをカバーするfleet-examples (<https://github.com/rancher/fleet-examples>)  リポジトリが含まれています。

9 SUSE Linux Micro

SUSE Linux Micro公式ドキュメント (<https://documentation.suse.com/sle-micro/6.1/>)  を参照してください。

SUSE Linux Microは、エッジ向けの軽量でセキュアなオペレーティングシステムです。SUSE Linux Microには、SUSE Linux Enterpriseのエンタープライズ向けに強化されたコンポーネントと、開発者が最新のイミュータブルオペレーティングシステムに求める機能が統合されています。その結果、クラス最高のコンプライアンスを備えた信頼性の高いインフラストラクチャプラットフォームが実現し、使いやすさも向上しています。

9.1 SUSE EdgeでのSUSE Linux Microの用途

SUSEでは、SUSE Linux Microをプラットフォームスタックのベースオペレーティングシステムとして使用します。これにより、構築基盤となる、安全で安定した最小限のベースが提供されます。

SUSE Linux Microでは、独自の方法でファイルシステム(Btrfs)スナップショットを使用しており、アップグレードで問題が発生した場合に簡単にロールバックできます。これにより、問題が発生した場合、物理的にアクセスしなくてもプラットフォーム全体をリモートで安全にアップグレードできます。

9.2 ベストプラクティス

9.2.1 インストールメディア

SUSE Edgeは、Edge Image Builder (第11章「Edge Image Builder」)を使用して、SUSE Linux Microのセルフインストールのインストールイメージを事前設定します。

9.2.2 ローカル管理

SUSE Linux Microには、Webアプリケーションでホストをローカルに管理できるCockpitが付属しています。

このサービスはデフォルトでは無効になっていますが、systemdサービス`cockpit.socket`を有効にすることで開始できます。

9.3 既知の問題

- 現時点では、SUSE Linux Microで利用可能なデスクトップ環境はありませんが、コンテナ化されたソリューションが開発中です。

10 Metal³

Metal³ (<https://metal3.io/>)  は、Kubernetesにベアメタルインフラストラクチャ管理機能を提供するCNCFプロジェクトです。

Metal³は、Redfish (<https://www.dmtf.org/standards/redfish>)  などのアウトオブバンドプロトコルを介した管理をサポートするベアメタルサーバのライフサイクルを管理するためのKubernetesネイティブリソースを提供します。

また、Cluster API (CAPI) (<https://cluster-api.sigs.k8s.io/>)  も十分にサポートされており、広く採用されているベンダニュートラルなAPIを使用して、複数のインフラストラクチャプロバイダにわたってインフラストラクチャリソースを管理できます。

10.1 SUSE EdgeでのMetal³の用途

この方法は、ターゲットハードウェアがアウトオブバンド管理をサポートしていて、完全に自動化されたインフラストラクチャ管理フローが望まれるシナリオで役立ちます。

この方法では宣言型APIが提供されており、このAPIを使用することで、検査、クリーニング、プロビジョニング/プロビジョニング解除の自動化を含む、ベアメタルサーバのインベントリと状態の管理が可能になります。

10.2 既知の問題

- アップストリームのIPアドレス管理コントローラ (<https://github.com/metal3-io/ip-address-manager>)  は、SUSEが選択したネットワーク設定ツールとの互換性がまだないため、現在はサポートされていません。
- 関連して、IPAMリソースとMetal3DataTemplateのnetworkDataフィールドはサポートされていません。
- redfish-virtualmediaを介したデプロイメントのみが現在サポートされています。
- Ironi Python Agent (IPA)とターゲットオペレーティングシステム(SL Micro 6.0/6.1)の間で、特に、デバイスに予測可能な名前を設定しようとする際に、ネットワークデバイス名の不整合が発生する可能性があります。

これが発生するのは、Ironi Python Agent (IPA)のカーネルが現在、ターゲットオペレーティングシステム(SL Micro 6.0/6.1)のカーネルと整合していないためです。これにより、ネットワークドライバに不整合が生じ、SL Microが想定する命名パターンとは異なる命名パターンでIPAがネットワークデバイスを検出できるようになります。

当面の回避策として、以下の2つの異なるアプローチが利用可能です。* ネットワーク設定で2つの異なるシークレットを作成します。1つはIPAが検出するデバイス名を使用したIPAで使用されるシークレットで、BareMetalHost定義のpreprovisioningNetworkDataNameとして使用します。もう1つはSL Microが検出するデバイス名を使用したシークレットで、BareMetalHost定義のnetworkData.nameとして使用します。* 代わりに、生成されたnmconnectionファイル上の他のインタフェースを参照するためにUUIDを使用します。詳細は、[ヒントとコツ \(..tips/metal3.adoc\)](https://github.com/openshift/openshift-docs/blob/master/contributing/working-with-metal3.adoc#tips-and-tricks)  セクションを参照してください。

11 Edge Image Builder

公式リポジトリ (<https://github.com/suse-edge/edge-image-builder>)  を参照してください。

Edge Image Builder (EIB)は、マシンをブートストラップするためのCustomized, Ready-to-Boot (CRB)ディスクイメージの生成を効率化するツールです。これらのイメージにより、SUSEソフトウェアスタック全体を単一のイメージでエンドツーエンドにデプロイできます。

EIBはあらゆるプロビジョニングシナリオ向けのCRBイメージを作成できますが、EIBが非常に大きな価値を発揮するのは、ネットワークが制限されているか、完全に分離されているエアギャップデプロイメントにおいてです。

11.1 SUSE EdgeでのEdge Image Builderの用途

SUSE Edgeでは、さまざまなシナリオ用にカスタマイズされたSUSE Linux Microイメージをシンプルかつ迅速に設定するためにEIBを使用します。これらのシナリオには、以下を使用する仮想マシンとベアメタルマシンのブートストラップが含まれます。

- K3s/RKE2 Kubernetesの完全なエアギャップデプロイメント(シングルノードとマルチノード)
- HelmチャートとKubernetesマニフェストの完全なエアギャップデプロイメント
- Elemental APIを介したRancherへの登録
- Metal³
- カスタマイズされたネットワーキング(静的IP、ホスト名、VLAN、ボンディングなど)
- カスタマイズされたオペレーティングシステム設定(ユーザ、グループ、パスワード、SSHキー、プロキシ、NTP、カスタムSSL証明書など)
- ホストレベルおよびサイドロードRPMパッケージのエアギャップインストール(依存関係の解決を含む)
- OS管理のためのSUSE Multi-Linux Managerへの登録
- 組み込みコンテナイメージ
- カーネルコマンドライン引数

- ブート時に有効化/無効化されるsystemdユニット
- 手動タスク用のカスタムスクリプトとファイル

11.2 はじめに

Edge Image Builderの使用とテストに関する包括的なドキュメントについては、[こちら](https://github.com/suse-edge/edge-image-builder/tree/release-1.3/docs) (<https://github.com/suse-edge/edge-image-builder/tree/release-1.3/docs>) を参照してください。

また、基本的なデプロイメントシナリオを説明している第3章「Edge Image Builderを使用したスタンドアロンクラスタ」を参照してください。

このツールに慣れたら、EIBの「ヒントとコツ」セクション(パートIV「ヒントとコツ」)ページでさらに役立つ情報を見つけてください。

11.3 既知の問題

- EIBは、Helmチャートをテンプレート化してテンプレート内のすべてのイメージを解析することで、Helmチャートをエアギャップ化します。Helmチャートですべてのイメージをテンプレート内に保持せず、代わりにイメージをサイドロードする場合、EIBではこれらのイメージを自動的にエアギャップ化できません。これを解決するには、検出されないイメージを定義ファイルの`embeddedArtifactRegistry`セクションに手動で追加します。

12 Edgeネットワークキング

このセクションでは、SUSE Edgeソリューションにおけるネットワーク設定へのアプローチについて説明します。宣言的な方法でSUSE Linux Micro上でNetworkManagerを設定する方法を示し、関連ツールの統合方法について説明します。

12.1 NetworkManagerの概要

NetworkManagerは、プライマリネットワーク接続と他の接続インタフェースを管理するツールです。

NetworkManagerは、ネットワーク設定を、望ましい状態が含まれる接続ファイルとして保存します。これらの接続は、`/etc/NetworkManager/system-connections/`ディレクトリにファイルとして保存されます。

NetworkManagerの詳細については、[SUSE Linux Microのドキュメント \(https://documentation.suse.com/sle-micro/6.1/html/Micro-network-configuration/index.html\)](https://documentation.suse.com/sle-micro/6.1/html/Micro-network-configuration/index.html) を参照してください。

12.2 nmstateの概要

nmstateは広く採用されているライブラリ(CLIツールが付属)であり、定義済みスキーマを使用したネットワーク設定用の宣言型APIを提供します。

nmstateの詳細については、[アップストリームドキュメント \(https://nmstate.io/\)](https://nmstate.io/) を参照してください。

12.3 NetworkManager Configurator (nmc)の概要

SUSE Edgeで利用可能なネットワークのカスタマイズオプションは、NetworkManager Configurator (短縮名は**nmc**)と呼ばれるCLIツールを使用して実行します。このツールはnmstateライブラリによって提供される機能を利用しているため、静的IPアドレス、DNSサーバ、VLAN、ボンディング、ブリッジなどを完全に設定できます。このツールを使用して、事前定義された望ましい状態からネットワーク設定を生成し、その設定を多数のノードに自動的に適用できます。

NetworkManager Configurator (nmc)の詳細については、[アップストリームリポジトリ \(https://github.com/suse-edge/nm-configurator\)](https://github.com/suse-edge/nm-configurator) を参照してください。

12.4 SUSE EdgeでのNetworkManager Configuratorの用途

SUSE Edgeは、**nmc**を利用して次のようなさまざまなプロビジョニングモデルでネットワークをカスタマイズします。

- ダイレクトネットワークプロビジョニングシナリオにおけるカスタムネットワーク設定 (第1章 「Metal³を使用したBMCの自動デプロイメント」)
- イメージベースのプロビジョニングシナリオにおける宣言的な静的設定(第3章 「Edge Image Builderを使用したスタンドアロンクラスタ」)

12.5 Edge Image Builderを使用した設定

Edge Image Builder (EIB)は、1つのOSイメージで複数のホストを設定できるツールです。このセクションでは、宣言型アプローチを使用して、どのように目的のネットワーク状態を記述するかと、それらがどのように各NetworkManager接続に変換され、プロビジョニングプロセス中に適用されるかを示します。

12.5.1 前提条件

このガイドに従って操作を進める場合、以下がすでに用意されていることを想定しています。

- SLES 15 SP6またはopenSUSE Leap 15.6を実行しているAMD64/Intel 64物理ホスト(または仮想マシン)
- 利用可能なコンテナランタイム(Podmanなど)
- SUSE Linux Micro 6.1 RAWイメージのコピー(こちら (<https://www.suse.com/download/sle-micro/>) にあります)

12.5.2 Edge Image Builderのコンテナイメージの取得

EIBのコンテナイメージは一般に公開されており、次のコマンドを実行してSUSE Edgeレジストリからダウンロードできます。

```
podman pull registry.suse.com/edge/3.4/edge-image-builder:1.3.0
```

12.5.3 イメージ設定ディレクトリの作成

まず設定ディレクトリを作成しましょう。

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

ダウンロードしたゴールデンイメージのコピーを確実に設定ディレクトリに移動します。

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



注記

EIBは、ゴールデンイメージの入力を変更することはありません。変更を加えた新しいイメージを作成します。

この時点では、設定ディレクトリは次のようになっているはずです。

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

12.5.4 イメージ定義ファイルの作成

定義ファイルには、Edge Image Builderがサポートする設定オプションの大部分を記述します。

OSイメージの非常に基本的な定義ファイルから開始しましょう。

```
cat << EOF > $CONFIG_DIR/definition.yaml
apiVersion: 1.3
image:
  arch: x86_64
  imageType: raw
  baseImage: SL-Micro.x86_64-6.1-Base-GM.raw
  outputImageName: modified-image.raw
operatingSystem:
  users:
    - username: root
      encryptedPassword: $6$jHugJNNd3HElGsUZ
$eodjVe4te5ps44SVcWshdfWizrP.xAyd71CVEXazBJ/.v799/WRCBXxfYmunlB02yp1hm/zb4r8EmnrrNCF.P/
EOF
```

`image`セクションは必須で、入力イメージ、そのアーキテクチャとタイプ、および出力イメージの名前を指定します。`operatingSystem`セクションはオプションであり、プロビジョニングされたシステムに`root/eib`のユーザ名/パスワードでログインできるようにするための設定を含めます。



注記

`openssl passwd -6 <password>`を実行して、独自の暗号化パスワードを自由に使用してください。

この時点では、設定ディレクトリは次のようになっているはずです。

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

12.5.5 ネットワーク設定の定義

先ほど作成したイメージ定義ファイルには、望ましいネットワーク設定が含まれていません。そこで、`network/`という特別なディレクトリの下にその設定を入力します。では、作成してみましょう。

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

前述のように、NetworkManager Configurator (**nmc**)ツールでは、事前定義されたスキーマの形式での入力が必要です。さまざまなネットワーキングオプションの設定方法については、[アップストリームのNMStateの例のドキュメント \(https://nmstate.io/examples.html\)](https://nmstate.io/examples.html) を参照してください。

このガイドでは、次の3つの異なるノードでネットワーキングを設定する方法について説明します。

- 2つのEthernetインタフェースを使用するノード
- ネットワークボンディングを使用するノード
- ネットワークブリッジを使用するノード



警告

特にKubernetesクラスタを設定する場合、まったく異なるネットワークセットアップを運用ビルドで使用することは推奨されません。ネットワーキング設定は通常、特定のクラスタ内のノード間、または少なくともロール間で同種にすることをお勧めします。このガイドにはさまざまな異なるオプションが含まれていますが、これは参考例として提供することのみを目的としています。



注記

以下では、IPアドレス範囲192.168.122.1/24を使用するデフォルトのlibvirtネットワークを想定しています。ご自身の環境でこの範囲が異なる場合は、適宜調整してください。

node1.suse.comという名前の最初のノードに対して、望ましい状態を作成しましょう。

```
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
```

この例では、2つのEthernetインタフェース(eth0とeth3)、要求されたIPアドレス、ルーティング、およびDNS解決の望ましい状態を定義しています。



警告

必ず、すべてのEthernetインタフェースのMACアドレスを記述してください。これらのMACアドレスは、プロビジョニングプロセス中にノードの識別子として使用され、どの設定を適用すべきかを判断するのに役立ちます。このようにして、1つのISOまたはRAWイメージを使用して複数のノードを設定できます。

次は、node2.suse.comという名前の2つ目のノードです。このノードではネットワークボンディングを使用します。

```
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

```

この例では、IPアドレス指定を有効にしていない2つのEthernetインタフェース(eth0とeth1)の望ましい状態と、ラウンドロビンポリシーによるボンディング、およびネットワークトラフィックを転送するために使用する各アドレスを定義します。

最後に、3つ目となる、望ましい状態の最後のファイルを作成します。これはネットワークブリッジを利用し、node3.suse.comという名前です。

```

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

この時点では、設定ディレクトリは次のようになっているはずです。

```

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

```



注記

network/ディレクトリにあるファイル名は意図的なものです。これらの名前は、プロビジョニングプロセス中に設定されるホスト名に対応しています。

12.5.6 OSイメージの構築

これで必要な設定はすべて完了したので、次のコマンドを実行するだけでイメージを構築できます。

```
podman run --rm -it -v $CONFIG_DIR:/eib registry.suse.com/edge/3.4/edge-image-builder:1.3.0 build --definition-file definition.yaml
```

出力は次のようになります。

```
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!
```

上のスニペットからNetworkコンポーネントが正常に設定されていることがわかるので、エッジノードのプロビジョニングに進むことができます。



注記

ログファイル(`network-config.log`)とそれぞれのNetworkManager接続ファイルは、イメージ実行のタイムスタンプ付きディレクトリの下にある、結果の`_build`ディレクトリで検査できます。

12.5.7 エッジノードのプロビジョニング

作成されたRAWイメージをコピーしてみましょう。

```
mkdir edge-nodes && cd edge-nodes
```

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

構築されたイメージを4回コピーしましたが、3つのノードのネットワーク設定しか指定していません。これは、どの目的の設定にも一致しないノードをプロビジョニングするとどうなるかも紹介したいためです。



注記

このガイドでは、ノードプロビジョニングの例に仮想化を使用します。必要な拡張機能がBIOSで有効になっていることを確認してください(詳細については [こちら \(https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-virt-support.html#sec-kvm-requires-hardware\)](https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-virt-support.html#sec-kvm-requires-hardware) を参照してください)。

`virt-install` を使用し、コピーしたRAWディスクを使用して仮想マシンを作成します。各仮想マシンは10GBのRAMと6個のvCPUを使用します。

12.5.7.1 1つ目のノードのプロビジョニング

仮想マシンを作成しましょう。

```
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
```



注記

上記で説明した望ましい状態のMACアドレスと同じMACアドレスを持つネットワークインタフェースを作成することが重要です。

操作が完了すると、次のような内容が表示されます。

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

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

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

```
SSH host key: SHA256:ZN/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+wOWQ (ED25519)
eth0: 192.168.122.50
eth1:
```

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

node1 login:
```

これで、`root:eib`の資格情報ペアを使用してログインできます。ここで提示されている`virsh console`よりもSSHでホストに接続したい場合は、SSHで接続することもできます。

ログインしたら、すべての設定が完了していることを確認しましょう。

ホスト名が適切に設定されていることを確認します。

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

ルーティングが適切に設定されていることを確認します。

```
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
```

インターネット接続が利用できることを確認します。

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

2つのEthernetインタフェースが設定されていて、そのうちの1つだけがアクティブであることを確認します。

```
node1:~ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

```

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

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

```

2つ目のインタフェースが、目的のネットワーキング状態で指定されている定義済みのeth3ではなく、eth1になっていることがわかります。これは、NetworkManager Configurator (nmc)が、MACアドレス34:8a:b1:4b:16:e2を持つNICにOSによって別の名前が付けられていることを検出し、それに応じて設定を調整しているためです。

プロビジョニングのCombustionのフェーズを検査して、この調整が実際に行われたことを確認します。

```

nodel:~ # 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: nodel.suse.com
Apr 23 09:20:19 localhost.localdomain combustion[1360]: [2024-04-23T09:20:19Z INFO
nmc::apply_conf] Set hostname: nodel.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

```

続いて残りのノードをプロビジョニングしますが、ここでは最終的な設定の違いのみを示します。これからプロビジョニングするすべてのノードに対して、上記のチェックのいずれか、またはすべてを自由に適用してください。

12.5.7.2 2つ目のノードのプロビジョニング

仮想マシンを作成しましょう。

```
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
```

仮想マシンが稼働したら、このノードがボンディングされたインタフェースを使用しているかどうかを確認できます。

```
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
    altnam enp0s2
    altnam 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
    altnam enp0s3
    altnam 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
```

ルーティングでボンディングが使用されていることを確認します。

```
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
```

静的な接続ファイルが適切に利用されていることを確認します。

```
node2:~ # nmcli -f NAME,UUID,TYPE,DEVICE,FILENAME con show
NAME      UUID                                TYPE      DEVICE  FILENAME
bond99    4a920503-4862-5505-80fd-4738d07f44c6  bond      bond99  /etc/NetworkManager/
system-connections/bond99.nmconnection
```

```
eth0    dfd202f5-562f-5f07-8f2a-a7717756fb70  ethernet  eth0    /etc/NetworkManager/
system-connections/eth0.nmconnection
eth1    0523c0a1-5f5e-5603-bcf2-68155d5d322e  ethernet  eth1    /etc/NetworkManager/
system-connections/eth1.nmconnection
```

12.5.7.3 3つ目のノードのプロビジョニング

仮想マシンを作成しましょう。

```
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
```

仮想マシンが稼働したら、このノードがネットワークブリッジを使用していることを確認できます。

```
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
```

ルーティングでブリッジが使用されていることを確認します。

```
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
```

静的な接続ファイルが適切に利用されていることを確認します。

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

```
eth0      dfd202f5-562f-5f07-8f2a-a7717756fb70  ethernet  eth0      /etc/NetworkManager/  
system-connections/eth0.nmconnection
```

12.5.7.4 4つ目のノードのプロビジョニング

最後に、事前定義されたどの設定ともMACアドレスが一致しないノードをプロビジョニングします。このような場合は、DHCPをデフォルトにしてネットワークインタフェースを設定します。

仮想マシンを作成しましょう。

```
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
```

仮想マシンが稼働したら、このノードがそのネットワークインタフェースにランダムなIPアドレスを使用していることを確認できます。

```
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  
    altnam enp0s2  
    altnam 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
```

nmcがこのノードに静的な設定を適用できなかったことを確認します。

```
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
```

EthernetインタフェースがDHCPを介して設定されていることを確認します。

```
localhost:~ # journalctl | grep eth0  
Apr 23 12:15:29 localhost.localdomain NetworkManager[704]: <info> [1713874529.7801]  
manager: (eth0): new Ethernet device (/org/freedesktop/NetworkManager/Devices/2)
```



```

Apr 23 12:15:29 localhost.localdomain NetworkManager[704]: <info> [1713874529.7802]
device (eth0): state change: unmanaged -> unavailable (reason 'managed', sys-iface-state:
'external')
Apr 23 12:15:29 localhost.localdomain NetworkManager[704]: <info> [1713874529.7929]
device (eth0): carrier: link connected
Apr 23 12:15:29 localhost.localdomain NetworkManager[704]: <info> [1713874529.7931]
device (eth0): state change: unavailable -> disconnected (reason 'carrier-changed', sys-
iface-state: 'managed')
Apr 23 12:15:29 localhost.localdomain NetworkManager[704]: <info>
[1713874529.7944] device (eth0): Activation: starting connection 'Wired
Connection' (300ed658-08d4-4281-9f8c-d1b8882d29b9)
Apr 23 12:15:29 localhost.localdomain NetworkManager[704]: <info> [1713874529.7945]
device (eth0): state change: disconnected -> prepare (reason 'none', sys-iface-state:
'managed')
Apr 23 12:15:29 localhost.localdomain NetworkManager[704]: <info> [1713874529.7947]
device (eth0): state change: prepare -> config (reason 'none', sys-iface-state:
'managed')
Apr 23 12:15:29 localhost.localdomain NetworkManager[704]: <info> [1713874529.7953]
device (eth0): state change: config -> ip-config (reason 'none', sys-iface-state:
'managed')
Apr 23 12:15:29 localhost.localdomain NetworkManager[704]: <info> [1713874529.7964] dhcp4
(eth0): activation: beginning transaction (timeout in 90 seconds)
Apr 23 12:15:33 localhost.localdomain NetworkManager[704]: <info> [1713874533.1272] dhcp4
(eth0): state changed new lease, address=192.168.122.86

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

```

12.5.8 統合されたノード設定

既知のMACアドレスに依存できない場合もあります。このような場合は、いわゆる**統合設定**を選択できます。これにより、`_all.yaml`ファイルで設定を指定し、プロビジョニングされたノードすべてに適用することができます。

異なる設定構造を使用して、エッジノードを構築およびプロビジョニングします。12.5.3項「イメージ設定ディレクトリの作成」から12.5.5項「ネットワーク設定の定義」のすべての手順に従います。

この例では、2つのEthernetインタフェース(eth0とeth1)の望ましい状態を定義します。一方ではDHCPを使用し、他方には静的IPアドレスを割り当てます。

```

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

```

イメージを構築してみましょう。

```

podman run --rm -it -v $CONFIG_DIR:/eib registry.suse.com/edge/3.4/edge-image-builder:1.3.0 build --definition-file definition.yaml

```

イメージが正常に構築されたら、それを使用して仮想マシンを作成しましょう。

```

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

```

プロビジョニングプロセスには数分かかる場合があります。終了したら、指定された資格情報でシステムにログインします。

ルーティングが適切に設定されていることを確認します。

```

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

```

インターネット接続が利用できることを確認します。

```

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
```

Ethernetインタフェースが設定され、アクティブであることを確認します。

```
localhost:~ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
  1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
  default qlen 1000
    link/ether 52:54:00:26:44:7a brd ff:ff:ff:ff:ff:ff
    altname enp1s0
    inet 192.168.122.100/24 brd 192.168.122.255 scope global dynamic noprefixroute eth0
        valid_lft 3505sec preferred_lft 3505sec
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
  default qlen 1000
    link/ether 52:54:00:ec:57:9e brd ff:ff:ff:ff:ff:ff
    altname enp7s0
    inet 10.0.0.1/24 brd 10.0.0.255 scope global noprefixroute eth1
        valid_lft forever preferred_lft forever

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

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

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

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

localhost:~ # cat /etc/NetworkManager/system-connections/eth1.nmconnection
[connection]
autoconnect=true
autoconnect-slaves=-1
id=eth1
interface-name=eth1
type=802-3-ethernet
uuid=0523c0a1-5f5e-5603-bcf2-68155d5d322e

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

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

12.5.9 カスタムネットワーク設定

ここまでは、NetworkManager Configuratorを利用した、Edge Image Builderのデフォルトのネットワーク設定について説明してきました。一方で、カスタムスクリプトを使用してネットワーク設定を変更するオプションもあります。このオプションは非常に柔軟性が高く、MACアドレスにも依存しませんが、1つのイメージで複数のノードをブートストラップする場合に使用してもあまり便利ではないという制限があります。



注記

/networkディレクトリにある、望ましいネットワーク状態を記述したファイルを介して、デフォルトのネットワーク設定を使用することをお勧めします。カスタムスクリプトを選択するのは、デフォルト設定の動作がユースケースに当てはまらない場合のみにしてください。

異なる設定構造を使用して、エッジノードを構築およびプロビジョニングします。[12.5.3項「イメージ設定ディレクトリの作成」](#)から[12.5.5項「ネットワーク設定の定義」](#)のすべての手順に従います。

この例では、プロビジョニングされたすべてのノードでeth0インタフェースに静的設定を適用し、NetworkManagerによって自動的に作成された有線接続を削除して無効にするカスタムスクリプトを作成します。これは、クラスタ内のすべてのノードに同一のネットワーク設定を確実に適用したい場合に便利です。その結果、イメージの作成前に各ノードのMACアドレスを気にする必要がなくなります。

まず、`/custom/files`ディレクトリに接続ファイルを保存しましょう。

```
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
```

静的設定が作成されたので、カスタムネットワークスクリプトも作成します。

```
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
```



注記

nmcのバイナリはこれまで同様にデフォルトで含まれるため、必要に応じてconfigure-network.shスクリプトで使用することもできます。



警告

カスタムスクリプトは常に設定ディレクトリの/network/configure-network.shで提供する必要があります。このファイルが存在する場合、他のファイルはすべて無視されます。YAML形式の静的設定とカスタムスクリプトの両方を同時に使用してネットワークを設定することはできません。

この時点では、設定ディレクトリは次のようになっているはずです。

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

イメージを構築してみましょう。

```
podman run --rm -it -v $CONFIG_DIR:/eib registry.suse.com/edge/3.4/edge-image-builder:1.3.0 build --definition-file definition.yaml
```

イメージが正常に構築されたら、それを使用して仮想マシンを作成しましょう。

```
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
```

プロビジョニングプロセスには数分かかる場合があります。終了したら、指定された資格情報でシステムにログインします。

ルーティングが適切に設定されていることを確認します。

```
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
```

インターネット接続が利用できることを確認します。

```
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
```

接続ファイルを使用してEthernetインタフェースが静的に設定されていて、アクティブであることを確認します。

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

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

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

[ipv4]
dhcp-timeout=2147483647
method=auto

[ipv6]
```

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


13 Elemental

Elementalは、Kubernetesを使用した完全にクラウドネイティブな集中型のOS管理を可能にするソフトウェアスタックです。Elementalスタックは、Rancher自体またはエッジノード上に存在する多数のコンポーネントで構成されます。中核となるコンポーネントは次のとおりです。

- **elemental-operator** - Rancher上に存在し、クライアントからの登録リクエストを処理するコアオペレータ。
- **elemental-register** - エッジノード上で動作し、elemental-operatorを介して登録できるようにするクライアント。
- **elemental-system-agent** - エッジノードに存在するエージェント。その設定はelemental-registerから提供され、rancher-system-agentを設定するためのplanを受け取ります。
- **rancher-system-agent** - エッジノードが完全に登録された後に、elemental-system-agentから処理を引き継ぎ、Rancher Managerからの他のplansを待機します (Kubernetesのインストールなど)。

Elemental、およびElementalとRancherとの関係の詳細については、[Elementalのアップストリームドキュメント \(https://elemental.docs.rancher.com/\)](https://elemental.docs.rancher.com/) を参照してください。

13.1 SUSE EdgeでのElementalの用途

SUSEでは、Metal³を選択できないリモートデバイス(たとえば、BMCがない、デバイスがNATゲートウェイの背後にあるなど)の管理にElementalの一部を使用しています。このツールにより、オペレータは、デバイスがいつどこに配置されるかがわかる前に、ラボでデバイスをブートストラップできます。すなわち、elemental-registerとelemental-system-agentコンポーネントを利用して、「Phone Home」ネットワークプロビジョニングのユースケースでSUSE Linux MicroホストをRancherにオンボードできます。Edge Image Builder (EIB)を使用してデプロイメントイメージを作成する場合、EIBの設定ディレクトリで登録設定を指定することで、Rancherを使用してElemental経由で自動登録を行うことができます。



注記

SUSE Edge 3.4では、Elementalのオペレーティングシステム管理の側面を利用していないため、Rancher経由でオペレーティングシステムのパッチを管理することはできません。SUSE Edgeでは、Elementalツールを使用してデプロイメントイメージを構築する代わりに、登録設定を使用するEdge Image Builderツールを使用します。

13.2 ベストプラクティス

13.2.1 インストールメディア

「Phone Homeネットワークプロビジョニング」のデプロイメントフットプリントでElementalを利用してRancherに登録可能なデプロイメントイメージを構築する場合、SUSE Edgeでは、Elementalを使用したりリモートホストのオンボーディング(第2章「[Elementalを使用したりリモートホストのオンボーディング](#)」)のクイックスタートで詳しく説明されている手順に従う方法をお勧めします。

13.2.2 ラベル

Elementalは、MachineInventory CRDを使用してインベントリを追跡し、インベントリを選択する方法を提供します。たとえば、Kubernetesクラスタのデプロイ先のマシンをラベルに基づいて選択できます。これにより、ユーザはハードウェアを購入する前に、インフラストラクチャのニーズの(すべてではないにしても)ほとんどを事前に定義しておくことができます。また、ノードはその各インベントリオブジェクトのラベルを追加/削除できるので(elemental-registerを、追加のフラグ `--label "F00=BAR "`を指定して再実行する)、ノードがブートされた場所を検出してRancherに知らせるスクリプトを作成できます。

13.3 既知の問題

- 現在のところ、Elemental UIは、インストールメディアの構築方法を認識したり、「Elemental Teal」以外のオペレーティングシステムを更新したりすることはできません。これは将来のリリースで対応予定です。

14 K3s

K3s (<https://k3s.io/>) は、リソースに制約のあるリモートの無人の場所やIoTアプライアンス内の運用ワークロード向けに設計された、高可用性のKubernetes認定ディストリビューションです。

単一の小さなバイナリとしてパッケージ化されているため、迅速かつ簡単にインストールおよび更新できます。

14.1 SUSE EdgeでのK3sの用途

K3sは、SUSE Edgeスタックを支えるKubernetesディストリビューションとして使用できます。K3sはSUSE Linux Microオペレーティングシステムにインストールすることが意図されています。

K3sをSUSE EdgeスタックのKubernetesディストリビューションとして使用することは、etcdをバックエンドとして使用したのでは制約に合わない場合にのみ推奨します。etcdをバックエンドとして使用できる場合は、RKE2 (第15章 「RKE2」) を使用することをお勧めします。

14.2 ベストプラクティス

14.2.1 インストール

K3sをSUSE Edgeスタックの一部としてインストールする場合に推奨する方法は、Edge Image Builder (EIB)を使用することです。K3sをデプロイするようにEIBを設定する方法の詳細については、EIBのドキュメント(第11章 「Edge Image Builder」)を参照してください。この方法では、自動的にHAセットアップとElementalセットアップがサポートされます。

14.2.2 GitOpsワークフローでのFleet

SUSE Edgeスタックでは、GitOpsの推奨ツールとしてFleetを使用します。Fleetのインストールと使用の詳細については、このドキュメントの「Fleet」のセクション(第8章 「Fleet」)を参照してください。

14.2.3 ストレージ管理

K3sではローカルパスストレージが事前設定されており、これはシングルノードクラスタに適しています。複数のノードにまたがるクラスタの場合は、SUSE Storage (第16章「[SUSE Storage](#)」)を使用することをお勧めします。

14.2.4 負荷分散とHA

EIBを使用してK3sをインストールした場合、ここで説明する部分は、EIBのドキュメントの「HA」のセクションで説明済みです。

EIBを使用しないでK3sをインストールした場合は、MetalLBのドキュメント(第24章「[K3s上のMetalLB \(レイヤ2モードを使用\)](#)」)に従ってMetalLBをインストールおよび設定する必要があります。

15 RKE2

RKE2の公式ドキュメント (<https://docs.rke2.io/>) を参照してください。

RKE2は、以下によってセキュリティとコンプライアンスに重点を置いた、完全準拠のKubernetesディストリビューションです。

- クラスタがCIS Kubernetes Benchmark v1.6またはv1.23に合格できるデフォルト値と設定オプションを、オペレータの介入を最小限に抑えながら提供する
- FIPS 140-2準拠を可能にする
- [trivy](https://trivy.dev) (<https://trivy.dev>) を使用し、コンポーネントを定期的にスキャンしてRKE2ビルドパイプラインにCVEがないかどうかを確認する

RKE2は、コントロールプレーンコンポーネントを、kubeletによって管理される静的Podとして起動します。組み込みコンテナランタイムはcontainerdです。

メモ: RKE2はRKE Governmentとしても知られます。これは、RKE2が現在ターゲットにしている別のユースケースと分野を表すためです。

15.1 RKE2とK3s

K3sはエッジ、IoT、ARMに焦点を当てた、完全準拠の軽量なKubernetes ディストリビューションであり、使いやすさとリソースに制約のある環境向けに最適化されています。

RKE2は、RKEの1.xバージョン(以下「RKE1」)とK3sの両方の長所を兼ね備えています。

RKE2は、K3sから使いやすさ、操作のしやすさ、およびデプロイメントモデルを継承しています。

RKE1から継承しているのは、アップストリームのKubernetesとの緊密な連携です。K3sはエッジデプロイメントに合わせて最適化されているため、アップストリームのKubernetesとは各所で異なりますが、RKE1とRKE2はアップストリームと緊密な連携を保つことができます。

15.2 SUSE EdgeでのRKE2の用途

RKE2はSUSE Edgeスタックの基礎を成す部分です。RKE2はSUSE Linux Micro (第9章「[SUSE Linux Micro](#)」)上に位置し、Edgeワークロードをデプロイするために必要な標準Kubernetesインタフェースを提供します。

15.3 ベストプラクティス

15.3.1 インストール

RKE2をSUSE Edgeスタックの一部としてインストールする場合に推奨される方法は、Edge Image Builder (EIB)を使用することです。RKE2をデプロイするようにEIBを設定する方法の詳細については、EIBのドキュメント(第11章「Edge Image Builder」)を参照してください。

EIBは十分な柔軟性を備えているため、RKE2のバージョン、サーバ (https://docs.rke2.io/reference/server_config)、またはエージェント (https://docs.rke2.io/reference/linux_agent_config) 設定の指定など、RKE2で要求されるあらゆるパラメータをサポートすることができ、Edgeのすべてのユースケースに対応できます。

Metal³に関連する他のユースケースでも、RKE2が使用およびインストールされます。このような特定のケースでは、Cluster API Provider RKE2 (<https://github.com/rancher-sandbox/cluster-api-provider-rke2>) によって、Edgeスタックを使用してMetal³でプロビジョニングされるクラスタにRKE2が自動的にデプロイされます。

このような場合、関係する各種のCRDにRKE2設定を適用する必要があります。RKE2ControlPlane CRDを使用して異なるCNIを提供する方法の例は、次のようになります。

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

Metal³のユースケースの詳細については、第10章「Metal³」を参照してください。

15.3.2 高可用性

HAデプロイメントの場合、EIBはMetalLB (第18章「MetalLB」)とEndpoint Copier Operator (第19章「Endpoint Copier Operator」)を自動的にデプロイして設定し、RKE2 APIエンドポイントを外部に公開します。

15.3.3 ネットワーキング

SUSE Edgeスタックは、[Cilium](https://docs.cilium.io/en/stable/) (<https://docs.cilium.io/en/stable/>)、[Calico](https://docs.tigera.io/calico/latest/about/) (<https://docs.tigera.io/calico/latest/about/>)をサポートしており、CiliumをデフォルトのCNIとしています。Podが複数のネットワークインタフェースを必要とする場合には[Multus](https://github.com/k8snetworkplumbingwg/multus-cni) (<https://github.com/k8snetworkplumbingwg/multus-cni>) meta-pluginも使用できます。RKE2スタンドアロンは、より幅広いCNIオプション (https://docs.rke2.io/install/network_options)をサポートしています。

15.3.4 ストレージ

RKE2は、どのような種類の永続ストレージクラスやオペレータも提供していません。複数のノードにまたがるクラスタの場合は、SUSE Storage (第16章「[SUSE Storage](#)」)を使用することをお勧めします。

16 SUSE Storage

SUSE Storageは、Kubernetes向けに設計された、信頼性が高くユーザフレンドリな軽量の分散ブロックストレージシステムです。当初はRancher Labsによって開発され、現在はCNCFの下でインキュベートされているオープンソースプロジェクトであるLonghornをベースにした製品です。

16.1 前提条件

このガイドに従って操作を進める場合、以下がすでに用意されていることを想定しています。

- SUSE Linux Micro 6.1がインストールされた最低1台のホスト(物理ホストでも仮想ホストでも可)
- インストール済みのKubernetesクラスター1つ(K3sまたはRKE2)
- Helm

16.2 SUSE Storageの手動インストール

16.2.1 Open-iSCSIのインストール

SUSE Storageをデプロイして使用するための中心的な要件は、open-iscsiパッケージをインストールすることと、iscsidデーモンをすべてのKubernetesノード上で実行することです。これは、Longhornがホスト上のiscsiadmを利用してKubernetesに永続ボリュームを提供するために必要です。

インストールしてみましょう。

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

SUSE Linux Microはイミュータブルオペレーティングシステムであるため、操作が完了すると、パッケージは新しいスナップショットにのみインストールされることに注意することが重要です。パッケージをロードし、iscsidデーモンの実行を開始するには、作成した新しいスナップショットで再起動する必要があります。準備が整ったら、rebootコマンドを発行します。

```
reboot
```




ヒント

open-iscsiのインストールに関する追加のヘルプについては、Longhornの公式ドキュメント (<https://longhorn.io/docs/1.9.1/deploy/install/#installing-open-iscsi>) を参照してください。

16.2.2 SUSE Storageのインストール

KubernetesクラスタにSUSE Storageをインストールするには複数の方法があります。このガイドでは、Helmでのインストールに従いますが、別のアプローチが必要な場合は[公式ドキュメント \(https://longhorn.io/docs/1.9.1/deploy/install/\)](https://longhorn.io/docs/1.9.1/deploy/install/) に従ってください。

1. Rancher Charts Helm リポジトリを追加します。

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

2. リポジトリから最新のチャートをフェッチします。

```
helm repo update
```

3. `longhorn-system` ネームスペースにSUSE Storageをインストールします。

```
helm install longhorn-crd rancher-charts/longhorn-crd --namespace longhorn-system --create-namespace --version 107.0.0+up1.9.1
helm install longhorn rancher-charts/longhorn --namespace longhorn-system --version 107.0.0+up1.9.1
```

4. デプロイメントが成功したことを確認します。

```
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	

16.3 SUSE Storageボリュームの作成

SUSE Storageは、StorageClassというKubernetesリソースを利用して、PodのPersistentVolumeオブジェクトを自動的にプロビジョニングします。StorageClassは、管理者が、自身が提供する**クラス**または**プロファイル**を記述する方法だと考えてください。デフォルトのオプションをいくつか使用してStorageClassを作成しましょう。

```
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
```

StorageClassを作成したので、それを参照するPersistentVolumeClaimが必要です。PersistentVolumeClaim (PVC)は、ユーザによるストレージの要求です。PVCはPersistentVolumeリソースを使用します。クレームでは、特定のサイズとアクセスモードを要求できます(たとえば、1つのノードで読み取り/書き込み可能でマウントすることも、複数のノードで読み取り専用でマウントすることもできます)。

PersistentVolumeClaimを作成しましょう。

```
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
```

完了です。PersistentVolumeClaimを作成したら、それをPodにアタッチする手順に進むことができます。Podがデプロイされると、KubernetesはLonghornボリュームを作成し、ストレージが利用可能な場合はPodにバインドします。

```
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
```



ヒント

Kubernetesにおけるストレージの概念は複雑であると同時に重要なトピックです。最も一般的なKubernetesリソースのいくつかを簡単に説明しましたが、Longhornが提供している用語のドキュメント (<https://longhorn.io/docs/1.9.1/terminology/>) をよく理解しておくことをお勧めします。

この例では、結果は次のようになります。

```
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
```

16.4 UIへのアクセス

kubectlまたはHelmを使用してLonghornをインストールした場合は、クラスタへの外部トラフィックを許可するようにIngressコントローラを設定する必要があります。認証はデフォルトでは有効になっていません。Rancherカタログアプリを使用していた場合、IngressコントローラはRancherによって自動的に作成され、アクセス制御が設定されています(rancher-proxy)。

1. Longhornの外部サービスのIPアドレスを取得します。

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

2. longhorn-frontendのIPアドレスを取得したら、ブラウザでそのアドレスに移動してUIの使用を開始できます。

16.5 Edge Image Builderを使用したインストール

SUSE Edgeは、[第11章「Edge Image Builder」](#)を使用して、ベースとなるSUSE Linux Micro OSイメージをカスタマイズしています。ここでは、イメージをカスタマイズしてRKE2クラスタとLonghornをSUSE Linux Micro上にプロビジョニングする方法について説明します。

定義ファイルを作成しましょう。

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

cat << EOF > $CONFIG_DIR/iso-definition.yaml
apiVersion: 1.3
image:
  imageType: iso
  baseImage: SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso
  arch: x86_64
  outputImageName: eib-image.iso
kubernetes:
  version: v1.33.3+rke2r1
  helm:
    charts:
      - name: longhorn
        version: 107.0.0+up1.9.1
        repositoryName: longhorn
        targetNamespace: longhorn-system
```

```

    createNamespace: true
    installationNamespace: kube-system
  - name: longhorn-crd
    version: 107.0.0+up1.9.1
    repositoryName: longhorn
    targetNamespace: longhorn-system
    createNamespace: true
    installationNamespace: kube-system
repositories:
  - name: longhorn
    url: https://charts.rancher.io
operatingSystem:
  packages:
    sccRegistrationCode: <reg-code>
  packageList:
    - open-iscsi
  users:
    - username: root
      encryptedPassword: \$6\$jHugJNNd3HElGsUZ\
$eodjVe4te5ps44SVcWshdfWizrP.xAyd7lCVEXazBJ/.v799/WRCBXxfYmunlB02yp1hm/zb4r8EmnrrNCF.P/
EOF

```



注記

Helmチャートの値のカスタマイズは、`helm.charts[].valuesFile`で提供されている別個のファイルを使用して実行できます。詳細については、[アップストリームドキュメント \(https://github.com/suse-edge/edge-image-builder/blob/release-1.3/docs/building-images.md#kubernetes\)](https://github.com/suse-edge/edge-image-builder/blob/release-1.3/docs/building-images.md#kubernetes) を参照してください。

イメージを構築してみましょう。

```
podman run --rm --privileged -it -v $CONFIG_DIR:/eib registry.suse.com/edge/3.4/edge-image-builder:1.3.0 build --definition-file $CONFIG_DIR/iso-definition.yaml
```

イメージが構築されたら、それを使用して物理ホストまたは仮想ホストにOSをインストールできます。プロビジョニングが完了すると、`root:eib`の資格情報ペアを使用してシステムにログインできます。

Longhornが正常にデプロイされていることを確認します。

```
localhost:~ # /var/lib/rancher/rke2/bin/kubectl --kubeconfig /etc/rancher/rke2/rke2.yaml -n longhorn-system get pods
```

NAME	READY	STATUS	RESTARTS	AGE
csi-attacher-5c4bfdcf59-qmjtz	1/1	Running	0	103s
csi-attacher-5c4bfdcf59-s7n65	1/1	Running	0	103s
csi-attacher-5c4bfdcf59-w9xgs	1/1	Running	0	103s
csi-provisioner-667796df57-fmz2d	1/1	Running	0	103s

csi-provisioner-667796df57-p7rjr	1/1	Running	0	103s
csi-provisioner-667796df57-w9fdq	1/1	Running	0	103s
csi-resizer-694f8f5f64-2rb8v	1/1	Running	0	103s
csi-resizer-694f8f5f64-z9v9x	1/1	Running	0	103s
csi-resizer-694f8f5f64-zlncz	1/1	Running	0	103s
csi-snapshotter-959b69d4b-5dpvj	1/1	Running	0	103s
csi-snapshotter-959b69d4b-lwwkv	1/1	Running	0	103s
csi-snapshotter-959b69d4b-tzhwc	1/1	Running	0	103s
engine-image-ei-5cefaf2b-hvdv5	1/1	Running	0	109s
instance-manager-0ee452a2e9583753e35ad00602250c5b	1/1	Running	0	109s
longhorn-csi-plugin-gd2jx	3/3	Running	0	103s
longhorn-driver-deployer-9f4fc86-j6h2b	1/1	Running	0	
2m28s				
longhorn-manager-z4lnl	1/1	Running	0	
2m28s				
longhorn-ui-5f4b7bbf69-bln7h	1/1	Running	3 (2m7s ago)	
2m28s				
longhorn-ui-5f4b7bbf69-lh97n	1/1	Running	3 (2m10s ago)	
2m28s				



注記

このインストールは、完全なエアギャップ環境では動作しません。このような場合は、[27.8項「SUSE Storageのインストール」](#)を参照してください。

17 SUSE Security

SUSE SecurityはKubernetes向けのセキュリティソリューションであり、L7ネットワークセキュリティ、ランタイムセキュリティ、サプライチェーンセキュリティ、およびコンプライアンスチェックを1つの統合パッケージで提供します。

SUSE Securityは、複数のコンテナのプラットフォームとしてデプロイされ、各コンテナがさまざまなポートやインタフェースを介して通信する製品です。内部的には、その基礎となるコンテナセキュリティコンポーネントとしてNeuVectorを使用しています。SUSE Securityプラットフォームを構成するコンテナは次のとおりです。

- **Manager**。Webベースのコンソールを提供するステートレスコンテナです。通常、Managerは1つだけで必要で、どこでも実行できます。Managerにエラーが発生しても、ControllerやEnforcerの動作には影響しません。ただし、特定の通知(イベント)と最近の接続データはManagerによってメモリ内にキャッシュされているため、これらの表示には影響があります。
- **Controller**。SUSE Securityの「コントロールプレーン」は必ずHA設定でデプロイされるため、ノードのエラーで設定が失われることはありません。Controllerはどこでも実行できますが、その重要性から、顧客はほとんどの場合、「管理」ノード、マスタノード、またはインフラノードに配置することを選択します。
- **Enforcer**。このコンテナはDaemonSetとしてデプロイされるため、保護する各ノードに1つのEnforcerが存在します。通常はすべてのワーカーノードにデプロイされますが、スケジュールを有効にしてマスタノードやインフラノードにデプロイすることもできます。メモ: Enforcerがクラスターノードに存在しない状況で、そのノード上のPodから接続が行われた場合、その接続はSUSE Securityによって「unmanaged」ワークロードとしてラベル付けされます。
- **Scanner**。コントローラの指示に従って、ビルトインCVEデータベースを使用して脆弱性スキャンを実行します。複数のScannerをデプロイしてスキャン能力を拡張できます。Scannerはどこでも実行できますが、コントローラが実行されるノードで実行されることがほとんどです。Scannerノードのサイジングに関する考慮事項については、以下を参照してください。ビルドフェーズのスキャンに使用する場合、Scannerを独立して呼び出すこともできます。たとえば、スキャンをトリガし、結果を取得してScannerを停止するパイプライン内で使用する場合などです。Scannerには最新のCVEデータベースが含まれているため、毎日更新する必要があります。
- **Updater**。Updaterは、CVEデータベースの更新が必要な場合に、Kubernetes cronジョブを通じてScannerの更新をトリガします。必ず使用環境に合わせて設定してください。

SUSE Securityのオンボーディングの詳細とベストプラクティスのドキュメントについては、[こちら \(https://open-docs.neuvector.com/\)](https://open-docs.neuvector.com/)  をご覧ください。

17.1 SUSE EdgeでのSUSE Securityの用途

SUSE Edgeは、エッジデプロイメントの開始点として簡潔なSUSE Security設定を提供します。

17.2 重要なメモ

- Scanner コンテナには、スキャンするイメージをメモリに取り込んで解凍するのに十分なメモリが必要です。1GBを超えるイメージをスキャンするには、Scannerのメモリを、予想される最大イメージサイズをわずかに上回るサイズまで増やしてください。
- 保護モードでは、大量のネットワーク接続が予想されます。保護(インラインファイアウォールでのブロック)モードの場合、Enforcerでは、接続および想定されるペイロードを保持して検査(DLP)するためにCPUとメモリが必要です。メモリを増やし、1つのCPUコアをEnforcer専用にするすることで、十分なパケットフィルタリング容量を確保できます。

17.3 Edge Image Builderを使用したインストール

SUSE Edgeは、SUSE Linux Micro OSのゴールデンイメージをカスタマイズするために第11章「[Edge Image Builder](#)」を使用しています。EIBでプロビジョニングしたKubernetesクラスタ上にSUSE Securityをエアギャップインストールするには、27.7項「[SUSE Securityのインストール](#)」に従ってください。

18 MetalLB

MetalLBの公式ドキュメント (<https://metallb.universe.tf/>) を参照してください。

MetalLBは、標準のルーティングプロトコルを使用する、ベアメタル Kubernetes クラスタ用のロードバランサの実装です。

ベアメタル環境では、ネットワークロードバランサの設定がクラウドセットアップよりも著しく複雑になります。クラウド設定でのわかりやすいAPIコールとは異なり、ベアメタルでは、高可用性(HA)を管理したり、シングルノードのロードバランサに特有の潜在的な単一障害点(SPOF)に対処したりするために、専用のネットワークアプライアンス、またはロードバランサと仮想IP (VIP) 設定の組み合わせが必要になります。このような設定は自動化しにくく、コンポーネントが動的にスケールアップ/ダウンするKubernetesのデプロイメントでは課題となります。

MetalLBでは、こうした課題に対処するために、Kubernetesモデルを利用してLoadBalancerタイプのサービスを作成し、ベアメタルセットアップであってもクラウド環境であるかのように動作させます。

2つの異なるアプローチがあります。L2モード (<https://metallb.universe.tf/concepts/layer2/>) (ARPトリックを使用する) アプローチか、BGP (<https://metallb.universe.tf/concepts/bgp/>) を使用するアプローチです。主にL2では特別なネットワーク機器は必要ありませんが、一般的にはBGPのほうが優れています。これはユースケースによって異なります。

18.1 SUSE EdgeでのMetalLBの用途

SUSE Edgeは、MetalLBを主に次の3つの方法で使用します。

- ロードバランサソリューションとして: MetalLBは、ベアメタルマシン用のロードバランサソリューションとして機能します。
- HA K3s/RKE2セットアップの場合: MetalLBでは、仮想IPアドレスを使用してKubernetes APIを負荷分散できます。
- MetalLBがサービスIPへのルートを近隣ルータにアドバタイズするL3 BGPソリューションとして。



注記

APIを公開できるようにするため、Endpoint Copier Operator (第19章「[Endpoint Copier Operator](#)」)を使用して、[kubernetes](#)サービスから[kubernetes-vip](#) LoadBalancerサービスへのK8s APIエンドポイントの同期を維持します。

18.2 ベストプラクティス

L2モードでのMetalLBのインストールについては、第24章「[K3s上のMetalLB \(レイヤ2モードを使用\)](#)」で説明されており、L3モードの場合は第25章「[K3s上のMetalLB \(レイヤ3モードを使用\)](#)」で説明されています。

[kube-api-server](#)の前面にインストールして高可用性トポロジを実現する方法のガイドについては、第26章「[Kubernetes APIサーバの前面のMetalLB](#)」を参照してください。

18.3 既知の問題

- K3sには、[Klipper](#)というロードバランサソリューションが付属しています。MetalLBを使用するには、[Klipper](#)を無効にする必要があります。このためには、[K3sのドキュメント](#) (<https://docs.k3s.io/networking>) [🔗](#)で説明されているように、`--disable servicelb`オプションを指定してK3sサーバを起動します。

19 Endpoint Copier Operator

Endpoint Copier Operator (<https://github.com/suse-edge/endpoint-copier-operator>) は、Kubernetesサービスとエンドポイントのコピーを作成し、同期を維持することを目的としたKubernetesオペレータです。

19.1 SUSE EdgeでのEndpoint Copier Operatorの用途

SUSE Edgeでは、Endpoint Copier OperatorがK3s/RKE2クラスタの高可用性(HA)セットアップを実現するために重要な役割を果たします。これは、LoadBalancerタイプのkubernetes-vipサービスを作成することで達成され、そのエンドポイントがkubernetesエンドポイントと常に同期を維持するようにします。公開されているIPアドレスはクラスタに参加するために他のノードから使用されるため、MetalLB (第18章 「MetalLB」)は kubernetes-vipサービスを管理するために活用されます。

19.2 ベストプラクティス

Endpoint Copier Operatorの使用に関する包括的なドキュメントについては、[こちら](https://github.com/suse-edge/endpoint-copier-operator/blob/main/README.md) (<https://github.com/suse-edge/endpoint-copier-operator/blob/main/README.md>) をご覧ください。

また、Endpoint Copier OperatorとMetalLBを使用してK3s/RKE2HA セットアップを実行する方法については、SUSEのガイド(第24章 「K3s上のMetalLB (レイヤ2モードを使用)」)を参照してください。

19.3 既知の問題

現在、Endpoint Copier Operatorは1つのサービス/エンドポイントにのみ対応しています。今後、複数のサービス/エンドポイントをサポートする機能拡張が予定されています。

20 Edge Virtualization

このセクションでは、Edge Virtualizationを使用してエッジノードで仮想マシンを実行する方法について説明します。Edge Virtualizationは、軽量な仮想化ユースケース向けに設計されており、仮想化およびコンテナ化されたアプリケーションのデプロイメントと管理に共通のワークフローが利用されることが想定されています。

SUSE Edge Virtualizationでは、仮想マシンの実行方法として次の2つをサポートしています。

1. ホストレベルでlibvirt+qemu-kvmを介して仮想マシンを手動でデプロイする
(Kubernetesは関与しない)
2. KubeVirtオペレータをデプロイし、Kubernetesベースで仮想マシンを管理する

どちらのオプションも有効ですが、以下では2番目のオプションのみを説明しています。SUSE Linux Microで提供されている、すぐに使用できる標準の仮想化メカニズムを使用する場合は、[こちら \(https://documentation.suse.com/sles/15-SP6/html/SLES-all/chap-virtualization-introduction.html\)](https://documentation.suse.com/sles/15-SP6/html/SLES-all/chap-virtualization-introduction.html) で包括的なガイドを参照してください。このガイドは主にSUSE Linux Enterprise Server用に記載されていますが、概念はほぼ同じです。

このガイドではまず、事前にデプロイ済みのシステムに追加の仮想化コンポーネントをデプロイする方法について説明しますが、その後続くセクションでは、Edge Image Builderを使用してこの設定を最初のデプロイメントに組み込む方法を説明しています。基本手順を実行して環境を手動で設定する必要がない場合は、そちらのセクションに進んでください。

20.1 KubeVirtの概要

KubeVirtでは、仮想マシンと他のコンテナ化ワークロードを併せてKubernetesで管理できます。これを実現するために、Linux仮想化スタックのユーザスペース部分をコンテナ内で実行します。これにより、ホストシステムの要件が最小限に抑えられ、セットアップと管理が容易になります。

KubeVirtのアーキテクチャの詳細については、[アップストリームドキュメント \(https://kubevirt.io/user-guide/architecture/\)](https://kubevirt.io/user-guide/architecture/) を参照してください。

20.2 前提条件

このガイドに従って操作を進める場合、以下がすでに用意されていることを想定しています。

- SUSE Linux Micro 6.1がインストールされ、BIOSで仮想化拡張機能が有効になっている少なくとも1台の物理ホスト(詳細については、[こちら \(https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-virt-support.html#sec-kvm-requires-hardware\)](https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-virt-support.html#sec-kvm-requires-hardware) を参照してください)。
- ノード全体で、K3s/RKE2 Kubernetesクラスタがすでにデプロイされており、クラスタへのスーパーユーザアクセスを可能にする適切なkubeconfigが設定されている。
- ルートユーザへのアクセス – 以下の説明では、自身がルートユーザであり、`sudo`を使用して特権を昇格して**いない**ことを想定しています。
- Helm (<https://helm.sh/docs/intro/install/>) がローカルで利用可能で、適切なネットワーク接続を備えていて、Kubernetesクラスタに設定をプッシュし、必要なイメージをダウンロードできる。

20.3 Edge Virtualizationの手動インストール

このガイドでは、Kubernetesのデプロイメント手順については説明しませんが、SUSE Edgeに適したバージョンのK3s (<https://k3s.io/>) またはRKE2 (<https://docs.rke2.io/install/quickstart>) がインストールされていること、およびkubeconfigが適切に設定されていて標準の`kubectl`コマンドをスーパーユーザとして実行できることを想定しています。また、シングルノードクラスタを形成することを想定していますが、マルチノードのデプロイメントでも大きな違いはないと考えられます。

SUSE Edge Virtualizationは、3つの別個のHelmチャートを使用してデプロイします。具体的には次のとおりです。

- **KubeVirt:** 中心的な仮想化コンポーネント。つまり、Kubernetesが仮想マシンをデプロイおよび管理できるようにするために必要なKubernetes CRD、オペレータ、およびその他のコンポーネント。
- **KubeVirtダッシュボード拡張機能:** 仮想マシンの起動/停止やコンソールへのアクセスなど、基本的な仮想マシン管理を実行できるオプションのRancher UI拡張機能。
- **Containerized Data Importer (CDI):** KubeVirtの永続ストレージの統合を可能にする追加コンポーネント。仮想マシンが既存のKubernetesストレージバックエンドをデータ用に使用する機能を提供するだけでなく、ユーザが仮想マシンのデータボリュームのインポートまたはクローンの作成を行うことも可能にします。

これらの各Helmチャートは、現在使用しているSUSE Edgeのリリースに従ってバージョン管理されています。運用での使用/サポートされる使用のためには、SUSEレジストリにあるアーティファクトを使用してください。

まず、kubectlのアクセスが機能していることを確認します。

```
$ kubectl get nodes
```

次のような画面が表示されます。

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

これで、**KubeVirt**および**Containerized Data Importer (CDI)**のHelmチャートのインストールに進むことができます。

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

数分ですべてのKubeVirtおよびCDIコンポーネントがデプロイされるはずです。これを検証するには、kubevirt-systemおよびcdi-systemのネームスペース内にデプロイされたすべてのリソースを確認します。

KubeVirtリソースを確認します。

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

次のような画面が表示されます。

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	AGE	DESIRED	CURRENT	READY	UP-T0-DATE	AVAILABLE	NODE
SELECTOR	AGE						

daemonset.apps/virt-handler	1	1	1	1	1
kubernetes.io/os=linux	93s				
NAME	READY	UP-TO-DATE	AVAILABLE	AGE	
deployment.apps/virt-operator	2/2	2	2	2m24s	
deployment.apps/virt-controller	2/2	2	2	93s	
deployment.apps/virt-api	1/1	1	1	118s	
NAME		DESIRED	CURRENT	READY	AGE
replicaset.apps/virt-operator-5fbcf48d58		2	2	2	2m24s
replicaset.apps/virt-controller-5f84c69884		2	2	2	93s
replicaset.apps/virt-api-7dfc54cf95		1	1	1	118s
NAME	AGE	PHASE			
kubevirt.kubevirt.io/kubevirt	2m24s	Deployed			

CDIリソースを確認します。

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

次のような画面が表示されます。

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

VirtualMachineカスタムリソース定義(CRD)がデプロイされていることを確認するには、次のコマンドで検証できます。

```
$ kubectl explain virtualmachine
```

VirtualMachineオブジェクトの定義が出力され、次のような画面が表示されます。


```
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)
```

20.4 仮想マシンのデプロイ

KubeVirtとCDIがデプロイされたので、[openSUSE Tumbleweed \(https://get.opensuse.org/tumbleweed/\)](https://get.opensuse.org/tumbleweed/) に基づくシンプルな仮想マシンを定義してみましょう。この仮想マシンは最もシンプルな設定であり、標準の「Podネットワーキング」を使用して、他のPodと同じネットワーキング設定を行います。また、非永続ストレージを使用するため、[PVC \(https://kubernetes.io/docs/concepts/storage/persistent-volumes/\)](https://kubernetes.io/docs/concepts/storage/persistent-volumes/) を持たないコンテナと同様に、ストレージは一時的なものになります。

```
$ 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:
I2Nsb3VklWNvbmZpZwpkaXNhYmxlX3Jvb3Q6IGZhbHNlCnNzaF9wd2F1dGg6IFRydWUKdXNlcnM6CiAgLSBkZWZhdWx0CiAgLSBuYW1
          name: cloudinitdisk
EOF
```

これにより、`VirtualMachine`が作成されたことを示すメッセージが出力されます。

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

この`VirtualMachine`定義は最小限であり、設定はほとんど指定されていません。この定義は単に、この仮想マシンが、一時的な`containerDisk`に基づくディスクイメージ(つまり、リモートイメージリポジトリからのコンテナイメージに保存されるディスクイメージ)を使用する、2GBのメモリを備えたマシンタイプ「[q35 \(https://wiki.qemu.org/Features/Q35\)](https://wiki.qemu.org/Features/Q35)」であることを示しています。また、base64でエンコードされたcloudInitディスクを指定しており、このディスクはブート時にユーザを作成してパスワードを適用する目的にのみ使用します(デコードには`base64 -d`を使用します)。



注記

この仮想マシンイメージはテスト専用です。このイメージは公式にサポートされておらず、ドキュメントの例としてのみ使用されています。

このマシンは、openSUSE Tumbleweedのディスクイメージをダウンロードする必要があるためブートに数分かかりますが、ブートが完了したら、次のコマンドで仮想マシンの情報をチェックして、仮想マシンの詳細を確認できます。

```
$ kubectl get vmi
```

これにより、仮想マシンが起動されたノードと、仮想マシンのIPアドレスが出力されます。Podネットワークングを使用しているため、報告されるIPアドレスは他のPodと同様であり、ルーティング可能であることに注意してください。

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

これらのコマンドをKubernetesクラスターノード自体で実行する場合は、トラフィックをPodに直接ルーティングするCNI (Ciliumなど)を使用して、マシン自体に直接sshで接続できるはずです。次のIPアドレスを、仮想マシンに割り当てられているIPアドレスに置き換えます。

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

この仮想マシンに接続すると、さまざまな操作を試すことができますが、リソースの点で制限があり、ディスク容量は1GBしかないことに注意してください。終了したら、`Ctrl-D`または`exit`でSSHセッションを切断します。

仮想マシンプロセスは、依然として標準のKubernetes Podでラップされています。VirtualMachine CRDは目的の仮想マシンを表していますが、仮想マシンが実際に起動されるプロセスは、他のアプリケーションと同様に、標準のKubernetes Podである`virt-launcher` Podを介して行われます。起動されたすべての仮想マシンに対して、`virt-launcher` Podが存在することがわかります。

```
$ kubectl get pods
```

次に、定義したTumbleweedマシンの1つの`virt-launcher` Podが表示されます。

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

この`virt-launcher` Podを調べてみると、`libvirt`プロセスと`qemu-kvm`プロセスを実行していることがわかります。このPod自体を起動して詳細を確認できます。次のコマンドは、使用しているPodの名前に合わせて調整する必要があることに注意してください。

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

Podが起動したら、`virsh`コマンドを実行するのと併せて、プロセスを確認してみます。`qemu-system-x86_64`バイナリに加え、仮想マシンを監視するための特定のプロセスも実行されていることがわかります。また、ディスクイメージの場所と、ネットワーキングが(タップデバイスとして)どのように接続されているかもわかります。

```
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

```

最後に、この仮称マシンを削除して、クリーンアップしましょう。

```

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

```

20.5 virtctlの使用

KubeVirtには、標準のKubernetes CLIツールであるkubectlとともに、仮想化の世界とKubernetesが設計された世界との間のギャップを埋める方法でクラスタとのインタフェースを可能にするCLIユーティリティが付属しています。たとえば、virtctlツールは、APIやCRDを直接使用することなく、仮想マシンのライフサイクル(起動、停止、再起動など)の管理、仮想コンソールへのアクセスの提供、仮想マシンイメージのアップロード、サービスなどのKubernetesコンストラクトとのインタフェースを行う機能を提供します。

virtctlツールの最新の安定バージョンをダウンロードしましょう。

```

$ export VERSION=v1.5.2
$ wget https://github.com/kubevirt/kubevirt/releases/download/$VERSION/virtctl-$VERSION-linux-amd64

```

別のアーキテクチャまたはLinux以外のマシンを使用している場合は、他のリリースを[こちら](https://github.com/kubevirt/kubevirt/releases) (https://github.com/kubevirt/kubevirt/releases) で見つけることができます。続行する前に、この実行可能ファイルを作成する必要があります。また、実行可能ファイルを\$PATH内の特定の場所に移動すると便利な場合があります。

```

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

```

その後、virtctlコマンドラインツールを使用して、仮想マシンを作成できます。出力をkubectl applyに直接パイプしていることに注意して、前の仮想マシンを複製してみましょう。

```
$ virtctl create vm --name virtctl-example --memory=1Gi \
  --volume-containerdisk=src:registry.opensuse.org/home/roxenham/tumbleweed-container-
disk/containerfile/cloud-image:latest \
  --cloud-init-user-data
  "I2Nsb3VkLWNvbmZpZwpkaXNhYmxlX3Jvb3Q6IGZhbHNlCnNzaF9wd2F1dGg6IFRydWUKdXNlcnM6CiAgLSBkZWZhdWx0CiAgLSBuY
| kubectl apply -f -
```

これで、仮想マシンが実行されているのがわかります(コンテナイメージがキャッシュされるため、今回はかなり早く起動するはずです)。

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

これで、virtctlを使用して仮想マシンに直接接続できるようになりました。

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

virtctlで使用可能なコマンドはほかにも多数あります。たとえば、virtctl consoleを使用すると、ネットワーキングが機能していない場合にシリアルコンソールにアクセスでき、virtctl guestosinfoを使用すると、ゲストにqemu-guest-agentがインストールされていて実行されていれば、包括的なOS情報を取得できます。

最後に、仮想マシンを一時停止し、再開してみましょう。

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

VirtualMachineオブジェクトが「**Paused**」と表示され、VirtualMachineInstanceオブジェクトは「**Running**」ですが「**READY=False**」と表示されているのがわかります。

```
$ 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
```

また、仮想マシンに接続できなくなっていることもわかります。

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

仮想マシンを再開して、もう一度試してみましょう。

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

これで、接続を再確立できるはずです。

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

最後に、仮想マシンを削除しましょう。

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

20.6 シンプルなIngressネットワークング

このセクションでは、仮想マシンを標準のKubernetesサービスとして公開し、NGINXとRKE2 (https://docs.rke2.io/networking/networking_services#nginx-ingress-controller)、TraefikとK3s (<https://docs.k3s.io/networking/networking-services#traefik-ingress-controller>)などのKubernetes Ingressサービスを介して利用可能にする方法を示します。このドキュメントでは、これらのコンポーネントがすでに適切に設定されていること、およびKubernetesサーバードまたはIngress仮想IPを指す適切なDNSポインタが設定されていて(ワイルドカードを使用するなど)、Ingressを適切に解決できることを前提としています。



注記

SUSE Edge 3.1以降では、マルチサーバード設定でK3sを使用している場合、Ingress用にMetalLBベースのVIPを設定する必要がある場合があります。RKE2では、これは不要です。

この例の環境では、別のopenSUSE Tumbleweed仮想マシンをデプロイし、cloud-initを使用して、ブート時にNGINXをシンプルなWebサーバとしてインストールしています。また、呼び出しの実行時に期待どおりに動作することを確認するためにシンプルなメッセージを返すように設定しています。この処理を確認するには、以下の出力のcloud-initのセクションに対して`base64 -d`を実行するだけです。

では、この仮想マシンを作成してみましょう。

```
$ kubectl apply -f - <<EOF
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: ingress-example
  namespace: default
spec:
  runStrategy: Always
  template:
```

```

metadata:
  labels:
    app: nginx
spec:
  domain:
    devices: {}
    machine:
      type: q35
    memory:
      guest: 2Gi
    resources: {}
  volumes:
  - containerDisk:
      image: registry.opensuse.org/home/roxenham/tumbleweed-container-disk/
containerfile/cloud-image:latest
      name: tumbleweed-containerdisk-0
  - cloudInitNoCloud:
      userDataBase64:
I2Nsb3VklWNvbmZpZwpkaXNhYmxlX3Jvb3Q6IGZhbHNlCnNzaF9wd2F1dGg6IFRydWUKdXNlcnM6CiAgLSBkZWZhdWx0CiAgLSBuYW
name: cloudinitdisk
EOF

```

この仮想マシンが正常に起動したら、`virtctl`コマンドを使用して、外部ポート8080とターゲットポート80 (NGINXがデフォルトでリスンするポート)でVirtualMachineInstanceを公開できます。`virtctl`コマンドは、仮想マシンオブジェクトとPodのマッピングを理解しているため、ここではこのコマンドを使用します。これにより、新しいサービスが作成されます。

```

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

```

これで、適切なサービスが自動的に作成されます。

```

$ 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

次に、`kubectl create ingress`を使用すると、このサービスを指すIngressオブジェクトを作成できます。ここでURL ([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/) オブジェクトの「ホスト」として知られている)をDNS設定に合わせて調整し、ポート8080を指すようにします。

```

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

```

DNSが正しく設定されたら、URLに対してすぐにcurlを実行できます。

```

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

```

この仮想マシンとそのサービス、およびIngressリソースを削除して、クリーンアップしましょう。

```
$ 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
```

20.7 Rancher UI拡張機能の使用

SUSE Edge VirtualizationはRancher Manager用のUI拡張機能を提供しており、Rancher Dashboard UIを使用して基本的な仮想マシン管理を行うことができます。

20.7.1 インストール

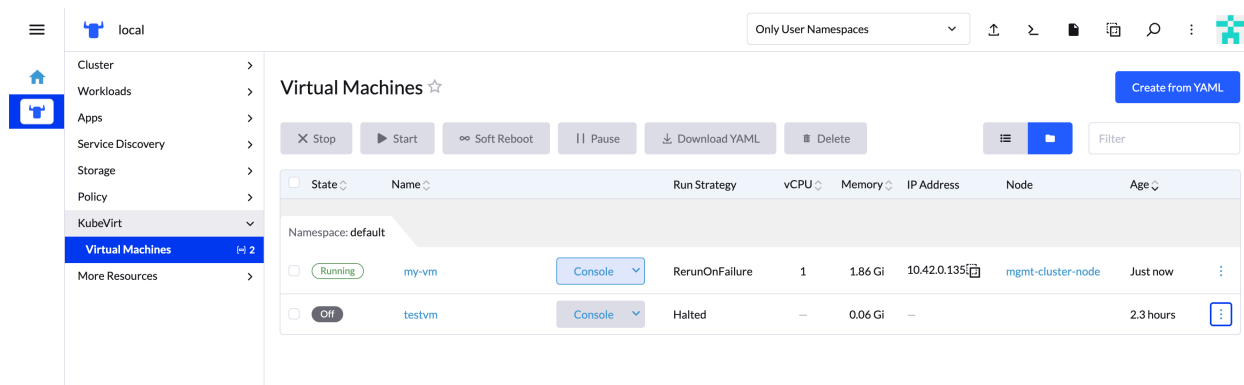
インストールのガイダンスについては、Rancher Dashboard拡張機能([第6章「Rancher Dashboard拡張機能」](#))を参照してください。

20.7.2 KubeVirt Rancher Dashboard拡張機能の使用

この拡張機能により、Cluster Explorerに新たに **[KubeVirt]** セクションが導入されます。このセクションは、KubeVirtがインストールされている管理対象クラスタに追加されます。この拡張機能を使用すると、KubeVirt仮想マシンリソースを直接操作し、仮想マシンのライフサイクルを管理できます。

20.7.2.1 仮想マシンの作成

1. 左側のナビゲーションでKubeVirtが有効な管理対象クラスタをクリックして、**[Cluster Explorer]** に移動します。
2. **[KubeVirt]** > [Virtual Machines (仮想マシン)] ページで、画面の右上にある **[Create from YAML (YAMLから作成)]** をクリックします。
3. 仮想マシンの定義を入力するか貼り付けて、**[Create (作成)]** を押します。「仮想マシンのデプロイ」セクションの仮想マシンの定義を参考にしてください。



20.7.2.2 仮想マシンのアクション

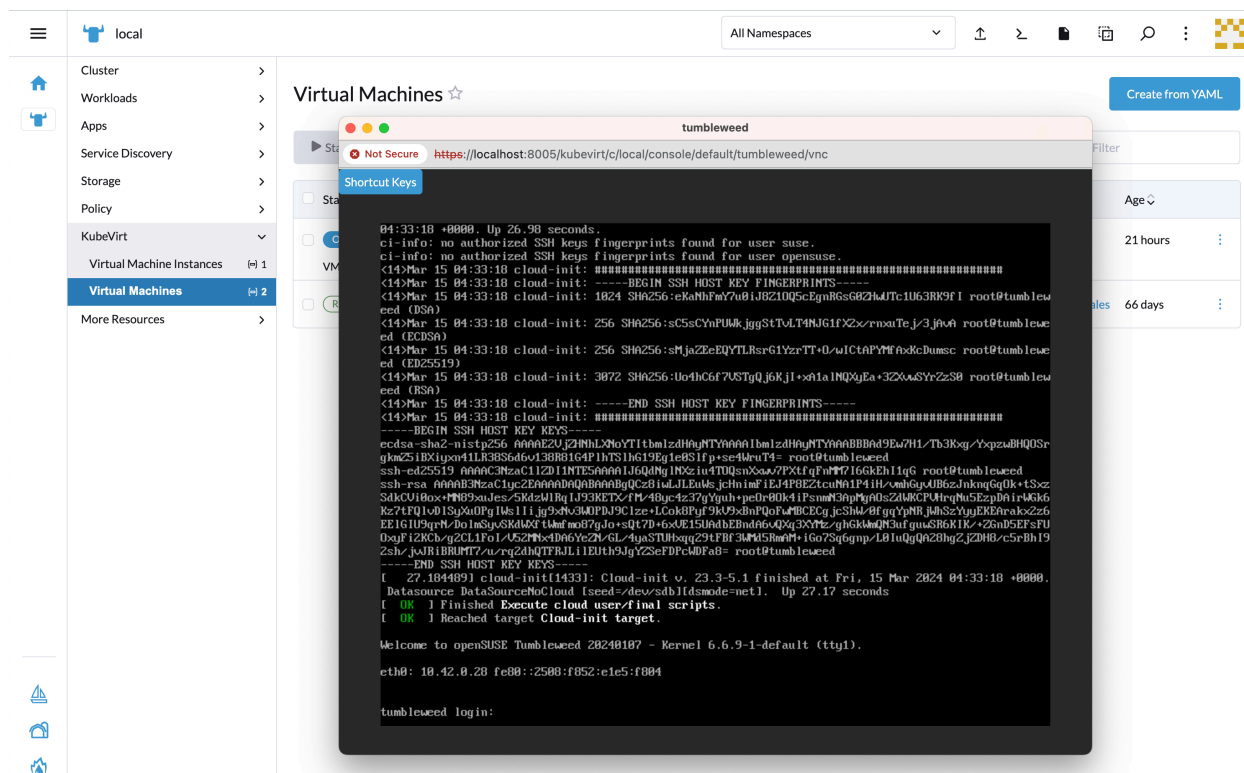
各仮想マシンの右側にある # ドロップダウンリストからアクセスできるアクションメニューを使用して、起動、停止、一時停止、またはソフト再起動のアクションを実行できます。または、アクションを実行する仮想マシンを選択して、リストの一番上にあるグループアクションを使用することもできます。

アクションを実行すると、仮想マシン実行戦略に影響する場合があります。詳細については [KubeVirtドキュメントの表 \(https://kubevirt.io/user-guide/compute/run_strategies/#virtctl\)](https://kubevirt.io/user-guide/compute/run_strategies/#virtctl) を参照してください。

20.7.2.3 仮想マシンコンソールへのアクセス

[Virtual Machines (仮想マシン)] リストには [Console (コンソール)] ドロップダウンリストがあり、ここから **VNC** または **シリアルコンソール** を使用してマシンに接続できます。このアクションは、実行中のマシンでのみ使用できます。

新しく起動した仮想マシンでは、コンソールにアクセスできるようになるまでにしばらく時間がかかることがあります。



20.8 Edge Image Builderを使用したインストール

SUSE Edgeは、SUSE Linux Micro OSのゴールデンイメージをカスタマイズするために第11章「Edge Image Builder」を使用しています。EIBでプロビジョニングしたKubernetesクラスタ上にKubeVirtとCDIの両方をエアギャップインストールするには、27.9項「KubeVirtとCDIのインストール」に従ってください。

21 System Upgrade Controller

System Upgrade Controllerのドキュメント (<https://github.com/rancher/system-upgrade-controller>) を参照してください。

System Upgrade Controller (SUC)は、汎用のKubernetesネイティブアップグレードコントローラ(ノード用)を提供することを目的としています。あらゆるアップグレードポリシー/要件を定義するための新しいCRDであるPlanが導入されています。Planは、クラスタ内のノードを変更する明確な意図です。

21.1 SUSE EdgeでのSystem Upgrade Controllerの用途

SUSE Edgeでは、SUCを使用して、管理クラスタとダウンストリームクラスタでのOSとKubernetesのバージョンアップグレードに関連するさまざまな「Day 2」操作を容易にします。

「Day 2」操作は、SUC Planによって定義されます。これらのプランに基づき、SUCは各ノードにワークロードをデプロイし、それぞれの「Day 2」操作を実行します。

SUCは、[第22章「Upgrade Controller」](#)内でも使用されます。SUCとUpgrade Controllerの主な違いに関する詳細については、[22.2項「Upgrade ControllerとSystem Upgrade Controller」](#)を参照してください。

21.2 System Upgrade Controllerのインストール

！ 重要

Rancher [v2.10.0](https://github.com/rancher/rancher/releases/tag/v2.10.0) (<https://github.com/rancher/rancher/releases/tag/v2.10.0>) 以降、System Upgrade Controllerは自動的にインストールされます。

ご使用の環境がRancherによって管理されていない場合、またはRancherのバージョンが[v2.10.0](#)より前の場合にのみ、以下の手順に従ってください。

[suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples) (<https://github.com/suse-edge/fleet-examples>) リポジトリにあるFleet ([第8章「Fleet」](#)) からSUCをインストールすることをお勧めします。



注記

`suse-edge/fleet-examples` リポジトリで提供されるリソースは常に有効な `fleet-examples release` (<https://github.com/suse-edge/fleet-examples/releases>) から使用される**必要があります**。使用する必要のあるリリースを確認するには、リリースノート (52.1項 「要約」) を参照してください。

SUCのインストールにFleetを使用できない場合は、RancherのHelmチャートリポジトリからインストールするか、独自のサードパーティGitOpsワークフローにRancherのHelmチャートを組み込むことができます。

このセクションでは以下の内容を取り上げます。

- Fleetのインストール(21.2.1項 「System Upgrade Controller Fleetのインストール」)
- Helmのインストール(21.2.2項 「System Upgrade Controller Helmのインストール」)

21.2.1 System Upgrade Controller Fleetのインストール

Fleetを使用して、SUCをデプロイするために使用できるリソースは2つあります。

- `GitRepo` (<https://fleet.rancher.io/ref-gitrepo>) リソース - 外部/ローカルGitサーバが利用できるユースケース用。インストール手順については、「System Upgrade Controllerのインストール - GitRepo」(21.2.1.1項 「System Upgrade Controllerのインストール - GitRepo」) を参照してください。
- `バンドル` (<https://fleet.rancher.io/bundle-add>) リソース - ローカルGitサーバオプションをサポートしないエアギャップ環境のユースケース用。インストール手順については、「System Upgrade Controllerのインストール - バンドル」(21.2.1.2項 「System Upgrade Controllerのインストール - バンドル」) を参照してください。

21.2.1.1 System Upgrade Controllerのインストール - GitRepo



注記

このプロセスは、使用できる場合はRancher UIから実行することもできます。詳細については、「[Rancher UIでのFleetへのアクセス \(https://ranchermanager.docs.rancher.com/v2.12/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui\)](https://ranchermanager.docs.rancher.com/v2.12/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui)」を参照してください。

管理クラスタで、次の操作を実行します。

1. SUCをデプロイするクラスタを決定します。これは、**管理**クラスタ内の適切なFleetワークスペースにSUC `GitRepo` リソースをデプロイすることで実行されます。デフォルトでは、Fleetには2つのワークスペースがあります。

- `fleet-local` - **管理**クラスタにデプロイする必要があるリソース用。
- `fleet-default` - **ダウンストリーム**クラスタにデプロイする必要があるリソース用。
Fleetワークスペースの詳細については、[アップストリーム \(https://fleet.rancher.io/namespaces#gitrepos-bundles-clusters-clustergroups\)](https://fleet.rancher.io/namespaces#gitrepos-bundles-clusters-clustergroups)  ドキュメントを参照してください。

2. `GitRepo` リソースをデプロイします。

- 管理クラスタにSUCをデプロイするには:

```
kubectl apply -n fleet-local -f - <<EOF
apiVersion: fleet.cattle.io/v1alpha1
kind: GitRepo
metadata:
  name: system-upgrade-controller
spec:
  revision: release-3.4.0
  paths:
    - fleets/day2/system-upgrade-controller
  repo: https://github.com/suse-edge/fleet-examples.git
EOF
```

- ダウンストリームクラスタにSUCをデプロイするには:



注記

以下のリソースをデプロイする前に、有効なターゲット設定を提供する**必要があります**。Fleetがリソースをデプロイするダウンストリームクラスタを認識できるようにするためです。ダウンストリームクラスタへのマッピング方法については、「[Mapping to Downstream Clusters \(ダウンストリームクラスタへのマッピング\) \(https://fleet.rancher.io/gitrepo-targets\)](https://fleet.rancher.io/gitrepo-targets) 

```
kubectl apply -n fleet-default -f - <<EOF
apiVersion: fleet.cattle.io/v1alpha1
```

```
kind: GitRepo
metadata:
  name: system-upgrade-controller
spec:
  revision: release-3.4.0
  paths:
  - fleets/day2/system-upgrade-controller
  repo: https://github.com/suse-edge/fleet-examples.git
  targets:
  - clusterSelector: CHANGEME
  # Example matching all clusters:
  # targets:
  # - clusterSelector: {}
EOF
```

3. GitRepoリソースがデプロイされていることを検証します。

```
# Namespace will vary based on where you want to deploy SUC
kubectl get gitrepo system-upgrade-controller -n <fleet-local/fleet-default>
```

NAME	BUNDLEDEPLOYMENTS-READY	REPO	STATUS	COMMIT
system-upgrade-controller	release-3.4.0 1/1	https://github.com/suse-edge/fleet-examples.git		

4. System Upgrade Controllerのデプロイメントを検証します。

```
kubectl get deployment system-upgrade-controller -n cattle-system
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
system-upgrade-controller	1/1	1	1	2m20s

21.2.1.2 System Upgrade Controllerのインストール - バンドル

このセクションでは、[fleet-cli \(https://fleet.rancher.io/cli/fleet-cli/fleet\)](https://fleet.rancher.io/cli/fleet-cli/fleet) を使用して、標準のFleet設定からバンドルリソースをビルドしてデプロイする方法を説明します。

1. ネットワークにアクセスできるマシンで、fleet-cliをダウンロードします。



注記

ダウンロードしたfleet-cliのバージョンが、クラスタにデプロイされているFleetのバージョンと一致していることを確認してください。

- Macユーザの場合、[fleet-cli \(https://formulae.brew.sh/formula/fleet-cli\)](https://formulae.brew.sh/formula/fleet-cli)  Homebrew Formulaeがあります。
- LinuxおよびWindowsユーザの場合、Fleetリリース (<https://github.com/rancher/fleet/releases>)  ごとに**アセット**としてバイナリが存在します。

- Linux AMD:

```
curl -L -o fleet-cli https://github.com/rancher/fleet/releases/download/v0.13.1/fleet-linux-amd64
```

- Linux ARM:

```
curl -L -o fleet-cli https://github.com/rancher/fleet/releases/download/v0.13.1/fleet-linux-arm64
```

2. [fleet-cli](#)を実行可能にします。

```
chmod +x fleet-cli
```

3. 使用する[suse-edge/fleet-examples](#) リリース (<https://github.com/suse-edge/fleet-examples/releases>) のクローンを作成します。

```
git clone -b release-3.4.0 https://github.com/suse-edge/fleet-examples.git
```

4. [fleet-examples](#)リポジトリにある、SUC Fleetに移動します。

```
cd fleet-examples/fleets/day2/system-upgrade-controller
```

5. SUCをデプロイするクラスタを決定します。これは、管理クラスタ内の適切な FleetワークスペースにSUCバンドルをデプロイすることで実行されます。デフォルトでは、Fleetには2つのワークスペースがあります。

- [fleet-local](#) - **管理**クラスタにデプロイする必要があるリソース用。
- [fleet-default](#) - **ダウンストリーム**クラスタにデプロイする必要があるリソース用。
Fleetワークスペースの詳細については、[アップストリーム \(https://fleet.rancher.io/namespaces#gitrepos-bundles-clusters-clustergroups\)](https://fleet.rancher.io/namespaces#gitrepos-bundles-clusters-clustergroups)  ドキュメントを参照してください。

6. ダウンストリームクラスタにのみSUCをデプロイする場合は、特定のクラスタに一致する[targets.yaml](#)ファイルを作成します。

```
cat > targets.yaml <<EOF
targets:
- clusterSelector: CHANGEME
EOF
```

ダウストリームクラスタへのマッピング方法については、「[Mapping to Downstream Clusters \(ダウストリームクラスタへのマッピング\)](https://fleet.rancher.io/gitrepo-targets) (<https://fleet.rancher.io/gitrepo-targets>) 

」を参照してください。

7. バンドルの構築に進みます。



注記

`fleet-examples/fleets/day2/system-upgrade-controller`ディレクトリの `fleet-cli` をダウンロード **していない** ことを確認してください。これをダウンロードすると、バンドルでパッケージ化され、これは推奨されません。

- 管理クラスタにSUCをデプロイするには、次のコマンドを実行します。

```
fleet-cli apply --compress -n fleet-local -o - system-upgrade-controller . >
system-upgrade-controller-bundle.yaml
```

- ダウストリームクラスタにSUCをデプロイするには、次のコマンドを実行します。

```
fleet-cli apply --compress --targets-file=targets.yaml -n fleet-default -o -
system-upgrade-controller . > system-upgrade-controller-bundle.yaml
```

このプロセスの詳細については、「[Convert a Helm Chart into a Bundle \(Helm チャートをバンドルに変換する\)](https://fleet.rancher.io/bundle-add#convert-a-helm-chart-into-a-bundle) (<https://fleet.rancher.io/bundle-add#convert-a-helm-chart-into-a-bundle>) 

」を参照してください。

`fleet-cli apply` コマンドの詳細については、「[fleet apply](https://fleet.rancher.io/cli/fleet-cli/fleet_apply) (https://fleet.rancher.io/cli/fleet-cli/fleet_apply) 

」を参照してください。

8. `system-upgrade-controller-bundle.yaml` バンドルを管理クラスタマシンに転送します。

```
scp system-upgrade-controller-bundle.yaml <machine-address>:<filesystem-path>
```

9. 管理クラスタで、`system-upgrade-controller-bundle.yaml` バンドルをデプロイします。


```
kubectl apply -f system-upgrade-controller-bundle.yaml
```

10. 管理クラスタで、バンドルがデプロイされていることを検証します。

```
# Namespace will vary based on where you want to deploy SUC
kubectl get bundle system-upgrade-controller -n <fleet-local/fleet-default>
```

NAME	BUNDLEDEPLOYMENTS-READY	STATUS
system-upgrade-controller	1/1	

11. バンドルをデプロイしたFleetワークスペースに基づいて、クラスタに移動し、SUCデプロイメントを検証します。



注記

SUCは常に、**cattle-system**ネームスペースにデプロイされます。

```
kubectl get deployment system-upgrade-controller -n cattle-system
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
system-upgrade-controller	1/1	1	1	111s

21.2.2 System Upgrade Controller Helmのインストール

1. Rancherチャートリポジトリを追加します。

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

2. SUCチャートをデプロイします。

```
helm install system-upgrade-controller rancher-charts/system-upgrade-controller --
version 107.0.0 --set global.cattle.psp.enabled=false -n cattle-system --create-
namespace
```

これにより、Edge 3.4プラットフォームに必要なSUCバージョン0.16.0がインストールされます。

3. SUCデプロイメントを検証します。

```
kubectl get deployment system-upgrade-controller -n cattle-system
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
system-upgrade-controller	1/1	1	1	37s

21.3 System Upgrade Controller Planのモニタリング

SUC Planは次の方法で確認できます。

- Rancher UI (21.3.1項「System Upgrade Controller Planのモニタリング - Rancher UI」) を通じて
- クラスタ内の手動モニタリング(21.3.2項「System Upgrade Controller Planのモニタリング - 手動」)を通じて



重要

SUC Plan用にデプロイされたPodは正常実行後、**15分間**維持されます。その後、作成元の対応するJobによって削除されます。この時間後もPodのログにアクセスできるようにするには、クラスタのログ記録を有効にする必要があります。Rancherでこれを実行する方法については、「[Rancher Integration with Logging Services \(Rancherとログ記録サービスの統合\)](https://ranchermanager.docs.rancher.com/v2.12/integrations-in-rancher/logging) (<https://ranchermanager.docs.rancher.com/v2.12/integrations-in-rancher/logging>)」を参照してください。

21.3.1 System Upgrade Controller Planのモニタリング - Rancher UI

特定のSUC PlanのPodログを確認するには、次の手順を実行します。

1. 左上隅で、**[#]** → **[<クラスタ名>]** を選択します。
2. **[Workloads (ワークロード)]** → **[Pods]** を選択します。
3. **[Only User Namespaces (ユーザネームスペースのみ)]** ドロップダウンメニューを選択し、**[cattle-system]** ネームスペースを追加します。
4. Podフィルタバーに、SUC Plan Podの名前を入力します。名前は次のテンプレート形式になります: apply-<plan_name>-on-<node_name>。



注記

特定のSUC Planに対して [Completed (完了)] Podと [Unknown (不明)] Podの両方が存在する場合があります。これは予期されており、一部のアップグレードの性質により発生します。

5. ログを確認するPodを選択し、 [#] → [View Logs (ログの表示)] に移動します。

21.3.2 System Upgrade Controller Planのモニタリング - 手動



注記

以下の手順は、`kubectl`が、**SUC Plan**がデプロイされたクラスタに接続するように設定されていることを前提としています。

1. デプロイした**SUC Plan**を一覧にします。

```
kubectl get plans -n cattle-system
```

2. **SUC Plan**用Podを入手します。

```
kubectl get pods -l upgrade.cattle.io/plan=<plan_name> -n cattle-system
```



注記

特定のSUC Planに対して [Completed (完了)] Podと [Unknown (不明)] Podの両方が存在する場合があります。これは予期されており、一部のアップグレードの性質により発生します。

3. Podのログを取得します。

```
kubectl logs <pod_name> -n cattle-system
```

22 Upgrade Controller

次のSUSE Edgeプラットフォームコンポーネントに対してアップグレードを実行できるKubernetesコントローラ:

- オペレーティングシステム(SUSE Linux Micro)
- Kubernetes (K3sとRKE2)
- 追加のコンポーネント(Rancher、Elemental、SUSE Securityなど)

[Upgrade Controller \(https://github.com/suse-edge/upgrade-controller\)](https://github.com/suse-edge/upgrade-controller) は、アップグレードのトリガとして機能する単一のユーザ向けリソース内に上記コンポーネントの複雑さをカプセル化することで、アップグレードプロセスを効率化します。ユーザはこのリソースを設定する必要があるのみで、[Upgrade Controller](#)が残りの作業を行います。



注記

[Upgrade Controller](#)は現在、**非エアギャップ管理**クラスタに対してのみSUSE Edgeプラットフォームのアップグレードをサポートしています。詳細については、[22.7項「既知の制限事項」](#)セクションを参照してください。

22.1 SUSE EdgeでのUpgrade Controllerの用途

Upgrade Controllerは、管理クラスタをあるSUSE Edgeリリースバージョンから次のバージョンにアップグレードするために必要な(以前は手動の)「Day 2」操作を自動化するために必要不可欠です。

この自動化のために、Upgrade ControllerはSystem Upgrade Controller ([第21章「System Upgrade Controller」](#))やHelm Controller (<https://github.com/k3s-io/helm-controller/>)などのツールを利用します。

Upgrade Controllerの仕組みの詳細については、[22.4項「Upgrade Controllerの仕組み」](#)を参照してください。

Upgrade Controllerの既知の制限事項については、[22.7項「既知の制限事項」](#)を参照してください。

Upgrade ControllerとSystem Upgrade Controllerの違いについては、[22.2項「Upgrade ControllerとSystem Upgrade Controller」](#)を参照してください。

22.2 Upgrade ControllerとSystem Upgrade Controller

System Upgrade Controller (SUC) (第21章「[System Upgrade Controller](#)」)は、アップグレード手順を特定のKubernetesノードに伝播する汎用ツールです。

System Upgrade ControllerはSUSE Edgeプラットフォームの一部の「Day 2」操作をサポートしていますが、すべてをカバーしているわけでは**ありません**。また、サポートされている操作であっても、ユーザは複数のSUC Planを手動で設定、維持、およびデプロイする必要があります。これはエラーが発生しやすいプロセスであり、予期しない問題が発生する可能性があります。

これにより、SUSE Edgeプラットフォームのさまざまな「Day 2」操作の複雑さを**自動化**および**抽象化**するツールが必要になりました。こうしてUpgrade Controllerが開発されました。Upgrade Controllerは、アップグレードを推進する単一のユーザ向けリソースを導入することで、アップグレードプロセスをシンプルにします。ユーザはこのリソースを管理する必要があるのみで、Upgrade Controllerが残りの作業を行います。

22.3 Upgrade Controllerのインストール

22.3.1 前提条件

- Helm (<https://helm.sh/docs/intro/install/>) 
- cert-manager (<https://cert-manager.io/v1.15-docs/installation/helm/#installing-with-helm>) 
- System Upgrade Controller (21.2項「[System Upgrade Controllerのインストール](#)」)
- Kubernetesクラスタ: K3sまたはRKE2のいずれか

22.3.2 手順

1. 管理クラスタにUpgrade Controller Helmチャートをインストールします。

```
helm install upgrade-controller oci://registry.suse.com/edge/charts/upgrade-controller --version 304.0.1+up0.1.1 --create-namespace --namespace upgrade-controller-system
```

2. Upgrade Controllerのデプロイメントを検証します。

```
kubectl get deployment -n upgrade-controller-system
```

3. Upgrade Controller Podを検証します。

```
kubectl get pods -n upgrade-controller-system
```

4. Upgrade Controller Podログを検証します。

```
kubectl logs <pod_name> -n upgrade-controller-system
```

22.4 Upgrade Controllerの仕組み

Edgeリリースのアップグレードを実行するため、Upgrade Controllerでは2つの新しいKubernetesカスタムリソース (<https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/>) が導入されました。

- UpgradePlan (22.5.1項「UpgradePlan」) - ユーザが作成、Edgeリリースアップグレードに関する設定を保持。
- ReleaseManifest (22.5.2項「ReleaseManifest」) - Upgrade Controllerが作成。特定のEdgeリリースバージョンに固有のコンポーネントバージョンを保持する。**このファイルはユーザが編集する必要はありません。**

Upgrade Controllerは、UpgradePlanリソースの`releaseVersion`プロパティでユーザによって指定されたEdgeリリースバージョンのコンポーネントデータを保持するReleaseManifestリソースの作成に進みます。

Upgrade Controllerは、ReleaseManifestのコンポーネントデータを使用して、次の順序のEdgeリリースコンポーネントのアップグレードに進みます。

1. オペレーティングシステム(OS) (22.4.1項「オペレーティングシステムのアップグレード」)
2. Kubernetes (22.4.2項「Kubernetesのアップグレード」)
3. 追加のコンポーネント(22.4.3項「追加のコンポーネントのアップグレード」)



注記

アップグレードプロセス中に、Upgrade Controllerは、作成したUpgradePlanにアップグレード情報を絶えず出力します。アップグレードプロセスを追跡する方法の詳細については、「アップグレードプロセスの追跡」(22.6項「アップグレードプロセスの追跡」)を参照してください。

22.4.1 オペレーティングシステムのアップグレード

オペレーティングシステムをアップグレードするため、Upgrade Controllerは、次の命名テンプレートを持つSUC (第21章「System Upgrade Controller」) Planを作成します。

- コントロールプレーンノードのOSアップグレードに関連するSUC Planの場合 - `control-plane-<os-name>-<os-version>-<suffix>`
- ワーカーノードのOSアップグレードに関連するSUC Planの場合 - `workers-<os-name>-<os-version>-<suffix>`

これらのプランに基づいて、SUCは、実際のOSアップグレードを実行するクラスタの各ノードにワークロードを作成します。

ReleaseManifestによって、OSアップグレードには次のものが含まれる場合があります。

- Package only updates (パッケージのみの更新) - OSバージョンがEdgeリリース間で変更されないユースケース用。
- Full OS migration (完全なOSマイグレーション) - OSバージョンがEdgeリリース間で変更されるユースケース用。

アップグレードは、コントロールプレーンノードから順に一度に**1台**のノードが実行されます。コントロールプレーンノードのアップグレードが終了した場合にのみ、ワーカーノードのアップグレードが開始されます。



注記

クラスタに特定のタイプの**1台**を超えるノードがある場合、Upgrade Controllerは、クラスタノードの`drain` (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_drain/) を実行するようにOS SUC Planを設定します。

コントロールノードが**1台を超え、1台のみ**ワーカーノードがあるクラスタでは、コントロールプレーンノードに対してのみ`drain`が実行されます。その逆も同様です。

ノードのdrainを完全に無効にする方法については、「UpgradePlan」(22.5.1項「UpgradePlan」)のセクションを参照してください。

22.4.2 Kubernetesのアップグレード

クラスタのKubernetesディストリビューションをアップグレードするため、Upgrade Controllerは、次の命名テンプレートを持つSUC (第21章「System Upgrade Controller」) Planを作成します。

- コントロールプレーンノードのKubernetesアップグレードに関連するSUC Planの場合 - `control-plane-<k8s-version>-<suffix>`。
- ワーカーノードのKubernetesアップグレードに関連するSUC Planの場合 - `workers-<k8s-version>-<suffix>`。

これらのプランに基づいて、SUCは、実際のKubernetesアップグレードを実行するクラスタの各ノードにワークロードを作成します。

Kubernetesアップグレードは、コントロールプレーンノードから順に一度に**1台**のノードが実行されます。コントロールプレーンノードのアップグレードが終了した場合にのみ、ワーカーノードのアップグレードが開始されます。



注記

クラスタに特定のタイプの**1台**を超えるノードがある場合、Upgrade Controllerは、クラスタノードのdrain (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_drain/) を実行するようにKubernetes SUC Planを設定します。

コントロールノードが1台を**超え**、**1台のみ**ワーカーノードがあるクラスタでは、コントロールプレーンノードに対してのみdrainが実行されます。その逆も同様です。

ノードのdrainを完全に無効にする方法については、22.5.1項「UpgradePlan」を参照してください。

22.4.3 追加のコンポーネントのアップグレード

現在、追加のコンポーネントはすべてHelmチャートを介してインストールされます。特定のリリースのコンポーネントの全リストについては、リリースノート(52.1項「要約」)を参照してください。

EIB (第11章「Edge Image Builder」)を通じてデプロイされたHelmチャートの場合、Upgrade Controllerは各コンポーネントの既存のHelmChart CR (<https://docs.rke2.io/helm#using-the-helm-crd>) を更新します。

EIB以外でデプロイされたHelmチャートの場合、Upgrade Controllerは、コンポーネントごとにHelmChartリソースを作成します。

HelmChartリソースの作成/更新後、Upgrade Controllerは、[helm-controller](https://github.com/k3s-io/helm-controller/) (<https://github.com/k3s-io/helm-controller/>) に依存してこの変更を取得し、実際のコンポーネントのアップグレードを続行します。

チャートはReleaseManifestの順序に基づいて順次アップグレードされます。追加の値はUpgradePlanを通じて渡すこともできます。新しいSUSE Edgeリリースでチャートのバージョンが変更されない場合、アップグレードされません。これに関する詳細については、22.5.1項「UpgradePlan」を参照してください。

22.5 Kubernetes API拡張機能

Upgrade Controllerによって導入されたKubernetes APIの拡張機能。

22.5.1 UpgradePlan

Upgrade Controllerは、UpgradePlanと呼ばれる新たなKubernetes [カスタムリソース](https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/) (<https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/>) を導入しました。

UpgradePlanは、Upgrade Controllerの指示メカニズムとして機能し、次の設定をサポートします。

- [releaseVersion](#) - クラスタをアップグレードする必要があるEdgeリリースバージョン。リリースバージョンは、[セマンティック](https://semver.org) (<https://semver.org>) バージョン管理に従う必要があります、リリースノート(52.1項「要約」)から取得する必要があります。
- [disableDrain](#) - **オプション**。Upgrade Controllerにノードの[drains](https://kubernetes.io/docs/reference/kubectl/generated/kubectl_drain/) (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_drain/) を無効にするかどうかを指示します。Disruption Budgets (<https://kubernetes.io/docs/tasks/run-application/configure-pdb/>) を含むワークロードがある場合に役立ちます。
- コントロールプレーンノードのdrain無効化の例:

```
spec:
```

```
disableDrain:
  controlPlane: true
```

- コントロールプレーンとワーカーノードのdrain無効化の例:

```
spec:
  disableDrain:
    controlPlane: true
    worker: true
```

- **helm - オプション**。Helmを介してインストールされたコンポーネントの追加の値を指定します。



警告

アップグレードに重要な値についてのみこのフィールドを使用することをお勧めします。標準のチャート値の更新は、各チャートが次のバージョンにアップグレードされた後に実行する必要があります。

- 例:

```
spec:
  helm:
    - chart: foo
      values:
        bar: baz
```

22.5.2 ReleaseManifest

Upgrade Controllerは、ReleaseManifestと呼ばれる新たなKubernetes **カスタムリソース** (<https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/>) を導入しました。

ReleaseManifestリソースは、Upgrade Controllerによって作成され、**1つ**の特定のEdgeリリースバージョンのコンポーネントデータを保持します。つまり、各Edgeリリースバージョンのアップグレードは、異なるReleaseManifestリソースによって表されます。



警告

Release Manifestは常に、Upgrade Controllerによって作成される必要があります。

ReleaseManifestリソースを手動で作成または編集することはお勧めしていません。
ユーザが手動で作成または編集することを決めた場合は、**自己責任**で行う必要があります。

Release Manifestが提供するコンポーネントデータには、以下が含まれますが、これに制限されません。

- オペレーティングシステムデータ - バージョン、サポートされているアーキテクチャ、追加のアップグレードデータなど
- Kubernetesディストリビューションデータ - RKE2 (<https://docs.rke2.io>)  /K3s (<https://k3s.io>)  でサポートされているバージョン
- 追加のコンポーネントデータ - SUSE Helmチャートデータ(場所、バージョン、名前など)

ReleaseManifestの例については、[アップストリーム \(https://github.com/suse-edge/upgrade-controller/blob/main/config/samples/lifecycle_v1alpha1_releasemanifest.yaml\)](https://github.com/suse-edge/upgrade-controller/blob/main/config/samples/lifecycle_v1alpha1_releasemanifest.yaml)  ドキュメントを参照してください。これは単なる例であって、有効なReleaseManifestリソースとして作成することを目的としたものではないことに注意してください。

22.6 アップグレードプロセスの追跡

このセクションは、ユーザがUpgradePlanリソースを作成すると、Upgrade Controllerが開始するアップグレードプロセスを追跡およびデバッグする手段として機能します。

22.6.1 一般

アップグレードプロセスの状態に関する一般情報は、Upgrade Planのステータス状況で確認できます。

Upgrade Planリソースのステータスは、次の方法で確認できます。

```
kubectl get upgradeplan <upgradeplan_name> -n upgrade-controller-system -o yaml
```

Upgrade Planの実行例:

```
apiVersion: lifecycle.suse.com/v1alpha1
kind: UpgradePlan
metadata:
```

```
name: upgrade-plan-mgmt
namespace: upgrade-controller-system

spec:
  releaseVersion: 3.4

status:
  conditions:
    - lastTransitionTime: "2024-10-01T06:26:27Z"
      message: Control plane nodes are being upgraded
      reason: InProgress
      status: "False"
      type: OSUpgraded
    - lastTransitionTime: "2024-10-01T06:26:27Z"
      message: Kubernetes upgrade is not yet started
      reason: Pending
      status: Unknown
      type: KubernetesUpgraded
    - lastTransitionTime: "2024-10-01T06:26:27Z"
      message: Rancher upgrade is not yet started
      reason: Pending
      status: Unknown
      type: RancherUpgraded
    - lastTransitionTime: "2024-10-01T06:26:27Z"
      message: Longhorn upgrade is not yet started
      reason: Pending
      status: Unknown
      type: LonghornUpgraded
    - lastTransitionTime: "2024-10-01T06:26:27Z"
      message: MetalLB upgrade is not yet started
      reason: Pending
      status: Unknown
      type: MetalLBUpgraded
    - lastTransitionTime: "2024-10-01T06:26:27Z"
      message: CDI upgrade is not yet started
      reason: Pending
      status: Unknown
      type: CDIUpgraded
    - lastTransitionTime: "2024-10-01T06:26:27Z"
      message: KubeVirt upgrade is not yet started
```

```
- lastTransitionTime: "2024-10-01T06:26:27Z"
```

```
  message: Elemental upgrade is not yet started
```

```
  reason: Pending
```

```
  status: Unknown
```

```
  type: ElementalUpgraded
```

```
- lastTransitionTime: "2024-10-01T06:26:27Z"
```

```
  message: SRIOV upgrade is not yet started
```

```
  reason: Pending
```

```
  status: Unknown
```

```
  type: SRIOVUpgraded
```

```
- lastTransitionTime: "2024-10-01T06:26:27Z"
```

```
  message: Metal3 upgrade is not yet started
```

```
  reason: Pending
```

```
  status: Unknown
```

```
  type: Metal3Upgraded
```

```
- lastTransitionTime: "2024-10-01T06:26:27Z"
```

```
  message: RancherTurtles upgrade is not yet started
```

```
  reason: Pending
```

```
  status: Unknown
```

```
  type: RancherTurtlesUpgraded
```

```
observedGeneration: 1
```

```
sucNameSuffix: 90315a2b6d
```

ここで、Upgrade Controllerがアップグレードをスケジュールしようとするすべてのコンポーネントを確認できます。各状態は、次のテンプレートに従います。

- `lastTransitionTime` - このコンポーネントの状態が、あるステータスから別のステータスに遷移した最後の時刻。
- `message` (メッセージ) - 特定のコンポーネントの状態の現在のアップグレード状態を示すメッセージ。
- `reason` (理由) - 特定のコンポーネントの状態の現在のアップグレード状態。考えられる理由には次のものが含まれます。

- `Succeeded` (成功) - 特定のコンポーネントのアップグレードが成功しました。
- `Failed` (失敗) - 特定のコンポーネントアップグレードが失敗しました。
- `InProgress` (進行中) - 特定のコンポーネントのアップグレードは現在進行中です。

165

- `Pending` (保留中) - 特定のコンポーネントのアップグレードは、まだスケジュール般されていません。

- `Skipped` (スキップ) - 特定のコンポーネントがクラスタで見つからないため、アップグレードはスキップされます。

- `status` (ステータス) - 現在の状態 `タイプ`のステータス。 `True`、 `False`、 `Unknown` (不明)のいずれか。
- `type` (タイプ) - 現在アップグレードされたコンポーネントのインジケータ。

Upgrade Controllerは、`OSUpgraded`および`KubernetesUpgraded`タイプのコンポーネント状態のSUC Planを作成します。これらのコンポーネント用に作成されたSUC Planを詳細に追跡するには、[21.3項「System Upgrade Controller Planのモニタリング」](#)を参照してください。他のすべてのコンポーネント状態のタイプは、`helm-controller` (<https://github.com/k3s-io/helm-controller/>) によって作成されたリソースを表示することで詳細に追跡できます。詳細については、[22.6.2項「Helm Controller」](#)を参照してください。

Upgrade ControllerによってスケジュールされたUpgrade Planは、次の場合に`successful` (成功)とマーク付けすることができます。

1. `Pending` (保留中)または`InProgress` (進行中)のコンポーネントの状態がありません。
2. `lastSuccessfulReleaseVersion`プロパティがUpgrade Planの設定で指定された`releaseVersion`を指しています。このプロパティは、アップグレードプロセスが成功すると、Upgrade ControllerによってUpgrade Planのステータスに追加されます。

成功したUpgradePlanの例:

```
apiVersion: lifecycle.suse.com/v1alpha1
kind: UpgradePlan
metadata:
  name: upgrade-plan-mgmt
  namespace: upgrade-controller-system
spec:
  releaseVersion: 3.4
status:
  conditions:
    - lastTransitionTime: "2024-10-01T06:26:48Z"
      message: All cluster nodes are upgraded
      reason: Succeeded
      status: "True"
      type: OSUpgraded
    - lastTransitionTime: "2024-10-01T06:26:59Z"
      message: All cluster nodes are upgraded
      reason: Succeeded
      status: "True"
```

166 `type: KubernetesUpgraded`

```
- lastTransitionTime: "2024-10-01T06:27:13Z"
  message: Chart rancher upgrade succeeded
  reason: Succeeded
```

- lastTransitionTime: "2024-10-01T06:27:13Z"
message: Chart longhorn is not installed
reason: Skipped
status: "False"
type: LonghornUpgraded
- lastTransitionTime: "2024-10-01T06:27:13Z"
message: Specified version of chart metallb is already installed
reason: Skipped
status: "False"
type: MetalLBUpgraded
- lastTransitionTime: "2024-10-01T06:27:13Z"
message: Chart cdi is not installed
reason: Skipped
status: "False"
type: CDIUpgraded
- lastTransitionTime: "2024-10-01T06:27:13Z"
message: Chart kubevirt is not installed
reason: Skipped
status: "False"
type: KubeVirtUpgraded
- lastTransitionTime: "2024-10-01T06:27:13Z"
message: Chart neuvector-crd is not installed
reason: Skipped
status: "False"
type: NeuVectorUpgraded
- lastTransitionTime: "2024-10-01T06:27:14Z"
message: Specified version of chart endpoint-copier-operator is already installed
reason: Skipped
status: "False"
type: EndpointCopierOperatorUpgraded
- lastTransitionTime: "2024-10-01T06:27:14Z"
message: Chart elemental-operator upgrade succeeded
reason: Succeeded
status: "True"
type: ElementalUpgraded
- lastTransitionTime: "2024-10-01T06:27:15Z"
message: Chart sriov-crd is not installed
reason: Skipped

```
observedGeneration: 1
sucNameSuffix: 90315a2b6d
```

22.6.2 Helm Controller

このセクションでは、[helm-controller \(https://github.com/k3s-io/helm-controller/\)](https://github.com/k3s-io/helm-controller/)  によって作成されたリソースを追跡する方法について説明します。



注記

以下の手順は、`kubectl`がUpgrade Controllerが導入されているクラスタに接続するように設定されていることを前提としています。

1. 特定のコンポーネントのHelmChartリソースを見つけます。

```
kubectl get helmcharts -n kube-system
```

2. HelmChartリソースの名前を使用して、`helm-controller`によって作成されたアップグレードPodを見つけます。

```
kubectl get pods -l helmcharts.helm.cattle.io/chart=<helmchart_name> -n kube-system

# Example for Rancher
kubectl get pods -l helmcharts.helm.cattle.io/chart=rancher -n kube-system
NAME                                READY   STATUS    RESTARTS   AGE
helm-install-rancher-tv9wn          0/1     Completed 0           16m
```

3. コンポーネント固有のPodのログを表示します。

```
kubectl logs <pod_name> -n kube-system
```

22.7 既知の制限事項

- ダウンストリームクラスタのアップグレードはUpgrade Controllerによってまだ管理されていません。ダウンストリームクラスタをアップグレードする方法については、[第36章「ダウンストリームクラスタ」](#)を参照してください。
- Upgrade Controllerは、EIB ([第11章「Edge Image Builder」](#))を通じてデプロイされる追加のSUSE Edge Helmチャートについて、そのHelmChart CR (<https://docs.rke2.io/helm#using-the-helm-crd>)  がkube-system名前空間にデプ

ロイされていることを想定しています。これを実行するには、EIB定義ファイルで`installationNamespace`プロパティを設定します。詳細については、[アップストリーム \(https://github.com/suse-edge/edge-image-builder/blob/main/docs/building-images.md#kubernetes\)](https://github.com/suse-edge/edge-image-builder/blob/main/docs/building-images.md#kubernetes)  ドキュメントを参照してください。

- 現在、Upgrade Controllerには、管理クラスタで現在実行中のEdgeリリースバージョンを判断する方法がありません。クラスタで現在実行しているEdgeリリースバージョンより大きいEdgeリリースバージョンを指定するようにしてください。
- 現在、Upgrade Controllerは、**非エアギャップ**環境のアップグレードのみをサポートしています。**エアギャップ**アップグレードは、まだ可能ではありません。

23 SUSE Multi-Linux Manager

SUSE Multi-Linux ManagerはSUSE Edgeに含まれており、エッジデプロイメントのすべてのノードでSUSE Linux Microを基盤となるオペレーティングシステムとして常に最新の状態に保つための自動化と制御を提供します。

詳細については、[第4章「SUSE Multi-Linux Manager」](#)と[SUSE Multi-Linux Managerのドキュメント](#) (<https://documentation.suse.com/suma/5.0/en/suse-manager/index.html>)  を参照してください。

III ハウツーガイド

- 24 K3s上のMetalLB (レイヤ2モードを使用) **172**
- 25 K3s上のMetalLB (レイヤ3モードを使用) **181**
- 26 Kubernetes APIサーバの前面のMetalLB **187**
- 27 Edge Image Builderを使用したエアギャップデプロイメント **194**
- 28 Kiwiを使用したSUSE Linux Microの更新イメージの構築 **214**
- 29 ClusterClassを使用したダウンストリームクラスタのデプロイ **219**

ハウツーガイドとベストプラクティス

24 K3s上のMetalLB (レイヤ2モードを使用)

MetalLBは、標準のルーティングプロトコルを使用する、ベアメタルKubernetesクラスタ用のロードバランサの実装です。

このガイドでは、MetalLBをレイヤ2 (L2) モードでデプロイする方法について説明します。

24.1 MetalLBを使用する理由

MetalLBは、いくつかの理由により、ベアメタルKubernetesクラスタでの負荷分散に最適な選択肢です。

1. Kubernetesとのネイティブ統合: MetalLBはKubernetesとシームレスに統合されており、使い慣れたKubernetesツールとプラクティスを使用して簡単にデプロイおよび管理できます。
2. ベアメタルとの互換性: クラウドベースのロードバランサとは異なり、MetalLBは、従来のロードバランサが利用できない、または実現できないオンプレミスデプロイメント向けに特別に設計されています。
3. 複数のプロトコルをサポート: MetalLBはレイヤ2モードとBGP (Border Gateway Protocol) モードの両方をサポートし、さまざまなネットワークアーキテクチャと要件に柔軟に対応します。
4. 高可用性: MetalLBは、負荷分散の責任を複数のノードに分散することで、サービスの高可用性と信頼性を保証します。
5. スケーラビリティ: MetalLBは大規模なデプロイメントに対応し、Kubernetesクラスタに合わせてスケーリングして需要の増加に対応します。

レイヤ2モードでは、1つのノードがローカルネットワークにサービスをアドバタイズする責任を負います。ネットワークの視点からは、マシンのネットワークインタフェースに複数のIPアドレスが割り当てられているように見えます。

レイヤ2モードの主な利点は、その汎用性です。あらゆるEthernetネットワークで動作し、特別なハードウェアも、高価なルータも必要ありません。

24.2 K3s上のMetalLB (L2を使用)

このクイックスタートではL2モードを使用します。つまり、特別なネットワーク機器は必要ありませんが、ネットワーク範囲内の空きIPが3つ必要です。

24.3 前提条件

- MetalLBがデプロイされるK3sクラスタ。



警告

K3sには、Klipperという名前の独自のサービスロードバランサが付属しています。MetalLBを実行するにはKlipperを無効にする必要があります (<https://metallb.universe.tf/configuration/k3s/>) 。Klipperを無効にするには、`--disable=service` フラグを使用してK3sをインストールする必要があります。

- Helm
- ネットワーク範囲内の3つの空きIPアドレス。この例では 192.168.122.10-192.168.122.12



重要

これらのIPアドレスが未割り当てであることを確認する必要があります。DHCP環境では、二重割り当てを避けるため、これらのアドレスをDHCPプールに含めないでください。

24.4 デプロイメント

SUSEでは、SUSE Edgeソリューションの一部として公開されているMetalLB Helmチャートを使用します。

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

while ! kubectl wait --for condition=ready -n metallb-system $(kubectl get \
  pods -n metallb-system -l app.kubernetes.io/component=controller -o name) \
  --timeout=10s; do
  sleep 2
done
```

24.5 設定

この時点でインストールは完了しています。次に、サンプル値を使用して設定 (<https://metallb.universe.tf/configuration/>) を行います。

```
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
```

これで、MetalLBを使用する準備ができました。L2モードでは、次のようにさまざまな設定をカスタマイズできます。

- IPv6とデュアルスタックサービス (<https://metallb.universe.tf/usage/#ipv6-and-dual-stack-services>) 
- アドレスの自動割り当てを制御する (https://metallb.universe.tf/configuration/_advanced_ipaddresspool_configuration/#controlling-automatic-address-allocation) 
- アドレス割り当ての範囲を特定のネームスペースとサービスに縮小する (https://metallb.universe.tf/configuration/_advanced_ipaddresspool_configuration/#reduce-scope-of-address-allocation-to-specific-namespace-and-service) 

- サービスをアナウンスできるノードのセットを制限する (https://metallb.universe.tf/configuration/_advanced_l2_configuration/#limiting-the-set-of-nodes-where-the-service-can-be-announced-from) 
- LBのIPをアナウンスできるネットワークインタフェースを指定する (https://metallb.universe.tf/configuration/_advanced_l2_configuration/#specify-network-interfaces-that-lb-ip-can-be-announced-from) 

また、BGP (https://metallb.universe.tf/configuration/_advanced_bgp_configuration/)  についても、さらに多くのカスタマイズが可能です。

24.5.1 TraefikとMetalLB

Traefikは、デフォルトではK3sとともにデプロイされます(`--disable=traefik`を使用してK3sを無効にできます (<https://docs.k3s.io/networking#traefik-ingress-controller>) )。また、デフォルトでLoadBalancerとして公開されます(Klipperで使用するため)。ただし、Klipperを無効にする必要があるため、Ingress用TraefikサービスはLoadBalancerタイプのままです。そのため、MetalLBをデプロイした時点では、最初のIPは自動的にTraefik Ingressに割り当てられます。

```
# 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
```

これは、このプロセスの後半(24.6.1項「IngressとMetalLB」)で適用されます。

24.6 使用法

デプロイメントの例を作成してみましょう。

```
cat <<- EOF | kubectl apply -f -
---
apiVersion: v1
kind: Namespace
metadata:
  name: hello-kubernetes
---
```

```

apiVersion: v1
kind: ServiceAccount
metadata:
  name: hello-kubernetes
  namespace: hello-kubernetes
  labels:
    app.kubernetes.io/name: hello-kubernetes
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-kubernetes
  namespace: hello-kubernetes
  labels:
    app.kubernetes.io/name: hello-kubernetes
spec:
  replicas: 2
  selector:
    matchLabels:
      app.kubernetes.io/name: hello-kubernetes
  template:
    metadata:
      labels:
        app.kubernetes.io/name: hello-kubernetes
    spec:
      serviceAccountName: hello-kubernetes
      containers:
        - name: hello-kubernetes
          image: "paulbouwer/hello-kubernetes:1.10"
          imagePullPolicy: IfNotPresent
          ports:
            - name: http
              containerPort: 8080
              protocol: TCP
          livenessProbe:
            httpGet:
              path: /
              port: http
          readinessProbe:
            httpGet:
              path: /
              port: http
          env:
            - name: HANDLER_PATH_PREFIX
              value: ""
            - name: RENDER_PATH_PREFIX
              value: ""
            - name: KUBERNETES_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace

```



```

- name: KUBERNETES_POD_NAME
  valueFrom:
    fieldRef:
      fieldPath: metadata.name
- name: KUBERNETES_NODE_NAME
  valueFrom:
    fieldRef:
      fieldPath: spec.nodeName
- name: CONTAINER_IMAGE
  value: "paulbouwer/hello-kubernetes:1.10"

```

EOF

最終的にサービスは次のようになります。

```

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

```

実際の動作を見てみましょう。

```

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>
    <div id="info">
      <table>
        <tr>
          <th>namespace:</th>
          <td>hello-kubernetes</td>
        </tr>
        <tr>
          <th>pod:</th>
          <td>hello-kubernetes-7c8575c848-2c6ps</td>
        </tr>
        <tr>
          <th>node:</th>
          <td>allinone (Linux 5.14.21-150400.24.46-default)</td>
        </tr>
      </table>
    </div>
    <div id="footer">
      paulbouwer/hello-kubernetes:1.10 (linux/amd64)
    </div>
  </div>
</div>

</body>
</html>

```

24.6.1 IngressとMetalLB

TraefikはすでにIngressコントローラとして機能しているため、次のようなIngressオブジェクトを介してHTTP/HTTPSトラフィックを公開できます。

```

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

これにより、次のような結果が返されます。

```
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>
</body>
</html>
```

MetalLBが正しく動作していることを確認します。

```
% arping hellok3s.${IP}.sslip.io

ARPING 192.168.64.210
60 bytes from 92:12:36:00:d3:58 (192.168.64.210): index=0 time=1.169 msec
60 bytes from 92:12:36:00:d3:58 (192.168.64.210): index=1 time=2.992 msec
60 bytes from 92:12:36:00:d3:58 (192.168.64.210): index=2 time=2.884 msec
```

上記の例では、トラフィックは次のように流れます。

1. hellok3s.\${IP}.sslip.ioが実際のIPに解決されます。
2. 続いて、トラフィックがmetallb-speaker Podによって処理されます。
3. metallb-speakerがトラフィックをtraefikコントローラにリダイレクトします。
4. 最後に、Traefikが要求をhello-kubernetesサービスに転送します。

25 K3s上のMetalLB (レイヤ3モードを使用)

MetalLBは、標準のルーティングプロトコルを使用する、ベアメタルKubernetesクラスタ用のロードバランサの実装です。

このガイドでは、MetalLBをレイヤ3 (L3) BGPモードでデプロイする方法について説明します。

25.1 MetalLBを使用する理由

MetalLBは、いくつかの理由により、ベアメタルKubernetesクラスタでの負荷分散に最適な選択肢です。

1. **Kubernetesとのネイティブ統合:** MetalLBはKubernetesとシームレスに統合されており、使い慣れたKubernetesツールとプラクティスを使用して簡単にデプロイおよび管理できます。
2. **ベアメタルとの互換性:** クラウドベースのロードバランサとは異なり、MetalLBは、従来のロードバランサが利用できない、または実現できないオンプレミスデプロイメント向けに特別に設計されています。
3. **複数のプロトコルをサポート:** MetalLBはレイヤ2モードとレイヤ3 BGP (Border Gateway Protocol) モードの両方をサポートし、さまざまなネットワークアーキテクチャと要件に柔軟に対応します。
4. **高可用性:** MetalLBは、負荷分散の責任を複数のノードに分散することで、サービスの高可用性と信頼性を保証します。
5. **スケーラビリティ:** MetalLBは大規模なデプロイメントに対応し、Kubernetesクラスタに合わせてスケーリングして需要の増加に対応します。

レイヤ2モードでは、1つのノードがローカルネットワークにサービスをアドバタイズする責任を負います。ネットワークの視点からは、マシンのネットワークインタフェースに複数のIPアドレスが割り当てられているように見えます。

レイヤ2モードの主な利点は、その汎用性です。あらゆるEthernetネットワークで動作し、特別なハードウェアも、高価なルータも必要ありません。

25.2 K3s上のMetalLB (L3を使用)

このクイックスタートではL3モードを使用します。つまり、ネットワーク範囲内にBGP機能を備えた近隣ルータが必要です。

25.3 前提条件

- MetalLBがデプロイされるK3sクラスタ。
- BGPプロトコルをサポートするネットワーク上のルータ。
- サービス用のネットワーク範囲内の空きIPアドレス。この例では 192.168.10.100



重要

このIPアドレスが未割り当てであることを確認する必要があります。DHCP環境では、二重割り当てを避けるため、このアドレスをDHCPプールに含めないでください。

25.4 サービスIPアドレスをアドバタイズするための設定

すぐに利用可能なBGPは設定済みのすべてのピアにサービスIPアドレスをアドバタイズします。これらのピア(通常はルータ)は、各サービスIPアドレスに対して32ビットのネットワークマスクを持つルートを受け取ります。この例では、クラスタと同じネットワーク上にあるFRRベースのルータを使用します。その後、MetalLBのBGP機能を使用して、そのFRRベースのルータにサービスをアドバタイズします。

25.5 デプロイメント

SUSEでは、SUSE Edgeソリューションの一部として公開されているMetalLB Helmチャートを使用します。

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

```
while ! kubectl wait --for condition=ready -n metallb-system $(kubectl get\
pods -n metallb-system -l app.kubernetes.io/component=controller -o name)\
--timeout=10s; do
sleep 2
done
```

25.6 設定

1. この時点でインストールは完了します。 IPAddressPoolを作成します。

```
cat <<-EOF | kubectl apply -f -
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: bgp-pool
  namespace: metallb-system
  labels:
    app: httpd
spec:
  addresses:
  - 192.168.10.100/32
  autoAssign: true
  avoidBuggyIPs: false
  serviceAllocation:
    namespaces:
    - metallb-system
  priority: 100
  serviceSelectors:
  - matchExpressions:
    - key: serviceType
      operator: In
      values:
      - httpd
EOF
```

2. BGPPeerを設定します。



注記

FRRルータのASNは1000であり、BGPPeerは1001です。また、FRRルータのIPアドレスは192.168.3.140であることも確認できます。

```
cat <<-EOF | kubectl apply -f -
apiVersion: metallb.io/v1beta2
```

```
kind: BGPPeer
metadata:
  namespace: metallb-system
  name: mypeertest
spec:
  peerAddress: 192.168.3.140
  peerASN: 1000
  myASN: 1001
  routerID: 4.4.4.4
EOF
```

3. BGPAdvertisement (L3)を作成します。

```
cat <<-EOF | kubectl apply -f -
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-test
  namespace: metallb-system
spec:
  ipAddressPools:
    - bgp-pool
EOF
```

25.7 使用法

1. サービスを含むサンプルアプリケーションを作成します。この場合、そのサービスに対する IPAddressPoolからのIPアドレスは192.168.10.100です。

```
cat <<- EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: httpd-deployment
  namespace: metallb-system
  labels:
    app: httpd
spec:
  replicas: 3
  selector:
    matchLabels:
      pod-label: httpd
  template:
    metadata:
      labels:
        pod-label: httpd
```



```

spec:
  containers:
  - name: httpdcontainer
    image: image: docker.io/library/httpd:2.4
    ports:
    - containerPort: 80
      protocol: TCP
    restartPolicy: Always

---
apiVersion: v1
kind: Service
metadata:
  name: http-service
  namespace: metallb-system
  labels:
    serviceType: httpd
spec:
  selector:
    pod-label: httpd
  type: LoadBalancer
  ports:
  - protocol: TCP
    port: 8080
    name: 8080-tcp
    targetPort: 80
EOF

```

2. 確認するには、FRRルータにログオンし、BGPアドバタイズメントから作成されたルートを確認できます。

```

42178089cba5# show ip bgp all

For address family: IPv4 Unicast
BGP table version is 3, local router ID is 2.2.2.2, vrf id 0
Default local pref 100, local AS 1000
Status codes: s suppressed, d damped, h history, * valid, > best, = multipath,
               i internal, r RIB-failure, S Stale, R Removed
Nexthop codes: @NNN nexthop's vrf id, < announce-nh-self
Origin codes: i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

   Network          Next Hop           Metric LocPrf Weight Path
* i172.16.0.0/24    1.1.1.1             0      100      0 i
*>                  0.0.0.0             0           32768 i
* i172.17.0.0/24    3.3.3.3             0      100      0 i
*>                  0.0.0.0             0           32768 i
*= 192.168.10.100/32
                    192.168.3.162           0 1001 i
*=                  192.168.3.163           0 1001 i

```

```
*> 192.168.3.161 0 1001 i

Displayed 3 routes and 7 total paths
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
```

3. このルータがネットワークのデフォルトゲートウェイである場合、そのネットワーク上のボックスから`curl`コマンドを実行し、`httpd`サンプルアプリに到達できることを確認できます。

```
# curl http://192.168.10.100:8080
<html><body><h1>It works!</h1></body></html>
#
```

26 Kubernetes APIサーバの前面のMetalLB

このガイドでは、MetalLBサービスを使用して、3つのコントロールプレーンノードを持つHAクラスタ上でRKE2/K3s APIを外部に公開する方法を示します。これを実現するために、LoadBalancerタイプのKubernetes Serviceを手動で作成します。その後、EndpointSlicesオブジェクトが自動的に作成され、クラスタで使用可能なすべてのコントロールプレーンノードのIPが保持されます。EndpointSlicesをクラスタで発生するイベント(ノードの追加/削除やノードのオフライン化)と継続的に同期するため、Endpoint Copier Operator (第19章「Endpoint Copier Operator」)がデプロイされます。このオペレータはデフォルトのkubernetes EndpointSlicesで発生するイベントを監視し、管理対象を自動的に更新して同期を維持します。管理対象のServiceはLoadBalancerタイプであるため、MetalLBは静的なExternalIPを割り当てます。このExternalIPはAPIサーバとの通信に使用されます。

26.1 前提条件

- RKE2/K3sをデプロイするための3つのホスト。
 - ホスト名は各ホストで違う名前にしてください。
 - テストの場合は仮想マシンを使用できます。
- ネットワーク内で2つ以上のIPが使用可能(Traefik/Nginx用に1つ、管理対象サービス用に1つ)。
- Helm

26.2 RKE2/K3sのインストール



注記

新しいクラスタを使用せず、既存のクラスタを使用する場合は、この手順をスキップして次の手順に進んでください。

まず、ネットワーク内の空きIPを、後で管理対象のServiceのExternalIPでできるように予約する必要があります。

最初のホストにSSHで接続して、クラスタモードに必要なディストリビューションをインストールします。

RKE2の場合:

```
# 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)
```

K3sの場合:

```
# 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 -
```



注記

必ず、`k3s server`コマンドで`--disable=serviceLB`フラグを指定してください。



重要

これ以降、コマンドはローカルマシンで実行する必要があります。

APIサーバに外部からアクセスするには、RKE2/K3s VMのIPを使用します。

```
# Replace <node-ip> with the actual IP of the machine
export NODE_IP=<node-ip>
export KUBE_DISTRIBUTION=<k3s/rke2>

scp ${NODE_IP}:/etc/rancher/${KUBE_DISTRIBUTION}/${KUBE_DISTRIBUTION}.yaml ~/.kube/config
&& sed \
-i 's/127.0.0.1/${NODE_IP}/g' ~/.kube/config && chmod 600 ~/.kube/config
```

26.3 既存のクラスタの設定



注記

この手順は、既存のRKE2/K3sクラスタを使用する場合にのみ有効です。

既存のクラスタを使用するには、`tls-san`フラグを修正する必要があります。また、K3sでは`servicelb` LBを無効にする必要があります。

RKE2またはK3sサーバのフラグを変更するには、ディストリビューションに応じて、クラスタのすべてのVM上で`/etc/systemd/system/rke2.service`または`/etc/systemd/system/k3s.service`ファイルを変更する必要があります。

フラグは`ExecStart`に挿入する必要があります。例:

RKE2の場合:

```
# 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' \
```

K3sの場合:

```
# 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' \
```

次に、次のコマンドを実行して、新しい設定をロードする必要があります。

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

26.4 MetalLBのインストール

`MetalLB`をデプロイするには、K3s上の`MetalLB` (第24章「[K3s上のMetalLB \(レイヤ2モードを使用\)](#)」)のガイドを使用できます。

メモ: `VIP_SERVICE_IP` IPアドレスがクラスタ内の既存の `IPAddressPools` と重複していないことを確認してください。

管理対象のServiceにのみ使用される個別の `IpAddressPool` と `L2Advertisement` を作成します。

メモ: 以下の `IpAddressPool` は、`default` ネームスペースの `LoadBalancer` タイプのServiceに割り当てられます。複数の `LoadBalancer` サービスがそこに存在する場合、追加の `ServiceSelectors` (https://metallb.universe.tf/configuration/_advanced_ipaddresspool_configuration/#reduce-scope-of-address-allocation-to-specific-namespace-and-service) をこのVIPサービスと明示的に一致するように設定できます。

```
# 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: kubernetes-vip-l2-adv
  namespace: metallb-system
spec:
  ipAddressPools:
  - kubernetes-vip-ip-pool
EOF
```

26.5 Endpoint Copier Operatorのインストール

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

```
--namespace endpoint-copier-operator \  
--create-namespace
```

上記のコマンドは2つのレプリカを持つ`endpoint-copier-operator`オペレータのデプロイメントをデプロイします。一方がリーダーとなり、他方は必要に応じてリーダーの役割を引き継ぎます。

これで、`kubernetes-vip` Serviceがデプロイされ、オペレータによって調整され、設定されたポートとIPを持つEndpointSlicesが作成されます。

RKE2の場合:

```
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
```

K3sの場合:

```
cat <<-EOF | kubectl apply -f -  
apiVersion: v1  
kind: Service  
metadata:  
  name: kubernetes-vip  
  namespace: default  
spec:  
  internalTrafficPolicy: Cluster  
  ipFamilies:  
    - IPv4  
  ipFamilyPolicy: SingleStack  
  ports:  
    - name: https  
      port: 6443  
      protocol: TCP  
      targetPort: 6443  
  sessionAffinity: None
```

```
type: LoadBalancer
EOF
```

kubernetes-vipサービスのIPアドレスが正しいことを確認します。

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

default名前空間のkubernetes-vip-*およびkubernetesのEndpointSlicesリソースが同じIPを指していることを確認します。

```
kubectl get endpointslices | grep kubernetes
```

すべて問題なければ、最後にKubeconfigでVIP_SERVICE_IPを使用します。

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

これ以降、kubectlはすべてkubernetes-vipサービスを経由するようになります。

26.6 コントロールプレーンノードの追加

プロセス全体を監視するため、端末タブを2つ以上開くことができます。

最初の端末:

```
watch kubectl get nodes
```

2つ目の端末:

```
watch kubectl get endpointslices
```

次に、以下のコマンドを2つ目のノードと3つ目のノードで実行します。

RKE2の場合:

```
# 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
```

K3sの場合:

```
# 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 -
```

27 Edge Image Builderを使用したエアギャップデプロイメント

27.1 概要

このガイドでは、Edge Image Builder (EIB) (第11章「Edge Image Builder」)を使用し、完全にエアギャップされた環境で複数のSUSE EdgeコンポーネントをSUSE Linux Micro 6.1上にデプロイする方法を示します。これにより、EIBで作成したCustomized, Ready to Boot (CRB) イメージでブートし、指定したコンポーネントをインターネット接続や手動手順なしにRKE2 クラスタまたはK3sクラスタにデプロイできます。この設定は、デプロイメントに必要なアーティファクトをすべてOSイメージにプリベイクし、ブート後すぐに利用できるようにしたいお客様にとって非常に便利です。

ここでは、以下のエアギャップインストールについて説明します。

- 第5章「Rancher」
- 第17章「SUSE Security」
- 第16章「SUSE Storage」
- 第20章「Edge Virtualization」



警告

EIBは、指定したHelmチャートとKubernetesマニフェストで参照されているイメージをすべて解析し、事前にダウンロードします。ただし、その一部がコンテナイメージをプルし、そのイメージに基づいて実行時にKubernetesリソースを作成しようとする場合があります。このような場合、完全なエアギャップ環境を設定するには、必要なイメージを定義ファイルに手動で指定する必要があります。

27.2 前提条件

このガイドに従って操作を進める場合、すでにEIB (第11章「Edge Image Builder」)に精通していることを想定しています。まだEIBに精通していない場合は、クイックスタートガイド(第3章「Edge Image Builderを使用したスタンドアロンクラスタ」)に従って、以下の演習で示されている概念の理解を深めてください。

27.3 Libvirtのネットワーク設定



注記

エアギャップデプロイメントのデモを示すため、このガイドはシミュレートされたエアギャップlibvirtネットワークを使用して実施し、それに合わせて以下の設定を調整します。ご自身のデプロイメントでは、host1.local.yamlの設定の変更が必要になる場合があります。これについては、次の手順で説明します。

同じlibvirtネットワーク設定を使用する場合は、このまま読み進めてください。そうでない場合は、[27.4項「ベースディレクトリの設定」](#)までスキップしてください。

DHCPのIPアドレス範囲192.168.100.2/24で、分離されたネットワーク設定を作成してみましょう。

```
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
```

あとはネットワークを作成して起動するだけです。

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

27.4 ベースディレクトリの設定

ベースディレクトリの設定は、各種のコンポーネントすべてで同じであるため、ここで設定します。

まず、必要なサブディレクトリを作成します。

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

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

必ず、使用する予定のゴールデンイメージをbase-imagesディレクトリに追加してください。このガイドでは、[こちら \(https://www.suse.com/download/sle-micro/\)](https://www.suse.com/download/sle-micro/) にあるセルフインストールISOに焦点を当てて説明します。

ダウンロードしたイメージをコピーしましょう。

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



注記

EIBは、ゴールデンイメージの入力を変更することはありません。

目的のネットワーク設定を含むファイルを作成しましょう。

```
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
```

この設定により、プロビジョニングされたシステムに以下が確実に存在するようになります (指定されたMACアドレスを使用)。

- 静的IPアドレスを持つEthernetインタフェース
- ルーティング
- DNS
- ホスト名(host1.local)

結果のファイル構造は次のようになります。

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

27.5 ベース定義ファイル

Edge Image Builderでは、**定義ファイル**を使用してSUSE Linux Microイメージを変更します。定義ファイルには、設定可能なオプションの大部分が含まれています。これらのオプションの多くは、異なるコンポーネントのセクションで繰り返し使用されるため、ここで一覧にして説明します。



ヒント

定義ファイルのカスタマイズオプションの全リストについては、[アップストリームドキュメント \(https://github.com/suse-edge/edge-image-builder/blob/release-1.1/docs/building-images.md#image-definition-file\)](https://github.com/suse-edge/edge-image-builder/blob/release-1.1/docs/building-images.md#image-definition-file) を参照してください。

すべての定義ファイルに存在する次のフィールドを見てみましょう。

```
apiVersion: 1.3
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: v1.33.3+rke2r1
embeddedArtifactRegistry:
  images:
    - ...

```

`image`セクションは必須であり、入力イメージ、そのアーキテクチャとタイプ、および出力イメージの名前を指定します。

`operatingSystem`セクションはオプションであり、プロビジョニングされたシステムに `root/eib` のユーザ名/パスワードでログインできるようにするための設定が含まれます。

`kubernetes`セクションはオプションであり、Kubernetesタイプとバージョンを定義しています。RKE2ディストリビューションを使用します。代わりにK3sが必要な場合は、`kubernetes.version: v1.33.3+k3s1`を使用します。`kubernetes.nodes`フィールドを介して明示的に設定しない限り、このガイドでブートストラップするすべてのクラスタは、シングルノードクラスタになります。

`embeddedArtifactRegistry`セクションには、実行時に特定のコンポーネントでのみ参照されてプルされるイメージがすべて含まれます。

27.6 Rancherのインストール



注記

デモで示すRancher (第5章「Rancher」) のデプロイメントは、デモのために非常にスリム化されています。実際のデプロイメントでは、設定に応じて追加のアーティファクトが必要な場合があります。

Rancher 2.12.1 (<https://github.com/rancher/rancher/releases/tag/v2.12.1>)  リリースアセットには、エアギャップインストールに必要なすべてのイメージをリストする `rancher-images.txt` ファイルが含まれています。

コンテナイメージは合計で600個以上あり、結果として得られるCRBイメージは約30GBになります。Rancherのインストールでは、そのリストを最小の動作設定にまで削減します。そこから、デプロイメントに必要なイメージを追加し直すことができます。

定義ファイルを作成し、必要最小限のイメージリストを含めます。

```
apiVersion: 1.3
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword: $6$jHugJNNd3HElGsUZ
$eodjVe4te5ps44SVcWshdfWizrP.xAyd7lCVExazBJ/.v799/WRCBXxfYmunlB02yp1hm/zb4r8EmnrNCF.P/
kubernetes:
  version: v1.33.3+rke2r1
  manifests:
    urls:
      - https://github.com/cert-manager/cert-manager/releases/download/v1.15.3/cert-
manager.crd.yaml
  helm:
    charts:
      - name: rancher
        version: 2.12.1
        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: 1.18.2
    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:v8.0.0
    - name: registry.rancher.com/rancher/compliance-operator:v1.1.0
    - name: registry.rancher.com/rancher/fleet-agent:v0.13.1
    - name: registry.rancher.com/rancher/fleet:v0.13.1
    - name: registry.rancher.com/rancher/hardened-addon-resizer:1.8.23-build20250612
    - name: registry.rancher.com/rancher/hardened-calico:v3.30.2-build20250711
    - name: registry.rancher.com/rancher/hardened-cluster-autoscaler:v1.10.2-build20250611
    - name: registry.rancher.com/rancher/hardened-cni-plugins:v1.7.1-build20250611
    - name: registry.rancher.com/rancher/hardened-coredns:v1.12.2-build20250611
```

- name: registry.rancher.com/rancher/hardened-dns-node-cache:1.26.0-build20250611
- name: registry.rancher.com/rancher/hardened-etcd:v3.5.21-k3s1-build20250612
- name: registry.rancher.com/rancher/hardened-flannel:v0.27.1-build20250710
- name: registry.rancher.com/rancher/hardened-k8s-metrics-server:v0.8.0-build20250704
- name: registry.rancher.com/rancher/hardened-kubernetes:v1.33.3-rke2r1-build20250716
- name: registry.rancher.com/rancher/hardened-multus-cni:v4.2.1-build20250627
- name: registry.rancher.com/rancher/hardened-multus-dynamic-networks-controller:v0.3.7-build20250711
- name: registry.rancher.com/rancher/hardened-multus-thick:v4.2.1-build20250627
- name: registry.rancher.com/rancher/hardened-whereabouts:v0.9.1-build20250704
- name: registry.rancher.com/rancher/k3s-upgrade:v1.33.3-k3s1
- name: registry.rancher.com/rancher/klipper-helm:v0.9.8-build20250709
- name: registry.rancher.com/rancher/klipper-lb:v0.4.13
- name: registry.rancher.com/rancher/kubectrl:v1.33.1
- name: registry.rancher.com/rancher/kuberlr-kubectrl:v5.0.0
- name: registry.rancher.com/rancher/local-path-provisioner:v0.0.31
- name: registry.rancher.com/rancher/machine:v0.15.0-rancher131
- name: registry.rancher.com/rancher/mirrored-cluster-api-controller:v1.10.2
- name: registry.rancher.com/rancher/nginx-ingress-controller:v1.12.4-hardened2
- name: registry.rancher.com/rancher/prom-prometheus:v3.2.1
- name: registry.rancher.com/rancher/prometheus-federator:v4.1.0
- name: registry.rancher.com/rancher/pushprox-client:v0.1.5-rancher2-client
- name: registry.rancher.com/rancher/pushprox-proxy:v0.1.5-rancher2-proxy
- name: registry.rancher.com/rancher/rancher-agent:v2.12.1
- name: registry.rancher.com/rancher/rancher-csp-adapter:v7.0.0
- name: registry.rancher.com/rancher/rancher-webhook:v0.8.1
- name: registry.rancher.com/rancher/rancher:v2.12.1
- name: registry.rancher.com/rancher/remotedialer-proxy:v0.5.0
- name: registry.rancher.com/rancher/rke2-cloud-provider:v1.33.1-0.20250516163953-99d91538b132-build20250612
- name: registry.rancher.com/rancher/rke2-runtime:v1.33.3-rke2r1
- name: registry.rancher.com/rancher/rke2-upgrade:v1.33.3-rke2r1
- name: registry.rancher.com/rancher/scc-operator:v0.1.1
- name: registry.rancher.com/rancher/security-scan:v0.7.1
- name: registry.rancher.com/rancher/shell:v0.5.0
- name: registry.rancher.com/rancher/system-agent-installer-k3s:v1.33.3-k3s1
- name: registry.rancher.com/rancher/system-agent-installer-rke2:v1.33.3-rke2r1
- name: registry.rancher.com/rancher/system-agent:v0.3.13-suc
- name: registry.rancher.com/rancher/system-upgrade-controller:v0.16.0
- name: registry.rancher.com/rancher/ui-plugin-catalog:4.0.3
- 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:v1.5.0
- name: registry.rancher.com/rancher/mirrored-ingress-nginx-kube-webhook-certgen:v1.5.1
- name: registry.rancher.com/rancher/mirrored-ingress-nginx-kube-webhook-certgen:v1.5.2
- name: registry.rancher.com/rancher/mirrored-ingress-nginx-kube-webhook-certgen:v1.5.3


```
- name: registry.rancher.com/rancher/mirrored-ingress-nginx-kube-webhook-  
certgen:v1.6.0
```

600個以上のコンテナイメージの全リストと比較すると、このスリム化されたバージョンには約60個しか含まれておらず、新しいCRBイメージは約7GBになります。

RancherのHelm値も作成する必要があります。

```
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
```



敬告

systemDefaultRegistryをregistry.rancher.comに設定することで、Rancherは、ブート時にCRBイメージ内で起動される組み込みのアーティファクトレジストリ内でイメージを自動的に検索できます。このフィールドを省略すると、ノードでコンテナイメージを見つけられない場合があります。

イメージを構築してみましょう。

```
podman run --rm -it --privileged -v $CONFIG_DIR:/eib \
registry.suse.com/edge/3.4/edge-image-builder:1.3.0 \
build --definition-file eib-iso-definition.yaml
```

出力は次のようになります。

```

Downloading file: dl-manifest-1.yaml 100% |
kB, 12 MB/s)
Pulling selected Helm charts... 100% |
(2/2, 3 it/s)
Generating image customization components...
Identifier ..... [SUCCESS]
Custom Files ..... [SKIPPED]
Time ..... [SKIPPED]
Network ..... [SUCCESS]
Groups ..... [SKIPPED]
Users ..... [SUCCESS]
Proxy ..... [SKIPPED]
Rpm ..... [SKIPPED]
Os Files ..... [SKIPPED]
Systemd ..... [SKIPPED]
Fips ..... [SKIPPED]

```

```
Elemental ..... [SKIPPED]
Suma ..... [SKIPPED]
Populating Embedded Artifact Registry... 100% |
██████████████████████████████████████████████████████████████████| (56/56, 8 it/
min)
Embedded Artifact Registry ... [SUCCESS]
Keymap ..... [SUCCESS]
Configuring Kubernetes component...
The Kubernetes CNI is not explicitly set, defaulting to 'cilium'.
Downloading file: rke2_installer.sh
Downloading file: rke2-images-core.linux-amd64.tar.zst 100% |
██████████████████████████████████████████████████████████████████| (644/644 MB, 29 MB/s)
Downloading file: rke2-images-cilium.linux-amd64.tar.zst 100% |
██████████████████████████████████████████████████████████████████| (400/400 MB, 29 MB/s)
Downloading file: rke2.linux-amd64.tar.gz 100% |
██████████████████████████████████████████████████████████████████| (36/36 MB, 30
MB/s)
Downloading file: sha256sum-amd64.txt 100% |
██████████████████████████████████████████████████████████████████| (4.3/4.3
kB, 29 MB/s)
Kubernetes ..... [SUCCESS]
Certificates ..... [SKIPPED]
Cleanup ..... [SKIPPED]
Building ISO image...
Kernel Params ..... [SKIPPED]
Build complete, the image can be found at: eib-image.iso
```

構築したイメージを使用するノードがプロビジョニングされたら、Rancherのインストールを確認できます。

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

出力は次のようになり、すべてが正常にデプロイされていることがわかります。

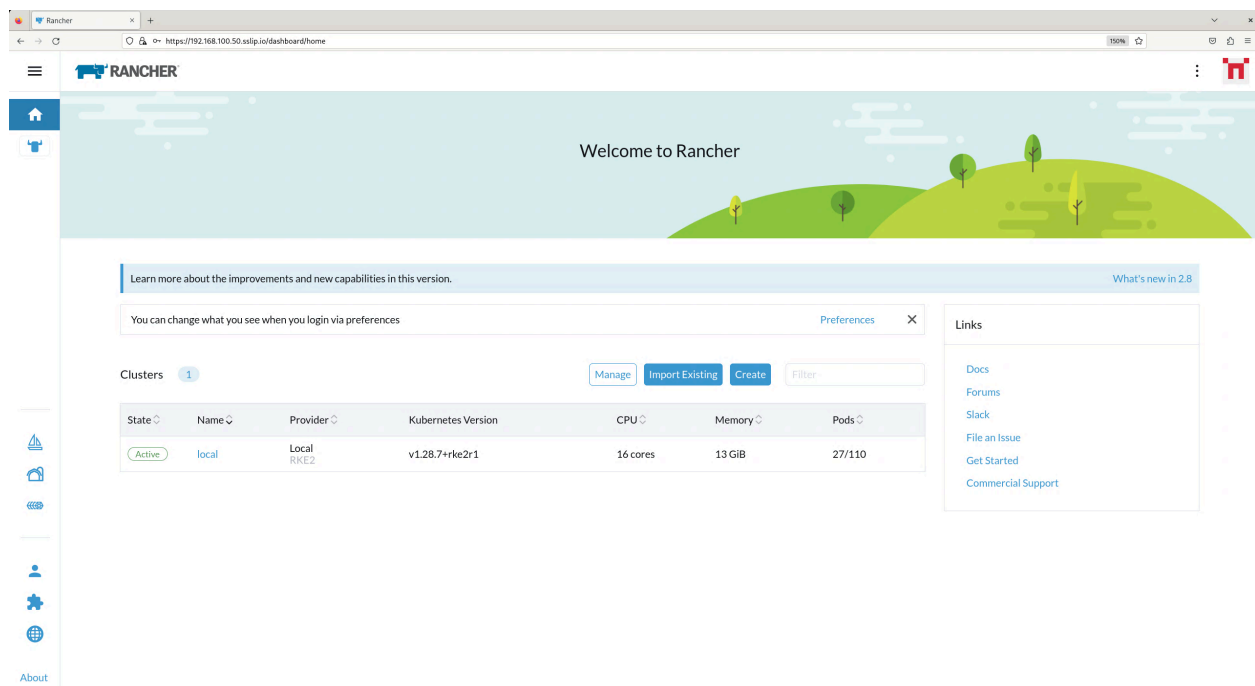
NAME		READY	STATUS	RESTARTS	AGE
pod/helm-operation-6l6ld		0/2	Completed	0	107s
pod/helm-operation-8tk2v		0/2	Completed	0	2m2s
pod/helm-operation-blrrr		0/2	Completed	0	2m49s
pod/helm-operation-hdcmt		0/2	Completed	0	3m19s
pod/helm-operation-m74c7		0/2	Completed	0	97s
pod/helm-operation-qzrz4		0/2	Completed	0	2m30s
pod/helm-operation-s9jh5		0/2	Completed	0	3m
pod/helm-operation-tq7ts		0/2	Completed	0	2m41s
pod/rancher-99d599967-ftjkk		1/1	Running	0	4m15s
pod/rancher-webhook-79798674c5-6w28t		1/1	Running	0	2m27s
pod/system-upgrade-controller-56696956b-trq5c		1/1	Running	0	104s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/rancher	ClusterIP	10.43.255.80	<none>	80/TCP,443/TCP	4m15s
service/rancher-webhook	ClusterIP	10.43.7.238	<none>	443/TCP	2m27s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/rancher	1/1	1	1	4m15s
deployment.apps/rancher-webhook	1/1	1	1	2m27s
deployment.apps/system-upgrade-controller	1/1	1	1	104s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/rancher-99d599967	1	1	1	4m15s
replicaset.apps/rancher-webhook-79798674c5	1	1	1	2m27s
replicaset.apps/system-upgrade-controller-56696956b	1	1	1	104s

また、<https://192.168.100.50.sslip.io>に移動し、以前に設定したadminadminadminパスワードでログインすると、Rancherダッシュボードが表示されます。



27.7 SUSE Securityのインストール

Rancherのインストールとは異なり、SUSE SecurityのインストールではEIBで特別な処理を行う必要はありません。EIBは基盤となるコンポーネントNeuVectorに必要なすべてのイメージを自動的にエアギャップ化します。

定義ファイルを作成します。

```
apiVersion: 1.3
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: v1.33.3+rke2r1
  helm:
    charts:
      - name: neuvector-crd
        version: 107.0.0+up2.8.7
        repositoryName: rancher-charts
        targetNamespace: neuvector
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: neuvector-values.yaml
      - name: neuvector
        version: 107.0.0+up2.8.7
        repositoryName: rancher-charts
        targetNamespace: neuvector
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: neuvector-values.yaml
    repositories:
      - name: rancher-charts
        url: https://charts.rancher.io/

```

NeuVector用のHelm値ファイルも作成します。

```

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

```

イメージを構築してみましょう。

```

podman run --rm -it --privileged -v $CONFIG_DIR:/eib \
registry.suse.com/edge/3.4/edge-image-builder:1.3.0 \
build --definition-file eib-iso-definition.yaml

```

出力は次のようになります。

```

Pulling selected Helm charts... 100% |
(2/2, 4 it/s)
Generating image customization components...
Identifier ..... [SUCCESS]
Custom Files ..... [SKIPPED]
Time ..... [SKIPPED]
Network ..... [SUCCESS]
Groups ..... [SKIPPED]
Users ..... [SUCCESS]
Proxy ..... [SKIPPED]
Rpm ..... [SKIPPED]
Os Files ..... [SKIPPED]
Systemd ..... [SKIPPED]
Fips ..... [SKIPPED]
Elemental ..... [SKIPPED]
Suma ..... [SKIPPED]
Populating Embedded Artifact Registry... 100% |
(5/5, 13 it/min)
Embedded Artifact Registry ... [SUCCESS]
Keymap ..... [SUCCESS]
Configuring Kubernetes component...
The Kubernetes CNI is not explicitly set, defaulting to 'cilium'.
Downloading file: rke2_installer.sh
Kubernetes ..... [SUCCESS]
Certificates ..... [SKIPPED]
Cleanup ..... [SKIPPED]
Building ISO image...
Kernel Params ..... [SKIPPED]
Build complete, the image can be found at: eib-image.iso

```

構築したイメージを使用するノードがプロビジョニングされたら、SUSE Securityのインストールを確認できます。

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

出力は次のようになり、すべてが正常にデプロイされていることがわかります。

NAME	READY	STATUS	RESTARTS	AGE
pod/neuvector-cert-upgrader-job-bxbnz	0/1	Completed	0	3m39s
pod/neuvector-controller-pod-7d854bfdc7-nhxjf	1/1	Running	0	3m44s
pod/neuvector-enforcer-pod-ct8jm	1/1	Running	0	3m44s

NAME	AGE	TYPE	CLUSTER-IP	EXTERNAL-IP
service/neuvector-svc-admission-webhook	3m44s	ClusterIP	10.43.234.241	<none> 443/TCP
service/neuvector-svc-controller	3m44s	ClusterIP	None	<none> 18300/TCP,18301/TCP,18301/UDP

service/neuvector-svc-crd-webhook			ClusterIP	10.43.50.190	<none>	443/	
TCP			3m44s				
NAME			DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE
NODE SELECTOR							
AGE							
daemonset.apps/neuvector-enforcer-pod			1	1	1	1	1
<none>			3m44s				
NAME			READY	UP-TO-DATE	AVAILABLE	AGE	
deployment.apps/neuvector-controller-pod			1/1	1	1	3m44s	
NAME			DESIRED	CURRENT	READY	AGE	
replicaset.apps/neuvector-controller-pod-7d854bfdc7			1	1	1	3m44s	
NAME			SCHEDULE	TIMEZONE	SUSPEND	ACTIVE	LAST
SCHEDULE							
AGE							
cronjob.batch/neuvector-cert-upgrader-pod			0 0 1 1 *	<none>	True	0	
<none>			3m44s				
cronjob.batch/neuvector-updater-pod			0 0 * * *	<none>	False	0	
<none>			3m44s				
NAME			STATUS	COMPLETIONS	DURATION	AGE	
job.batch/neuvector-cert-upgrader-job			Complete	1/1	7s	3m39s	

27.8 SUSE Storageのインストール

Longhornの公式ドキュメント (<https://longhorn.io/docs/1.9.1/deploy/install/airgap/>)⁷には、エアギャップインストールに必要なすべてのイメージをリストしたlonghorn-images.txtファイルが含まれています。定義ファイルには、Rancherコンテナレジストリからのミラー化された対応するイメージを含めます。作成してみましょう。

```
apiVersion: 1.3
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword: $6$jHugJNNd3HElGsUZ
$eodjVe4te5ps44SVcWshdfWizrP.xAyd7lCVEXazBJ/.v799/WRCBXxfYmunlB02yp1hm/zb4r8EmnrrNCF.P/
packages:
  sccRegistrationCode: [reg-code]
  packageList:
    - open-iscsi
kubernetes:
```

```

version: v1.33.3+rke2r1
helm:
  charts:
    - name: longhorn
      repositoryName: longhorn
      targetNamespace: longhorn-system
      createNamespace: true
      version: 107.0.0+up1.9.1
    - name: longhorn-crd
      repositoryName: longhorn
      targetNamespace: longhorn-system
      createNamespace: true
      installationNamespace: kube-system
      version: 107.0.0+up1.9.1
  repositories:
    - name: longhorn
      url: https://charts.rancher.io
embeddedArtifactRegistry:
  images:
    - name: registry.rancher.com/rancher/mirrored-longhornio-csi-attacher:v4.9.0-20250709
    - name: registry.rancher.com/rancher/mirrored-longhornio-csi-node-driver-
  registrar:v2.14.0-20250709
    - name: registry.rancher.com/rancher/mirrored-longhornio-csi-
  provisioner:v5.3.0-20250709
    - name: registry.rancher.com/rancher/mirrored-longhornio-csi-resizer:v1.14.0-20250709
    - name: registry.rancher.com/rancher/mirrored-longhornio-csi-
  snapshotter:v8.3.0-20250709
    - name: registry.rancher.com/rancher/mirrored-longhornio-
  livenessprobe:v2.16.0-20250709
    - name: registry.rancher.com/rancher/mirrored-longhornio-longhorn-engine:v1.9.1
    - name: registry.rancher.com/rancher/mirrored-longhornio-longhorn-instance-
  manager:v1.9.1
    - name: registry.rancher.com/rancher/mirrored-longhornio-longhorn-manager:v1.9.1
    - name: registry.rancher.com/rancher/mirrored-longhornio-longhorn-share-manager:v1.9.1
    - name: registry.rancher.com/rancher/mirrored-longhornio-longhorn-ui:v1.9.1
    - name: registry.suse.com/rancher/mirrored-longhornio-support-bundle-kit:v0.0.52
    - name: registry.suse.com/rancher/mirrored-longhornio-longhorn-cli:v1.9.1

```



注記

定義ファイルには`open-iscsi`パッケージがリストされていることに気づくでしょう。

これは、LonghornがKubernetesに永続ボリュームを提供するために、さまざまなノードで実行されている`iscsiadm`デーモンに依存しているために必要です。

イメージを構築してみましょう。

```

podman run --rm -it --privileged -v $CONFIG_DIR:/eib \
registry.suse.com/edge/3.4/edge-image-builder:1.3.0 \

```

```
build --definition-file eib-iso-definition.yaml
```

出力は次のようになります。

```
Setting up Podman API listener...
Pulling selected Helm charts... 100% |
(2/2, 3 it/s)
Generating image customization components...
Identifier ..... [SUCCESS]
Custom Files ..... [SKIPPED]
Time ..... [SKIPPED]
Network ..... [SUCCESS]
Groups ..... [SKIPPED]
Users ..... [SUCCESS]
Proxy ..... [SKIPPED]
Resolving package dependencies...
Rpm ..... [SUCCESS]
Os Files ..... [SKIPPED]
Systemd ..... [SKIPPED]
Fips ..... [SKIPPED]
Elemental ..... [SKIPPED]
Suma ..... [SKIPPED]
Populating Embedded Artifact Registry... 100% |
(15/15, 20956 it/s)
Embedded Artifact Registry ... [SUCCESS]
Keymap ..... [SUCCESS]
Configuring Kubernetes component...
The Kubernetes CNI is not explicitly set, defaulting to 'cilium'.
Downloading file: rke2_installer.sh
Downloading file: rke2-images-core.linux-amd64.tar.zst 100% (782/782 MB, 108 MB/s)
Downloading file: rke2-images-cilium.linux-amd64.tar.zst 100% (367/367 MB, 104 MB/s)
Downloading file: rke2.linux-amd64.tar.gz 100% (34/34 MB, 108 MB/s)
Downloading file: sha256sum-amd64.txt 100% (3.9/3.9 kB, 7.5 MB/s)
Kubernetes ..... [SUCCESS]
Certificates ..... [SKIPPED]
Cleanup ..... [SKIPPED]
Building ISO image...
Kernel Params ..... [SKIPPED]
Build complete, the image can be found at: eib-image.iso
```

構築したイメージを使用するノードがプロビジョニングされたら、Longhornのインストールを確認できます。

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

出力は次のようになり、すべてが正常にデプロイされていることがわかります。

NAME	READY	STATUS	RESTARTS	AGE
pod/csi-attacher-787fd9c6c8-sf42d	1/1	Running	0	2m28s

pod/csi-attacher-787fd9c6c8-tb82p	1/1	Running	0	2m28s
pod/csi-attacher-787fd9c6c8-zhc6s	1/1	Running	0	2m28s
pod/csi-provisioner-74486b95c6-b2v9s	1/1	Running	0	2m28s
pod/csi-provisioner-74486b95c6-hwllt	1/1	Running	0	2m28s
pod/csi-provisioner-74486b95c6-mlrpk	1/1	Running	0	2m28s
pod/csi-resizer-859d4557fd-t54zk	1/1	Running	0	2m28s
pod/csi-resizer-859d4557fd-vdt5d	1/1	Running	0	2m28s
pod/csi-resizer-859d4557fd-x9kh4	1/1	Running	0	2m28s
pod/csi-snapshotter-6f69c6c8cc-r62gr	1/1	Running	0	2m28s
pod/csi-snapshotter-6f69c6c8cc-vrwjn	1/1	Running	0	2m28s
pod/csi-snapshotter-6f69c6c8cc-z65nb	1/1	Running	0	2m28s
pod/engine-image-ei-4623b511-9vhkb	1/1	Running	0	3m13s
pod/instance-manager-6f95fd57d4a4cd0459e469d75a300552	1/1	Running	0	2m43s
pod/longhorn-csi-plugin-gx98x	3/3	Running	0	2m28s
pod/longhorn-driver-deployer-55f9c88499-fbm6q	1/1	Running	0	3m28s
pod/longhorn-manager-dpdp7	2/2	Running	0	3m28s
pod/longhorn-ui-59c85fcf94-gg5hq	1/1	Running	0	3m28s
pod/longhorn-ui-59c85fcf94-s49jc	1/1	Running	0	3m28s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
AGE				
service/longhorn-admission-webhook 3m28s	ClusterIP	10.43.77.89	<none>	9502/TCP
service/longhorn-backend 3m28s	ClusterIP	10.43.56.17	<none>	9500/TCP
service/longhorn-conversion-webhook 3m28s	ClusterIP	10.43.54.73	<none>	9501/TCP
service/longhorn-frontend 3m28s	ClusterIP	10.43.22.82	<none>	80/TCP
service/longhorn-recovery-backend 3m28s	ClusterIP	10.43.45.143	<none>	9503/TCP

NAME	DESIRED	CURRENT	READY	UP-TO-DATE
AVAILABLE				
nodeSelector				
AGE				
daemonset.apps/engine-image-ei-4623b511 <none> 3m13s	1	1	1	1
daemonset.apps/longhorn-csi-plugin <none> 2m28s	1	1	1	1
daemonset.apps/longhorn-manager <none> 3m28s	1	1	1	1

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/csi-attacher	3/3	3	3	2m28s
deployment.apps/csi-provisioner	3/3	3	3	2m28s
deployment.apps/csi-resizer	3/3	3	3	2m28s
deployment.apps/csi-snapshotter	3/3	3	3	2m28s
deployment.apps/longhorn-driver-deployer	1/1	1	1	3m28s
deployment.apps/longhorn-ui	2/2	2	2	3m28s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/csi-attacher-787fd9c6c8	3	3	3	2m28s

replicaset.apps/csi-provisioner-74486b95c6	3	3	3	2m28s
replicaset.apps/csi-resizer-859d4557fd	3	3	3	2m28s
replicaset.apps/csi-snapshotter-6f69c6c8cc	3	3	3	2m28s
replicaset.apps/longhorn-driver-deployer-55f9c88499	1	1	1	3m28s
replicaset.apps/longhorn-ui-59c85fcf94	2	2	2	3m28s

27.9 KubeVirtとCDIのインストール

KubeVirtとCDIの両方のHelmチャートでインストールされるのは、それぞれのオペレータのみです。残りのシステムのデプロイはオペレータに任されています。つまり、必要なコンテナイメージすべてを定義ファイルに含める必要があります。作成してみましょう。

```

apiVersion: 1.3
image:
  imageType: iso
  arch: x86_64
  baseImage: slemicro.iso
  outputImageName: eib-image.iso
operatingSystem:
  users:
    - username: root
      encryptedPassword: $6$jHugJNNd3HElGsUZ
$eodjVe4te5ps44SVcWshdfWizrP.xAyd7lCVEXazBJ/.v799/WRCBXxfYmunlB02yp1hm/zb4r8EmnrrNCF.P/
kubernetes:
  version: v1.33.3+rke2r1
  helm:
    charts:
      - name: kubevirt
        repositoryName: suse-edge
        version: 304.0.1+up0.6.0
        targetNamespace: kubevirt-system
        createNamespace: true
        installationNamespace: kube-system
      - name: cdi
        repositoryName: suse-edge
        version: 304.0.1+up0.6.0
        targetNamespace: cdi-system
        createNamespace: true
        installationNamespace: kube-system
    repositories:
      - name: suse-edge
        url: oci://registry.suse.com/edge/charts
embeddedArtifactRegistry:
  images:
    - name: registry.suse.com/suse/sles/15.7/cdi-apiserver:1.62.0-150700.9.3.1
    - name: registry.suse.com/suse/sles/15.7/cdi-controller:1.62.0-150700.9.3.1
    - name: registry.suse.com/suse/sles/15.7/cdi-importer:1.62.0-150700.9.3.1

```

```
- name: registry.suse.com/suse/sles/15.7/cdi-uploadproxy:1.62.0-150700.9.3.1
- name: registry.suse.com/suse/sles/15.7/cdi-uploadserver:1.62.0-150700.9.3.1
- name: registry.suse.com/suse/sles/15.7/cdi-cloner:1.62.0-150700.9.3.1
- name: registry.suse.com/suse/sles/15.7/virt-api:1.5.2-150700.3.5.2
- name: registry.suse.com/suse/sles/15.7/virt-controller:1.5.2-150700.3.5.2
- name: registry.suse.com/suse/sles/15.7/virt-handler:1.5.2-150700.3.5.2
- name: registry.suse.com/suse/sles/15.7/virt-launcher:1.5.2-150700.3.5.2
- name: registry.suse.com/suse/sles/15.7/virt-exportproxy:1.5.2-150700.3.5.2
- name: registry.suse.com/suse/sles/15.7/virt-exportserver:1.5.2-150700.3.5.2
```

イメージを構築してみましょう。

```
podman run --rm -it --privileged -v $CONFIG_DIR:/eib \
registry.suse.com/edge/3.4/edge-image-builder:1.3.0 \
build --definition-file eib-iso-definition.yaml
```

出力は次のようになります。

```
Pulling selected Helm charts... 100% |
(2/2, 48 it/min)
Generating image customization components...
Identifier ..... [SUCCESS]
Custom Files ..... [SKIPPED]
Time ..... [SKIPPED]
Network ..... [SUCCESS]
Groups ..... [SKIPPED]
Users ..... [SUCCESS]
Proxy ..... [SKIPPED]
Rpm ..... [SKIPPED]
Os Files ..... [SKIPPED]
Systemd ..... [SKIPPED]
Fips ..... [SKIPPED]
Elemental ..... [SKIPPED]
Suma ..... [SKIPPED]
Populating Embedded Artifact Registry... 100% |
(15/15, 4 it/min)
Embedded Artifact Registry ... [SUCCESS]
Keymap ..... [SUCCESS]
Configuring Kubernetes component...
The Kubernetes CNI is not explicitly set, defaulting to 'cilium'.
Downloading file: rke2_installer.sh
Kubernetes ..... [SUCCESS]
Certificates ..... [SKIPPED]
Cleanup ..... [SKIPPED]
Building ISO image...
Kernel Params ..... [SKIPPED]
Build complete, the image can be found at: eib-image.iso
```

構築したイメージを使用するノードがプロビジョニングされたら、KubeVirtとCDIの両方のインストールを確認できます。

KubeVirtを確認します。

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

出力は次のようになり、すべてが正常にデプロイされていることがわかります。

NAME	READY	STATUS	RESTARTS	AGE
pod/virt-api-59cb997648-mmt67	1/1	Running	0	2m34s
pod/virt-controller-69786b785-7cc96	1/1	Running	0	2m8s
pod/virt-controller-69786b785-wq2dz	1/1	Running	0	2m8s
pod/virt-handler-2l4dm	1/1	Running	0	2m8s
pod/virt-operator-7c444cff46-nps4l	1/1	Running	0	3m1s
pod/virt-operator-7c444cff46-r25xq	1/1	Running	0	3m1s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
AGE				
service/kubevirt-operator-webhook	ClusterIP	10.43.167.109	<none>	443/TCP
2m36s				
service/kubevirt-prometheus-metrics	ClusterIP	None	<none>	443/TCP
2m36s				
service/virt-api	ClusterIP	10.43.18.202	<none>	443/TCP
2m36s				
service/virt-exportproxy	ClusterIP	10.43.142.188	<none>	443/TCP
2m36s				

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE
SELECTOR						
AGE						
daemonset.apps/virt-handler	1	1	1	1	1	
kubernetes.io/os=linux						
2m8s						

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/virt-api	1/1	1	1	2m34s
deployment.apps/virt-controller	2/2	2	2	2m8s
deployment.apps/virt-operator	2/2	2	2	3m1s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/virt-api-59cb997648	1	1	1	2m34s
replicaset.apps/virt-controller-69786b785	2	2	2	2m8s
replicaset.apps/virt-operator-7c444cff46	2	2	2	3m1s

NAME	AGE	PHASE
kubevirt.kubevirt.io/kubevirt	3m1s	Deployed

CDIを確認します。

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

出力は次のようになり、すべてが正常にデプロイされていることがわかります。

NAME	READY	STATUS	RESTARTS	AGE
pod/cdi-apiserver-5598c9bf47-pqfxw	1/1	Running	0	3m44s
pod/cdi-deployment-7cbc5db7f8-g46z7	1/1	Running	0	3m44s
pod/cdi-operator-777c865745-2qcnj	1/1	Running	0	3m48s
pod/cdi-uploadproxy-646f4cd7f7-fzkv7	1/1	Running	0	3m44s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/cdi-api	ClusterIP	10.43.2.224	<none>	443/TCP	3m44s
service/cdi-prometheus-metrics	ClusterIP	10.43.237.13	<none>	8080/TCP	3m44s
service/cdi-uploadproxy	ClusterIP	10.43.114.91	<none>	443/TCP	3m44s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/cdi-apiserver	1/1	1	1	3m44s
deployment.apps/cdi-deployment	1/1	1	1	3m44s
deployment.apps/cdi-operator	1/1	1	1	3m48s
deployment.apps/cdi-uploadproxy	1/1	1	1	3m44s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/cdi-apiserver-5598c9bf47	1	1	1	3m44s
replicaset.apps/cdi-deployment-7cbc5db7f8	1	1	1	3m44s
replicaset.apps/cdi-operator-777c865745	1	1	1	3m48s
replicaset.apps/cdi-uploadproxy-646f4cd7f7	1	1	1	3m44s

27.10 トラブルシューティング

イメージの構築中に問題が発生した場合、またはプロセスをさらにテストおよびデバッグしたい場合は、[アップストリームドキュメント \(https://github.com/suse-edge/edge-image-builder/tree/release-1.1/docs\)](https://github.com/suse-edge/edge-image-builder/tree/release-1.1/docs) を参照してください。

28 Kiwiを使用したSUSE Linux Microの更新イメージの構築

このセクションでは、SUSE Linux Microの更新イメージを生成して、Edge Image BuilderまたはCluster API (CAPI) + Metal³で使用するか、あるいはディスクイメージをブロックデバイスに直接書き込む方法について説明します。このプロセスは、初期のシステムブートイメージに最新のパッチを含める必要がある状況や(インストール後のパッチ転送を最小化するため)、CAPIを使用するシナリオにおいて、ホストをインプレースでアップグレードするのではなく、新しいイメージを使用してオペレーティングシステムを再インストールする場合に役立ちます。

このプロセスでは、[Kiwi \(https://osinside.github.io/kiwi/\)](https://osinside.github.io/kiwi/) を利用してイメージの構築を行います。SUSE Edgeには、組み込みのヘルパーユーティリティによって全体的なプロセスが簡素化されるコンテナ化バージョンが付属しており、必要なターゲット**プロファイル**を指定できます。このプロファイルで、必要な出力イメージのタイプを定義します。以下に一般的なものを示します。

- 「**Base**」 - 縮小版のパッケージセットが含まれるSUSE Linux Microディスクイメージ (podmanを含む)。
- 「**Base-SelfInstall**」 - 上の「Base」に基づくセルフインストールイメージ。
- 「**Base-RT**」 - 上の「Base」と同じですが、代わりにリアルタイム(rt)カーネルを使用します。
- 「**Base-RT-SelfInstall**」 - 上の「Base-RT」に基づくセルフインストールイメージ。
- 「**Default**」 - 上の「Base」に基づくSUSE Linux Microディスクイメージですが、仮想化スタック、Cockpit、salt-minionなどのツールがいくつか追加されています。
- 「**Default-SelfInstall**」 - 上の「Default」に基づくセルフインストールイメージ。

詳細については、[SUSE Linux Micro 6.1 \(https://documentation.suse.com/sle-micro/6.1/html/Micro-deployment-images/index.html#alp-images-installer-type\)](https://documentation.suse.com/sle-micro/6.1/html/Micro-deployment-images/index.html#alp-images-installer-type) のドキュメントを参照してください。

このプロセスはAMD64/Intel 64およびAArch64のどちらのアーキテクチャでも動作しますが、両方のアーキテクチャですべてのイメージプロファイルが利用できるわけではありません。たとえば、SUSE Edge 3.4では、SUSE Linux Micro 6.1が使用されており、リアルタイムカーネルが含まれるプロファイル(つまり「Base-RT」または「Base-RT-SelfInstall」)は現在のところAArch64では利用できません。



注記

構築するイメージと同じアーキテクチャの構築ホストを使用する必要があります。つまり、AArch64のイメージを構築するにはAArch64の構築ホストを使用する必要があります。AMD64/Intel 64の場合も同様です。クロス構築は現在のところサポートされていません。

28.1 前提条件

Kiwi Image Builderには以下が必要です。

- 構築するイメージと同じアーキテクチャのSUSE Linux Micro 6.1ホスト(「構築システム」)。
- 構築システムはSUSEConnectを介して登録済みである必要があります(この登録を使用してSUSEリポジトリから最新のパッケージをプルします)。
- 必要なパッケージをプルするために使用できるインターネット接続。プロキシ経由で接続している場合、構築ホストを再設定する必要があります。
- 構築ホスト上でSELinuxを無効化する必要があります(コンテナ内でSELinuxのラベル付けが実行され、これがホストのポリシーと競合する可能性があるためです)。
- コンテナイメージ、構築ホスト、および作成される出力イメージを格納するための10GB以上の空きディスク容量。

28.2 はじめに

特定の制限のため、現在はSELinuxを無効化する必要があります。SUSE Linux Micro 6.1イメージ構築ホストに接続し、SELinuxを確実に無効化します。

```
# setenforce 0
```

作成されるイメージを保存するためにKiwi構築コンテナと共有する出力ディレクトリを作成します。

```
# mkdir ~/output
```

SUSEレジストリから最新のKiwi Builderイメージをプルします。

```
# podman pull registry.suse.com/edge/3.4/kiwi-builder:10.2.12.0
```

(...)

28.3 デフォルトイメージの構築

コンテナイメージの実行時に引数が指定されていない場合、これがKiwiイメージコンテナのデフォルトの動作です。以下のコマンドは、次の2つのディレクトリをコンテナにマップした状態でpodmanを実行します。

- 基になるホストのSUSE Linux Microパッケージリポジトリディレクトリである`/etc/zypp/repos.d`。
- 上記で作成される出力の`~/output`ディレクトリ。

Kiwiイメージコンテナでは、`build-image`ヘルパースクリプトを次のように実行する必要があります。

```
# podman run --privileged -v /etc/zypp/repos.d:/etc/zypp/repos.d -v ~/output:/tmp/output \
-it registry.suse.com/edge/3.4/kiwi-builder:10.2.12.0 build-image
(...)
```



注記

このスクリプトを初めて実行すると、開始直後にスクリプトが失敗し、「**ERROR: Early loop device test failed, please retry the container run. (エラー: ループデバイスの早期テストが失敗しました。もう一度コンテナの実行を試してください。)**」というエラーが発生すると予想されます。これは、基になるホストシステム上に作成されるループデバイスがすぐにはコンテナイメージ内に表示されないことを示す症状です。単にコマンドを再実行すれば、問題なく続行されるはずです。

数分後、ローカルのoutputディレクトリ内にイメージを確認できます。

```
(...)
INFO: Image build successful, generated images are available in the 'output' directory.

# ls -l output/
SLE-Micro.x86_64-6.1.changes
SLE-Micro.x86_64-6.1.packages
SLE-Micro.x86_64-6.1.raw
SLE-Micro.x86_64-6.1.verified
build
kiwi.result
```


28.4 他のプロファイルを使用したイメージの構築

異なるイメージプロファイルを構築するには、Kiwiコンテナイメージのヘルパースクリプトで「**-p**」コマンドオプションを使用します。たとえば、「**Default-SelfInstall**」のISOイメージを構築するには、次のコマンドを使用します。

```
# podman run --privileged -v /etc/zypp/repos.d:/micro-sdk/repos/ -v ~/output:/tmp/output \
  -it registry.suse.com/edge/3.4/kiwi-builder:10.2.12.0 build-image -p Default-
  SelfInstall
  (...)
```



注記

データの消失を防ぐため、Kiwiは、outputディレクトリにイメージが存在する場合、実行を拒否します。rm -f output/*を使用して続行する前に、outputディレクトリの内容を削除する必要があります。

または、リアルタイムカーネル(「**kernel-rt**」)を使用してSelfInstall ISOイメージを構築するには、次のコマンドを使用します。

```
# podman run --privileged -v /etc/zypp/repos.d:/micro-sdk/repos/ -v ~/output:/tmp/output \
  -it registry.suse.com/edge/3.4/kiwi-builder:10.2.12.0 build-image -p Base-RT-
  SelfInstall
  (...)
```

28.5 大きいセクタサイズを使用したイメージの構築

ハードウェアによっては、大きいセクタサイズ、つまり標準の512バイトではなく**4096バイト**のイメージが必要になることがあります。コンテナ化されたKiwi Builderでは、「**-b**」パラメータを指定することで、大きいブロックサイズのイメージを生成できます。たとえば、大きいセクタサイズを使用して「**Default-SelfInstall**」イメージを構築するには、次のコマンドを使用します。

```
# podman run --privileged -v /etc/zypp/repos.d:/micro-sdk/repos/ -v ~/output:/tmp/output \
  -it registry.suse.com/edge/3.4/kiwi-builder:10.2.12.0 build-image -p Default-
  SelfInstall -b
```

(...)

28.6 カスタムのKiwiイメージ定義ファイルの使用

高度なユースケースでは、カスタムのKiwiイメージ定義ファイル(SL-Micro.kiwi)を、必要な構築後スクリプトとともに使用できます。このためには、SUSE Edgeチームによって事前にパッケージ化されているデフォルトの定義を上書きする必要があります。

新しいディレクトリを作成し、ヘルパースクリプトが参照するコンテナイメージにマップします(/micro-sdk/defs)。

```
# mkdir ~/mydefs/
# cp /path/to/SL-Micro.kiwi ~/mydefs/
# cp /path/to/config.sh ~/mydefs/
# podman run --privileged -v /etc/zypp/repos.d:/micro-sdk/repos/ -v ~/output:/tmp/output -v ~/mydefs:/micro-sdk/defs/ \
  -it registry.suse.com/edge/3.4/kiwi-builder:10.2.12.0 build-image
(...)
```



警告

これは、高度なユースケースにおいて、サポート性に問題が生じる可能性がある場合のみ必要です。詳細なアドバイスとガイダンスについては、SUSEの担当者にお問い合わせください。

コンテナに含まれるデフォルトのKiwiイメージ定義ファイルを取得するには、次のコマンドを使用できます。

```
$ podman create --name kiwi-builder registry.suse.com/edge/3.4/kiwi-builder:10.2.12.0
$ podman cp kiwi-builder:/micro-sdk/defs/SL-Micro.kiwi .
$ podman cp kiwi-builder:/micro-sdk/defs/SL-Micro.kiwi.4096 .
$ podman rm kiwi-builder
$ ls ./SL-Micro.*
(...)
```

29 ClusterClassを使用したダウンストリームクラスタのデプロイ

29.1 はじめに

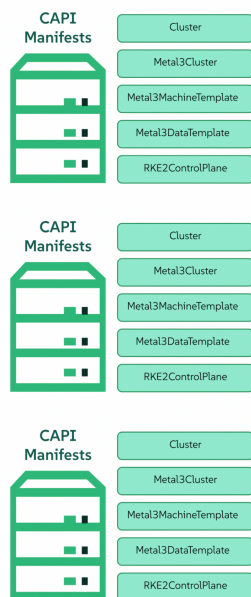
Kubernetesクラスタのプロビジョニングは、クラスタコンポーネントの設定に深い専門知識が求められる複雑な作業です。設定が複雑化したり、さまざまなプロバイダからの要求により多数のプロバイダ固有のリソース定義が導入されたりすると、クラスタの作成が困難に感じられるかもしれません。幸いなことに、Kubernetes Cluster API (CAPI)は、ClusterClassによってさらに強化された、より洗練された宣言型アプローチを提供します。この機能はテンプレート駆動型モデルを導入し、複雑さをカプセル化して一貫性を促進する再利用可能なクラスタクラスを定義できます。

29.2 ClusterClassとは

CAPIプロジェクトは、クラスタのインスタンス化にテンプレートベースの手法を採用することで、Kubernetesクラスタのライフサイクル管理におけるパラダイムシフトとしてClusterClass機能を導入しました。ユーザがクラスタごとにリソースを個別に定義する代わりに、ClusterClassを定義することで、ClusterClassは包括的で再利用可能なブループリントとして機能します。この抽象データ型表現は、Kubernetesクラスタの望ましい状態と設定をカプセル化し、定義された仕様に準拠した複数のクラスタを迅速かつ一貫して作成できるようにします。この抽象データ型により、設定の負担が軽減され、より管理しやすいデプロイメントマニフェストが実現されます。つまり、ワークロードクラスタのコアコンポーネントはクラスレベルで定義されるため、ユーザはこれらのテンプレートをKubernetesクラスタのフレーバーとして使用し、クラスタのプロビジョニングに1回または複数回再利用できるようになります。ClusterClassの実装は、従来の大規模なCAPI管理に固有の課題に対処するいくつかの重要な利点をもたらします。

- 複雑さとYAMLの冗長性を大幅に削減
- 最適化された保守と更新プロセス
- デプロイメント全体における一貫性と標準化の強化
- スケーラビリティと自動化機能の向上
- 宣言型管理と堅牢なバージョン管理

CAPI deployments

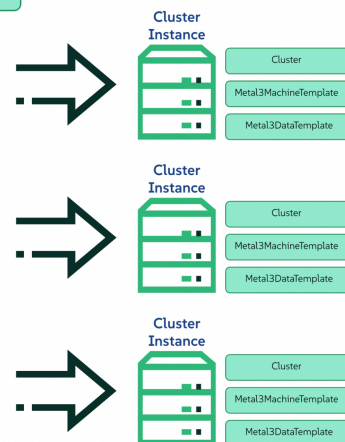


Create Once

ClusterClass



Deploy Many



29.3 現在のCAPIプロビジョニングファイルの例

Cluster API (CAPI)とRKE2プロバイダを活用するKubernetesクラスタのデプロイメントには、いくつかのカスタムリソースの定義が必要です。これらのリソースは、クラスタとその基盤となるインフラストラクチャの望ましい状態を定義し、CAPIがプロビジョニングと管理のライフサイクルをオーケストレーションできるようにします。以下のコードスニペットは設定する必要のあるリソースタイプを示しています。

- **Cluster:** このリソースは、ノード間の通信とサービスディスカバリを制御するネットワークトポロジを含む、高レベルな設定をカプセル化します。さらに、コントロールプレーン仕様と指定されたインフラストラクチャプロバイダリソースとの重要なリンクを確立し、これにより望ましいクラスタアーキテクチャと、そのプロビジョニング対象となる基盤インフラストラクチャについてCAPIに通知します。
- **Metal3Cluster:** このリソースは、Metal3に固有のインフラストラクチャレベルの属性を定義します。たとえば、Kubernetes APIサーバにアクセスできる外部エンドポイントなど。
- **RKE2ControlPlane:** RKE2ControlPlaneリソースは、クラスタのコントロールプレーンノードの特性と動作を定義します。この仕様では、コントロールプレーンレプリカの必要な数(高可用性と耐障害性を確保するために重要)、特定のKubernetesディストリビューションのバージョン(選択したRKE2リリースと一致)、コントロールプレーンコン

ポーネントへの更新展開戦略などのパラメータが設定されます。さらに、このリソースはクラスタ内で使用されるContainer Network Interface (CNI)を指定し、エージェント固有の設定の注入を容易にします。これには多くの場合、コントロールプレーンノード上のRKE2エージェントのシームレスで自動化されたプロビジョニングのためにIgnitionが活用されます。

- **Metal3MachineTemplate:** このリソースは、使用されるイメージを定義するKubernetesクラスタのワーカーノードを形成する、個々のコンピューティングインスタンス作成のブループリントとして機能します。
- **Metal3DataTemplate:** Metal3MachineTemplateを補完するMetal3DataTemplateリソースは、追加のメタデータを新たにプロビジョニングされたマシンインスタンスに指定できるようにします。

```
---
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: emea-spa-cluster-3
  namespace: emea-spa
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - 192.168.0.0/18
    services:
      cidrBlocks:
        - 10.96.0.0/12
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: RKE2ControlPlane
    name: emea-spa-cluster-3
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3Cluster
    name: emea-spa-cluster-3
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3Cluster
metadata:
  name: emea-spa-cluster-3
  namespace: emea-spa
spec:
  controlPlaneEndpoint:
    host: 192.168.122.203
    port: 6443
  noCloudProvider: true
---
```

```

apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
metadata:
  name: emea-spa-cluster-3
  namespace: emea-spa
spec:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3MachineTemplate
    name: emea-spa-cluster-3
  replicas: 1
  version: v1.33.3+rke2r1
  rolloutStrategy:
    type: "RollingUpdate"
    rollingUpdate:
      maxSurge: 1
  registrationMethod: "control-plane-endpoint"
  registrationAddress: 192.168.122.203
  serverConfig:
    cni: cilium
    cniMultusEnable: true
    tlsSan:
      - 192.168.122.203
      - https://192.168.122.203.sslip.io
  agentConfig:
    format: ignition
    additionalUserData:
      config: |
        variant: fcos
        version: 1.4.0
        storage:
          files:
            - path: /var/lib/rancher/rke2/server/manifests/endpoint-copier-operator.yaml
              overwrite: true
              contents:
                inline: |
                  apiVersion: helm.cattle.io/v1
                  kind: HelmChart
                  metadata:
                    name: endpoint-copier-operator
                    namespace: kube-system
                  spec:
                    chart: oci://registry.suse.com/edge/charts/endpoint-copier-operator
                    targetNamespace: endpoint-copier-operator
                    version: 304.0.1+up0.3.0
                    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/charts/metallb
      targetNamespace: metallb-system
      version: 304.0.0+up0.14.9
      createNamespace: true

- path: /var/lib/rancher/rke2/server/manifests/metallb-cr.yaml
  overwrite: true
  contents:
    inline: |
      apiVersion: metallb.io/v1beta1
      kind: IPAddressPool
      metadata:
        name: kubernetes-vip-ip-pool
        namespace: metallb-system
      spec:
        addresses:
          - 192.168.122.203/32
        serviceAllocation:
          priority: 100
          namespaces:
            - default
          serviceSelectors:
            - matchExpressions:
                - {key: "serviceType", operator: In, values: [kubernetes-vip]}
      ---
      apiVersion: metallb.io/v1beta1
      kind: L2Advertisement
      metadata:
        name: ip-pool-l2-adv
        namespace: metallb-system
      spec:
        ipAddressPools:
          - kubernetes-vip-ip-pool
- path: /var/lib/rancher/rke2/server/manifests/endpoint-svc.yaml
  overwrite: true
  contents:
    inline: |
      apiVersion: v1
      kind: Service
      metadata:
        name: kubernetes-vip
        namespace: default
        labels:
          serviceType: kubernetes-vip
      spec:
        ports:

```

```

      - name: rke2-api
        port: 9345
        protocol: TCP
        targetPort: 9345
      - name: k8s-api
        port: 6443
        protocol: TCP
        targetPort: 6443
      type: LoadBalancer
systemd:
  units:
    - name: rke2-preinstall.service
      enabled: true
      contents: |
        [Unit]
        Description=rke2-preinstall
        Wants=network-online.target
        Before=rke2-install.service
        ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
        [Service]
        Type=oneshot
        User=root
        ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
        ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/$(jq -r .uuid /mnt/
openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
        ExecStart=/bin/sh -c "echo \"node-name: $(jq -r .name /mnt/openstack/
latest/meta_data.json)\" >> /etc/rancher/rke2/config.yaml"
        ExecStartPost=/bin/sh -c "umount /mnt"
        [Install]
        WantedBy=multi-user.target
  kubelet:
    extraArgs:
      - provider-id=metal3://BAREMETALHOST_UUID
    nodeName: "localhost.localdomain"
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3MachineTemplate
metadata:
  name: emea-spa-cluster-3
  namespace: emea-spa
spec:
  nodeReuse: True
  template:
    spec:
      automatedCleaningMode: metadata
      dataTemplate:
        name: emea-spa-cluster-3
      hostSelector:
        matchLabels:
          cluster-role: control-plane
          deploy-region: emea-spa

```



```

        node: group-3
      image:
        checksum: http://fileserver.local:8080/eibimage-downstream-cluster.raw.sha256
        checksumType: sha256
        format: raw
        url: http://fileserver.local:8080/eibimage-downstream-cluster.raw
    ---
  apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
  kind: Metal3DataTemplate
  metadata:
    name: emea-spa-cluster-3
    namespace: emea-spa
  spec:
    clusterName: emea-spa-cluster-3
    metaData:
      objectNames:
        - key: name
          object: machine
        - key: local-hostname
          object: machine
        - key: local_hostname
          object: machine

```

29.4 CAPIプロビジョニングファイルのClusterClassへの変換

29.4.1 ClusterClass定義

以下のコードは、特定のタイプのKubernetesクラスタを一貫してデプロイするための宣言型テンプレートである、ClusterClassリソースを定義します。この仕様には、共通のインフラストラクチャとコントロールプレーンの設定が含まれており、クラスタフリート全体にわたる効率的なプロビジョニングと統一されたライフサイクル管理が可能になります。以下のClusterClassの例にはいくつかの変数があり、実際の値を使用してクラスタインスタンス化プロセス中に置き換えられます。例では次の変数が使用されています。

- controlPlaneMachineTemplate: 使用されるControlPlane Machine Template参照を定義する名前です
- controlPlaneEndpointHost: コントロールプレーンエンドポイントのホスト名またはIPアドレスです
- tlsSan: コントロールプレーンエンドポイントのTLSサブジェクト代替名です

ClusterClass定義ファイルは、次の3つのリソースに基づいて定義されます。

- **ClusterClass:** このリソースは、コントロールプレーンとインフラストラクチャテンプレートを含むClusterClass定義全体をカプセル化します。また、インスタンス化プロセス中に置き換えられる変数のリストも含まれます。
- **RKE2ControlPlaneTemplate:** このリソースは、コントロールプレーンテンプレートを定義し、コントロールプレーンノードの目的の設定を指定します。レプリカ数、Kubernetesバージョン、使用するCNIなどのパラメータが含まれます。また、一部のパラメータは、インスタンス化プロセス中に適切な値に置き換えられます。
- **Metal3ClusterTemplate:** このリソースは、インフラストラクチャテンプレートを定義し、基盤となるインフラストラクチャの目的の設定を指定します。コントロールプレーンエンドポイントやnoCloudProviderフラグなどのパラメータが含まれます。また、一部のパラメータは、インスタンス化プロセス中に適切な値に置き換えられます。

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlaneTemplate
metadata:
  name: example-controlplane-type2
  namespace: emea-spa
spec:
  template:
    spec:
      infrastructureRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
        kind: Metal3MachineTemplate
        name: example-controlplane    # This will be replaced by the patch applied in each
cluster instances
        namespace: emea-spa
        replicas: 1
        version: v1.33.3+rke2r1
        rolloutStrategy:
          type: "RollingUpdate"
          rollingUpdate:
            maxSurge: 1
        registrationMethod: "control-plane-endpoint"
        registrationAddress: "default" # This will be replaced by the patch applied in each
cluster instances
      serverConfig:
        cni: cilium
        cniMultusEnable: true
        tlsSan:
          - "default" # This will be replaced by the patch applied in each cluster
instances
      agentConfig:
        format: ignition
        additionalUserData:
```

```

        config: |
          default
        kubelet:
          extraArgs:
            - provider-id=metal3://BAREMETALHOST_UUID
        nodeName: "localhost.localdomain"
    ---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3ClusterTemplate
metadata:
  name: example-cluster-template-type2
  namespace: emea-spa
spec:
  template:
    spec:
      controlPlaneEndpoint:
        host: "default" # This will be replaced by the patch applied in each cluster
instances
      port: 6443
      noCloudProvider: true
    ---
apiVersion: cluster.x-k8s.io/v1beta1
kind: ClusterClass
metadata:
  name: example-clusterclass-type2
  namespace: emea-spa
spec:
  variables:
    - name: controlPlaneMachineTemplate
      required: true
      schema:
        openAPIV3Schema:
          type: string
    - name: controlPlaneEndpointHost
      required: true
      schema:
        openAPIV3Schema:
          type: string
    - name: tlsSan
      required: true
      schema:
        openAPIV3Schema:
          type: array
          items:
            type: string
  infrastructure:
    ref:
      kind: Metal3ClusterTemplate
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
      name: example-cluster-template-type2
  controlPlane:

```

```

ref:
  kind: RKE2ControlPlaneTemplate
  apiVersion: controlplane.cluster.x-k8s.io/v1beta1
  name: example-controlplane-type2
patches:
- name: setControlPlaneMachineTemplate
  definitions:
    - selector:
        apiVersion: controlplane.cluster.x-k8s.io/v1beta1
        kind: RKE2ControlPlaneTemplate
        matchResources:
          controlPlane: true
      jsonPatches:
        - op: replace
          path: "/spec/template/spec/infrastructureRef/name"
          valueFrom:
            variable: controlPlaneMachineTemplate
- name: setControlPlaneEndpoint
  definitions:
    - selector:
        apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
        kind: Metal3ClusterTemplate
        matchResources:
          infrastructureCluster: true # Added to select InfraCluster
      jsonPatches:
        - op: replace
          path: "/spec/template/spec/controlPlaneEndpoint/host"
          valueFrom:
            variable: controlPlaneEndpointHost
- name: setRegistrationAddress
  definitions:
    - selector:
        apiVersion: controlplane.cluster.x-k8s.io/v1beta1
        kind: RKE2ControlPlaneTemplate
        matchResources:
          controlPlane: true # Added to select ControlPlane
      jsonPatches:
        - op: replace
          path: "/spec/template/spec/registrationAddress"
          valueFrom:
            variable: controlPlaneEndpointHost
- name: setTlsSan
  definitions:
    - selector:
        apiVersion: controlplane.cluster.x-k8s.io/v1beta1
        kind: RKE2ControlPlaneTemplate
        matchResources:
          controlPlane: true # Added to select ControlPlane
      jsonPatches:
        - op: replace
          path: "/spec/template/spec/serverConfig/tlsSan"

```

```

        valueFrom:
          variable: tlsSan
- name: updateAdditionalUserData
  definitions:
    - selector:
        apiVersion: controlplane.cluster.x-k8s.io/v1beta1
        kind: RKE2ControlPlaneTemplate
        matchResources:
          controlPlane: true
    jsonPatches:
      - op: replace
        path: "/spec/template/spec/agentConfig/additionalUserData"
        valueFrom:
          template: |
            config: |
              variant: fcos
              version: 1.4.0
              storage:
                files:
                  - path: /var/lib/rancher/rke2/server/manifests/endpoint-copier-
operator.yaml
                    overwrite: true
                    contents:
                      inline: |
                        apiVersion: helm.cattle.io/v1
                        kind: HelmChart
                        metadata:
                          name: endpoint-copier-operator
                          namespace: kube-system
                        spec:
                          chart: oci://registry.suse.com/edge/charts/endpoint-
copier-operator
                          targetNamespace: endpoint-copier-operator
                          version: 304.0.1+up0.3.0
                          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/charts/metallb
                          targetNamespace: metallb-system
                          version: 304.0.0+up0.14.9
                          createNamespace: true
                  - path: /var/lib/rancher/rke2/server/manifests/metallb-cr.yaml
                    overwrite: true

```

```

      contents:
        inline: |
          apiVersion: metallb.io/v1beta1
          kind: IPAddressPool
          metadata:
            name: kubernetes-vip-ip-pool
            namespace: metallb-system
          spec:
            addresses:
              - {{ .controlPlaneEndpointHost }}/32
            serviceAllocation:
              priority: 100
              namespaces:
                - default
            serviceSelectors:
              - matchExpressions:
                - {key: "serviceType", operator: In, values:
[kubernetes-vip]}
            ---
            apiVersion: metallb.io/v1beta1
            kind: L2Advertisement
            metadata:
              name: ip-pool-l2-adv
              namespace: metallb-system
            spec:
              ipAddressPools:
                - kubernetes-vip-ip-pool
- path: /var/lib/rancher/rke2/server/manifests/endpoint-svc.yaml
  overwrite: true
  contents:
    inline: |
      apiVersion: v1
      kind: Service
      metadata:
        name: kubernetes-vip
        namespace: default
        labels:
          serviceType: kubernetes-vip
      spec:
        ports:
          - name: rke2-api
            port: 9345
            protocol: TCP
            targetPort: 9345
          - name: k8s-api
            port: 6443
            protocol: TCP
            targetPort: 6443
        type: LoadBalancer
systemd:
  units:

```

```

- name: rke2-preinstall.service
  enabled: true
  contents: |
    [Unit]
    Description=rke2-preinstall
    Wants=network-online.target
    Before=rke2-install.service
    ConditionPathExists=!/run/cluster-api/bootstrap-
success.complete

    [Service]
    Type=oneshot
    User=root
    ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
    ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/$(jq -
r .uuid /mnt/openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
    ExecStart=/bin/sh -c "echo \"node-name: $(jq -r .name /mnt/
openstack/latest/meta_data.json)\" >> /etc/rancher/rke2/config.yaml"
    ExecStartPost=/bin/sh -c "umount /mnt"
    [Install]
    WantedBy=multi-user.target

```

29.4.2 クラスティンスタンス定義

ClusterClassのコンテキストにおいて、クラスティンスタンスとは、定義されているClusterClassに基づいて作成されたクラスタの特定の実行中のインスタンスを指します。これは、ClusterClassで指定されたブループリントから直接派生した固有の設定、リソース、および運用状態を持つ具体的なデプロイメントを表します。これには、アクティブに実行されている特定のマシンのセット、ネットワーキング設定、および関連するKubernetesコンポーネントが含まれます。クラスティンスタンスを理解することは、ClusterClassフレームワークを使用してプロビジョニングされた特定のデプロイ済みクラスタのライフサイクルの管理、アップグレードの実行、スケーリング操作の実行、および監視の実行に必要不可欠です。

クラスティンスタンスを定義するには、次のリソースを定義する必要があります。

- Cluster
- Metal3MachineTemplate
- Metal3DataTemplate

テンプレート(ClusterClass定義ファイル)で以前に定義された変数は、クラスタのインスタンス化の最終値に置き換えられます。

```

apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: emea-spa-cluster-3

```

```

  namespace: emea-spa
spec:
  topology:
    class: example-clusterclass-type2 # Correct way to reference ClusterClass
    version: v1.33.3+rke2r1
    controlPlane:
      replicas: 1
    variables: # Variables to be replaced for this cluster
instance
  - name: controlPlaneMachineTemplate
    value: emea-spa-cluster-3-machinetemplate
  - name: controlPlaneEndpointHost
    value: 192.168.122.203
  - name: tlsSan
    value:
      - 192.168.122.203
      - https://192.168.122.203.sslip.io
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3MachineTemplate
metadata:
  name: emea-spa-cluster-3-machinetemplate
  namespace: emea-spa
spec:
  nodeReuse: True
  template:
    spec:
      automatedCleaningMode: metadata
      dataTemplate:
        name: emea-spa-cluster-3
      hostSelector:
        matchLabels:
          cluster-role: control-plane
          deploy-region: emea-spa
          cluster-type: type2
      image:
        checksum: http://fileserver.local:8080/eibimage-downstream-cluster.raw.sha256
        checksumType: sha256
        format: raw
        url: http://fileserver.local:8080/eibimage-downstream-cluster.raw
---
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3DataTemplate
metadata:
  name: emea-spa-cluster-3
  namespace: emea-spa
spec:
  clusterName: emea-spa-cluster-3
  metaData:
    objectNames:
      - key: name

```



```
object: machine
- key: local-hostname
  object: machine
```

このアプローチにより、ClusterClassを定義したら、3つのみのリソースでクラスタをデプロイする、より効率的なプロセスが可能になります。

IV ヒントとコツ

30 Edge Image Builder 235

31 Elemental 237

Edgeコンポーネントのヒントとコツ

30 Edge Image Builder

30.1 一般

- Linux以外の環境でこれらの手順に従ってイメージを構築する場合、仮想マシン経由でPodmanを実行している可能性があります。デフォルトでは、この仮想マシンにはシステムリソースが少量割り当てられるように設定され、RPM解決プロセスなどのリソース集約的な操作を実行する際に、Edge Image Builderが不安定な動作を引き起こす可能性があります。その場合は、Podmanマシンのリソースを調整する必要があります。調整するには、Podman Desktop (設定の歯車→Podmanマシン編集アイコン)を使用するか、直接podman-machine-set コマンド (<https://docs.podman.io/en/stable/markdown/podman-machine-set.1.html>) を実行してください。
- 現時点では、Edge Image Builderはクロスアーキテクチャセットアップでイメージを構築できません。すなわち、以下の環境で実行する必要があります。
 - SL Micro aarch64イメージを構築する場合は、AArch64システム(Apple Siliconなど)
 - SL Micro x86_64イメージを構築する場合は、AMD64/Intel 64システム

30.2 Kubernetes

- マルチノードのKubernetesクラスタを作成するには、定義ファイルのkubernetesセクションを以下を実行するように調整する必要があります。

- `kubernetes.nodes`の下にすべてのサーバおよびエージェントノードをリストする。
- `kubernetes.network.apiVIP`の下に、すべてのnon-initializerノードがクラスタに参加するために使用する仮想IPアドレスを設定する。
- オプションで、`kubernetes.network.apiHost`の下に、クラスタにアクセスするためのドメインアドレスを指定するようにAPIホストを設定する。この設定の詳細については、[Kubernetesセクションのドキュメント \(https://github.com/suse-edge/edge-image-builder/blob/main/docs/building-images.md#kubernetes\)](https://github.com/suse-edge/edge-image-builder/blob/main/docs/building-images.md#kubernetes) を参照してください。
- `Edge Image Builder`は、異なるノードのホスト名に基づいてそのKubernetesタイプ（サーバまたはエージェント）を決定します。この設定は定義ファイルで管理されていますが、マシンの一般的なネットワーキングセットアップには、[第12章「Edgeネットワーキング」](#)で説明されているDHCP設定を使用できます。

31 Elemental

31.1 一般

31.1.1 Rancherサービスの公開

RKE2またはK3sを使用する場合、サービス(このコンテキストではRancher)がデフォルトでは公開されていないため、管理クラスタから公開する必要があります。RKE2ではNGINX Ingressコントローラが使用されており、k3sではTraefikが使用されています。現在のワークフローでは、サービスの通知(L2またはBGPアドバタイズメント経由)にはMetalLBを使用し、それぞれのIngressコントローラを使用してHelmChartConfig経由でIngressを作成することが推奨されています。これは、新しいIngressオブジェクトを作成すると既存のセットアップが上書きされるためです。

1. Rancher Prime (Helm経由)をインストールし、必要な値を設定します。

```
hostname: rancher-192.168.64.101.sslip.io
replicas: 1
bootstrapPassword: Admin
global.cattle.psp.enabled: "false"
```



ヒント

詳細については、[Rancherのインストール \(https://ranchermanager.docs.rancher.com/v2.12/getting-started/installation-and-upgrade/install-upgrade-on-a-kubernetes-cluster\)](https://ranchermanager.docs.rancher.com/v2.12/getting-started/installation-and-upgrade/install-upgrade-on-a-kubernetes-cluster) のドキュメントに従ってください。

2. Rancherを公開するためのLoadBalancerサービスを作成します。

```
kubectl apply -f - <<EOF
apiVersion: helm.cattle.io/v1
kind: HelmChartConfig
metadata:
  name: rke2-ingress-nginx
  namespace: kube-system
spec:
  valuesContent: |-
    controller:
      config:
```

```
    use-forwarded-headers: "true"
    enable-real-ip: "true"
  publishService:
    enabled: true
  service:
    enabled: true
    type: LoadBalancer
    externalTrafficPolicy: Local
EOF
```

3. Helm値で以前に設定したIPアドレスを使用して、サービスのIPアドレスプールを作成します。

```
kubectl apply -f - <<EOF
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: ingress-ippool
  namespace: metallb-system
spec:
  addresses:
  - 192.168.64.101/32
  serviceAllocation:
    priority: 100
    serviceSelectors:
    - matchExpressions:
      - {key: app.kubernetes.io/name, operator: In, values: [rke2-ingress-nginx]}
EOF
```

4. IPアドレスプール用のL2アドバタイズメントを作成します。

```
kubectl apply -f - <<EOF
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: ingress-l2-adv
  namespace: metallb-system
spec:
  ipAddressPools:
  - ingress-ippool
EOF
```

5. Elementalが適切にインストールされていることを確認します。
 - a. 管理ノードにElemental OperatorとElemental UIをインストールします。
 - b. ダウンストリームノードにElementalの設定を登録コードとともに追加します。これによりEdge Image Builderにマシンのリモート登録オプションを含めるように要求されます。



ヒント

追加情報と例については、2.5項「Elementalのインストール」と2.6項「Elementalの設定」を確認してください。

31.2 ハードウェア固有

31.2.1 Trusted Platform Module

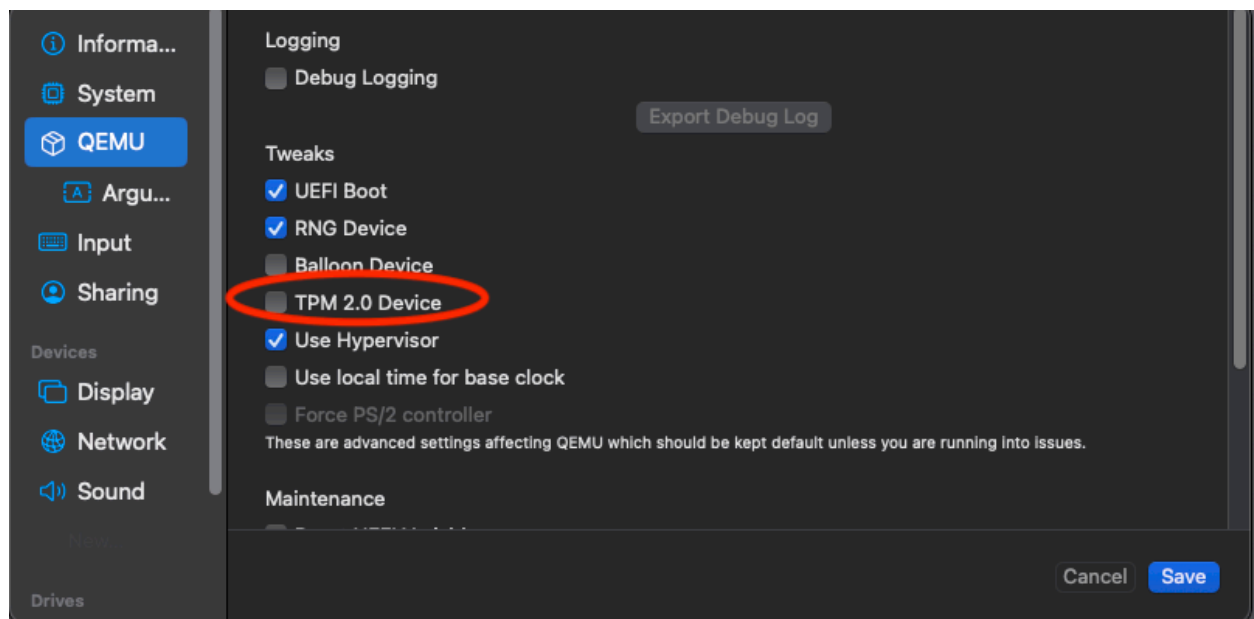
Trusted Platform Module (<https://elemental.docs.rancher.com/tpm/>) (TPM)設定を適切に行う必要があります。適切に設定しないと、次のようなエラーが発生します。

```
Nov 25 18:17:06 eled elemental-register[4038]: Error: registering machine: cannot generate authentication token: opening tpm for getting attestation data: TPM device not available
```

このエラーは、以下のアプローチのいずれかで軽減できます。

- 仮想マシン設定でTPMを有効にする

MacOSでのUTMの例



- `MachineRegistration`リソースでTPMシードに負の値を使用してTPMをエミュレートする

```
apiVersion: elemental.cattle.io/v1beta1
```

```
kind: MachineRegistration
metadata:
  name: ...
  namespace: ...
spec:
  ...
  elemental:
    ...
    registration:
      emulate-tpm: true
      emulated-tpm-seed: -1
```

- MachineRegistrationリソースでTPMを無効にする

```
apiVersion: elemental.cattle.io/v1beta1
kind: MachineRegistration
metadata:
  name: ...
  namespace: ...
spec:
  ...
  elemental:
    ...
    registration:
      emulate-tpm: false
```


V サードパーティの統合

32 NATS **242**

33 SUSE Linux Micro上のNVIDIA GPU **247**

サードパーティツールの統合方法

32 NATS

NATS (<https://nats.io/>)  は、ますますハイパーコネクテッド化が進む世界のために構築された接続テクノロジーです。NATSは、クラウドベンダ、オンプレミス、エッジ、Web、モバイルデバイスがどのように組み合わせられていてもアプリケーションが安全に通信することを可能にする単一のテクノロジーです。NATSはオープンソース製品ファミリーで構成されており、各製品は緊密に統合されている一方で、簡単に個別にデプロイできます。NATSは世界中で数千社もの企業で使用されており、マイクロサービス、エッジコンピューティング、モバイル、IoTなどのユースケースに幅広く対応しているため、NATSを使用して従来のメッセージングの強化や置き換えを図ることができます。

32.1 アーキテクチャ

NATSは、メッセージの形式でアプリケーション間のデータ交換を可能にするインフラストラクチャです。

32.1.1 NATSクライアントアプリケーション

NATSクライアントライブラリを使用すると、アプリケーションが異なるインスタンス間でパブリッシュ、サブスクライブ、要求、および応答できるようになります。このようなアプリケーションを一般的に クライアントアプリケーション と呼びます。

32.1.2 NATSサービスインフラストラクチャ

NATSサービスは、相互接続されてNATSサービスインフラストラクチャを提供するように設定された1つ以上のNATSサーバプロセスによって提供されます。NATSサービスインフラストラクチャは、1つのエンドデバイスで動作する単一のNATSサーバプロセスから、すべての主要クラウドプロバイダと世界のあらゆる地域にまたがる多数のクラスタからなるパブリックなグローバルスーパークラスタまで拡張可能です。

32.1.3 シンプルなメッセージングデザイン

NATSを使用すると、アプリケーションはメッセージを送受信して簡単に通信できます。これらのメッセージはサブジェクト文字列によってアドレス指定および識別され、ネットワークの場所には依存しません。データはエンコードされてメッセージとしてフレーム化され、パブリッシャによって送信されます。メッセージは1人以上のサブスクライバによって受信、デコード、処理されます。

32.1.4 NATS JetStream

NATSにはJetStreamと呼ばれる分散型の永続化システムが組み込まれています。JetStreamは、今日のテクノロジーにおけるストリーミングで明らかになった問題、すなわち複雑性、脆弱性、スケーラビリティの欠如を解決するために作成されました。また、JetStreamは、パブリッシャとサブスクライバのカップリングに関する問題(パブリッシュされたメッセージを受信するにはサブスクライバが稼働している必要がある)も解決します。NATS JetStreamの詳細については、[こちら \(https://docs.nats.io/nats-concepts/jetstream\)](https://docs.nats.io/nats-concepts/jetstream) を参照してください。

32.2 インストール

32.2.1 K3s上へのNATSのインストール

NATSは複数のアーキテクチャ向けに構築されているため、K3s (第14章 「K3s」) 上に簡単にインストールできます。

NATSのデフォルト値を上書きするvaluesファイルを作成しましょう。

```
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
```

では、Helmを介してNATSをインストールしてみましょう。

```
helm repo add nats https://nats-io.github.io/k8s/helm/charts/
helm install nats nats/nats --namespace nats --values values.yaml \
--create-namespace
```

上記の`values.yaml`ファイルでは、次のコンポーネントが`nats`ネームスペースに配置されます。

1. NATS StatefulsetのHAバージョン。3つのコンテナ(NATSサーバ + Configリローダと Metricsサイドカー)が含まれます。
2. NATS boxコンテナ。セットアップの確認に使用できる一連のNATSユーティリティが付属します。
3. JetStreamは、PodにバインドされたPVCが付属するKey-Valueバックエンドも利用します。

32.2.1.1 セットアップのテスト

```
kubectl exec -n nats -it deployment/nats-box -- /bin/sh -l
```

1. テストサブジェクトのサブスクリプションを作成します。

```
nats sub test &
```

2. テストサブジェクトにメッセージを送信します。

```
nats pub test hi
```

32.2.1.2 クリーンアップ

```
helm -n nats uninstall nats
rm values.yaml
```

32.2.2 K3sのバックエンドとしてのNATS

K3sが利用するコンポーネントの1つがKINE (<https://github.com/k3s-io/kine>) です。KINEは、最初からリレーショナルデータベースをターゲットとした代替ストレージバックエンドでetcdを置き換えることを可能にするシムです。JetStreamはKey Value APIを備えているので、NATSをK3sクラスタのバックエンドとして利用することが可能です。

K3sのビルトインNATSが容易になるマージ済みのPRがありますが、この変更はまだK3sリリースに含まれていません (<https://github.com/k3s-io/k3s/issues/7410#issue-1692989394>) .

このため、K3sのバイナリを手動で構築する必要があります。

32.2.2.1 K3sの構築

```
git clone --depth 1 https://github.com/k3s-io/k3s.git && cd k3s
```

次のコマンドは、ビルドタグにnatsを追加して、K3sでNATSビルトイン機能を有効にします。

```
sed -i '' 's/TAGS="ctrd"/TAGS="nats ctrd/g' scripts/build  
make local
```

<node-ip>は、K3sを起動するノードの実際のIPに置き換えます。

```
export NODE_IP=<node-ip>  
sudo scp dist/artifacts/k3s-arm64 ${NODE_IP}:/usr/local/bin/k3s
```



注記

K3sをローカルで構築するには、buildx Docker CLIプラグインが必要です。\$ `make local`が失敗する場合は、[手動でインストール \(https://github.com/docker/buildx#manual-download\)](https://github.com/docker/buildx#manual-download) できます。

32.2.2.2 NATS CLIのインストール

```
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}
```

32.2.2.3 K3sのバックエンドとしてのNATSの実行

ノードで`ssh`を実行し、`--datastore-endpoint`フラグで`nats`を指してK3sを実行しましょう。



注記

次のコマンドでは、K3sをフォアグラウンドプロセスとして起動するので、ログを簡単に追跡して問題がないかどうかを確認できます。現在の端末をブロックしないようにするには、コマンドの前に`&`フラグを追加して、バックグラウンドプロセスとして起動できます。

```
k3s server --datastore-endpoint=nats://
```



注記

NATSバックエンドを使用するK3sサーバを`slemicro` VM上で永続化するには、次のスクリプトを実行して、必要な設定で`systemd`サービスを作成します。

```
export INSTALL_K3S_SKIP_START=false
export INSTALL_K3S_SKIP_DOWNLOAD=true

curl -sfl https://get.k3s.io | INSTALL_K3S_EXEC="server \
--datastore-endpoint=nats://" sh -
```

32.2.2.4 トラブルシューティング

次のコマンドをノード上で実行して、ストリームのすべてが適切に動作していることを確認できます。

```
nats str report -a
nats str view -a
```

33 SUSE Linux Micro上のNVIDIA GPU

33.1 概要

このガイドでは、事前構築済みのオープンソースドライバ (<https://github.com/NVIDIA/open-gpu-kernel-modules>) を使用してホストレベルのNVIDIA GPUサポートをSUSE Linux Micro 6.1に実装する方法を説明します。これらのドライバは、NVIDIAのGPU Operator (<https://github.com/NVIDIA/gpu-operator>) によって動的にロードされるのではなく、オペレーティングシステムにベイクされているドライバです。この設定は、デプロイメントに必要なすべてのアーティファクトをあらかじめイメージにベイクしておき、ドライバのバージョンを動的に選択する必要がない(つまり、ユーザがKubernetesを介してドライバのバージョンを選択する必要がない)お客様に非常に適しています。このガイドでは最初に、すでに事前にデプロイされているシステムに追加コンポーネントをデプロイする方法を説明しますが、その後のセクションでは、Edge Image Builderを使用してこの設定を初期デプロイメントに組み込む方法について説明します。基本的な操作を読む必要がない場合や、手動でセットアップしたくない場合は、スキップしてそちらのセクションに進んでください。

これらのドライバのサポートは、SUSEとNVIDIAの両社が緊密に連携して提供しており、ドライバはパッケージリポジトリの一部としてSUSEによって構築および出荷されている点を強調しておくことが重要です。ただし、ドライバを使用する組み合わせについて不安や質問がある場合は、SUSEまたはNVIDIAのアカウントマネージャに問い合わせるサポートを受けてください。NVIDIA AI Enterprise (<https://www.nvidia.com/en-gb/data-center/products/ai-enterprise/>) (NVAIE)を使用する予定の場合は、NVAIE認定GPU (<https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/platform-support.html#supported-nvidia-gpus-and-systems>) を使用していることを確認してください。NVAIE認定GPUでは、独自のNVIDIAドライバを使用する必要がある**場合があります**。不明な点がある場合は、NVIDIAの担当者に問い合わせてください。

NVIDIA GPU Operatorの統合の詳細は、このガイドでは説明しません。Kubernetes用のNVIDIA GPU Operatorの統合についてはここでは説明ませんが、このガイドのほとんどの手順に従って、基礎となるオペレーティングシステムをセットアップできます。そして、NVIDIA GPU OperatorのHelmチャートの`driver.enabled=false`フラグを使用して**プリインストール**されたドライバをGPU Operatorが使用できるようにするだけで、ホスト上にインストールされたドライバが取得されます。より包括的な手順については、NVIDIA ([こちら](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>))で参照できます。

33.2 前提条件

このガイドに従って操作を進める場合、以下がすでに用意されていることを想定しています。

- SUSE Linux Micro 6.1がインストールされている少なくとも1台のホスト。物理でも仮想でも構いません。
- パッケージへのアクセスにはサブスクリプションが必要であるため、ホストがサブスクリプションに接続されていること。評価版は[こちら \(https://www.suse.com/download/sle-micro/\)](https://www.suse.com/download/sle-micro/) から入手できます。
- 互換性のあるNVIDIA GPU (<https://github.com/NVIDIA/open-gpu-kernel-modules#compatible-gpus>) がインストールされていること(またはSUSE Linux Microが実行されている仮想マシンに**完全に**パススルーされていること)。
- ルートユーザへのアクセス – 以下の説明では、自身がルートユーザであり、`sudo`を使用して特権を昇格して**いない**ことを想定しています。

33.3 手動インストール

このセクションでは、NVIDIAドライバをSUSE Linux Microオペレーティングシステムに直接インストールします。これはopen版NVIDIAドライバがSUSE Linux Microのコアパッケージリポジトリの一部となったためであり、必須のRPMパッケージをインストールするのと同じように簡単にインストールできるようになりました。実行可能パッケージのコンパイルやダウンロードは必要ありません。以下では、最新のGPUをサポートする「G06」世代ドライバのデプロイについて手順を追って説明します(詳細については[こちら \(https://en.opensuse.org/SDB:NVIDIA_drivers#Install\)](https://en.opensuse.org/SDB:NVIDIA_drivers#Install) を参照してください)。ご使用のシステムに搭載されているNVIDIA GPUに適切なドライバ世代を選択してください。最新のGPUでは、「G06」ドライバが最も一般的な選択肢です。

始める前に、SUSEがSUSE Linux Microの一部として出荷するopen版NVIDIAドライバのほか、ご自身のセットアップに追加のNVIDIAコンポーネントも必要な場合があることを認識しておくことが重要です。たとえば、OpenGLライブラリ、CUDAツールキット、`nvidia-smi`などのコマンドラインユーティリティ、`nvidia-container-toolkit`などのコンテナ統合コンポーネントです。これらのコンポーネントの多くはNVIDIA独自のソフトウェアであるため、SUSEからは出荷されません。また、NVIDIAの代わりにSUSEが出荷しても意味がありません。そのため、説明の一環として、これらのコンポーネントにアクセスできるようにする追加のリポジトリを設定し、これらのツールの使用方法の例をいくつか説明し、完全に機能するシステムを作成します。SUSEのリポジトリとNVIDIAのリポジトリを区別することが重要です。これは、NVIDIAが提供するパッケージのバージョンとSUSEが構築したものが一致しな

い場合があるためです。これは通常、SUSEがopen版ドライバの新バージョンを利用可能にしたときに発生し、NVIDIAのリポジトリで同等のパッケージが利用可能になるまでに数日かかります。

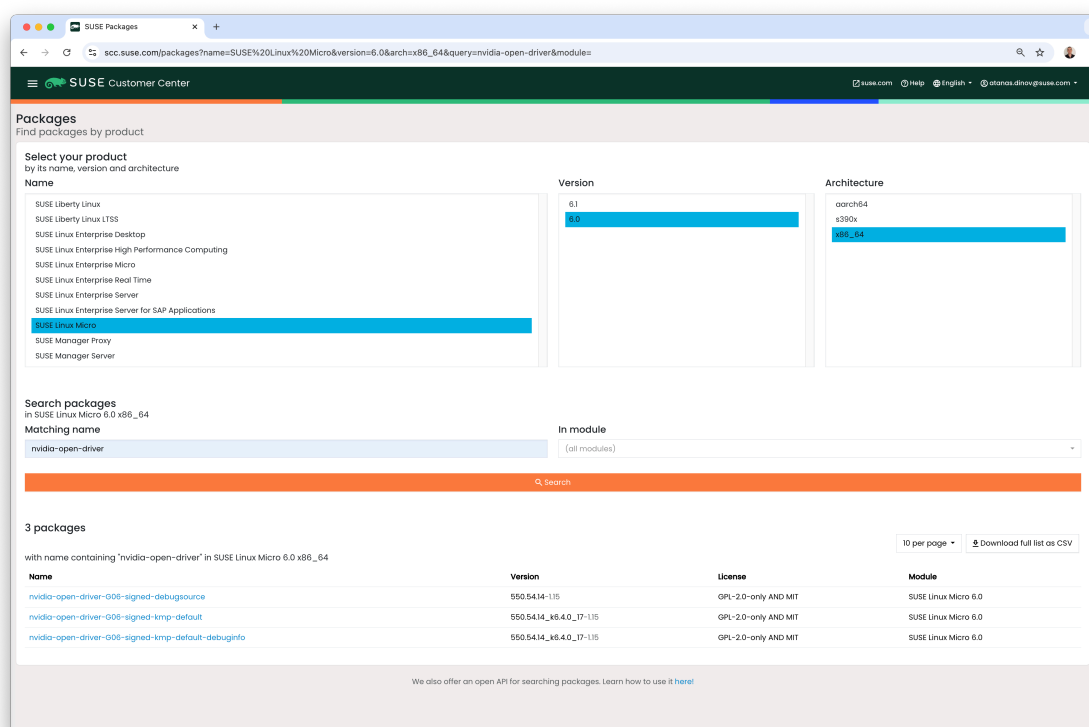
以下をチェックして、選択するドライババージョンがGPUと互換性があり、CUDAの要件を満たしていることを確認することをお勧めします。

- CUDAリリースノート (<https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/>) 
- デプロイを計画しているドライババージョンが、NVIDIAリポジトリ (https://download.nvidia.com/suse/sle15sp6/x86_64/)  に一致するバージョンがあり、サポートコンポーネントの同等のパッケージバージョンが利用可能であることを確認します。



ヒント

NVIDIAオープンドライババージョンを確認するには、ターゲットマシンで `zypper se -s nvidia-open-driver` を実行するか、**または**SUSE Customer Centerで [SUSE Linux Micro 6.1 for AMD64/Intel 64](https://scc.suse.com/packages?name=SUSE%20Linux%20Micro&version=6.1&arch=x86_64) (https://scc.suse.com/packages?name=SUSE%20Linux%20Micro&version=6.1&arch=x86_64)  の「nvidia-open-driver」を検索します。



The screenshot shows the SUSE Customer Center web interface. The search results section displays three packages matching the criteria:

Name	Version	License	Module
nvidia-open-driver-006-signed-debugsource	550.54.14-1.15	GPL-2.0-only AND MIT	SUSE Linux Micro 6.0
nvidia-open-driver-006-signed-kmp-default	550.54.14_k8.4.0_17-1.15	GPL-2.0-only AND MIT	SUSE Linux Micro 6.0
nvidia-open-driver-006-signed-kmp-default-debuginfo	550.54.14_k8.4.0_17-1.15	GPL-2.0-only AND MIT	SUSE Linux Micro 6.0

NVIDIAリポジトリで同等のバージョンが利用可能であることを確認したら、ホストオペレーティングシステムにパッケージをインストールできます。そのためには、`transactional-update`セッションを開く必要があります。これにより、基礎となるオペレーティングシステムの読み込み/書き込みスナップショットが新しく作成され、イミュータブルプラットフォームに変更を加えることが可能になります(`transactional-update`の詳細については、[こちら \(https://documentation.suse.com/sle-micro/6.1/html/Micro-transactional-updates/transactional-updates.html\)](https://documentation.suse.com/sle-micro/6.1/html/Micro-transactional-updates/transactional-updates.html) を参照してください)。

```
transactional-update shell
```

`transactional-update`シェルを起動したら、NVIDIAからパッケージリポジトリを追加します。これにより、`nvidia-smi`などの追加ユーティリティをプルできます。

```
zypper ar https://download.nvidia.com/suse/sle15sp6/ nvidia-suse-main
zypper --gpg-auto-import-keys refresh
```

その後、ドライバと、追加ユーティリティの`nvidia-compute-utils`をインストールできます。ユーティリティが不要の場合は省略できますが、テスト目的の場合は、この段階でインストールする価値があります。

```
zypper install -y --auto-agree-with-licenses nvidia-open-driver-G06-signed-kmp nvidia-compute-utils-G06
```



注記

インストールが失敗する場合、選択したドライババージョンとNVIDIAがリポジトリで配布しているバージョンとの間で依存関係の不一致があることを示している可能性があります。前のセクションを参照して、バージョンが一致していることを確認してください。また、別のドライババージョンをインストールしてみてください。たとえば、NVIDIAリポジトリに以前のバージョンがある場合、インストールコマンドに`nvidia-open-driver-G06-signed-kmp=550.54.14`を指定して、一致するバージョンを指定してみることができます。

次に、サポートされているGPUを使用していない場合は([こちら \(https://github.com/NVIDIA/open-gpu-kernel-modules#compatible-gpus\)](https://github.com/NVIDIA/open-gpu-kernel-modules#compatible-gpus) でリストを確認できます)、モジュールレベルでサポートを有効にすることで、ドライバが動作するかどうかを確認できますが、結果はユーザによって異なります。**サポートされているGPUを使用している場合は、この手順はスキップしてください。**

```
sed -i '/NVreg_OpenRmEnableUnsupportedGpus/s/^#//g' /etc/modprobe.d/50-nvidia-default.conf
```

これらのパッケージをインストールしたので、transactional-updateセッションを終了します。

```
exit
```



注記

次に進む前に、transactional-updateセッションを終了していることを確認してください。

ドライバをインストールしたら、再起動します。SUSE Linux Microはイミュータブルオペレーティングシステムであるため、前の手順で作成した新しいスナップショットで再起動する必要があります。ドライバはこの新しいスナップショットにのみインストールされるため、この新しいスナップショットで再起動しないとドライバをロードできません(新しいスナップショットでの再起動は自動的に実行されます)。準備ができればrebootコマンドを発行します。

```
reboot
```

システムが正常に再起動したら、ログインし直し、nvidia-smiツールを使用して、ドライバが正常にロードされていて、GPUへのアクセスと列挙をどちらも実行できることを確認します。

```
nvidia-smi
```

このコマンドの出力は次のような出力になります。以下の例では、GPUが2つあることに注意してください。

```
+-----+
| 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							

これで、SUSE Linux MicroシステムへのNVIDIAドライバのインストールと検証プロセスは完了です。

33.4 手動インストールの追加検証

この段階で確認できるのは、ホストレベルでNVIDIAデバイスにアクセスできること、およびドライバが正常にロードされていることだけです。ただし、それが機能していることを確認したい場合は、簡単なテストを実施して、GPUがユーザスペースアプリケーションから、理想的にはコンテナ経由で命令を受け取れること、および実際のワークロードが通常使用するものであるCUDAライブラリを通じて命令を受け取れることを検証します。このためには、`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)) をインストールしてホストOSにさらに変更を加えることができます。まず、別の `transactional-update` シェルを開きます。前の手順ではこれを1つのトランザクションで実行できたことに注目し、後のセクションでこれを完全に自動的に実行する方法を確認します。

```
transactional-update shell
```

次に、NVIDIA Container Toolkitリポジトリから `nvidia-container-toolkit` パッケージをインストールします。

- 次の `nvidia-container-toolkit.repo` には、安定版 (`nvidia-container-toolkit`) と実験版 (`nvidia-container-toolkit-experimental`) のリポジトリが含まれています。運用環境での使用には、安定版リポジトリをお勧めします。実験版リポジトリはデフォルトで無効になっています。

```
zypper ar https://nvidia.github.io/libnvidia-container/stable/rpm/nvidia-container-toolkit.repo
zypper --gpg-auto-import-keys install -y nvidia-container-toolkit
```

準備ができれば、`transactional-update` シェルを終了できます。

```
exit
```

…そして新しいスナップショットでマシンを再起動します。

```
reboot
```



注記

前述同様に、変更を有効にするには、必ず `transactional-shell` を終了し、マシンを再起動する必要があります。

マシンが再起動したら、システムがNVIDIA Container Toolkitを使用してデバイスを正常に列挙できることを確認できます。出力は詳細で、INFOとWARNのメッセージがありますが、ERRORのメッセージはありません。

```
nvidia-ctk cdi generate --output=/etc/cdi/nvidia.yaml
```

これにより、そのマシンで起動するコンテナはすべて、検出されたNVIDIA GPUデバイスを使用できることが確認されます。準備ができれば、podmanベースのコンテナを実行できます。これをpodmanを介して行うことで、コンテナ内からNVIDIAデバイスへのアクセスを効果的に検証することができ、後の段階でKubernetesで同じ操作をするための自信が得られます。podmanに対し、前のコマンドで [SLE BCI \(https://registry.suse.com/repositories/bci-bci-base-15sp6\)](https://registry.suse.com/repositories/bci-bci-base-15sp6) に基づいて処理したラベル付きのNVIDIAデバイスへのアクセス権を与え、バッシュコマンドを実行します。

```
podman run --rm --device nvidia.com/gpu=all --security-opt=label=disable -it  
registry.suse.com/bci/bci-base:latest bash
```

続いて、一時的なpodmanコンテナ内からコマンドを実行します。このコンテナは基盤となるシステムにはアクセスできず一時的であるため、ここで行う操作は永続せず、基盤となるホスト上にあるものを壊すことは一切できないはずです。現在はコンテナ内で作業しているため、必要なCUDAライブラリをインストールできます。ここでもう一度、ご使用のドライバにあったCUDAバージョンを [こちら \(https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/\)](https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/) で確認してください。ただし、必要なCUDAバージョンは `nvidia-smi` の前の出力に表示されているはずです。以下の例では、**CUDA 12.3**をインストールして多数の例、デモ、開発キットをプルし、GPUを完全に検証できるようにしています。

```
zypper ar https://developer.download.nvidia.com/compute/cuda/repos/sles15/x86_64/ cuda-  
suse  
zypper in -y cuda-libraries-devel-12-3 cuda-minimal-build-12-3 cuda-demo-suite-12-3
```

これが正常にインストールされた後にコンテナを終了しないでください。 `deviceQuery` CUDAの例を実行し、CUDAを介して、およびコンテナ自体からGPUアクセスを包括的に検証します。

```
/usr/local/cuda-12/extras/demo_suite/deviceQuery
```

成功すると、次のような出力が表示されます。コマンドの最後にある「Result = PASS」というメッセージに注意してください。また、次の出力では2つのGPUが正しく識別されていますが、ご使用の環境では1つしかない場合があることにも注意してください。

```
/usr/local/cuda-12/extras/demo_suite/deviceQuery Starting...
```

```
CUDA Device Query (Runtime API) version (CUDART static linking)
```

```
Detected 2 CUDA Capable device(s)
```

```
Device 0: "NVIDIA A100-PCIE-40GB"
```

```
CUDA Driver Version / Runtime Version      12.2 / 12.1
CUDA Capability Major/Minor version number: 8.0
Total amount of global memory:              40339 MBytes (42298834944 bytes)
(108) Multiprocessors, ( 64) CUDA Cores/MP: 6912 CUDA Cores
GPU Max Clock rate:                        1410 MHz (1.41 GHz)
Memory Clock rate:                         1215 Mhz
Memory Bus Width:                          5120-bit
L2 Cache Size:                             41943040 bytes
Maximum Texture Dimension Size (x,y,z)      1D=(131072), 2D=(131072, 65536),
3D=(16384, 16384, 16384)
Maximum Layered 1D Texture Size, (num) layers 1D=(32768), 2048 layers
Maximum Layered 2D Texture Size, (num) layers 2D=(32768, 32768), 2048 layers
Total amount of constant memory:            65536 bytes
Total amount of shared memory per block:     49152 bytes
Total number of registers available per block: 65536
Warp size:                                  32
Maximum number of threads per multiprocessor: 2048
Maximum number of threads per block:         1024
Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
Maximum memory pitch:                       2147483647 bytes
Texture alignment:                          512 bytes
Concurrent copy and kernel execution:        Yes with 3 copy engine(s)
Run time limit on kernels:                   No
Integrated GPU sharing Host Memory:          No
Support host page-locked memory mapping:     Yes
Alignment requirement for Surfaces:          Yes
Device has ECC support:                      Enabled
Device supports Unified Addressing (UVA):     Yes
Device supports Compute Preemption:          Yes
Supports Cooperative Kernel Launch:          Yes
Supports MultiDevice Co-op Kernel Launch:    Yes
Device PCI Domain ID / Bus ID / location ID: 0 / 23 / 0
Compute Mode:
  < Default (multiple host threads can use ::cudaSetDevice() with device
simultaneously) >
```

```
Device 1: <snip to reduce output for multiple devices>
    < Default (multiple host threads can use ::cudaSetDevice() with device
simultaneously) >
> Peer access from NVIDIA A100-PCIE-40GB (GPU0) -> NVIDIA A100-PCIE-40GB (GPU1) : Yes
> Peer access from NVIDIA A100-PCIE-40GB (GPU1) -> NVIDIA A100-PCIE-40GB (GPU0) : Yes

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 12.3, CUDA Runtime Version =
12.3, NumDevs = 2, Device0 = NVIDIA A100-PCIE-40GB, Device1 = NVIDIA A100-PCIE-40GB
Result = PASS
```

ここから、続いて他のCUDAワークロードを実行できます。コンパイラやCUDAエコシステムの他の側面を使用して、さらにテストを実行できます。完了したら、コンテナを終了できます。コンテナにインストールしたものはすべて一時的なものであるため(したがって失われるため)、基盤となるオペレーティングシステムには影響がないことに注意してください。

```
exit
```

33.5 Kubernetesを使用した実装

open版NVIDIAドライバをSUSE Linux Microにインストールして使用できることが証明されたので、同じマシンにKubernetesを設定してみましょう。このガイドでは、Kubernetesのデプロイについては説明しませんが、[K3s \(https://k3s.io/\)](https://k3s.io/) または [RKE2 \(https://docs.rke2.io/install/quickstart\)](https://docs.rke2.io/install/quickstart) をインストール済みで、kubeconfigが適宜設定されており、標準の `kubectl` コマンドをスーパーユーザとして実行できることを前提としています。ここではノードがシングルノードクラスタを形成していることを想定していますが、中心となる手順はマルチノードクラスタでも同様です。まず、`kubectl` のアクセスが機能していることを確認します。

```
kubectl get nodes
```

次のような画面が表示されます。

NAME	STATUS	ROLES	AGE	VERSION
node0001	Ready	control-plane,etcd,master	13d	v1.33.3+rke2r1

k3s/rke2のインストールによってホスト上のNVIDIA Container Toolkitが検出され、NVIDIAランタイム統合が `containerd` (k3s/rke2が使用するContainer Runtime Interface) に自動設定されていることを確認する必要があります。確認するには、`containerd` の `config.toml` ファイルをチェックします。

```
tail -n8 /var/lib/rancher/rke2/agent/etc/containerd/config.toml
```

次のような画面が表示される必要があります。K3sの場合に相当する場所は `/var/lib/rancher/k3s/agent/etc/containerd/config.toml` です。


```
[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"
```



注記

これらのエントリが存在しない場合は、検出が失敗している可能性があります。この原因として考えられるのは、マシンまたはKubernetesサービスを再起動していないことです。必要に応じて、上記のようにこれらを手動で追加してください。

次に、`NVIDIA RuntimeClass`を追加のKubernetesランタイムとしてデフォルト値に設定する必要があります。これにより、GPUへのアクセスが必要なPodに対するユーザ要求が、`containerd`の設定に従って、`NVIDIA Container Toolkit`を使用して`nvidia-container-runtime`を介してアクセスできるようにします。

```
kubectl apply -f - <<EOF
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: nvidia
handler: nvidia
EOF
```

次の手順は、[NVIDIA Device Plugin \(https://github.com/NVIDIA/k8s-device-plugin\)](https://github.com/NVIDIA/k8s-device-plugin) を設定することです。これにより、`NVIDIA Container Toolkit`と連携して、クラスタ内で使用可能なリソースとして`NVIDIA GPU`を利用するようにKubernetesを設定します。このツールはまず、基盤となるホスト上のすべての機能(GPU、ドライバ、その他の機能(GLなど))を検出し、その後、ユーザがGPUリソースを要求してアプリケーションの一部として使用できるようにします。

まず、`NVIDIA Device Plugin`用のHelmリポジトリを追加して更新する必要があります。

```
helm repo add nvdp https://nvidia.github.io/k8s-device-plugin
helm repo update
```

これで、`NVIDIA Device Plugin`をインストールできます。

```
helm upgrade -i nvdp nvdp/nvidia-device-plugin --namespace nvidia-device-plugin --create-namespace --version 0.14.5 --set runtimeClassName=nvidia
```

数分後、新しいPodが実行されているのがわかります。これで、利用可能なノード上での検出は完了し、検出されたGPUの数を示すタグがノードに付けられます。


```
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"
}
```

これで、このGPUを使用するNVIDIA Podを作成する準備ができました。CUDA Benchmarkコンテナで試してみましょう。

```
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
```

すべて問題なければ、ログを見て、ベンチマーク情報を確認できます。

```
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
```

最後に、アプリケーションでOpenGLが必要な場合は、必要なNVIDIA OpenGLライブラリをホストレベルでインストールし、NVIDIA Device PluginとNVIDIA Container Toolkitを使用してそのライブラリをコンテナで利用できるようにすることができます。これを行うには、次のようにパッケージをインストールします。

```
transactional-update pkg install nvidia-gl-G06
```



注記

このパッケージをアプリケーションで使用できるようにするには再起動が必要です。NVIDIA Device Pluginは、NVIDIA Container Toolkitを介してこれを自動的に再検出します。

33.6 Edge Image Builderを使用した統合

さて、SUSE Linux Micro上のアプリケーションとGPUの全機能をデモで示したので、[第11章「Edge Image Builder」](#)を使用してすべてをまとめ、デプロイ可能/使用可能なISOまたはRAWディスクイメージで提供したいと思います。このガイドでは、Edge Image Builderの使用方法は説明せずに、このようなイメージを構築するために必要な設定について説明します。以下に、必要なすべてのコンポーネントを追加設定なしにデプロイするためのイメージ定義の例と、必要なKubernetes設定ファイルを示します。以下に示す例では、Edge Image Builderディレクトリは次のようなディレクトリ構造になっています。

```

.
├─ base-images
│   └─ SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso
├─ eib-config-iso.yaml
├─ kubernetes
│   ├── config
│   │   └─ server.yaml
│   ├── helm
│   │   └─ values
│   │       └─ nvidia-device-plugin.yaml
│   └─ manifests
│       └─ nvidia-runtime-class.yaml
└─ rpms
    ├── gpg-keys
    └─ nvidia-container-toolkit.key

```

これらのファイルを調べてみましょう。まず、K3sを実行するシングルノードクラスタのサンプルイメージ定義を次に示します。このイメージ定義では、ユーティリティとOpenGLパッケージもデプロイします([eib-config-iso.yaml](#))。

```

apiVersion: 1.3
image:
  arch: x86_64
  imageType: iso
  baseImage: SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso
  outputImageName: deployimage.iso
operatingSystem:
  time:
    timezone: Europe/London
  ntp:
    pools:
      - 2.suse.pool.ntp.org
isoConfiguration:
  installDevice: /dev/sda
users:
  - username: root
    encryptedPassword: $6$XcQN1xkuQKjWEtQG$WbhV80rbveDLJDz1c93K5Ga9JDjt3mF.ZUnhYtsS7uE52FR8mmT8Cnii/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/sle15sp6/
    - url: https://nvidia.github.io/libnvidia-container/stable/rpm/x86_64
  sccRegistrationCode: [snip]
kubernetes:
  version: v1.33.3+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
```



注記

これは単なる例です。要件や期待に合うようにカスタマイズする必要がある場合があります。また、SUSE Linux Microを使用する場合は、パッケージの依存関係を解決して NVIDIAドライバをプルするために、独自の `sccRegistrationCode` を指定する必要があります。

これに加えて、他のコンポーネントを追加して、ブート時にKubernetesによってロードされるようにする必要があります。EIBディレクトリにはまず `kubernetes` ディレクトリが必要で、その下に設定、Helmチャート値、必要な追加のマニフェスト用のサブディレクトリが必要です。

```
mkdir -p kubernetes/config kubernetes/helm/values kubernetes/manifests
```

CNIを選択し(選択しない場合はデフォルトでCiliumになります)、SELinuxを有効にして、(オプションの)Kubernetes設定を行きましょう。

```
cat << EOF > kubernetes/config/server.yaml
cni: cilium
selinux: true
EOF
```

続いて、NVIDIA RuntimeClassがKubernetesクラスタ上に作成されていることを確認します。

```
cat << EOF > kubernetes/manifests/nvidia-runtime-class.yaml
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: nvidia
handler: nvidia
EOF
```

ビルトインHelmコントローラを使用して、Kubernetes自体を使用してNVIDIA Device Pluginをデプロイします。チャートの値ファイルでランタイムクラスを指定しましょう。

```
cat << EOF > kubernetes/helm/values/nvidia-device-plugin.yaml
runtimeClassName: nvidia
EOF
```

次に進む前に、NVIDIA Container Toolkit RPMの公開鍵を取得する必要があります。

```
mkdir -p rpms/gpg-keys
curl -o rpms/gpg-keys/nvidia-container-toolkit.key https://nvidia.github.io/libnvidia-
container/gpgkey
```

Kubernetesバイナリ、コンテナイメージ、Helmチャート(および参照イメージ)など、必要なアーティファクトがすべて自動的にエアギャップ化されます。つまり、デプロイ時のシステムにはデフォルトでインターネット接続は不要です。ここで必要なのは[SUSEダウンロードページ \(https://www.suse.com/download/sle-micro/\)](https://www.suse.com/download/sle-micro/) からSUSE Linux Micro ISOを取得する(そしてそれをbase-imagesディレクトリに配置する)ことだけです。そうすれば、Edge Image Builderツールを呼び出してISOを生成できます。この例を完了するために、イメージの構築に使用したコマンドを次に示します。

```
podman run --rm --privileged -it -v /path/to/eib-files:/eib \
registry.suse.com/edge/3.4/edge-image-builder:1.3.0 \
build --definition-file eib-config-iso.yaml
```

Edge Image Builderの詳細については、[ドキュメント \(https://github.com/suse-edge/edge-image-builder/blob/release-1.3/docs/building-images.md\)](https://github.com/suse-edge/edge-image-builder/blob/release-1.3/docs/building-images.md) を参照してください。

33.7 問題の解決

33.7.1 nvidia-smiでGPUが検出されない

dmesgを使用してカーネルメッセージを確認します。NvKMSKapDeviceを割り当てることができないことを示している場合は、サポート対象外のGPUの回避策を適用します。

```
sed -i '/NVreg_OpenRmEnableUnsupportedGpus/s/^#//g' /etc/modprobe.d/50-nvidia-default.conf
```

メモ: 上記の手順でカーネルモジュールの設定を変更した場合は、変更を有効にするために、カーネルモジュールを再ロードするか、再起動する必要があります。

VI Day 2操作

34 Edge 3.4のマイグレーション 263

35 管理クラスタ 268

36 ダウンストリームクラスタ 322

このセクションでは、管理者がさまざまな「Day 2」操作タスクを管理クラスタとダウンストリームクラスタの両方で処理する方法について説明します。

34 Edge 3.4のマイグレーション

このセクションでは、管理クラスタとダウストリームクラスタをEdge 3.3からEdge 3.4.0に移行する方法について説明します。

重要

クラスタマイグレーションは常に、Edge 3.3の 最新のZ-streamリリースから実行してください。

常に、Edge 3.4.0リリースに移行してください。マイグレーション後のアップグレードについては、管理クラスタ(第35章「[管理クラスタ](#)」)およびダウストリームクラスタ(第36章「[ダウストリームクラスタ](#)」)のセクションを参照してください。

34.1 管理クラスタ

このセクションでは、次のトピックについて説明します。

[34.1.1項「前提条件」](#) - マイグレーションを開始する前に完了する必要がある前提条件の手順。

[34.1.2項「Upgrade Controller」](#) - 第22章「[Upgrade Controller](#)」を使用して管理クラスタのマイグレーションを実行する方法。

[34.1.3項「Fleet」](#) - 第8章「[Fleet](#)」を使用して管理クラスタのマイグレーションを実行する方法。

34.1.1 前提条件

34.1.1.1 Bare Metal Operator CRDのアップグレード

注記

[第10章「Metal³」](#) チャートのアップグレードが必要なクラスタにのみ適用されます。

Metal3 Helmチャートには、[Bare Metal Operator \(BMO\)](https://book.metal3.io/bmo/introduction.html) (<https://book.metal3.io/bmo/introduction.html>)  CRDが含まれており、これはHelmのCRD (https://helm.sh/docs/chart_best_practices/custom_resource_definitions/#method-1-let-helm-do-it-for-you)  ディレクトリを活用することで提供されています。

ただし、このアプローチには特定の制限があり、特にHelmを使用してこのディレクトリ内のCRDをアップグレードできません。詳細については、[Helmのドキュメント](https://helm.sh/docs/chart_best_practices/custom_resource_definitions/#some-caveats-and-explanations) (https://helm.sh/docs/chart_best_practices/custom_resource_definitions/#some-caveats-and-explanations)  を参照してください。

そのため、Metal³をEdge 3.4.0互換のバージョンにアップグレードする前に、ユーザは基盤となるBMO CRDを手動でアップグレードする必要があります。

Helmがインストールされ、[kubectl](#)が管理 クラスタを指すように設定されているマシンで、次の手順を実行します。

1. BMO CRDを手動で適用します。

```
helm show crds oci://registry.suse.com/edge/charts/metal3 --version  
304.0.16+up0.12.6 | kubectl apply -f -
```

34.1.2 Upgrade Controller

重要

[Upgrade Controller](#)は現在、**非エアギャップ管理**クラスタに対してのみEdgeリリースのマイグレーションをサポートしています。

次のトピックは、このセクションの一部として説明します。

[34.1.2.1項「前提条件」](#) - [Upgrade Controller](#)に固有の前提条件。

[34.1.2.2項「マイグレーション手順」](#) - [Upgrade Controller](#)を使用して、[管理](#)クラスタを新しいEdgeバージョンに移行するための手順。

34.1.2.1 前提条件

34.1.2.1.1 Edge 3.4 Upgrade Controller

Upgrade Controllerを使用する前に、まず目的のEdgeリリースに移行できるバージョンが実行されていることを確認する必要があります。

このためには:

1. 以前のEdgeリリースからUpgrade Controllerがすでにデプロイされている場合は、そのチャートをアップグレードします。

```
helm upgrade upgrade-controller -n upgrade-controller-system oci://  
registry.suse.com/edge/charts/upgrade-controller --version 304.0.1+up0.1.1
```

2. Upgrade Controllerがデプロイされていない場合は、22.3項「Upgrade Controllerのインストール」に従います。

34.1.2.2 マイグレーション手順

Upgrade Controllerを使用した管理クラスタのマイグレーションの実行は、アップグレードの実行と基本的に類似しています。

唯一の違いは、UpgradePlan が3.4.0リリースバージョンを指定する**必要がある**ことです。

```
apiVersion: lifecycle.suse.com/v1alpha1  
kind: UpgradePlan  
metadata:  
  name: upgrade-plan-mgmt  
  # Change to the namespace of your Upgrade Controller  
  namespace: CHANGE_ME  
spec:  
  releaseVersion: 3.4.0
```

上記のUpgradePlanを使用してマイグレーションを実行する方法については、Upgrade Controllerのアップグレードプロセス(35.1項「Upgrade Controller」)を参照してください。

34.1.3 Fleet



注記

可能な限り、マイグレーションには34.1.2項「Upgrade Controller」を使用してください。

Upgrade Controllerでカバーされていないユースケースの場合にのみ、このセクションを参照してください。

Fleetを使用した管理クラスタのマイグレーションの実行は、基本的にアップグレードの実行と類似しています。

主な違いは次のとおりです。

1. フリートは suse-edge/fleet-examples リポジトリの release-3.4.0 (<https://github.com/suse-edge/fleet-examples/releases/tag/release-3.4.0>)  リリースから **使用される必要があります**。
2. アップグレードがスケジュールされているチャートは、Edge 3.4.0 リリースと互換性のあるバージョンにアップグレードする **必要があります**。Edge 3.4.0 コンポーネントのリストについては、[52.3項 「リリース3.4.0」](#) を参照してください。

！ 重要

Edge 3.4.0 へのマイグレーションを成功させるには、ユーザが上記の点に従うことが重要です。

上記の点を踏まえて、ユーザはマイグレーションの実行に必要な手順に関する包括的なガイドとして、管理 クラスタ Fleet ([35.2項 「Fleet」](#)) のドキュメントに従うことができます。

34.2 ダウンストリームクラスタ

[34.2.1項 「Fleet」](#) - 第8章 「Fleet」 を使用して ダウンストリーム クラスタのマイグレーションを実行する方法。

34.2.1 Fleet

Fleetを使用した ダウンストリーム クラスタのマイグレーションの実行は、アップグレードの実行と基本的に類似しています。

主な違いは次のとおりです。

1. フリートは [suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples/releases/tag/release-3.4.0) リポジトリの [release-3.4.0 \(https://github.com/suse-edge/fleet-examples/releases/tag/release-3.4.0\)](https://github.com/suse-edge/fleet-examples/releases/tag/release-3.4.0) リリースから**使用される必要があります**。
2. アップグレードがスケジュールされているチャートは、[Edge 3.4.0](#) リリースと互換性のあるバージョンにアップグレードする **必要があります**。[Edge 3.4.0](#) コンポーネントのリストについては、[52.3項 「リリース3.4.0」](#) を参照してください。

！ 重要

[Edge 3.4.0](#) へのマイグレーションを成功させるには、ユーザが上記の点に従うことが重要です。

上記の点を踏まえて、ユーザはマイグレーションの実行に必要な手順に関する包括的なガイドとして、[ダウンストリームクラスタFleet \(36.1項 「Fleet」\)](#) のドキュメントに従うことができます。

35 管理クラスタ

現在、管理クラスタで「Day 2」操作を実行するには、次の2つの方法があります。

1. 第22章「Upgrade Controller」 - 35.1項「Upgrade Controller」を通じて
2. 第8章「Fleet」 - 35.2項「Fleet」を通じて

35.1 Upgrade Controller



重要

Upgrade Controllerは現在、**非エアギャップ管理**クラスタのDay 2操作のみをサポートしています。

このセクションでは、あるEdgeプラットフォームバージョンから別のバージョンに管理クラスタをアップグレードする方法に関連するさまざまなDay 2操作を実行する方法について説明します。

Day 2操作は、Upgrade Controller (第22章「Upgrade Controller」)によって自動化され、以下のものが含まれます。

- SUSE Linux Micro (第9章「SUSE Linux Micro」) OSのアップグレード
- 第15章「RKE2」または第14章「K3s」 Kubernetesのアップグレード
- SUSE追加コンポーネント(SUSE Rancher Prime、SUSE Securityなど)のアップグレード

35.1.1 前提条件

管理クラスタをアップグレードする前に、次の前提条件が満たされている必要があります。

1. SCC登録ノード - クラスタのノードのOSが、アップグレードするEdgeリリース(52.1項「要約」)で指定されているOSバージョンをサポートするサブスクリプションキーで登録されていることを確認してください。
2. Upgrade Controller - Upgrade Controllerが管理クラスタにデプロイされていることを確認します。インストール手順については、22.3項「Upgrade Controllerのインストール」を参照してください。

35.1.2 アップグレード

1. 管理クラスタをアップグレードするEdgeリリース(52.1項「要約」)バージョンを決定します。
2. 管理クラスタに、目的のリリースバージョンを指定するUpgradePlanをデプロイします。UpgradePlanはUpgrade Controllerのネームスペースにデプロイする必要があります。

```
kubectl apply -n <upgrade_controller_namespace> -f - <<EOF
apiVersion: lifecycle.suse.com/v1alpha1
kind: UpgradePlan
metadata:
  name: upgrade-plan-mgmt
spec:
  # Version retrieved from release notes
  releaseVersion: 3.X.Y
EOF
```



注記

UpgradePlanに対して追加の設定を行いたいユースケースがある場合があります。すべての可能な設定については、22.5.1項「UpgradePlan」を参照してください。

3. Upgrade ControllerのネームスペースにUpgradePlanをデプロイすると、アップグレードプロセスが開始されます。



注記

実際のアップグレードプロセスの詳細については、22.4項「Upgrade Controllerの仕組み」を参照してください。

アップグレードプロセスを追跡する方法については、22.6項「アップグレードプロセスの追跡」を参照してください。

35.2 Fleet

このセクションでは、Fleet (第8章「Fleet」)コンポーネントを使用して「Day 2」操作を実行する方法について説明します。

次のトピックは、このセクションの一部として説明します。

1. 35.2.1項「コンポーネント」 - すべての「Day 2」操作に使用されるデフォルトコンポーネント。
2. 35.2.2項「ユースケースの決定」 - 使用されるFleetカスタムリソースの概要と、これらのリソースがさまざまな「Day 2」操作のユースケースに適しているかどうかを説明します。
3. 35.2.3項「Day 2ワークフロー」 - Fleetを使用して「Day 2」操作を実行するためのワークフローガイドを提供します。
4. 35.2.4項「OSアップグレード」 - Fleetを使用してOSアップグレードを実行する方法を説明します。
5. 35.2.5項「Kubernetesバージョンアップグレード」 - Fleetを使用してKubernetesバージョンのアップグレードを実行する方法を説明します。
6. 35.2.6項「Helmチャートのアップグレード」 - Fleetを使用してHelmチャートのアップグレードを実行する方法を説明します。

35.2.1 コンポーネント

以下に、Fleetを使用して「Day 2」操作を正常に実行できるように管理クラスタに設定する必要があるデフォルトコンポーネントの説明を記載しています。

35.2.1.1 Rancher

オプション: ダウンストリームクラスタを管理し、System Upgrade Controllerを管理クラスタ上にデプロイする責任。

詳細については、[第5章「Rancher」](#)を参照してください。

35.2.1.2 System Upgrade Controller (SUC)

System Upgrade Controllerは、Planと呼ばれるカスタムリソースを通じて提供される設定データに基づいて指定されたノードでタスクを実行する責任を負います。

SUCは、オペレーティングシステムとKubernetesディストリビューションのアップグレードに積極的に活用されます。

SUCコンポーネントとそれがEdgeスタックにどのように適合するかに関する詳細については、[第21章「System Upgrade Controller」](#)を参照してください。

35.2.2 ユースケースの決定

Fleetは2種類のカスタムリソース (<https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/>)  を使用してKubernetesおよびHelmリソースの管理を有効にします。

以下に、これらのリソースの目的と、「Day 2」操作のコンテキストに最適なユースケースに関する情報を示します。

35.2.2.1 GitRepo

GitRepoは、Fleetがバンドルの作成元として使用できるGitリポジトリを表すFleet (第8章「Fleet」) リソースです。各バンドルは、GitRepoリソースの内部で定義された設定パスに基づいて作成されます。詳細については、GitRepo (<https://fleet.rancher.io/gitrepo-add>)  のドキュメントを参照してください。

「Day 2」操作のコンテキストでは、GitRepoリソースは通常、**Fleet GitOps**アプローチを利用する**非エアギャップ**環境にSUCまたはSUC Planをデプロイするために使用されます。

または、**リポジトリのセットアップをローカルGitサーバ経由でミラーリングする場合**、GitRepoリソースを使用して、SUCまたはSUC Planを**エアギャップ**環境にデプロイすることもできます。

35.2.2.2 バンドル

バンドルは、ターゲットクラスタにデプロイする**生**のKubernetesリソースを保持します。バンドルは通常、GitRepoリソースから作成されますが、手動でデプロイできるユースケースもあります。詳細については、バンドル (<https://fleet.rancher.io/bundle-add>)  のドキュメントを参照してください。

「Day 2」操作のコンテキストでは、バンドルリソースは通常、何らかの形態の**ローカルGitOps**手法を使用しない**エアギャップ**環境(**ローカルGitサーバ**など)でSUCまたはSUC Planをデプロイするために使用されます。

または、ご自身のユースケースで**GitOps**ワークフローを使用できない場合は(Gitリポジトリを使用する場合など)、バンドルリソースを使用して、SUCまたはSUC Planを**非エアギャップ**環境にデプロイすることもできます。

35.2.3 Day 2ワークフロー

以下に、管理クラスタを特定のEdgeリリースにアップグレードする際に従う必要がある「Day 2」ワークフローを示します。

1. OSアップグレード(35.2.4項「OSアップグレード」)
2. Kubernetesバージョンアップグレード(35.2.5項「Kubernetesバージョンアップグレード」)
3. Helmチャートアップグレード(35.2.6項「Helmチャートのアップグレード」)

35.2.4 OSアップグレード

このセクションでは、第8章「Fleet」と第21章「System Upgrade Controller」を使用してオペレーティングシステムのアップグレードを行う方法について説明します。

次のトピックは、このセクションの一部として説明します。

1. 35.2.4.1項「コンポーネント」 - アップグレードプロセスで使用する追加のコンポーネント。
2. 35.2.4.2項「概要」 - アップグレードプロセスの概要。
3. 35.2.4.3項「要件」 - アップグレードプロセスの要件。
4. 35.2.4.4項「OSアップグレード - SUC Planのデプロイメント」 - SUC Planのデプロイ方法に関する情報。このプランはアップグレードプロセスをトリガする責任を負います。

35.2.4.1 コンポーネント

このセクションでは、OSアップグレードプロセスがデフォルトの「Day 2」コンポーネント(35.2.1項「コンポーネント」)よりも使用するカスタムコンポーネントについて説明します。

35.2.4.1.1 systemd.service

特定のノードでのOSアップグレードは、[systemd.service \(https://www.freedesktop.org/software/systemd/man/latest/systemd.service.html\)](https://www.freedesktop.org/software/systemd/man/latest/systemd.service.html) によって処理されます。

あるEdgeバージョンから別のEdgeバージョンにOSが必要とするアップグレードのタイプに応じて、異なるサービスが作成されます。

- 同じOSバージョン(例: 6.0)を必要とするEdgeバージョンの場合、`os-pkg-update.service`が作成されます。このサービスは[transactional-update \(https://kubic.opensuse.org/documentation/man-pages/transactional-update.8.html\)](https://kubic.opensuse.org/documentation/man-pages/transactional-update.8.html) を使用して、通常のパッケージアップグレード (https://en.opensuse.org/SDB:Zypper_usage#Updating_packages) を実行します。
- OSバージョンのマイグレーション(例: 6.0 → 6.1)が必要なEdgeバージョンの場合、`os-migration.service`が作成されます。このサービスは、[transactional-update \(https://kubic.opensuse.org/documentation/man-pages/transactional-update.8.html\)](https://kubic.opensuse.org/documentation/man-pages/transactional-update.8.html) を使用して以下の処理を実行します。
 - a. 通常のパッケージのアップグレード (https://en.opensuse.org/SDB:Zypper_usage#Updating_packages) 。このアップグレードにより、すべてのパッケージを最新にすることで、古いパッケージバージョンに関連するマイグレーション時の障害を軽減します。
 - b. `zypper migration`コマンドを利用したOSマイグレーション。

上記サービスは、OSアップグレードが必要な管理クラスタ上に配置されている必要がある SUC Planを通じて各ノードに配布されます。

35.2.4.2 概要

管理クラスタノードのオペレーティングシステムのアップグレードは、FleetとSystem Upgrade Controller (SUC)を利用して実行されます。

Fleetは、SUC Planを目的のクラスタにデプロイおよび管理するために使用されます。



注記

SUC Planは、特定のタスクをノードセットで実行するために SUC が従う必要がある手順を記述した カスタムリソース (<https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/>)  です。 SUC Plan の例については、アップストリームリポジトリ (<https://github.com/rancher/system-upgrade-controller?tab=readme-ov-file#example-plans>)  を参照してください。

OS SUC Planは、GitRepo (<https://fleet.rancher.io/gitrepo-add>) またはバンドル (<https://fleet.rancher.io/bundle-add>) リソースを特定のFleetワークスペース (<https://fleet.rancher.io/namespaces#gitrepos-bundles-clusters-clustergroups>) にデプロイすることで、各クラスタに配布されます。FleetはデプロイされたGitRepo/Bundleを取得し、その内容 (OS SUC plan) を目的のクラスタにデプロイします。



注記

GitRepo/バンドルリソースは常に、管理クラスタ上にデプロイされます。GitRepoリソースを使用するかバンドルリソースを使用するかは、ユースケースによって異なります。詳細については、[35.2.2項「ユースケースの決定」](#)をご確認ください。

OS SUC Planは、次のワークフローを記述します。

1. 常に、OSをアップグレードする前に、ノードを[cordon](https://kubernetes.io/docs/reference/kubectl/generated/kubectl_cordon/) (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_cordon/) します。
2. 常に、ワーカーノードの前にコントロールプレーンノードをアップグレードします。
3. 常に、一度に1ノードずつクラスタをアップグレードします。

OS SUC Planがデプロイされると、ワークフローは次のようになります。

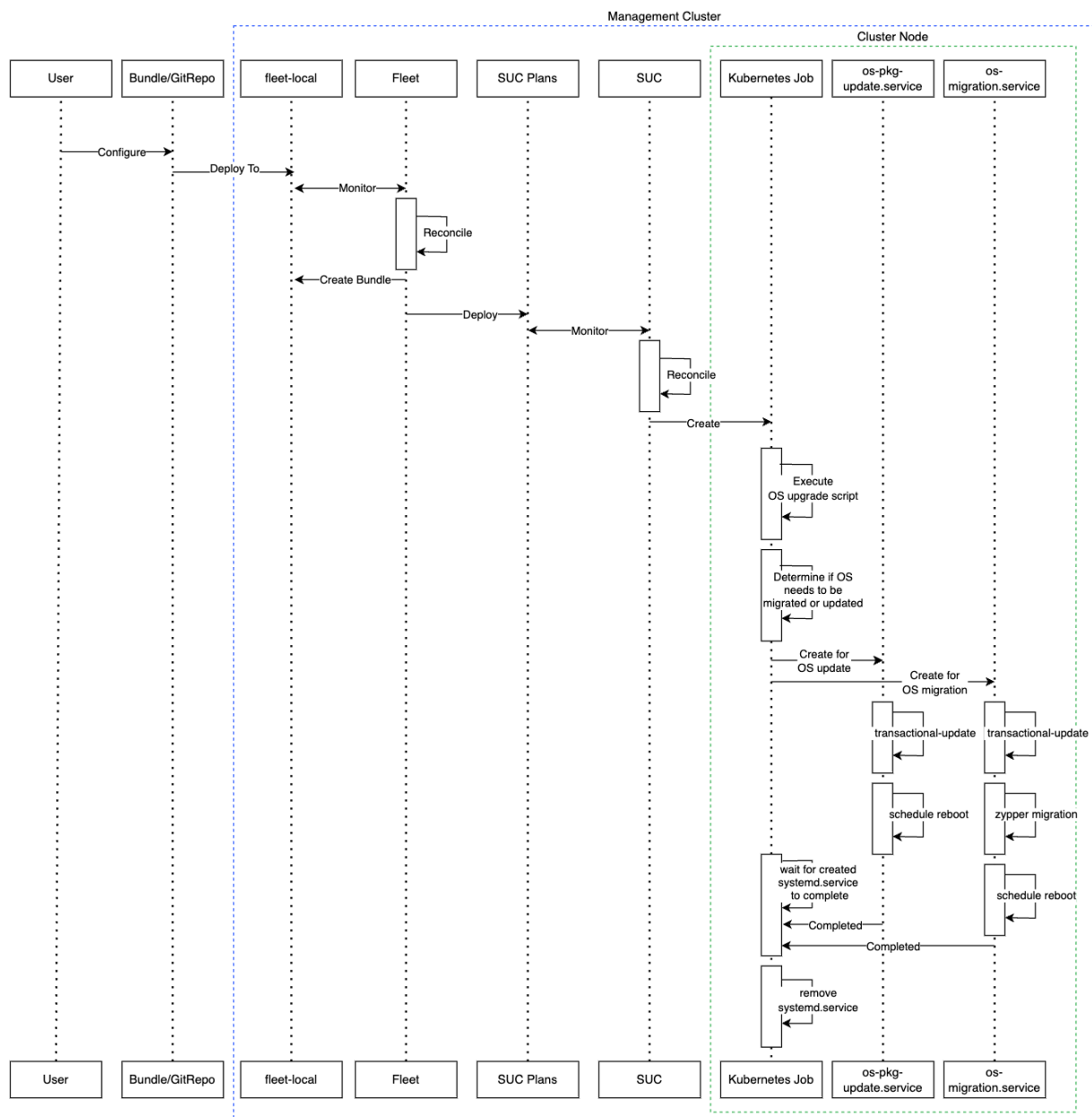
1. SUCは、デプロイされたOS SUC Planを照合し、Kubernetes Jobを**各ノード**に作成します。
2. Kubernetes Jobは、パッケージのアップグレードまたはOSマイグレーションのためにsystemd.service ([35.2.4.1.1項「systemd.service」](#))を作成します。
3. 作成されたsystemd.serviceは、特定のノードでOSアップグレードプロセスをトリガします。



重要

OSアップグレードプロセスが終了すると、対応するノードが再起動され、システムに更新が適用されます。

上記の説明を以下に図示します。



35.2.4.3 要件

全般:

1. **SCC登録マシン** - すべての管理クラスタノードを<https://scc.suse.com/>に登録する必要があります。これは、各systemd.serviceが目的のRPMリポジトリに正常に接続できるようにするために必要です。



重要

OSバージョンのマイグレーション (例: 6.0 → 6.1)が必要なEdgeリリースの場合、SCCキーが新しいバージョンへのマイグレーションをサポートしていることを確認してください。

2. **SUC PlanのTolerationがノードのTolerationと一致すること** - KubernetesクラスターノードにカスタムのTaintが設定されている場合は、**SUC Plan**にそのTaintに対するToleration (<https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/>) ➦を追加してください。デフォルトでは、**SUC Plan**には、**コントロールプレーン**ノードのTolerationのみが含まれます。デフォルトのTolerationは次のとおりです。

- **CriticalAddonsOnly=true:NoExecute**
- **node-role.kubernetes.io/control-plane:NoSchedule**
- **node-role.kubernetes.io/etcd:NoExecute**



注記

追加のTolerationは、各Planの.spec.tolerationsセクションに追加する必要があります。OSアップグレードに関連する**SUC Plan**は、[fleets/day2/system-upgrade-controller-plans/os-upgrade](https://github.com/suse-edge/fleet-examples)の下での[suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples) (<https://github.com/suse-edge/fleet-examples>) ➦リポジトリにあります。**有効なリポジトリリリース** (<https://github.com/suse-edge/fleet-examples/releases>) ➦タグからのPlanを使用してください。

control-plane SUC Planに対してカスタムのTolerationを定義する例は次のようになります。

```
apiVersion: upgrade.cattle.io/v1
kind: Plan
metadata:
  name: os-upgrade-control-plane
spec:
  ...
  tolerations:
    # default tolerations
    - key: "CriticalAddonsOnly"
      operator: "Equal"
      value: "true"
      effect: "NoExecute"
```

```
- key: "node-role.kubernetes.io/control-plane"
  operator: "Equal"
  effect: "NoSchedule"
- key: "node-role.kubernetes.io/etcd"
  operator: "Equal"
  effect: "NoExecute"
# custom toleration
- key: "foo"
  operator: "Equal"
  value: "bar"
  effect: "NoSchedule"
...
```

エアギャップ:

1. **SUSE RPMリポジトリのミラーリング** - OS RPMリポジトリをローカルにミラーリングし、`systemd.service`がそのリポジトリにアクセスできるようにする必要があります。このためには、[RMT \(https://documentation.suse.com/sles/15-SP6/html/SLES-all/book-rmt.html\)](https://documentation.suse.com/sles/15-SP6/html/SLES-all/book-rmt.html) または [SUMA \(https://documentation.suse.com/suma/5.0/en/suse-manager/index.html\)](https://documentation.suse.com/suma/5.0/en/suse-manager/index.html) のいずれかを使用します。

35.2.4.4 OSアップグレード - SUC Planのデプロイメント

! 重要

この手順を使用して以前にアップグレードした環境の場合、ユーザは次の手順の**いずれか**を完了していることを確認する必要があります。

- 管理クラスタから古いEdgeリリースバージョンに関連する以前にデプロイしたSUC Planを削除する - これは、既存のGitRepo/バンドル ターゲット設定 (<https://fleet.rancher.io/gitrepo-targets#target-matching>) から目的のクラスタを削除するか、GitRepo/バンドル リソースを完全に削除することで、実行できます。
- 既存のGitRepo/バンドルリソースを再利用する - 目的の suse-edge/fleet-examples リリース (<https://github.com/suse-edge/fleet-examples/releases>) の正しいフリートを保持する新しいタグにリソースのリビジョンをポイントすることで実行できます。

これは古いEdgeリリースバージョンのSUC Plan間のクラッシュを回避するために実行されます。

ユーザがアップグレードを試す場合、管理クラスタに既存のSUC Planがある場合は、次のフリートエラーが表示されます。

```
Not installed: Unable to continue with install: Plan <plan_name> in namespace
<plan_namespace> exists and cannot be imported into the current release: invalid
ownership metadata; annotation validation error..
```

35.2.4.2項「概要」で説明したように、OSアップグレードは、次のいずれかの方法を使用してSUC Planを目的のクラスタに配布することで実行されます。

- Fleet GitRepo リソース - 35.2.4.4.1項「SUC Planのデプロイメント - GitRepoリソース」。
- Fleet バンドル リソース - 35.2.4.4.2項「SUC Planのデプロイメント - バンドルリソース」。

どのリソースを使用すべきかを判断するには、35.2.2項「ユースケースの決定」を参照してください。

OS SUC PlanをサードパーティのGitOpsツールからデプロイするユースケースの場合は、35.2.4.4.3項「SUC Planのデプロイメント - サードパーティのGitOpsワークフロー」を参照してください。

35.2.4.4.1 SUC Planのデプロイメント - GitRepoリソース

必要なOS SUC Planを配布する、**GitRepo**リソースは、次の方法のいずれかでデプロイできます。

1. Rancher UI - 35.2.4.4.1.1項「GitRepoの作成 - Rancher UI」を通じて(Rancherが利用可能な場合)。
2. リソースを管理クラスタに手動でデプロイする(35.2.4.4.1.2項「GitRepoの作成 - 手動」)。

デプロイ後に、ターゲットクラスタのノードのOSアップグレードプロセスを監視するには、21.3項「System Upgrade Controller Planのモニタリング」を参照してください。

35.2.4.4.1.1 GitRepoの作成 - Rancher UI

Rancher UIを通じてGitRepoリソースを作成するには、公式のドキュメント (<https://ranchermanager.docs.rancher.com/v2.12/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui>) に従ってください。

Edgeチームは、すぐに使用できるフリート (<https://github.com/suse-edge/fleet-examples/tree/release-3.4.0/fleets/day2/system-upgrade-controller-plans/os-upgrade>) を維持しています。環境によっては、このフリートを直接使用することも、テンプレートとして使用することもできます。

！ 重要

常に、このフリートは有効なEdge リリース (<https://github.com/suse-edge/fleet-examples/releases>) タグから使用してください。

フリートが配布するSUC Planにカスタム変更を含める必要がないユースケースでは、ユーザは[suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples)リポジトリから [os-upgrade](https://github.com/suse-edge/fleet-examples/tree/release-3.4.0/fleets/day2/system-upgrade-controller-plans/os-upgrade)フリートを直接参照できます。

カスタム変更(カスタム許容値の追加など)が必要な場合、ユーザは別のリポジトリから[os-upgrade](https://github.com/suse-edge/fleet-examples/tree/release-3.4.0/fleets/day2/system-upgrade-controller-plans/os-upgrade)フリートを参照して、必要に応じてSUC Planに変更を追加できるようにする必要があります。

GitRepoを[suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples)リポジトリのフリートを使用するように設定する方法の例については、[こちら](https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/gitrepos/day2/os-upgrade-gitrepo.yaml) (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/gitrepos/day2/os-upgrade-gitrepo.yaml>) を参照してください。

35.2.4.4.1.2 GitRepoの作成 - 手動

1. GitRepoリソースをプルします。

```
curl -o os-upgrade-gitrepo.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.4.0/gitrepos/day2/os-upgrade-gitrepo.yaml
```

2. GitRepo設定を編集します。

- `spec.targets`セクションを削除します。ダウンストリームクラスタにのみ必要です。

```
# Example using sed
sed -i.bak '/^ targets:/, $d' os-upgrade-gitrepo.yaml && rm -f os-upgrade-gitrepo.yaml.bak

# Example using yq (v4+)
```

```
yq eval 'del(.spec.targets)' -i os-upgrade-gitrepo.yaml
```

- **GitRepo**のネームスペースを`fleet-local`ネームスペースにポイントします。これは、リソースを管理クラスタにデプロイするために実行します。

```
# Example using sed
sed -i.bak 's/namespace: fleet-default/namespace: fleet-local/' os-upgrade-
gitrepo.yaml && rm -f os-upgrade-gitrepo.yaml.bak

# Example using yq (v4+)
yq eval '.metadata.namespace = "fleet-local"' -i os-upgrade-gitrepo.yaml
```

3. **GitRepo**リソースを管理クラスタに適用します。

```
kubectl apply -f os-upgrade-gitrepo.yaml
```

4. `fleet-local`ネームスペースの下に、作成した **GitRepo**リソースを表示します。

```
kubectl get gitrepo os-upgrade -n fleet-local

# Example output
NAME                                REPO                                COMMIT
BUNDLEDEPLOYMENTS-READY    STATUS
os-upgrade                    https://github.com/suse-edge/fleet-examples.git  release-3.4.0  0/0
```

35.2.4.4.2 **SUC Planのデプロイメント - バンドルリソース**

必要なOS `SUC Plan`を配布する**バンドル**リソースは、次の方法のいずれかでデプロイできます。

1. **Rancher UI** - 35.2.4.4.2.1項「**バンドルの作成 - Rancher UI**」を通じて(**Rancher**が利用可能な場合)。
2. リソースを**管理クラスタ**に手動でデプロイする(35.2.4.4.2.2項「**バンドルの作成 - 手動**」)。

デプロイ後に、ターゲットクラスタのノードのOSアップグレードプロセスを監視するには、21.3項「**System Upgrade Controller Planのモニタリング**」を参照してください。

35.2.4.4.2.1 **バンドルの作成 - Rancher UI**

Edgeチームは、以下の手順で使用可能な、すぐに使用できる**バンドル** (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/bundles/day2/system-upgrade-controller-plans/os-upgrade/os-upgrade-bundle.yaml>) を維持しています。

！ 重要

このバンドルは常に有効なEdgeリリース (<https://github.com/suse-edge/fleet-examples/releases>) タグから使用してください。

RancherのUIを通じてバンドルを作成するには、次の手順に従います。

1. 左上隅で、 **[#]** → **[Continuous Delivery (継続的デリバリー)]** をクリックします。
2. **[Advanced (詳細)]** > **[Bundles (バンドル)]** に移動します。
3. **[Create from YAML (YAMLから作成)]** を選択します。
4. ここから次のいずれかの方法でバンドルを作成できます。

📎 注記

バンドルが配布するSUC Planにカスタム変更を含める必要があるユースケースがある場合があります(カスタム許容値の追加など)。これらの変更を、以下の手順で生成されるバンドルに必ず含めてください。

- a. バンドルコンテンツ (<https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.4.0/bundles/day2/system-upgrade-controller-plans/os-upgrade/os-upgrade-bundle.yaml>) を [suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples) から **[Create from YAML (YAMLから作成)]** ページに手動でコピーする。
- b. 目的のリリース (<https://github.com/suse-edge/fleet-examples/releases>) タグから [suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples) (<https://github.com/suse-edge/fleet-examples>) のクローンを作成し、**[Create from YAML (YAMLから作成)]** ページの **[Read from File (ファイルから読み取り)]** オプションを選択する。ここからバンドルの場所([bundles/day2/system-upgrade-controller-plans/os-upgrade](https://github.com/suse-edge/fleet-examples/blob/main/bundles/day2/system-upgrade-controller-plans/os-upgrade/os-upgrade-bundle.yaml))に移動して、バンドルファイルを選択できます。これにより、バンドルコンテンツを持つ **[Create from YAML (YAMLから作成)]** ページが自動入力されます。

5. Rancher UIでバンドルを編集します。

- バンドルのネームスペースを `fleet-local` ネームスペースを指すように変更します。

```
# Example
kind: Bundle
```

```
apiVersion: fleet.cattle.io/v1alpha1
metadata:
  name: os-upgrade
  namespace: fleet-local
...
```

- バンドルの**ターゲット**クラスタをローカル(管理)クラスタを指すように変更します。

```
spec:
  targets:
    - clusterName: local
```



注記

一部のユースケースでは、ローカルクラスタに異なる名前を付けることができます。

ローカルクラスタ名を取得するには、次のコマンドを実行します。

```
kubectl get clusters.fleet.cattle.io -n fleet-local
```

6. [Create (作成)] を選択します。

35.2.4.4.2.2 バンドルの作成 - 手動

1. バンドルリソースをプルします。

```
curl -o os-upgrade-bundle.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.4.0/bundles/day2/system-upgrade-controller-plans/os-upgrade/os-upgrade-bundle.yaml
```

2. バンドル設定を編集します。

- バンドルの**ターゲット**クラスタをローカル(管理)クラスタを指すように変更します。

```
spec:
  targets:
    - clusterName: local
```



注記

一部のユースケースでは、ローカルクラスタに異なる名前を付けることができます。

ローカルクラスタ名を取得するには、次のコマンドを実行します。

```
kubectl get clusters.fleet.cattle.io -n fleet-local
```

- バンドルのネームスペースをfleet-localネームスペースを指すように変更します。

```
# Example
kind: Bundle
apiVersion: fleet.cattle.io/v1alpha1
metadata:
  name: os-upgrade
  namespace: fleet-local
...
```

3. バンドルリソースを管理クラスタに適用します。

```
kubectl apply -f os-upgrade-bundle.yaml
```

4. 作成したバンドルリソースをfleet-localネームスペースの下に表示します。

```
kubectl get bundles -n fleet-local
```

35.2.4.4.3 SUC Planのデプロイメント - サードパーティのGitOpsワークフロー

ユーザがOS SUC Planを独自のサードパーティGitOpsワークフロー(例: Flux)に組み込むユースケースがある場合があります。

必要なOSアップグレードリソースを取得するには、まず使用する[suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples) (<https://github.com/suse-edge/fleet-examples>)  リポジトリのEdge リリース (<https://github.com/suse-edge/fleet-examples/releases>)  タグを決定します。

その後、リソースは[fleets/day2/system-upgrade-controller-plans/os-upgrade](#)で見つかります。ここでは次のようになります。

- plan-control-plane.yamlは、**コントロールプレーン**ノードのSUC Planリソースです。
- plan-worker.yamlは、**ワーカー**ノードのSUC Planリソースです。
- secret.yamlは、systemd.service (35.2.4.1.1項「[systemd.service](#)」)の作成を担当する、upgrade.shスクリプトを含むSecretです。
- config-map.yamlは、upgrade.shスクリプトによって使用される設定を保持するConfigMapです。

！ 重要

これらのPlanリソースは、System Upgrade Controllerによって解釈され、アップグレードする各ダウンストリームクラスタにデプロイする必要があります。SUCデプロイメントの情報については、[21.2項「System Upgrade Controllerのインストール」](#)を参照してください。

GitOpsワークフローをOSアップグレードの**SUC Plan**をデプロイするために使用する方法をよりよく理解するために、概要([35.2.4.2項「概要」](#))を確認すると役立つ場合があります。

35.2.5 Kubernetesバージョンアップグレード

このセクションでは、[第8章「Fleet」](#)と[第21章「System Upgrade Controller」](#)を使用してKubernetesアップグレードを実行する方法について説明します。

次のトピックは、このセクションの一部として説明します。

1. [35.2.5.1項「コンポーネント」](#) - アップグレードプロセスで使用する追加のコンポーネント。
2. [35.2.5.2項「概要」](#) - アップグレードプロセスの概要。
3. [35.2.5.3項「要件」](#) - アップグレードプロセスの要件。
4. [35.2.5.4項「K8sアップグレード - SUC Planのデプロイメント」](#) - SUC Planのデプロイ方法に関する情報。このプランはアップグレードプロセスをトリガする責任を負います。

35.2.5.1 コンポーネント

このセクションでは、K8sアップグレードプロセスがデフォルトの「Day 2」コンポーネント([35.2.1項「コンポーネント」](#))よりも使用するカスタムコンポーネントについて説明します。

35.2.5.1.1 rke2-upgrade

特定のノードのRKE2バージョンのアップグレードを担当するコンテナイメージ。

SUC Planに基づいて**SUC**によって作成されたPodを通じて配布されます。このPlanは、RKE2のアップグレードが必要な各**クラスタ**に配置する必要があります。

rke2-upgradeイメージによるアップグレードの実行方法の詳細については、[アップストリーム \(https://github.com/rancher/rke2-upgrade/tree/master\)](https://github.com/rancher/rke2-upgrade/tree/master)  ドキュメントを参照してください。

35.2.5.1.2 k3s-upgrade

特定のノードのK3sバージョンのアップグレードを担当するコンテナイメージ。

SUC Planに基づいて**SUC**によって作成されたPodを通じて配布されます。このPlanは、K3sのアップグレードが必要な各**クラスタ**に配置する必要があります。

k3s-upgradeイメージによるアップグレードの実行方法の詳細については、[アップストリーム \(https://github.com/k3s-io/k3s-upgrade\)](https://github.com/k3s-io/k3s-upgrade)  ドキュメントを参照してください。

35.2.5.2 概要

管理クラスタノードのKubernetesディストリビューションのアップグレードは、[Fleet](#)とSystem Upgrade Controller (SUC)を利用して実行されます。

[Fleet](#)は、[SUC Plan](#)を目的のクラスタにデプロイし、管理するために使用されます。



注記

SUC Planは、**SUC**が特定のタスクをノードのセットで実行するために従う必要がある手順を記述した**カスタムリソース** (<https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/>)  です。SUC Planの例については、[アップストリームリポジトリ \(https://github.com/rancher/system-upgrade-controller?tab=readme-ov-file#example-plans\)](https://github.com/rancher/system-upgrade-controller?tab=readme-ov-file#example-plans)  を参照してください。

K8s SUC Planは、[GitRepo \(https://fleet.rancher.io/gitrepo-add\)](https://fleet.rancher.io/gitrepo-add)  または**バンドル** (<https://fleet.rancher.io/bundle-add>)  リソースを特定のFleetワークスペース (<https://fleet.rancher.io/namespaces#gitrepos-bundles-clusters-clustergroups>)  にデプロイすることで各クラスタに配布されます。Fleetはデプロイされた[GitRepo/バンドル](#)を取得し、その内容(K8s SUC Plan)を目的のクラスタにデプロイします。



注記

GitRepo/バンドルリソースは常に、管理クラスタ上にデプロイされます。GitRepoリソースを使用するかバンドルリソースを使用するかは、ユースケースによって異なります。詳細については、[35.2.2項「ユースケースの決定」](#)をご確認ください。

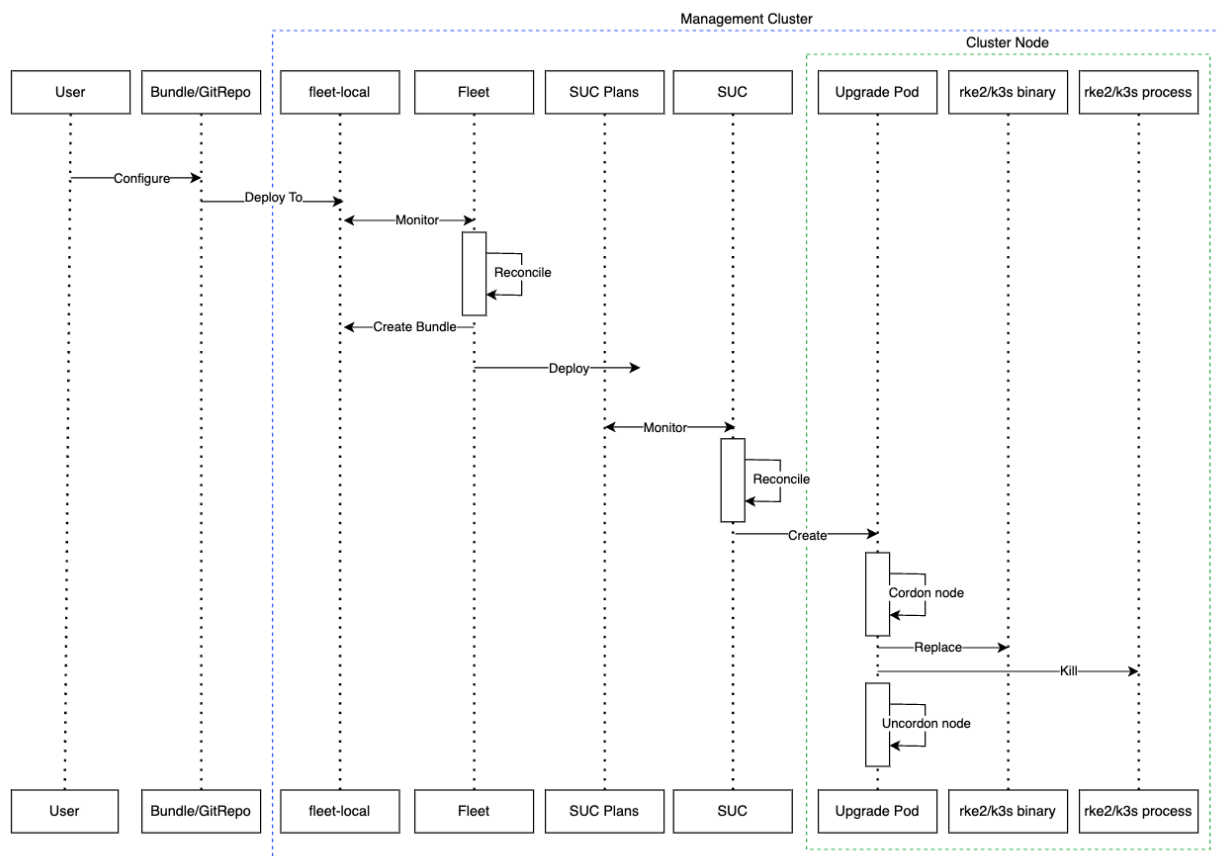
K8s SUC Planは、次のワークフローを記述します。

1. 常に、K8sをアップグレードする前に、ノードをcordon (https://kubernetes.io/docs/reference/kubect/generated/kubectl_cordon/) [🔗](#)します。
2. 常に、ワーカーノードの前にコントロールプレーンノードをアップグレードします。
3. 常に、一度に1ノードずつコントロールプレーンノードをアップグレードし、一度に2ノードずつワーカーノードをアップグレードします。

K8s SUC Planがデプロイされたら、ワークフローは次のようになります。

1. SUCは、デプロイ済みのK8s SUC Planを照合し、Kubernetes Jobを**各ノード**に作成します。
2. Kubernetesディストリビューションによって、Jobはrke2-upgrade ([35.2.5.1.1項「rke2-upgrade」](#))またはk3s-upgrade ([35.2.5.1.2項「k3s-upgrade」](#))のコンテナイメージを実行するPodを作成します。
3. 作成されたPodは次のワークフローを実行します。
 - a. ノード上の既存のrke2/k3sバイナリをrke2-upgrade/k3s-upgradeイメージのバイナリで置き換えます。
 - b. 実行中のrke2/k3sプロセスを強制終了します。
4. rke2/k3sプロセスを強制終了すると、再起動がトリガされ、更新されたバイナリを実行する新しいプロセスが起動し、Kubernetesディストリビューションバージョンがアップグレードされます。

上記の説明を以下に図示します。



35.2.5.3 要件

1. Kubernetesディストリビューションをバックアップします。

- RKE2クラスタについては、RKE2のバックアップと復元 (https://docs.rke2.io/datastore/backup_restore) に関するドキュメントを参照してください。
- K3sクラスタについては、K3sのバックアップと復元 (<https://docs.k3s.io/datastore/backup-restore>) に関するドキュメントを参照してください。

2. SUC PlanのTolerationがノードのTolerationと一致すること - KubernetesクラスタノードにカスタムのTaintが設定されている場合は、SUC PlanにそのTaintに対するToleration (<https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/>) を追加してください。デフォルトでは、SUC Planには、control-planeノードのTolerationのみが含まれます。デフォルトのTolerationは次のとおりです。

- **CriticalAddonsOnly=true:NoExecute**
- **node-role.kubernetes.io/control-plane:NoSchedule**
- **node-role.kubernetes.io/etcd:NoExecute**



注記

追加のTolerationは、各Planの`.spec.tolerations`セクションに追加する必要があります。Kubernetesバージョンアップグレードに関する**SUC Plan**は、次の場所の[suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples) (<https://github.com/suse-edge/fleet-examples>) リポジトリにあります。

- **RKE2** - [fleets/day2/system-upgrade-controller-plans/rke2-upgrade](#)
- **K3s** - [fleets/day2/system-upgrade-controller-plans/k3s-upgrade](#)

必ず、有効なリポジトリリリース (<https://github.com/suse-edge/fleet-examples/releases>) **タグからのPlan**を使用してください。

RKE2 **control-plane** SUC PlanのカスタムTolerationの定義例は次のようになります。

```
apiVersion: upgrade.cattle.io/v1
kind: Plan
metadata:
  name: rke2-upgrade-control-plane
spec:
  ...
  tolerations:
    # default tolerations
    - key: "CriticalAddonsOnly"
      operator: "Equal"
      value: "true"
      effect: "NoExecute"
    - key: "node-role.kubernetes.io/control-plane"
      operator: "Equal"
      effect: "NoSchedule"
    - key: "node-role.kubernetes.io/etcd"
      operator: "Equal"
      effect: "NoExecute"
    # custom toleration
    - key: "foo"
      operator: "Equal"
      value: "bar"
      effect: "NoSchedule"
  ...
```

35.2.5.4 K8sアップグレード - SUC Planのデプロイメント

！ 重要

この手順を使用して以前にアップグレードした環境の場合、ユーザは次の手順の**いずれか**を完了していることを確認する必要があります。

- 管理クラスタから古いEdgeリリースバージョンに関連する以前にデプロイしたSUC Planを削除する - これは、既存のGitRepo/バンドル ターゲット設定 (<https://fleet.rancher.io/gitrepo-targets#target-matching>) から目的のクラスタを削除するか、GitRepo/バンドルリソースを完全に削除することで、実行できます。
- 既存のGitRepo/バンドルリソースを再利用する - 目的の suse-edge/fleet-examples リリース (<https://github.com/suse-edge/fleet-examples/releases>) の正しいフリートを保持する新しいタグにリソースのリビジョンをポイントすることで実行できます。

これは古いEdgeリリースバージョンの SUC Plan間のクラッシュを回避するために実行されます。

ユーザがアップグレードを試す場合、管理クラスタに既存の SUC Planがある場合は、次のフリートエラーが表示されます。

```
Not installed: Unable to continue with install: Plan <plan_name> in namespace <plan_namespace> exists and cannot be imported into the current release: invalid ownership metadata; annotation validation error..
```

35.2.5.2項「概要」で説明したように、Kubernetesアップグレードは、次のいずれかの方法を使用して SUC Planを目的のクラスタに配布することで実行されます。

- Fleet GitRepoリソース(35.2.5.4.1項「SUC Planのデプロイメント - GitRepoリソース」)
- Fleetバンドルリソース(35.2.5.4.2項「SUC Planのデプロイメント - バンドルリソース」)

どのリソースを使用すべきかを判断するには、35.2.2項「ユースケースの決定」を参照してください。

K8s SUC PlanをサードパーティのGitOpsツールからデプロイするユースケースの場合は、35.2.5.4.3項「SUC Planのデプロイメント - サードパーティのGitOpsワークフロー」を参照してください。

35.2.5.4.1 SUC Planのデプロイメント - GitRepoリソース

必要なK8s SUC Planを配布する、**GitRepo**リソースは、次の方法のいずれかでデプロイできます。

1. Rancher UI - 35.2.5.4.1.1項「GitRepoの作成 - Rancher UI」を通じて(Rancherが利用可能な場合)。
2. リソースを管理クラスタに手動でデプロイする(35.2.5.4.1.2項「GitRepoの作成 - 手動」)。

デプロイ後に、ターゲットクラスタのノードのKubernetesアップグレードプロセスを監視するには、21.3項「System Upgrade Controller Planのモニタリング」を参照してください。

35.2.5.4.1.1 GitRepoの作成 - Rancher UI

Rancher UIを通じてGitRepoリソースを作成するには、公式のドキュメント(<https://ranchermanager.docs.rancher.com/v2.12/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui>)に従ってください。

Edgeチームは、rke2 (<https://github.com/suse-edge/fleet-examples/tree/release-3.4.0/fleets/day2/system-upgrade-controller-plans/rke2-upgrade>)とk3s (<https://github.com/suse-edge/fleet-examples/tree/release-3.4.0/fleets/day2/system-upgrade-controller-plans/k3s-upgrade>) Kubernetesディストリビューションの両方のすぐに使用できるフリートを維持しています。環境によっては、このフリートを直接使用することも、テンプレートとして使用することもできます。

! 重要

常に、これらのフリートは有効なEdgeリリース (<https://github.com/suse-edge/fleet-examples/releases>) タグから使用してください。

これらのフリートが配布するSUC Planにカスタム変更を含める必要がないユースケースの場合、ユーザはsuse-edge/fleet-examplesリポジトリからフリートを直接参照できます。

カスタム変更(カスタム許容値の追加など)が必要な場合、ユーザは別のリポジトリからフリートを参照して、必要に応じてSUC Planに変更を追加できるようにする必要があります。

suse-edge/fleet-examplesリポジトリからフリートを使用したGitRepoリソースの設定例は、次のとおりです。

- RKE2 (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/gitrepos/day2/rke2-upgrade-gitrepo.yaml>) ↗
- K3s (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/gitrepos/day2/k3s-upgrade-gitrepo.yaml>) ↗

35.2.5.4.1.2 GitRepoの作成 - 手動

1. GitRepoリソースをプルします。

- RKE2クラスタの場合:

```
curl -o rke2-upgrade-gitrepo.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.4.0/gitrepos/day2/rke2-upgrade-gitrepo.yaml
```

- K3sクラスタの場合:

```
curl -o k3s-upgrade-gitrepo.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.4.0/gitrepos/day2/k3s-upgrade-gitrepo.yaml
```

2. GitRepo設定を編集します。

- spec.targetsセクションを削除します。ダウンストリームクラスタにのみ必要です。

- RKE2の場合:

```
# Example using sed
sed -i.bak '/^ targets:/$d' rke2-upgrade-gitrepo.yaml && rm -f rke2-upgrade-gitrepo.yaml.bak

# Example using yq (v4+)
yq eval 'del(.spec.targets)' -i rke2-upgrade-gitrepo.yaml
```

- K3sの場合:

```
# Example using sed
sed -i.bak '/^ targets:/$d' k3s-upgrade-gitrepo.yaml && rm -f k3s-upgrade-gitrepo.yaml.bak

# Example using yq (v4+)
yq eval 'del(.spec.targets)' -i k3s-upgrade-gitrepo.yaml
```

- GitRepoのネームスペースをfleet-localネームスペースにポイントします。これは、リソースを管理クラスタにデプロイするために実行します。

- RKE2の場合:

```
# Example using sed
sed -i.bak 's/namespace: fleet-default/namespace: fleet-local/' rke2-
upgrade-gitrepo.yaml && rm -f rke2-upgrade-gitrepo.yaml.bak

# Example using yq (v4+)
yq eval '.metadata.namespace = "fleet-local"' -i rke2-upgrade-gitrepo.yaml
```

- K3sの場合:

```
# Example using sed
sed -i.bak 's/namespace: fleet-default/namespace: fleet-local/' k3s-
upgrade-gitrepo.yaml && rm -f k3s-upgrade-gitrepo.yaml.bak

# Example using yq (v4+)
yq eval '.metadata.namespace = "fleet-local"' -i k3s-upgrade-gitrepo.yaml
```

3. GitRepoリソースを管理クラスタに適用します。

```
# RKE2
kubectl apply -f rke2-upgrade-gitrepo.yaml

# K3s
kubectl apply -f k3s-upgrade-gitrepo.yaml
```

4. fleet-localネームスペースの下に、作成した **GitRepo** リソースを表示します。

```
# RKE2
kubectl get gitrepo rke2-upgrade -n fleet-local

# K3s
kubectl get gitrepo k3s-upgrade -n fleet-local

# Example output
NAME                                REPO                                COMMIT
BUNDLEDEPLOYMENTS-READY  STATUS
k3s-upgrade                https://github.com/suse-edge/fleet-examples.git  fleet-local  0/0
rke2-upgrade                https://github.com/suse-edge/fleet-examples.git  fleet-local  0/0
```

35.2.5.4.2 SUC Planのデプロイメント - バンドルリソース

必要なKubernetesアップグレードSUC Planを配布する、バンドルリソースは、次の方法のいずれかでデプロイできます。

1. Rancher UI - 35.2.5.4.2.1項「バンドルの作成 - Rancher UI」を通じて(Rancher が利用可能な場合)。
2. リソースを管理クラスタに手動でデプロイする(35.2.5.4.2.2項「バンドルの作成 - 手動」)。

デプロイ後に、ターゲットクラスタのノードのKubernetesアップグレードプロセスを監視するには、21.3項「System Upgrade Controller Planのモニタリング」を参照してください。

35.2.5.4.2.1 バンドルの作成 - Rancher UI

Edgeチームは、rke2 (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/bundles/day2/system-upgrade-controller-plans/rke2-upgrade/plan-bundle.yaml>) と k3s (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/bundles/day2/system-upgrade-controller-plans/k3s-upgrade/plan-bundle.yaml>)  Kubernetesディストリビューションの両方のすぐに使用できるバンドルを維持しています。環境によっては、これらのバンドルを直接使用することも、テンプレートとして使用することもできます。

重要

このバンドルは常に有効なEdgeリリース (<https://github.com/suse-edge/fleet-examples/releases>)  タグから使用してください。

RancherのUIを通じてバンドルを作成するには、次の手順に従います。

1. 左上隅で、**[#]** → **[Continuous Delivery (継続的デリバリ)]** をクリックします。
2. **[Advanced (詳細)]** > **[Bundles (バンドル)]** に移動します。
3. **[Create from YAML (YAMLから作成)]** を選択します。
4. ここから次のいずれかの方法でバンドルを作成できます。

注記

バンドルが配布するSUC Planにカスタム変更を含める必要があるユースケースがある場合があります(カスタム許容値の追加など)。これらの変更を、以下の手順で生成されるバンドルに必ず含めてください。

- a. [suse-edge/fleet-examples](#)から **【Create from YAML (YAMLから作成)】** ページに RKE2 (<https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.4.0/bundles/day2/system-upgrade-controller-plans/rke2-upgrade/plan-bundle.yaml>) または K3s (<https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.4.0/bundles/day2/system-upgrade-controller-plans/k3s-upgrade/plan-bundle.yaml>) のバンドルコンテンツを手動でコピーする。
- b. 目的のリリース (<https://github.com/suse-edge/fleet-examples/releases>) タグから [suse-edge/fleet-examples](#) (<https://github.com/suse-edge/fleet-examples.git>) リポジトリのクローンを作成し、 **【Create from YAML (YAMLから作成)】** ページの **【Read from File (ファイルから読み取り)】** オプションを選択する。そこから、必要なバンドルに移動します(RKE2の場合は `bundles/day2/system-upgrade-controller-plans/rke2-upgrade/plan-bundle.yaml`、K3sの場合は `bundles/day2/system-upgrade-controller-plans/k3s-upgrade/plan-bundle.yaml`)。 **【Create from YAML (YAMLから作成)】** ページにバンドルコンテンツが自動的に入力されます。

5. Rancher UIでバンドルを編集します。

- バンドルの**ネームスペース**を `fleet-local` ネームスペースを指すように変更します。

```
# Example
kind: Bundle
apiVersion: fleet.cattle.io/v1alpha1
metadata:
  name: rke2-upgrade
  namespace: fleet-local
...
```

- バンドルの**ターゲット**クラスタを ローカル(管理)クラスタを指すように変更します。

```
spec:
  targets:
    - clusterName: local
```



注記

一部のユースケースでは、ローカルクラスタに異なる名前を付けることができます。

ローカル クラスタ名を取得するには、次のコマンドを実行します。

```
kubectll get clusters.fleet.cattle.io -n fleet-local
```

6. [Create (作成)] を選択します。

35.2.5.4.2.2 バンドルの作成 - 手動

1. バンドルリソースをプルします。

- **RKE2** クラスタの場合:

```
curl -o rke2-plan-bundle.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.4.0/bundles/day2/system-upgrade-controller-plans/rke2-upgrade/plan-bundle.yaml
```

- **K3s** クラスタの場合:

```
curl -o k3s-plan-bundle.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.4.0/bundles/day2/system-upgrade-controller-plans/k3s-upgrade/plan-bundle.yaml
```

2. バンドル設定を編集します。

- バンドル の **ターゲット** クラスタを ローカル (管理) クラスタを指すように変更します。

```
spec:
  targets:
    - clusterName: local
```



注記

一部のユースケースでは、ローカル クラスタに異なる名前を付けることができます。

ローカル クラスタ名を取得するには、次のコマンドを実行します。

```
kubectll get clusters.fleet.cattle.io -n fleet-local
```

- バンドル の **ネームスペース** を fleet-local ネームスペースを指すように変更します。


```
# Example
kind: Bundle
apiVersion: fleet.cattle.io/v1alpha1
metadata:
  name: rke2-upgrade
  namespace: fleet-local
...
```

3. バンドルリソースを管理クラスタに適用します。

```
# For RKE2
kubectl apply -f rke2-plan-bundle.yaml

# For K3s
kubectl apply -f k3s-plan-bundle.yaml
```

4. 作成したバンドルリソースをfleet-local名前空間の下に表示します。

```
# For RKE2
kubectl get bundles rke2-upgrade -n fleet-local

# For K3s
kubectl get bundles k3s-upgrade -n fleet-local

# Example output
NAME                BUNDLEDEPLOYMENTS-READY  STATUS
k3s-upgrade         0/0
rke2-upgrade        0/0
```

35.2.5.4.3 SUC Planのデプロイメント - サードパーティのGitOpsワークフロー

ユーザがKubernetesアップグレードSUC Planを独自のサードパーティGitOpsワークフロー(例: Flux)に組み込むユースケースがある場合があります。

必要なK8sアップグレードリソースを取得するには、まず使用するsuse-edge/fleet-examples (<https://github.com/suse-edge/fleet-examples.git>) リポジトリのEdge リリース (<https://github.com/suse-edge/fleet-examples/releases>) タグを決定します。

その後、次の場所でリソースを確認できます。

- RKE2クラスタのアップグレードの場合:
 - [control-plane](#)ノード用 - [fleets/day2/system-upgrade-controller-plans/rke2-upgrade/plan-control-plane.yaml](#)
 - ワーカーノードの場合 - [fleets/day2/system-upgrade-controller-plans/rke2-upgrade/plan-worker.yaml](#)
- K3sクラスタのアップグレードの場合:
 - [control-plane](#)ノード用 - [fleets/day2/system-upgrade-controller-plans/k3s-upgrade/plan-control-plane.yaml](#)
 - ワーカーノードの場合 - [fleets/day2/system-upgrade-controller-plans/k3s-upgrade/plan-worker.yaml](#)

！ 重要

これらのPlanリソースは、[System Upgrade Controller](#)によって解釈され、アップグレードする各ダウンストリームクラスタにデプロイする必要があります。SUCデプロイメントの情報については、[21.2項「System Upgrade Controllerのインストール」](#)を参照してください。

GitOpsワークフローをKubernetesバージョンアップグレードの**SUC Plan**をデプロイするために使用する方法をよりよく理解するために、[Fleet](#)を使用する更新手順の概要([35.2.5.2項「概要」](#))を確認すると役立つ場合があります。

35.2.6 Helmチャートのアップグレード

このセクションでは、次の内容について説明します。

1. [35.2.6.1項「エアギャップ環境の準備」](#) - Edgeに関連するOCIチャートとイメージをプライベートレジストリに配布する方法に関する情報を保持します。
2. [35.2.6.2項「アップグレード手順」](#) - 異なるHelmチャートアップグレードのユースケースとそのアップグレード手順に関する情報を保持します。

35.2.6.1 エアギャップ環境の準備

35.2.6.1.1 HelmチャートFleetにアクセスできることの確認

環境でサポートする内容によって、次のオプションのいずれかを選択できます。

1. 管理クラスタからアクセス可能なローカルGitサーバでチャートのFleetリソースをホストします。
2. FleetのCLIを使用して、直接使用可能でどこかにホストする必要のないバンドルにHelmチャートを変換 (<https://fleet.rancher.io/bundle-add#convert-a-helm-chart-into-a-bundle>) します。FleetのCLIは、リリース (<https://github.com/rancher/fleet/releases/tag/v0.13.1>) ページから取得できます。Macユーザの場合は、fleet-cli (<https://formulae.brew.sh/formula/fleet-cli>) Homebrew Formulaeがあります。

35.2.6.1.2 Edgeリリースバージョンに必要なアセットの検索

1. 「Day 2」 リリース (<https://github.com/suse-edge/fleet-examples/releases>) のページに移動し、チャートのアップグレード先のEdgeリリースを見つけ、**[Assets (アセット)]** をクリックします。
2. **[Assets (アセット)]** セクションから、次のファイルをダウンロードします。

リリースファイル	説明
edge-save-images.sh	edge-release-images.txtファイルで指定されたイメージを取得し、それらを「.tar.gz」アーカイブにパッケージ化します。
edge-save-oci-artefacts.sh	特定のEdgeリリースに関連するOCIチャートイメージを取得し、それらを「.tar.gz」アーカイブにパッケージ化します。
edge-load-images.sh	「.tar.gz」アーカイブからイメージをロードし、再タグ付けして、プライベートレジストリにプッシュします。

edge-load-oci-artefacts.sh	Edge OCI 「.tgz」 チャートパッケージを含むディレクトリを取得し、プライベートレジストリにロードします。
edge-release-helm-oci-artefacts.txt	特定のEdgeリリースに関連するOCI チャートイメージのリストを含みます。
edge-release-images.txt	特定のEdgeリリースに関連するイメージのリストを含みます。

35.2.6.1.3 Edgeリリースイメージアーカイブの作成

インターネットにアクセスできるマシンで次の手順を実行します。

1. edge-save-images.shを実行可能にします。

```
chmod +x edge-save-images.sh
```

2. イメージアーカイブを生成します。

```
./edge-save-images.sh --source-registry registry.suse.com
```

3. これにより、edge-images.tar.gzという名前のすぐにロードできるアーカイブが作成されます。



注記

-i|--imagesオプションが指定される場合、アーカイブの名前は異なる場合があります。

4. このアーカイブを**エアギャップ**マシンにコピーします。

```
scp edge-images.tar.gz <user>@<machine_ip>:/path
```

35.2.6.1.4 Edge OCIチャートイメージアーカイブの作成

インターネットにアクセスできるマシンで次の手順を実行します。

1. edge-save-oci-artefacts.shを実行可能にします。

```
chmod +x edge-save-oci-artefacts.sh
```

2. OCIチャートイメージアーカイブを生成します。

```
./edge-save-oci-artefacts.sh --source-registry registry.suse.com
```

3. これにより、oci-artefacts.tar.gzという名前のアーカイブが作成されます。



注記

-a|--archiveオプションが指定される場合、アーカイブの名前は異なる場合があります。

4. このアーカイブを**エアギャップマシン**にコピーします。

```
scp oci-artefacts.tar.gz <user>@<machine_ip>:/path
```

35.2.6.1.5 Edgeリリースイメージをエアギャップマシンにロード

エアギャップマシンで次の手順を実行します。

1. プライベートレジストリにログインします(必要な場合)。

```
podman login <REGISTRY.YOURDOMAIN.COM:PORT>
```

2. edge-load-images.shを実行可能にします。

```
chmod +x edge-load-images.sh
```

3. 以前に**コピーした** edge-images.tar.gzアーカイブを渡して、スクリプトを実行します。

```
./edge-load-images.sh --source-registry registry.suse.com --registry  
<REGISTRY.YOURDOMAIN.COM:PORT> --images edge-images.tar.gz
```



注記

これにより、edge-images.tar.gzからすべてのイメージがロードされ、再タグ付けされて、それらを--registryオプションで指定されているレジストリにプッシュします。

35.2.6.1.6 Edge OCIチャートイメージのエアギャップマシンへのロード

エアギャップマシンで次の手順を実行します。

1. プライベートレジストリにログインします(必要な場合)。

```
podman login <REGISTRY.YOURDOMAIN.COM:PORT>
```

2. `edge-load-oci-artefacts.sh`を実行可能にします。

```
chmod +x edge-load-oci-artefacts.sh
```

3. コピーした`oci-artefacts.tar.gz`アーカイブをuntarします。

```
tar -xvf oci-artefacts.tar.gz
```

4. 命名テンプレート`edge-release-oci-tgz-<date>`を含むディレクトリが生成されます。
5. このディレクトリを`edge-load-oci-artefacts.sh`スクリプトに渡し、Edge OCIチャートイメージをプライベートレジストリにロードします。



注記

このスクリプトは、`Helm` CLIが環境にプリインストールされていることを想定しています。`Helm`のインストール手順については、「[Installing Helm \(Helmのインストール\)](https://helm.sh/docs/intro/install/) (<https://helm.sh/docs/intro/install/>)」を参照してください。

```
./edge-load-oci-artefacts.sh --archive-directory edge-release-oci-tgz-<date> --  
registry <REGISTRY.YOURDOMAIN.COM:PORT> --source-registry registry.suse.com
```

35.2.6.1.7 Kubernetesディストリビューションでプライベートレジストリを設定する

RKE2の場合は、「[Private Registry Configuration \(プライベートレジストリの設定\)](https://docs.rke2.io/install/private_registry) (https://docs.rke2.io/install/private_registry)」を参照してください。

K3sの場合は、「[Private Registry Configuration \(プライベートレジストリの設定\)](https://docs.k3s.io/installation/private-registry) (<https://docs.k3s.io/installation/private-registry>)」を参照してください。

35.2.6.2 アップグレード手順

このセクションでは、`Helm`アップグレード手順の次のユースケースを中心に説明します。

1. 35.2.6.2.1項 「新しいクラスタがあり、Edge Helmチャートをデプロイして管理したい」
2. 35.2.6.2.2項 「Fleetで管理されているHelmチャートをアップグレードしたい場合」
3. 35.2.6.2.3項 「EIBを介してデプロイされたHelmチャートをアップグレードしたい」

！ 重要

手動でデプロイしたHelmチャートを確実にアップグレードすることはできません。35.2.6.2.1項 「新しいクラスタがあり、Edge Helmチャートをデプロイして管理したい」で説明する方法を使用してHelmチャートを再デプロイすることをお勧めします。

35.2.6.2.1 新しいクラスタがあり、Edge Helmチャートをデプロイして管理したい

このセクションでは、以下の実行方法について説明します。

1. 35.2.6.2.1.1項 「チャートのFleetリソースの準備」.
2. 35.2.6.2.1.2項 「チャートのFleetのデプロイ」.
3. 35.2.6.2.1.3項 「デプロイされたHelmチャートの管理」.

35.2.6.2.1.1 チャートのFleetリソースの準備

1. チャートのFleetリソースを、使用するEdgeリリース (<https://github.com/suse-edge/fleet-examples/releases>)  タグから取得します。
2. HelmチャートのFleet (`fleets/day2/chart-templates/<chart>`)に移動します。
3. **GitOpsワークフローを使用する場合は**、チャートFleetディレクトリをGitOpsを実行するGitリポジトリにコピーします。
4. (任意) Helmチャートで値を設定する必要がある場合は、コピーしたディレクトリの`fleet.yaml`ファイルに含まれる設定`.helm.values`を編集します。
5. (任意) 環境に合わせて、チャートのFleetにリソースを追加しなければならないユースケースがあります。Fleetディレクトリを拡張する方法については、「[Git Repository Contents \(Gitリポジトリのコンテンツ\)](https://fleet.rancher.io/gitrepo-content) (<https://fleet.rancher.io/gitrepo-content>) 」を参照してください。



注記

場合によっては、Fleet がHelm操作に使用するデフォルトのタイムアウトが不十分で、次のエラーが発生する可能性があります。

```
failed pre-install: context deadline exceeded
```

このような場合は、`fleet.yaml`ファイルの`helm`設定の下に、`timeoutSeconds` (<https://fleet.rancher.io/ref-crds#helmoptions>)  プロパティを追加してください。

longhorn Helmチャートの例は次のようになります。

- ユーザGitリポジトリ構造:

```
<user_repository_root>
├─ longhorn
│   └─ fleet.yaml
└─ longhorn-crd
    └─ fleet.yaml
```

- ユーザのLonghornデータが入力された`fleet.yaml`の内容:

```
defaultNamespace: longhorn-system

helm:
  # timeoutSeconds: 10
  releaseName: "longhorn"
  chart: "longhorn"
  repo: "https://charts.rancher.io/"
  version: "107.0.0+up1.9.1"
  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"}
```



注記

これらは値の例であり、longhornチャートのカスタム設定を示すために使用しているだけです。longhornチャートのデプロイメントのガイドラインとみなさないでください。

35.2.6.2.1.2 チャートのFleetのデプロイ

チャートのFleetをデプロイするには、GitRepo (35.2.6.2.1.2.1項「GitRepo」) またはバンドル (35.2.6.2.1.2.2項「バンドル」) のいずれかを使用できます。



注記

Fleetをデプロイする場合に、Modifiedメッセージを取得した場合、Fleetのdiffセクションに対応するcomparePatchesエントリを追加してください。詳細については、「[Generating Diffs to Ignore Modified GitRepos \(変更されたGitReposを無視する差分を生成する\)](https://fleet.rancher.io/bundle-diffs) (<https://fleet.rancher.io/bundle-diffs>)」を参照してください。

35.2.6.2.1.2.1 GitRepo

FleetのGitRepo (<https://fleet.rancher.io/ref-gitrepo>) リソースは、チャートのFleetリソースへのアクセス方法、それらのリソースを適用するのに必要なクラスタに関する情報を保持しています。

GitRepoリソースは、[Rancher UI \(https://ranchermanager.docs.rancher.com/v2.12/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui\)](https://ranchermanager.docs.rancher.com/v2.12/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui) を通じて、または手動で管理クラスタにリソースをデプロイ (<https://fleet.rancher.io/tut-deployment>) することで、デプロイできます。

手動デプロイメントのLonghorn GitRepoリソースの例:

```
apiVersion: fleet.cattle.io/v1alpha1
kind: GitRepo
metadata:
  name: longhorn-git-repo
  namespace: fleet-local
spec:
  # If using a tag
  # revision: user_repository_tag
  #
  # If using a branch
  # branch: user_repository_branch
  paths:
    # As seen in the 'Prepare your Fleet resources' example
    - longhorn
    - longhorn-crd
  repo: user_repository_url
```

35.2.6.2.1.2.2 バンドル

バンドル (<https://fleet.rancher.io/bundle-add>) リソースは、Fleetによってデプロイされる必要がある生のKubernetesリソースを保持しています。通常、GitRepoアプローチを使用することを推奨されますが、環境がエアギャップされ、ローカルGitサーバをサポートできないユースケース用に、**バンドル**がHelmチャートFleetをターゲットクラスタに伝播するのに役立ちます。

バンドルは、Rancher UI ([\[Continuous Delivery \(継続的デリバリ\)\]](#) → [\[Advanced \(詳細\)\]](#)) → [\[Bundles \(バンドル\)\]](#) → [\[Create from YAML \(YAMLから作成\)\]](#)) を介して、またはバンドルリソースを正しいFleet名前スペースに手動でデプロイして、デプロイできます。Fleet名前スペースについては、アップストリームドキュメント (<https://fleet.rancher.io/namespaces#gitrepos-bundles-clusters-clustergroups>) を参照してください。

Edge Helmチャートのバンドルは、Fleetの「[Convert a Helm Chart into a Bundle \(Helmチャートをバンドルに変換する\)](#) (<https://fleet.rancher.io/bundle-add#convert-a-helm-chart-into-a-bundle>)」アプローチを利用して作成できます。

以下に、バンドルリソースをlonghorn (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/fleets/day2/chart-templates/longhorn/longhorn/fleet.yaml>) と longhorn-crd (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/fleets/day2/chart-templates/longhorn/longhorn-crd/fleet.yaml>) HelmチャートFleetテンプレートから作成し、このバンドルを管理クラスタに手動でデプロイする方法の例を示します。



注記

ワークフローを説明するため、以下の例では[suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples) (<https://github.com/suse-edge/fleet-examples>) ディレクトリ構造を使用しています。

1. longhorn (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/fleets/day2/chart-templates/longhorn/longhorn/fleet.yaml>) チャートFleetテンプレートに移動します。

```
cd fleets/day2/chart-templates/longhorn/longhorn
```

2. FleetにHelmチャートのデプロイ先クラスタを指示する `targets.yaml` ファイルを作成します。

```
cat > targets.yaml <<EOF
targets:
# Match your local (management) cluster
- clusterName: local
EOF
```



注記

ローカルクラスタの名前が異なる場合があるユースケースがいくつかあります。
ローカルクラスタ名を取得するには、次のコマンドを実行します。

```
kubectl get clusters.fleet.cattle.io -n fleet-local
```

3. `fleet-cli` (<https://fleet.rancher.io/cli/fleet-cli/fleet>) を使用して、Longhorn HelmチャートFleetをバンドルリソースに変換します。



注記

FleetのCLIは、リリース (<https://github.com/rancher/fleet/releases/tag/v0.13.1>) アセットページから取得できます(`fleet-linux-amd64`)。

Macユーザの場合、[fleet-cli \(https://formulae.brew.sh/formula/fleet-cli\)](https://formulae.brew.sh/formula/fleet-cli)  Homebrew Formulaeがあります。

```
fleet apply --compress --targets-file=targets.yaml -n fleet-local -o - longhorn-bundle > longhorn-bundle.yaml
```

4. [longhorn-crd \(https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/fleets/day2/chart-templates/longhorn/longhorn-crd/fleet.yaml\)](https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/fleets/day2/chart-templates/longhorn/longhorn-crd/fleet.yaml)  チャートFleetテンプレートに移動します。

```
cd fleets/day2/chart-templates/longhorn/longhorn-crd
```

5. FleetにHelmチャートのデプロイ先クラスタを指示する targets.yaml ファイルを作成します。

```
cat > targets.yaml <<EOF
targets:
# Match your local (management) cluster
- clusterName: local
EOF
```

6. [fleet-cli \(https://fleet.rancher.io/cli/fleet-cli/fleet\)](https://fleet.rancher.io/cli/fleet-cli/fleet)  を使用して、Longhorn CRD HelmチャートFleetをバンドルリソースに変換します。

```
fleet apply --compress --targets-file=targets.yaml -n fleet-local -o - longhorn-crd-bundle > longhorn-crd-bundle.yaml
```

7. longhorn-bundle.yaml と longhorn-crd-bundle.yaml ファイルを 管理クラスタ にデプロイします。

```
kubectl apply -f longhorn-crd-bundle.yaml
kubectl apply -f longhorn-bundle.yaml
```

これらの手順に従うと、SUSE Storageが指定された管理クラスタのすべてにデプロイされます。

35.2.6.2.1.3 デプロイされたHelmチャートの管理

Fleetでデプロイした後のHelmチャートのアップグレードについては、[35.2.6.2.2項「Fleetで管理されているHelmチャートをアップグレードしたい場合」](#)を参照してください。

35.2.6.2.2 Fleetで管理されているHelmチャートをアップグレードしたい場合

1. 目的のEdgeリリースと互換性を持つように、チャートをアップグレードする必要があるバージョンを決定します。EdgeリリースごとのHelmチャートバージョンはリリースノート(52.1項「要約」)から表示できます。
2. Fleetで監視されているGitリポジトリで、Helmチャートのfleet.yamlファイルを、リリースノート(52.1項「要約」)から取得した正しいチャートのバージョンとリポジトリで編集します。
3. 変更をコミットしてリポジトリにプッシュした後で、目的のHelmチャートのアップグレードがトリガされます。

35.2.6.2.3 EIBを介してデプロイされたHelmチャートをアップグレードしたい

第11章「Edge Image Builder」は、HelmChartリソースを作成し、RKE2 (<https://docs.rke2.io/helm>) / K3s (<https://docs.k3s.io/helm>) Helm統合機能によって導入されたhelm-controllerを利用して、Helmチャートをデプロイします。

EIBを介してデプロイされたHelmチャートが正常にアップグレードされるようにするには、ユーザは各HelmChartリソースに対してアップグレードを実行する必要があります。

以下に関する情報が提供されています。

- アップグレードプロセスの一般的な概要(35.2.6.2.3.1項「概要」)。
- 必要なアップグレード手順(35.2.6.2.3.2項「アップグレード手順」)。
- 説明した方法を使用したLonghorn (<https://longhorn.io>) チャートアップグレードを示す例(35.2.6.2.3.3項「例」)。
- 異なるGitOpsツールを使用したアップグレードプロセスを使用する方法(35.2.6.2.3.4項「サードパーティのGitOpsツールを使用したHelmチャートのアップグレード」)。

35.2.6.2.3.1 概要

EIBを介してデプロイされたHelmチャートは、eib-charts-upgrader (<https://github.com/suse-edge/fleet-examples/tree/release-3.4.0/fleets/day2/eib-charts-upgrader>) と呼ばれるフリートを通じてアップグレードされます。

このフリートは、ユーザが提供したデータを処理し、特定のHelmChartリソースセットを更新します。

これらのリソースを更新すると、[helm-controller](https://github.com/k3s-io/helm-controller) (<https://github.com/k3s-io/helm-controller>) がトリガされ、変更されたHelmChartリソースに関連付けられているHelmチャートが**アップグレード**されます。

ユーザは以下を行うことのみ求められます。

1. アップグレードが必要な各Helmチャートのアーカイブをローカルでプル (https://helm.sh/docs/helm/helm_pull/) します。
2. これらのアーカイブを[generate-chart-upgrade-data.sh](https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/scripts/day2/generate-chart-upgrade-data.sh) (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/scripts/day2/generate-chart-upgrade-data.sh>) generate-chart-upgrade-data.shスクリプトに渡します。これにより、これらのアーカイブのデータが [eib-charts-upgrader](#) フリートに含められます。
3. [eib-charts-upgrader](#) Fleetを管理クラスタにデプロイします。これは[GitRepo](#)リソースまたは[バンドル](#)リソースのいずれか通じて実行されます。

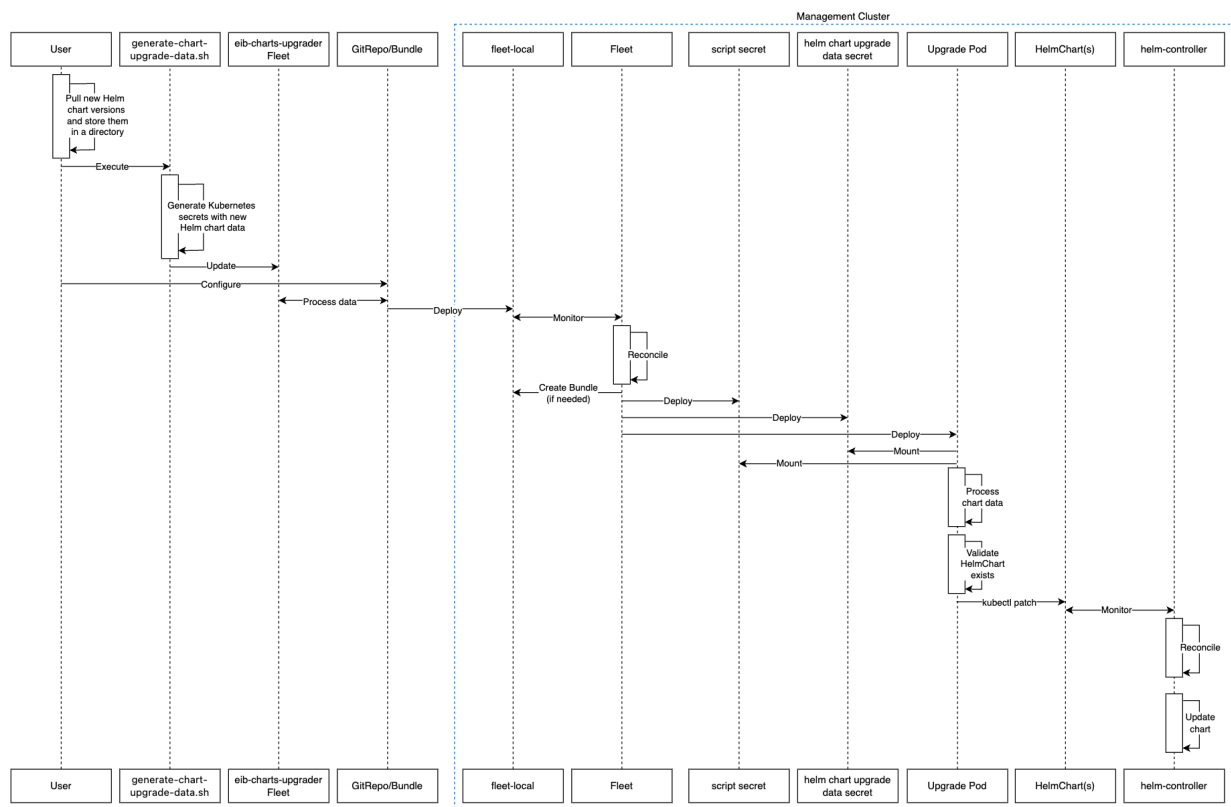
デプロイされたら、[eib-charts-upgrader](#)がFleetのサポートによって、そのリソースを目的の管理クラスタに配布します。

これらのリソースには、次のものが含まれます。

1. ユーザが提供するHelmチャートデータを保持する[Secret](#)のセット。
2. 前述の[Secret](#)をマウントし、それらに基づいて対応するHelmChartリソースに[パッチを適用する](#) (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_patch/) Podをデプロイする[Kubernetes Job](#)。

前述のとおり、これにより、[helm-controller](#)がトリガされ、実際のHelmチャートアップグレードが実行されます。

上記の説明を以下に図示します。



35.2.6.2.3.2 アップグレード手順

1. 正しいリリース タグ (<https://github.com/suse-edge/fleet-examples/releases/tag/release-3.4.0>) から `suse-edge/fleet-examples` リポジトリのクローンを作成します。
2. 取得したHelmチャートアーカイブを保存するディレクトリを作成します。

```
mkdir archives
```

3. 新規に作成されたアーカイブディレクトリ内で、アップグレードするHelmチャートのアーカイブをプル (https://helm.sh/docs/helm/helm_pull/) します。

```
cd archives
helm pull [chart URL | repo/chartname]

# Alternatively if you want to pull a specific version:
# helm pull [chart URL | repo/chartname] --version 0.0.0
```

4. 目的のリリースタグ (<https://github.com/suse-edge/fleet-examples/releases/tag/release-3.4.0>) のアセットから、`generate-chart-upgrade-data.sh` スクリプトをダウンロードします。
5. `generate-chart-upgrade-data.sh` スクリプトを実行します。

```
chmod +x ./generate-chart-upgrade-data.sh

./generate-chart-upgrade-data.sh --archive-dir /foo/bar/archives/ --fleet-path /foo/bar/fleet-examples/fleets/day2/eib-charts-upgrader
```

`--archive-dir` ディレクトリ内のチャートアーカイブごとに、スクリプトにより、チャートアップグレードデータを含む `Kubernetes Secret YAML` ファイルが生成され、`--fleet-path` によって指定されたフリートの `base/secrets` ディレクトリに保存されます。

`generate-chart-upgrade-data.sh` スクリプトは、フリートに追加の変更も適用し、生成された `Kubernetes Secret YAML` ファイルが、フリートによってデプロイされたワークロードによって正しく利用されるようにします。

❗ 重要

ユーザは、`generate-chart-upgrade-data.sh` スクリプトが生成する内容に変更を加えてはなりません。

以下の手順は、実行している環境によって異なります。

1. GitOpsをサポートする環境の場合(例: エアギャップされていない、エアギャップされたがローカルGitサーバサポートが可能):
 - a. `fleets/day2/eib-charts-upgrader` Fleetを、GitOpsに使用するリポジトリにコピーします。

📌 注記

Fleetに`generate-chart-upgrade-data.sh` スクリプトによって加えられた変更が含まれていることを確認してください。

- b. `eib-charts-upgrader` Fleetのすべてのリソースを配布するために使用される `GitRepo` リソースを設定します。

- i. Rancher UIを通じたGitRepoの設定およびデプロイメントについては、「[Accessing Fleet in the Rancher UI \(Rancher UIでのFleetへのアクセス\)](https://ranchermanager.docs.rancher.com/v2.12/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui) (<https://ranchermanager.docs.rancher.com/v2.12/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui>)」を参照してください。
 - ii. GitRepo手動設定およびデプロイメントの場合、「[Creating a Deployment \(デプロイメントの作成\)](https://fleet.rancher.io/tut-deployment) (<https://fleet.rancher.io/tut-deployment>)」を参照してください。
2. GitOpsをサポートしていない環境の場合 (例: エアギャップされ、ローカルGitサーバの使用が許可されていない):

- a. [rancher/fleet](https://github.com/rancher/fleet/releases/tag/v0.13.1) リリース (<https://github.com/rancher/fleet/releases/tag/v0.13.1>) ページから `fleet-cli` バイナリをダウンロードします (Linuxの場合は `fleet-linux-amd64`)。Macユーザの場合、使用可能なHomebrew Formulaeがあります (`fleet-cli` (<https://formulae.brew.sh/formula/fleet-cli>))。

- b. `eib-charts-upgrader` Fleetに移動します。

```
cd /foo/bar/fleet-examples/fleets/day2/eib-charts-upgrader
```

- c. リソースをデプロイする場所をFleetに指示する `targets.yaml` ファイルを作成します。

```
cat > targets.yaml <<EOF
targets:
# To map the local(management) cluster
- clusterName: local
EOF
```



注記

一部のユースケースでは、ローカル クラスタに異なる名前を付けることができます。

ローカル クラスタ名を取得するには、次のコマンドを実行します。

```
kubectl get clusters.fleet.cattle.io -n fleet-local
```

- d. `fleet-cli` を使用して、Fleetを バンドル リソースに変換します。

```
fleet apply --compress --targets-file=targets.yaml -n fleet-local -o - eib-  
charts-upgrade > bundle.yaml
```

これにより、eib-charts-upgrader Fleetからのすべてのテンプレート化されたリソースを保持するバンドル(bundle.yaml)が作成されます。

fleet applyコマンドに関する詳細については、「[fleet apply \(https://fleet.rancher.io/cli/fleet-cli/fleet_apply\)](https://fleet.rancher.io/cli/fleet-cli/fleet_apply)」を参照してください。

Fleetをバンドルに変換する方法に関する詳細については、「[Convert a Helm Chart into a Bundle \(Helmチャートのバンドルへの変換\) \(https://fleet.rancher.io/bundle-add#convert-a-helm-chart-into-a-bundle\)](https://fleet.rancher.io/bundle-add#convert-a-helm-chart-into-a-bundle)」を参照してください。

e. バンドルをデプロイします。これは次の2つの方法のいずれかで実行できます。

i. RancherのUIを通じて - **[Continuous Delivery (継続的デリバリー)] → [Advanced (詳細)] → [Bundles (バンドル)] → [Create from YAML (YAMLから作成)]** に移動して、bundle.yaml コンテンツを解析するか、**[Read from File (ファイルから読み取り)]** オプションをクリックして、ファイル自体を渡します。

ii. 手動 - 管理クラスタ内にbundle.yamlファイルを手動でデプロイします。

これらの手順を実行すると、正常にGitRepo/バンドルリソースがデプロイされます。リソースはFleetによって取得され、そのコンテンツはユーザが以前の手順で指定したターゲットクラスタにデプロイされます。プロセスの概要については、[35.2.6.2.3.1項「概要」](#)を参照してください。

アップグレードプロセスの追跡方法については、[35.2.6.2.3.3項「例」](#)を参照してください。

！ 重要

チャートアップグレードが正常に確認されたら、バンドル/GitRepoリソースを削除します。

これにより、管理クラスタから不要になったアップグレードリソースが削除され、今後バージョンクラッシュが発生しないようになります。



注記

以下の例は、管理クラスタ上で、EIBを介してデプロイされたHelmチャートを、あるバージョンから別のバージョンにアップグレードする方法を示しています。この例で使われているバージョンは、推奨バージョン**ではない**ことに注意してください。Edgeリリースに固有のバージョン推奨事項については、リリースノート(52.1項「要約」)を参照してください。

ユースケース:

- 管理クラスタで、古いバージョンのLonghorn (<https://longhorn.io>) が実行されています。
- クラスタは、次のイメージ定義スニペットを使用して、EIBを通じてデプロイされました。

```
kubernetes:
  helm:
    charts:
      - name: longhorn-crd
        repositoryName: rancher-charts
        targetNamespace: longhorn-system
        createNamespace: true
        version: 104.2.0+up1.7.1
        installationNamespace: kube-system
      - name: longhorn
        repositoryName: rancher-charts
        targetNamespace: longhorn-system
        createNamespace: true
        version: 104.2.0+up1.7.1
        installationNamespace: kube-system
    repositories:
      - name: rancher-charts
        url: https://charts.rancher.io/
  ...
```

- SUSE Storageは、Edge 3.4リリースと互換性のあるバージョンにアップグレードする必要があります。つまり、107.0.0+up1.9.1にアップグレードする必要があります。
- 管理クラスタがローカルGitサーバのサポートなしで**エアギャップ**されており、Rancherセットアップが動作していることを前提としています。

アップグレード手順(35.2.6.2.3.2項「アップグレード手順」)に従います。

1. release-3.4.0タグからsuse-edge/fleet-exampleリポジトリのクローンを作成します。

```
git clone -b release-3.4.0 https://github.com/suse-edge/fleet-examples.git
```

2. Longhornアップグレードアーカイブが保存されるディレクトリを作成します。

```
mkdir archives
```

3. 目的のLonghornチャートアーカイブバージョンを取得します。

```
# First add the Rancher Helm chart repository
helm repo add rancher-charts https://charts.rancher.io/

# Pull the Longhorn 1.9.1 CRD archive
helm pull rancher-charts/longhorn-crd --version 107.0.0+up1.9.1

# Pull the Longhorn 1.9.1 chart archive
helm pull rancher-charts/longhorn --version 107.0.0+up1.9.1
```

4. archivesディレクトリ以外で、suse-edge/fleet-examplesリリースタグ (<https://github.com/suse-edge/fleet-examples/releases/tag/release-3.4.0>) から generate-chart-upgrade-data.sh スクリプトをダウンロードします。

5. ディレクトリセットアップは次のようになるはずです。

```
.
├─ archives
│   ├── longhorn-107.0.0+up1.9.1.tgz
│   └── longhorn-crd-107.0.0+up1.9.1.tgz
├─ fleet-examples
└─ ...
    ├── fleets
    │   └── day2
    │       ├── ...
    │       └── eib-charts-upgrader
    │           ├── base
    │           │   ├── job.yaml
    │           │   ├── kustomization.yaml
    │           │   ├── patches
    │           │   │   └── job-patch.yaml
    │           │   └── rbac
    │           │       ├── cluster-role-binding.yaml
    │           │       ├── cluster-role.yaml
    │           │       ├── kustomization.yaml
    │           │       ├── sa.yaml
    │           └── secrets
    └── eib-charts-upgrader-script.yaml
```

```

| | | | | └─ kustomization.yaml
| | | | └─ fleet.yaml
| | | └─ kustomization.yaml
| | └─ ...
| └─ ...
└─ generate-chart-upgrade-data.sh

```

6. `generate-chart-upgrade-data.sh` スクリプトを実行します。

```

# First make the script executable
chmod +x ./generate-chart-upgrade-data.sh

# Then execute the script
./generate-chart-upgrade-data.sh --archive-dir ./archives --fleet-path ./fleet-
examples/fleets/day2/eib-charts-upgrader

```

スクリプト実行後のディレクトリ構造は次のようになるはずです。

```

.
├─ archives
│   ├── longhorn-107.0.0+up1.9.1.tgz
│   └─ longhorn-crd-107.0.0+up1.9.1.tgz
├─ fleet-examples
├─ ...
├─ fleets
│   └─ day2
│       ├── ...
│       └─ eib-charts-upgrader
│           ├── base
│           │   ├── job.yaml
│           │   ├── kustomization.yaml
│           │   ├── patches
│           │   │   └─ job-patch.yaml
│           │   └─ rbac
│           │       ├── cluster-role-binding.yaml
│           │       ├── cluster-role.yaml
│           │       ├── kustomization.yaml
│           │       └─ sa.yaml
│           └─ secrets
│               ├── eib-charts-upgrader-script.yaml
│               ├── kustomization.yaml
│               └─ longhorn-VERSION.yaml - secret created by the generate-
chart-upgrade-data.sh script
├─ longhorn-crd-VERSION.yaml - secret created by the
generate-chart-upgrade-data.sh script
├─ fleet.yaml
├─ kustomization.yaml
├─ ...
└─ generate-chart-upgrade-data.sh

```

Gitで変更されたファイルは次のようになるはずです。

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   fleets/day2/eib-charts-upgrader/base/patches/job-patch.yaml
    modified:   fleets/day2/eib-charts-upgrader/base/secrets/kustomization.yaml

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    fleets/day2/eib-charts-upgrader/base/secrets/longhorn-VERSION.yaml
    fleets/day2/eib-charts-upgrader/base/secrets/longhorn-crd-VERSION.yaml
```

7. eib-charts-upgrader Fleetのバンドルを作成します。

a. まず、Fleet自体に移動します。

```
cd ./fleet-examples/fleets/day2/eib-charts-upgrader
```

b. 次に、targets.yamlファイルを作成します。

```
cat > targets.yaml <<EOF
targets:
- clusterName: local
EOF
```

c. 次に、fleet-cliバイナリを使用して、Fleetをバンドルに変換します。

```
fleet apply --compress --targets-file=targets.yaml -n fleet-local -o - eib-
charts-upgrade > bundle.yaml
```

8. Rancher UIを通じてバンドルをデプロイします。

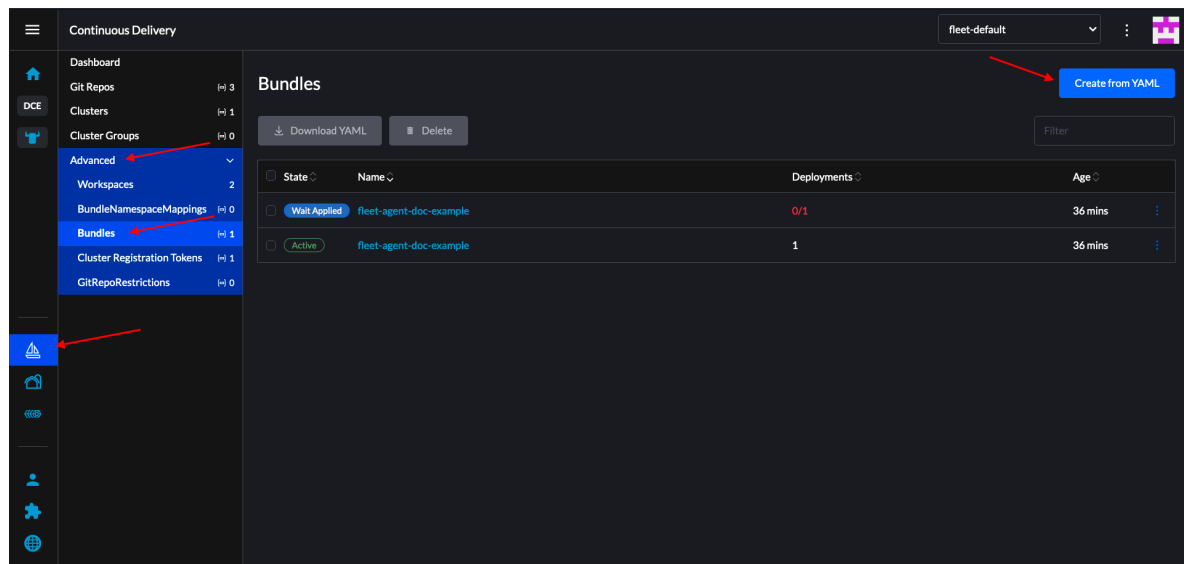


図 35.1: RANCHER UIを通じたバンドルのデプロイ

ここから、**Read from File (ファイルから読み取り)** を選択し、システムで `bundle.yaml` ファイルを見つけます。

これにより、Rancher UI内に**バンドル**が自動入力されます。

Create (作成) を選択します。

9. 正常にデプロイされたら、バンドルは次のようになります。

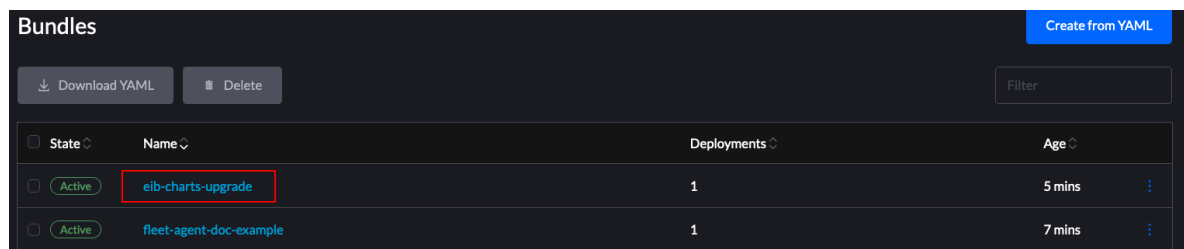
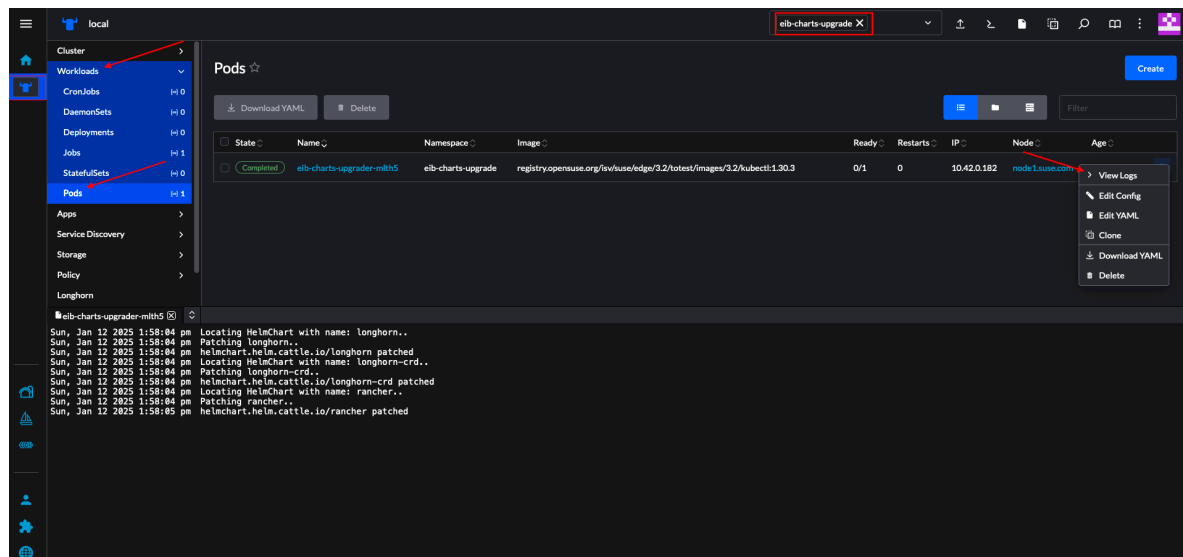


図 35.2: 正常にデプロイされたバンドル

バンドルのデプロイメントが成功した後で、アップグレードプロセスを監視するには次のようになります。

1. アップグレードPodのログを確認します。



2. ここで、helm-controllerによってアップグレードに作成されたPodのログを確認します。

- Pod名は次のテンプレートを使用します - helm-install-longhorn-<random-suffix>
- Podは、HelmChartリソースがデプロイされた名前空間にあります。この場合、この名前空間はkube-systemです。

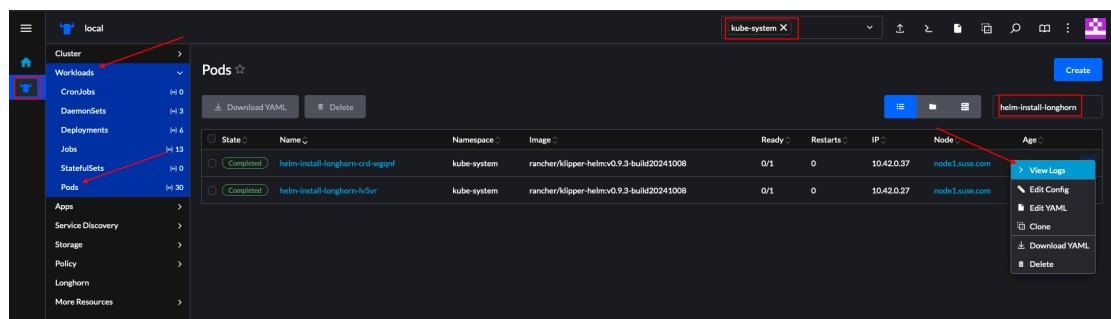


図 35.3: 正常にアップグレードされたLONGHORNチャートのログ

- RancherのHelmChartsセクション([More Resources (その他のリソース)] → [HelmCharts])に移動して、HelmChartのバージョンが更新されていることを確認します。チャートがデプロイされた名前空間を選択します。この例では、kube-systemです。
- 最後に、Longhorn Podが実行されていることを確認します。

上記の検証後、Longhorn Helmチャートが107.0.0+up1.9.1バージョンにアップグレードされていると仮定しても問題ないでしょう。

35.2.6.2.3.4 サードパーティのGitOpsツールを使用したHelmチャートのアップグレード

ユーザがこのアップグレード手順をFleet以外のGitOpsワークフロー(Fluxなど)で実行したいユースケースが存在する場合があります。

アップグレード手順に必要なリソースを生成するには、`generate-chart-upgrade-data.sh`スクリプトを使用して、ユーザが提供するデータを`eib-charts-upgrader` Fleetに入力することができます。この実行方法の詳細については、35.2.6.2.3.2項「アップグレード手順」を参照してください。

完全なセットアップ後、[kustomize \(https://kustomize.io\)](https://kustomize.io) を使用して、クラスタにデプロイ可能な完全な動作ソリューションを生成することができます。

```
cd /foo/bar/fleets/day2/eib-charts-upgrader

kustomize build .
```

GitOpsワークフローにソリューションを含める場合は、`fleet.yaml`ファイルを削除して、残ったものを有効なKustomizeセットアップとして使用できます。まず、`generate-chart-upgrade-data.sh`スクリプトを実行し、アップグレードするHelmチャートのデータをKustomizeセットアップに読み込ませることを忘れないでください。

このワークフローの使用方法を理解するには、35.2.6.2.3.1項「概要」と35.2.6.2.3.2項「アップグレード手順」を参照すると役立つでしょう。

36 ダウンストリームクラスタ

！ 重要

以下の手順は、SUSE Telco Cloud (第37章「SUSE Telco Cloud」)によって管理されるダウンストリームクラスタには適用されません。ダウンストリームクラスタのアップグレードに関するガイダンスについては、43.2項「ダウンストリームクラスタのアップグレード」を参照してください。

このセクションでは、ダウンストリームクラスタのさまざまな部分に対して「Day 2」操作を実行可能な方法について説明します。

36.1 Fleet

このセクションでは、Fleet (第8章「Fleet」)コンポーネントを使用して「Day 2」操作を実行する方法について説明します。

次のトピックは、このセクションの一部として説明します。

1. 36.1.1項「コンポーネント」 - すべての「Day 2」操作に使用されるデフォルトコンポーネント。
2. 36.1.2項「ユースケースの決定」 - 使用されるFleetカスタムリソースの概要と、これらのリソースがさまざまな「Day 2」操作のユースケースに適しているかどうかを説明します。
3. 36.1.3項「Day 2ワークフロー」 - Fleetを使用して「Day 2」操作を実行するためのワークフローガイドを提供します。
4. 36.1.4項「OSアップグレード」 - Fleetを使用してOSアップグレードを実行する方法を説明します。
5. 36.1.5項「Kubernetesバージョンアップグレード」 - Fleetを使用してKubernetesバージョンアップグレードを実行する方法を説明します。
6. 36.1.6項「Helmチャートのアップグレード」 - Fleetを使用してHelmチャートアップグレードを実行する方法を説明します。

36.1.1 コンポーネント

以下に、Fleetを使用して「Day 2」操作を正常に実行できるように、ダウンストリームクラスターで設定する必要があるデフォルトコンポーネントについて説明します。

36.1.1.1 System Upgrade Controller (SUC)



注記

各ダウンストリームクラスター上にデプロイする**必要があります**。

System Upgrade Controllerは、Planと呼ばれるカスタムリソースを通じて提供される設定データに基づいて指定されたノードでタスクを実行する責任を負います。

SUCは、オペレーティングシステムとKubernetesディストリビューションのアップグレードに積極的に活用されます。

SUCコンポーネントとそれがEdgeスタックにどのように適合するかに関する詳細については、[第21章「System Upgrade Controller」](#)を参照してください。

SUCをデプロイする方法については、まずユースケースを決定し([36.1.2項「ユースケースの決定」](#))、次に「System Upgrade Controllerのインストール - GitRepo」([21.2.1.1項「System Upgrade Controllerのインストール - GitRepo」](#))、または「System Upgrade Controllerのインストール - バンドル」([21.2.1.2項「System Upgrade Controllerのインストール - バンドル」](#))を参照してください。

36.1.2 ユースケースの決定

Fleetは2種類の**カスタムリソース** (<https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/>) を使用してKubernetesおよびHelmリソースの管理を有効にします。

以下に、これらのリソースの目的と、「Day 2」操作のコンテキストに最適なユースケースに関する情報を示します。

36.1.2.1 GitRepo

GitRepoは、Fleetがバンドルの作成元として使用できるGitリポジトリを表すFleet (第8章「Fleet」)リソースです。各バンドルは、GitRepoリソースの内部で定義された設定パスに基づいて作成されます。詳細については、GitRepo (<https://fleet.rancher.io/gitrepo-add>) のドキュメントを参照してください。

「Day 2」操作のコンテキストでは、GitRepoリソースは通常、Fleet GitOpsアプローチを利用する非エアギャップ環境にSUCまたはSUC Planをデプロイするために使用されます。

または、リポジトリのセットアップをローカルGitサーバ経由でミラーリングする場合、GitRepoリソースを使用して、SUCまたはSUC Planをエアギャップ環境にデプロイすることもできます。

36.1.2.2 バンドル

バンドルは、ターゲットクラスタにデプロイする生のKubernetesリソースを保持します。バンドルは通常、GitRepoリソースから作成されますが、手動でデプロイできるユースケースもあります。詳細については、バンドル (<https://fleet.rancher.io/bundle-add>) のドキュメントを参照してください。

「Day 2」操作のコンテキストでは、バンドルリソースは通常、何らかの形態のローカルGitOps手法を使用しないエアギャップ環境(ローカルGitサーバなど)でSUCまたはSUC Planをデプロイするために使用されます。

または、ご自身のユースケースでGitOpsワークフローを使用できない場合は(Gitリポジトリを使用する場合など)、バンドルリソースを使用して、SUCまたはSUC Planを非エアギャップ環境にデプロイすることもできます。

36.1.3 Day 2ワークフロー

以下に、ダウンストリームクラスタを特定のEdgeリリースにアップグレードする際に従う必要のある「Day 2」ワークフローを示します。

1. OSアップグレード(36.1.4項「OSアップグレード」)
2. Kubernetesバージョンアップグレード(36.1.5項「Kubernetesバージョンアップグレード」)
3. Helmチャートアップグレード(36.1.6項「Helmチャートのアップグレード」)

36.1.4 OSアップグレード

このセクションでは、第8章「Fleet」と第21章「System Upgrade Controller」を使用してオペレーティングシステムのアップグレードを行う方法について説明します。

次のトピックは、このセクションの一部として説明します。

1. 36.1.4.1項「コンポーネント」 - アップグレードプロセスによって使用される追加のコンポーネント。
2. 36.1.4.2項「概要」 - アップグレードプロセスの概要。
3. 36.1.4.3項「要件」 - アップグレードプロセスの要件。
4. 36.1.4.4項「OSアップグレード - SUC Planのデプロイメント」 - アップグレードプロセスのトリガを担当する、SUC Planをデプロイする方法に関する情報。

36.1.4.1 コンポーネント

このセクションでは、OSアップグレードプロセスがデフォルトの「Day 2」コンポーネント(36.1.1項「コンポーネント」)よりも使用するカスタムコンポーネントについて説明します。

36.1.4.1.1 systemd.service

特定のノードでのOSアップグレードは、systemd.service (<https://www.freedesktop.org/software/systemd/man/latest/systemd.service.html>) によって処理されます。

あるEdgeバージョンから別のEdgeバージョンにOSが必要とするアップグレードのタイプに応じて、異なるサービスが作成されます。

- 同じOSバージョン(例: 6.0)を必要とするEdgeバージョンの場合、os-pkg-update.serviceが作成されます。このサービスはtransactional-update (<https://kubic.opensuse.org/documentation/man-pages/transactional-update.8.html>) を使用して、通常のパッケージアップグレード (https://en.opensuse.org/SDB:Zypper_usage#Updating_packages) を実行します。
- OSバージョンのマイグレーション(例: 6.0 → 6.1)が必要なEdgeバージョンの場合、os-migration.serviceが作成されます。このサービスは、transactional-update (<https://kubic.opensuse.org/documentation/man-pages/transactional-update.8.html>) を使用して以下の処理を実行します。

- a. 通常のパッケージのアップグレード (https://en.opensuse.org/SDB:Zypper_usage#Updating_packages) 。このアップグレードにより、すべてのパッケージを最新にすることで、古いパッケージバージョンに関連するマイグレーション時の障害を軽減します。
- b. `zypper migration` コマンドを利用したOSマイグレーション。

上記サービスは、OSアップグレードが必要なダウストリームクラスタ上に配置する必要があります。SUC Planを通じて各ノードに配布されます。

36.1.4.2 概要

ダウストリームクラスタノードのオペレーティングシステムのアップグレードは、FleetとSystem Upgrade Controller (SUC)を利用して実行されます。

Fleetは、SUC Planを目的のクラスタにデプロイおよび管理するために使用されます。



注記

SUC Planは、特定のタスクをノードセットで実行するためにSUCが従う必要がある手順を記述したカスタムリソース (<https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/>)  です。SUC Planの例については、アップストリームリポジトリ (<https://github.com/rancher/system-upgrade-controller?tab=readme-ov-file#example-plans>)  を参照してください。

OS SUC Planは、GitRepo (<https://fleet.rancher.io/gitrepo-add>)  または バンドル (<https://fleet.rancher.io/bundle-add>)  リソースを特定のFleetワークスペース (<https://fleet.rancher.io/namespaces#gitrepos-bundles-clusters-clustergroups>)  にデプロイすることで、各クラスタに配布されます。FleetはデプロイされたGitRepo/Bundleを取得し、その内容 (OS SUC plan) を目的のクラスタにデプロイします。



注記

GitRepo/バンドル リソースは常に、管理クラスタ上にデプロイされます。GitRepo リソースを使用するか バンドル リソースを使用するかは、ユースケースによって異なります。詳細については、[36.1.2項「ユースケースの決定」](#) をご確認ください。

OS SUC Planは、次のワークフローを記述します。

1. 常に、OSをアップグレードする前に、ノードを`cordon` (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_cordon/)  します。
2. 常に、ワーカーノードの前にコントロールプレーンノードをアップグレードします。
3. 常に、一度に **1**ノードずつクラスタをアップグレードします。

OS SUC Planがデプロイされると、ワークフローは次のようになります。

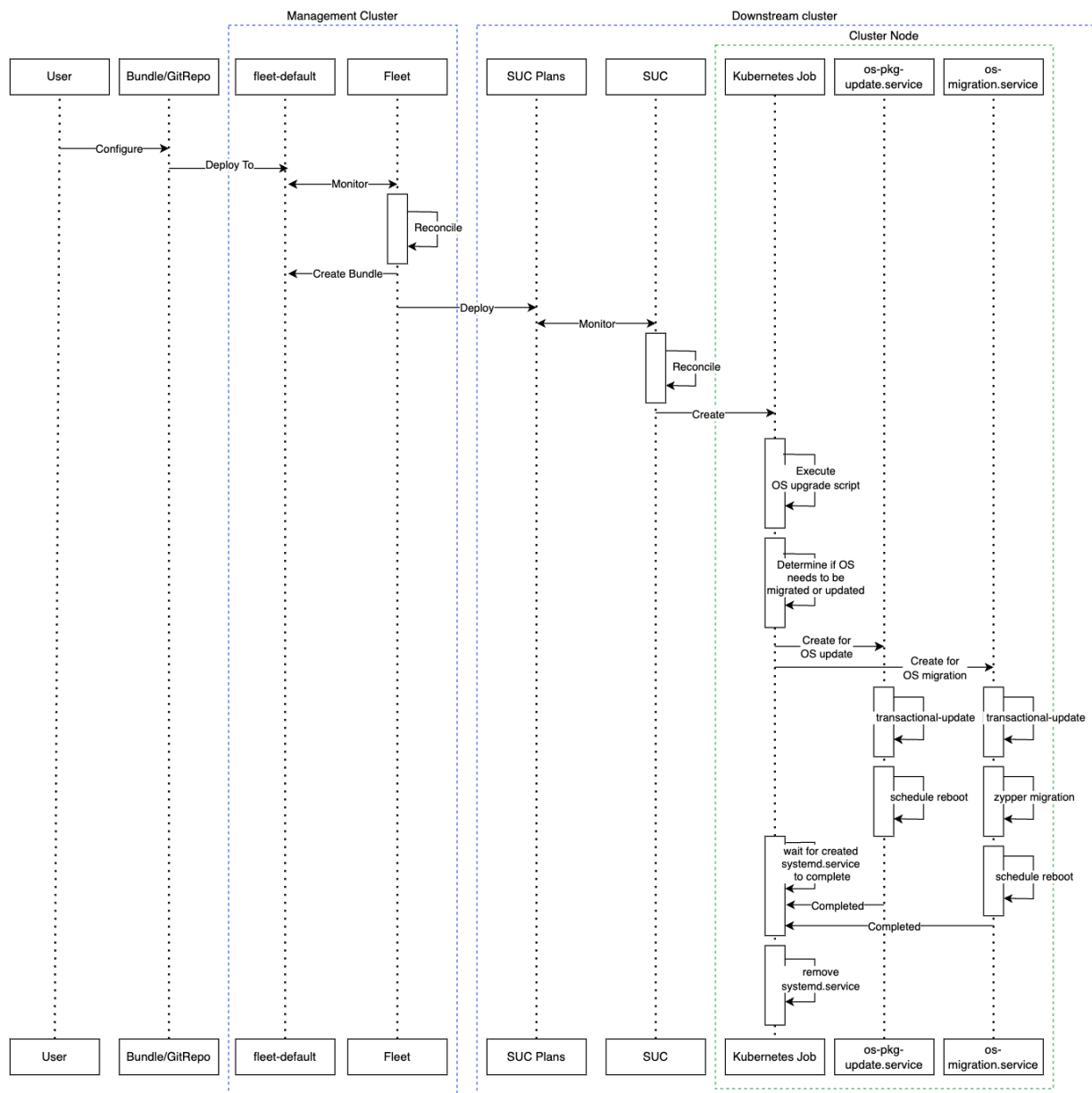
1. SUCは、デプロイされたOS SUC Planを照合し、Kubernetes Jobを**各ノード**に作成します。
2. Kubernetes Jobは、パッケージのアップグレードまたはOSマイグレーションのために `systemd.service` ([36.1.4.1項 「systemd.service」](#))を作成します。
3. 作成された `systemd.service`は、特定のノードでOSアップグレードプロセスをトリガします。



重要

OSアップグレードプロセスが終了すると、対応するノードが 再起動 され、システムに更新が適用されます。

上記の説明を以下に図示します。



36.1.4.3 要件

全般:

- SCC登録マシン** - すべてのダウンストリームクラスターノードを <https://scc.suse.com/> に登録する必要があります。これは、各 `systemd.service` が目的のRPMリポジトリに正常に接続できるようにするために必要です。



重要

OSバージョンのマイグレーション (例: 6.0 → 6.1)が必要なEdgeリリースの場合、SCCキーが新しいバージョンへのマイグレーションをサポートしていることを確認してください。

2. **SUC PlanのTolerationがノードのTolerationと一致すること** - KubernetesクラスターノードにカスタムのTaintが設定されている場合は、**SUC Plan**にそのTaintに対するToleration (<https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/>) ➦を追加してください。デフォルトでは、**SUC Plan**には、**コントロールプレーン**ノードのTolerationのみが含まれます。デフォルトのTolerationは次のとおりです。

- **CriticalAddonsOnly=true:NoExecute**
- **node-role.kubernetes.io/control-plane:NoSchedule**
- **node-role.kubernetes.io/etcd:NoExecute**



注記

追加のTolerationは、各Planの.spec.tolerationsセクションに追加する必要があります。OSアップグレードに関連する**SUC Plan**は、[fleets/day2/system-upgrade-controller-plans/os-upgrade](#)の下での[suse-edge/fleet-examples](#) (<https://github.com/suse-edge/fleet-examples>) ➦リポジトリにあります。**有効なリポジトリリリース** (<https://github.com/suse-edge/fleet-examples/releases>) ➦タグからのPlanを使用してください。

control-plane SUC Planに対してカスタムのTolerationを定義する例は次のようになります。

```
apiVersion: upgrade.cattle.io/v1
kind: Plan
metadata:
  name: os-upgrade-control-plane
spec:
  ...
  tolerations:
    # default tolerations
    - key: "CriticalAddonsOnly"
      operator: "Equal"
      value: "true"
      effect: "NoExecute"
```

```
- key: "node-role.kubernetes.io/control-plane"
  operator: "Equal"
  effect: "NoSchedule"
- key: "node-role.kubernetes.io/etcd"
  operator: "Equal"
  effect: "NoExecute"
# custom toleration
- key: "foo"
  operator: "Equal"
  value: "bar"
  effect: "NoSchedule"
...
```

エアギャップ:

1. **SUSE RPMリポジトリのミラーリング** - OS RPMリポジトリをローカルにミラーリングし、`systemd.service`がそのリポジトリにアクセスできるようにする必要があります。このためには、[RMT \(https://documentation.suse.com/sles/15-SP6/html/SLES-all/book-rmt.html\)](https://documentation.suse.com/sles/15-SP6/html/SLES-all/book-rmt.html) または [SUMA \(https://documentation.suse.com/suma/5.0/en/suse-manager/index.html\)](https://documentation.suse.com/suma/5.0/en/suse-manager/index.html) のいずれかを使用します。

36.1.4.4 OSアップグレード - SUC Planのデプロイメント

! 重要

この手順を使用して以前にアップグレードした環境の場合、ユーザは次の手順の**いずれか**を完了していることを確認する必要があります。

- ダウンストリームクラスタから古いEdgeリリースバージョンに関連する以前にデプロイしたSUC Planを削除する - これは、既存のGitRepo/バンドルターゲット設定 (<https://fleet.rancher.io/gitrepo-targets#target-matching>) から目的のクラスタを削除するか、GitRepo/バンドルリソースを完全に削除することで、実行できます。
- 既存のGitRepo/バンドルリソースを再利用する - 目的のsuse-edge/fleet-examplesリリース (<https://github.com/suse-edge/fleet-examples/releases>) の正しいフリートを保持する新しいタグにリソースのリビジョンをポイントすることで実行できます。

これは古いEdgeリリースバージョンのSUC Plan間のクラッシュを回避するために実行されます。

ユーザがアップグレードを試す場合、ダウンストリームクラスタに既存の SUC Plan がある場合は、次のフリートエラーが表示されます。

```
Not installed: Unable to continue with install: Plan <plan_name> in namespace <plan_namespace> exists and cannot be imported into the current release: invalid ownership metadata; annotation validation error..
```

36.1.4.2項「概要」で説明したように、OSアップグレードは、次の方法のいずれかの方法を使用して、SUC Planを目的のクラスタに配布することで実行されます。

- Fleet GitRepoリソース - 36.1.4.4.1項「SUC Planのデプロイメント - GitRepoリソース」。
- Fleet バンドルリソース - 36.1.4.4.2項「SUC Planのデプロイメント - バンドルリソース」。

どのリソースを使用すべきかを判断するには、36.1.2項「ユースケースの決定」を参照してください。

OS SUC Plan をサードパーティのGitOpsツールからデプロイするユースケースの場合は、36.1.4.4.3項「SUC Planのデプロイメント - サードパーティのGitOpsワークフロー」を参照してください。

36.1.4.4.1 SUC Planのデプロイメント - GitRepoリソース

必要なOS SUC Planを配布する、**GitRepo**リソースは、次の方法のいずれかでデプロイできます。

1. Rancher UI - 36.1.4.4.1.1項「GitRepoの作成 - Rancher UI」を通じて(Rancherが使用可能な場合)。
2. リソースを管理クラスタに手動でデプロイする(36.1.4.4.1.2項「GitRepoの作成 - 手動」)。

デプロイ後に、ターゲットクラスタのノードのOSアップグレードプロセスを監視するには、21.3項「System Upgrade Controller Planのモニタリング」を参照してください。

36.1.4.4.1.1 GitRepoの作成 - Rancher UI

Rancher UIを通じてGitRepoリソースを作成するには、公式のドキュメント (<https://ranchermanager.docs.rancher.com/v2.12/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui>) に従ってください。

Edgeチームは、すぐに使用できるフリート (<https://github.com/suse-edge/fleet-examples/tree/release-3.4.0/fleets/day2/system-upgrade-controller-plans/os-upgrade>) を維持しています。環境によっては、このフリートを直接使用することも、テンプレートとして使用することもできます。

！ 重要

常に、このフリートは有効なEdge リリース (<https://github.com/suse-edge/fleet-examples/releases>) タグから使用してください。

フリートが配布するSUC Planにカスタム変更を含める必要がないユースケースでは、ユーザは [suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples) リポジトリから [os-upgrade](#) フリートを直接参照できます。

カスタム変更(カスタム許容値の追加など)が必要な場合、ユーザは別のリポジトリから [os-upgrade](#) フリートを参照して、必要に応じてSUC Planに変更を追加できるようにする必要があります。

GitRepoを [suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples) リポジトリのフリートを使用するように設定する方法の例については、[こちら](https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/gitrepos/day2/os-upgrade-gitrepo.yaml) (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/gitrepos/day2/os-upgrade-gitrepo.yaml>) を参照してください。

36.1.4.4.1.2 GitRepoの作成 - 手動

1. GitRepoリソースをプルします。

```
curl -o os-upgrade-gitrepo.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.4.0/gitrepos/day2/os-upgrade-gitrepo.yaml
```

2. GitRepo設定を編集し、spec.targetsで目的のターゲットリストを指定します。デフォルトでは、[suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples) のGitRepoリソースはどのダウンストリームクラスタにもマップされません。

- すべてのクラスタ変更に一致させるには、デフォルトのGitRepoターゲットを次のように変更します。

```
spec:
```

```
targets:
- clusterSelector: {}
```

- または、クラスタをより細かく選択したい場合は、「[Mapping to Downstream Cluster \(ダウンストリームクラスタへのマッピング\)](https://fleet.rancher.io/gitrepo-targets) (<https://fleet.rancher.io/gitrepo-targets>)[🔗](#)」を参照してください。

3. **GitRepo**リソースを管理クラスタに適用します。

```
kubectl apply -f os-upgrade-gitrepo.yaml
```

4. `fleet-default`ネームスペースで、作成した**GitRepo**リソースを表示します。

```
kubectl get gitrepo os-upgrade -n fleet-default
```

Example output

NAME	REPO	COMMIT
BUNDLEDEPLOYMENTS-READY	STATUS	
os-upgrade	https://github.com/suse-edge/fleet-examples.git	release-3.4.0 0/0

36.1.4.4.2 **SUC Planのデプロイメント - バンドルリソース**

必要なOS `SUC Plan`を配布する**バンドルリソース**は、次の方法のいずれかでデプロイできます。

1. **Rancher UI** - 36.1.4.4.2.1項「**バンドルの作成 - Rancher UI**」を通じて(Rancherが使用可能な場合)。
2. リソースを管理クラスタに手動でデプロイする(36.1.4.4.2.2項「**バンドルの作成 - 手動**」)。

デプロイ後に、ターゲットクラスタのノードのOSアップグレードプロセスを監視するには、21.3項「[System Upgrade Controller Planのモニタリング](#)」を参照してください。

36.1.4.4.2.1 **バンドルの作成 - Rancher UI**

Edgeチームは、以下の手順で使用可能な、すぐに使用できる**バンドル** (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/bundles/day2/system-upgrade-controller-plans/os-upgrade/os-upgrade-bundle.yaml>)[🔗](#)を維持しています。

！ 重要

このバンドルは常に有効なEdgeリリース (<https://github.com/suse-edge/fleet-examples/releases>) をタグから使用してください。

RancherのUIを通じてバンドルを作成するには、次の手順に従います。

1. 左上隅で、 **[#]** → **[Continuous Delivery (継続的デリバリー)]** をクリックします。
2. **[Advanced (詳細)]** > **[Bundles (バンドル)]** に移動します。
3. **[Create from YAML (YAMLから作成)]** を選択します。
4. ここから次のいずれかの方法でバンドルを作成できます。

📎 注記

バンドルが配布するSUC Planにカスタム変更を含める必要があるユースケースがある場合があります(カスタム許容値の追加など)。これらの変更を、以下の手順で生成されるバンドルに必ず含めてください。

- a. バンドルコンテンツ (<https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.4.0/bundles/day2/system-upgrade-controller-plans/os-upgrade/os-upgrade-bundle.yaml>) を [suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples) から **[Create from YAML (YAMLから作成)]** ページに手動でコピーする。
- b. 目的のリリース (<https://github.com/suse-edge/fleet-examples/releases>) をタグから [suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples) (<https://github.com/suse-edge/fleet-examples>) のクローンを作成し、**[Create from YAML (YAMLから作成)]** ページの **[Read from File (ファイルから読み取り)]** オプションを選択する。ここからバンドルの場所([bundles/day2/system-upgrade-controller-plans/os-upgrade](https://github.com/suse-edge/fleet-examples/blob/main/bundles/day2/system-upgrade-controller-plans/os-upgrade/os-upgrade-bundle.yaml))に移動して、バンドルファイルを選択できます。これにより、バンドルコンテンツを持つ **[Create from YAML (YAMLから作成)]** ページが自動入力されます。

5. バンドルのターゲットクラスタを次のように変更します。

- すべてのダウンストリームクラスタに一致させるには、デフォルトのバンドル `.spec.targets` を次のように変更します。

```
spec:
  targets:
```

```
- clusterSelector: {}
```

- より細かくダウンストリームクラスタにマッピングするには、「[Mapping to Downstream Clusters \(ダウンストリームクラスタへのマッピング\)](https://fleet.rancher.io/gitrepo-targets) (<https://fleet.rancher.io/gitrepo-targets>) 

6. **【Create (作成)】** を選択します。

36.1.4.4.2 バンドルの作成 - 手動

1. バンドルリソースをプルします。

```
curl -o os-upgrade-bundle.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.4.0/bundles/day2/system-upgrade-controller-plans/os-upgrade/os-upgrade-bundle.yaml
```

2. バンドルの**ターゲット**設定を編集し、`spec.targets`に目的のターゲットリストを指定します。デフォルトでは、`suse-edge/fleet-examples`の**バンドル**リソースは、どのダウンストリームクラスタにもマップされません。

- すべてのクラスタに一致させるには、デフォルトの**バンドル**の**ターゲット**を次のように変更します。

```
spec:
  targets:
    - clusterSelector: {}
```

- または、クラスタをより細かく選択したい場合は、「[Mapping to Downstream Cluster \(ダウンストリームクラスタへのマッピング\)](https://fleet.rancher.io/gitrepo-targets) (<https://fleet.rancher.io/gitrepo-targets>) 

3. バンドルリソースを**管理クラスタ**に適用します。

```
kubectl apply -f os-upgrade-bundle.yaml
```

4. `fleet-default`名前スペースで、作成した**バンドル**リソースを表示します。

```
kubectl get bundles -n fleet-default
```

36.1.4.4.3 **SUC Planのデプロイメント - サードパーティのGitOpsワークフロー**

ユーザがOS SUC Planを独自のサードパーティGitOpsワークフロー(例: [Flux](#))に組み込むユースケースがある場合があります。

必要なOSアップグレードリソースを取得するには、まず使用する[suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples) (<https://github.com/suse-edge/fleet-examples>) リポジトリのEdge リリース (<https://github.com/suse-edge/fleet-examples/releases>) タグを決定します。

その後、リソースは[fleets/day2/system-upgrade-controller-plans/os-upgrade](#)で見つかります。ここでは次のようになります。

- [plan-control-plane.yaml](#)は、**コントロールプレーン**ノードのSUC Planリソースです。
- [plan-worker.yaml](#)は、**ワーカー**ノードのSUC Planリソースです。
- [secret.yaml](#)は、`systemd.service` (36.1.4.1.1項「`systemd.service`」)の作成を担当する、[upgrade.sh](#)スクリプトを含むSecretです。
- [config-map.yaml](#)は、[upgrade.sh](#)スクリプトによって使用される設定を保持するConfigMapです。

！ 重要

これらのPlanリソースは、`System Upgrade Controller`によって解釈され、アップグレードする各ダウンストリームクラスタにデプロイする必要があります。SUCデプロイメントの情報については、21.2項「`System Upgrade Controller`のインストール」を参照してください。

GitOpsワークフローをOSアップグレードの**SUC Plan**をデプロイするために使用する方法をよりよく理解するために、概要(36.1.4.2項「概要」)を確認すると役立つ場合があります。

36.1.5 Kubernetesバージョンアップグレード

！ 重要

このセクションでは、Rancher (第5章「`Rancher`」)インスタンスを使用して作成されていないダウンストリームクラスタのKubernetesアップグレードについて説明します。Rancherで作成したクラスタのKubernetesバージョンをアップグレードする方法については、「[Upgrading and Rolling Back Kubernetes \(Kubernetesのアップグレードとロールバック\)](https://ranchermanager.docs.rancher.com/v2.12/getting-started/installation-and-upgrade/upgrade-and-roll-back-kubernetes#upgrading-the-kubernetes-version) (<https://ranchermanager.docs.rancher.com/v2.12/getting-started/installation-and-upgrade/upgrade-and-roll-back-kubernetes#upgrading-the-kubernetes-version>)」を参照してください。

このセクションでは、第8章「Fleet」と第21章「System Upgrade Controller」を使用してKubernetesアップグレードを実行する方法について説明します。

次のトピックは、このセクションの一部として説明します。

1. 36.1.5.1項「コンポーネント」 - アップグレードプロセスで使用する追加のコンポーネント。
2. 36.1.5.2項「概要」 - アップグレードプロセスの概要。
3. 36.1.5.3項「要件」 - アップグレードプロセスの要件。
4. 36.1.5.4項「K8sアップグレード - SUC Planのデプロイメント」 - アップグレードプロセスのトリガを担当する、SUC Planをデプロイする方法に関する情報。

36.1.5.1 コンポーネント

このセクションでは、K8sアップグレードプロセスがデフォルトの「Day 2」コンポーネント(36.1.1項「コンポーネント」)よりも使用するカスタムコンポーネントについて説明します。

36.1.5.1.1 rke2-upgrade

特定のノードのRKE2バージョンのアップグレードを担当するコンテナイメージ。

SUC Planに基づいて**SUC**によって作成されたPodを通じて配布されます。このPlanは、RKE2のアップグレードが必要な各**クラスタ**に配置する必要があります。

rke2-upgradeイメージによるアップグレードの実行方法の詳細については、[アップストリーム \(https://github.com/rancher/rke2-upgrade/tree/master\)](https://github.com/rancher/rke2-upgrade/tree/master) ドキュメントを参照してください。

36.1.5.1.2 k3s-upgrade

特定のノードのK3sバージョンのアップグレードを担当するコンテナイメージ。

SUC Planに基づいて**SUC**によって作成されたPodを通じて配布されます。このPlanは、K3sのアップグレードが必要な各**クラスタ**に配置する必要があります。

k3s-upgradeイメージによるアップグレードの実行方法の詳細については、[アップストリーム \(https://github.com/k3s-io/k3s-upgrade\)](https://github.com/k3s-io/k3s-upgrade) ドキュメントを参照してください。

36.1.5.2 概要

ダウンストリームクラスタのKubernetesディストリビューションアップグレードは、FleetとSystem Upgrade Controller (SUC)を利用して実行されます。

Fleetは、SUC Planを目的のクラスタにデプロイし、管理するために使用されます。



注記

SUC Planは、**SUC**が特定のタスクをノードのセットで実行するために従う必要がある手順を記述したカスタムリソース (<https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/>)  です。SUC Planの例については、アップストリームリポジトリ (<https://github.com/rancher/system-upgrade-controller?tab=readme-ov-file#example-plans>)  を参照してください。

K8s SUC Planは、GitRepo (<https://fleet.rancher.io/gitrepo-add>)  または**バンドル** (<https://fleet.rancher.io/bundle-add>)  リソースを特定のFleetワークスペース (<https://fleet.rancher.io/namespaces#gitrepos-bundles-clusters-clustergroups>)  にデプロイすることで各クラスタに配布されます。FleetはデプロイされたGitRepo/バンドルを取得し、その内容(K8s SUC Plan)を目的のクラスタにデプロイします。



注記

GitRepo/バンドルリソースは常に、管理クラスタ上にデプロイされます。GitRepoリソースを使用するか**バンドル**リソースを使用するかは、ユースケースによって異なります。詳細については、[36.1.2項「ユースケースの決定」](#)をご確認ください。

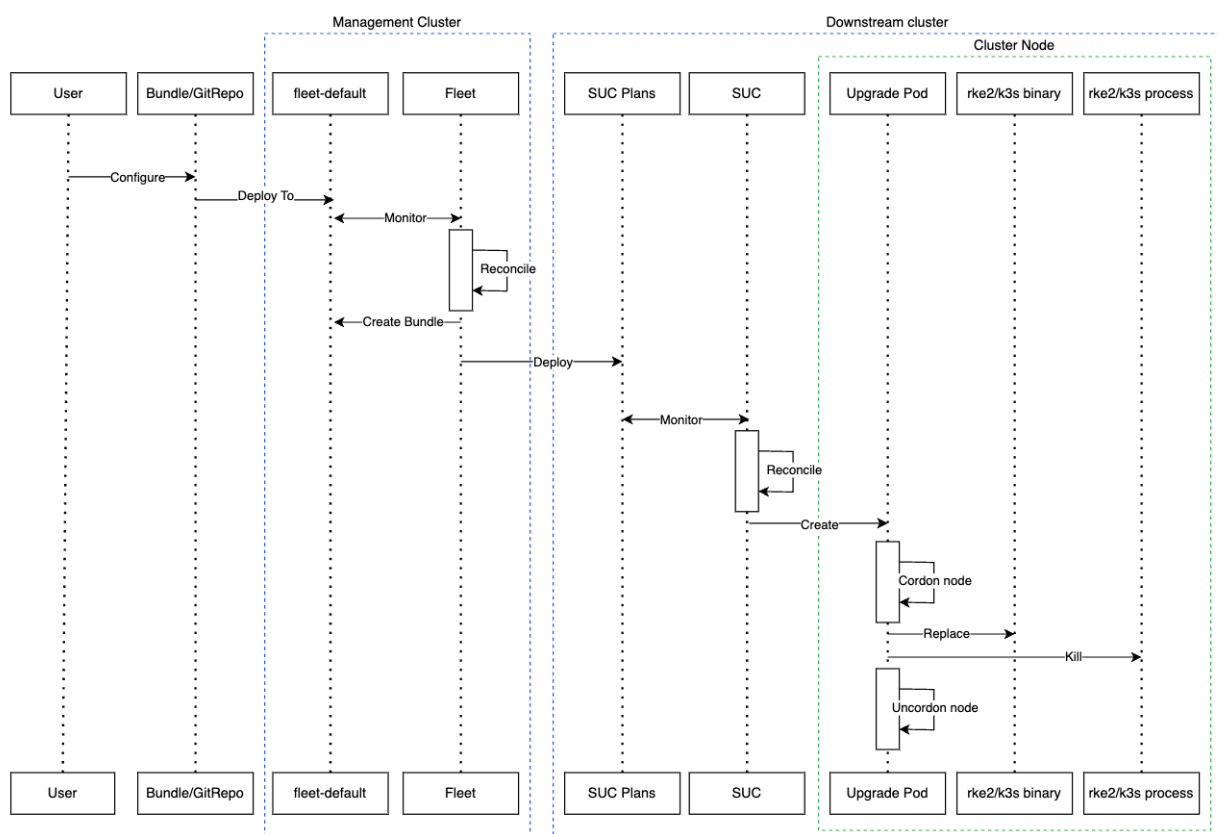
K8s SUC Planは、次のワークフローを記述します。

1. 常に、K8sをアップグレードする前に、ノードを**cordons** (https://kubernetes.io/docs/reference/kubectl/generated/kubectl_cordon/)  します。
2. 常に、ワーカーノードの前に**コントロールプレーン**ノードをアップグレードします。
3. 常に、一度に**1**ノードずつ**コントロールプレーン**ノードをアップグレードし、一度に**2**ノードずつ**ワーカーノード**をアップグレードします。

K8s SUC Planがデプロイされたら、ワークフローは次のようになります。

1. SUCは、デプロイ済みのK8s SUC Planを照合し、Kubernetes Jobを各ノードに作成します。
2. Kubernetesディストリビューションによって、Jobはrke2-upgrade (36.1.5.1.1項「rke2-upgrade」)またはk3s-upgrade (36.1.5.1.2項「k3s-upgrade」)のコンテナイメージを実行するPodを作成します。
3. 作成されたPodは次のワークフローを実行します。
 - a. ノード上の既存のrke2/k3sバイナリをrke2-upgrade/k3s-upgradeイメージのバイナリで置き換えます。
 - b. 実行中のrke2/k3sプロセスを強制終了します。
4. rke2/k3sプロセスを強制終了すると、再起動がトリガされ、更新されたバイナリを実行する新しいプロセスが起動し、Kubernetesディストリビューションバージョンがアップグレードされます。

上記の説明を以下に図示します。



36.1.5.3 要件

1. Kubernetesディストリビューションをバックアップします。

- a. RKE2クラスタについては、RKE2のバックアップと復元 (https://docs.rke2.io/datastore/backup_restore)  に関するドキュメントを参照してください。
- b. K3sクラスタについては、K3sのバックアップと復元 (<https://docs.k3s.io/datastore/backup-restore>)  に関するドキュメントを参照してください。

2. SUC PlanのTolerationがノードのTolerationと一致すること - KubernetesクラスタノードにカスタムのTaintが設定されている場合は、**SUC Plan**にそのTaintに対するToleration (<https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/>) を追加してください。デフォルトでは、**SUC Plan**には、**control-plane**ノードのTolerationのみが含まれます。デフォルトのTolerationは次のとおりです。

- **CriticalAddonsOnly=true:NoExecute**
- **node-role.kubernetes.io/control-plane:NoSchedule**
- **node-role.kubernetes.io/etcd:NoExecute**



注記

追加のTolerationは、各Planの `.spec.tolerations` セクションに追加する必要があります。Kubernetesバージョンアップグレードに関する**SUC Plan**は、次の場所の [suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples) (<https://github.com/suse-edge/fleet-examples>)  リポジトリにあります。

- **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)
- **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)

必ず、有効なリポジトリリリース (<https://github.com/suse-edge/fleet-examples/releases>)  **タグからのPlanを使用してください。**

RKE2 **control-plane** SUC PlanのカスタムTolerationの定義例は次のようになります。

```
apiVersion: upgrade.cattle.io/v1
kind: Plan
```

```

metadata:
  name: rke2-upgrade-control-plane
spec:
  ...
  tolerations:
    # default tolerations
    - key: "CriticalAddonsOnly"
      operator: "Equal"
      value: "true"
      effect: "NoExecute"
    - key: "node-role.kubernetes.io/control-plane"
      operator: "Equal"
      effect: "NoSchedule"
    - key: "node-role.kubernetes.io/etcd"
      operator: "Equal"
      effect: "NoExecute"
    # custom toleration
    - key: "foo"
      operator: "Equal"
      value: "bar"
      effect: "NoSchedule"
  ...

```

36.1.5.4 K8sアップグレード - SUC Planのデプロイメント

！ 重要

この手順を使用して以前にアップグレードした環境の場合、ユーザは次の手順の**いずれか**を完了していることを確認する必要があります。

- ダウンストリームクラスタから古いEdgeリリースバージョンに関連する以前にデプロイしたSUC Planを削除する - これは、既存のGitRepo/バンドル ターゲット設定 (<https://fleet.rancher.io/gitrepo-targets#target-matching>) から目的のクラスタを削除するか、GitRepo/バンドルリソースを完全に削除することで、実行できます。
- 既存のGitRepo/バンドルリソースを再利用する - 目的の suse-edge/fleet-examples リリース (<https://github.com/suse-edge/fleet-examples/releases>) の正しいフリートを保持する新しいタグにリソースのリビジョンをポイントすることで実行できます。

これは古いEdgeリリースバージョンの SUC Plan間のクラッシュを回避するために実行されます。

ユーザがアップグレードを試す場合、ダウンストリームクラスタに既存の SUC Plan がある場合は、次のフリートエラーが表示されます。

```
Not installed: Unable to continue with install: Plan <plan_name> in namespace
<plan_namespace> exists and cannot be imported into the current release: invalid
ownership metadata; annotation validation error..
```

36.1.5.2項「概要」で説明したように、Kubernetesアップグレードは、次のいずれかの方法を使用して SUC Plan を目的のクラスタに配布することで実行されます。

- Fleet GitRepoリソース(36.1.5.4.1項「SUC Planのデプロイメント - GitRepoリソース」)
- Fleetバンドルリソース(36.1.5.4.2項「SUC Planのデプロイメント - バンドルリソース」)

どのリソースを使用すべきかを判断するには、36.1.2項「ユースケースの決定」を参照してください。

K8s SUC Plan をサードパーティのGitOpsツールからデプロイするユースケースの場合は、36.1.5.4.3項「SUC Planのデプロイメント - サードパーティのGitOpsワークフロー」を参照してください。

36.1.5.4.1 SUC Planのデプロイメント - GitRepoリソース

必要なK8s SUC Plan を配布する、**GitRepo**リソースは、次の方法のいずれかでデプロイできます。

1. Rancher UI - 36.1.5.4.1.1項「GitRepoの作成 - Rancher UI」を通じて(Rancherが利用可能な場合)。
2. リソースを 管理クラスタ に手動でデプロイする(36.1.5.4.1.2項「GitRepoの作成 - 手動」)。

デプロイ後に、ターゲットクラスタのノードのKubernetesアップグレードプロセスを監視するには、21.3項「System Upgrade Controller Planのモニタリング」を参照してください。

36.1.5.4.1.1 GitRepoの作成 - Rancher UI

Rancher UIを通じてGitRepoリソースを作成するには、公式のドキュメント(<https://ranchermanager.docs.rancher.com/v2.12/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui>)に従ってください。

Edgeチームは、[rke2 \(https://github.com/suse-edge/fleet-examples/tree/release-3.4.0/fleets/day2/system-upgrade-controller-plans/rke2-upgrade\)](https://github.com/suse-edge/fleet-examples/tree/release-3.4.0/fleets/day2/system-upgrade-controller-plans/rke2-upgrade) と [k3s \(https://github.com/suse-edge/fleet-examples/tree/release-3.4.0/fleets/day2/system-upgrade-controller-plans/k3s-upgrade\)](https://github.com/suse-edge/fleet-examples/tree/release-3.4.0/fleets/day2/system-upgrade-controller-plans/k3s-upgrade) Kubernetesディストリビューションの両方のすぐに使用できるフリートを持っています。環境によっては、このフリートを直接使用することも、テンプレートとして使用することもできます。

！ 重要

常に、これらのフリートは有効なEdgeリリース (<https://github.com/suse-edge/fleet-examples/releases>) タグから使用してください。

これらのフリートが配布するSUC Planにカスタム変更を含める必要がないユースケースの場合、ユーザは[suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples)リポジトリからフリートを直接参照できます。

カスタム変更(カスタム許容値の追加など)が必要な場合、ユーザは別のリポジトリからフリートを参照して、必要に応じてSUC Planに変更を追加できるようにする必要があります。

[suse-edge/fleet-examples](https://github.com/suse-edge/fleet-examples)リポジトリからフリートを使用したGitRepoリソースの設定例は、次のとおりです。

- RKE2 (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/gitrepos/day2/rke2-upgrade-gitrepo.yaml>)
- K3s (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/gitrepos/day2/k3s-upgrade-gitrepo.yaml>)

36.1.5.4.1.2 GitRepoの作成 - 手動

1. GitRepoリソースをプルします。

- **RKE2**クラスタの場合:

```
curl -o rke2-upgrade-gitrepo.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.4.0/gitrepos/day2/rke2-upgrade-gitrepo.yaml
```

- **K3s**クラスタの場合:

```
curl -o k3s-upgrade-gitrepo.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.4.0/gitrepos/day2/k3s-upgrade-gitrepo.yaml
```

2. **GitRepo**設定を編集し、`spec.targets`で目的のターゲットリストを指定します。デフォルトでは、`suse-edge/fleet-examples`の**GitRepo**リソースはどのダウンストリームクラスタにもマップされません。

- すべてのクラスタ変更に一致させるには、デフォルトの**GitRepo****ターゲット**を次のように変更します。

```
spec:
  targets:
  - clusterSelector: {}
```

- または、クラスタをより細かく選択したい場合は、「[Mapping to Downstream Cluster \(ダウンストリームクラスタへのマッピング\)](https://fleet.rancher.io/gitrepo-targets) (<https://fleet.rancher.io/gitrepo-targets>)」を参照してください。

3. **GitRepo**リソースを**管理クラスタ**に適用します。

```
# RKE2
kubectl apply -f rke2-upgrade-gitrepo.yaml

# K3s
kubectl apply -f k3s-upgrade-gitrepo.yaml
```

4. `fleet-default`名前スペースで、作成した**GitRepo**リソースを表示します。

```
# RKE2
kubectl get gitrepo rke2-upgrade -n fleet-default

# K3s
kubectl get gitrepo k3s-upgrade -n fleet-default

# Example output
NAME                                REPO                                COMMIT
BUNDLEDEPLOYMENTS-READY  STATUS
k3s-upgrade                https://github.com/suse-edge/fleet-examples.git  fleet-default  0/0
rke2-upgrade                https://github.com/suse-edge/fleet-examples.git  fleet-default  0/0
```


36.1.5.4.2 SUC Planのデプロイメント - バンドルリソース

必要なKubernetesアップグレードSUC Planを配布する、バンドルリソースは、次の方法のいずれかでデプロイできます。

1. Rancher UI - 36.1.5.4.2.1項「バンドルの作成 - Rancher UI」を通じて(Rancherが利用可能な場合)。
2. リソースを管理クラスタに手動でデプロイする(36.1.5.4.2.2項「バンドルの作成 - 手動」)。

デプロイ後に、ターゲットクラスタのノードのKubernetesアップグレードプロセスを監視するには、21.3項「System Upgrade Controller Planのモニタリング」を参照してください。

36.1.5.4.2.1 バンドルの作成 - Rancher UI

Edgeチームは、rke2 (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/bundles/day2/system-upgrade-controller-plans/rke2-upgrade/plan-bundle.yaml>) と k3s (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/bundles/day2/system-upgrade-controller-plans/k3s-upgrade/plan-bundle.yaml>) Kubernetesディストリビューションの両方のすぐに使用できるバンドルを維持しています。環境によっては、これらのバンドルを直接使用することも、テンプレートとして使用することもできます。



重要

このバンドルは常に有効なEdgeリリース (<https://github.com/suse-edge/fleet-examples/releases>) タグから使用してください。

RancherのUIを通じてバンドルを作成するには、次の手順に従います。

1. 左上隅で、[#] → [Continuous Delivery (継続的デリバリー)] をクリックします。
2. [Advanced (詳細)] > [Bundles (バンドル)] に移動します。
3. [Create from YAML (YAMLから作成)] を選択します。
4. ここから次のいずれかの方法でバンドルを作成できます。



注記

バンドルが配布するSUC Planにカスタム変更を含める必要があるユースケースがある場合があります(カスタム許容値の追加など)。これらの変更を、以下の手順で生成されるバンドルに必ず含めてください。

- a. [suse-edge/fleet-examples](#)から **【Create from YAML (YAMLから作成)】** ページに RKE2 (<https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.4.0/bundles/day2/system-upgrade-controller-plans/rke2-upgrade/plan-bundle.yaml>) または K3s (<https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.4.0/bundles/day2/system-upgrade-controller-plans/k3s-upgrade/plan-bundle.yaml>) のバンドルコンテンツを手動でコピーする。
- b. 目的のリリース (<https://github.com/suse-edge/fleet-examples/releases>) タグから [suse-edge/fleet-examples](#) (<https://github.com/suse-edge/fleet-examples.git>) リポジトリのクローンを作成し、**【Create from YAML (YAMLから作成)】** ページの **【Read from File (ファイルから読み取り)】** オプションを選択する。そこから、必要なバンドルに移動します(RKE2の場合は `bundles/day2/system-upgrade-controller-plans/rke2-upgrade/plan-bundle.yaml`、K3sの場合は `bundles/day2/system-upgrade-controller-plans/k3s-upgrade/plan-bundle.yaml`)。 **【Create from YAML (YAMLから作成)】** ページにバンドルコンテンツが自動的に入力されます。

5. バンドルのターゲットクラスタを次のように変更します。

- すべてのダウンストリームクラスタに一致させるには、デフォルトのバンドル `.spec.targets` を次のように変更します。

```
spec:
  targets:
    - clusterSelector: {}
```

- より細かくダウンストリームクラスタにマッピングするには、「[Mapping to Downstream Clusters \(ダウンストリームクラスタへのマッピング\)](https://fleet.rancher.io/gitrepo-targets) (<https://fleet.rancher.io/gitrepo-targets>)」を参照してください。

6. **【Create (作成)】** を選択します。

36.1.5.4.2.2 バンドルの作成 - 手動

1. バンドルリソースをプルします。

- RKE2クラスタの場合:

```
curl -o rke2-plan-bundle.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.4.0/bundles/day2/system-upgrade-controller-plans/rke2-upgrade/plan-bundle.yaml
```

- K3sクラスタの場合:

```
curl -o k3s-plan-bundle.yaml https://raw.githubusercontent.com/suse-edge/fleet-examples/refs/tags/release-3.4.0/bundles/day2/system-upgrade-controller-plans/k3s-upgrade/plan-bundle.yaml
```

2. バンドルのターゲット設定を編集し、spec.targetsに目的のターゲットリストを指定します。デフォルトでは、suse-edge/fleet-examplesのバンドルリソースは、どのダウンストリームクラスタにもマップされません。

- すべてのクラスタに一致させるには、デフォルトのバンドルのターゲットを次のように変更します。

```
spec:
  targets:
  - clusterSelector: {}
```

- または、クラスタをより細かく選択したい場合は、「[Mapping to Downstream Cluster \(ダウンストリームクラスタへのマッピング\)](https://fleet.rancher.io/gitrepo-targets) (<https://fleet.rancher.io/gitrepo-targets>)」を参照してください。

3. バンドルリソースを管理クラスタに適用します。

```
# For RKE2
kubectl apply -f rke2-plan-bundle.yaml

# For K3s
kubectl apply -f k3s-plan-bundle.yaml
```

4. fleet-default名前スペースで、作成したバンドルリソースを表示します。

```
# For RKE2
kubectl get bundles rke2-upgrade -n fleet-default

# For K3s
kubectl get bundles k3s-upgrade -n fleet-default
```

```
# Example output
NAME                BUNDLEDEPLOYMENTS-READY  STATUS
k3s-upgrade         0/0
rke2-upgrade        0/0
```

36.1.5.4.3 SUC Planのデプロイメント - サードパーティのGitOpsワークフロー

ユーザがKubernetesアップグレードSUC Planを独自のサードパーティGitOpsワークフロー(例: Flux)に組み込むユースケースがある場合があります。

必要なK8sアップグレードリソースを取得するには、まず使用するsuse-edge/fleet-examples (<https://github.com/suse-edge/fleet-examples.git>) リポジトリのEdge リリース (<https://github.com/suse-edge/fleet-examples/releases>) タグを決定します。

その後、次の場所でリソースを確認できます。

- RKE2クラスタのアップグレードの場合:
 - control-planeノード用 - [fleets/day2/system-upgrade-controller-plans/rke2-upgrade/plan-control-plane.yaml](#)
 - ワーカーノードの場合 - [fleets/day2/system-upgrade-controller-plans/rke2-upgrade/plan-worker.yaml](#)
- K3sクラスタのアップグレードの場合:
 - control-planeノード用 - [fleets/day2/system-upgrade-controller-plans/k3s-upgrade/plan-control-plane.yaml](#)
 - ワーカーノードの場合 - [fleets/day2/system-upgrade-controller-plans/k3s-upgrade/plan-worker.yaml](#)

! 重要

これらのPlanリソースは、System Upgrade Controllerによって解釈され、アップグレードする各ダウンストリームクラスタにデプロイする必要があります。SUCデプロイメントの情報については、[21.2項「System Upgrade Controllerのインストール」](#)を参照してください。

GitOpsワークフローをKubernetesバージョンアップグレードの**SUC Plan**をデプロイするために使用する方法をよりよく理解するために、[Fleet](#)を使用する更新手順の概要([36.1.5.2項「概要」](#))を確認すると役立つ場合があります。

36.1.6 Helmチャートのアップグレード

このセクションでは、次の内容について説明します。

1. 36.1.6.1項「エアギャップ環境の準備」 - Edgeに関連するOCIチャートとイメージをプライベートレジストリに配布する方法に関する情報を保持します。
2. 36.1.6.2項「アップグレード手順」 - 異なるHelmチャートアップグレードのユースケースとそのアップグレード手順に関する情報を保持します。

36.1.6.1 エアギャップ環境の準備

36.1.6.1.1 HelmチャートFleetにアクセスできることの確認

環境でサポートする内容によって、次のオプションのいずれかを選択できます。

1. 管理クラスタからアクセス可能なローカルGitサーバでチャートのFleetリソースをホストします。
2. FleetのCLIを使用して、直接使用可能でどこかにホストする必要のない**バンドル**にHelmチャートを変換 (<https://fleet.rancher.io/bundle-add#convert-a-helm-chart-into-a-bundle>) します。FleetのCLIは、**リリース** (<https://github.com/rancher/fleet/releases/tag/v0.13.1>) ページから取得できます。Macユーザの場合は、**fleet-cli** (<https://formulae.brew.sh/formula/fleet-cli>) Homebrew Formulaeがあります。

36.1.6.1.2 Edgeリリースバージョンに必要なアセットの検索

1. 「Day 2」 **リリース** (<https://github.com/suse-edge/fleet-examples/releases>) のページに移動し、チャートのアップグレード先のEdgeリリースを見つけ、**[Assets (アセット)]** をクリックします。
2. **[Assets (アセット)]** セクションから、次のファイルをダウンロードします。

リリースファイル	説明
edge-save-images.sh	<code>edge-release-images.txt</code> ファイルで指定されたイメージを取得し、それらを「.tar.gz」アーカイブにパッケージ化します。

edge-save-oci-artefacts.sh	特定のEdgeリリースに関連するOCIチャートイメージを取得し、それらを「tar.gz」アーカイブにパッケージ化します。
edge-load-images.sh	「tar.gz」アーカイブからイメージをロードし、再タグ付けして、プライベートレジストリにプッシュします。
edge-load-oci-artefacts.sh	Edge OCI「.tgz」チャートパッケージを含むディレクトリを取得し、プライベートレジストリにロードします。
edge-release-helm-oci-artefacts.txt	特定のEdgeリリースに関連するOCIチャートイメージのリストを含みます。
edge-release-images.txt	特定のEdgeリリースに関連するイメージのリストを含みます。

36.1.6.1.3 Edgeリリースイメージアーカイブの作成

インターネットにアクセスできるマシンで次の手順を実行します。

1. `edge-save-images.sh`を実行可能にします。

```
chmod +x edge-save-images.sh
```

2. イメージアーカイブを生成します。

```
./edge-save-images.sh --source-registry registry.suse.com
```

3. これにより、`edge-images.tar.gz`という名前のすぐにロードできるアーカイブが作成されます。



注記

`-i|--images`オプションが指定される場合、アーカイブの名前は異なる場合があります。

4. このアーカイブをエアギャップマシンにコピーします。

```
scp edge-images.tar.gz <user>@<machine_ip>:/path
```

36.1.6.1.4 Edge OCIチャートイメージアーカイブの作成

インターネットにアクセスできるマシンで次の手順を実行します。

1. `edge-save-oci-artefacts.sh`を実行可能にします。

```
chmod +x edge-save-oci-artefacts.sh
```

2. OCIチャートイメージアーカイブを生成します。

```
./edge-save-oci-artefacts.sh --source-registry registry.suse.com
```

3. これにより、`oci-artefacts.tar.gz`という名前のアーカイブが作成されます。



注記

`-a|--archive`オプションが指定される場合、アーカイブの名前は異なる場合があります。

4. このアーカイブをエアギャップマシンにコピーします。

```
scp oci-artefacts.tar.gz <user>@<machine_ip>:/path
```

36.1.6.1.5 Edgeリリースイメージをエアギャップマシンにロード

エアギャップマシンで次の手順を実行します。

1. プライベートレジストリにログインします(必要な場合)。

```
podman login <REGISTRY.YOURDOMAIN.COM:PORT>
```

2. `edge-load-images.sh`を実行可能にします。

```
chmod +x edge-load-images.sh
```

3. 以前にコピーした `edge-images.tar.gz` アーカイブを渡して、スクリプトを実行します。

```
./edge-load-images.sh --source-registry registry.suse.com --registry  
<REGISTRY.YOURDOMAIN.COM:PORT> --images edge-images.tar.gz
```



注記

これにより、`edge-images.tar.gz`からすべてのイメージがロードされ、再タグ付けされて、それらを `--registry` オプションで指定されているレジストリにプッシュします。

36.1.6.1.6 Edge OCIチャートイメージのエアギャップマシンへのロード

エアギャップマシンで次の手順を実行します。

1. プライベートレジストリにログインします(必要な場合)。

```
podman login <REGISTRY.YOURDOMAIN.COM:PORT>
```

2. `edge-load-oci-artefacts.sh`を実行可能にします。

```
chmod +x edge-load-oci-artefacts.sh
```

3. コピーした`oci-artefacts.tar.gz`アーカイブをuntarします。

```
tar -xvf oci-artefacts.tar.gz
```

4. 命名テンプレート`edge-release-oci-tgz-<date>`を含むディレクトリが生成されます。
5. このディレクトリを`edge-load-oci-artefacts.sh`スクリプトに渡し、Edge OCIチャートイメージをプライベートレジストリにロードします。



注記

このスクリプトは、`Helm` CLIが環境にプリインストールされていることを想定しています。`Helm`のインストール手順については、「[Installing Helm \(Helmのインストール\)](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
```


36.1.6.1.7 Kubernetesディストリビューションでプライベートレジストリを設定する

RKE2の場合は、「Private Registry Configuration (プライベートレジストリの設定) (https://docs.rke2.io/install/private_registry)」を参照してください。

K3sの場合は、「Private Registry Configuration (プライベートレジストリの設定) (<https://docs.k3s.io/installation/private-registry>)」を参照してください。

36.1.6.2 アップグレード手順

このセクションでは、Helmアップグレード手順の次のユースケースを中心に説明します。

1. 36.1.6.2.1項「新しいクラスタがあり、Edge Helmチャートをデプロイして管理したい」
2. 36.1.6.2.2項「Fleetで管理されているHelmチャートをアップグレードしたい場合」
3. 36.1.6.2.3項「EIBを介してデプロイされたHelmチャートをアップグレードしたい」

！ 重要

手動でデプロイしたHelmチャートを確実にアップグレードすることはできません。36.1.6.2.1項「新しいクラスタがあり、Edge Helmチャートをデプロイして管理したい」で説明する方法を使用してHelmチャートを再デプロイすることをお勧めします。

36.1.6.2.1 新しいクラスタがあり、Edge Helmチャートをデプロイして管理したい

このセクションでは、以下の実行方法について説明します。

1. 36.1.6.2.1.1項「チャートのFleetリソースの準備」.
2. 36.1.6.2.1.2項「チャートのFleetのデプロイ」.
3. 36.1.6.2.1.3項「デプロイされたHelmチャートの管理」.

36.1.6.2.1.1 チャートのFleetリソースの準備

1. チャートのFleetリソースを、使用するEdgeリリース (<https://github.com/suse-edge/fleet-examples/releases>)  タグから取得します。
2. HelmチャートのFleet (`fleets/day2/chart-templates/<chart>`)に移動します。
3. **GitOpsワークフローを使用する場合は**、チャートFleetディレクトリをGitOpsを実行するGitリポジトリにコピーします。
4. (任意) Helmチャートで値を設定する必要がある場合は、コピーしたディレクトリの`fleet.yaml`ファイルに含まれる設定`.helm.values`を編集します。
5. (任意) 環境に合わせて、チャートのFleetにリソースを追加しなければならないユースケースがあります。Fleetディレクトリを拡張する方法については、「[Git Repository Contents \(Gitリポジトリのコンテンツ\)](https://fleet.rancher.io/gitrepo-content) (<https://fleet.rancher.io/gitrepo-content>) 



注記

場合によっては、Fleet がHelm操作に使用するデフォルトのタイムアウトが不十分で、次のエラーが発生する可能性があります。

```
failed pre-install: context deadline exceeded
```

このような場合は、`fleet.yaml`ファイルの`helm`設定の下に、`timeoutSeconds` (<https://fleet.rancher.io/ref-crds#helmoptions>)  プロパティを追加してください。

longhorn Helmチャートの例は次のようになります。

- ユーザGitリポジトリ構造:

```
<user_repository_root>
├─ longhorn
│   └─ fleet.yaml
└─ longhorn-crd
    └─ fleet.yaml
```

- ユーザのLonghornデータが入力された`fleet.yaml`の内容:

```
defaultNamespace: longhorn-system

helm:
  # timeoutSeconds: 10
```

```

releaseName: "longhorn"
chart: "longhorn"
repo: "https://charts.rancher.io/"
version: "107.0.0+up1.9.1"
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"}

```



注記

これらは値の例であり、longhornチャートのカスタム設定を示すために使用しているだけです。longhornチャートのデプロイメントのガイドラインとみなさないでください。

36.1.6.2.1.2 チャートのFleetのデプロイ

チャートのFleetをデプロイするには、GitRepo (36.1.6.2.1.2.1項「GitRepo」) またはバンドル (36.1.6.2.1.2.2項「バンドル」) のいずれかを使用できます。



注記

Fleetをデプロイする場合に、Modifiedメッセージを取得した場合、Fleetのdiffセクションに対応するcomparePatchesエントリを追加してください。詳細については、「[Generating Diffs to Ignore Modified GitRepos \(変更されたGitReposを無視する差分を生成する\)](https://fleet.rancher.io/bundle-diffs) (<https://fleet.rancher.io/bundle-diffs>)

36.1.6.2.1.2.1 GitRepo

FleetのGitRepo (<https://fleet.rancher.io/ref-gitrepo>) リソースは、チャートのFleetリソースへのアクセス方法、それらのリソースを適用するのに必要なクラスタに関する情報を保持しています。

GitRepoリソースは、[Rancher UI](https://ranchermanager.docs.rancher.com/v2.12/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui) (<https://ranchermanager.docs.rancher.com/v2.12/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui>) を通じて、または手動で管理クラスタにリソースをデプロイ (<https://fleet.rancher.io/tut-deployment>) することで、デプロイできます。

手動デプロイメントのLonghorn GitRepoリソースの例:

```
apiVersion: fleet.cattle.io/v1alpha1
kind: GitRepo
metadata:
  name: longhorn-git-repo
  namespace: fleet-default
spec:
  # If using a tag
  # revision: user_repository_tag
  #
  # If using a branch
  # branch: user_repository_branch
  paths:
    # As seen in the 'Prepare your Fleet resources' example
    - longhorn
    - longhorn-crd
  repo: user_repository_url
  targets:
    # Match all clusters
    - clusterSelector: {}
```

36.1.6.2.1.2.2 バンドル

バンドル (<https://fleet.rancher.io/bundle-add>)  リソースは、Fleetによってデプロイされる必要がある生のKubernetesリソースを保持しています。通常、[GitRepo](#)アプローチを使用することを推奨されますが、環境がエアギャップされ、ローカルGitサーバをサポートできないユースケース用に、**バンドル**がHelmチャートFleetをターゲットクラスタに伝播するのに役立ちます。

バンドルは、Rancher UI ([Continuous Delivery (継続的デリバリ)] → [Advanced (詳細)] → [Bundles (バンドル)] → [Create from YAML (YAMLから作成)])を介して、または**バンドル**リソースを正しいFleet名前スペースに手動でデプロイして、デプロイできます。Fleet名前スペースについては、[アップストリームドキュメント \(https://fleet.rancher.io/namespaces#gitrepos-bundles-clusters-clustergroups\)](https://fleet.rancher.io/namespaces#gitrepos-bundles-clusters-clustergroups)  を参照してください。

Edge Helmチャートの**バンドル**は、Fleetの「[Convert a Helm Chart into a Bundle \(Helmチャートをバンドルに変換する\)](https://fleet.rancher.io/bundle-add#convert-a-helm-chart-into-a-bundle) (<https://fleet.rancher.io/bundle-add#convert-a-helm-chart-into-a-bundle>) 」アプローチを利用して作成できます。

以下に、**バンドル**リソースを[longhorn \(https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/fleets/day2/chart-templates/longhorn/longhorn/fleet.yaml\)](https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/fleets/day2/chart-templates/longhorn/longhorn/fleet.yaml)  と[longhorn-crd \(https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/fleets/day2/chart-templates/longhorn/longhorn-crd/fleet.yaml\)](https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/fleets/day2/chart-templates/longhorn/longhorn-crd/fleet.yaml)  HelmチャートFleetテンプレートから作成し、このバンドルを**管理クラスタ**に手動でデプロイする方法の例を示します。



注記

ワークフローを説明するため、以下の例では[suse-edge/fleet-examples \(https://github.com/suse-edge/fleet-examples\)](https://github.com/suse-edge/fleet-examples)  ディレクトリ構造を使用しています。

1. [longhorn \(https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/fleets/day2/chart-templates/longhorn/longhorn/fleet.yaml\)](https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/fleets/day2/chart-templates/longhorn/longhorn/fleet.yaml)  チャートFleetテンプレートに移動します。

```
cd fleets/day2/chart-templates/longhorn/longhorn
```

2. FleetにHelmチャートのデプロイ先クラスタを指示する**targets.yaml**ファイルを作成します。

```
cat > targets.yaml <<EOF
targets:
# Matches all downstream clusters
- clusterSelector: {}
EOF
```

ダウンストリームクラスタをより細かく選択したい場合は、「[Mapping to Downstream Cluster \(ダウンストリームクラスタへのマッピング\)](https://fleet.rancher.io/gitrepo-targets) (<https://fleet.rancher.io/gitrepo-targets>) 

」を参照してください。

3. [fleet-cli](https://fleet.rancher.io/cli/fleet-cli/fleet) (<https://fleet.rancher.io/cli/fleet-cli/fleet>)  を使用して、[Longhorn Helm](#) チャート Fleet をバンドルリソースに変換します。



注記

Fleet の CLI は、[リリース](https://github.com/rancher/fleet/releases/tag/v0.13.1) (<https://github.com/rancher/fleet/releases/tag/v0.13.1>)  [アセット](#) ページから取得できます (`fleet-linux-amd64`)。

Mac ユーザの場合、[fleet-cli](https://formulae.brew.sh/formula/fleet-cli) (<https://formulae.brew.sh/formula/fleet-cli>)  Homebrew Formulae があります。

```
fleet apply --compress --targets-file=targets.yaml -n fleet-default -o - longhorn-bundle > longhorn-bundle.yaml
```

4. [longhorn-crd](https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/fleets/day2/chart-templates/longhorn/longhorn-crd/fleet.yaml) (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/fleets/day2/chart-templates/longhorn/longhorn-crd/fleet.yaml>)  チャート Fleet テンプレートに移動します。

```
cd fleets/day2/chart-templates/longhorn/longhorn-crd
```

5. Fleet に Helm チャートのデプロイ先クラスタを指示する `targets.yaml` ファイルを作成します。

```
cat > targets.yaml <<EOF
targets:
# Matches all downstream clusters
- clusterSelector: {}
EOF
```

6. [fleet-cli](https://fleet.rancher.io/cli/fleet-cli/fleet) (<https://fleet.rancher.io/cli/fleet-cli/fleet>)  を使用して、[Longhorn CRD Helm](#) チャート Fleet をバンドルリソースに変換します。

```
fleet apply --compress --targets-file=targets.yaml -n fleet-default -o - longhorn-crd-bundle > longhorn-crd-bundle.yaml
```

7. `longhorn-bundle.yaml` と `longhorn-crd-bundle.yaml` ファイルを [管理クラスタ](#) にデプロイします。

```
kubectl apply -f longhorn-crd-bundle.yaml
```

```
kubectl apply -f longhorn-bundle.yaml
```

これらの手順に従うと、SUSE Storageが指定されたダウストリームクラスタのすべてにデプロイされます。

36.1.6.2.1.3 デプロイされたHelmチャートの管理

Fleetでデプロイした後のHelmチャートのアップグレードについては、[36.1.6.2.2項「Fleetで管理されているHelmチャートをアップグレードしたい場合」](#)を参照してください。

36.1.6.2.2 Fleetで管理されているHelmチャートをアップグレードしたい場合

1. 目的のEdgeリリースと互換性を持つように、チャートをアップブレードする必要があるバージョンを決定します。EdgeリリースごとのHelmチャートバージョンはリリースノート([52.1項「要約」](#))から表示できます。
2. Fleetで監視されているGitリポジトリで、Helmチャートのfleet.yamlファイルを、リリースノート([52.1項「要約」](#))から取得した正しいチャートのバージョンとリポジトリで編集します。
3. 変更をコミットしてリポジトリにプッシュした後で、目的のHelmチャートのアップグレードがトリガされます。

36.1.6.2.3 EIBを介してデプロイされたHelmチャートをアップグレードしたい

第11章「Edge Image Builder」は、HelmChartリソースを作成し、[RKE2 \(https://docs.rke2.io/helm\)](https://docs.rke2.io/helm)  [K3s \(https://docs.k3s.io/helm\)](https://docs.k3s.io/helm)  Helm統合機能によって導入されたhelm-controllerを利用して、Helmチャートをデプロイします。

EIBを介してデプロイされたHelmチャートが正常にアップグレードされるようにするには、ユーザは各HelmChartリソースに対してアップグレードを実行する必要があります。

以下に関する情報が提供されています。

- アップグレードプロセスの一般的な概要([36.1.6.2.3.1項「概要」](#))。
- 必要なアップグレード手順([36.1.6.2.3.2項「アップグレード手順」](#))。
- 説明した方法を使用したLonghorn (<https://longhorn.io>)  チャートアップグレードを示す例([36.1.6.2.3.3項「例」](#))。
- 異なるGitOpsツールを使用したアップグレードプロセスを使用する方法([36.1.6.2.3.4項「サードパーティのGitOpsツールを使用したHelmチャートのアップグレード」](#))。

36.1.6.2.3.1 概要

EIBを介してデプロイされたHelmチャートは、[eib-charts-upgrader \(https://github.com/suse-edge/fleet-examples/tree/release-3.4.0/fleets/day2/eib-charts-upgrader\)](https://github.com/suse-edge/fleet-examples/tree/release-3.4.0/fleets/day2/eib-charts-upgrader)  と呼ばれるフリートを通じてアップグレードされます。

このフリートは、**ユーザが提供した**データを処理し、特定のHelmChartリソースセットを更新します。

これらのリソースを更新すると、[helm-controller \(https://github.com/k3s-io/helm-controller\)](https://github.com/k3s-io/helm-controller)  がトリガされ、変更されたHelmChartリソースに関連付けられているHelmチャートが**アップグレード**されます。

ユーザは以下を行うことのみ求められます。

1. アップグレードが必要な各Helmチャートのアーカイブをローカルでプル (https://helm.sh/docs/helm/helm_pull/)  します。
2. これらのアーカイブを[generate-chart-upgrade-data.sh \(https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/scripts/day2/generate-chart-upgrade-data.sh\)](https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/scripts/day2/generate-chart-upgrade-data.sh)  [generate-chart-upgrade-data.sh](#)スクリプトに渡します。これにより、これらのアーカイブのデータが [eib-charts-upgrader](#) フリートに含まれます。
3. [eib-charts-upgrader](#) Fleetを管理クラスタにデプロイします。これは[GitRepo](#)リソースまたは[バンドル](#)リソースのいずれかを通じて実行されます。

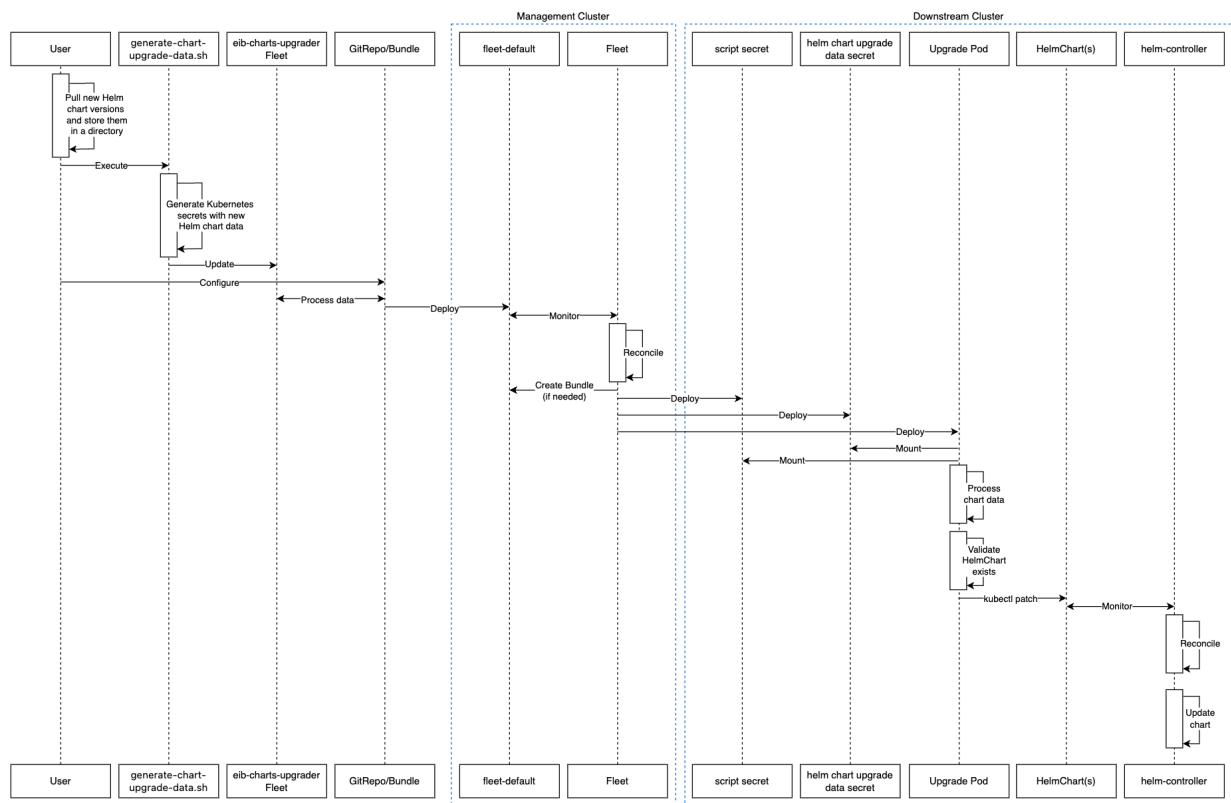
デプロイされたら、[eib-charts-upgrader](#)がFleetのサポートによって、そのリソースを目的のダウンストリームクラスタに配布します。

これらのリソースには、次のものが含まれます。

1. **ユーザが提供する**Helmチャートデータを保持する[Secret](#)のセット。
2. 前述のSecretをマウントし、それらに基づいて対応するHelmChartリソースに[パッチを適用する \(https://kubernetes.io/docs/reference/kubectl/generated/kubectl_patch/\)](#)  [Pod](#)をデプロイする[Kubernetes Job](#)。

前述のとおり、これにより、[helm-controller](#)がトリガされ、実際のHelmチャートアップグレードが実行されます。

上記の説明を以下に図示します。



36.1.6.2.3.2 アップグレード手順

- 正しいリリース タグ (<https://github.com/suse-edge/fleet-examples/releases/tag/release-3.4.0>) から `suse-edge/fleet-examples` リポジトリのクローンを作成します。
- 取得したHelmチャートアーカイブを保存するディレクトリを作成します。

```
mkdir archives
```

- 新規に作成されたアーカイブディレクトリ内で、アップグレードするHelmチャートのアーカイブをプル (https://helm.sh/docs/helm/helm_pull/) します。

```
cd archives
helm pull [chart URL | repo/chartname]

# Alternatively if you want to pull a specific version:
# helm pull [chart URL | repo/chartname] --version 0.0.0
```

4. 目的のリリースタグ (<https://github.com/suse-edge/fleet-examples/releases/tag/release-3.4.0>) のアセットから、`generate-chart-upgrade-data.sh` スクリプトをダウンロードします。
5. `generate-chart-upgrade-data.sh` スクリプトを実行します。

```
chmod +x ./generate-chart-upgrade-data.sh

./generate-chart-upgrade-data.sh --archive-dir /foo/bar/archives/ --fleet-path /foo/bar/fleet-examples/fleets/day2/eib-charts-upgrader
```

`--archive-dir` ディレクトリ内のチャートアーカイブごとに、スクリプトにより、チャートアップグレードデータを含む `Kubernetes Secret YAML` ファイルが生成され、`--fleet-path` によって指定されたフリートの `base/secrets` ディレクトリに保存されます。

`generate-chart-upgrade-data.sh` スクリプトは、フリートに追加の変更も適用し、生成された `Kubernetes Secret YAML` ファイルが、フリートによってデプロイされたワークロードによって正しく利用されるようにします。

！ 重要

ユーザは、`generate-chart-upgrade-data.sh` スクリプトが生成する内容に変更を加えてはなりません。

以下の手順は、実行している環境によって異なります。

1. GitOpsをサポートする環境の場合(例: エアギャップされていない、エアギャップされたがローカルGitサーバサポートが可能):
 - a. `fleets/day2/eib-charts-upgrader` Fleetを、GitOpsに使用するリポジトリにコピーします。

📎 注記

Fleetに`generate-chart-upgrade-data.sh` スクリプトによって加えられた変更が含まれていることを確認してください。

- b. `eib-charts-upgrader` Fleetのすべてのリソースを配布するために使用される `GitRepo` リソースを設定します。

- i. Rancher UIを通じたGitRepoの設定およびデプロイメントについては、「[Accessing Fleet in the Rancher UI \(Rancher UIでのFleetへのアクセス\)](https://ranchermanager.docs.rancher.com/v2.12/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui) (<https://ranchermanager.docs.rancher.com/v2.12/integrations-in-rancher/fleet/overview#accessing-fleet-in-the-rancher-ui>)」を参照してください。
- ii. GitRepo手動設定およびデプロイメントの場合、「[Creating a Deployment \(デプロイメントの作成\)](https://fleet.rancher.io/tut-deployment) (<https://fleet.rancher.io/tut-deployment>)」を参照してください。

2. GitOpsをサポートしていない環境の場合 (例: エアギャップされ、ローカルGitサーバの使用が許可されていない):

- a. [rancher/fleet](https://github.com/rancher/fleet/releases/tag/v0.13.1) リリース (<https://github.com/rancher/fleet/releases/tag/v0.13.1>) ページから `fleet-cli` バイナリをダウンロードします (Linuxの場合は `fleet-linux-amd64`)。Macユーザの場合、使用可能なHomebrew Formulaeがあります (`fleet-cli` (<https://formulae.brew.sh/formula/fleet-cli>))。

- b. `eib-charts-upgrader` Fleetに移動します。

```
cd /foo/bar/fleet-examples/fleets/day2/eib-charts-upgrader
```

- c. リソースをデプロイする場所をFleetに指示する `targets.yaml` ファイルを作成します。

```
cat > targets.yaml <<EOF
targets:
# To match all downstream clusters
- clusterSelector: {}
EOF
```

ターゲットクラスタをマップする方法については、アップストリーム [ドキュメント](https://fleet.rancher.io/gitrepo-targets) (<https://fleet.rancher.io/gitrepo-targets>) を参照してください。

- d. `fleet-cli` を使用して、Fleetを `バンドル` リソースに変換します。

```
fleet apply --compress --targets-file=targets.yaml -n fleet-default -o - eib-charts-upgrade > bundle.yaml
```

これにより、`eib-charts-upgrader` Fleetからのすべてのテンプレート化されたリソースを保持するバンドル (`bundle.yaml`) が作成されます。

`fleet apply` コマンドに関する詳細については、「[fleet apply](https://fleet.rancher.io/cli/fleet-cli/fleet_apply) (https://fleet.rancher.io/cli/fleet-cli/fleet_apply)」を参照してください。

Fleetをバンドルに変換する方法に関する詳細については、「[Convert a Helm Chart into a Bundle \(Helmチャートのバンドルへの変換\)](https://fleet.rancher.io/bundle-add#convert-a-helm-chart-into-a-bundle) (<https://fleet.rancher.io/bundle-add#convert-a-helm-chart-into-a-bundle>)」を参照してください。

- e. バンドルをデプロイします。これは次の2つの方法のいずれかで実行できます。
 - i. RancherのUIを通じて - **[Continuous Delivery (継続的デリバリー)]** → **[Advanced (詳細)]** → **[Bundles (バンドル)]** → **[Create from YAML (YAMLから作成)]** に移動して、`bundle.yaml` コンテンツを解析するか、**[Read from File (ファイルから読み取り)]** オプションをクリックして、ファイル自体を渡します。
 - ii. 手動 - 管理クラスタ内に`bundle.yaml`ファイルを手動でデプロイします。

これらの手順を実行すると、正常にGitRepo/バンドルリソースがデプロイされます。リソースはFleetによって取得され、そのコンテンツはユーザが以前の手順で指定したターゲットクラスタにデプロイされます。プロセスの概要については、[36.1.6.2.3.1項「概要」](#)を参照してください。

アップグレードプロセスの追跡方法については、[36.1.6.2.3.3項「例」](#)を参照してください。

！ 重要

チャートアップグレードが正常に確認されたら、バンドル/GitRepoリソースを削除します。

これにより、ダウンストリームクラスタから不要になったアップグレードリソースが削除され、今後バージョンクラッシュが発生しないようになります。

36.1.6.2.3.3 例

📝 注記

以下の例は、ダウンストリームクラスタ上で、EIBを介してデプロイされたHelmチャートを、あるバージョンから別のバージョンにアップグレードする方法を示しています。この例で使用されているバージョンは、**推奨バージョンではない**ことに注意してください。Edgeリリースに固有のバージョン推奨事項については、リリースノート([52.1項「要約」](#))を参照してください。

ユースケース:

- クラスタ名 `doc-example` が [Longhorn \(https://longhorn.io\)](https://longhorn.io) の古いバージョンを実行しています。
- クラスタは、次のイメージ定義スニペットを使用して、EIBを通じてデプロイされました。

```
kubernetes:
  helm:
    charts:
      - name: longhorn-crd
        repositoryName: rancher-charts
        targetNamespace: longhorn-system
        createNamespace: true
        version: 104.2.0+up1.7.1
        installationNamespace: kube-system
      - name: longhorn
        repositoryName: rancher-charts
        targetNamespace: longhorn-system
        createNamespace: true
        version: 104.2.0+up1.7.1
        installationNamespace: kube-system
    repositories:
      - name: rancher-charts
        url: https://charts.rancher.io/
  ...
```

- [SUSE Storage](#) は、Edge 3.4 リリースと互換性のあるバージョンにアップグレードする必要があります。つまり、`107.0.0+up1.9.1` にアップグレードする必要があります。
- `doc-example` の管理を担当する管理クラスタがローカル Git サーバのサポートなしで **エアギャップ** されており、Rancher セットアップが動作していることを前提としています。

アップグレード手順(36.1.6.2.3.2項「アップグレード手順」)に従います。

1. `release-3.4.0` タグから `suse-edge/fleet-example` リポジトリのクローンを作成します。

```
git clone -b release-3.4.0 https://github.com/suse-edge/fleet-examples.git
```

2. [Longhorn](#) アップグレードアーカイブが保存されるディレクトリを作成します。

```
mkdir archives
```

3. 目的の [Longhorn](#) チャートアーカイブバージョンを取得します。

```
# First add the Rancher Helm chart repository
```

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

# Pull the Longhorn 1.9.1 CRD archive
helm pull rancher-charts/longhorn-crd --version 107.0.0+up1.9.1

# Pull the Longhorn 1.9.1 chart archive
helm pull rancher-charts/longhorn --version 107.0.0+up1.9.1
```

4. archivesディレクトリ以外で、`suse-edge/fleet-examples`リリースタグ (<https://github.com/suse-edge/fleet-examples/releases/tag/release-3.4.0>) から `generate-chart-upgrade-data.sh` スクリプトをダウンロードします。

5. ディレクトリセットアップは次のようになるはずです。

```
.
├─ archives
│   ├── longhorn-107.0.0+up1.9.1.tgz
│   └─ longhorn-crd-107.0.0+up1.9.1.tgz
├─ fleet-examples
│   ...
│   ├── fleets
│   │   ├── day2
│   │   │   ├── ...
│   │   │   ├── eib-charts-upgrader
│   │   │   │   ├── base
│   │   │   │   │   ├── job.yaml
│   │   │   │   │   ├── kustomization.yaml
│   │   │   │   │   ├── patches
│   │   │   │   │   │   └─ job-patch.yaml
│   │   │   │   │   └─ rbac
│   │   │   │   │       ├── cluster-role-binding.yaml
│   │   │   │   │       ├── cluster-role.yaml
│   │   │   │   │       ├── kustomization.yaml
│   │   │   │   │       └─ sa.yaml
│   │   │   │   └─ secrets
│   │   │   │       ├── eib-charts-upgrader-script.yaml
│   │   │   │       └─ kustomization.yaml
│   │   │   └─ fleet.yaml
│   │   │       └─ kustomization.yaml
│   │   └─ ...
│   └─ ...
└─ generate-chart-upgrade-data.sh
```

6. `generate-chart-upgrade-data.sh` スクリプトを実行します。

```
# First make the script executable
chmod +x ./generate-chart-upgrade-data.sh

# Then execute the script
```

```
./generate-chart-upgrade-data.sh --archive-dir ./archives --fleet-path ./fleet-examples/fleets/day2/eib-charts-upgrader
```

スクリプト実行後のディレクトリ構造は次のようになるはずです。

```
.
├── archives
│   ├── longhorn-107.0.0+up1.9.1.tgz
│   └── longhorn-crd-107.0.0+up1.9.1.tgz
├── fleet-examples
│   └── ...
├── fleets
│   └── day2
│       ├── ...
│       └── eib-charts-upgrader
│           ├── base
│           │   ├── job.yaml
│           │   ├── kustomization.yaml
│           │   ├── patches
│           │   │   └── job-patch.yaml
│           │   ├── rbac
│           │   │   ├── cluster-role-binding.yaml
│           │   │   ├── cluster-role.yaml
│           │   │   ├── kustomization.yaml
│           │   │   └── sa.yaml
│           │   └── secrets
│           │       ├── eib-charts-upgrader-script.yaml
│           │       ├── kustomization.yaml
│           │       └── longhorn-VERSION.yaml - secret created by the generate-
│           │           chart-upgrade-data.sh script
│           │           └── longhorn-crd-VERSION.yaml - secret created by the
│           │               generate-chart-upgrade-data.sh script
│           ├── fleet.yaml
│           └── kustomization.yaml
│           └── ...
└── generate-chart-upgrade-data.sh
```

Gitで変更されたファイルは次のようになるはずです。

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   fleets/day2/eib-charts-upgrader/base/patches/job-patch.yaml
    modified:   fleets/day2/eib-charts-upgrader/base/secrets/kustomization.yaml

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    fleets/day2/eib-charts-upgrader/base/secrets/longhorn-VERSION.yaml
    fleets/day2/eib-charts-upgrader/base/secrets/longhorn-crd-VERSION.yaml
```

7. eib-charts-upgrader Fleetのバンドルを作成します。

- a. まず、Fleet自体に移動します。

```
cd ./fleet-examples/fleets/day2/eib-charts-upgrader
```

- b. 次に、targets.yamlファイルを作成します。

```
cat > targets.yaml <<EOF
targets:
- clusterName: doc-example
EOF
```

- c. 次に、fleet-cliバイナリを使用して、Fleetをバンドルに変換します。

```
fleet apply --compress --targets-file=targets.yaml -n fleet-default -o - eib-
charts-upgrade > bundle.yaml
```

- d. ここで、管理クラスタマシンにbundle.yamlを転送します。

8. Rancher UIを通じてバンドルをデプロイします。

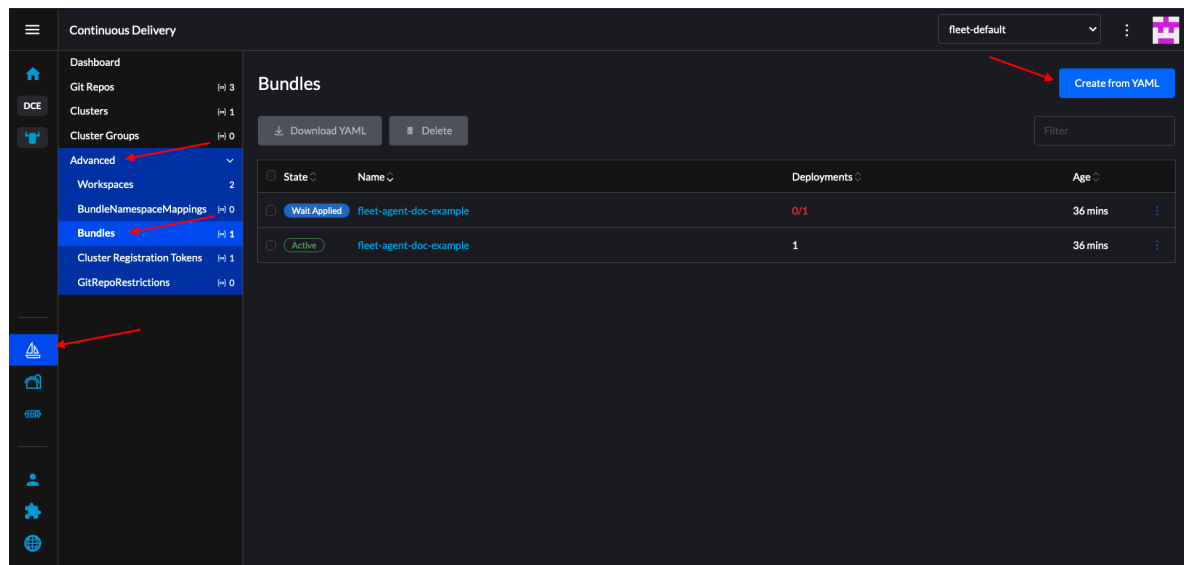


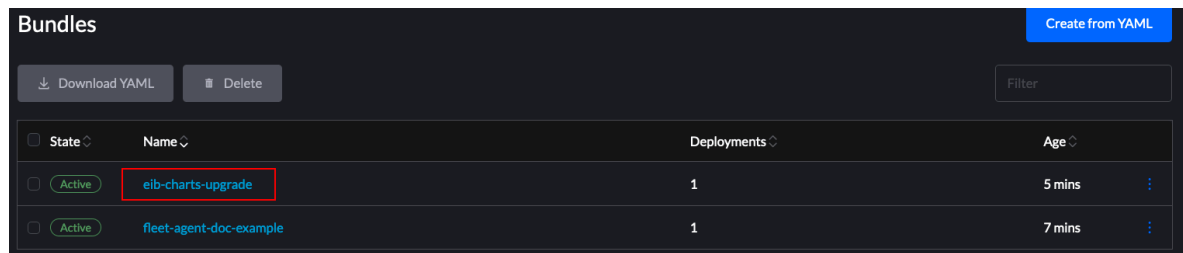
図 36.1: RANCHER UIを通じたバンドルのデプロイ

ここから、**【Read from File (ファイルから読み取り)】**を選択し、システムでbundle.yamlファイルを見つけます。

これにより、Rancher UI内にバンドルが自動入力されます。

【Create (作成)】を選択します。

9. 正常にデプロイされたら、バンドルは次のようになります。

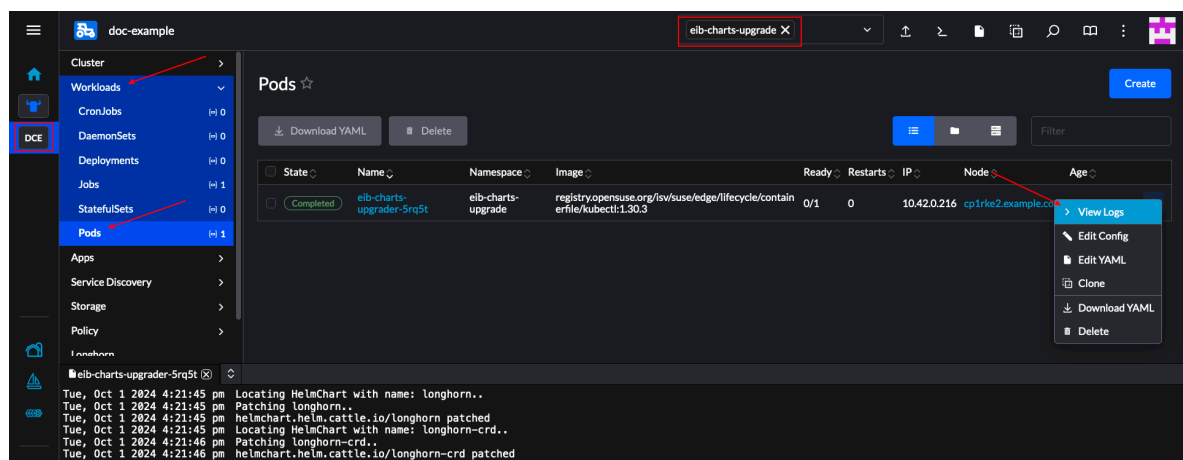


State	Name	Deployments	Age
Active	elb-charts-upgrade	1	5 mins
Active	fleet-agent-doc-example	1	7 mins

図 36.2: 正常にデプロイされたバンドル

バンドルのデプロイメントが成功した後で、アップグレードプロセスを監視するには次のようになります。

1. アップグレードPodのログを確認します。



The screenshot shows the 'elb-charts-upgrade' pod in the 'Completed' state. The logs for this pod are displayed at the bottom of the screen, showing the following sequence of events:

```

Tue, Oct 1 2024 4:21:45 pm Locating HelmChart with name: longhorn..
Tue, Oct 1 2024 4:21:45 pm Patching longhorn..
Tue, Oct 1 2024 4:21:45 pm helmchart.helm.cattle.io/longhorn patched
Tue, Oct 1 2024 4:21:45 pm Locating HelmChart with name: longhorn-crd..
Tue, Oct 1 2024 4:21:46 pm Patching longhorn-crd..
Tue, Oct 1 2024 4:21:46 pm helmchart.helm.cattle.io/longhorn-crd patched
  
```

2. ここで、helm-controllerによってアップグレードに作成されたPodのログを確認します。

- Pod名は次のテンプレートを使用します - `helm-install-longhorn-<random-suffix>`
- Podは、HelmChartリソースがデプロイされた名前空間にあります。この場合、この名前空間は`kube-system`です。

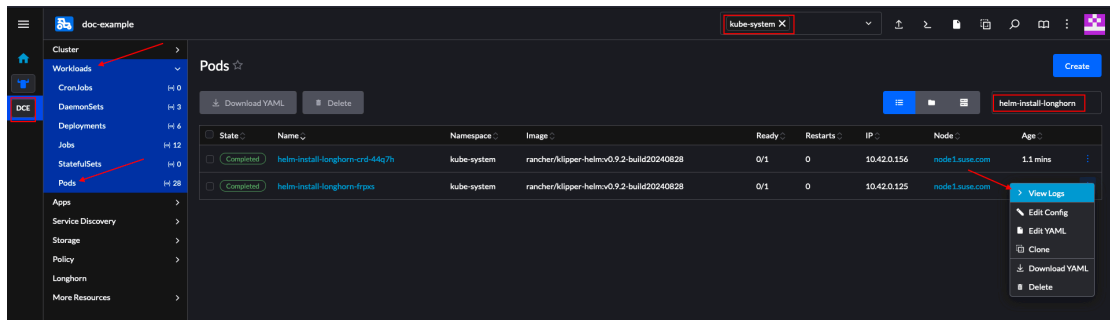


図 36.3: 正常にアップグレードされたLONGHORNチャートのログ

3. RancherのHelmChartsセクション([More Resources (その他のリソース)] → [HelmCharts])に移動して、HelmChartのバージョンが更新されていることを確認します。チャートがデプロイされたネームスペースを選択します。この例では、kube-systemです。

4. 最後に、Longhorn Podが実行されていることを確認します。

上記の検証後、Longhorn Helmチャートが107.0.0+up1.9.1バージョンにアップグレードされていると仮定しても問題ないでしょう。

36.1.6.2.3.4 サードパーティのGitOpsツールを使用したHelmチャートのアップグレード

ユーザがこのアップグレード手順をFleet以外のGitOpsワークフロー(Fluxなど)で実行したいユースケースが存在する場合があります。

アップグレード手順に必要なリソースを生成するには、generate-chart-upgrade-data.shスクリプトを使用して、ユーザが提供するデータをeib-charts-upgrader Fleetに入力することができます。この実行方法の詳細については、36.1.6.2.3.2項「アップグレード手順」を参照してください。

完全なセットアップ後、kustomize (<https://kustomize.io>) を使用して、クラスタにデプロイ可能な完全な動作ソリューションを生成することができます。

```
cd /foo/bar/fleets/day2/eib-charts-upgrader
```

```
kustomize build .
```

GitOpsワークフローにソリューションを含める場合は、fleet.yamlファイルを削除して、残ったものを有効なKustomizeセットアップとして使用できます。まず、generate-chart-upgrade-data.shスクリプトを実行し、アップグレードするHelmチャートのデータをKustomizeセットアップに読み込ませることを忘れないでください。

このワークフローの使用方法を理解するには[36.1.6.2.3.1項「概要」](#)と[36.1.6.2.3.2項「アップグレード手順」](#)を参照すると役立つでしょう。

VII SUSE Telco Cloudのドキュメント

- 37 SUSE Telco Cloud **373**
- 38 コンセプトとアーキテクチャ **374**
- 39 要件と前提 **379**
- 40 管理クラスタの設定 **390**
- 41 通信機能の設定 **423**
- 42 完全に自動化されたダイレクトネットワークプロビジョニング **470**
- 43 ライフサイクルのアクション **527**

SUSE Telco Cloudのドキュメントはこちらでご確認ください。

37 SUSE Telco Cloud

SUSE Telco Cloud (旧称: SUSE Edge for Telco)は、通信事業者向けに最適化されたコンピューティングプラットフォームであり、通信事業者や通信ネットワークベンダはそのネットワークを刷新し、その最新化を加速できます。

SUSE Telco Cloudは、CNFをホストするための完全な通信事業者対応のクラウドネイティブなスタックであり、すべての通信ドメイン(Packet Core、RAN、OSS、BSS)をカバーします。

- エッジスタックの複雑な設定を通信事業者の規模で自動的にゼロタッチでロールアウトし、ライフサイクルを管理します。
- 通信事業者に固有の設定とワークロードを使用して、通信事業者グレードのハードウェアの品質を継続的に保証します。
- エッジ専用に設計されたコンポーネントで構成されているため、フットプリントが小さく、ワットパフォーマンスが高くなっています。
- ベンダに依存しないAPIを備え、100%オープンソースであるため、柔軟なプラットフォーム戦略を維持します。

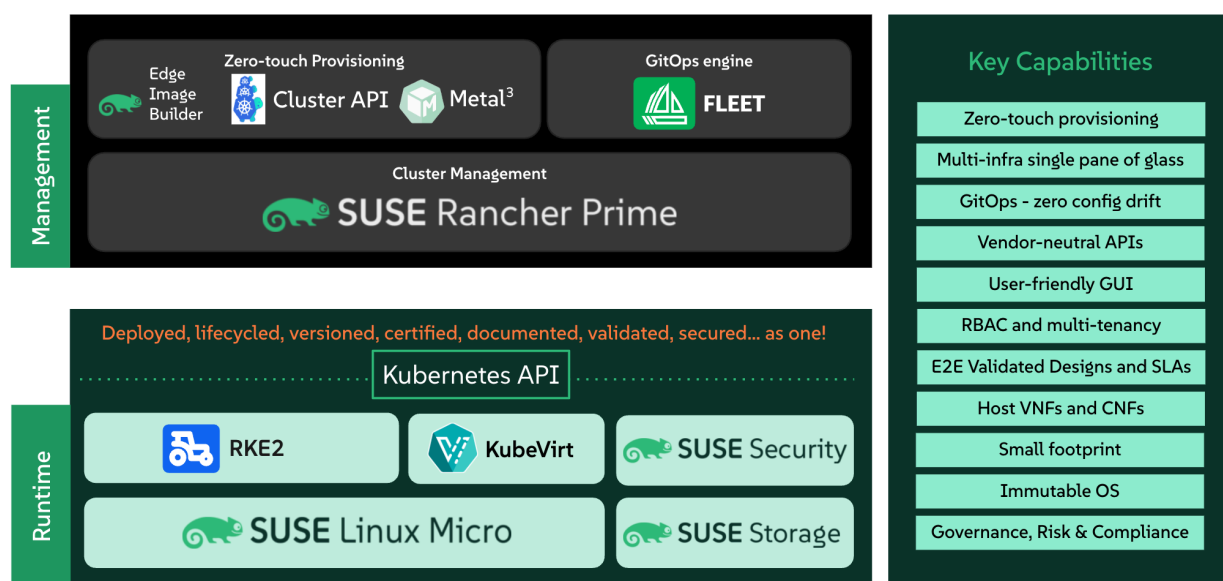
38 コンセプトとアーキテクチャ

SUSE Telco Cloudは、クラウドネイティブな最新の通信事業者向けアプリケーションをコアからエッジまで大規模にホストするために設計されたプラットフォームです。

このページでは、SUSE Telco Cloudで使用されるアーキテクチャとコンポーネントについて説明します。

38.1 SUSE Telco Cloudアーキテクチャ

次の図は、SUSE Telco Cloudの高レベルアーキテクチャを示しています。



38.2 コンポーネント

異なる2つのブロックがあります。管理スタックとランタイムスタックです。

- **管理スタック:** SUSE Telco Cloud内でランタイムスタックのプロビジョニングとライフサイクルを管理するために使用される部分です。次のコンポーネントが含まれます。
 - Rancher (第5章「[Rancher](#)」)を使用した、パブリッククラウド環境とプライベートクラウド環境のマルチクラスタ管理
 - Metal3 (第10章「[Metal³](#)」)、MetalLB (第18章「[MetalLB](#)」)、およびCAPI (Cluster API)インフラストラクチャプロバイダを使用したベアメタルサポート

- 包括的なテナント分離とIDP (IDプロバイダ)の統合
- サードパーティ統合と拡張機能の大規模なマーケットプレイス
- ベンダに依存しないAPIと充実したプロバイダエコシステム
- SUSE Linux Microのトランザクション更新の制御
- GitリポジトリとFleet (第8章 「Fleet」)を使用してクラスタのライフサイクルを管理するGitOpsエンジン
- **ランタイムスタック**: SUSE Telco Cloud内でワークロードを実行するために使用される部分です。
 - RKE2 (第15章 「RKE2」)。エッジ環境およびコンプライアンス重視の通信環境向けに最適化された、セキュリティが強化された軽量のKubernetesディストリビューションとして機能します。
 - SUSE Security (第17章 「SUSE Security」)。イメージの脆弱性スキャン、ディープパケットインスペクション、クラスタ内の自動トラフィック制御などのセキュリティ機能を実現します。
 - ブロックストレージとSUSE Storage (第16章 「SUSE Storage」)。クラウドネイティブのストレージソリューションをシンプルかつ簡単に使用できます。
 - SUSE Linux Micro (第9章 「SUSE Linux Micro」)で最適化されたオペレーティングシステム。コンテナ運用のための、安全、軽量でイミュータブルな(トランザクショナルファイルシステムを備えた) OSを実現します。SUSE Linux Microは、AArch64アーキテクチャとAMD64/Intel 64アーキテクチャで利用でき、通信事業者およびエッジのユースケース向けにリアルタイムカーネルもサポートしています。

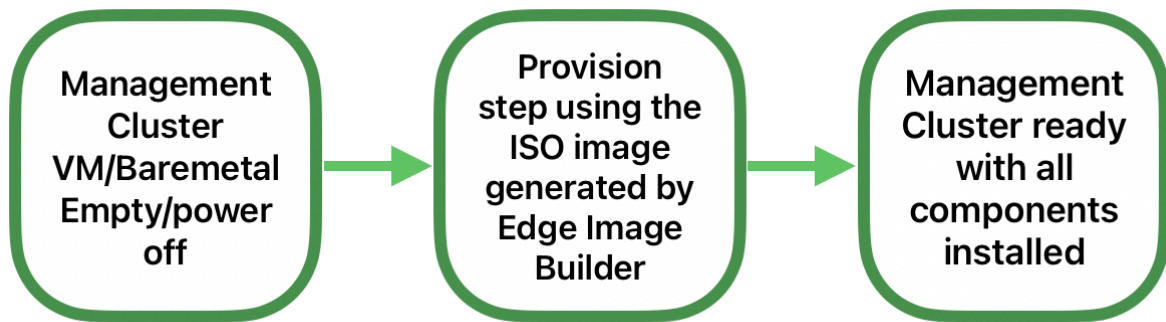
38.3 デプロイメントフローの例

管理コンポーネントとランタイムコンポーネントの関係を理解できるように、以下にワークフローの概要の例を示します。

ダイレクトネットワークプロビジョニングは、すべてのコンポーネントを事前設定した状態で新しいダウンストリームクラスタをデプロイできるワークフローであり、手動操作なしですぐにワークロードを実行できます。

38.3.1 例1: すべてのコンポーネントがインストールされた新しい管理クラスタをデプロイする

Edge Image Builder (第11章「Edge Image Builder」)を使用して、管理スタックが含まれる新しいISOイメージを作成します。その後、このISOイメージを使用して、新しい管理クラスタをVMまたはベアメタルにインストールできます。



注記

新しい管理クラスタをデプロイする方法の詳細については、SUSE Telco Cloudの管理クラスタに関するガイド(第40章「管理クラスタの設定」)を参照してください。



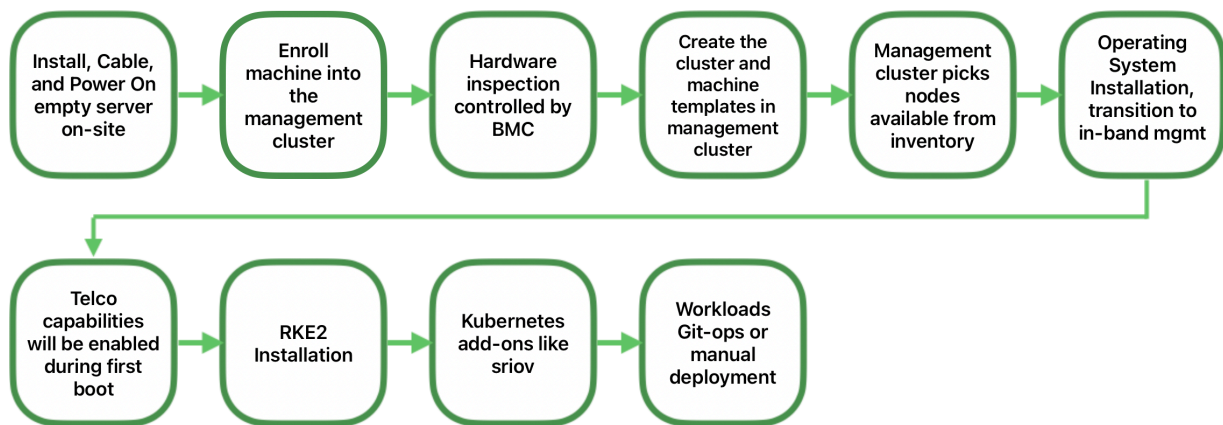
注記

Edge Image Builderの使用方法については、Edge Image Builderのガイド(第3章「Edge Image Builderを使用したスタンドアロンクラスタ」)を参照してください。

38.3.2 例2: 通信事業者プロファイルを使用してシングルノードのダウストリームクラスタをデプロイして通信ワークロードを実行可能にする

管理クラスタが稼働したら、その管理クラスタを使用して、ダイレクトネットワークプロビジョニングワークフローにより、すべての通信機能が有効化および設定された状態でシングルノードのダウストリームクラスタをデプロイすることができます。

次の図に、これをデプロイするワークフローの概要を示します。



注記

ダウンストリームクラスタのデプロイ方法の詳細については、SUSE Telco Cloudの自動化されたプロビジョニングに関するガイド(第42章「完全に自動化されたダイレクトネットワークプロビジョニング」)を参照してください。



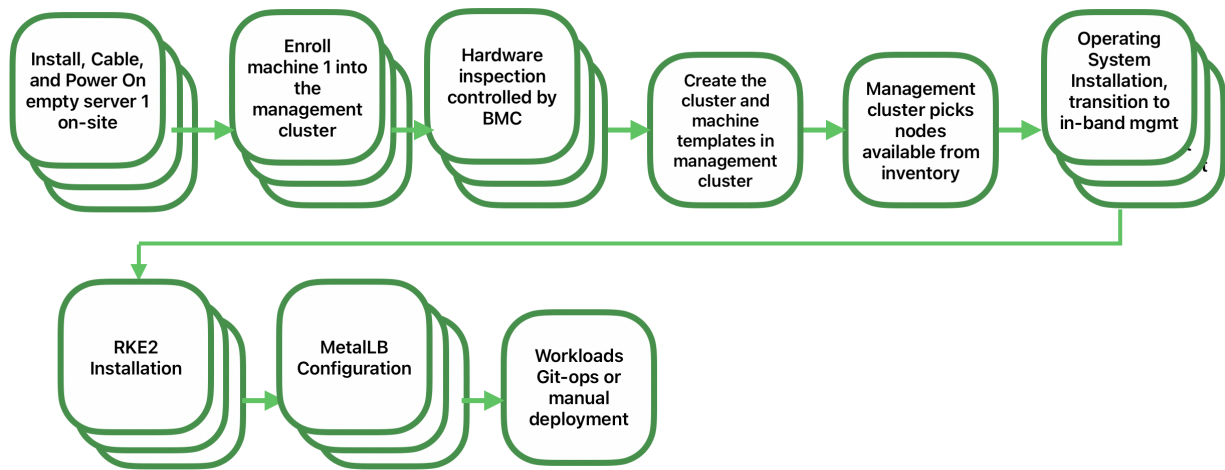
注記

通信機能の詳細については、SUSE Telco Cloudの通信機能に関するガイド(第41章「通信機能の設定」)を参照してください。

38.3.3 例3: MetalLBをロードバランサとして使用して高可用性ダウンストリームクラスタをデプロイする

管理クラスタが稼働したら、その管理クラスタを使用して、ダイレクトネットワークプロビジョニングワークフローにより、MetalLBをロードバランサとして使用する高可用性ダウンストリームクラスタをデプロイすることができます。

次の図に、これをデプロイするワークフローの概要を示します。



注記

ダウンストリームクラスタのデプロイ方法の詳細については、SUSE Telco Cloudの自動化されたプロビジョニングに関するガイド([第42章「完全に自動化されたダイレクトネットワークプロビジョニング」](#))を参照してください。



注記

MetalLBの詳細については、[第18章「MetalLB」](#)を参照してください。

39 要件と前提

39.1 ハードウェア

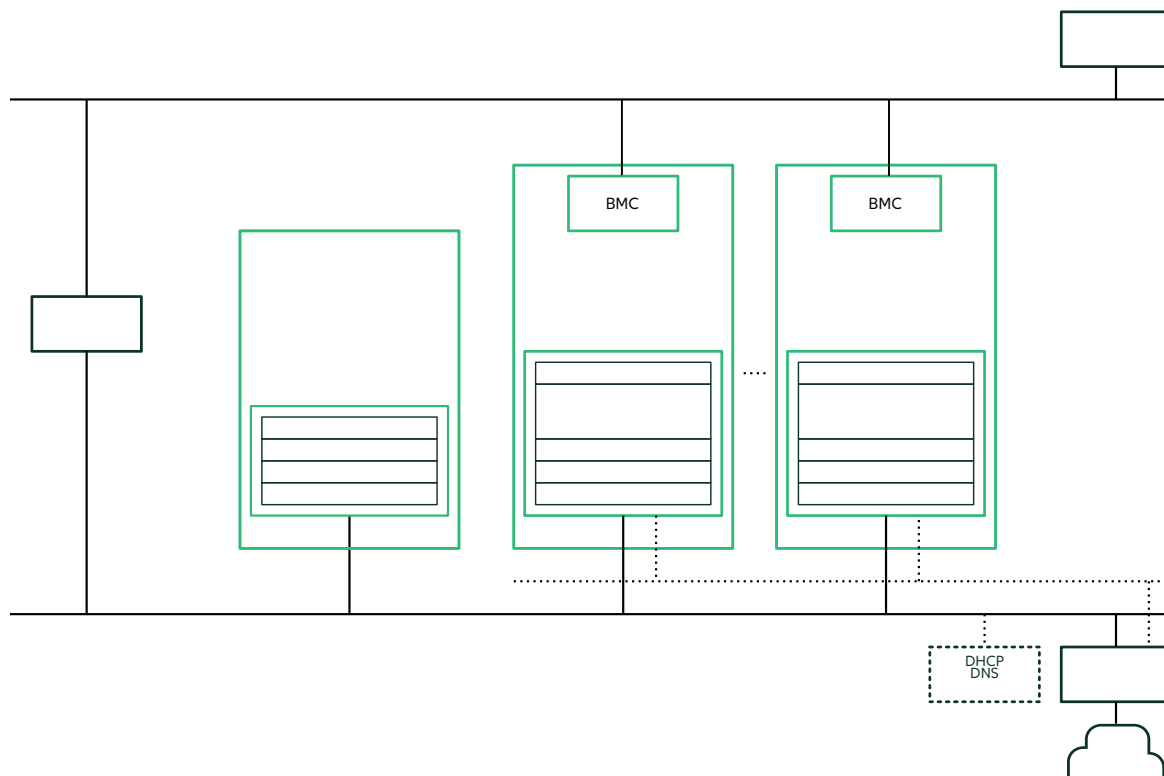
SUSE Telco Cloudのハードウェア要件は次のとおりです。

- **管理クラスタ:** 管理クラスタには、SUSE Linux Micro、RKE2、SUSE Rancher Prime、Metal3などのコンポーネントが含まれ、管理クラスタを使用して複数のダウンストリームクラスタを管理します。管理するダウンストリームクラスタの数によっては、サーバのハードウェア要件は変わる場合があります。
 - サーバ(VMまたはベアメタル)の最小要件は次のとおりです。
 - RAM: 8GB以上(16GB以上を推奨)
 - CPU: 2個以上(4個以上を推奨)
- **ダウンストリームクラスタ:** ダウンストリームクラスタは、通信ワークロードを実行するためにデプロイされるクラスタです。SR-IOV、CPUパフォーマンス最適化などの特定の通信機能を有効にするには、固有の要件が必要になります。
 - SR-IOV: VF (仮想機能)をCNF/VNFにパススルーモードでアタッチするには、NICがSR-IOVをサポートしていて、BIOSでVT-d/AMD-Viが有効化されている必要があります。
 - CPUプロセッサ: 特定の通信ワークロードを実行するには、こちらの参照表([第41章「通信機能の設定」](#))に記載されているほとんどの機能を利用できるようにCPUプロセッサモデルを適応させる必要があります。
 - 仮想メディアでインストールするためのファームウェア要件:

サーバハードウェア	BMCモデル	管理
Dell製ハードウェア	第15世代	iDRAC9
Supermicro製ハードウェア	01.00.25	Supermicro SMC - redfish
HPE製ハードウェア	1.50	iLO6

39.2 ネットワーク

ネットワークアーキテクチャの参考として、次の図に、通信事業者環境の一般的なネットワークアーキテクチャを示します。



このネットワークアーキテクチャは次のコンポーネントに基づきます。

- **管理ネットワーク:** このネットワークは、ダウンストリームクラスタノードの管理や帯域外管理に使用されます。通常は独立した管理スイッチに接続しますが、同じサービススイッチに接続し、VLANを使ってトラフィックを分離することもできます。
- **コントロールプレーンネットワーク:** このネットワークは、ダウンストリームクラスタノードと、そこで実行されているサービスとの間の通信に使用されます。また、ダウンストリームクラスタノードと外部サービス(DHCPサーバやDNSサーバなど)との間の通信にも使用されます。接続環境では、スイッチやルータでインターネット経由のトラフィックを処理できる場合もあります。
- **その他のネットワーク:** 場合によっては、特定の目的に合わせてノードを他のネットワークに接続できます。



注記

ダイレクトネットワークプロビジョニングワークフローを使用するには、管理クラスタがダウンストリームクラスタサーバのBaseboard Management Controller (BMC)とネットワークで接続されていて、ホストの準備とプロビジョニングを自動化できる必要があります。

39.3 ポート要件

SUSE Telco Cloudのデプロイメントが適切に動作するには、管理ノードおよびダウンストリームKubernetesクラスタノードにアクセス可能な複数のポートが必要です。



注記

正確なリストは、デプロイされているオプションコンポーネントと選択したデプロイメントオプション(CNIプラグインなど)によって異なります。

39.3.1 管理ノード

次の表は、管理クラスタを実行しているノードで開いているポートをリストしています。



注記

CNIプラグイン関連のポートについては、CNI固有のポート要件(39.3.3項「CNI固有のポート要件」)を参照してください。

表 39.1: 管理ノードの受信ネットワークルール

プロトコル	ポート	ソース	説明
TCP	22	SSHアクセスを必要とする任意のソース	管理クラスタノードへのSSHアクセス
TCP	80	外部TLS終端を行うロードバランサ/プロキシ	外部TLS終端が使用される場合はRancher UI/API

プロトコル	ポート	ソース	説明
TCP	443	Rancher UI/APIへのTLSアクセスを必要とする任意のソース	Rancherエージェント、Rancher UI/API
TCP	2379	RKE2 (管理クラスタ) サーバノード	<u>etcd</u> クライアントポート
TCP	2380	RKE2 (管理クラスタ) サーバノード	<u>etcd</u> ピアポート
TCP	6180	この公開ポート (非TLS)からIPA ⁽²⁾ ramdiskイメージをプルするようにMetal3/ <u>ironic</u> によって以前に指示された任意のBMC ⁽¹⁾	仮想メディアベースのブート用のIPA ⁽²⁾ ISOイメージを提供するIronic <u>httpd</u> 非TLS Webサーバ。 このポートが有効な場合、機能的に同等だがTLS対応のポート(下記参照)は開かれませんか
TCP	6185	この公開ポート(TLS)からIPA ⁽²⁾ ramdiskイメージをプルするようにMetal3/ <u>ironic</u> によって以前に指示された任意のBMC ⁽¹⁾	仮想メディアベースのブート用のIPA ⁽²⁾ ISOイメージを提供するIronic <u>httpd</u> TLS対応Webサーバ。 このポートが有効な場合、機能的には同等だがTLSが無効なポート(上記参照)は開かれませんか
TCP	6385	「登録済み」 <u>BareMetalHost</u> インスタンスにデプロイされ実行されている任意	Ironic API

プロトコル	ポート	ソース	説明
		のMetal3/ironic IPA ⁽¹⁾ ramdiskイメージ	
TCP	6443	任意の管理クラスター ノード、(管理クラスター)外部の任意の Kubernetesクライアント	Kubernetes API
TCP	6545	任意の管理クラスター ノード	OCI準拠レジストリ (Hauler)からアー ティファクトをプル します
TCP	9345	RKE2サーバとエー ジェントノード(管理 クラスター)	ノード登録用のRKE2 スーパーバイザAPI (すべてのRKE2サー バノードで開かれた ポート)
TCP	10250	任意の管理クラスター ノード	<u>kubelet</u> メトリクス
TCP/UDP/SCTP	30000~32767	<u>spec.type:</u> <u>NodePort</u> また は <u>spec.type:</u> <u>LoadBalancer</u> <u>Service APIオブ</u> <u>ジェクト</u> (https:// kubernetes.io/docs/ concepts/services- networking/service/ #publishing- services-service- types) を介してプ ライマリネットワー クで公開されている	使用可能な <u>NodePort</u> ポート範囲

プロトコル	ポート	ソース	説明
		サービスにアクセスする、(管理クラスター)外部の任意のソース	

(1) BMC: ベースボード管理コントローラ

(2) IPA: Ironic Python Agent

39.3.2 ダウンストリームノード

SUSE Telco Cloudでは、任意の(ダウンストリーム)サーバが実行中のダウンストリームKubernetesクラスタの一部となる前(または、それ自体が単一ノードのダウンストリームKubernetesクラスタとして実行される前)に、[BaremetalHostのプロビジョニング状態](https://github.com/metal3-io/baremetal-operator/blob/main/docs/baremetalhost-states.md) (<https://github.com/metal3-io/baremetal-operator/blob/main/docs/baremetalhost-states.md>) のいくつかを経由する必要があります。

- 宣言されたばかりのダウンストリームサーバのベースボード管理コントローラ(BMC)は、帯域外ネットワーク経由でアクセスできる必要があります。BMCは(管理クラスタ上で実行されているIronicサービスから)以下の初期実行手順について指示されます。
 1. BMCで提供される[仮想メディア](#)内の指定されたIPA ramdiskイメージをプルしてロードします。
 2. サーバの電源を入れます。

以下のポートは、BMCから公開される予定です(実際のハードウェアによって異なる可能性があります)。

表 39.2: ベースボード管理コントローラの受信ネットワークルール

プロトコル	ポート	ソース	説明
TCP	80	Ironic Conductor (管理クラスタから)	Redfish APIアクセス (HTTP)
TCP	443	Ironic Conductor (管理クラスタから)	Redfish APIアクセス (HTTPS)

- BMC 仮想メディア にロードされたIPA ramdiskイメージを使用してダウンストリームサーバイメージを起動すると、ハードウェア検査フェーズが開始されます。以下の表は、実行中のIPA ramdiskイメージによって公開されるポートをリストしています。

表 39.3: ダウンストリームノードの受信ネットワークルール - Metal3/Ironicプロビジョニングフェーズ

プロトコル	ポート	ソース	説明
TCP	22	IPA ramdiskイメージへのSSHアクセスを必要とする任意のソース	検査対象のダウンストリームクラスターノードへのSSHアクセス
TCP	9999	Ironic Conductor (管理クラスターから)	実行中のramdiskイメージに対するIronicコマンド

- ベアメタルホストが適切にプロビジョニングされ、ダウンストリームKubernetesクラスターに参加すると、以下のポートが公開されます。



注記

CNIプラグイン関連のポートについては、CNI固有のポート要件(39.3.3項「CNI固有のポート要件」)を参照してください。

表 39.4: ダウンストリームノードの受信ネットワークルール

プロトコル	ポート	ソース	説明
TCP	22	SSHアクセスを必要とする任意のソース	ダウンストリームクラスターノードへのSSHアクセス
TCP	80	外部TLS終端を行うロードバランサ/プロキシ	外部TLS終端が使用される場合はRancher UI/API
TCP	443	Rancher UI/APIへのTLSアクセスを必要とする任意のソース	Rancherエージェント、Rancher UI/API

プロトコル	ポート	ソース	説明
TCP	2379	RKE2 (ダウンストリームクラスタ)サーバノード	<u>etcd</u> クライアントポート
TCP	2380	RKE2 (ダウンストリームクラスタ)サーバノード	<u>etcd</u> ピアポート
TCP	6443	任意のダウンストリームクラスタノード、(ダウンストリームクラスタ)外部の任意のKubernetesクライアント	Kubernetes API
TCP	9345	RKE2サーバおよびエージェントノード (ダウンストリームクラスタ)	ノード登録用のRKE2スーパーバイザAPI (すべてのRKE2サーバノードで開かれたポート)
TCP	10250	任意のダウンストリームクラスタノード	<u>kubelet</u> メトリクス
TCP	10255	任意のダウンストリームクラスタノード	<u>kubelet</u> 読み取り専用アクセス
TCP/UDP/SCTP	30000～32767	spec.type: NodePortまたはspec.type: LoadBalancer Service APIオブジェクト (https://kubernetes.io/docs/concepts/services-networking/service/)	使用可能な <u>NodePort</u> ポート範囲

プロトコル	ポート	ソース	説明
		#publishing-services-service-types を介してプライマリネットワークで公開されているサービスにアクセスする、(ダウストリームクラスター)外部の任意のソース	

39.3.3 CNI固有のポート要件

サポートされている各CNIバリエーションには、独自のポート要件が伴います。詳細については、RKE2ドキュメントの「[CNI Specific Inbound Network Rules \(CNI固有の受信ネットワークルール\)](https://docs.rke2.io/install/requirements#cni-specific-inbound-network-rules) (<https://docs.rke2.io/install/requirements#cni-specific-inbound-network-rules>)」を参照してください。

`cilium`がデフォルト/プライマリCNIプラグインとして設定されている場合、`cilium-operator`ワークロードが、デプロイされるKubernetesクラスターの外部にメトリクスを公開するように設定されると、以下のTCPポートが付加的に公開されます。これにより、そのKubernetesクラスター外部で実行されている外部Prometheusサービンスタンスが引き続きこれらのメトリクスを収集できるようになります。



注記

これは、`rke2-cilium` Helmチャートを介して`cilium`をデプロイする際のデフォルトオプションです。

表 39.5: 管理/ダウストリームノード用の受信ネットワークルール - `cilium-operator`からの外部メトリクス公開が有効

プロトコル	ポート	ソース	説明
TCP	9963	(Kubernetesクラスター)外部のメトリクスコレクタ	<code>cilium-operator</code> メトリクス公開

39.4 サービス(DHCP、DNSなど)

デプロイ先の環境の種類によっては、DHCP、DNSなどの外部サービスが必要な場合があります。

- **接続環境:** この場合、ノードはインターネットに接続され(L3ルーティングプロトコルを使用)、外部サービスはお客様が提供します。
- **非接続/エアギャップ環境:** この場合、ノードはインターネットにIPで接続されないため、サービスを追加して、ダイレクトネットワークプロビジョニングワークフローに必要なコンテンツをローカルにミラーリングする必要があります。
- **ファイルサーバ:** ファイルサーバは、ダイレクトネットワークプロビジョニングワークフローの中で、ダウンストリームクラスターノードにプロビジョニングするOSイメージを保存するために使用されます。Meta13 HelmチャートでメディアサーバをデプロイしてOSイメージを保存できます。次のセクション([注記](#))を確認してください。ただし、既存のローカルWebサーバを使用することもできます。

39.5 systemdサービスの無効化

通信ワークロードの場合、ノード上で実行されているワークロードのパフォーマンス(レイテンシ)に影響を及ぼさないように、ノード上で実行されている一部のサービスを無効にしたり、適切に設定することが重要です。

- rebootmgrは、システムに保留中の更新がある場合の再起動方針を設定できるサービスです。通信ワークロードでは、システムによってスケジュールされた更新がある場合、rebootmgrサービスを無効にするか正しく設定してノードの再起動を回避し、ノードで実行中のサービスへの影響を避けることが非常に重要です。



注記

rebootmgrの詳細については、rebootmgrのGitHubリポジトリ (<https://github.com/SUSE/rebootmgr>) [🔗](#)を参照してください。

次のコマンドを実行して、使用する方針を検証します。

```
cat /etc/rebootmgr.conf
[rebootmgr]
window-start=03:30
window-duration=1h30m
```

```
strategy=best-effort
lock-group=default
```

また、次のコマンドを実行すると無効にすることができます。

```
sed -i 's/strategy=best-effort/strategy=off/g' /etc/rebootmgr.conf
```

または、`rebootmgrctl`コマンドを次のように使用できます。

```
rebootmgrctl strategy off
```



注記

`rebootmgr`の方針を設定するこの設定は、ダイレクトネットワークプロビジョニングワークフローを使用して自動化できます。詳細については、自動化されたプロビジョニングに関するドキュメント([第42章「完全に自動化されたダイレクトネットワークプロビジョニング」](#))を確認してください。

- `transactional-update`は、システムによって制御される自動更新を可能にするサービスです。通信ワークロードの場合、ノードで実行中のサービスに影響を及ぼさないように、自動更新を無効にすることが重要です。

自動更新を無効にするには、次のコマンドを実行できます。

```
systemctl --now disable transactional-update.timer
systemctl --now disable transactional-update-cleanup.timer
```

- `fstrim`は、ファイルシステムを毎週自動的にトリミングできるサービスです。通信ワークロードでは、ノードで実行中のサービスに影響を及ぼさないように、自動トリミングを無効にすることが重要です。

自動トリミングを無効にするには、次のコマンドを実行できます。

```
systemctl --now disable fstrim.timer
```

40 管理クラスタの設定

40.1 はじめに

管理クラスタは、SUSE Telco Cloud内でランタイムスタックのプロビジョニングとライフサイクルを管理するために使用されます。技術的観点からは、管理クラスタには次のコンポーネントが含まれています。

- SUSE Linux Micro (OS)。ユースケースに応じて、ネットワーキング、ストレージ、ユーザ、カーネル引数などの一部の設定をカスタマイズできます。
- RKE2 (Kubernetesクラスタ)。ユースケースに応じて、Multus、Cilium、Calicoなどの特定のCNIプラグインを使用するように設定できます。
- Rancher (管理プラットフォーム)。クラスタのライフサイクルを管理します。
- Metal3 (コンポーネント)。ベアメタルノードのライフサイクルを管理します。
- CAPI (コンポーネント)。Kubernetesクラスタ(ダウンストリームクラスタ)のライフサイクルを管理します。RKE2 CAPI Providerは、RKE2クラスタのライフサイクルを管理するために使用されます。

上記のコンポーネントをすべて使用すると、管理クラスタは、宣言型アプローチを使用してインフラストラクチャやアプリケーションを管理し、ダウンストリームクラスタのライフサイクルを管理できます。



注記

SUSE Linux Microの詳細については、「SUSE Linux Micro」(第9章「SUSE Linux Micro」)を参照してください。

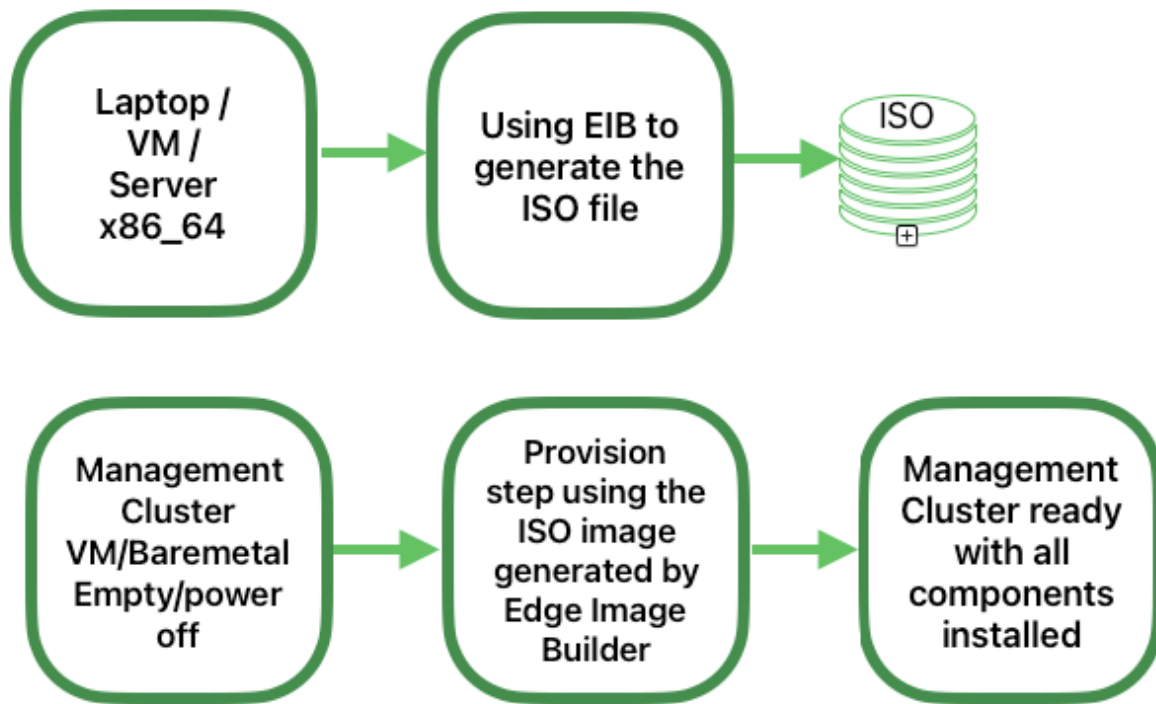
RKE2の詳細については、「RKE2」(第15章「RKE2」)を参照してください。

Rancherの詳細については、「Rancher」(第5章「Rancher」)を参照してください。

Metal3の詳細については、「Metal3」(第10章「Metal³」)を参照してください。

40.2 管理クラスタの設定手順

管理クラスタを設定するには、次の手順が必要です(シングルノードを使用)。



宣言型アプローチを使用して管理クラスタを設定するための主な手順は次のとおりです。

1. **接続環境のイメージの準備**([40.3項「接続環境用のイメージの準備」](#)): 最初の手順では、接続環境で使用する必要がある設定をすべて含むマニフェストとファイルを準備します。
 - **接続環境のディレクトリ構造**([40.3.1項「ディレクトリ構造」](#)): この手順では、Edge Image Builderで使用するディレクトリ構造を作成し、設定ファイルとイメージそのものを保存します。
 - **管理クラスタ定義ファイル**([40.3.2項「管理クラスタ定義ファイル」](#)): `mgmt-cluster.yaml`ファイルが管理クラスタのメイン定義ファイルです。このファイルには、作成するイメージに関する次の情報が含まれています。
 - **イメージ情報**: ゴールデンイメージを使用して作成するイメージに関する情報。
 - **オペレーティングシステム**: イメージで使用するオペレーティングシステムの設定。
 - **Kubernetes**: Helmチャートとリポジトリ、Kubernetesのバージョン、ネットワーク設定、およびクラスタで使用するノード。

- Customフォルダ(40.3.3項「Customフォルダ」): customフォルダには設定ファイルとスクリプトが含まれ、Edge Image Builderはこれらを使用して完全に機能する管理クラスタをデプロイします。
 - ファイル: 管理クラスタが使用する設定ファイルが含まれています。
 - スクリプト: 管理クラスタが使用するスクリプトが含まれています。
- Kubernetesフォルダ(40.3.4項「Kubernetesフォルダ」): kubernetesフォルダには、管理クラスタが使用する設定ファイルが含まれています。
 - Manifests: 管理クラスタが使用するマニフェストが含まれています。
 - Helm: 管理クラスタによって使用されるHelm値ファイルが含まれます。
 - Config: 管理クラスタが使用する設定ファイルが含まれています。
- Networkフォルダ(40.3.5項「ネットワーキングフォルダ」): networkフォルダには、管理クラスタノードが使用するネットワーク設定ファイルが含まれています。

2. エアギャップ環境でのイメージの準備(40.4項「エアギャップ環境のイメージの準備」): この手順では、エアギャップシナリオで使用するマニフェストとファイルを準備する際の相違点を示します。

- 定義ファイルの変更(40.4.1項「定義ファイルの変更」): mgmt-cluster.yamlファイルを変更してembeddedArtifactRegistryセクションを含め、imagesフィールドに、EIBの出力イメージに組み込むすべてのコンテナイメージを設定する必要があります。
- customフォルダの変更(40.4.2項「カスタムフォルダの変更」): customフォルダを変更し、管理クラスタをエアギャップ環境で実行するために必要なリソースを含める必要があります。
 - 登録スクリプト: エアギャップ環境を使用する場合、custom/scripts/99-register.shスクリプトを削除する必要があります。
- helm値フォルダ(40.4.3項「Helm値フォルダの変更」)での変更: helm/valuesフォルダはエアギャップ環境で管理クラスタを実行するために必要な設定を含むように変更する必要があります。

3. **イメージの作成**(40.5項「**イメージの作成**」): この手順では、Edge Image Builderツールを使用してイメージを作成します(接続環境とエアギャップ環境の両方が対象です)。ご使用のシステムでEdge Image Builderツールを実行するための前提条件(第11章「Edge Image Builder」)を確認してください。
4. **管理クラスタのプロビジョニング**(40.6項「**管理クラスタのプロビジョニング**」): この手順では、前の手順で作成したイメージを使用して管理クラスタをプロビジョニングする方法について説明します(接続シナリオとエアギャップシナリオの両方が対象です)。この手順は、ラップトップ、サーバ、VM、またはUSBポートを搭載した他の任意のAMD64/Intel 64システムを使用して実行できます。



注記

Edge Image Builderの詳細については、「Edge Image Builder」(第11章「Edge Image Builder」)およびEdge Image Builderのクイックスタート(第3章「Edge Image Builderを使用したスタンドアロンクラスタ」)を参照してください。

40.3 接続環境用のイメージの準備

Edge Image Builderは、管理クラスタのイメージを作成するために使用されます。このドキュメントでは、管理クラスタのセットアップに必要な最小設定について説明します。

Edge Image Builderは、コンテナ内で実行されるため、Podman (<https://podman.io>)  や Rancher Desktop (<https://rancherdesktop.io>)  などのコンテナランタイムが必要です。このガイドでは、Podmanが使用できることを前提としています。

また、高可用性管理クラスタをデプロイするための前提条件として、ネットワークで次の3つのIPを予約する必要があります。

- apiVIP (API VIPアドレス用(Kubernetes APIサーバへのアクセスに使用))。
- ingressVIP (Ingress VIPアドレス(Rancher UIなどで使用))。
- metal3VIP (Metal3 VIPアドレス用)。

40.3.1 ディレクトリ構造

EIBを実行する場合、ディレクトリはホストからマウントされます。したがって、最初に行う手順は、EIBが設定ファイルとイメージ自体を保存するために使用するディレクトリ構造を作成することです。このディレクトリの構造は次のとおりです。

```

eib
├── mgmt-cluster.yaml
├── network
│   └── mgmt-cluster-node1.yaml
├── os-files
│   ├── var
│   │   ├── lib
│   │   │   ├── rancher
│   │   │   │   ├── rke2
│   │   │   │   │   ├── server
│   │   │   │   │   │   └── manifests
│   │   │   │   │   │       └── rke2-ingress-config.yaml
│   │   │   └── rke2-ingress-config.yaml
│   └── kubernetes
│       ├── manifests
│       │   ├── neuvector-namespace.yaml
│       │   ├── ingress-l2-adv.yaml
│       │   └── ingress-ippool.yaml
│       ├── helm
│       │   ├── values
│       │   │   ├── rancher.yaml
│       │   │   ├── neuvector.yaml
│       │   │   ├── longhorn.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

```



注記

SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.isoイメージは、[SUSE Customer Center \(https://scc.suse.com/\)](https://scc.suse.com/) または [SUSEダウンロードページ \(https://www.suse.com/download/sle-micro/\)](https://www.suse.com/download/sle-micro/) からダウンロードする必要があり、また、base-images フォルダの下に配置されている必要があります。

イメージのSHA256チェックサムを確認し、イメージが改ざんされていないことを確認する必要があります。このチェックサムは、イメージをダウンロードした場所と同じ場所にあります。

ディレクトリ構造の例は、SUSE Edge GitHubリポジトリの「telco-examples」フォルダ (<https://github.com/suse-edge/atip>) にあります。

40.3.2 管理クラスタ定義ファイル

`mgmt-cluster.yaml` ファイルは管理クラスタのメイン定義ファイルで、次の情報が含まれます。

```
apiVersion: 1.3
image:
  imageType: iso
  arch: x86_64
  baseImage: SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso
  outputImageName: eib-mgmt-cluster-image.iso
operatingSystem:
  isoConfiguration:
    installDevice: /dev/sda
  users:
    - username: root
      encryptedPassword: $ROOT_PASSWORD
  packages:
    packageList:
      - jq
      - open-iscsi
    sccRegistrationCode: $SCC_REGISTRATION_CODE
kubernetes:
  version: v1.33.3+rke2r1
helm:
  charts:
    - name: cert-manager
      repositoryName: jetstack
      version: 1.18.2
      targetNamespace: cert-manager
      valuesFile: certmanager.yaml
      createNamespace: true
      installationNamespace: kube-system
    - name: longhorn-crd
      version: 107.0.0+up1.9.1
      repositoryName: rancher-charts
      targetNamespace: longhorn-system
      createNamespace: true
      installationNamespace: kube-system
    - name: longhorn
      version: 107.0.0+up1.9.1
      repositoryName: rancher-charts
      targetNamespace: longhorn-system
      createNamespace: true
```

```

    installationNamespace: kube-system
    valuesFile: longhorn.yaml
  - name: metal3
    version: 304.0.16+up0.12.6
    repositoryName: suse-edge-charts
    targetNamespace: metal3-system
    createNamespace: true
    installationNamespace: kube-system
    valuesFile: metal3.yaml
  - name: rancher-turtles
    version: 304.0.6+up0.24.0
    repositoryName: suse-edge-charts
    targetNamespace: rancher-turtles-system
    createNamespace: true
    installationNamespace: kube-system
  - name: neuvector-crd
    version: 107.0.0+up2.8.7
    repositoryName: rancher-charts
    targetNamespace: neuvector
    createNamespace: true
    installationNamespace: kube-system
    valuesFile: neuvector.yaml
  - name: neuvector
    version: 107.0.0+up2.8.7
    repositoryName: rancher-charts
    targetNamespace: neuvector
    createNamespace: true
    installationNamespace: kube-system
    valuesFile: neuvector.yaml
  - name: rancher
    version: 2.12.1
    repositoryName: rancher-prime
    targetNamespace: cattle-system
    createNamespace: true
    installationNamespace: kube-system
    valuesFile: rancher.yaml
repositories:
  - name: jetstack
    url: https://charts.jetstack.io
  - name: rancher-charts
    url: https://charts.rancher.io/
  - name: suse-edge-charts
    url: oci://registry.suse.com/edge/charts
  - name: rancher-prime
    url: https://charts.rancher.com/server-charts/prime
network:
  apiHost: $API_HOST
  apiVIP: $API_VIP
nodes:
  - hostname: mgmt-cluster-node1
    initializer: true

```

```

    type: server
#   - hostname: mgmt-cluster-node2
#   type: server
#   - hostname: mgmt-cluster-node3
#   type: server

```

`mgmt-cluster.yaml` 定義ファイルのフィールドと値について説明するために、ここではこのファイルを次のセクションに分割しています。

- イメージセクション(定義ファイル):

```

image:
  imageType: iso
  arch: x86_64
  baseImage: SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso
  outputImageName: eib-mgmt-cluster-image.iso

```

ここで、`baseImage`は、SUSE Customer CenterまたはSUSEダウンロードページからダウンロードした元のイメージです。`outputImageName`は、管理クラスタのプロビジョニングに使用する新しいイメージの名前です。

- オペレーティングシステムセクション(定義ファイル):

```

operatingSystem:
  isoConfiguration:
    installDevice: /dev/sda
  users:
    - username: root
      encryptedPassword: $ROOT_PASSWORD
  packages:
    packageList:
      - jq
    sccRegistrationCode: $SCC_REGISTRATION_CODE

```

ここで、`installDevice`はオペレーティングシステムのインストールに使用するデバイス、`username`および`encryptedPassword`はシステムへのアクセスに使用する資格情報、`packageList`はインストールするパッケージのリスト(`jq`はインストールプロセス中に内部的に必要)です。`sccRegistrationCode`は構築時にパッケージと依存関係を取得するために使用する登録コードで、SUSE Customer Centerから取得できます。暗号化パスワードは次のように`openssl`コマンドを使用して生成できます。

```
openssl passwd -6 MyPassword!123
```

この出力は次のようになります。

```

$6$UrXB1sAGs46D0iSq$HSwi9GFJLCorm0J53nF2Sq8YEoyINhHc0bHzX2R8h13mswUIsMwzx4eUzn/
rRx0QPv4JIb0eWCoNrxGiKH4R31

```

- Kubernetesセクション(定義ファイル):

```
kubernetes:
  version: v1.33.3+rke2r1
  helm:
    charts:
      - name: cert-manager
        repositoryName: jetstack
        version: 1.18.2
        targetNamespace: cert-manager
        valuesFile: certmanager.yaml
        createNamespace: true
        installationNamespace: kube-system
      - name: longhorn-crd
        version: 107.0.0+up1.9.1
        repositoryName: rancher-charts
        targetNamespace: longhorn-system
        createNamespace: true
        installationNamespace: kube-system
      - name: longhorn
        version: 107.0.0+up1.9.1
        repositoryName: rancher-charts
        targetNamespace: longhorn-system
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: longhorn.yaml
      - name: metal3
        version: 304.0.16+up0.12.6
        repositoryName: suse-edge-charts
        targetNamespace: metal3-system
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: metal3.yaml
      - name: rancher-turtles
        version: 304.0.6+up0.24.0
        repositoryName: suse-edge-charts
        targetNamespace: rancher-turtles-system
        createNamespace: true
        installationNamespace: kube-system
      - name: neuvector-crd
        version: 107.0.0+up2.8.7
        repositoryName: rancher-charts
        targetNamespace: neuvector
        createNamespace: true
        installationNamespace: kube-system
        valuesFile: neuvector.yaml
      - name: neuvector
        version: 107.0.0+up2.8.7
        repositoryName: rancher-charts
        targetNamespace: neuvector
```

```

    createNamespace: true
    installationNamespace: kube-system
    valuesFile: neuvector.yaml
  - name: rancher
    version: 2.12.1
    repositoryName: rancher-prime
    targetNamespace: cattle-system
    createNamespace: true
    installationNamespace: kube-system
    valuesFile: rancher.yaml
repositories:
  - name: jetstack
    url: https://charts.jetstack.io
  - name: rancher-charts
    url: https://charts.rancher.io/
  - name: suse-edge-charts
    url: oci://registry.suse.com/edge/charts
  - name: rancher-prime
    url: https://charts.rancher.com/server-charts/prime
network:
  apiHost: $API_HOST
  apiVIP: $API_VIP
nodes:
  - hostname: mgmt-cluster-node1
    initializer: true
    type: server
# - hostname: mgmt-cluster-node2
#   type: server
# - hostname: mgmt-cluster-node3
#   type: server

```

helmセクションには、インストールするHelmチャートのリスト、使用するリポジトリ、およびこれらすべてのバージョン設定が含まれます。

networkセクションには、RKE2コンポーネントが使用するapiHostやapiVIPなどのネットワーク設定が含まれます。apiVIPは、ネットワーク内で使用されていないIPアドレスにし、DHCPを使用する場合はDHCPプールから除外してください。マルチノードクラスタでは、apiVIPがKubernetes APIサーバへのアクセスに使用されます。apiHostは、RKE2コンポーネントが使用するapiVIPへの名前解決として機能します。

nodesセクションには、クラスタで使用するノードのリストが含まれています。この例では、シングルノードクラスタを使用していますが、リストにノードを追加する(行のコメントを解除する)ことによってマルチノードクラスタに拡張できます。

注記

- ノードの名前はクラスタ内で固有である必要があります。
- オプションで、`initializer`フィールドを使用してブートストラップ ホストを指定します。これを指定しない場合、リストの最初のノードになります。
- ネットワーク設定が必要な場合、ノードの名前はネットワークフォルダ(40.3.5項「[ネットワーキングフォルダ](#)」)で定義されたホスト名と同じである必要があります。

40.3.3 Custom フォルダ

`custom` フォルダには次のサブフォルダが含まれます。

```
...
├─ custom
│  └─ scripts
│     ├── 99-register.sh
│     ├── 99-mgmt-setup.sh
│     └─ 99-alias.sh
│  └─ files
│     ├── rancher.sh
│     ├── mgmt-stack-setup.service
│     ├── metal3.sh
│     └─ basic-setup.sh
...
```

- `custom/files` フォルダには、管理クラスタが使用する設定ファイルが含まれます。
- `custom/scripts` フォルダには、管理クラスタが使用するスクリプトが含まれます。

`custom/files` フォルダには、次のファイルが含まれます。

- `basic-setup.sh`: Metal3、Rancher、MetalLBの設定パラメータが含まれます。このファイルを変更するのは、使用するネームスペースを変更する場合のみにしてください。

```
#!/bin/bash
# Pre-requisites. Cluster already running
export KUBECTL="/var/lib/rancher/rke2/bin/kubectll"
export KUBECONFIG="/etc/rancher/rke2/rke2.yaml"

#####
```



```
# METAL3 DETAILS #
#####
export METAL3_CHART_TARGETNAMESPACE="metal3-system"

#####
# METALLB #
#####
export METALLB_NAMESPACE="metallb-system"

#####
# RANCHER #
#####
export RANCHER_CHART_TARGETNAMESPACE="cattle-system"
export RANCHER_FINALPASSWORD="adminadminadmin"

die(){
    echo ${1} 1>&2
    exit ${2}
}
```

- metal3.sh: 使用するMetal3コンポーネントの設定が含まれます(変更不要)。今後のバージョンでは、このスクリプトは代わりにRancher Turtlesを使用するように置き換えられて、使いやすくなる予定です。

```
#!/bin/bash
set -euo pipefail

BASEDIR="$(dirname "$0")"
source ${BASEDIR}/basic-setup.sh

METAL3_LOCK_NAMESPACE="default"
METAL3_LOCK_CM_NAME="metal3-lock"

trap 'catch $? $LINENO' EXIT

catch() {
    if [ "$1" != "0" ]; then
        echo "Error $1 occurred on $2"
        ${KUBECTL} delete configmap ${METAL3_LOCK_CM_NAME} -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 ${METAL3_LOCK_NAMESPACE} ${METAL3_LOCK_CM_NAME} -o name | wc -l) -lt 1 ); then
    ${KUBECTL} create configmap ${METAL3_LOCK_CM_NAME} -n ${METAL3_LOCK_NAMESPACE} --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 -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 ${METALLB_NAMESPACE} $(KUBECTL
get pods -n ${METALLB_NAMESPACE} -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 ${METALLB_NAMESPACE} ironic-ip-pool -o name || cat
<<-EOF | ${KUBECTL} apply -f -
    apiVersion: metallb.io/v1beta1
    kind: IPAddressPool
    metadata:
      name: ironic-ip-pool
      namespace: ${METALLB_NAMESPACE}
    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 ${METALLB_NAMESPACE} 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 rancher is deployed
if [ $((${KUBECTL} get pods -n ${RANCHER_CHART_TARGETNAMESPACE} -l app=rancher -o
name | wc -l) -ge 1 ); then
    cat <<-EOF | ${KUBECTL} apply -f -
        apiVersion: management.cattle.io/v3
        kind: Feature
        metadata:
            name: embedded-cluster-api
        spec:
            value: false
        EOF

    # Disable Rancher webhooks for CAPI
    ${KUBECTL} delete --ignore-not-found=true
    mutatingwebhookconfiguration.admissionregistration.k8s.io mutating-webhook-
    configuration
    ${KUBECTL} delete --ignore-not-found=true
    validatingwebhookconfigurations.admissionregistration.k8s.io validating-webhook-
    configuration
    ${KUBECTL} wait --for=delete namespace/cattle-provisioning-capi-system --
    timeout=300s
fi

# Clean up the lock cm

${KUBECTL} delete configmap ${METAL3LOCKCMNAME} -n ${METAL3LOCKNAMESPACE}
```

- rancher.sh: 使用する Rancher コンポーネントの設定が含まれます(変更不要)。

```
#!/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
}
```

```

    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: systemdサービスを作成して初回ブート時にスク립トを実行するための設定が含まれます(変更不要)。

```

[Unit]
Description=Setup Management stack components
Wants=network-online.target
# It requires rke2 or k3s running, but it will not fail if those services are
not present
After=network.target network-online.target rke2-server.service k3s.service
# At least, the basic-setup.sh one needs to be present
ConditionPathExists=/opt/mgmt/bin/basic-setup.sh

[Service]
User=root
Type=forking
# Metal3 can take A LOT to download the IPA image
TimeoutStartSec=1800

ExecStartPre=/bin/sh -c "echo 'Setting up Management components...'"
# Scripts are executed in StartPre because Start can only run a single one
ExecStartPre=/opt/mgmt/bin/rancher.sh
ExecStartPre=/opt/mgmt/bin/metal3.sh
ExecStart=/bin/sh -c "echo 'Finished setting up Management components'"
RemainAfterExit=yes
KillMode=process
# Disable & delete everything
ExecStartPost=rm -f /opt/mgmt/bin/rancher.sh
ExecStartPost=rm -f /opt/mgmt/bin/metal3.sh
ExecStartPost=rm -f /opt/mgmt/bin/basic-setup.sh
ExecStartPost=/bin/sh -c "systemctl disable mgmt-stack-setup.service"
ExecStartPost=rm -f /etc/systemd/system/mgmt-stack-setup.service

[Install]
WantedBy=multi-user.target

```

custom/scriptsフォルダには次のファイルが含まれます。

- 99-alias.sh スクリプト: 管理クラスタが初回ブート時に kubeconfig ファイルを読み込むために使用するエイリアスが含まれます(変更不要)。

```
#!/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 スクリプト: 初回ブート時にスクリプトをコピーするための設定が含まれます(変更不要)。

```
#!/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 スクリプト: SCC登録コードを使用してシステムを登録するための設定が含まれます。アカウントにシステムを登録するには、`${SCC_ACCOUNT_EMAIL}`および`${SCC_REGISTRATION_CODE}`が正しく設定されている必要があります。

```
#!/bin/bash
set -euo pipefail

# Registration https://www.suse.com/support/kb/doc/?id=000018564
if ! which SUSEConnect > /dev/null 2>&1; then
    zypper --non-interactive install suseconnect-ng
fi
SUSEConnect --email "${SCC_ACCOUNT_EMAIL}" --url "https://scc.suse.com" --regcode
"${SCC_REGISTRATION_CODE}"
```

40.3.4 Kubernetes フォルダ

kubernetes フォルダには次のサブフォルダが含まれます。

```
...
├─ 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
...

```

kubernetes/config フォルダには次のファイルが含まれます。

- server.yaml: デフォルトでは、デフォルトでインストールされているCNIプラグインはCiliumであるため、このフォルダとファイルを作成する必要はありません。CNIプラグインをカスタマイズする必要がある場合に備えて、kubernetes/config フォルダにある server.yaml ファイルを使用できます。このファイルには次の情報が含まれます。

```

cni:
- multus
- cilium
write-kubeconfig-mode: '0644'
selinux: true
system-default-registry: registry.rancher.com

```



注記

これは、使用するCNIプラグインなどの特定のKubernetesカスタマイズを定義する任意のファイルです。さまざまなオプションについては、[公式ドキュメント \(https://docs.rke2.io/install/configuration\)](https://docs.rke2.io/install/configuration) で確認できます。

os-files/var/lib/rancher/rke2/server/manifests フォルダには次のファイルが含まれます。

- rke2-ingress-config.yaml: 管理クラスタ用のIngressサービスを作成するための設定が含まれます(変更不要)。

```

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
```



注記

HelmChartConfigは、以前のリリースで説明している [kubernetes/manifests](#) 経由ではなく、[os-files](#) 経由で [/var/lib/rancher/rke2/server/manifests](#) ディレクトリに含める必要があります。

[kubernetes/manifests](#) フォルダには次のファイルが含まれます。

- [neuvector-namespace.yaml](#): [NeuVector](#) ネームスペースを作成するための設定が含まれます(変更不要)。

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    pod-security.kubernetes.io/enforce: privileged
  name: neuvector
```

- [ingress-l2-adv.yaml](#): [MetalLB](#) コンポーネントの [L2Advertisement](#) を作成するための設定が含まれます(変更不要)。

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: ingress-l2-adv
  namespace: metallb-system
spec:
  ipAddressPools:
    - ingress-ippool
```

- [ingress-ippool.yaml](#): [rke2-ingress-nginx](#) コンポーネントの [IPAddressPool](#) を作成するための設定が含まれます。 `${INGRESS_VIP}` を正しく設定し、[rke2-ingress-nginx](#) コンポーネントで使用するために予約するIPアドレスを定義する必要があります。


```

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]}

```

kubernetes/helm/values フォルダには次のファイルが含まれます。

- rancher.yaml: Rancher コンポーネントを作成する設定が含まれます。\${INGRESS_VIP} を正しく設定して、Rancher コンポーネントで使用する IP アドレスを定義する必要があります。Rancher コンポーネントにアクセスする URL は、[https://rancher-\\${INGRESS_VIP}.sslip.io](https://rancher-${INGRESS_VIP}.sslip.io) です。

```

hostname: rancher-${INGRESS_VIP}.sslip.io
bootstrapPassword: "foobar"
replicas: 1
global:
  cattle:
    systemDefaultRegistry: "registry.rancher.com"

```

- neuvector.yaml: NeuVector コンポーネントを作成するための設定が含まれます(変更不要)。

```

controller:
  replicas: 1
  ranchersso:
    enabled: true
manager:
  enabled: false
cve:
  scanner:
    enabled: false
    replicas: 1
k3s:
  enabled: true
crdwebhook:
  enabled: false
registry: "registry.rancher.com"

```

```
global:
  cattle:
    systemDefaultRegistry: "registry.rancher.com"
```

- longhorn.yaml: Longhornコンポーネントを作成するための設定が含まれます(変更不要)。

```
global:
  cattle:
    systemDefaultRegistry: "registry.rancher.com"
```

- metal3.yaml: Metal3コンポーネントを作成するための設定が含まれます。`${METAL3_VIP}`を正しく設定して、Metal3コンポーネントで使用するIPアドレスを定義する必要があります。

```
global:
  ironicIP: ${METAL3_VIP}
  enable_vmedia_tls: false
  additionalTrustedCAs: false
metal3-ironic:
  global:
    predictableNicNames: "true"
  persistence:
    ironic:
      size: "5Gi"
```



注記

メディアサーバは、Metal³に含まれるオプションの機能です(デフォルトでは無効になっています)。このMetal3の機能を使用するには、以前のマニフェストで設定する必要があります。Metal³メディアサーバを使用するには、次の変数を指定します。

- メディアサーバ機能を有効にするために、`global`セクションに`enable_metal3_media_server`を追加して`true`に設定します。
- メディアサーバ設定に次の内容を含めます。`${MEDIA_VOLUME_PATH}`はメディア内のメディアボリュームのパスです(例: `/home/metal3/bmh-image-cache`)。

```
metal3-media:
  mediaVolume:
    hostPath: ${MEDIA_VOLUME_PATH}
```

外部のメディアサーバを使用してイメージを保存できます。外部のメディアサーバをTLSで使用する場合は、次の設定を変更する必要があります。

- 前の`metal3.yaml`ファイルで`additionalTrustedCAs`を`true`に設定し、外部のメディアサーバから、信頼できる追加のCAを有効にします。
- `kubernetes/manifests/metal3-cacert-secret.yaml`フォルダに次のシークレット設定を含めて、外部のメディアサーバのCA証明書を保存します。

```
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 }}
```

`additional_ca_cert`は、外部のメディアサーバのbase64エンコードCA証明書です。次のコマンドを使用し、証明書をエンコードして手動でシークレットを生成できます。

```
kubectyl -n meta3-system create secret generic tls-ca-additional --from-file=ca-additional.crt=./ca-additional.crt
```

- `certmanager.yaml`: `Cert-Manager`コンポーネントを作成するための設定が含まれます(変更不要)。

```
installCRDs: true
```

40.3.5 ネットワーキングフォルダ

`network`フォルダには、管理クラスタのノードと同じ数のファイルが含まれます。ここでは、ノードは1つのみであるため、`mgmt-cluster-node1.yaml`というファイルが1つあるだけです。ファイルの名前は、上述の`network/node`セクションで`mgmt-cluster.yaml`定義ファイルに定義されているホスト名と一致させる必要があります。

ネットワーキング設定をカスタマイズして特定の静的IPアドレスを使用する必要がある場合(たとえばDHCPを使用しないシナリオの場合)、`network`フォルダにある`mgmt-cluster-node1.yaml`ファイルを使用できます。このファイルには次の情報が含まれます。

- `${MGMT_GATEWAY}`: ゲートウェイのIPアドレス。
- `${MGMT_DNS}`: DNSサーバのIPアドレス。
- `${MGMT_MAC}`: ネットワークインタフェースのMACアドレス。
- `${MGMT_NODE_IP}`: 管理クラスタのIPアドレス。

```

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

```

DHCPを使用してIPアドレスを取得する場合、次の設定を使用できます(`${MGMT_MAC}`変数を使用して、MACアドレスを正しく設定する必要があります)。

```

## 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

```

注記

- 管理クラスタのノード数に応じて、`mgmt-cluster-node2.yaml`、`mgmt-cluster-node3.yaml`などのように追加のファイルを作成して残りのノードを設定できます。
- `routes`セクションは、管理クラスタのルーティングテーブルを定義するために使用します。

40.4 エアギャップ環境のイメージの準備

このセクションでは、エアギャップ環境のイメージを準備する方法について説明し、前の各セクションとの相違点のみを示します。エアギャップ環境のイメージを準備するには、前のセクション(接続環境のイメージの準備(40.3項「[接続環境用のイメージの準備](#)」))を次のように変更する必要があります。

- `mgmt-cluster.yaml`ファイルを変更して`embeddedArtifactRegistry`セクションを含め、`images`フィールドに、EIB出力イメージに組み込むすべてのコンテナイメージを設定する必要があります。
- `mgmt-cluster.yaml`ファイルは、`rancher-turtles-airgap-resources` helmチャートを含むように変更される必要があります。
- エアギャップ環境を使用する場合、`custom/scripts/99-register.sh`スクリプトは削除する必要があります。

40.4.1 定義ファイルの変更

`mgmt-cluster.yaml`ファイルを変更して`embeddedArtifactRegistry`セクションを含める必要があります。このセクションで、`images`フィールドには、出力イメージに含めるすべてのコンテナイメージのリストを含める必要があります。

注記

次に、`embeddedArtifactRegistry`セクションが含まれる`mgmt-cluster.yaml`ファイルの例を示します。リストされているイメージに、必要なコンポーネントのバージョンが含まれていることを確認してください。

`rancher-turtles-airgap-resources` helmチャートは追加される必要もあり、これにより、[Rancher Turtlesエアギャップドキュメント \(https://documentation.suse.com/cloudnative/cluster-api/v0.19/en/getting-started/air-gapped-environment.html\)](https://documentation.suse.com/cloudnative/cluster-api/v0.19/en/getting-started/air-gapped-environment.html) で説明されているリソースが作成されます。また、必要な設定を指定するために、`rancher-turtles` チャート用の `turtles.yaml` 値ファイルも必要です。

```
apiVersion: 1.3
image:
  imageType: iso
  arch: x86_64
  baseImage: SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso
  outputImageName: eib-mgmt-cluster-image.iso
operatingSystem:
  isoConfiguration:
    installDevice: /dev/sda
  users:
    - username: root
      encryptedPassword: $ROOT_PASSWORD
  packages:
    packageList:
      - jq
    sccRegistrationCode: $SCC_REGISTRATION_CODE
kubernetes:
  version: v1.33.3+rke2r1
helm:
  charts:
    - name: cert-manager
      repositoryName: jetstack
      version: 1.18.2
      targetNamespace: cert-manager
      valuesFile: certmanager.yaml
      createNamespace: true
      installationNamespace: kube-system
    - name: longhorn-crd
      version: 107.0.0+up1.9.1
      repositoryName: rancher-charts
      targetNamespace: longhorn-system
      createNamespace: true
      installationNamespace: kube-system
    - name: longhorn
      version: 107.0.0+up1.9.1
      repositoryName: rancher-charts
      targetNamespace: longhorn-system
      createNamespace: true
      installationNamespace: kube-system
      valuesFile: longhorn.yaml
    - name: metal3
      version: 304.0.16+up0.12.6
      repositoryName: suse-edge-charts
```

```

    targetNamespace: metal3-system
    createNamespace: true
    installationNamespace: kube-system
    valuesFile: metal3.yaml
  - name: rancher-turtles
    version: 304.0.6+up0.24.0
    repositoryName: suse-edge-charts
    targetNamespace: rancher-turtles-system
    createNamespace: true
    installationNamespace: kube-system
    valuesFile: turtles.yaml
  - name: rancher-turtles-airgap-resources
    version: 304.0.6+up0.24.0
    repositoryName: suse-edge-charts
    targetNamespace: rancher-turtles-system
    createNamespace: true
    installationNamespace: kube-system
  - name: neuvector-crd
    version: 107.0.0+up2.8.7
    repositoryName: rancher-charts
    targetNamespace: neuvector
    createNamespace: true
    installationNamespace: kube-system
    valuesFile: neuvector.yaml
  - name: neuvector
    version: 107.0.0+up2.8.7
    repositoryName: rancher-charts
    targetNamespace: neuvector
    createNamespace: true
    installationNamespace: kube-system
    valuesFile: neuvector.yaml
  - name: rancher
    version: 2.12.1
    repositoryName: rancher-prime
    targetNamespace: cattle-system
    createNamespace: true
    installationNamespace: kube-system
    valuesFile: rancher.yaml
repositories:
  - name: jetstack
    url: https://charts.jetstack.io
  - name: rancher-charts
    url: https://charts.rancher.io/
  - name: suse-edge-charts
    url: oci://registry.suse.com/edge/charts
  - name: rancher-prime
    url: https://charts.rancher.com/server-charts/prime
network:
  apiHost: $API_HOST
  apiVIP: $API_VIP
nodes:

```

```

- hostname: mgmt-cluster-node1
  initializer: true
  type: server
# - hostname: mgmt-cluster-node2
#   type: server
# - hostname: mgmt-cluster-node3
#   type: server
#     type: server
embeddedArtifactRegistry:
  images:
    - name: registry.rancher.com/rancher/hardened-cluster-autoscaler:v1.10.2-build20250611
    - name: registry.rancher.com/rancher/hardened-cni-plugins:v1.7.1-build20250611
    - name: registry.rancher.com/rancher/hardened-coredns:v1.12.2-build20250611
    - name: registry.rancher.com/rancher/hardened-k8s-metrics-server:v0.8.0-build20250704
    - name: registry.rancher.com/rancher/hardened-multus-cni:v4.2.1-build20250627
    - name: registry.rancher.com/rancher/klipper-helm:v0.9.8-build20250709
    - name: registry.rancher.com/rancher/mirrored-cilium-cilium:v1.17.6
    - name: registry.rancher.com/rancher/mirrored-cilium-operator-generic:v1.17.6
    - name: registry.rancher.com/rancher/mirrored-longhornio-csi-attacher:v4.9.0-20250709
    - name: registry.rancher.com/rancher/mirrored-longhornio-csi-node-driver-
registrar:v2.14.0-20250709
  - name: registry.rancher.com/rancher/mirrored-longhornio-csi-
provisioner:v5.3.0-20250709
  - name: registry.rancher.com/rancher/mirrored-longhornio-csi-resizer:v1.14.0-20250709
  - name: registry.rancher.com/rancher/mirrored-longhornio-csi-
snapshotter:v8.3.0-20250709
  - name: registry.rancher.com/rancher/mirrored-longhornio-
livenessprobe:v2.16.0-20250709
  - name: registry.rancher.com/rancher/mirrored-longhornio-longhorn-engine:v1.9.1
  - name: registry.rancher.com/rancher/mirrored-longhornio-longhorn-instance-
manager:v1.9.1
  - name: registry.rancher.com/rancher/mirrored-longhornio-longhorn-manager:v1.9.1
  - name: registry.rancher.com/rancher/mirrored-longhornio-longhorn-share-manager:v1.9.1
  - name: registry.rancher.com/rancher/mirrored-longhornio-longhorn-ui:v1.9.1
  - name: registry.rancher.com/rancher/mirrored-sig-storage-snapshot-controller:v8.2.0
  - name: registry.rancher.com/rancher/neuvector-compliance-config:1.0.6
  - name: registry.rancher.com/rancher/neuvector-controller:5.4.5
  - name: registry.rancher.com/rancher/neuvector-enforcer:5.4.5
  - name: registry.rancher.com/rancher/nginx-ingress-controller:v1.12.4-hardened2
  - name: registry.suse.com/rancher/cluster-api-addon-provider-fleet:v0.11.0
  - name: registry.rancher.com/rancher/cluster-api-operator:v0.18.1
  - name: registry.rancher.com/rancher/fleet-agent:v0.13.1
  - name: registry.rancher.com/rancher/fleet:v0.13.1
  - name: registry.rancher.com/rancher/hardened-node-feature-discovery:v0.15.7-
build20250425
  - name: registry.rancher.com/rancher/rancher-webhook:v0.8.1
  - name: registry.rancher.com/rancher/rancher/turtles:v0.24.0
  - name: registry.rancher.com/rancher/rancher:v2.12.1
  - name: registry.rancher.com/rancher/shell:v0.4.1
  - name: registry.rancher.com/rancher/system-upgrade-controller:v0.16.0
  - name: registry.suse.com/rancher/cluster-api-controller:v1.10.5

```



```
- name: registry.suse.com/rancher/cluster-api-provider-metal3:v1.10.2
- name: registry.suse.com/rancher/cluster-api-provider-rke2-bootstrap:v0.20.1
- name: registry.suse.com/rancher/cluster-api-provider-rke2-controlplane:v0.20.1
- name: registry.suse.com/rancher/hardened-sriov-network-operator:v1.5.0-build20250425
- name: registry.suse.com/rancher/ip-address-manager:v1.10.2
- name: registry.rancher.com/rancher/kubectrl:v1.32.2
- name: registry.rancher.com/rancher/mirrored-cluster-api-controller:v1.9.5
```

40.4.2 カスタムフォルダの変更

- エアギャップ環境を使用する場合、`custom/scripts/99-register.sh`スクリプトを削除する必要があります。ディレクトリ構造からわかるように、`99-register.sh`スクリプトは`custom/scripts`フォルダに含まれていません。

40.4.3 Helm値フォルダの変更

- `turtles.yaml`: Rancher Turtlesのエアギャップ操作を指定するために必要な設定が含まれます。これは`rancher-turtles-airgap-resources`チャートのインストールによって異なることに注意してください。

```
cluster-api-operator:
  cluster-api:
    core:
      fetchConfig:
        selector: "{\"matchLabels\": {\"provider-components\": \"core\"}}"
    rke2:
      bootstrap:
        fetchConfig:
          selector: "{\"matchLabels\": {\"provider-components\": \"rke2-bootstrap\"}}"
    controlPlane:
      fetchConfig:
        selector: "{\"matchLabels\": {\"provider-components\": \"rke2-control-plane\"}}"
    metal3:
      infrastructure:
        fetchConfig:
```

```
selector: "{\"matchLabels\": {\"provider-components\": \"metal3\"}}\"
```

40.5 イメージの作成

前の各セクションに従ってディレクトリ構造を準備したら(接続シナリオとエアギャップシナリオの両方が対象です)、次のコマンドを実行してイメージを構築します。

```
podman run --rm --privileged -it -v $PWD:/eib \
registry.suse.com/edge/3.4/edge-image-builder:1.3.0 \
build --definition-file mgmt-cluster.yaml
```

ISO出力イメージファイルが作成されます。ここでは、上記のイメージ定義に基づく `eib-mgmt-cluster-image.iso` という名前のファイルです。

40.6 管理クラスタのプロビジョニング

前のイメージには、上述のコンポーネントがすべて含まれています。このイメージを使って、仮想マシンまたはベアメタルサーバを使用して(仮想メディア機能を使用して)管理クラスタをプロビジョニングできます。

40.7 デュアルスタックの考慮事項と設定

前のセクションで示した例では、シングルスタックIPv4管理クラスタの設定方法に関するガイドンスと例を示しています。このような管理クラスタは、ダウンストリームクラスタの動作状態とは無関係であり、デプロイ後にIPv4/IPv6シングルスタック設定またはデュアルスタック設定のいずれかで動作するように個別に設定できます。ただし、管理クラスタの設定方法は、プロビジョニングフェーズ中に使用可能な通信プロトコルに直接影響します。このフェーズでは、管理クラスタとダウンストリームホストがサポートするプロトコルに応じて、インバンド通信とアウトオブバンド通信の両方が行われる必要があります。BMCやダウンストリームクラスタノードの一部またはすべてがIPv6を使用することが想定される場合、管理クラスタにはデュアルスタックセットアップが必要となります。



注記

シングルスタックIPv6管理クラスタはまだサポートされていません。

デュアルスタック機能を実現するには、KubernetesにPodとService用のIPv4とIPv6両方のCIDRを提供する必要があります。ただし、EIBを使用して管理クラスティメージを構築する前に、他のコンポーネントについても特定の調整が必要です。Metal³プロビジョニングサービス(Ironic)は、インフラストラクチャや要件に応じて異なる方法で設定できます。

- Ironicサービスは、単一のIPアドレスではなくシステム上のすべてのインタフェースでリスンするように設定できます。したがって、管理クラスタホストで関連インタフェースにIPv4とIPv6の両方のアドレスが割り当てられている限り、プロビジョニング中にいずれのアドレスも使用される可能性があります。この時点では、URL生成(Baremetal OperatorやBMCなどの他のサービスが利用するため)には、これらのアドレスの1つしか選択できないことに注意してください。結果として、BMCとのIPv6通信を有効にするには、IPv6アドレスを含むBMH定義を処理する際に、Baremetal OperatorはIPv6 URLを公開して渡すように指示できます。つまり、BMCがIPv6対応と識別された場合、プロビジョニングはIPv6経由でのみ実行され、それ以外の場合はすべてIPv4経由で実行されます。
- IPv4とIPv6の両方に解決される単一のホスト名をMetal³で使用することで、IronicがこれらのアドレスをバインディングとURL作成に使用できるようになります。このアプローチにより、簡単な設定と柔軟な動作が可能になります(プロビジョニングの各ステップでIPv4とIPv6の両方が使用可能)。ただし、既存のDNSサーバ、IP割り当て、およびレコードがすでに設定されているインフラストラクチャが必要です。

両方のケースで、KubernetesはIPv4とIPv6の両方に使用するCIDRを認識する必要があります。そのため、EIBディレクトリ内の`kubernetes/config/server.yaml`に次の行を追加できます。必ずIPv4を最初にリストしてください。

```
service-cidr: 10.96.0.0/12,fd12:4567:789c::/112
cluster-cidr: 193.168.0.0/18,fd12:4567:789b::/48
```

一部のコンテナはホストネットワーキングを利用します。そのため、`network`ディレクトリの下でホストのネットワーク設定を変更して、IPv6接続を有効にします。

```
routes:
  config:
  - destination: 0.0.0.0/0
    next-hop-address: ${MGMT_GATEWAY_V4}
    next-hop-interface: eth0
  - destination: ::/0
    next-hop-address: ${MGMT_GATEWAY_V6}
    next-hop-interface: eth0
dns-resolver:
  config:
    server:
    - ${MGMT_DNS}
```

```

- 8.8.8.8
- 2001:4860:4860::8888
interfaces:
- name: eth0
  type: ethernet
  state: up
  mac-address: ${MGMT_MAC}
  ipv4:
    address:
      - ip: ${MGMT_CLUSTER_IP_V4}
        prefix-length: 24
    dhcp: false
    enabled: true
  ipv6:
    address:
      - ip: ${MGMT_CLUSTER_IP_V6}
        prefix-length: 128
    dhcp: false
    autoconf: false
    enabled: true

```

プレースホルダーをゲートウェイIPアドレス、追加のDNSサーバ(必要な場合)、ネットワークインタフェースのMACアドレス、および管理クラスタのIPアドレスで置き換えてください。代わりにアドレス自動設定を希望する場合は、以下の抜粋を参照すれば、`${MGMT_MAC}`変数を設定するだけで済みます。

```

interfaces:
- name: eth0
  type: ethernet
  state: up
  mac-address: ${MGMT_MAC}
  ipv4:
    enabled: true
    dhcp: true
  ipv6:
    enabled: false
    dhcp: true
    autoconf: true

```

最初のオプションから開始し、`kubernetes/helm/values/metal3.yaml`を以下のように作成することで、単一ノード設定の残りのファイルを定義できます。

```

global:
  ironicIP: ${MGMT_CLUSTER_IP_V4}
  enable_vmedia_tls: false
  additionalTrustedCAs: false
metal3-ironic:
  global:
    predictableNicNames: true

```

```
listenOnAll: true
persistence:
  ironic:
    size: "5Gi"
service:
  type: NodePort
metal3-baremetal-operator:
  baremetaloperator:
    externalHttpIPv6: ${MGMT_CLUSTER_IP_V6}
```

また、kubernetes/helm/values/rancher.yamlを次のようにします。

```
hostname: rancher-${MGMT_CLUSTER_IP_V4}.sslip.io
bootstrapPassword: "foobar"
replicas: 1
global:
  cattle:
    systemDefaultRegistry: "registry.rancher.com"
```

ここで、\${MGMT_CLUSTER_IP_V4}と\${MGMT_CLUSTER_IP_V6}は、ホストに以前に割り当てられたIPアドレスです。

または、IPアドレスの代わりにホスト名を使用するには、kubernetes/helm/values/metal3.yamlを次のように変更します。

```
global:
  provisioningHostname: `${MGMT_CLUSTER_HOSTNAME}`
  enable_vmedia_tls: false
  additionalTrustedCAs: false
metal3-ironic:
  global:
    predictableNicNames: true
  persistence:
    ironic:
      size: "5Gi"
  service:
    type: NodePort
```

また、kubernetes/helm/values/rancher.yamlを次のように変更します。

```
hostname: rancher-${MGMT_CLUSTER_HOSTNAME}.sslip.io
bootstrapPassword: "foobar"
replicas: 1
global:
  cattle:
    systemDefaultRegistry: "registry.rancher.com"
```

ここで、\${MGMT_CLUSTER_HOSTNAME}は、ホストのIPアドレスに解決される完全修飾ドメイン名である必要があります。

詳細については、SUSE Edge GitHubリポジトリの「dual-stack」フォルダ (<https://github.com/suse-edge/atip/tree/main/telco-examples/mgmt-cluster/dual-stack>)  を参照してください。ここでディレクトリ構造の例を確認できます。

41 通信機能の設定

このセクションでは、SUSE Telco Cloudを介してデプロイされたクラスタの通信事業者固有の機能の設定について記述および説明します。

ダイレクトネットワークプロビジョニングのデプロイメント方法を使用します。この方法については、自動化されたプロビジョニング(第42章「完全に自動化されたダイレクトネットワークプロビジョニング」)に関するセクションで説明しています。

このセクションでは、次のトピックについて説明します。

- リアルタイム用のカーネルイメージ(41.1項「リアルタイム用のカーネルイメージ」): リアルタイムカーネルで使用するカーネルイメージ。
- 低レイテンシとハイパフォーマンスのためのカーネル引数(41.2項「低レイテンシとハイパフォーマンスのためのカーネル引数」): 通信ワークロードを最大のパフォーマンスと低レイテンシで実行するために、リアルタイムカーネルによって使用されるカーネル引数。
- Tunedとカーネル引数によるCPUピンニング(41.3項「Tunedとカーネル引数によるCPUピンニング」): カーネル引数とTunedプロファイルを使用してCPUを分離します。
- CNI設定(41.4項「CNI設定」): Kubernetesクラスタで使用するCNI設定。
- SR-IOV設定(41.5項「SR-IOV」): Kubernetesワークロードで使用するSR-IOV設定。
- DPDK設定(41.6項「DPDK」): システムで使用するDPDK設定。
- vRANアクセラレーションカード(41.7項「vRANアクセラレーション(Intel ACC100/ACC200)」): Kubernetesワークロードで使用するアクセラレーションカードの設定。
- Huge Page (41.8項「Huge Page」): Kubernetesワークロードで使用するHuge Pageの設定。
- KubernetesでのCPUピンニング(41.9項「KubernetesでのCPUピンニング」): CPUピンニングを活用するようにKubernetesとアプリケーションを設定する。
- NUMA対応のスケジューリング設定(41.10項「NUMA対応のスケジューリング」): Kubernetesワークロードで使用するNUMA対応のスケジューリング設定。
- Metal LB設定(41.11項「MetalLB」): Kubernetesワークロードで使用するMetal LB設定。

- プライベートレジストリ設定(41.12項「プライベートレジストリ設定」): Kubernetes ワークロードで使用するプライベートレジストリ設定。
- Precision Time Protocolの設定(41.13項「Precision Time Protocol」): 実行中のPTP通信事業者プロファイルの設定ファイル。

41.1 リアルタイム用のカーネルイメージ

リアルタイムカーネルイメージは必ずしも標準カーネルより優れているとは限りません。リアルタイムカーネルは、特定のユースケース用に調整された別のカーネルです。低レイテンシを実現するために調整されていますが、その結果、スループットが犠牲になります。リアルタイムカーネルは一般的な用途には推奨されませんが、ここでは低レイテンシが重要な要因である通信ワークロード用のカーネルとして推奨されています。

主に4つの機能があります。

- 決定論的実行:
予測可能性の向上 — 高負荷状態でも重要なビジネスプロセスが期限内に確実に完了し、常に高品質なサービスを提供します。高優先度プロセスのために重要なシステムリソースを保護することで、時間に依存するアプリケーションの予測可能性を向上できます。
- 低ジッター:
高度に決定論的な技術に基づいてジッターが低く抑えられているため、アプリケーションと実世界との同期を維持できます。これは、継続的に繰り返し計算を行う必要があるサービスで役立ちます。
- 優先度の継承:
優先度の継承とは、優先度の高いプロセスがある状況において、そのプロセスがタスクを完了するためには優先度の低いプロセスが完了するのを待つ必要がある場合に、優先度の低いプロセスが高優先度を一時的に引き受ける機能です。SUSE Linux Enterprise Real Timeは、ミッションクリティカルなプロセスにおけるこのような優先度の逆転の問題を解決します。
- スレッドの割り込み:
一般的なオペレーティングシステムでは、割り込みモードで実行中のプロセスはプリエンプト可能ではありません。SUSE Linux Enterprise Real Timeでは、このような割り込みをカーネルスレッドでカプセル化して割り込み可能にし、ユーザが定義した高優先度プロセスでハード割り込みとソフト割り込みをプリエンプトできます。

ここでは、SUSE Linux Micro RTのようなリアルタイムイメージをインストール済みの場合、カーネルリアルタイムはすでにインストールされています。リアルタイムカーネルイメージは[SUSE Customer Center \(https://scc.suse.com/\)](https://scc.suse.com/) からダウンロードできます。



注記

リアルタイムカーネルの詳細については、[SUSE Real Time \(https://www.suse.com/products/realtime/\)](https://www.suse.com/products/realtime/) を参照してください。

41.2 低レイテンシとハイパフォーマンスのためのカーネル引数

リアルタイムカーネルを適切に動作させ、通信ワークロードを実行する際には、最高のパフォーマンスと低レイテンシ実現できるようにカーネル引数を設定することが重要です。このユースケースのカーネル引数を設定する際には、いくつかの重要な概念を念頭に置く必要があります。

- SUSEリアルタイムカーネルを使用する際には、kthread_cpusを削除します。このパラメータは、カーネルスレッドが作成されるCPUを制御します。また、PID 1とカーネルモジュール(kmodユーザスペースヘルパー)のロードにどのCPUが許可されるかも制御します。このパラメータは認識されず、影響は何もありません。
- isolcpus、nohz_full、rcu_nocbs、およびirqaffinityを使用して、CPUコアを分離します。CPUピニング技術の包括的なリストについては、「Tunedとカーネル引数によるCPUピニング」(41.3項「Tunedとカーネル引数によるCPUピニング」)の章を参照してください。
- domain、nohz、managed_irqフラグをisolcpusカーネル引数に追加します。何もフラグを指定しない場合、isolcpusはdomainフラグのみを指定するのと同等になります。これにより、指定したCPUがカーネルタスクを含むスケジューリングから分離されます。nohzフラグは指定されたCPUのスケジューラティックを停止し(CPUで実行できるタスクが1つのみの場合)、managed_irqフラグは指定したCPUの管理対象の外部(デバイス)割り込みのルーティングを回避します。NVMeデバイスのIRQラインはカーネルによって完全に管理されており、その結果、非分離(ハウスキーピング)コアにルーティングされることに注意してください。たとえば、このセクションの最後に示されたコマンドラインを実行すると、システムに割り当てられるキューは4つ(および管理/制御キュー)のみとなります。

```
for I in $(grep nvme0 /proc/interrupts | cut -d ':' -f1); do cat /proc/irq/${I}/  
effective_affinity_list; done | column  
39      0      19      20      39
```

この動作は、分離されたコア上で実行中の時間的制約のあるアプリケーションに対するディスク I/O による中断を防止します。ただし、ストレージに重点を置いたワークロードの場合、注意と慎重な設計が必要になる可能性があります。

- ティック(カーネルの定期的なタイマー割り込み)を調整します。
 - skew_tick=1: ティックは同時に発生する場合があります。skew_tick=1を指定すると、すべてのCPUがタイマーティックを正確に同じタイミングで受信するのではなく、わずかにオフセットされたタイミングで発生させます。これにより、システムジッターが軽減され、より一貫性があり低い割り込み応答時間(レイテンシに敏感なアプリケーションにとって不可欠な要件)が可能になります。
 - nohz=on: アイドル状態のCPUにおける定期的なタイマーティックを停止します。
 - nohz_full=<cpu-cores>: リアルタイムアプリケーション専用の指定されたCPUにおける定期的なタイマーティックを停止します。
- mce=offを指定して、マシンチェック例外(MCE)の処理を無効にします。MCEはプロセッサによって検出されるハードウェアエラーであり、無効にすることで、ノイズの多いログを回避できます。
- nowatchdogを追加して、タイマーのハード割り込みコンテキストで実行されるタイマーとして実装されるソフトロックアップウォッチドッグを無効にします。有効期限が切れると(すなわち、ソフトロックアップが検出されると)、(ハード割り込みコンテキストで)警告が出力され、あらゆるレイテンシターゲットを実行します。有効期限が切れていない場合でも、タイマーリストに追加され、タイマー割り込みのオーバーヘッドがわずかに増加します。このオプションは、NMIウォッチドッグも無効にするため、NMIが干渉できなくなります。
- nmi_watchdog=0は、NMI (Non-Maskable Interrupt)ウォッチドッグを無効にします。nowatchdogが使用される場合は省略できます。
- RCU (Read-Copy-Update)は、共有データに対して多数のリーダが同時にロックなしでアクセスできるようにするカーネルメカニズムです。RCUコールバックは、一定の「猶予期間」後にトリガされる関数で、すべての以前のリーダが終了していることを確認し、古いデータを安全に再利用できるようにします。SUSEでは、特に機密性の高いワー

クロード向けにRCUを微調整し、これらのコールバックを専用(ピンングされた)CPUからオフロードすることで、カーネル操作が重要な時間的制約のあるタスクと干渉しないようにしています。

- `rcu_nocbs`でピンングされたCPUを指定し、RCUコールバックがそのCPU上で実行されないようにします。これにより、リアルタイムワークロードのジッターとレイテンシを軽減できます。
 - `rcu_nocb_poll`は、コールバック処理が必要かどうかを確認するために、no-callback CPUを定期的に「ポーリング」させます。これにより、割り込みのオーバーヘッドを削減できます。
 - `rcupdate.rcu_cpu_stall_suppress=1`は、RCU CPUストール警告を抑制します。これは、高負荷のリアルタイムシステムでは偽陽性となる場合があります。
 - `rcupdate.rcu_expedited=1`は、RCU操作の猶予期間を短縮し、読み取り側のクリティカルセクションの応答性を向上させます。
 - `rcupdate.rcu_normal_after_boot=1`: `rcu_expedited`とともに使用する場合、システム起動後にRCUが通常の(非迅速)動作に戻ることができます。
 - `rcupdate.rcu_task_stall_timeout=0`は、RCUタスクのストール検出機能を無効にし、長時間実行されるRCUタスクによる潜在的な警告やシステム停止を防止します。
 - `rcutree.kthread_prio=99`は、RCUコールバックカーネルスレッドの優先度を可能な限り最高(99)に設定し、必要に応じてRCUコールバックがスケジュールされ、迅速に処理されるようにします。
-
- Metal3とCluster APIに`ignition.platform.id=openstack`を追加し、クラスタのプロビジョニング/プロビジョニング解除が正常に実行されるようにします。これは、Openstack Ironic由来のMetal3 Pythonエージェントによって使用されます。
 - `intel_pstate=passive`を削除します。このオプションは、`intel_pstate`を汎用cpufreqガバナーと連携するように設定しますが、この連携を行うには、副作用として、ハードウェア管理P状態(HWP)を無効にします。ハードウェアのレイテンシを削減するため、このオプションはリアルタイムワークロードには推奨されません。
 - `intel_idle.max_cstate=0 processor.max_cstate=1`を`idle=poll`に置き換えます。C-Stateの遷移を回避するには、`idle=poll`オプションを使用してC-Stateの遷移を無効にし、CPUを最高のC-Stateに維持します。`intel_idle.max_cstate=0`オプションは、`intel_idle`を無効にするた

め、`acpi_idle`が使用され、`acpi_idle.max_cstate=1`が`acpi_idle`の最大のC-stateを設定します。AMD64/Intel 64アーキテクチャでは、最初のACPI C-Stateは常にP0LLですが、`poll_idle()`関数を使用しており、定期的にクロックを読み取ることで、タイムアウト後に`do_idle()`でメインループを再起動する際に、若干のレイテンシが発生する可能性があります(これにはTIF_POLLタスクフラグのクリアと設定も含まれます)。これに対して、`idle=poll`はタイトなループで実行され、タスクが再スケジュールされるのをビジーウェイトします。これにより、アイドル状態から抜け出すまでのレイテンシが最小化されますが、その代償としてCPUがアイドルスレッドでフルスピードで稼働し続けることになります。

- BIOSのC1Eを無効にします。このオプションは、BIOSでC1E状態を無効にし、アイドル時にCPUがC1E状態になるのを回避します。C1E状態は、CPUがアイドル状態のときにレイテンシが発生される可能性のある低電力状態です。

このドキュメントの残りの部分では、追加のパラメータ(Huge PageやIOMMUなど)について説明します。

以下は、前述の調整を含む32コアのIntelサーバのカーネル引数の例です。

```
$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-6.4.0-9-rt root=UUID=77b713de-5cc7-4d4c-8fc6-f5eca0a43cf9
skew_tick=1 rd.timeout=60 rd.retry=45 console=ttyS1,115200 console=tty0
default_hugepagesz=1G hugepagesz=1G hugepages=40 hugepagesz=2M hugepages=0
ignition.platform.id=openstack intel_iommu=on iommu=pt irqaffinity=0,31,32,63
isolcpus=domain,nohz,managed_irq,1-30,33-62 nohz_full=1-30,33-62 nohz=on
mce=off net.ifnames=0 nosoftlockup nowatchdog nmi_watchdog=0 quiet rcu_nocb_poll
rcu_nocbs=1-30,33-62 rcupdate.rcu_cpu_stall_suppress=1 rcupdate.rcu_expedited=1
rcupdate.rcu_normal_after_boot=1 rcupdate.rcu_task_stall_timeout=0
rcutree.kthread_prio=99 security=selinux selinux=1 idle=poll
```

以下は、64コアのAMDサーバのもう1つの設定例です。128個の論理プロセッサ(0-127)のうち、最初の8コア(0-7)はハウスキーピング用に指定され、残りの120コア(8-127)はアプリケーション用に固定されます。

```
$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-6.4.0-9-rt root=UUID=575291cf-74e8-42cf-8f2c-408a20dc00b8
skew_tick=1 console=ttyS1,115200 console=tty0 default_hugepagesz=1G hugepagesz=1G
hugepages=40 hugepagesz=2M hugepages=0 ignition.platform.id=openstack amd_iommu=on
iommu=pt irqaffinity=0-7 isolcpus=domain,nohz,managed_irq,8-127 nohz_full=8-127
rcu_nocbs=8-127 mce=off nohz=on net.ifnames=0 nowatchdog nmi_watchdog=0 nosoftlockup
quiet rcu_nocb_poll rcupdate.rcu_cpu_stall_suppress=1 rcupdate.rcu_expedited=1
rcupdate.rcu_normal_after_boot=1 rcupdate.rcu_task_stall_timeout=0
rcutree.kthread_prio=99 security=selinux selinux=1 idle=poll
```

41.3 Tunedとカーネル引数によるCPUピニング

tunedは、さまざまな事前定義済みプロファイルを使用して、システム状態を監視し、パフォーマンスを最適化するシステムチューニングツールです。主要な機能は、リアルタイムアプリケーションのなどの特定のワークロードに対してCPUコアを分離する機能です。この機能により、OSがこれらのコアを使用するのを防止し、潜在的にレイテンシが増加するのを回避します。

この機能を有効にして設定するには、まず、分離するCPUコアのプロファイルを作成します。この例では、64コアのうち、60コア(1-30、33-62)をアプリケーション専用にし、残りの4コアをハウスキーピングに使用します。分離されたCPUの設計は、リアルタイムアプリケーションに大きく依存することに注意してください。

```
$ 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.
```

次に、GRUBオプションを変更して、CPUコアと、CPUの使用法に関するその他の重要なパラメータを分離する必要があります。現在のハードウェア仕様で次のオプションをカスタマイズすることが重要です。

パラメータ	値	説明
isolcpus	domain、nohz、 managed_irq、1-30、33-62	コア1-30と3-62を分離します。 <u>domain</u> は、これらのCPUが分離ドメインに属していることを示します。 <u>nohz</u> は、これらの分離されたCPUがアイドル状態である場合にティックレス操作を有効にし、割り込みを減らします。 <u>managed_irq</u> は、ピニングされたCPUがIRQの対象とならないように分離します。これは、ほとんどのIRQをハウスキーピングコアに指定する <u>irqaffinity=0-7</u> を考慮しています。

パラメータ	値	説明
skew_tick	1	このオプションを使用すると、カーネルは分離されたCPU全体でタイマー割り込みをずらすことができます。
nohz	on	有効にすると、カーネルの定期的なタイマー割り込み(「ティック」)は、アイドル状態のCPUコアで停止します。これにより、主にハウスキープング用CPU(0、31、32、63)にメリットをもたらします。これにより、電力が節約され、汎用コアでの不要なウェイクアップが削減されます。
nohz_full	1-30、33-62	分離されたコアの場合、これによりティックが停止され、CPUが単一のアクティブなタスクを実行している場合でも停止されます。つまり、CPUは完全なティックレスモード(または「dyntick」)で動作します。カーネルは、実際に必要な場合にのみタイマー割り込みを発生させます。
rcu_nocbs	1-30、33-62	このオプションは、指定のCPUコアからRCUコールバック処理をオフロードします。
rcu_nocb_poll		このオプションを設定すると、no-RCU-callback CPUは、他のCPUによって明示的にウェイクアップされるので

パラメータ	値	説明
		はなく、コールバック処理が必要かどうかを確認するために定期的に「ポーリング」します。これにより、割り込みオーバーヘッドを削減できます。
irqaffinity	0、31、32、63	このオプションを使用すると、カーネルはハウスキーピング用コアに割り込みを実行できるようになります。
idle	poll	これにより、アイドル状態から抜け出すまでのレイテンシが最小化されますが、その代償として、CPUがアイドルスレッドでフルスピードで稼働し続けることになります。
nmi_watchdog	0	このオプションはNMIウォッチドッグのみを無効にします。nowatchdogが設定されている場合は省略できます。
nowatchdog		このオプションは、タイマーのハード割り込みコンテキストで実行されるタイマーとして実装されるソフトロックアップウォッチドッグを無効にします。

次のコマンドでGRUB設定を変更し、上記の変更を次回ブート時に適用します。

上記パラメータを使用して /etc/default/grub ファイルを編集します。ファイルは次のようになります。

```
GRUB_CMDLINE_LINUX="BOOT_IMAGE=/boot/vmlinuz-6.4.0-9-rt root=UUID=77b713de-5cc7-4d4c-8fc6-f5eca0a43cf9 skew_tick=1 rd.timeout=60 rd.retry=45 console=ttyS1,115200 console=tty0
default_hugepagesz=1G hugepagesz=1G hugepages=40 hugepagesz=2M hugepages=0
ignition.platform.id=openstack intel_iommu=on iommu=pt irqaffinity=0,31,32,63"
```

```
isolcpus=domain,nohz,managed_irq,1-30,33-62 nohz_full=1-30,33-62 nohz=on  
mce=off net.ifnames=0 nosoftlockup nowatchdog nmi_watchdog=0 quiet rcu_nocb_poll  
rcu_nocbs=1-30,33-62 rcupdate.rcu_cpu_stall_suppress=1 rcupdate.rcu_expedited=1  
rcupdate.rcu_normal_after_boot=1 rcupdate.rcu_task_stall_timeout=0  
rcutree.kthread_prio=99 security=selinux selinux=1 idle=poll"
```

GRUB設定を更新します。

```
$ transactional-update grub.cfg  
$ reboot
```

再起動後にパラメータが適用されていることを検証するには、次のコマンドを使用してカーネルコマンドラインを確認できます。

```
$ cat /proc/cmdline
```

CPU設定を調整するために使用可能な別のスクリプトがあります。これは基本的には次の手順を実行します。

- CPUガバナーを パフォーマンス に設定します。
- 分離されたCPUへのタイマーのマイグレーションの設定を解除します。
- kdaemonスレッドをハウスキープ用CPUに移行します。
- 分離したCPUレイテンシを可能な限り低い値に設定します。
- vmstatの更新を300秒に遅延させます。

スクリプトは、[SUSE Telco Cloud サンプルリポジトリ \(https://raw.githubusercontent.com/suse-edge/atip/refs/heads/release-3.4/telco-examples/edge-clusters/dhcp-less/eib/custom/files/performance-settings.sh\)](https://raw.githubusercontent.com/suse-edge/atip/refs/heads/release-3.4/telco-examples/edge-clusters/dhcp-less/eib/custom/files/performance-settings.sh) で入手できます。

41.4 CNI設定

41.4.1 Cilium

CiliumはSUSE Telco CloudのデフォルトのCNIプラグインです。RKE2クラスタでCiliumをデフォルトのプラグインとして有効にするには、/etc/rancher/rke2/config.yaml ファイルに次の設定が必要です。

```
cni:  
- cilium
```

これはコマンドライン引数でも指定できます。具体的には、/etc/systemd/system/rke2-server ファイルのサーバの行に --cni=cilium を追加します。

次のセクション(41.5項「SR-IOV」(444 ページ))で説明するSR-IOV Network Operatorを使用するには、Multusとともに、CiliumやCalicoなどの別のCNIプラグインをセカンダリプラグインとして使用します。

```
cni:
- multus
- cilium
```

41.4.2 Calico

Calicoは、SUSE Edge for Telco用の別のCNIプラグインです。RKE2クラスタ上でデフォルトプラグインとしてCalicoを有効にするには、`/etc/rancher/rke2/config.yaml`ファイルに次の設定が必要です。

```
cni:
- calico
```

これは、コマンドライン引数で指定することもできます。つまり、`--cni=calico`を`/etc/systemd/system/rke2-server`ファイルのサーバ行に追加します。

次のセクション(41.5項「SR-IOV」(444 ページ))で説明するSR-IOV Network Operatorを使用するには、Multusとともに、CiliumやCalicoなどの別のCNIプラグインをセカンダリプラグインとして使用します。

```
cni:
- multus
- calico
```



注記

CNIプラグインの詳細については、「[Network Options \(ネットワークオプション\)](https://docs.rke2.io/install/network_options) (https://docs.rke2.io/install/network_options)」を参照してください。

41.4.3 Bond CNI

一般的に、ボンディングは複数のネットワークインタフェースを単一の論理的な「ボンディング」インタフェースに集約する方法を提供します。これは通常、冗長なネットワーキングパスを導入することでサービスの可用性を向上させるために使用されますが、特定のボンディングモードで帯域幅を増やすためにも使用できます。multusと組み合わせてBond CNIプラグインでは、以下のCNIプラグインがサポートされています。

- MACVLAN
- ホストデバイス
- SR-IOV

41.4.3.1 Bond CNIとMACVLAN

Bond CNIプラグインとMACVLANを組み合わせて使用するには、コンテナ内に2つの空きインタフェースが必要です。以下の例では、「enp8s0」と「enp9s0」を使用しています。まず、これらのインタフェースに対するネットワークアタッチメント定義を作成します。

NetworkAttachmentDefinition enp8s0

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: enp8s0-conf
spec:
  config: '{
    "cniVersion": "0.3.1",
    "plugins": [
      {
        "type": "macvlan",
        "capabilities": { "ips": true },
        "master": "enp8s0",
        "mode": "bridge",
        "ipam": {}
      }, {
        "capabilities": { "mac": true },
        "type": "tuning"
      }
    ]
  }'
```

NetworkAttachmentDefinition enp9s0

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: enp9s0-conf
spec:
  config: '{
    "cniVersion": "0.3.1",
    "plugins": [
      {
        "type": "macvlan",
        "capabilities": { "ips": true },
        "master": "enp9s0",
```

```

        "mode": "bridge",
        "ipam": {}
    }, {
        "capabilities": { "mac": true },
        "type": "tuning"
    }
]
}'

```

この後で、ボンド自体のネットワークアタッチメント定義を追加します。

NetworkAttachmentDefinition bond

```

apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bond-net1
spec:
  config: '{
    "type": "bond",
    "cniVersion": "0.3.1",
    "name": "bond-net1",
    "mode": "active-backup",
    "failOverMac": 1,
    "linksInContainer": true,
    "miimon": "100",
    "mtu": 1500,
    "links": [
      {"name": "net1"},
      {"name": "net2"}
    ],
    "ipam": {
      "type": "static",
      "addresses": [
        {
          "address": "192.168.200.100/24",
          "gateway": "192.168.200.1"
        }
      ],
      "subnet": "192.168.200.0/24",
      "routes": [{
        "dst": "0.0.0.0/0"
      }]
    }
  }
}'

```

ここでのIPアドレス割り当ては静的であり、ボンドのアドレスを/24ネットワーク上の「192.168.200.100」と定義し、ゲートウェイはネットワークで最初に利用可能なアドレス上に配置されます。ボンドのネットワークアタッチメントでは、必要なボンドのタイプも定義します。この場合はactive-backupです。

このボンドを使用するには、ポッドがすべてのインタフェースについて認識している必要があります。ポッド定義例は次のようになります。

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
{"name": "enp8s0-conf",
"interface": "net1"
},
{"name": "enp9s0-conf",
"interface": "net2"
},
{"name": "bond-net1",
"interface": "bond0"
}]'
spec:
  restartPolicy: Never
  containers:
  - name: bond-test
    image: alpine:latest
    command:
      - /bin/sh
      - "-c"
      - "sleep 60m"
    imagePullPolicy: IfNotPresent
```

アノテーションがすべてのネットワークをどのように参照しているか、およびインタフェース「enp8s0→net1」と「enp9s0→net2」間のマッピングをどのように定義しているかに注意してください。

41.4.3.2 Bond CNIとホストデバイス

Bond CNIプラグインとホストデバイスを組み合わせて使用するには、ホスト上に2つの空きインタフェースが必要です。これらのインタフェースはコンテナにマッピングされます。以下の例では、「enp8s0」と「enp9s0」を使用しています。まず、これらのインタフェースに対するネットワークアタッチメント定義を作成します。

NetworkAttachmentDefinition enp8s0

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: enp8s0-hostdev
```

```
spec:
  config: '{
    "cniVersion": "0.3.1",
    "plugins": [
      {
        "type": "host-device",
        "name": "host0",
        "device": "enp8s0",
        "ipam": {}
      }
    ]
  }'
```

NetworkAttachmentDefinition enp9s0

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: enp9s0-hostdev
spec:
  config: '{
    "cniVersion": "0.3.1",
    "plugins": [
      {
        "type": "host-device",
        "name": "host0",
        "device": "enp9s0",
        "ipam": {}
      }
    ]
  }'
```

この後で、ボンド自体のネットワークアタッチメント定義を追加します。これはMACVLANの使用例と同様です。

NetworkAttachmentDefinition bond

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bond-net1
spec:
  config: '{
    "type": "bond",
    "cniVersion": "0.3.1",
    "name": "bond-net1",
    "mode": "active-backup",
    "failOverMac": 1,
    "linksInContainer": true,
    "miimon": "100",
    "mtu": 1500,
    "links": [
      {"name": "net1"},
    ]
  }'
```

```

        {"name": "net2"}
    ],
    "ipam": {
        "type": "static",
        "addresses": [
            {
                "address": "192.168.200.100/24",
                "gateway": "192.168.200.1"
            }
        ],
        "subnet": "192.168.200.0/24",
        "routes": [{
            "dst": "0.0.0.0/0"
        }]
    }
}
}'

```

ここでのIPアドレス割り当ては静的であり、ボンドのアドレスを/24ネットワーク上の「192.168.200.100」と定義し、ゲートウェイはネットワークで最初に利用可能なアドレス上に配置されます。ボンドのネットワークアタッチメントでは、ボンドのタイプを定義します。この場合はactive-backupです。

このボンドを使用するには、ポッドがすべてのインタフェースについて認識する必要があります。ホストデバイスとのボンドのポッド定義例は次のようになります。

```

apiVersion: v1
kind: Pod
metadata:
  name: test-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
{"name": "enp8s0-hostdev",
"interface": "net1"
},
{"name": "enp9s0-hostdev",
"interface": "net2"
},
{"name": "bond-net1",
"interface": "bond0"
}]'
spec:
  restartPolicy: Never
  containers:
  - name: bond-test
    image: alpine:latest
    command:
      - /bin/sh
      - "-c"
      - "sleep 60m"
    imagePullPolicy: IfNotPresent

```

41.4.3.3 Bond CNIとSR-IOV

Bond CNIをSR-IOVと組み合わせて使用するのは非常に簡単です。SR-IOVの設定方法の詳細については、[41.5項「SR-IOV」](#)を参照してください。そこに記載されているとおり、`SriovNetworkNodePolicies`を作成し、`resourceNames`や仮想機能の数などを定義する必要があります。`resourceNames`は、ポッド定義でインタフェースとして使用されるSriovNetworkによって使用されます。ボンドの定義は、MACVLANとホストデバイスの場合とまったく同じです。



注記

Bond CNIとSR-IOVの組み合わせは、カーネルドライバを使用するSRIOV仮想機能(VF)にのみ適用されます。DPDKワークロードで使用されるようなユーザスペースドライバのVFは、Bond CNIでボンディングできません。

41.5 SR-IOV

SR-IOVを使用すると、ネットワークアダプタなどのデバイスで、そのリソースへのアクセスをさまざまなPCIeハードウェア機能の間で分離することができます。[SR-IOV](#)をデプロイするにはさまざまな方法がありますが、ここでは2つの方法を示します。

- オプション1: [SR-IOV CNIデバイスプラグイン](#)と設定マップを使用して適切に設定する。
- オプション2 (推奨): Rancher Primeから[SR-IOV Helmチャート](#)を使用してこのデプロイメントを簡単に行えるようにする。

オプション1 - SR-IOV CNIデバイスプラグインと設定マップをインストールして適切に設定する

- デバイスプラグインの設定マップを準備する

設定マップに入力する情報を`lspci`コマンドから取得します。

```
$ 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)
```

設定マップはJSONファイルで構成され、このファイルで、フィルタを使用して検出を行うデバイスを記述し、インタフェースのグループを作成します。フィルタとグループを理解することが重要です。フィルタはデバイスを検出するために使用され、グループはインタフェースを作成するために使用されます。

フィルタを設定することもできます。

- vendorID: 8086 (Intel)
- deviceID: 0d5c (アクセラレータカード)
- driver: vfio-pci (ドライバ)
- pfNames: p2p1 (物理インタフェース名)

フィルタを設定して、より複雑なインタフェース構文に一致させることもできます。次に例を示します。

- pfNames: ["eth1#1,2,3,4,5,6"] または [eth1#1-6] (物理インタフェース名)

グループに関連して、FECカード用のグループを1つと、Intelカード用のグループを1つ作成し、さらに、ユースケースに応じてプレフィックスを作成することもできます。

- resourceName: pci_sriov_net_bh_dpdk
- resourcePrefix: Rancher.io

リソースグループを検出して作成し、一部のVFをPodに割り当てる組み合わせは多数あります。



注記

フィルタとグループの詳細については、「[sriov-network-device-plugin \(sr-iovネットワークデバイスプラグイン\)](https://github.com/k8snetworkplumbingwg/sriov-network-device-plugin) (<https://github.com/k8snetworkplumbingwg/sriov-network-device-plugin>)」を参照してください。

フィルタとグループを設定して、ハードウェアとユースケースに応じたインタフェースに一致させると、使用する例が次の設定マップに表示されます。

```
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"]
          }
        }
      ]
    }
```

- daemonset ファイルを準備して、デバイスプラグインをデプロイします。

このデバイスプラグインは、複数のアーキテクチャ(arm、amd、ppc64le)をサポートしています。したがって、同じファイルを異なるアーキテクチャに使用して、各アーキテクチャに複数の daemonset をデプロイできます。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sriov-device-plugin
  namespace: kube-system
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: kube-sriov-device-plugin-amd64
  namespace: kube-system
  labels:
    tier: node
    app: sriovdp
spec:
  selector:
    matchLabels:
      name: sriov-device-plugin
  template:
    metadata:
      labels:
        name: sriov-device-plugin
        tier: node
        app: sriovdp
    spec:
      hostNetwork: true
      nodeSelector:
        kubernetes.io/arch: amd64
      tolerations:
        - key: node-role.kubernetes.io/master
          operator: Exists
          effect: NoSchedule
      serviceAccountName: sriov-device-plugin
      containers:
        - name: kube-sriovdp
          image: rancher/hardened-sriov-network-device-plugin:v3.7.0-build20240816
          imagePullPolicy: IfNotPresent
          args:
            - --log-dir=sriovdp
            - --log-level=10
          securityContext:
            privileged: true
          resources:
            requests:
```

```

      cpu: "250m"
      memory: "40Mi"
    limits:
      cpu: 1
      memory: "200Mi"
    volumeMounts:
    - name: devicesock
      mountPath: /var/lib/kubelet/
      readOnly: false
    - name: log
      mountPath: /var/log
    - name: config-volume
      mountPath: /etc/pcidp
    - name: device-info
      mountPath: /var/run/k8s.cni.cncf.io/devinfo/dp
    volumes:
    - name: devicesock
      hostPath:
        path: /var/lib/kubelet/
    - name: log
      hostPath:
        path: /var/log
    - name: device-info
      hostPath:
        path: /var/run/k8s.cni.cncf.io/devinfo/dp
        type: DirectoryOrCreate
    - name: config-volume
      configMap:
        name: sriovdp-config
        items:
        - key: config.json
          path: config.json

```

- 設定マップとdaemonsetを適用すると、デバイスプラグインがデプロイされ、インタフェースが検出されてPodで使用できるようになります。

```

$ kubectl get pods -n kube-system | grep sriov
kube-system  kube-sriov-device-plugin-amd64-twjfl  1/1  Running  0   2m

```

- Podで使用するノードでインタフェースが検出されて利用可能であることを確認します。

```

$ 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"  
}
```

- FECはintel.com/intel_fec_5gで、値は1です。
- Helmチャートを使用せずに、デバイスプラグインと設定マップを使用してデプロイした場合、VFは、intel.com/intel_sriov_oduまたはintel.com/intel_sriov_oruです。

！ 重要

ここにインタフェースがない場合、そのインタフェースをPodで使用することはできないため、続行しても意味がありません。まず、設定マップとフィルタを確認して問題を解決してください。

オプション2 (推奨) - Rancherを使用し、SR-IOV CNIおよびデバイスプラグイン用のHelmチャートを使用したインストール

- Helmがない場合は入手します。

```
$ curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
```

- SR-IOVをインストールします。

```
helm install sriov-crd oci://registry.suse.com/edge/charts/sriov-crd -n sriov-network-operator  
helm install sriov-network-operator oci://registry.suse.com/edge/charts/sriov-network-operator -n sriov-network-operator
```

- デプロイしたリソースのcrdとPodを確認します。

```
$ kubectl get crd  
$ kubectl -n sriov-network-operator get pods
```

- ノードのラベルを確認します。

すべてのリソースが実行されていると、ラベルがノードに自動的に表示されます。

```
$ kubectl get nodes -oyaml | grep feature.node.kubernetes.io/network-sriov.capable  
  
feature.node.kubernetes.io/network-sriov.capable: "true"
```

- daemonsetを確認し、新しいsriov-network-config-daemonおよびsriov-rancher-nfd-workerがアクティブで準備できていることを確認します。

```
$ kubectl get daemonset -A
```

NAMESPACE	NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR
calico-system	calico-node	1	1	1	1	1	kubernetes.io/os=linux
sriov-network-operator	sriov-network-config-daemon	1	1	1	1	1	feature.node.kubernetes.io/network-sriov.capable=true
sriov-network-operator	sriov-rancher-nfd-worker	1	1	1	1	1	<none>
kube-system	rke2-ingress-nginx-controller	1	1	1	1	1	kubernetes.io/os=linux
kube-system	rke2-multus-ds	1	1	1	1	1	kubernetes.io/arch=amd64,kubernetes.io/os=linux

数分後(更新に最大で10分かかる可能性があります)、ノードが検出されて、SR-IOVの機能が設定されます。

```
$ kubectl get sriovnetworknodestates.sriovnetwork.openshift.io -A
```

NAMESPACE	NAME	AGE
sriov-network-operator	xr11-2	83s

- 検出されたインタフェースを確認します。

検出されたインタフェースはネットワークデバイスのPCIアドレスである必要があります。この情報は、ホストでlspciコマンドを使用して確認します。

```
$ kubectl get sriovnetworknodestates.sriovnetwork.openshift.io -n kube-system -o yaml
```

```
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: ""
```



注記

ここでインタフェースが検出されていない場合は、インタフェースが次の設定マップに存在することを確認してください。

```
$ kubectl get cm supported-nic-ids -oyaml -n sriov-network-operator
```

ここにデバイスがない場合は、設定マップを編集して、検出すべき適切な値を追加します(sriov-network-config-daemonデーモンセットの再起動が必要になります)。

- NetworkNodeポリシーを作成してVFを設定します。

VF (numVfs)がデバイス(rootDevices)から作成され、ドライバdeviceTypeとMTUが設定されます。



注記

`resourceName`フィールドには特殊文字を含めないでください。また、このフィールドはクラスタ全体で一意である必要があります。この例では、`dpdk`を`sr-iov`と組み合わせて使用するため、`deviceType: vfio-pci`を使用しています。`dpdk`を使用しない場合は、`deviceType`を`deviceType: netdevice` (デフォルト値)にする必要があります。

```
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
```

- 設定を検証します。

```
$ kubectl get $(kubectl get nodes -oname) -o jsonpath='{.status.allocatable}' | jq
{
  "cpu": "64",
  "ephemeral-storage": "256196109726",
  "hugepages-1Gi": "60Gi",
  "hugepages-2Mi": "0",
  "intel.com/intel_fec_5g": "1",
  "memory": "200424836Ki",
  "pods": "110",
  "rancher.io/intelNicsDpdk": "8"
}
```

- `sr-iov`ネットワークを作成します(別のネットワークが必要な場合のオプション)。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: network-dpdk
  namespace: sriov-network-operator
```

```
spec:
  ipam: |
    {
      "type": "host-local",
      "subnet": "192.168.0.0/24",
      "rangeStart": "192.168.0.20",
      "rangeEnd": "192.168.0.60",
      "routes": [{
        "dst": "0.0.0.0/0"
      }],
      "gateway": "192.168.0.1"
    }
  vlan: 500
  resourceName: intelnicsDpdk
```

- 作成されたネットワークを確認します。

```
$ kubectl get network-attachment-definitions.k8s.cni.cncf.io -A -oyaml

apiVersion: v1
items:
- apiVersion: k8s.cni.cncf.io/v1
  kind: NetworkAttachmentDefinition
  metadata:
    annotations:
      k8s.v1.cni.cncf.io/resourceName: rancher.io/intelnicsDpdk
    creationTimestamp: "2023-06-08T11:22:27Z"
    generation: 1
    name: network-dpdk
    namespace: sriov-network-operator
    resourceVersion: "13124"
    uid: df7c89f5-177c-4f30-ae72-7aef3294fb15
  spec:
    config: '{ "cniVersion":"0.4.0", "name":"network-dpdk", "type":"sriov", "vlan":500, "vlanQoS":0, "ipam":{"type":"host-local", "subnet":"192.168.0.0/24", "rangeStart":"192.168.0.10", "rangeEnd":"192.168.0.60", "routes":[{"dst":"0.0.0.0/0"}]}, "gateway":"192.168.0.1"}'
  kind: List
  metadata:
    resourceVersion: ""
```


41.6 DPDK

DPDK (データプレーン開発キット)は、パケットの高速処理用の一連のライブラリとドライバです。DPDKは、広範なCPUアーキテクチャ上で実行されるパケット処理ワークロードを高速化するために使用されます。DPDKには、データプレーンライブラリと、以下のために最適化されたネットワークインタフェースコントローラ(NIC)ドライバが含まれています。

1. キューマネージャはロックなしのキューを実装します。
2. バッファマネージャは固定サイズのバッファを事前割り当てします。
3. メモリマネージャは、メモリ内にオブジェクトのプールを割り当て、リングを使用してフリーオブジェクトを格納します。オブジェクトがすべてのDRAMチャンネルに均等に分散されるようにします。
4. ポールモードドライバ(PMD)は、非同期通知なしで動作するように設計されているため、オーバーヘッドが軽減されます。
5. パケット処理を開発するためのヘルパーである一連のライブラリとしてのパケットフレームワーク。

次の手順では、DPDKを有効にする方法と、DPDKインタフェースが使用するNICからVFを作成する方法を示します。

- DPDKパッケージをインストールします。

```
$ transactional-update pkg install dpdk dpdk-tools libdpdk-23
$ reboot
```

- カーネルパラメータ:

DPDKを使用するには、ドライバをいくつか使用して、カーネルの特定のパラメータを有効にします。

パラメータ	値	説明
iommu	pt	このオプションを使用すると、DPDKインタフェースにvfiioドライバを使用できます。
intel_iommuまたはamd_iommu	on	このオプションを使用すると、VFにvfiioを使用できます。

これらのパラメータを有効にするには、各パラメータを /etc/default/grub ファイルに追加します。

```
GRUB_CMDLINE_LINUX="BOOT_IMAGE=/boot/vmlinuz-6.4.0-9-rt root=UUID=77b713de-5cc7-4d4c-8fc6-f5eca0a43cf9 skew_tick=1 rd.timeout=60 rd.retry=45 console=ttyS1,115200 console=tty0 default_hugepagesz=1G hugepagesz=1G hugepages=40 hugepagesz=2M hugepages=0 ignition.platform.id=openstack intel_iommu=on iommu=pt irqaffinity=0,31,32,63 isolcpus=domain,nohz,managed_irq,1-30,33-62 nohz_full=1-30,33-62 nohz=on mce=off net.ifnames=0 nosoftlockup nowatchdog nmi_watchdog=0 quiet rcu_nocb_poll rcu_nocbs=1-30,33-62 rcupdate.rcu_cpu_stall_suppress=1 rcupdate.rcu_expedited=1 rcupdate.rcu_normal_after_boot=1 rcupdate.rcu_task_stall_timeout=0 rcutree.kthread_prio=99 security=selinux selinux=1 idle=poll"
```

GRUBの設定を更新し、システムを再起動して変更を適用します。

```
$ transactional-update grub.cfg
$ reboot
```

- vfio-pciカーネルモジュールを読み込み、NICでSR-IOVを有効にします。

```
$ modprobe vfio-pci enable_sriov=1 disable_idle_d3=1
```

- NICから仮想機能(VF)をいくつか作成します。

たとえば、2つの異なるNICに対してVFを作成するには、次のコマンドが必要です。

```
$ echo 4 > /sys/bus/pci/devices/0000:51:00.0/sriov_numvfs
$ echo 4 > /sys/bus/pci/devices/0000:51:00.1/sriov_numvfs
```

- 新しいVFを vfio-pci ドライバにバインドします。

```
$ 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
```

- 設定が正しく適用されたことを確認します。

```
$ dpdk-devbind.py -s
```

```
Network devices using DPDK-compatible driver
```

```
=====
```

```
0000:51:01.0 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci unused=iavf,igb_uio
0000:51:01.1 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci unused=iavf,igb_uio
0000:51:01.2 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci unused=iavf,igb_uio
0000:51:01.3 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci unused=iavf,igb_uio
0000:51:01.0 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci unused=iavf,igb_uio
0000:51:11.1 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci unused=iavf,igb_uio
0000:51:21.2 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci unused=iavf,igb_uio
0000:51:31.3 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci unused=iavf,igb_uio
```

```

Network devices using kernel driver
=====
0000:19:00.0 'BCM57504 NetXtreme-E 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet 1751' if=em1
drv=bnxt_en unused=igb_uio,vfio-pci *Active*
0000:19:00.1 'BCM57504 NetXtreme-E 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet 1751' if=em2
drv=bnxt_en unused=igb_uio,vfio-pci
0000:19:00.2 'BCM57504 NetXtreme-E 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet 1751' if=em3
drv=bnxt_en unused=igb_uio,vfio-pci
0000:19:00.3 'BCM57504 NetXtreme-E 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet 1751' if=em4
drv=bnxt_en unused=igb_uio,vfio-pci
0000:51:00.0 'Ethernet Controller E810-C for QSFP 1592' if=eth13 drv=ice
unused=igb_uio,vfio-pci
0000:51:00.1 'Ethernet Controller E810-C for QSFP 1592' if=rename8 drv=ice
unused=igb_uio,vfio-pci

```

41.7 vRANアクセラレーション(Intel ACC100/ACC200)

4Gから5Gネットワークへの移行に伴い、多くの通信サービスプロバイダが仮想化無線アクセスネットワーク(vRAN)アーキテクチャを採用して、チャンネル容量を増やし、エッジベースのサービスとアプリケーションのデプロイメントを容易にしようとしています。vRANソリューションは、ネットワーク上のリアルタイムのトラフィックと需要の量に応じて容量を柔軟に増減できるため、低レイテンシのサービスを提供するのに理想的です。

4Gおよび5Gで最も計算負荷が高いワークロードの1つがRANレイヤ1 (L1)のFECです。これは、信頼性の低い通信チャンネルやノイズの多い通信チャンネルでのデータ伝送エラーを解消するものです。FEC技術は、4Gまたは5Gデータの一定数のエラーを検出して訂正することで、再送信の必要性を解消します。FECアクセラレーショントランザクションにはセルの状態情報が含まれないため、簡単に仮想化でき、プールするメリットとセルの容易な移行が実現します。

- カーネルパラメータ

vRANアクセラレーションを有効にするには、次のカーネルパラメータを有効にする必要があります(まだ存在しない場合)。

パラメータ	値	説明
iommu	pt	このオプションを使用すると、DPDKインタフェースにvfioを使用できます。

パラメータ	値	説明
intel_iommuまたは amd_iommu	on	このオプションを使用すると、VFにvfioを使用できます。

GRUBファイル/etc/default/grubを変更して、これらのパラメータをカーネルコマンドラインに追加します。

```
GRUB_CMDLINE_LINUX="BOOT_IMAGE=/boot/vmlinuz-6.4.0-9-rt root=UUID=77b713de-5cc7-4d4c-8fc6-f5eca0a43cf9 skew_tick=1 rd.timeout=60 rd.retry=45 console=ttyS1,115200 console=tty0
default_hugepagesz=1G hugepagesz=1G hugepages=40 hugepagesz=2M hugepages=0
ignition.platform.id=openstack intel_iommu=on iommu=pt irqaffinity=0,31,32,63
isolcpus=domain,nohz,managed_irq,1-30,33-62 nohz_full=1-30,33-62 nohz=on
mce=off net.ifnames=0 nosoftlockup nowatchdog nmi_watchdog=0 quiet rcu_nocb_poll
rcu_nocbs=1-30,33-62 rcupdate.rcu_cpu_stall_suppress=1 rcupdate.rcu_expedited=1
rcupdate.rcu_normal_after_boot=1 rcupdate.rcu_task_stall_timeout=0
rcutree.kthread_prio=99 security=selinux selinux=1 idle=poll"
```

GRUBの設定を更新し、システムを再起動して変更を適用します。

```
$ transactional-update grub.cfg
$ reboot
```

再起動後にパラメータが適用されていることを確認するには、コマンドラインを確認します。

```
$ cat /proc/cmdline
```

- vfio-pciカーネルモジュールを読み込み、vRANアクセラレーションを有効にします。

```
$ modprobe vfio-pci enable_sriov=1 disable_idle_d3=1
```

- インタフェース情報Acc100を取得します。

```
$ lspci | grep -i acc
8a:00.0 Processing accelerators: Intel Corporation Device 0d5c
```

- 物理インタフェース(PF)をvfio-pciドライバにバインドします。

```
$ dpdk-devbind.py -b vfio-pci 0000:8a:00.0
```

- 仮想機能(VF)を物理インタフェース(PF)から作成します。

2つのVFをPFから作成し、次の手順に従ってvfio-pciにバインドします。

```
$ echo 2 > /sys/bus/pci/devices/0000:8a:00.0/sriov_numvfs
$ dpdk-devbind.py -b vfio-pci 0000:8b:00.0
```

- 提案された設定ファイルを使用してacc100を設定します。

```
$ 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!
```

- FEC PFから作成した新しいVFを確認します。

```
$ dpdk-devbind.py -s
Baseband devices using DPDK-compatible driver
=====
0000:8a:00.0 'Device 0d5c' drv=vfio-pci unused=
0000:8b:00.0 'Device 0d5d' drv=vfio-pci unused=

Other Baseband devices
=====
0000:8b:00.1 'Device 0d5d' unused=
```

41.8 Huge Page

プロセスがRAMを使用すると、CPUはそのメモリ領域をプロセスが使用中であるとマークします。効率を高めるために、CPUはRAMをチャンクで割り当てます。多くのプラットフォームでは4Kバイトがチャンクのデフォルト値です。これらのチャンクをページと呼び、ディスクなどにスワップできます。

プロセスのアドレススペースは仮想であるため、CPUとオペレーティングシステムは、どのページがどのプロセスに属していて、各ページがどこに保管されているかを覚えておく必要があります。ページ数が多いほど、メモリマッピングの検索に時間がかかります。プロセスが1GBのメモリを使用する場合、検索するエントリは262,144個になります(1GB / 4K)。1つのページテーブルエントリが8バイトを消費する場合、2MB (262,144 * 8)を検索することになります。

最新のCPUアーキテクチャはデフォルトより大きいページをサポートしているので、CPU/OSが検索するエントリが減少します。

- カーネルパラメータ

Huge Pageを有効にするには、次のカーネルパラメータを追加する必要があります。この例では、40個の1Gページを設定していますが、Huge Pageのサイズと正確な数は、アプリケーションのメモリ要件に合わせて調整する必要があります。

パラメータ	値	説明
hugepagesz	1G	このオプションを使用すると、Huge Pageを1Gに設定できます
hugepages	40	前に定義したHuge Pageの数です
default_hugepagesz	1G	Huge Pageを取得するためのデフォルト値です

GRUBファイル`/etc/default/grub`を変更して、これらのパラメータを`GRUB_CMDLINE_LINUX`に追加します。

```
default_hugepagesz=1G hugepagesz=1G hugepages=40 hugepagesz=2M hugepages=0
```

GRUBの設定を更新し、システムを再起動して変更を適用します。

```
$ transactional-update grub.cfg
$ reboot
```

再起動後にパラメータが適用されていることを検証するには、次のコマンドラインを確認できます。

```
$ cat /proc/cmdline
```

- Huge Pageの使用

Huge Pageを使用するには、Huge Pageをマウントする必要があります。

```
$ mkdir -p /hugepages
$ mount -t hugetlbfs nodev /hugepages
```

Kubernetesワークロードをデプロイし、リソースとボリュームを作成します。

```
...
resources:
  requests:
    memory: "24Gi"
    hugepages-1Gi: 16Gi
    intel.com/intel_sriov_oru: '4'
  limits:
    memory: "24Gi"
    hugepages-1Gi: 16Gi
```

```
intel.com/intel_sriov_oru: '4'
...
```

```
...
volumeMounts:
  - name: hugepage
    mountPath: /hugepages
...
volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
...
```

41.9 KubernetesでのCPUピニング

41.9.1 前提条件

こちらのセクション(41.3項「[Tunedとカーネル引数によるCPUピニング](#)」)で説明したパフォーマンスプロファイルに合わせてCPUが調整されていること。

41.9.2 CPUピニング用のKubernetesの設定

RKE2クラスタでCPU管理を実装するためにkubelet引数を設定します。次の例のような設定ブロックを/etc/rancher/rke2/config.yamlファイルに追加します。[kubelet-reserved](#)および[system-reserved](#)引数でハウスキューピング用CPUコアを指定していることを確認します。

```
kubelet-arg:
- "cpu-manager-policy=static"
- "cpu-manager-policy-options=full-pcpus-only=true"
- "cpu-manager-reconcile-period=0s"
- "kubelet-reserved=cpu=0,31,32,63"
- "system-reserved=cpu=0,31,32,63"
```

41.9.3 ワークロードにピニングされたCPUを活用する

kubeletで定義された[静的ポリシー](#)を使ってCPUピニング機能を使用する方法は、ワークロードに対して定義した要求と制限に応じて3つあります。

1. BestEffort QoSクラス: CPUに対して要求または制限を定義していない場合、Podはシステムで利用できる最初のCPUでスケジュールされます。

BestEffort QoSクラスを使用する例を次に示します。

```
spec:
  containers:
  - name: nginx
    image: nginx
```

2. Burstable QoSクラス: CPUに対して要求を定義し、その要求が制限と同じではない場合、またはCPUの要求がない場合。

Burstable QoSクラスを使用する例を次に示します。

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
      requests:
        memory: "100Mi"
```

または

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
        cpu: "2"
      requests:
        memory: "100Mi"
        cpu: "1"
```

3. Guaranteed QoSクラス: CPUに対して要求を定義し、その要求が制限と同じである場合。

Guaranteed QoSクラスを使用する例を次に示します。

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
```



```
memory: "200Mi"
cpu: "2"
requests:
  memory: "200Mi"
  cpu: "2"
```

41.10 NUMA対応のスケジューリング

Non-Uniform Memory AccessまたはNon-Uniform Memory Architecture (NUMA)は、SMP (マルチプロセッサ)アーキテクチャにおいて使用される物理メモリ設計であり、メモリアクセス時間がプロセッサからのメモリの相対的な位置によって異なります。NUMAでは、プロセッサは専用のローカルメモリに、非ローカルメモリ、つまり別のプロセッサにローカルなメモリや複数のプロセッサで共有されているメモリよりも高速にアクセスできます。

41.10.1 NUMAノードの特定

NUMAノードを特定するには、システムで次のコマンドを使用します。

```
$ lscpu | grep NUMA
NUMA node(s):                1
NUMA node0 CPU(s):           0-63
```



注記

この例では、NUMAノードが1つだけあり、64個のCPUが表示されています。

NUMAはBIOSで有効にする必要があります。dmesgにブート時のNUMA初期化レコードがない場合、カーネルリングバッファ内のNUMA関連のメッセージが上書きされた可能性があります。

41.11 MetalLB

MetalLBは、ベアメタルKubernetesクラスタ用のロードバランサの実装であり、L2やBGPなどの標準ルーティングプロトコルをアドバタイズプロトコルとして使用します。ベアメタル環境ではKubernetesサービスタイプLoadBalancerを使用する必要があるため、Kubernetesクラスタ内のサービスを外部に公開するために使用できるのは、ネットワークロードバランサです。

RKE2クラスタでMetalLBを有効にするには、次の手順を実行する必要があります。

- 次のコマンドを使用してMetalLBをインストールします。

```
$ kubectl apply <<EOF -f
apiVersion: helm.cattle.io/v1
kind: HelmChart
metadata:
  name: metallb
  namespace: kube-system
spec:
  chart: oci://registry.suse.com/edge/charts/metallb
  targetNamespace: metallb-system
  version: 304.0.0+up0.14.9
  createNamespace: true
---
apiVersion: helm.cattle.io/v1
kind: HelmChart
metadata:
  name: endpoint-copier-operator
  namespace: kube-system
spec:
  chart: oci://registry.suse.com/edge/charts/endpoint-copier-operator
  targetNamespace: endpoint-copier-operator
  version: 304.0.1+up0.3.0
  createNamespace: true
EOF
```

- IPAddressPoolおよびL2advertisementの設定を作成します。

```
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
```

- VIPを公開するためのエンドポイントサービスを作成します。

```
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
```

- VIPが作成され、MetaLBのPodが実行中であることを確認します。

```
$ kubectl get svc -n default
$ kubectl get pods -n default
```

41.12 プライベートレジストリ設定

Containerdをプライベートレジストリに接続するように設定し、そのプライベートレジストリを使用して各ノードにプライベートイメージをプルできます。

起動時に、RKE2は、registries.yamlファイルが/etc/rancher/rke2/に存在するかどうかを確認し、このファイルで定義されたレジストリを使用するようにcontainerdに指示します。プライベートレジストリを使用するには、このファイルを、レジストリを使用する各ノードにルートとして作成します。

プライベートレジストリを追加するには、ファイル/etc/rancher/rke2/registries.yamlを作成して次の内容を設定します。

```
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
```

または、認証を使用しない場合は次のように設定します。

```
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
```

レジストリの変更を有効にするには、ノード上でRKE2を起動する前にこのファイルを設定するか、または設定した各ノードでRKE2を再起動します。



注記

この詳細については、「[containerd registry configuration rke2 \(containerdレジストリ設定rke2\)](https://documentation.suse.com/cloudnative/rke2/latest/en/install/containerd_registry_configuration.html#_registries_configuration_file) (https://documentation.suse.com/cloudnative/rke2/latest/en/install/containerd_registry_configuration.html#_registries_configuration_file)」を確認してください。

41.13 Precision Time Protocol

Precision Time Protocol (PTP)は、電気電子学会(IEEE)によって開発されたネットワークプロトコルで、コンピュータネットワークにおけるサブマイクロ秒単位の時間同期を可能にします。PTPは、その誕生以来数十年にわたり、多くの業界で利用されてきました。最近では、5G

ネットワークの重要な要素として、通信ネットワークにおける採用が急増しています。比較的シンプルなプロトコルであるものの、アプリケーションに応じて設定が大幅に異なります。そのため、複数のプロファイルが定義され標準化されています。

このセクションでは、通信事業者固有のプロファイルのみについて説明します。そのため、NICにタイムスタンプ機能とPTPハードウェアクロック(PHC)が搭載されていることを前提とします。現在、すべての通信事業者グレードのネットワークアダプタにはハードウェアPTPサポートが付属していますが、以下のコマンドでその機能を確認できます。

```
# ethtool -T p1p1
Time stamping parameters for p1p1:
Capabilities:
    hardware-transmit
    software-transmit
    hardware-receive
    software-receive
    software-system-clock
    hardware-raw-clock
PTP Hardware Clock: 0
Hardware Transmit Timestamp Modes:
    off
    on
Hardware Receive Filter Modes:
    none
    all
```

p1p1を、PTPに使用されるインタフェースの名前に置き換えます。

以下のセクションでは、SUSE Telco CloudにPTPをインストールして設定する方法について具体的に説明しますが、PTPの基本的な概念を理解していることが前提となります。PTPの概要と、SUSE Telco Cloudに含まれる実装については、<https://documentation.suse.com/sles/html/SLES-all/cha-tuning-ntp.html> を参照してください。

41.13.1 PTPソフトウェアコンポーネントのインストール

SUSE Telco Cloudでは、PTP実装がlinuxptpパッケージで提供されています。このパッケージには次の2つのコンポーネントが含まれています。

- ptp4l: NIC上のPHCを制御し、PTPプロトコルを実行するデーモン
- phc2sys: NIC上のPTP同期PHCとシステムクロックとの同期を維持するデーモン

システム同期が完全に機能するには、両方のデーモンが必要であり、セットアップに従って正しく設定する必要があります。詳細については、[41.13.2項「通信事業者のデプロイメント向けPTPの設定」](#)を参照してください。

ダウンストリームクラスタにPTPを統合する最も簡単で最良の方法は、Edge Image Builder (EIB)定義ファイルのpackageListの下にlinuxptpパッケージを追加することです。これにより、クラスタプロビジョニング中にPTPコントロールプレーンソフトウェアが自動的にインストールされます。パッケージのインストールの詳細については、EIBのドキュメント(3.3.4項「RPMパッケージの設定」)を参照してください。

以下に、linuxptpを使用したEIBマニフェストのサンプルを示します。

```
apiVersion: 1.0
image:
  imageType: RAW
  arch: x86_64
  baseImage: {micro-base-rt-image-raw}
  outputImageName: eibimage-slmicro-rt-telco.raw
operatingSystem:
  time:
    timezone: America/New_York
  kernelArgs:
    - ignition.platform.id=openstack
    - net.ifnames=1
  systemd:
    disable:
      - rebootmgr
      - transactional-update.timer
      - transactional-update-cleanup.timer
      - fstrim
      - time-sync.target
    enable:
      - ptp4l
      - phc2sys
  users:
    - username: root
      encryptedPassword: ${ROOT_PASSWORD}
  packages:
    packageList:
      - jq
      - dpdk
      - dpdk-tools
      - libdpdk-23
      - pf-bb-config
      - open-iscsi
      - tuned
      - cpupower
      - linuxptp
    sccRegistrationCode: ${SCC_REGISTRATION_CODE}
```



注記

SUSE Telco Cloudに含まれる`linuxptp`パッケージでは、デフォルトで`ptp4l`および`phc2sys`が有効になりません。プロビジョニング時にシステム固有の設定ファイルがデプロイされている場合は(41.13.3項「Cluster APIの統合」を参照)、これらを有効にする必要があります。上記の例のように、マニフェストの`systemd`セクションに追加して有効にしてください。

EIBのドキュメントで説明されている通常のプロセスに従って、イメージを構築し(3.4項「イメージの構築」)、そのイメージを使用してクラスタをデプロイします。EIBを初めて使用する場合は、第11章「Edge Image Builder」から開始してください。

41.13.2 通信事業者のデプロイメント向けPTPの設定

多くの通信事業者向けアプリケーションでは、わずかな偏差で厳格な位相と時刻同期が求められるため、通信事業者向けの2つのプロファイルが定義されました(ITU-T G.8275.1とITU-T G.8275.2)。これら両プロファイルは、同期メッセージの送信頻度が高く、代替Best Master Clock Algorithm (BMCA)の使用など、他の特徴的な特性を備えています。このような動作は、`ptp4l`が使用する設定ファイルに特定の設定を必要とします。これらの設定は以下のセクションで参照用に提供されています。



注記

- 両セクションは、タイムレシーバー設定の通常のクロックの場合のみを対象としています。
- このようなプロファイルは、適切に計画されたPTPインフラストラクチャで使用する必要があります。
- 特定のPTPネットワークには、追加の設定の調整が必要な場合があります。必要に応じて、提供された例を確認して調整してください。

41.13.2.1 PTPプロファイルITU-T G.8275.1

G.8275.1プロファイルには、次の特徴があります。

- Ethernet上で直接実行し、完全なネットワークサポートが必要です(隣接するノード/スイッチはPTPをサポートしている必要があります)。
- デフォルトのドメイン設定は24です。
- データセットの比較は、G.8275.xアルゴリズムと、priority2以降の localPriority値に基づいています。

次のコンテンツを/etc/ptp4l-G.8275.1.confという名前のファイルにコピーします。

```
# Telecom G.8275.1 example configuration
[global]
domainNumber          24
priority2              255
dataset_comparison    G.8275.x
G.8275.portDS.localPriority  128
G.8275.defaultDS.localPriority 128
maxStepsRemoved       255
logAnnounceInterval   -3
logSyncInterval       -4
logMinDelayReqInterval -4
announceReceiptTimeout 3
serverOnly            0
ptp_dst_mac           01:80:C2:00:00:0E
network_transport      L2
```

ファイルが作成されたら、デーモンが正常に起動するように/etc/sysconfig/ptp4lで参照される必要があります。これを行うには、OPTIONS=行を次のように変更します。

```
OPTIONS="-f /etc/ptp4l-G.8275.1.conf -i $IFNAME --message_tag ptp-8275.1"
```

より正確な説明は次のとおりです。

- -fには、使用する設定ファイルのファイル名が必要です。この場合は/etc/ptp4l-G.8275.1.conf。
- -iには、使用するインタフェースの名前が必要です。\$IFNAMEを実際のインタフェース名に置き換えます。
- --message_tagは、システムログ内のptp4l出力を適切に特定できるようにし、オプションです。

上記の手順が完了したら、ptp4lデーモンを(再)起動する必要があります。

```
# systemctl restart ptp4l
```

次のコマンドを使用してログを調べ、同期ステータスを確認します。

```
# journalctl -e -u ptp4l
```


41.13.2.2 PTPプロファイルITU-T G.8275.2

G.8275.2プロファイルには、次の特徴があります。

- IP上で実行し、完全なネットワークサポートは不要です(隣接するノード/スイッチはPTPをサポートしていない場合があります)。
- デフォルトのドメイン設定は44です。
- データセットの比較は、G.8275.xアルゴリズムと、priority2以降の localPriority値に基づいています。

次のコンテンツを /etc/ptp4l-G.8275.2.conf という名前のファイルにコピーします。

```
# Telecom G.8275.2 example configuration
[global]
domainNumber          44
priority2              255
dataset_comparison     G.8275.x
G.8275.portDS.localPriority  128
G.8275.defaultDS.localPriority 128
maxStepsRemoved        255
logAnnounceInterval    0
serverOnly             0
hybrid_e2e             1
inhibit_multicast_service 1
unicast_listen         1
unicast_req_duration    60
logSyncInterval        -5
logMinDelayReqInterval -4
announceReceiptTimeout  2
#
# Customize the following for slave operation:
#
[unicast_master_table]
table_id              1
logQueryInterval      2
UDPv4                 $PEER_IP_ADDRESS
[$IFNAME]
unicast_master_table  1
```

次のプレースホルダーを必ず置き換えます。

- \$PEER_IP_ADDRESS - 同期を提供するマスタまたは境界クロックなど、通信する次のPTPノードのIPアドレス。
- \$IFNAME - ptp4lにPTPに使用するインタフェースを示します。

ファイルが作成されると、デーモンが正常に起動するように、PTPに使用するインタフェースの名前とともに、`/etc/sysconfig/ptp4l`で参照される必要があります。これを行うには、`OPTIONS=`行を次のように変更します。

```
OPTIONS="-f /etc/ptp4l-G.8275.2.conf --message_tag ptp-8275.2"
```

より正確な説明は次のとおりです。

- `-f`には、使用する設定ファイルのファイル名が必要です。この場合は、`/etc/ptp4l-G.8275.2.conf`です。
- `--message_tag`は、システムログ内のptp4l出力を適切に特定できるようにし、オプションです。

上記の手順が完了したら、`ptp4l`デーモンを(再)起動する必要があります。

```
# systemctl restart ptp4l
```

次のコマンドを使用してログを調べ、同期ステータスを確認します。

```
# journalctl -e -u ptp4l
```

41.13.2.3 phc2sysの設定

必須ではありませんが、`phc2sys`に移る前に、`ptp4l`の設定を完全に完了しておくことをお勧めします。`phc2sys`は設定ファイルを必要とせず、その実行パラメータは`ptp4l`と同様に、`/etc/sysconfig/ptp4l`に存在する`OPTIONS=`変数を通じてのみ制御できます。

```
OPTIONS="-s $IFNAME -w"
```

ここで、`$IFNAME`はシステムクロックのソースとして使用される、`ptp4l`ですでに設定されているインタフェースの名前です。これはソースPHCを識別するために使用されます。

41.13.3 Cluster APIの統合

クラスタが管理クラスタおよびダイレクトネットワークプロビジョニングを通じてデプロイされるときは常に、プロビジョニング時に設定ファイルと`/etc/sysconfig`内の2つの設定変数の両方をホスト上にデプロイできます。以下は、同じG.8275.1設定ファイルをすべてのホスト上にデプロイする、変更された`RKE2ControlPlane`オブジェクトに焦点を当てたクラスタ定義の抜粋です。

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
```

```

metadata:
  name: single-node-cluster
  namespace: default
spec:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3MachineTemplate
    name: single-node-cluster-controlplane
  replicas: 1
  version: ${RKE2_VERSION}
  rolloutStrategy:
    type: "RollingUpdate"
    rollingUpdate:
      maxSurge: 0
  registrationMethod: "control-plane-endpoint"
  serverConfig:
    cni: canal
  agentConfig:
    format: ignition
    cisProfile: cis
    additionalUserData:
      config: |
        variant: fcos
        version: 1.4.0
        systemd:
          units:
            - name: rke2-preinstall.service
              enabled: true
              contents: |
                [Unit]
                Description=rke2-preinstall
                Wants=network-online.target
                Before=rke2-install.service
                ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
                [Service]
                Type=oneshot
                User=root
                ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
                ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/${jq -r .uuid /mnt/
openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
                ExecStart=/bin/sh -c "echo \"node-name: ${jq -r .name /mnt/openstack/
latest/meta_data.json}\" >> /etc/rancher/rke2/config.yaml"
                ExecStartPost=/bin/sh -c "umount /mnt"
                [Install]
                WantedBy=multi-user.target
      storage:
        files:
          - path: /etc/ptp4l-G.8275.1.conf
            overwrite: true
            contents:
              inline: |

```

```

# Telecom G.8275.1 example configuration
[global]
domainNumber                24
priority2                    255
dataset_comparison           G.8275.x
G.8275.portDS.localPriority  128
G.8275.defaultDS.localPriority 128
maxStepsRemoved              255
logAnnounceInterval          -3
logSyncInterval              -4
logMinDelayReqInterval       -4
announceReceiptTimeout        3
serverOnly                    0
ptp_dst_mac                   01:80:C2:00:00:0E
network_transport             L2
mode: 0644
user:
  name: root
group:
  name: root
- path: /etc/sysconfig/ptp4l
  overwrite: true
  contents:
    inline: |
      ## Path:          Network/LinuxPTP
      ## Description:    Precision Time Protocol (PTP): ptp4l settings
      ## Type:           string
      ## Default:        "-i eth0 -f /etc/ptp4l.conf"
      ## ServiceRestart: ptp4l
      #
      # Arguments when starting ptp4l(8).
      #
      OPTIONS="-f /etc/ptp4l-G.8275.1.conf -i $IFNAME --message_tag
ptp-8275.1"
mode: 0644
user:
  name: root
group:
  name: root
- path: /etc/sysconfig/phc2sys
  overwrite: true
  contents:
    inline: |
      ## Path:          Network/LinuxPTP
      ## Description:    Precision Time Protocol (PTP): phc2sys settings
      ## Type:           string
      ## Default:        "-s eth0 -w"
      ## ServiceRestart: phc2sys
      #
      # Arguments when starting phc2sys(8).
      #

```

```
    OPTIONS="-s $IFNAME -w"
    mode: 0644
    user:
      name: root
    group:
      name: root
  kubelet:
    extraArgs:
      - provider-id=metal3://BAREMETALHOST_UUID
    nodeName: "localhost.localdomain"
```

上記の定義は、他の変数に加え、[第42章「完全に自動化されたダイレクトネットワークプロビジョニング」](#)で説明されているように、インタフェース名と他のCluster APIオブジェクトで補完する必要があります。



注記

- このアプローチは、クラスタのハードウェアが均一であり、インタフェース名を含め、すべてのホスト上で同じ設定が必要な場合にのみ便利です。
- 代替アプローチが可能であり、今後のリリースで取り上げる予定です。

この時点で、ホストには機能していて実行中のPTPスタックがあり、PTP役割のネゴシエーションを開始するはずです。

42 完全に自動化されたダイレクトネットワークプロビジョニング

42.1 はじめに

ダイレクトネットワークプロビジョニングは、ダウストリームクラスタのプロビジョニングを自動化できる機能です。この機能は、プロビジョニングするダウストリームクラスタが多数あり、そのプロセスを自動化したい場合に便利です。

管理クラスタ(第40章「[管理クラスタの設定](#)」)は、次のコンポーネントのデプロイメントを自動化します。

- [SUSE Linux Micro RT \(OS\)](#)。ユースケースに応じて、ネットワーキング、ストレージ、ユーザ、カーネル引数などの設定をカスタマイズできます。
- [RKE2 \(Kubernetesクラスタ\)](#)。デフォルトのCNIプラグインは[Cilium](#)です。ユースケースに応じて、特定のCNIプラグイン([Cilium+Multus](#)など)を使用できます。
- [SUSE Storage](#)
- [SUSE Security](#)
- [MetalLB](#)。高可用性マルチノードクラスタのロードバランサとして使用できます。



注記

[SUSE Linux Micro](#)の詳細については、[第9章「SUSE Linux Micro」](#)を参照してください。[RKE2](#)の詳細については、[第15章「RKE2」](#)を参照してください。[SUSE Storage](#)の詳細については、[第16章「SUSE Storage」](#)を参照してください。[SUSE Security](#)の詳細については、[第17章「SUSE Security」](#)を参照してください。

以降のセクションでは、さまざまなダイレクトネットワークプロビジョニングワークフローと、プロビジョニングプロセスに追加できる機能について説明します。

- [42.2項「接続シナリオのダウストリームクラスタイメージの準備」](#)
- [42.3項「エアギャップシナリオ用のダウストリームクラスタイメージの準備」](#)
- [42.4項「ダイレクトネットワークプロビジョニングを使用したダウストリームクラスタのプロビジョニング\(シングルノード\)」](#)

- 42.5項 「ダイレクトネットワークプロビジョニングを使用したダウンストリームクラスタのプロビジョニング(マルチノード)」
- 42.6項 「高度なネットワーク設定」
- 42.7項 「通信機能(DPDK、SR-IOV、CPUの分離、Huge Page、NUMAなど)」
- 42.8項 「プライベートレジストリ」
- 42.9項 「エアギャップシナリオでのダウンストリームクラスタのプロビジョニング」



注記

次のセクションでは、SUSE Telco Cloudを使用してダイレクトネットワークプロビジョニングワークフローの異なるシナリオを準備する方法について説明します。デプロイメントの異なる設定オプション(エアギャップ環境、DHCPおよびDHCPなしのネットワーク、プライベートコンテナレジストリなどを含む)の例については[SUSE Telco Cloudリポジトリ \(https://github.com/suse-edge/atip/tree/release-3.4/telco-examples/edge-clusters\)](https://github.com/suse-edge/atip/tree/release-3.4/telco-examples/edge-clusters)を参照してください。

42.2 接続シナリオのダウンストリームクラスタイメージの準備

Edge Image Builder (第11章 「Edge Image Builder」)を使用して、ダウンストリームクラスタホスト上にプロビジョニングされる、変更されたSLEMicroゴールデンイメージを準備します。

ほとんどの設定はEdge Image Builderを使用して行うことができますが、このガイドではダウンストリームクラスタをセットアップするために必要な最小限の設定について説明します。

42.2.1 接続シナリオの前提条件

- Edge Image Builderを実行するには、[Podman \(https://podman.io\)](https://podman.io)や[Rancher Desktop \(https://rancherdesktop.io\)](https://rancherdesktop.io)などのコンテナランタイムが必要です。
- ゴールデンイメージは、`プロファイルBase-SelfInstall` (リアルタイムカーネルには`Base-RT-SelfInstall`)を使用し、次のガイド第28章 「[Kiwiを使用したSUSE Linux Microの更新イメージの構築](#)」に従って構築されます。このプロセスは両方のアーキテクチャ(x86-64とaarch64)で同じです。



注記

構築するイメージと同じアーキテクチャの構築ホストを使用する必要があります。つまり、aarch64のイメージを構築するにはaarch64の構築ホストを使用する必要があります。、x86-64の場合も同様です(クロス構築は現在のところサポートされていません)。

42.2.2 接続シナリオのイメージの設定

Edge Image Builderを実行すると、そのホストからディレクトリがマウントされるため、ターゲットイメージの定義に使用する設定ファイルを保存するディレクトリ構造を作成する必要があります。

- downstream-cluster-config.yamlはイメージ定義ファイルです。詳細については、[第3章「Edge Image Builderを使用したスタンドアロンクラスタ」](#)を参照してください。
- ゴールデンイメージフォルダには、ガイド[第28章「Kiwiを使用したSUSE Linux Microの更新イメージの構築」](#)に従って生成された出力RAWイメージが含まれます。プロファイルBase-SelfInstall (または、リアルタイムカーネルの場合はBase-RT-SelfInstall) は、base-imagesフォルダにコピー/移動する必要があります。
- networkフォルダはオプションです。詳細については、[42.2.2.6項「高度なネットワーク設定のための追加スクリプト」](#)を参照してください。
- custom/scriptsディレクトリには初回ブート時に実行するスクリプトが含まれます。
 1. 01-fix-growfs.shスクリプトは、デプロイメント時にOSルートパーティションをサイズ変更するために必要です。
 2. 02-performance.shスクリプトはオプションであり、パフォーマンス調整用にシステムを設定するために使用できます。
 3. 03-sriov.shスクリプトはオプションであり、SR-IOV用にシステムを設定するために使用できます。
- custom/filesディレクトリには、イメージ作成プロセス中にイメージにコピーされるperformance-settings.shおよびsriov-auto-filler.shファイルが含まれます。

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



```

└─ scripts/
  │   └─ 01-fix-growfs.sh
  │   └─ 02-performance.sh
  │   └─ 03-sriov.sh
  └─ files/
      └─ performance-settings.sh
      └─ sriov-auto-filler.sh

```

42.2.2.1 ダウンストリームクラスタイメージ定義ファイル

`downstream-cluster-config.yaml` ファイルは、ダウンストリームクラスタイメージの主要な設定ファイルです。次に、Metal³を介したデプロイメントの最小例を示します。

```

apiVersion: 1.3
image:
  imageType: raw
  arch: x86_64
  baseImage: SL-Micro.x86_64-6.1-Base-GM.raw
  outputImageName: eibimage-output-telco.raw
operatingSystem:
  kernelArgs:
    - ignition.platform.id=openstack
    - net.ifnames=1
  systemd:
    disable:
      - rebootmgr
      - transactional-update.timer
      - transactional-update-cleanup.timer
      - fstrim
      - time-sync.target
  users:
    - username: root
      encryptedPassword: $ROOT_PASSWORD
      sshKeys:
        - $USERKEY1
  packages:
    packageList:
      - jq
    sccRegistrationCode: $SCC_REGISTRATION_CODE

```

ここで、`$SCC_REGISTRATION_CODE`は [SUSE Customer Center \(https://scc.suse.com/\)](https://scc.suse.com/) からコピーした登録コードで、パッケージリストには必要な `jq` が含まれています。

`$ROOT_PASSWORD`はルートユーザの暗号化パスワードで、テスト/デバッグに役立ちます。このパスワードは、`openssl passwd -6 PASSWORD` コマンドで生成できます。

運用環境では、`$USERKEY1`を実際のSSHキーに置き換えて、`users`ブロックに追加できるSSHキーを使用することをお勧めします。



注記

`arch: x86_64`は、イメージのアーキテクチャです。arm64アーキテクチャの場合は、`arch: aarch64`を使用します。

`net.ifnames=1`は、[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) を有効にします。

これはmetal3チャートのデフォルト設定と一致しますが、この設定は、設定されたチャートの`predictableNicNames`の値と一致する必要があります。

また、`ignition.platform.id=openstack`は必須であり、この引数がないと、Metal³の自動化フローでIgnitionによるSLEMicroの設定が失敗することにも注意してください。

42.2.2.2 Growfsスクリプト

現在、プロビジョニング後の初回ブート時にディスクサイズに合わせてファイルシステムを拡張するには、カスタムスクリプト(`custom/scripts/01-fix-growfs.sh`)が必要です。`01-fix-growfs.sh`スクリプトには次の情報が含まれます。

```
#!/bin/bash
growfs() {
    mnt="$1"
    dev="$(findmnt --fstab --target ${mnt} --evaluate --real --output SOURCE --noheadings)"
    # /dev/sda3 -> /dev/sda, /dev/nvme0n1p3 -> /dev/nvme0n1
    parent_dev="/dev/$(lsblk --nodeps -rno PKNAME "${dev}")"
    # Last number in the device name: /dev/nvme0n1p42 -> 42
    partnum="$(echo "${dev}" | sed 's/^.*[^0-9]\([0-9]\+\)$/\1/')"
    ret=0
    growpart "$parent_dev" "$partnum" || ret=$?
    [ $ret -eq 0 ] || [ $ret -eq 1 ] || exit 1
    /usr/lib/systemd/systemd-growfs "$mnt"
}
growfs /
```

42.2.2.3 パフォーマンススクリプト

次のオプションのスクリプト(`custom/scripts/02-performance.sh`)は、パフォーマンス調整用にシステムを設定するために使用できます。

```
#!/bin/bash
```

```
# create the folder to extract the artifacts there
mkdir -p /opt/performance-settings

# copy the artifacts
cp performance-settings.sh /opt/performance-settings/
```

custom/files/performance-settings.shのコンテンツは、パフォーマンス調整用にシステムを設定するために使用可能なスクリプトであり、次のリンク (<https://github.com/suse-edge/atip/blob/release-3.4/telco-examples/edge-clusters/dhcp/eib/custom/files/performance-settings.sh>) からダウンロードできます。

42.2.2.4 SR-IOVスクリプト

次のオプションスクリプト([custom/scripts/03-sriov.sh](#))はSR-IOV用にシステムを設定するために使用できます。

```
#!/bin/bash

# create the folder to extract the artifacts there
mkdir -p /opt/sriov
# copy the artifacts
cp sriov-auto-filler.sh /opt/sriov/sriov-auto-filler.sh
```

custom/files/sriov-auto-filler.shのコンテンツは、SR-IOV用にシステムを設定するために使用可能なスクリプトであり、次のリンク (<https://github.com/suse-edge/atip/blob/release-3.4/telco-examples/edge-clusters/dhcp/eib/custom/files/sriov-auto-filler.sh>) からダウンロードできます。



注記

同じアプローチを使用して、プロビジョニングプロセス中に実行する独自のカスタムスクリプトを追加します。詳細については、[第3章「Edge Image Builderを使用したスタンドアロンクラスタ」](#)を参照してください。

42.2.2.5 通信ワークロードの追加設定

dpdk、sr-iov、FECなどの通信機能を有効にするには、次の例に示すように追加のパッケージが必要な場合があります。

```
apiVersion: 1.3
image:
  imageType: raw
```

```

arch: x86_64
baseImage: SL-Micro.x86_64-6.1-Base-GM.raw
outputImageName: eibimage-output-telco.raw
operatingSystem:
  kernelArgs:
    - ignition.platform.id=openstack
    - net.ifnames=1
  systemd:
    disable:
      - rebootmgr
      - transactional-update.timer
      - transactional-update-cleanup.timer
      - fstrim
      - time-sync.target
  users:
    - username: root
      encryptedPassword: $ROOT_PASSWORD
      sshKeys:
        - $user1Key1
  packages:
    packageList:
      - jq
      - dpdk
      - dpdk-tools
      - libdpdk-23
      - pf-bb-config
  sccRegistrationCode: $SCC_REGISTRATION_CODE

```

ここで、`$SCC_REGISTRATION_CODE`は[SUSE Customer Center \(https://scc.suse.com/\)](https://scc.suse.com/) からコピーした登録コードです。また、パッケージリストには通信事業者プロファイル用の最小限のパッケージが含まれています。



注記

`arch: x86_64`は、イメージのアーキテクチャです。arm64アーキテクチャの場合は、`arch: aarch64`を使用します。

42.2.2.6 高度なネットワーク設定のための追加スクリプト

42.6項「高度なネットワーク設定」で説明されている静的IPや、より高度なネットワーキングシナリオを設定する必要がある場合、次の追加設定が必要です。

`network`フォルダに、次の`configure-network.sh`ファイルを作成します。このファイルは、初回ブート時に設定ドライブデータを使用し、[NM Configuratorツール \(https://github.com/suse-edge/nm-configurator\)](https://github.com/suse-edge/nm-configurator) を使用してホストネットワーキングを設定します。

```
#!/bin/bash

set -eux

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

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

mount -o ro $CONFIG_DRIVE /mnt

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

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

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

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

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

42.2.3 イメージの作成

これまでのセクションに従ってディレクトリ構造を準備したら、次のコマンドを実行してイメージを構築します。

```
podman run --rm --privileged -it -v $PWD:/eib \
registry.suse.com/edge/3.4/edge-image-builder:1.3.0 \
build --definition-file downstream-cluster-config.yaml
```

これにより、上記の定義に基づいた、eibimage-output-telco.rawという名前の出力ISOイメージファイルが作成されます。

その後、この出力イメージをWebサーバ経由で利用できるようにする必要があります。その際、管理クラスタのドキュメントを使用して有効にしたメディアサーバコンテナ(注記)か、ローカルにアクセス可能な他のサーバのいずれかを使用します。以下の例では、このサーバを `imagecache.local:8080` として参照します。

42.3 エアギャップシナリオ用のダウンストリームクラスタイメージの準備

Edge Image Builder (第11章「Edge Image Builder」)を使用して、ダウンストリームクラスタホスト上にプロビジョニングされる、変更されたSLEMicroゴールデンイメージを準備します。

設定の多くはEdge Image Builderを使用して行うことができますが、このガイドではエアギャップシナリオ用のダウンストリームクラスタの設定に必要な最小限の設定について説明します。

42.3.1 エアギャップシナリオの前提条件

- Edge Image Builderを実行するには、Podman (<https://podman.io>)  や Rancher Desktop (<https://rancherdesktop.io>)  などのコンテナランタイムが必要です。
- ゴールデンイメージは、プロファイル `Base-SelfInstall` (リアルタイムカーネルには `Base-RT-SelfInstall`) を使用し、次のガイド第28章「Kiwiを使用したSUSE Linux Microの更新イメージの構築」に従って構築されます。このプロセスは両方のアーキテクチャ(x86-64とaarch64)で同じです。
- コンテナイメージが必要なSR-IOVなどのワークロードを使用する場合、ローカルのプライベートレジストリをデプロイして設定済みである必要があります(TLSまたは認証、あるいはその両方を使用/不使用)。このレジストリを使用して、イメージとHelmチャートOCIイメージを保存します。



注記

構築するイメージと同じアーキテクチャの構築ホストを使用する必要があります。つまり、aarch64のイメージを構築するにはaarch64の構築ホストを使用する必要があります、x86-64の場合も同様です(クロス構築は現在のところサポートされていません)。

42.3.2 エアギャップシナリオのイメージの設定

Edge Image Builderを実行すると、そのホストからディレクトリがマウントされるため、ターゲットイメージの定義に使用する設定ファイルを保存するディレクトリ構造を作成する必要があります。

- `downstream-cluster-airgap-config.yaml`はイメージ定義ファイルです。詳細については、[第3章「Edge Image Builderを使用したスタンドアロンクラスタ」](#)を参照してください。
- ゴールデンイメージフォルダには、ガイド[第28章「Kiwiを使用したSUSE Linux Microの更新イメージの構築」](#)に従って生成された出力RAWイメージが含まれます。プロファイル`Base-SelfInstall` (または、リアルタイムカーネルの場合は`Base-RT-SelfInstall`) は、`base-images`フォルダにコピー/移動する必要があります。
- `network`フォルダはオプションです。詳細については、[42.2.2.6項「高度なネットワーク設定のための追加スクリプト」](#)を参照してください。
- `custom/scripts`ディレクトリには初回ブート時に実行するスクリプトが含まれます。
 1. `01-fix-growfs.sh`スクリプトは、デプロイメント時にOSルートパーティションをサイズ変更するために必要です。
 2. `02-airgap.sh`スクリプトは、エアギャップ環境でのイメージ作成プロセス中にイメージを適切な場所にコピーするために必要です。
 3. `03-performance.sh`スクリプトはオプションであり、パフォーマンス調整用にシステムを設定するために使用できます。
 4. `04-sriov.sh`スクリプトはオプションであり、SR-IOV用にシステムを設定するために使用できます。
- `custom/files`ディレクトリには、イメージ作成プロセス中にイメージにコピーされる`rke2`および`cni`イメージが含まれています。また、オプションの`performance-settings.sh`および`sriov-auto-filler.sh`ファイルを含めることもできます。

```
├─ downstream-cluster-airgap-config.yaml
├─ base-images/
│   └─ SL-Micro.x86_64-6.1-Base-GM.raw
├─ network/
│   └─ configure-network.sh
└─ custom/
    └─ files/
        └─ install.sh
```

```

|   L rke2-images-cilium.linux-amd64.tar.zst
|   L rke2-images-core.linux-amd64.tar.zst
|   L rke2-images-multus.linux-amd64.tar.zst
|   L rke2-images.linux-amd64.tar.zst
|   L rke2.linux-amd64.tar.zst
|   L sha256sum-amd64.txt
|   L performance-settings.sh
|   L sriov-auto-filler.sh
L scripts/
  L 01-fix-growfs.sh
  L 02-airgap.sh
  L 03-performance.sh
  L 04-sriov.sh

```

42.3.2.1 ダウンストリームクラスタイメージ定義ファイル

`downstream-cluster-airgap-config.yaml` ファイルは、ダウンストリームクラスタ用のメイン設定ファイルです。その内容については、前のセクション(42.2.2.5項「通信ワークロードの追加設定」)で説明されています。

42.3.2.2 Growfsスクリプト

現在、プロビジョニング後の初回ブート時にディスクサイズに合わせてファイルシステムを拡張するには、カスタムスクリプト(`custom/scripts/01-fix-growfs.sh`)が必要です。`01-fix-growfs.sh` スクリプトには次の情報が含まれます。

```

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

```


42.3.2.3 エアギャップスクリプト

イメージ作成プロセス中にイメージを正しい場所にコピーするために、次のスクリプト([custom/scripts/02-airgap.sh](#))が必要です。

```
#!/bin/bash

# create the folder to extract the artifacts there
mkdir -p /opt/rke2-artifacts
mkdir -p /var/lib/rancher/rke2/agent/images

# copy the artifacts
cp install.sh /opt/
cp rke2-images*.tar.zst rke2.linux-amd64.tar.gz sha256sum-amd64.txt /opt/rke2-artifacts/
```

42.3.2.4 パフォーマンススクリプト

次のオプションのスクリプト([custom/scripts/03-performance.sh](#))は、パフォーマンス調整用にシステムを設定するために使用できます。

```
#!/bin/bash

# create the folder to extract the artifacts there
mkdir -p /opt/performance-settings

# copy the artifacts
cp performance-settings.sh /opt/performance-settings/
```

[custom/files/performance-settings.sh](#)のコンテンツは、パフォーマンス調整用にシステムを設定するために使用可能なスクリプトであり、次の[リンク \(https://github.com/suse-edge/atip/blob/release-3.4/telco-examples/edge-clusters/dhcp/eib/custom/files/performance-settings.sh\)](https://github.com/suse-edge/atip/blob/release-3.4/telco-examples/edge-clusters/dhcp/eib/custom/files/performance-settings.sh) からダウンロードできます。

42.3.2.5 SR-IOVスクリプト

次のオプションのスクリプト([custom/scripts/04-sriov.sh](#))は、SR-IOV用にシステムを設定するために使用できます。

```
#!/bin/bash

# create the folder to extract the artifacts there
mkdir -p /opt/sriov
# copy the artifacts
cp sriov-auto-filler.sh /opt/sriov/sriov-auto-filler.sh
```

`custom/files/sriov-auto-filler.sh`のコンテンツは、SR-IOV用にシステムを設定するために使用可能なスクリプトであり、次のリンク (<https://github.com/suse-edge/atip/blob/release-3.4/telco-examples/edge-clusters/dhcp/eib/custom/files/sriov-auto-filler.sh>) からダウンロードできます。

42.3.2.6 エアギャップシナリオのカスタムファイル

`custom/files`ディレクトリには、イメージ作成プロセス中にそのイメージにコピーする `rke2` イメージと `cni` イメージが含まれています。イメージを簡単に生成するには、次のスクリプト (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/scripts/day2/edge-save-images.sh>) と、こちら (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/scripts/day2/edge-release-rke2-images.txt>) にあるイメージのリストを使用してローカルでイメージを準備し、`custom/files`に含める必要があるアーティファクトを生成します。また、最新の `rke2-install` スクリプトをこちら (<https://get.rke2.io/>) からダウンロードすることもできます。

```
$ ./edge-save-rke2-images.sh -o custom/files -l ~/edge-release-rke2-images.txt
```

イメージをダウンロードした後、ディレクトリ構造は次のようになるはずです。

```
└─ custom/
   └─ files/
      ├── install.sh
      ├── rke2-images-cilium.linux-amd64.tar.zst
      ├── rke2-images-core.linux-amd64.tar.zst
      ├── rke2-images-multus.linux-amd64.tar.zst
      ├── rke2-images.linux-amd64.tar.zst
      ├── rke2.linux-amd64.tar.zst
      └── sha256sum-amd64.txt
```

42.3.2.7 エアギャップシナリオおよびSR-IOV (オプション)に必要なイメージのプライベートレジストリへのプリロード

エアギャップシナリオまたはその他のワークロードイメージでSR-IOVを使用する場合、次の各手順に従って、ローカルのプライベートレジストリにイメージをプリロードする必要があります。

- HelmチャートOCIイメージをダウンロードして抽出し、プライベートレジストリにプッシュする
- 必要な残りのイメージをダウンロードして抽出し、プライベートレジストリにプッシュする

次のスクリプトを使用して、イメージをダウンロードして抽出し、プライベートレジストリにプッシュできます。これからSR-IOVイメージをプリロードする例を示しますが、その他のカスタムイメージも同じ方法でプリロードすることができます。

1. SR-IOVのHelmチャートOCIイメージのプリロード:

- a. 必要なHelmチャートOCIイメージが含まれるリストを作成する必要があります。

```
$ cat > edge-release-helm-oci-artifacts.txt <<EOF
edge/sriov-network-operator-chart:304.0.2+up1.5.0
edge/sriov-crd-chart:304.0.2+up1.5.0
EOF
```

- b. 次のスクリプト (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/scripts/day2/edge-save-oci-artefacts.sh>) と上記で作成したリストを使用してローカルtarballファイルを生成します。

```
$ ./edge-save-oci-artefacts.sh -al ./edge-release-helm-oci-artifacts.txt -s
registry.suse.com
Pulled: registry.suse.com/edge/charts/sriov-network-operator:304.0.2+up1.5.0
Pulled: registry.suse.com/edge/charts/sriov-crd:304.0.2+up1.5.0
a edge-release-oci-tgz-20240705
a edge-release-oci-tgz-20240705/sriov-network-operator-
chart-304.0.2+up1.5.0.tgz
a edge-release-oci-tgz-20240705/sriov-crd-chart-304.0.2+up1.5.0.tgz
```

- c. 次のスクリプト (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/scripts/day2/edge-load-oci-artefacts.sh>) を使用してtarballファイルをプライベートレジストリ(例: myregistry:5000)にアップロードし、前の手順でダウンロードしたHelmチャートOCIイメージをレジストリにプリロードします。

```
$ tar zxvf edge-release-oci-tgz-20240705.tgz
$ ./edge-load-oci-artefacts.sh -ad edge-release-oci-tgz-20240705 -r
myregistry:5000
```

2. SR-IOVに必要な残りのイメージをプリロードします。

- a. ここでは、通信ワークロードのために「sr-iov」コンテナイメージを含める必要があります(例: 参考として、これは [helmチャート値 \(https://github.com/suse-edge/charts/blob/main/charts/sriov-network-operator/1.5.0/values.yaml\)](https://github.com/suse-edge/charts/blob/main/charts/sriov-network-operator/1.5.0/values.yaml) から取得できます)。

```
$ cat > edge-release-images.txt <<EOF
rancher/hardened-sriov-network-operator:v1.3.0-build20240816
rancher/hardened-sriov-network-config-daemon:v1.3.0-build20240816
rancher/hardened-sriov-cni:v2.8.1-build20240820
```

```
rancher/hardened-ib-sriov-cni:v1.1.1-build20240816
rancher/hardened-sriov-network-device-plugin:v3.7.0-build20240816
rancher/hardened-sriov-network-resources-injector:v1.6.0-build20240816
rancher/hardened-sriov-network-webhook:v1.3.0-build20240816
EOF
```

- b. 次のスクリプト (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/scripts/day2/edge-save-images.sh>) と上記で作成したリストを使用して、必要なイメージを含む、tarballファイルをローカルで生成する必要があります。

```
$ ./edge-save-images.sh -l ./edge-release-images.txt -s registry.suse.com
Image pull success: registry.suse.com/rancher/hardened-sriov-network-operator:v1.3.0-build20240816
Image pull success: registry.suse.com/rancher/hardened-sriov-network-config-daemon:v1.3.0-build20240816
Image pull success: registry.suse.com/rancher/hardened-sriov-cni:v2.8.1-build20240820
Image pull success: registry.suse.com/rancher/hardened-ib-sriov-cni:v1.1.1-build20240816
Image pull success: registry.suse.com/rancher/hardened-sriov-network-device-plugin:v3.7.0-build20240816
Image pull success: registry.suse.com/rancher/hardened-sriov-network-resources-injector:v1.6.0-build20240816
Image pull success: registry.suse.com/rancher/hardened-sriov-network-webhook:v1.3.0-build20240816
Creating edge-images.tar.gz with 7 images
```

- c. 次のスクリプト (<https://github.com/suse-edge/fleet-examples/blob/release-3.4.0/scripts/day2/edge-load-images.sh>) を使用してtarballファイルをプライベートレジストリ(例: myregistry:5000)にアップロードし、前の手順でダウンロードしたイメージをプライベートレジストリにプリロードします。

```
$ tar zxvf edge-release-images-tgz-20240705.tgz
$ ./edge-load-images.sh -ad edge-release-images-tgz-20240705 -r myregistry:5000
```

42.3.3 エアギャップシナリオのイメージの作成

これまでのセクションに従ってディレクトリ構造を準備したら、次のコマンドを実行してイメージを構築します。

```
podman run --rm --privileged -it -v $PWD:/eib \
registry.suse.com/edge/3.4/edge-image-builder:1.3.0 \
build --definition-file downstream-cluster-airgap-config.yaml
```

これにより、上記の定義に基づいた、`eibimage-output-telco.raw`という名前の出力ISOイメージファイルが作成されます。

その後、この出力イメージをWebサーバ経由で利用できるようにする必要があります。その際、管理クラスタドキュメントを使用して有効にしたメディアサーバコンテナ(注記)か、ローカルにアクセス可能な他のサーバのいずれかを使用します。以下の例では、このサーバを`imagecache.local:8080`として参照します。

42.4 ダイレクトネットワークプロビジョニングを使用したダウンストリームクラスタのプロビジョニング(シングルノード)

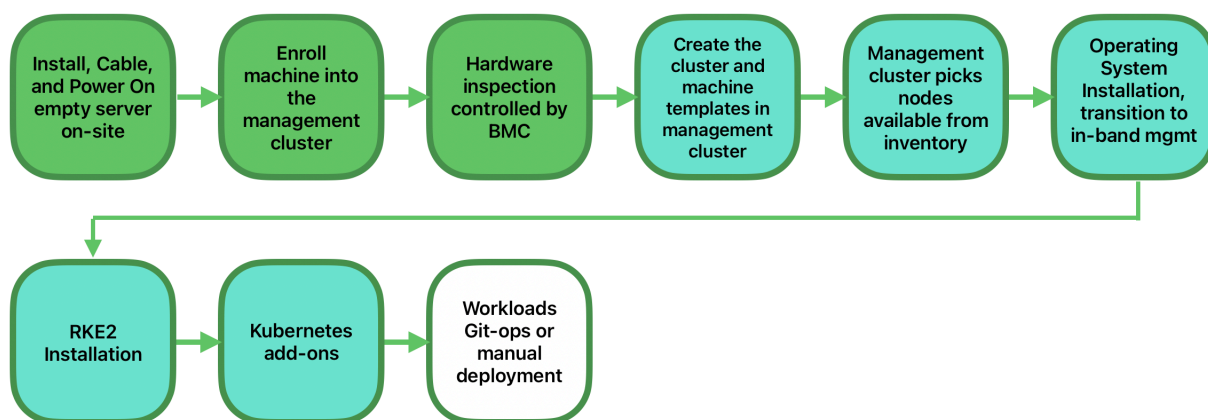
このセクションでは、ダイレクトネットワークプロビジョニングを使用してシングルノードのダウンストリームクラスタのプロビジョニングを自動化するために用いるワークフローについて説明します。これは、ダウンストリームクラスタのプロビジョニングを自動化する最もシンプルな方法です。

要件

- 前のセクション(42.2項「[接続シナリオのダウンストリームクラスタイメージの準備](#)」)で説明されているように、EIBを使用して、ダウンストリームクラスタを設定するための最小限の設定で生成されたイメージが、こちらのセクション(注記)で設定した正確なパス上にある管理クラスタに配置されている。
- 管理サーバが作成されていて、次の各セクションで使えるようになっている。詳細については、管理クラスタに関するセクション第40章「[管理クラスタの設定](#)」を参照してください。

ワークフロー

次の図は、ダイレクトネットワークプロビジョニングを使用してシングルノードのダウンストリームクラスタのプロビジョニングを自動化するために用いるワークフローを示しています。



ダイレクトネットワークプロビジョニングを使用してシングルノードのダウンストリームクラスタのプロビジョニングを自動化する手順は2種類です。

1. ベアメタルホストを登録して、プロビジョニングプロセスでできるようにする。
2. ベアメタルホストをプロビジョニングして、オペレーティングシステムとKubernetesクラスタをインストールして設定する。

ベアメタルホストの登録

最初の手順では、新しいベアメタルホストを管理クラスタに登録してプロビジョニングできるようにします。そのためには、ファイル**bmh-example.yaml**を管理クラスタ内に作成して、使用する**BMC**資格情報と登録する**BaremetalHost**オブジェクトを指定する必要があります。

```

apiVersion: v1
kind: Secret
metadata:
  name: example-demo-credentials
type: Opaque
data:
  username: ${BMC_USERNAME}
  password: ${BMC_PASSWORD}
---
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: example-demo
  labels:
    cluster-role: control-plane
spec:
  architecture: x86_64
  online: true
  bootMACAddress: ${BMC_MAC}
  rootDeviceHints:

```

```
deviceName: /dev/nvme0n1
bmc:
  address: ${BMC_ADDRESS}
  disableCertificateVerification: true
  credentialsName: example-demo-credentials
```

各項目の内容は次のとおりです。

- `${BMC_USERNAME}` — 新しいベアメタルホストのBMCのユーザ名。
- `${BMC_PASSWORD}` — 新しいベアメタルホストのBMCのパスワード。
- `${BMC_MAC}` — 使用する新しいベアメタルホストのMACアドレス。
- `${BMC_ADDRESS}` — ベアメタルホストのBMCのURL (例: `redfish-virtualmedia://192.168.200.75/redfish/v1/Systems/1/`)。ハードウェアプロバイダに応じて使用できる各種のオプションの詳細については、[こちらのリンク \(https://github.com/metal3-io/baremetal-operator/blob/main/docs/api.md\)](https://github.com/metal3-io/baremetal-operator/blob/main/docs/api.md) を確認してください。



注記

- アーキテクチャは、登録するベアメタルホストのアーキテクチャに応じて、`x86_64`または`aarch64`のいずれかである必要があります。
- ホストのネットワーク設定が、イメージのビルド時または`BareMetalHost`定義を通じて指定されていない場合、自動設定メカニズム(DHCP、DHCPv6、SLAAC)が使用されます。詳細または複雑な設定については、[42.6項「高度なネットワーク設定」](#)を参照してください。

ファイルを作成したら、管理クラスタで次のコマンドを実行し、管理クラスタで新しいベアメタルホストの登録を開始する必要があります。

```
$ kubectl apply -f bmh-example.yaml
```

新しいベアメタルホストオブジェクトが登録され、その状態が「Registering (登録中)」から「Inspecting (検査中)」に変わり、「Available (使用可能)」になります。この変更は次のコマンドを使用して確認できます。

```
$ kubectl get bmh
```



注記

BaremetalHostオブジェクトは、BMCの資格情報が検証されるまでは「Registering (登録中)」の状態です。資格情報の検証が完了すると、BaremetalHostオブジェクトの状態が「Inspecting (検査中)」に変わります。この手順はハードウェアによっては多少時間がかかる可能性があります(最大で20分)。「Inspecting (検査中)」のフェーズ中に、ハードウェア情報が取得されてKubernetesオブジェクトが更新されます。kubectl get bmh -o yamlコマンドを使用して情報を確認してください。

プロビジョニング手順

ベアメタルホストが登録されて使用可能になったら、次の手順で、ベアメタルホストをプロビジョニングしてオペレーティングシステムとKubernetesクラスタをインストールして設定します。そのためには、次の情報を使用して管理クラスタでファイルcapi-provisioning-example.yamlを作成する必要があります(capi-provisioning-example.yamlは、以下のブロックを結合して生成できます)。



注記

実際の値に置き換える必要があるのは、 $\${...}$ の中の値だけです。

次のブロックはクラスタ定義です。ここで、Podsブロックとservicesブロックを使用してネットワーキングを設定できます。また、ここにはコントロールプレーンオブジェクトと、使用するインフラストラクチャ(Metal3プロバイダを使用)オブジェクトへの参照も含まれています。

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: single-node-cluster
  namespace: default
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - 192.168.0.0/18
        - fd00:bad:cafe::/48
    services:
      cidrBlocks:
        - 10.96.0.0/12
        - fd00:bad:bad:cafe::/112
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
```



```
kind: RKE2ControlPlane
name: single-node-cluster
infrastructureRef:
  apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
  kind: Metal3Cluster
  name: single-node-cluster
```

注記

- シングルスタックとデュアルスタック両方のデプロイメントが可能です。IPv4のみのクラスタの場合は、上記の定義からIPv6 CIDRを削除してください。
- シングルスタックのIPv6デプロイメントは技術プレビュー状態であり、公式にはサポートされていません。

Metal3Clusterオブジェクトには、設定するコントロールプレーンのエンドポイント(`${DOWNSTREAM_CONTROL_PLANE_IPV4}`を置き換える)と、noCloudProvider(ベアメタルノードを使用しているため)を指定します。

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3Cluster
metadata:
  name: single-node-cluster
  namespace: default
spec:
  controlPlaneEndpoint:
    host: ${DOWNSTREAM_CONTROL_PLANE_IPV4}
    port: 6443
  noCloudProvider: true
```

RKE2ControlPlaneオブジェクトには、使用するコントロールプレーン設定を指定し、Metal3MachineTemplateオブジェクトには、使用するコントロールプレーンのイメージを指定します。また、ここには使用するレプリカの数(ここでは1)と、使用するCNIプラグイン(ここではCilium)に関する情報が含まれています。agentConfigブロックには、使用するIgnition形式と、使用するadditionalUserDataが含まれます。これを使用してRKE2ノードに`rke2-preinstall.service`という名前のsystemdサービスを設定し、プロビジョニングプロセス中にIronicの情報を使用してBAREMETALHOST_UUIDとnode-nameを自動的に置き換えます。ciliumでmultusを有効にするには、使用する設定を含むファイルをrke2サーバマニフェストディレクトリに`rke2-cilium-config.yaml`という名前で作成します。最後の情報ブロックには、使用するKubernetesバージョンが含まれています。`${RKE2_VERSION}`は使用するRKE2のバージョンであり、この値は置き換えます(例: `v1.33.3+rke2r1`)。

```

apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
metadata:
  name: single-node-cluster
  namespace: default
spec:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3MachineTemplate
    name: single-node-cluster-controlplane
  replicas: 1
  version: ${RKE2_VERSION}
  rolloutStrategy:
    type: "RollingUpdate"
    rollingUpdate:
      maxSurge: 0
  serverConfig:
    cni: cilium
  agentConfig:
    format: ignition
    additionalUserData:
      config: |
        variant: fcos
        version: 1.4.0
        systemd:
          units:
            - name: rke2-preinstall.service
              enabled: true
              contents: |
                [Unit]
                Description=rke2-preinstall
                Wants=network-online.target
                Before=rke2-install.service
                ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
                [Service]
                Type=oneshot
                User=root
                ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
                ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/${jq -r .uuid /mnt/
openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
                ExecStart=/bin/sh -c "echo \"node-name: ${jq -r .name /mnt/openstack/
latest/meta_data.json}\" >> /etc/rancher/rke2/config.yaml"
                ExecStartPost=/bin/sh -c "umount /mnt"
                [Install]
                WantedBy=multi-user.target
      storage:
        files:
          # https://docs.rke2.io/networking/multus_sriov#using-multus-with-cilium
          - path: /var/lib/rancher/rke2/server/manifests/rke2-cilium-config.yaml
            overwrite: true

```

```

    contents:
      inline: |
        apiVersion: helm.cattle.io/v1
        kind: HelmChartConfig
        metadata:
          name: rke2-cilium
          namespace: kube-system
        spec:
          valuesContent: |-
            cni:
              exclusive: false
        mode: 0644
        user:
          name: root
        group:
          name: root
  kubelet:
    extraArgs:
      - provider-id=metal3://BAREMETALHOST_UUID
    nodeName: "localhost.localdomain"

```

Metal3MachineTemplateオブジェクトには次の情報を指定します。

- テンプレートへの参照として使用するdataTemplate。
- 登録プロセス中に作成されたラベルとの一致に使用するhostSelector。
- 前のセクション(42.2項「接続シナリオのダウンストリームクラスタイメージの準備」)でEIBを使用して生成されたイメージへの参照として使用するimage、およびそのイメージを検証するために使用するchecksumとchecksumType。

```

apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3MachineTemplate
metadata:
  name: single-node-cluster-controlplane
  namespace: default
spec:
  template:
    spec:
      dataTemplate:
        name: single-node-cluster-controlplane-template
      hostSelector:
        matchLabels:
          cluster-role: control-plane
      image:
        checksum: http://imagecache.local:8080/eibimage-output-telco.raw.sha256
        checksumType: sha256
        format: raw
        url: http://imagecache.local:8080/eibimage-output-telco.raw

```

Metal3DataTemplateオブジェクトには、ダウンストリームクラスタのmetaDataを指定します。

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

前のブロックを結合してファイルを作成したら、管理クラスタで次のコマンドを実行して、新しいベアメタルホストのプロビジョニングを開始する必要があります。

```
$ kubectl apply -f capi-provisioning-example.yaml
```

42.5 ダイレクトネットワークプロビジョニングを使用したダウンストリームクラスタのプロビジョニング(マルチノード)

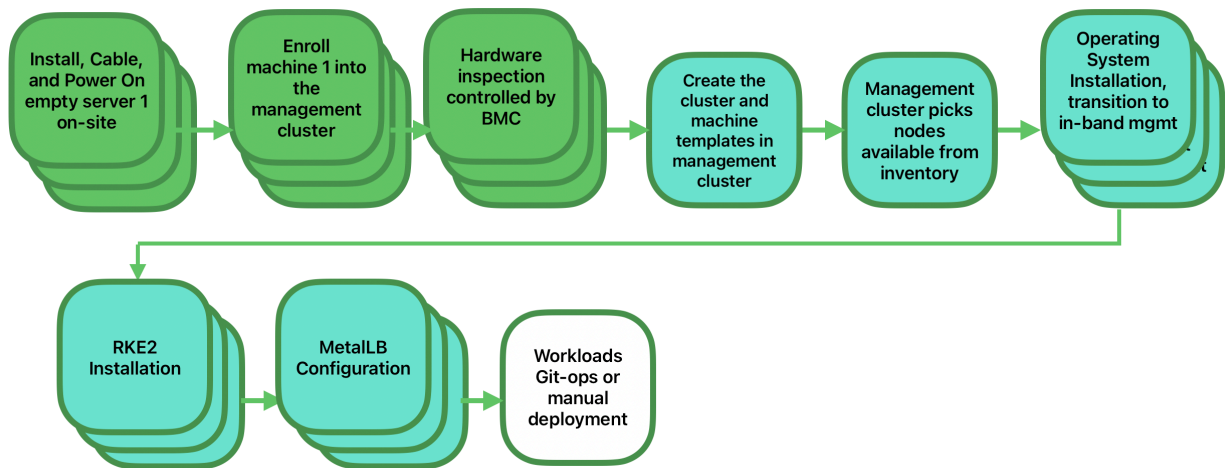
このセクションでは、ダイレクトネットワークプロビジョニングとMetalLBをロードバランサ戦略として使用して、マルチノードのダウンストリームクラスタのプロビジョニングを自動化するために使用するワークフローについて説明します。これはダウンストリームクラスタのプロビジョニングを自動化する最もシンプルな方法です。次の図は、ダイレクトネットワークプロビジョニングとMetalLBを使用してマルチノードのダウンストリームクラスタのプロビジョニングを自動化するためのワークフローを示しています。

要件

- 前のセクション(42.2項「[接続シナリオのダウストリームクラスティメージの準備](#)」)で説明されているように、[EIB](#)を使用して、ダウストリームクラスタを設定するための最小限の設定で生成されたイメージが、こちらのセクション(注記)で設定した正確なパス上にある管理クラスタに配置されている。
- 管理サーバが作成されていて、次の各セクションで使えるようになっている。詳細については、管理クラスタに関するセクション第40章「[管理クラスタの設定](#)」を参照してください。

ワークフロー

次の図は、ダイレクトネットワークプロビジョニングを使用してマルチノードのダウストリームクラスタのプロビジョニングを自動化するために使用するワークフローを示しています。



1. 3つのベアメタルホストを登録し、プロビジョニングプロセスで使えるようにする。
2. 3つのベアメタルホストをプロビジョニングし、オペレーティングシステムと、[MetalLB](#)を使用するKubernetesクラスタをインストールして設定する。

ベアメタルホストの登録

最初の手順では、管理クラスタに3つのベアメタルホストを登録してプロビジョニングできるようにします。そのためには、管理クラスタにファイル**[bmh-example-node1.yaml](#)**、**[bmh-example-node2.yaml](#)**、および**[bmh-example-node3.yaml](#)**を作成して、使用するBMC資格情報と、管理クラスタに登録する[BaremetalHost](#)オブジェクトを指定する必要があります。

注記

- 実際の値に置き換える必要があるのは、`${...}`の中の値だけです。
- 1つのホストのプロセスについてのみ説明します。他の2つのノードにも同じ手順が適用されます。

```
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:
  architecture: x86_64
  online: true
  bootMACAddress: ${BMC_NODE1_MAC}
  bmc:
    address: ${BMC_NODE1_ADDRESS}
    disableCertificateVerification: true
    credentialsName: node1-example-credentials
```

各項目の内容は次のとおりです。

- `${BMC_NODE1_USERNAME}` — 最初のベアメタルホストのBMCのユーザ名。
- `${BMC_NODE1_PASSWORD}` — 最初のベアメタルホストのBMCのパスワード。
- `${BMC_NODE1_MAC}` — 使用する最初のベアメタルホストのMACアドレス。
- `${BMC_NODE1_ADDRESS}` — 最初のベアメタルホストのBMCのURL (例: `redfish-virtualmedia://192.168.200.75/redfish/v1/Systems/1/`)。URLのホスト部分は、既存のインフラストラクチャで許可されている場合、IPアドレス(v4またはv6)またはドメイン名とすることができます。ハードウェアプロバイダに応じて使用できる各種のオプションの詳細については、[こちらのリンク \(https://github.com/metal3-io/baremetal-operator/blob/main/docs/api.md\)](https://github.com/metal3-io/baremetal-operator/blob/main/docs/api.md) を確認してください。

注記

- ホストのネットワーク設定が、イメージのビルド時またはBareMetalHost定義を通じて指定されていない場合、自動設定メカニズム(DHCP、DHCPv6、SLAAC)が使用されます。詳細または複雑な設定については、[42.6項「高度なネットワーク設定」](#)を参照してください。
- シングルスタックIPv6クラスタは技術プレビュー状態であり、まだ公式にはサポートされていません。
- アーキテクチャは、登録するベアメタルホストのアーキテクチャに応じて、[x86_64](#)または[aarch64](#)のいずれかである必要があります。
- すべての最新サーバにはデュアルスタック対応のBMCが搭載されていますが、デュアルスタック環境で運用する前に、IPv6のサポート(および仮想メディア機能にホスト名を使用するオプション)を確認する必要があります。

ファイルを作成したら、管理クラスタで次のコマンドを実行して、管理クラスタへのベアメタルホストの登録を開始する必要があります。

```
$ kubectl apply -f bmh-example-node1.yaml
$ kubectl apply -f bmh-example-node2.yaml
$ kubectl apply -f bmh-example-node3.yaml
```

新しいベアメタルホストオブジェクトが登録され、その状態が「Registering (登録中)」から「Inspecting (検査中)」に変わり、「Available (使用可能)」になります。この変更は次のコマンドを使用して確認できます。

```
$ kubectl get bmh -o wide
```

注記

BaremetalHostオブジェクトは、BMCの資格情報が検証されるまでは「Registering (登録中)」の状態です。資格情報の検証が完了すると、BaremetalHostオブジェクトの状態が「Inspecting (検査中)」に変わります。この手順はハードウェアによっては多少時間がかかる可能性があります(最大で20分)。「Inspecting (検査中)」のフェーズ中に、ハードウェア情報が取得されてKubernetesオブジェクトが更新されます。[kubectl get bmh -o yaml](#)コマンドを使用して情報を確認してください。

プロビジョニング手順

3つのベアメタルホストが登録されて使用可能になったら、次の手順は、ベアメタルホストをプロビジョニングしてオペレーティングシステムとKubernetesクラスタをインストールして設定し、ロードバランサを作成して3つのベアメタルホストを管理することです。そのためには、次の情報を使用して管理クラスタにファイル`capi-provisioning-example.yaml`を作成する必要があります(`capi-provisioning-example.yaml`は、次のブロックを結合して生成できます)。

注記

- 実際の値に置き換える必要があるのは、`$_{...}`の中の値だけです。
- VIPアドレスは、どのノードにも割り当てられていない予約済みのIPアドレスであり、ロードバランサを設定するために使用されます。デュアルスタッククラスタでは、IPv4とIPv6の両方を指定できますが、以下の例ではIPv4アドレスが優先されます。

以下はクラスタ定義です。ここで、 `pods` ブロックと `services` ブロックを使用してクラスタのネットワークを設定できます。また、ここにはコントロールプレーンと、使用するインフラストラクチャ(Metal3プロバイダを使用)のオブジェクトへの参照も含まれています。

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: multinode-cluster
  namespace: default
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - 192.168.0.0/18
        - fd00:1234:4321::/48
    services:
      cidrBlocks:
        - 10.96.0.0/12
        - fd00:5678:8765:4321::/112
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: RKE2ControlPlane
    name: multinode-cluster
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3Cluster
    name: multinode-cluster
```




注記

シングルスタックとデュアルスタックの両方のデプロイメントが可能です。IPv4のみのクラスタの場合は、IPv6 CIDRおよびIPv6 VIPアドレス(後述するセクション)を削除してください

`Metal3Cluster`オブジェクトには、予約済みのVIPアドレス(`${EDGE_VIP_ADDRESS_IPV4}`を置き換え)を使用して設定するコントロールプレーンのエンドポイントと、`noCloudProvider`(3つのベアメタルノードを使用しているため)を指定しています。

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3Cluster
metadata:
  name: multinode-cluster
  namespace: default
spec:
  controlPlaneEndpoint:
    host: ${EDGE_VIP_ADDRESS_IPV4}
    port: 6443
  noCloudProvider: true
```

`RKE2ControlPlane`オブジェクトには、使用するコントロールプレーンの設定を指定し、`Metal3MachineTemplate`オブジェクトには、使用するコントロールプレーンのイメージを指定します。

- 使用するレプリカの数(ここでは3)。
- ロードバランサで使用するアドバタイズモード(`address`ではL2実装を使用)、および使用するアドレス(`${EDGE_VIP_ADDRESS}`をVIPアドレスに置き換え)。
- 使用するCNIプラグイン(ここではCilium)と、`tlsSan`の下にリストされる追加のVIPアドレスと名前が含まれる`serverConfig`。
- `agentConfig`ブロックには、使用するIgnitionの形式と、RKE2ノードに次のような情報を設定するために使用する`additionalUserData`が含まれています。
 - プロビジョニングプロセス中にIronicの情報を使用してBAREMETALHOST_UUIDとnode-nameを自動的に置き換える、`rke2-preinstall.service`という名前のsystemdサービス。
 - `MetalLB`と`endpoint-copier-operator`のインストールに使用するHelmチャートが含まれている`storage`ブロック。

- 使用する `IPAddressPool` と `L2Advertisement` が含まれている `metalLB` カスタムリソースファイル(`${EDGE_VIP_ADDRESS_IPV4}`をVIPアドレスに置き換え)。
- `MetalLB`がVIPアドレスを管理するために使用する `kubernetes-vip` サービスの設定に使用する `endpoint-svc.yaml` ファイル。
- 最後の情報ブロックには、使用するKubernetesバージョンが含まれています。`${RKE2_VERSION}`は使用するRKE2のバージョンで、この値は置き換えます(例: `v1.33.3+rke2r1`)。

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
metadata:
  name: multinode-cluster
  namespace: default
spec:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3MachineTemplate
    name: multinode-cluster-controlplane
  replicas: 3
  version: ${RKE2_VERSION}
  rolloutStrategy:
    type: "RollingUpdate"
    rollingUpdate:
      maxSurge: 0
  registrationMethod: "control-plane-endpoint"
  registrationAddress: ${EDGE_VIP_ADDRESS}
  serverConfig:
    cni: cilium
    tlsSan:
      - ${EDGE_VIP_ADDRESS_IPV4}
      - ${EDGE_VIP_ADDRESS_IPV6}
      - https://${EDGE_VIP_ADDRESS_IPV4}.sslip.io
      - https://${EDGE_VIP_ADDRESS_IPV6}.sslip.io
  agentConfig:
    format: ignition
    additionalUserData:
      config: |
        variant: fcos
        version: 1.4.0
        systemd:
          units:
            - name: rke2-preinstall.service
              enabled: true
              contents: |
                [Unit]
                Description=rke2-preinstall
```

```

    Wants=network-online.target
    Before=rke2-install.service
    ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
    [Service]
    Type=oneshot
    User=root
    ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
    ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/${jq -r .uuid /mnt/
openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
    ExecStart=/bin/sh -c "echo \"node-name: ${jq -r .name /mnt/openstack/
latest/meta_data.json}\" >> /etc/rancher/rke2/config.yaml"
    ExecStartPost=/bin/sh -c "umount /mnt"
    [Install]
    WantedBy=multi-user.target
storage:
  files:
    # https://docs.rke2.io/networking/multus_sriov#using-multus-with-cilium
    - path: /var/lib/rancher/rke2/server/manifests/rke2-cilium-config.yaml
      overwrite: true
      contents:
        inline: |
          apiVersion: helm.cattle.io/v1
          kind: HelmChartConfig
          metadata:
            name: rke2-cilium
            namespace: kube-system
          spec:
            valuesContent: |-
              cni:
                exclusive: false
      mode: 0644
      user:
        name: root
      group:
        name: root
    - path: /var/lib/rancher/rke2/server/manifests/endpoint-copier-operator.yaml
      overwrite: true
      contents:
        inline: |
          apiVersion: helm.cattle.io/v1
          kind: HelmChart
          metadata:
            name: endpoint-copier-operator
            namespace: kube-system
          spec:
            chart: oci://registry.suse.com/edge/charts/endpoint-copier-operator
            targetNamespace: endpoint-copier-operator
            version: 304.0.1+up0.3.0
            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/charts/metallb
          targetNamespace: metallb-system
          version: 304.0.0+up0.14.9
          createNamespace: true

- path: /var/lib/rancher/rke2/server/manifests/metallb-cr.yaml
  overwrite: true
  contents:
    inline: |
      apiVersion: metallb.io/v1beta1
      kind: IPAddressPool
      metadata:
        name: kubernetes-vip-ip-pool
        namespace: metallb-system
      spec:
        addresses:
          - ${EDGE_VIP_ADDRESS_IPV4}/32
          - ${EDGE_VIP_ADDRESS_IPV6}/128
        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:
      ipFamilyPolicy: PreferDualStack
      ports:
        - name: rke2-api
          port: 9345
          protocol: TCP
          targetPort: 9345
        - name: k8s-api
          port: 6443
          protocol: TCP
          targetPort: 6443
      type: LoadBalancer
  kubelet:
    extraArgs:
      - provider-id=metal3://BAREMETALHOST_UUID
  nodeName: "Node-multinode-cluster"

```

Metal3MachineTemplateオブジェクトには次の情報を指定します。

- テンプレートへの参照として使用する dataTemplate。
- 登録プロセス中に作成されたラベルとの一致に使用する hostSelector。
- 前のセクション(42.2項「接続シナリオのダウンストリームクラスティメージの準備」)でEIBを使用して生成されたイメージへの参照として使用する image、およびそのイメージを検証するために使用する checksumと checksumType。

```

apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3MachineTemplate
metadata:
  name: multinode-cluster-controlplane
  namespace: default
spec:
  template:
    spec:
      dataTemplate:
        name: multinode-cluster-controlplane-template
      hostSelector:
        matchLabels:
          cluster-role: control-plane
      image:
        checksum: http://imagecache.local:8080/eibimage-output-telco.raw.sha256
        checksumType: sha256
        format: raw
        url: http://imagecache.local:8080/eibimage-output-telco.raw

```

Metal3DataTemplateオブジェクトには、ダウンストリームクラスタの metaDataを指定します。

```

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

```

以下のyamlファイルは、ワーカーノードの設定例です。

MachineDeployment:

```

apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineDeployment
metadata:
  labels:
    cluster.x-k8s.io/cluster-name: multinode-cluster
    nodepool: nodepool-0
  name: multinode-cluster-workers
  namespace: default
spec:
  clusterName: multinode-cluster
  replicas: 3
  selector:
    matchLabels:
      cluster.x-k8s.io/cluster-name: multinode-cluster
      nodepool: nodepool-0
  template:
    metadata:
      labels:
        cluster.x-k8s.io/cluster-name: multinode-cluster
        nodepool: nodepool-0
    spec:
      bootstrap:
        configRef:
          apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
          kind: RKE2ConfigTemplate
          name: multinode-cluster-workers
      clusterName: multinode-cluster
      infrastructureRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
        kind: Metal3MachineTemplate
        name: multinode-cluster-workers

```

```
nodeDrainTimeout: 0s
version: ${RKE2_VERSION}
```

RKE2ConfigTemplateオブジェクトには、マルチノードクラスタのワーカーノードに使用する設定テンプレートを指定します。

```
apiVersion: bootstrap.cluster.x-k8s.io/v1beta1
kind: RKE2ConfigTemplate
metadata:
  name: multinode-cluster-workers
  namespace: default
spec:
  template:
    spec:
      agentConfig:
        format: ignition
      kubelet:
        extraArgs:
          - provider-id=metal3://BAREMETALHOST_UUID
      nodeName: "Node-multinode-cluster-worker"
      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
```

Metal3MachineTemplateオブジェクトには、ワーカーノードの [dataTemplate](#)、[hostSelector](#)、[image](#) への参照が含まれています。

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
```

```
kind: Metal3MachineTemplate
metadata:
  name: multinode-cluster-workers
  namespace: default
spec:
  template:
    spec:
      dataTemplate:
        name: multinode-cluster-workers-template
      hostSelector:
        matchLabels:
          cluster-role: worker
      image:
        checksum: http://imagecache.local:8080/eibimage-slmicro-rt-telco.raw.sha256
        checksumType: sha256
        format: raw
        url: http://imagecache.local:8080/eibimage-slmicro-rt-telco.raw
```

Metal3DataTemplateオブジェクトには、ワーカーノードのダウンストリームクラスターの`metaData`を指定します。

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

前のブロックを結合してファイルが作成されたら、管理クラスターで次のコマンドを実行し、新しい3つのベアメタルホストのプロビジョニングを開始します。

```
$ kubectl apply -f capi-provisioning-example.yaml
```

42.6 高度なネットワーク設定

ダイレクトネットワークプロビジョニングワークフローを使用すると、静的IP、ボンディング、VLAN、IPv6など、ダウンストリームクラスターの特定のネットワーク設定が可能になります。

次の各セクションでは、高度なネットワーク設定を使用してダウストリームクラスタをプロビジョニングできるようにするために必要な追加手順について説明します。

要件

- このセクション(42.2.2.6項「高度なネットワーク設定のための追加スクリプト」)に従って、EIBを使用して生成したイメージにネットワークフォルダとスクリプトを含める必要があります。

設定

続行する前に、ホストの登録およびプロビジョニングに必要な手順に関するガイダンスについて、次のいずれかのセクションを参照してください。

- ダイレクトネットワークプロビジョニングを使用したダウストリームクラスタのプロビジョニング(シングルノード) (42.4項「ダイレクトネットワークプロビジョニングを使用したダウストリームクラスタのプロビジョニング(シングルノード)」)
- ダイレクトネットワークプロビジョニングを使用したダウストリームクラスタのプロビジョニング(マルチノード) (42.5項「ダイレクトネットワークプロビジョニングを使用したダウストリームクラスタのプロビジョニング(マルチノード)」)

高度なネットワーク設定は、登録時にBareMetalHostホスト定義およびnmstate形式のnetworkDataブロックを含む関連するシークレットを通じて適用する必要があります。次のサンプルファイルは、ダウストリームクラスタホストに静的IPとVLANを要求する、必須のnetworkDataを含むシークレットを定義しています。

```
apiVersion: v1
kind: Secret
metadata:
  name: controlplane-0-networkdata
type: Opaque
stringData:
  networkData: |
    interfaces:
    - name: ${CONTROLPLANE_INTERFACE}
      type: ethernet
      state: up
      mtu: 1500
      identifier: mac-address
      mac-address: "${CONTROLPLANE_MAC}"
      ipv4:
        address:
        - ip: "${CONTROLPLANE_IP}"
          prefix-length: "${CONTROLPLANE_PREFIX}"
        enabled: true
        dhcp: false
```

```

- name: floating
  type: vlan
  state: up
  vlan:
    base-iface: ${CONTROLPLANE_INTERFACE}
    id: ${VLAN_ID}
  dns-resolver:
    config:
      server:
        - "${DNS_SERVER}"
  routes:
    config:
      - destination: 0.0.0.0/0
        next-hop-address: "${CONTROLPLANE_GATEWAY}"
        next-hop-interface: ${CONTROLPLANE_INTERFACE}

```

ご覧のとおり、この例は、静的IPを使用してインタフェースを有効にするための設定と、ベースインタフェースを使用してVLANを有効にするための設定を示しています。インフラストラクチャに応じて、以下の変数を実際の値に置き換えてください。

- `${CONTROLPLANE1_INTERFACE}` — エッジクラスタに使用するコントロールプレーンインタフェース(例: `eth0`)。 `identifier: mac-address`を含めると、名前がMACアドレスに基づいて自動的に検査されるため、任意のインタフェース名を使用できます。
- `${CONTROLPLANE_IP}` — エッジクラスタのエンドポイントとして使用するIPアドレス (kubeapiサーバのエンドポイントと一致する必要があります)。
- `${CONTROLPLANE_PREFIX}` — エッジクラスタに使用するCIDR (例: `/24`または`255.255.255.0`を使用する場合には`24`)。
- `${CONTROLPLANE_GATEWAY}` — エッジクラスタに使用するゲートウェイ(例: `192.168.100.1`)。
- `${CONTROLPLANE_MAC}` — コントロールプレーンインタフェースに使用するMACアドレス(例: `00:0c:29:3e:3e:3e`)。
- `${DNS_SERVER}` — エッジクラスタに使用するDNS (例: `192.168.100.2`)。
- `${VLAN_ID}` — エッジクラスタに使用するVLAN ID (例: `100`)。

他の`nmstate`準拠の定義を使用して、ダウンストリームクラスタのネットワークを特定の要件に適合するように設定できます。たとえば、静的デュアルスタック設定を指定できます。

```

apiVersion: v1
kind: Secret
metadata:

```

```

name: controlplane-0-networkdata
type: Opaque
stringData:
  networkData: |
    interfaces:
    - name: ${CONTROLPLANE_INTERFACE}
      type: ethernet
      state: up
      mac-address: ${CONTROLPLANE_MAC}
      ipv4:
        enabled: true
        dhcp: false
        address:
        - ip: ${CONTROLPLANE_IP_V4}
          prefix-length: ${CONTROLPLANE_PREFIX_V4}
      ipv6:
        enabled: true
        dhcp: false
        autoconf: false
        address:
        - ip: ${CONTROLPLANE_IP_V6}
          prefix-length: ${CONTROLPLANE_PREFIX_V6}
    routes:
    config:
    - destination: 0.0.0.0/0
      next-hop-address: ${CONTROLPLANE_GATEWAY_V4}
      next-hop-interface: ${CONTROLPLANE_INTERFACE}
    - destination: ::/0
      next-hop-address: ${CONTROLPLANE_GATEWAY_V6}
      next-hop-interface: ${CONTROLPLANE_INTERFACE}
    dns-resolver:
    config:
    server:
    - ${DNS_SERVER_V4}
    - ${DNS_SERVER_V6}

```

前の例と同様に、インフラストラクチャに応じて、以下の変数を実際の値に置き換えてください。

- `${CONTROLPLANE_IP_V4}` - ホストに割り当てるIPv4アドレス
- `${CONTROLPLANE_PREFIX_V4}` - ホストIPが属するネットワークのIPv4プレフィックス
- `${CONTROLPLANE_IP_V6}` - ホストに割り当てるIPv6アドレス
- `${CONTROLPLANE_PREFIX_V6}` - ホストIPが属するネットワークのIPv6プレフィックス
- `${CONTROLPLANE_GATEWAY_V4}` - デフォルトルートに一致するトラフィック用のゲートウェイのIPv4アドレス

- `${CONTROLPLANE_GATEWAY_V6}` - デフォルトルートに一致するトラフィック用のゲートウェイのIPv6アドレス
- `${CONTROLPLANE_INTERFACE}` - IPv4とIPv6の両方で、デフォルトルートに一致する出力トラフィックにアドレスを割り当て、使用するインタフェースの名前
- `${DNS_SERVER_V4}`および/または`${DNS_SERVER_V6}` - 使用するDNSサーバのIPアドレス。単一または複数のエントリとして指定できます。IPv4および/またはIPv6アドレスがサポートされています。

注記

- IPv6専用設定およびデュアルスタック設定を含む、より複雑な例については、[SUSE Telco Cloud例のリポジトリ \(https://github.com/suse-edge/atip/tree/main/telco-examples/edge-clusters\)](https://github.com/suse-edge/atip/tree/main/telco-examples/edge-clusters)  を参照してください。
- シングルスタックのIPv6デプロイメントは技術プレビュー状態であり、公式にはサポートされていません。

最後に、ネットワーク設定の詳細に関わらず、ホストを管理クラスタに正常に登録するために、`BareMetalHost`オブジェクトに`preprovisioningNetworkDataName`を追加してシークレットが参照されるようにします。

```
apiVersion: v1
kind: Secret
metadata:
  name: example-demo-credentials
type: Opaque
data:
  username: ${BMC_USERNAME}
  password: ${BMC_PASSWORD}
---
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: example-demo
labels:
  cluster-role: control-plane
spec:
  architecture: x86_64
  online: true
  bootMACAddress: ${BMC_MAC}
  rootDeviceHints:
```

```
deviceName: /dev/nvme0n1
bmc:
  address: ${BMC_ADDRESS}
  disableCertificateVerification: true
  credentialsName: example-demo-credentials
  provisioningNetworkDataName: controlplane-0-networkdata
```



注記

- マルチノードクラスタをデプロイする必要がある場合は、同じプロセスを各ノードに対して実行する必要があります。
- `Metal3DataTemplate`、`networkData`、および `Metal3 IPAM` は現在サポートされていません。静的シークレットを介した設定のみが完全にサポートされています。
- アーキテクチャは、登録するベアメタルホストのアーキテクチャに応じて、`x86_64` または `aarch64` のいずれかである必要があります。

42.7 通信機能(DPDK、SR-IOV、CPUの分離、Huge Page、NUMAなど)

ダイレクトネットワークプロビジョニングのワークフローでは、ダウンストリームクラスタで使用する通信機能を自動化して、そのサーバ上で通信ワークロードを実行できます。

要件

- 前のセクション(42.2項「[接続シナリオのダウンストリームクラスタイメージの準備](#)」)で説明されているように、EIBを使用して生成したイメージが、こちらのセクション(注記)で設定した正確なパス上の管理クラスタに配置されている。
- こちらのセクション(42.2.2.5項「[通信ワークロードの追加設定](#)」)に従って、EIBを使用して生成したイメージに特定の通信機能を含める必要がある。
- 管理サーバが作成されていて、次の各セクションで使用できるようになっている。詳細については、管理クラスタに関するセクション第40章「[管理クラスタの設定](#)」を参照してください。

設定

次の2つのセクションをベースとして使用し、ホストを登録してプロビジョニングします。

- [ダイレクトネットワークプロビジョニングを使用したダウンストリームクラスタのプロビジョニング\(シングルノード\)](#) (42.4項「[ダイレクトネットワークプロビジョニングを使用したダウンストリームクラスタのプロビジョニング\(シングルノード\)](#)」)
- [ダイレクトネットワークプロビジョニングを使用したダウンストリームクラスタのプロビジョニング\(マルチノード\)](#) (42.5項「[ダイレクトネットワークプロビジョニングを使用したダウンストリームクラスタのプロビジョニング\(マルチノード\)](#)」)

このセクションで説明する通信機能を次に示します。

- DPDKとVFの作成
- ワークロードで使用されるSR-IOVとVFの割り当て
- CPUの分離とパフォーマンスの調整
- Huge Pageの設定
- カーネルパラメータの調整



注記

通信機能の詳細については、[第41章「通信機能の設定」](#)を参照してください。

上記の通信機能を有効にするために必要な変更はすべて、プロビジョニングファイル`capi-provisioning-example.yaml`の`RKE2ControlPlane`ブロック内にあります。ファイル`capi-provisioning-example.yaml`内の残りの情報は、プロビジョニングに関するセクション([42.4項「ダイレクトネットワークプロビジョニングを使用したダウンストリームクラスタのプロビジョニング\(シングルノード\)」](#) (488 ページ))で指定した情報と同じです。

このプロセスを明確にするために、通信機能を有効にするためにそのブロック(`RKE2ControlPlane`)で必要な変更を次に示します。

- `RKE2`インストールプロセスの前にコマンドを実行するために使用する`preRKE2Commands`。ここでは、`modprobe`コマンドを使用して、`vfio-pci`と`SR-IOV`のカーネルモジュールを有効にします。
- 作成してワークロードに公開するインタフェース、ドライバ、および`VF`の数を定義するために使用するIgnitionファイル`/var/lib/rancher/rke2/server/manifests/configmap-sriov-custom-auto.yaml`。
 - 実際の値に置き換える値は、設定マップ`sriov-custom-auto-config`内の値のみです。

- `${RESOURCE_NAME1}` — 最初のPFインタフェースに使用するリソース名(例: `sriov-resource-du1`)。このリソース名はプレフィックス `rancher.io` に追加されてラベルとして使用され、ワークロードで使用されます(例: `rancher.io/sriov-resource-du1`)。
- `${SRIOV-NIC-NAME1}` — 使用する最初のPFインタフェースの名前(例: `eth0`)。
- `${PF_NAME1}` — 使用する最初の物理機能PFの名前。これを使用して、より複雑なフィルタを生成します(例: `eth0#2-5`)。
- `${DRIVER_NAME1}` — 最初のVFインタフェースに使用するドライバ名(例: `vfio-pci`)。
- `${NUM_VFS1}` — 最初のPFインタフェース用に作成するVFの数(例: `8`)。
- `/var/sriov-auto-filler.sh` は、高レベルの設定マップ `sriov-custom-auto-config` と、低レベルのハードウェア情報を含む `sriovnetworknodepolicy` との間で情報を変換するために使用されます。このスクリプトは、ユーザがハードウェア情報を事前に把握する手間をなくすために作成されています。このファイルを変更する必要はありませんが、`sr-iov` を有効にしてVFを作成する必要がある場合は、このファイルが存在する必要があります。
- 次の機能を有効にするために使用するカーネル引数:

パラメータ	値	説明
<code>isolcpus</code>	<code>domain</code> 、 <code>nohz</code> 、 <code>managed_irq</code> 、 <code>1-30</code> 、 <code>33-62</code>	コア1-30および33-62を分離します。
<code>skew_tick</code>	<code>1</code>	分離されたCPU間でカーネルがタイマー割り込みをずらすことができるようにします。
<code>nohz</code>	<code>on</code>	システムがアイドル状態のときにカーネルが1つのCPUでタイマーティックを実行できるようにします。

nohz_full	1-30、33-62	カーネルブートパラメータは、完全なdynticksとCPU分離の設定を行うための現在の主要インタフェースです。
rcu_nocbs	1-30、33-62	システムがアイドル状態のときにカーネルが1つのCPUでRCUコールバックを実行できるようにします。
irqaffinity	0、31、32、63	システムがアイドル状態のときにカーネルが1つのCPUで割り込みを実行できるようにします。
idle	poll	アイドル状態から抜け出すまでのレイテンシを最小化します。
iommu	pt	vfioをdpdkインタフェースに使用できるようにします。
intel_iommu	on	vfioをVFに使用できるようにします。
hugepagesz	1G	Huge Pageのサイズを1Gに設定できるようにします。
hugepages	40	前に定義したHuge Pageの数。
default_hugepagesz	1G	Huge Pageを有効にする場合のデフォルト値。
nowatchdog		ウォッチドッグを無効にします。
nmi_watchdog	0	NMIウォッチドッグを無効にします。

- 次のsystemdサービスは、以下のサービスを有効にするために使用します。

- `rke2-preinstall.service` - プロビジョニングプロセス中にIronicの情報を利用してBAREMETALHOST_UUIDとnode-nameを自動的に置き換えます。
- `cpu-partitioning.service` - CPUの分離コアを有効にします(例: 1-30,33-62)。
- `performance-settings.service` - CPUのパフォーマンス調整を有効にします。
- `sriov-custom-auto-vfs.service` - sriov Helmチャートをインストールし、カスタムリソースが作成されるまで待機し、`/var/sriov-auto-filler.sh`を実行して設定マップ`sriov-custom-auto-config`の値を置き換えてワークロードが使用する`sriovnetworknodepolicy`を作成します。
- `${RKE2_VERSION}`は使用するRKE2のバージョンであり、この値は置き換えます(例: `v1.33.3+rke2r1`)。

これらの変更をすべて行くと、`capi-provisioning-example.yaml`のRKE2ControlPlaneブロックは次のようになります。

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
metadata:
  name: single-node-cluster
  namespace: default
spec:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3MachineTemplate
    name: single-node-cluster-controlplane
  replicas: 1
  version: ${RKE2_VERSION}
  rolloutStrategy:
    type: "RollingUpdate"
    rollingUpdate:
      maxSurge: 0
  serverConfig:
    cni: calico
    cniMultusEnable: true
  preRKE2Commands:
    - modprobe vfio-pci enable_sriov=1 disable_idle_d3=1
  agentConfig:
    format: ignition
    additionalUserData:
      config: |
        variant: fcos
        version: 1.4.0
        storage:
          files:
```

```

- path: /var/lib/rancher/rke2/server/manifests/configmap-sriov-custom-
auto.yaml
  overwrite: true
  contents:
    inline: |
      apiVersion: v1
      kind: ConfigMap
      metadata:
        name: sriov-custom-auto-config
        namespace: kube-system
      data:
        config.json: |
          [
            {
              "resourceName": "${RESOURCE_NAME1}",
              "interface": "${SRIOV-NIC-NAME1}",
              "pfname": "${PF_NAME1}",
              "driver": "${DRIVER_NAME1}",
              "numVFsToCreate": ${NUM_VFS1}
            },
            {
              "resourceName": "${RESOURCE_NAME2}",
              "interface": "${SRIOV-NIC-NAME2}",
              "pfname": "${PF_NAME2}",
              "driver": "${DRIVER_NAME2}",
              "numVFsToCreate": ${NUM_VFS2}
            }
          ]
      mode: 0644
      user:
        name: root
      group:
        name: root
- path: /var/lib/rancher/rke2/server/manifests/sriov-crd.yaml
  overwrite: true
  contents:
    inline: |
      apiVersion: helm.cattle.io/v1
      kind: HelmChart
      metadata:
        name: sriov-crd
        namespace: kube-system
      spec:
        chart: oci://registry.suse.com/edge/charts/sriov-crd
        targetNamespace: sriov-network-operator
        version: 304.0.2+up1.5.0
        createNamespace: true
- path: /var/lib/rancher/rke2/server/manifests/sriov-network-operator.yaml
  overwrite: true
  contents:
    inline: |

```

```

    apiVersion: helm.cattle.io/v1
    kind: HelmChart
    metadata:
      name: sriov-network-operator
      namespace: kube-system
    spec:
      chart: oci://registry.suse.com/edge/charts/sriov-network-operator
      targetNamespace: sriov-network-operator
      version: 304.0.2+up1.5.0
      createNamespace: true
  kernel_arguments:
    should_exist:
      - intel_iommu=on
      - iommu=pt
      - idle=poll
      - mce=off
      - hugepagesz=1G hugepages=40
      - hugepagesz=2M hugepages=0
      - default_hugepagesz=1G
      - irqaffinity=${NON-ISOLATED_CPU_CORES}
      - isolcpus=domain,nohz,managed_irq,${ISOLATED_CPU_CORES}
      - nohz_full=${ISOLATED_CPU_CORES}
      - rcu_nocbs=${ISOLATED_CPU_CORES}
      - rcu_nocb_poll
      - nosoftlockup
      - nowatchdog
      - nohz=on
      - nmi_watchdog=0
      - skew_tick=1
      - quiet
  systemd:
    units:
      - name: rke2-preinstall.service
        enabled: true
        contents: |
          [Unit]
          Description=rke2-preinstall
          Wants=network-online.target
          Before=rke2-install.service
          ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
          [Service]
          Type=oneshot
          User=root
          ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
          ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/${jq -r .uuid /mnt/
openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
          ExecStart=/bin/sh -c "echo \"node-name: ${jq -r .name /mnt/openstack/
latest/meta_data.json}\" >> /etc/rancher/rke2/config.yaml"
          ExecStartPost=/bin/sh -c "umount /mnt"
          [Install]
          WantedBy=multi-user.target

```

```

- name: cpu-partitioning.service
  enabled: true
  contents: |
    [Unit]
    Description=cpu-partitioning
    Wants=network-online.target
    After=network.target network-online.target
    [Service]
    Type=oneshot
    User=root
    ExecStart=/bin/sh -c "echo isolated_cores=${ISOLATED_CPU_CORES} > /etc/
tuned/cpu-partitioning-variables.conf"
    ExecStartPost=/bin/sh -c "tuned-adm profile cpu-partitioning"
    ExecStartPost=/bin/sh -c "systemctl enable tuned.service"
    [Install]
    WantedBy=multi-user.target
- name: performance-settings.service
  enabled: true
  contents: |
    [Unit]
    Description=performance-settings
    Wants=network-online.target
    After=network.target network-online.target cpu-partitioning.service
    [Service]
    Type=oneshot
    User=root
    ExecStart=/bin/sh -c "/opt/performance-settings/performance-settings.sh"
    [Install]
    WantedBy=multi-user.target
- name: sriov-custom-auto-vfs.service
  enabled: true
  contents: |
    [Unit]
    Description=SRIOV Custom Auto VF Creation
    Wants=network-online.target rke2-server.target
    After=network.target network-online.target rke2-server.target
    [Service]
    User=root
    Type=forking
    TimeoutStartSec=900
    ExecStart=/bin/sh -c "while ! /var/lib/rancher/rke2/bin/kubectl --
kubeconfig=/etc/rancher/rke2/rke2.yaml wait --for condition=ready nodes --all ; do sleep
2 ; done"
    ExecStartPost=/bin/sh -c "while [ $(/var/lib/rancher/
rke2/bin/kubectl --kubeconfig=/etc/rancher/rke2/rke2.yaml get
sriovnetworknodestates.sriovnetwork.openshift.io --ignore-not-found --no-headers -A | wc
-l) -eq 0 ]; do sleep 1; done"
    ExecStartPost=/bin/sh -c "/opt/sriov/sriov-auto-filler.sh"
    RemainAfterExit=yes
    KillMode=process
    [Install]

```

```
WantedBy=multi-user.target

kubelet:
  extraArgs:
    - provider-id=metal3://BAREMETALHOST_UUID
  nodeName: "localhost.localdomain"
```

前の各ブロックを結合してファイルを作成したら、管理クラスタで次のコマンドを実行し、通信機能を使用する新しいダウンストリームクラスタのプロビジョニングを開始する必要があります。

```
$ kubectl apply -f capi-provisioning-example.yaml
```

42.8 プライベートレジストリ

ワークロードで使用するイメージのミラーとしてプライベートレジストリを設定できます。そのために、ダウンストリームクラスタで使用するプライベートレジストリに関する情報を含むシークレットを作成します。

```
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}
```

tls.crt、tls.key、およびca.crtは、プライベートレジストリを認証するために使用する証明書です。usernameおよびpasswordは、プライベートレジストリを認証するために使用する資格情報です。



注記

tls.crt、tls.key、ca.crt、username、およびpasswordは、シークレットで使用する前にbase64形式でエンコードする必要があります。

これらの変更をすべて行くと、capi-provisioning-example.yamlのRKE2ControlPlaneブロックは次のようになります。

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
metadata:
  name: single-node-cluster
  namespace: default
spec:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3MachineTemplate
    name: single-node-cluster-controlplane
  replicas: 1
  version: ${RKE2_VERSION}
  rolloutStrategy:
    type: "RollingUpdate"
    rollingUpdate:
      maxSurge: 0
  privateRegistriesConfig:
    mirrors:
      "registry.example.com":
        endpoint:
          - "https://registry.example.com:5000"
    configs:
      "registry.example.com":
        authSecret:
          apiVersion: v1
          kind: Secret
          namespace: default
          name: private-registry-auth
        tls:
          tlsConfigSecret:
            apiVersion: v1
            kind: Secret
            namespace: default
            name: private-registry-cert
  serverConfig:
    cni: calico
    cniMultusEnable: true
  agentConfig:
    format: ignition
    additionalUserData:
```

```

config: |
  variant: fcso
  version: 1.4.0
  systemd:
    units:
      - name: rke2-preinstall.service
        enabled: true
        contents: |
          [Unit]
          Description=rke2-preinstall
          Wants=network-online.target
          Before=rke2-install.service
          ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
          [Service]
          Type=oneshot
          User=root
          ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
          ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/${jq -r .uuid /mnt/
openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
          ExecStart=/bin/sh -c "echo \"node-name: ${jq -r .name /mnt/openstack/
latest/meta_data.json}\" >> /etc/rancher/rke2/config.yaml"
          ExecStartPost=/bin/sh -c "umount /mnt"
          [Install]
          WantedBy=multi-user.target
  kubelet:
    extraArgs:
      - provider-id=metal3://BAREMETALHOST_UUID
    nodeName: "localhost.localdomain"

```

ここで、`registry.example.com`は、ダウンストリームクラスタで使用するプライベートレジストリの名前の例です。これは実際の値に置き換える必要があります。

42.9 エアギャップシナリオでのダウンストリームクラスタのプロビジョニング

ダイレクトネットワークプロビジョニングワークフローでは、エアギャップシナリオでのダウンストリームクラスタのプロビジョニングを自動化できます。

42.9.1 エアギャップシナリオの要件

1. EIBを使用して生成された生のイメージには、エアギャップシナリオでダウンストリームクラスタを実行するために必要な特定のコンテナイメージ(HelmチャートOCIイメージとコンテナイメージ)を含める必要があります。詳細については、こちらのセクション(42.3項「エアギャップシナリオ用のダウンストリームクラスタイメージの準備」)を参照してください。
2. SR-IOVまたはその他のカスタムワークロードを使用する場合、プライベートレジストリへのプリロードに関するセクション(42.3.2.7項「エアギャップシナリオおよびSR-IOV(オプション)に必要なイメージのプライベートレジストリへのプリロード」)に従って、ワークロードを実行するために必要なイメージをプライベートレジストリにプリロードする必要があります。

42.9.2 エアギャップシナリオでのベアメタルホストの登録

管理クラスタにベアメタルホストを登録するプロセスは、前のセクション(42.4項「ダイレクトネットワークプロビジョニングを使用したダウンストリームクラスタのプロビジョニング(シングルノード)」(486 ページ))で説明したプロセスと同じです。

42.9.3 エアギャップシナリオでのダウンストリームクラスタのプロビジョニング

エアギャップシナリオでダウンストリームクラスタをプロビジョニングするために必要となる重要な変更がいくつかあります。

1. capi-provisioning-example.yamlファイルのRKE2ControlPlaneブロックにspec.agentConfig.airGapped: trueディレクティブを含める必要があります。
2. プライベートレジストリに関するセクション(42.8項「プライベートレジストリ」)に従って、プライベートレジストリの設定をcapi-provisioning-airgap-example.yamlファイルのRKE2ControlPlaneブロックに含める必要があります。
3. SR-IOV、またはHelmチャートをインストールする必要があるその他のAdditionalUserData設定(combustionスクリプト)を使用している場合、内容を変更して、パブリックレジストリを使用するのではなくプライベートレジストリを参照する必要があります。

次の例は、プライベートレジストリを参照するために必要な変更を行った、`capi-provisioning-airgap-example.yaml`ファイルのAdditionalUserDataブロックのSR-IOVの設定を示しています。

- プライベートレジストリのシークレットの参照
- パブリックOCIイメージではなくプライベートレジストリを使用するHelmチャートの定義

```
# secret to include the private registry certificates
apiVersion: v1
kind: Secret
metadata:
  name: private-registry-cert
  namespace: default
data:
  tls.crt: ${TLS_BASE64_CERT}
  tls.key: ${TLS_BASE64_KEY}
  ca.crt: ${CA_BASE64_CERT}
type: kubernetes.io/tls
---
# secret to include the private registry auth credentials
apiVersion: v1
kind: Secret
metadata:
  name: private-registry-auth
  namespace: default
data:
  username: ${REGISTRY_USERNAME}
  password: ${REGISTRY_PASSWORD}
---
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
metadata:
  name: single-node-cluster
  namespace: default
spec:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3MachineTemplate
    name: single-node-cluster-controlplane
  replicas: 1
  version: ${RKE2_VERSION}
  rolloutStrategy:
    type: "RollingUpdate"
    rollingUpdate:
      maxSurge: 0
  privateRegistriesConfig:      # Private registry configuration to add your own mirror
                                and credentials
```

```

mirrors:
  docker.io:
    endpoint:
      - "https://$(PRIVATE_REGISTRY_URL)"
configs:
  "192.168.100.22:5000":
    authSecret:
      apiVersion: v1
      kind: Secret
      namespace: default
      name: private-registry-auth
    tls:
      tlsConfigSecret:
        apiVersion: v1
        kind: Secret
        namespace: default
        name: private-registry-cert
      insecureSkipVerify: false
serverConfig:
  cni: calico
  cniMultusEnable: true
preRKE2Commands:
  - modprobe vfio-pci enable_sriov=1 disable_idle_d3=1
agentConfig:
  airGapped: true          # Airgap true to enable airgap mode
  format: ignition
  additionalUserData:
    config: |
      variant: fcos
      version: 1.4.0
      storage:
        files:
          - path: /var/lib/rancher/rke2/server/manifests/configmap-sriov-custom-
auto.yaml
  overwrite: true
  contents:
    inline: |
      apiVersion: v1
      kind: ConfigMap
      metadata:
        name: sriov-custom-auto-config
        namespace: sriov-network-operator
      data:
        config.json: |
          [
            {
              "resourceName": "${RESOURCE_NAME1}",
              "interface": "${SRIOV-NIC-NAME1}",
              "pfname": "${PF_NAME1}",
              "driver": "${DRIVER_NAME1}",
              "numVFsToCreate": ${NUM_VFS1}

```

```

        },
        {
            "resourceName": "${RESOURCE_NAME2}",
            "interface": "${SRIOV-NIC-NAME2}",
            "pfname": "${PF_NAME2}",
            "driver": "${DRIVER_NAME2}",
            "numVFsToCreate": ${NUM_VFS2}
        }
    ]
mode: 0644
user:
  name: root
group:
  name: root
- path: /var/lib/rancher/rke2/server/manifests/sriov.yaml
  overwrite: true
  contents:
    inline: |
      apiVersion: v1
      data:
        .dockerconfigjson: ${REGISTRY_AUTH_DOCKERCONFIGJSON}
      kind: Secret
      metadata:
        name: privregauth
        namespace: kube-system
      type: kubernetes.io/dockerconfigjson
    ---
    apiVersion: v1
    kind: ConfigMap
    metadata:
      namespace: kube-system
      name: example-repo-ca
    data:
      ca.crt: |-
        -----BEGIN CERTIFICATE-----
        ${CA_BASE64_CERT}
        -----END CERTIFICATE-----
    ---
    apiVersion: helm.cattle.io/v1
    kind: HelmChart
    metadata:
      name: sriov-crd
      namespace: kube-system
    spec:
      chart: oci://${PRIVATE_REGISTRY_URL}/sriov-crd
      dockerRegistrySecret:
        name: privregauth
      repoCAConfigMap:
        name: example-repo-ca
      createNamespace: true
      set:

```

```

    global.clusterCIDR: 192.168.0.0/18
    global.clusterCIDRv4: 192.168.0.0/18
    global.clusterDNS: 10.96.0.10
    global.clusterDomain: cluster.local
    global.rke2DataDir: /var/lib/rancher/rke2
    global.serviceCIDR: 10.96.0.0/12
    targetNamespace: sriov-network-operator
    version: 304.0.2+up1.5.0
  ---
  apiVersion: helm.cattle.io/v1
  kind: HelmChart
  metadata:
    name: sriov-network-operator
    namespace: kube-system
  spec:
    chart: oci://${PRIVATE_REGISTRY_URL}/sriov-network-operator
    dockerRegistrySecret:
      name: privregauth
    repoCAConfigMap:
      name: example-repo-ca
    createNamespace: true
    set:
      global.clusterCIDR: 192.168.0.0/18
      global.clusterCIDRv4: 192.168.0.0/18
      global.clusterDNS: 10.96.0.10
      global.clusterDomain: cluster.local
      global.rke2DataDir: /var/lib/rancher/rke2
      global.serviceCIDR: 10.96.0.0/12
      targetNamespace: sriov-network-operator
      version: 304.0.2+up1.5.0
  mode: 0644
  user:
    name: root
  group:
    name: root
  kernel_arguments:
    should_exist:
      - intel_iommu=on
      - iommu=pt
      - idle=poll
      - mce=off
      - hugepagesz=1G hugepages=40
      - hugepagesz=2M hugepages=0
      - default_hugepagesz=1G
      - irqaffinity=${NON-ISOLATED_CPU_CORES}
      - isolcpus=domain,nohz,managed_irq,${ISOLATED_CPU_CORES}
      - nohz_full=${ISOLATED_CPU_CORES}
      - rcu_nocbs=${ISOLATED_CPU_CORES}
      - rcu_nocb_poll
      - nosoftlockup
      - nowatchdog

```

```

- nohz=on
- nmi_watchdog=0
- skew_tick=1
- quiet
systemd:
  units:
    - name: rke2-preinstall.service
      enabled: true
      contents: |
        [Unit]
        Description=rke2-preinstall
        Wants=network-online.target
        Before=rke2-install.service
        ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
        [Service]
        Type=oneshot
        User=root
        ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
        ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/${jq -r .uuid /mnt/
openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
        ExecStart=/bin/sh -c "echo \"node-name: ${jq -r .name /mnt/openstack/
latest/meta_data.json}\" >> /etc/rancher/rke2/config.yaml"
        ExecStartPost=/bin/sh -c "umount /mnt"
        [Install]
        WantedBy=multi-user.target
    - name: cpu-partitioning.service
      enabled: true
      contents: |
        [Unit]
        Description=cpu-partitioning
        Wants=network-online.target
        After=network.target network-online.target
        [Service]
        Type=oneshot
        User=root
        ExecStart=/bin/sh -c "echo isolated_cores=${ISOLATED_CPU_CORES} > /etc/
tuned/cpu-partitioning-variables.conf"
        ExecStartPost=/bin/sh -c "tuned-adm profile cpu-partitioning"
        ExecStartPost=/bin/sh -c "systemctl enable tuned.service"
        [Install]
        WantedBy=multi-user.target
    - name: performance-settings.service
      enabled: true
      contents: |
        [Unit]
        Description=performance-settings
        Wants=network-online.target
        After=network.target network-online.target cpu-partitioning.service
        [Service]
        Type=oneshot
        User=root

```

```

        ExecStart=/bin/sh -c "/opt/performance-settings/performance-settings.sh"
        [Install]
        WantedBy=multi-user.target
- name: sriov-custom-auto-vfs.service
  enabled: true
  contents: |
    [Unit]
    Description=SRIOV Custom Auto VF Creation
    Wants=network-online.target rke2-server.target
    After=network.target network-online.target rke2-server.target
    [Service]
    User=root
    Type=forking
    TimeoutStartSec=1800
    ExecStart=/bin/sh -c "while ! /var/lib/rancher/rke2/bin/kubectl --
kubecfg=/etc/rancher/rke2/rke2.yaml wait --for condition=ready nodes --timeout=30m --
all ; do sleep 10 ; done"
    ExecStartPost=/bin/sh -c "/opt/sriov/sriov-auto-filler.sh"
    RemainAfterExit=yes
    KillMode=process
    [Install]
    WantedBy=multi-user.target
  kubelet:
    extraArgs:
      - provider-id=metal3://BAREMETALHOST_UUID
    nodeName: "localhost.localdomain"

```

43 ライフサイクルのアクション

このセクションでは、SUSE Telco Cloudを介してデプロイされたクラスタのライフサイクル管理アクションについて説明します。

43.1 管理クラスタのアップグレード

管理クラスタのアップグレードには複数のコンポーネントが関係します。アップグレードする必要がある一般的なコンポーネントのリストについては、[Day 2管理クラスタ\(第35章「管理クラスタ」\)](#)のドキュメントを参照してください。

このセットアップに固有のコンポーネントのアップグレード手順は以下のとおりです。

Metal³のアップグレード

Metal3をアップグレードするには、次のコマンドを使用してHelmリポジトリキャッシュを更新し、最新のチャートをフェッチしてHelmチャートリポジトリからMetal3をインストールします。

```
helm repo update
helm fetch suse-edge/metal3
```

その後、現在の設定をファイルにエクスポートしてから、その前のファイルを使用してMetal3のバージョンをアップグレードすると、簡単にアップグレードできます。新しいバージョンで何らかの変更が必要な場合、アップグレードの前にそのファイルを編集できます。

```
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=304.0.16+up0.12.6
```

43.2 ダウンストリームクラスタのアップグレード

ダウンストリームクラスタをアップグレードするには、複数のコンポーネントを更新する必要があります。次の各セクションでは、各コンポーネントのアップグレードプロセスについて説明します。

オペレーティングシステムのアップグレード

このプロセスでは、次の参照資料(42.2項「[接続シナリオのダウンストリームクラスイメージの準備](#)」)を確認して、新しいオペレーティングシステムバージョンで新しいイメージを構築します。EIBで生成されたこの新しいイメージにより、次のプロビジョニングフェーズでは、指定した新しいオペレーティングシステムバージョンが使用されます。次の手順では、この新しいイメージを使用してノードをアップグレードします。

RKE2クラスタのアップグレード

自動化されたワークフローを使用してRKE2クラスタをアップグレードするために必要な変更は次のとおりです。

- 次のセクション(42.4項「[ダイレクトネットワークプロビジョニングを使用したダウンストリームクラスタのプロビジョニング\(シングルノード\)](#)」(488 ページ))に示す `capi-provisioning-example.yaml` のブロック `RKE2ControlPlane` を次のように変更します。
 - 目的の `rolloutStrategy` を指定します。
 - `${RKE2_NEW_VERSION}` を置き換えて、RKE2クラスタのバージョンを新しいバージョンに変更します。

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
metadata:
  name: single-node-cluster
  namespace: default
spec:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3MachineTemplate
    name: single-node-cluster-controlplane
  version: ${RKE2_NEW_VERSION}
  replicas: 1
  rolloutStrategy:
    type: "RollingUpdate"
    rollingUpdate:
      maxSurge: 0
  serverConfig:
    cni: cilium
  rolloutStrategy:
    rollingUpdate:
      maxSurge: 0
  registrationMethod: "control-plane-endpoint"
  agentConfig:
    format: ignition
    additionalUserData:
      config: |
        variant: fcos
        version: 1.4.0
```



```

systemd:
  units:
    - name: rke2-preinstall.service
      enabled: true
      contents: |
        [Unit]
        Description=rke2-preinstall
        Wants=network-online.target
        Before=rke2-install.service
        ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
        [Service]
        Type=oneshot
        User=root
        ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
        ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/${jq -r .uuid /mnt/
openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
        ExecStart=/bin/sh -c "echo \"node-name: ${jq -r .name /mnt/openstack/
latest/meta_data.json}\" >> /etc/rancher/rke2/config.yaml"
        ExecStartPost=/bin/sh -c "umount /mnt"
        [Install]
        WantedBy=multi-user.target
kubenet:
  extraArgs:
    - provider-id=metal3://BAREMETALHOST_UUID
  nodeName: "localhost.localdomain"

```

- 次のセクション(42.4項「[ダイレクトネットワークプロビジョニングを使用したダウンロードストリームクラスタのプロビジョニング\(シングルノード\)](#)」(488 ページ))に示す [capi-provisioning-example.yaml](#) のブロック [Metal3MachineTemplate](#) を次のように変更します。
 - イメージ名およびチェックサムを、前の手順で生成した新しいバージョンに変更します。
 - ディレクティブ [nodeReuse](#) を追加して [true](#) に設定し、新しいノードが作成されないようにします。
 - ディレクティブ [automatedCleaningMode](#) を追加して [metadata](#) に設定し、ノードの自動クリーニングを有効にします。

```

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
```

これらの変更を行った後、次にコマンドを使用して capi-provisioning-example.yaml ファイルをクラスタに適用できます。

```
kubectl apply -f capi-provisioning-example.yaml
```

VIII **トラブルシューティング**

- 44 一般的なトラブルシューティングの原則 **532**
- 45 Kiwiのトラブルシューティング **534**
- 46 Edge Image Builder (EIB)のトラブルシューティング **536**
- 47 Edgeネットワーキング(NMC)のトラブルシューティング **538**
- 48 Phone-Homeシナリオのトラブルシューティング **540**
- 49 ダイレクトネットワークプロビジョニングのトラブルシューティング **541**
- 50 その他のコンポーネントのトラブルシューティング **547**
- 51 サポートのための診断情報の収集 **548**

このセクションでは、SUSE Edgeのデプロイメントと操作に関する一般的な問題を診断し、解決するためのガイダンスを提供します。さまざまなトピックを網羅していて、コンポーネント固有のトラブルシューティング手順、主要ツール、関連するログの場所を説明しています。

44 一般的なトラブルシューティングの原則

コンポーネント固有の問題に取りかかる前に、以下の一般的な原則を考慮してください。

- **ログの確認:** ログは主要な情報源です。ほとんどの場合、エラーは説明不要で、何が失敗したかに関するヒントが含まれています。
- **クロックの確認:** システム間にクロックの相違があると、あらゆる種類の異なるエラーが発生する可能性があります。クロックが同期していることを確認します。EIBにブート時にクロック同期を強制するように指示することができます。「OSの時刻の設定」(第3章「Edge Image Builderを使用したスタンドアロンクラスタ」)を参照してください。
- **ブートの問題:** システムがブート中に停止した場合、最後に表示されたメッセージをメモします。コンソール(物理的またはBMC経由)にアクセスして、ブートメッセージを確認します。
- **ネットワークの問題:** ネットワークインターフェースの設定(`ip a`)、ルーティングテーブル(`ip route`)、他のノードと外部サービスとの間のテスト接続(`ping`、`nc`)を確認します。ファイアウォールルールがネットワークポートをブロックしていないことを確認します。
- **コンポーネントのステータスの確認:** Kubernetesリソースの場合は、`kubectl get`と`kubectl describe`を使用します。特定のKubernetes名前空間のイベントを確認するには、`kubectl get events --sort-by='.lastTimestamp' -n <namespace>`を使用します。
- **サービスステータスの確認:** `systemd`サービスの場合は、`systemctl status <service>`を使用します。
- **構文の確認:** ソフトウェアは設定ファイルに特定の構造と構文を想定しています。たとえばYAMLファイルの場合、`yamllint`や類似のツールを使用して、正しい構文を確認します。
- **問題の特定:** 問題を特定のコンポーネントまたはレイヤ(たとえば、ネットワーク、ストレージ、OS、Kubernetes、Metal³、Ironicなど)に絞り込みます。
- **ドキュメント:** 詳細については、常に公式のSUSE Edgeドキュメント (<https://documentation.suse.com/suse-edge/>) およびアップストリームドキュメントを参照してください。

- **バージョン:** SUSE Edgeは、さまざまなSUSEコンポーネントを独自に徹底的にテストしたバージョンです。SUSE Edgeリリースごとの各コンポーネントのバージョンは、[SUSE Edgeサポートマトリックス \(https://documentation.suse.com/suse-edge/support-matrix/html/support-matrix/index.html\)](https://documentation.suse.com/suse-edge/support-matrix/html/support-matrix/index.html) で確認できます。
- **既知の問題:** SUSE Edgeの各リリースには、リリースノートに「既知の問題」セクションがあり、今後のリリースで修正する予定だが、現在のリリースに影響する可能性のある問題に関する情報が記載されています。

45 Kiwiのトラブルシューティング

Kiwiを使用して、Edge Image Builderで使われる、更新されたSUSE Linux Microイメージを生成します。

一般的な問題

- **SL Microバージョンの不一致:** 構築ホストのオペレーティングシステムのバージョンは、構築されるオペレーティングシステムのバージョンと一致する必要があります(SL Micro 6.0ホスト → SL Micro 6.0イメージ)。
- **SELinuxをEnforcing状態にする:** 特定の制限のため、Kiwiを使用してイメージを構築できるように、現在はSELinuxを一時的に無効にする必要があります。`getenforce`でSELinuxステータスを確認し、`setenforce 0`を使用して構築プロセスを実行する前にSELinuxを無効にしてください。
- **構築ホストが登録されていない:** 構築プロセスでは、SUSE SCCからパッケージをプルできるように、構築ホストサブスクリプションを使用します。ホストが登録されていない場合は失敗します。
- **ループデバイステスト失敗:** 初めてKiwi構築プロセスが実行されると、開始直後に失敗し、「ERROR: Early loop device test failed, please retry the container run. (エラー: ループデバイスの早期テストが失敗しました。もう一度コンテナの実行を試してください)」というエラーメッセージが表示されます。これは、基になるホストシステム上に作成されるループデバイスがすぐにはコンテナイメージ内に表示されないことを示す症状です。Kiwi構築プロセスを再実行すれば、問題なく続行されるはずです。
- **許可がない:** 構築プロセスはルートユーザとして(またはsudoを介して)実行されることが想定されています。
- **間違った権限:** 構築プロセスは、コンテナを実行する際には`--privileged`フラグが指定されていることが想定されています。このフラグが指定されていることを再度確認してください。

ログ

- **構築コンテナのログ:** 構築コンテナのログを確認します。ログはアーティファクトの保存に使用されたディレクトリに生成されます。必要な情報については、dockerログまたはpodmanログも確認してください。
- **一時構築ディレクトリ:** Kiwiは、構築プロセス中に一時ディレクトリを作成します。メイン出力が不十分な場合は、これらのディレクトリに中間ログやアーティファクトがないか確認します。

トラブルシューティング手順

1. **build-image出力の確認:** コンソール出力のエラーメッセージは通常、非常にわかりやすいものです。
2. **構築環境の確認:** Kiwiを実行しているマシンでKiwi自体のすべての前提条件(docker/ podman、SELinux, 十分なディスク容量など)が満たされていることを確認します。
3. **構築コンテナログの検査:** 失敗したコンテナのログで詳細なエラーについて確認します (上記参照)。
4. **定義ファイルの確認:** カスタムKiwiイメージ定義ファイルを使用している場合、タイポや構文についてファイルを再確認します。



注記

Kiwiトラブルシューティングガイド (<https://documentation.suse.com/appliance/kiwi-9/html/kiwi/troubleshooting.html>) を確認してください。

46 Edge Image Builder (EIB)のトラブルシューティング

EIBは、カスタムSUSE Edgeイメージを作成するために使用されます。

一般的な問題

- **間違ったSCCコード:** EIB定義ファイルに使用されているSCCコードがSL Microのバージョンとアーキテクチャに一致していることを確認します。
- **依存関係がない:** 構築環境内に不足しているパッケージやツールがないか確認します。
- **不正なイメージサイズ:** RAWイメージの場合、`diskSize`パラメータが必要であり、イメージ、RPM、およびイメージに含まれている他のアーティファクトに大きく依存します。
- **許可:** `custom/files`ディレクトリにスクリプトを保存する場合、スクリプトに実行許可があることを確認します。これらのファイルはCombustion時にのみ利用可能であり、EIBによって変更は実行されないためです。
- **オペレーティングシステムグループの依存関係:** カスタムユーザとグループを使用してイメージを作成する場合、「`primaryGroup`」として設定されるグループを明示的に作成する必要があります。
- **オペレーティングシステムユーザのsshkeysにはホームフォルダが必要:** `sshkeys`を持つユーザでイメージを作成する場合、`createHomeDir=true`を指定してホームフォルダも作成する必要があります。
- **Combustionの問題:** EIBは、OSのカスタマイズおよびその他のすべてのSUSE Edgeコンポーネントのデプロイメントに対してCombustionに依存しています。これには、`custom/scripts`フォルダに配置されるカスタムスクリプトも含まれます。Combustionプロセスは`initrd`時に実行されるため、スクリプト実行時にシステムが完全に起動していないことに注意してください。
- **Podmanマシンサイズ:** EIBの「ヒントとコツ」セクション([パートIV「ヒントとコツ」](#))で説明されているように、Linux以外のオペレーティングシステムでEIBコンテナを実行するための十分なCPU/メモリがPodmanマシンにあることを確認します。

ログ

- **EIB出力:** `eib build`コマンドのコンソール出力は重要です。

- **構築コンテナのログ:** 構築コンテナのログを確認します。ログは、アーティファクトを保存するために使用されたディレクトリに生成されます。必要な情報については[dockerログ](#)または[podmanログ](#)も確認してください。



注記

詳細については、「[Debugging \(デバッグ\)](https://github.com/suse-edge/edge-image-builder/blob/main/docs/debugging.md) (<https://github.com/suse-edge/edge-image-builder/blob/main/docs/debugging.md>)」を参照してください。

- **一時的な構築ディレクトリ:** EIBは構築プロセス中に一時ディレクトリを作成します。メイン出力が不十分な場合は、これらのディレクトリで中間ログやアーティファクトを確認します。
- **Combustionログ:** EIBで構築されているイメージが何らかの理由で起動しない場合、ルートシェルを利用できます。ホストコンソールに(物理的に、またはBMC経由などで)接続し、`journalctl -u burning`でCombustionログを確認し、通常は[journalctl](#)ですべてのオペレーティングシステムログを確認して、失敗の根本原因を特定します。

トラブルシューティング手順

1. **eib-build出力の確認:** コンソール出力のエラーメッセージは通常、非常にわかりやすいです。
2. **構築環境の確認:** EIBを実行しているマシンで、EIB自体のすべての前提条件(docker/podman、十分なディスク容量など)が満たされていることを確認します。
3. **構築コンテナログの検査:** 失敗したコンテナのログで詳細なエラーについて確認します(上記参照)。
4. **eib設定の確認:** `eib`設定ファイルにタイポがないか、ソースファイルや構築スクリプトへのパスが間違っていないか、再確認します。
 - **コンポーネントの個別テスト:** EIBビルドにカスタムスクリプトやステージが含まれている場合は、個別に実行して失敗を特定します。



注記

Edge Image Builderの「[Debugging \(デバッグ\)](https://github.com/suse-edge/edge-image-builder/blob/main/docs/debugging.md)」(<https://github.com/suse-edge/edge-image-builder/blob/main/docs/debugging.md>)を確認してください。

47 Edgeネットワークキング(NMC)のトラブルシューティング

NMCは、SL Micro EIBイメージに挿入され、起動時に、Combustionを介してEdgeホストのネットワークを設定します。また、Metal3ワークフローでは検査プロセスの一環として実行されています。ホストの初回起動時、またはMetal3検査プロセス中に問題が発生する可能性があります。

一般的な問題

- **ホストが初回起動時に正常に起動できない:** 不正な形式のネットワーク定義ファイルにより、combustionフェーズが失敗し、ホストがルートシェルをドロップする可能性があります。
- **ファイルが適切に生成されない:** ネットワークファイルがNMState (<https://nmstate.io/examples.html>)  形式と一致していることを確認します。
- **ネットワークインタフェースが正常に設定されない:** MACアドレスがホストで使用されているインタフェースと一致していることを確認します。
- **インタフェース名の不一致:** `net.ifnames=1`カーネル引数により、[ネットワークインタフェースの予測可能な命名規則](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>)  が有効になるため、`eth0`はなくなり、`enp2s0`などの他の命名規則が使用されます。

ログ

- **Combustionログ:** Combustion時にnmcが使用されるため、プロビジョニング中のホスト上で、`journalctl -u combustion`を使用してCombustionログを確認します。
- **NetworkManagerログ:** Metal³デプロイメントワークフローでは、nmcはIPA実行の一部であり、systemdのExecStartPre機能を使用してNetworkManagerサービスの依存関係として実行されます。IPAホスト上で`journalctl -u NetworkManager`としてNetworkManagerログを確認します(IPAで起動したときにホストにアクセスする方法については、「[ダイレクトネットワークプロビジョニングのトラブルシューティング](#)」(第49章「[ダイレクトネットワークプロビジョニングのトラブルシューティング](#)」)セクションを参照してください)。

トラブルシューティング手順

1. **yaml構文の確認:** nmc設定ファイルはyamlファイルです。適切な構文かどうかをyamllintまたは同様のツールで確認します。
2. **nmcを手動で実行する:** nmcはEIBコンテナの一部であるため、問題をデバッグするには、ローカルpodmanコマンドを使用できます。

- a. nmcファイルを保存するための一時フォルダを作成します。

```
mkdir -p ${HOME}/tmp/foo
```

- b. その場所にnmcファイルを保存します。

```
> tree --noreport ${HOME}/tmp/foo
/Users/johndoe/tmp/foo
├── host1.example.com.yaml
└── host2.example.com.yaml
```

- c. エントリーポイントとしてnmcを使用してEIBコンテナを実行し、generateコマンドを使用して、Combustion時にnmcが実行するのと同じタスクを実行します。

```
podman run -it --rm -v ${HOME}/tmp/foo:/tmp/foo:Z --entrypoint=/usr/bin/nmc
registry.suse.com/edge/3.3/edge-image-builder:1.2.0 generate --config-dir /
tmp/foo --output-dir /tmp/foo/

[2025-06-04T11:58:37Z INFO  nmc::generate_conf] Generating config from "/tmp/
foo/host2.example.com.yaml"...
[2025-06-04T11:58:37Z INFO  nmc::generate_conf] Generating config from "/tmp/
foo/host1.example.com.yaml"...
[2025-06-04T11:58:37Z INFO  nmc] Successfully generated and stored network
config
```

- d. 一時フォルダに生成されるログとファイルを確認します。

48 Phone-Homeシナリオのトラブルシューティング

Phone-Homeシナリオには、Elementalを使用して管理クラスタに接続し直し、EIBを使用してElemental登録情報を含むOSイメージを作成することが含まれます。ホストの初回起動時、EIB構築プロセス中、または管理クラスタに登録しようとするときに問題が発生する可能性があります。

一般的な問題

- **システムを登録できない:** UIでノードが登録されていません。ホストが適切に起動していること、Rancherと通信できること、クロックが同期していること、Elementalサービスが正常であることを確認します。
- **システムをプロビジョニングできない:** ノードは登録されているが、プロビジョニングに失敗します。ホストがRancherと通信できること、クロックが同期していること、Elementalサービスに問題がないことを確認します。

ログ

- **システムログ:** `journalctl`
- **Elemental-system-agentログ:** `journalctl -u elemental-system-agent`
- **K3s/RKE2ログ:** `journalctl -u k3s`または`journalctl -u rke2-server` (または`rke2-agent`)
- **ElementalオペレータPod:** `kubectl logs -n cattle-elemental-system -l app=elemental-operator`

トラブルシューティング手順

1. **ログの確認:** ElementalオペレータPodのログで、問題がないかどうかを確認します。ノードが起動している場合は、ホストログを確認してください。
2. **MachineRegistrationとTPMの確認:** デフォルトで、TPMは認証 (<https://elemental.docs.rancher.com/authentication/>) に使用されますが、TPMのないホストには、代替が存在します。

49 ダイレクトネットワークプロビジョニングのトラブルシューティング

ダイレクトネットワークプロビジョニングシナリオには、Metal³およびCAPI要素を使用したダウストリームクラスタのプロビジョニングが含まれます。EIBを使用したOSイメージの作成も含まれます。ホストの初回起動時、または検査プロセスまたはプロビジョニングプロセス中に問題が発生する可能性があります。

一般的な問題

- **古いファームウェア:** 使用されている物理ホスト上のさまざまなファームウェアがすべて最新であることを確認します。これには、BMCファームウェアが含まれます。Metal³では特定のファームウェアや更新されたファームウェアが必要になる(https://book.metal3.io/bmo/supported_hardware#redfish-and-its-variants) 場合があります。
- **プロビジョニングがSSLエラーで失敗した:** イメージを提供するWebサーバがhttpsを使用する場合、Metal³はIPAイメージに証明書を挿入して信頼するように設定する必要があります。`ca-additional.crt`ファイルをMetal³チャートに含める方法については、Kubernetesフォルダ(40.3.4項「Kubernetesフォルダ」)を参照してください。
- **IPAを使用したホストの起動時に証明書の問題が発生する:** 一部のサーバベンダは仮想メディアISOイメージをBMCにアタッチする際にSSL接続を確認します。このため、Metal³デプロイメント用に生成された証明書が自己署名されているために、問題が発生する可能性があります。ホストが起動中であるにもかかわらずUEFIシェルに切り替わってしまう可能性があります。この問題の解決方法については、「仮想メディアISOをアタッチするためのTLSの無効化」(1.7.2項「仮想メディアISOをアタッチするためのTLSの無効化」)を参照してください。
- **間違った名前またはラベル参照:** クラスタが間違った名前またはラベルでノードを参照する場合、クラスタはデプロイ済みと表示されますが、BMHは「使用可能」のままとなります。BMHに関連するオブジェクトの参照を再確認します。
- **BMC通信の問題:** 管理クラスタ上で実行されているMetal³ Podがプロビジョニング中のホストのBMCにアクセスできることを確認します(通常BMCネットワークは非常に制限されています)。
- **不正なベアメタルホストの状態:** BMHオブジェクトは、その存続期間中にさまざまな状態(検査中、準備中、プロビジョニング済みなど)に移行します(存続期間中のマシンの状態 (https://book.metal3.io/bmo/state_machine))。不正な状態が検出された場

合は、BMHオブジェクトの`status`フィールドを確認してください。このフィールドには`kubectl get bmh <name> -o jsonpath='{.status}' | jq`のような詳細情報が含まれています。

- **ホストがプロビジョニング解除されない:** プロビジョニング解除しようとしていたホストが失敗した場合、BMHオブジェクトに次のように「detached」アノテーションを追加すると削除を試行できます: `kubectl annotate bmh/<BMH> baremetalhost.metal3.io/detached=""`。
- **イメージエラー:** ダウンストリームクラスタに対してEIBを使用して構築されるイメージが使用可能であり、適切なチェックサムがあり、解凍するには大きすぎない、またはディスクに対して大きすぎないことを確認します。
- **ディスクサイズの不一致:** デフォルトでは、全ディスクを満たすようにディスクは拡張されません。「Growfsスクリプト」(1.4.4.1.2項「Growfsスクリプト」)セクションで説明しているように、ダウンストリームクラスタホスト用にEIBで構築されるイメージにはgrowfsスクリプトを含める必要があります。
- **クリーニングプロセスの停止:** クリーニングプロセスは数回再試行されます。ホストの問題によりクリーニングができなくなった場合は、まず、BMHオブジェクトの`automatedCleanMode`フィールドを`disabled`に設定してクリーニングを無効にします。



警告

クリーニングプロセスに必要以上に時間がかかっている、または失敗している場合は、ファイナライザを手動で削除することは推奨されません。手動で削除すると、Kubernetesからホストレコードが削除されますが、Ironicには残ります。現在実行中のアクションはバックグラウンドで続行されるため、ホストを再度追加しようとする競合により失敗する可能性があります。

- **Metal3/Rancher Turtles/CAPI Podの問題:** すべての必要なコンポーネントのデプロイメントフローは次のとおりです。
 - Rancher TurtlesコントローラはCAPIオペレータコントローラをデプロイします。
 - 次に、CAPIオペレータコントローラはプロバイダコントローラ(CAPIコア、CAPM3、およびRKE2コントロールプレーン/ブートストラップ)をデプロイします。

すべてのPodが正常に実行されていることを確認し、そうでない場合はログを確認します。

ログ

- **Metal³ログ**: さまざまなPodのログを確認します。

```
kubectll logs -n metal3-system -l app.kubernetes.io/component=baremetal-operator  
kubectll logs -n metal3-system -l app.kubernetes.io/component=ironic
```



注記

metal3-ironic Podには、少なくとも4つの異なるコンテナ(ironic-httpd、「ironic-log-watch」、ironic、ironic-ipa-downloader (init))が同じPod上に含まれています。kubectllログを使用する場合は、-cフラグを指定して、各コンテナのログを確認します。



注記

ironic-log-watchコンテナは、ネットワーク接続により、管理クラスタにコンソールログを返送できる場合は、検査/プロビジョニング後にホストからこのログを公開します。これはプロビジョニングエラーが発生したが、BMCコンソールログに直接アクセスできない場合に役立つ可能性があります。

- **Rancher Turtlesログ**: さまざまなPodのログを確認します。

```
kubectll logs -n rancher-turtles-system -l control-plane=controller-manager  
kubectll logs -n rancher-turtles-system -l app.kubernetes.io/name=cluster-api-operator  
kubectll logs -n rke2-bootstrap-system -l cluster.x-k8s.io/provider=bootstrap-rke2  
kubectll logs -n rke2-control-plane-system -l cluster.x-k8s.io/provider=control-plane-rke2  
kubectll logs -n capi-system -l cluster.x-k8s.io/provider=cluster-api  
kubectll logs -n capm3-system -l cluster.x-k8s.io/provider=infrastructure-metal3
```

- **BMCログ**: 通常BMCには、ほとんどの操作を実行できるUIがあります。また、通常は潜在的な問題(イメージにアクセスできない、ハードウェアエラーなど)がないか確認できる「ログ」セクションもあります。
- **コンソールログ**: BMCコンソールに接続して(BMC Web UI、シリアルなどを經由)、書き込まれているログにエラーがないか確認します。

1. BareMetalHostステータスの確認:

- `kubectl get bmh -A`を実行すると、現在の状態が表示されます。 `provisioning`、`ready`、`error`、`registering`を探してください。
- `kubectl describe bmh -n <namespace> <bmh_name>`を実行すると、BMHが停止する可能性がある理由を説明する、詳細なイベントと状態が表示されます。

2. RedFishの接続性のテスト:

- Metal³コントロールプレーンから`curl`を使用して、RedFishを介してBMCへの接続性をテストします。
- `BareMetalHost-Secret`定義に正しいBMC資格情報が提供されていることを確認します。

3. turtles/CAPI/metal3 Podのステータスの確認: 管理クラスタ上のコンテナが稼働中であることを確認します: `kubectl get pods -n metal3-system`および`kubectl get pods -n rancher-turtles-system (capi-system、capm3-system、rke2-bootstrap-system、およびrke2-control-plane-systemも参照)`。

4. Ironicエンドポイントにプロビジョニング中のホストからアクセスできることを確認する: プロビジョニング中のホストは、レポートをMetal³に返すため、Ironicエンドポイントにアクセスする必要があります。 `kubectl get svc -n metal3-system metal3-metal3-ironic`を使用してIPを確認し、`curl/nc`経由でアクセスを試みてください。

5. IPAイメージがBMCからアクセスできることを確認する: IPAはIronicエンドポイントによって提供されており、仮想CDとして使用されているため、BMCからアクセスする必要があります。

6. OSイメージがプロビジョニング中のホストからアクセスできることを確認する: ホストのプロビジョニングに使用されるイメージは、一時的にダウンロードされ、ディスクに書き込まれるため、ホスト自体からアクセスする必要があります(IPA実行時)。

7. Metal³コンポーネントログの調査: 上記を参照してください。

8. **BMH検査の再トリガ**: 検査に失敗するか、使用可能なホストのハードウェアが変更された場合、BMHオブジェクトに`inspect.metal3.io: ""`というアノテーションを付けることで、新しい検査プロセスをトリガできます。詳細については、Metal³の「[Controlling inspection \(検査の制御\)](https://book.metal3.io/bmo/inspect_annotation)」(https://book.metal3.io/bmo/inspect_annotation)  ガイドを参照してください。

9. **ベアメタルIPAコンソール**: IPAの問題をトラブルシューティングするには、いくつかの代替手段が存在します。

- 「自動ログイン」を有効にします。これにより、IPAコンソールへの接続時に、ルートユーザが自動的にログインできるようになります。



警告

これはホストへの完全なアクセス権を付与するため、デバッグ目的のみに使用されます。

自動ログインを有効にするには、Metal3 helmの`global.ironicKernelParams`値が`console=ttyS0 suse.autologin=ttyS0`ようになる必要があります(コンソールによっては、`ttyS0`を変更できます)。次にMetal³チャートの再デプロイメントを実行する必要があります(`ttyS0`は一例であり、実際の端末と一致する必要があることに注意してください。たとえば、ベアメタルでは多くの場合`tty1`となります。これは、起動時にIPA RAMディスクのコンソール出力で`/etc/issue`にコンソール名が表示されていることを見ることで確認できます)。

別の方法は、`metal3-system`名前空間の`ironic-bmo configmap`の`IRONIC_KERNEL_PARAMS`パラメータを変更することです。この方法は、`kubectl edit`で実行できるため簡単ですが、チャートを更新すると上書きされます。その後、Metal³ Podを`kubectl delete pod -n metal3-system -l app.kubernetes.io/component=ironic`で再起動する必要があります。

- IPAにルートユーザのsshキーを挿入します。



警告

これはホストへの完全なアクセス権を付与するため、デバッグ目的のみに使用されます。

ルートのsshキーを挿入するには、Metal³ helm `debug.ironicRamdiskSshKey`値を使用する必要があります。その後、Metal³ チャートの再デプロイメントを実行する必要があります。

別の方法は、`metal3-system`名前空間の`ironic-bmo` `configmap`の`IRONIC_RAMDISK_SSH_KEY`パラメータを変更することです。この方法は、`kubectl edit`で実行できるため簡単ですが、チャートを更新すると上書きされます。その後、Metal³ Podを`kubectl delete pod -n metal3-system -l app.kubernetes.io/component=ironic`で再起動する必要があります。



注記

CAPIトラブルシューティング (<https://cluster-api.sigs.k8s.io/user/troubleshooting>)  ガイドとMetal³トラブルシューティング (<https://book.metal3.io/troubleshooting>)  ガイドを確認してください。

50 その他のコンポーネントのトラブルシューティング

その他のSUSE Edgeコンポーネントのトラブルシューティングガイドについては、それぞれの公式ドキュメントを参照してください。

- SUSE Linux Microのトラブルシューティング (<https://documentation.suse.com/smart/micro-clouds/html/SLE-Micro-5.5-admin/index.html#id-1.10>) 
- RKE2の既知の問題 (https://docs.rke2.io/known_issues) 
- K3sの既知の問題 (<https://docs.k3s.io/known-issues>) 
- Rancherの一般的なトラブルシューティング (<https://ranchermanager.docs.rancher.com/troubleshooting/general-troubleshooting>) 
- SUSE Multi-Linux Managerのトラブルシューティング (<https://documentation.suse.com/multi-linux-manager/5.1/en/docs/administration/troubleshooting/tshoot-intro.html>) 
- Elementalサポート (<https://elemental.docs.rancher.com/troubleshooting-support/>) 
- Rancher Turtlesのトラブルシューティング (<https://turtles.docs.rancher.com/turtles/stable/en/troubleshooting/troubleshooting.html>) 
- Longhornのトラブルシューティング (<https://longhorn.io/docs/1.9.1/troubleshoot/troubleshooting/>) 
- Neuvectorのトラブルシューティング (<https://open-docs.neuvector.com/next/troubleshooting/troubleshooting/>) 
- Fleetのトラブルシューティング (<https://fleet.rancher.io/troubleshooting>) 

また、SUSEナレッジベース (<https://www.suse.com/support/kb/>) も参照してください。

51 サポートのための診断情報の収集

SUSEサポートに連絡する場合は、包括的な診断情報を提供することが重要です。

収集すべき重要な情報

- **問題の詳細な説明:** 何が起こったか、いつ起こったか、その際に何をしていたのか、どのような動作が期待されるか、実際の動作はどのようなものか?
- **再現する手順:** 問題を忠実に再現できますか? 再現できる場合は、正確な手順をリストしてください。
- **コンポーネントのバージョン:** SUSE Edgeバージョン、コンポーネントのバージョン(RKE2/K3、EIB、Metal³、Elemental、..)。
- **関連するログ:**
 - journalctlの出力(可能な場合はサービス別にフィルタリング、または完全な起動ログ)。
 - Kubernetes Podのログ(kubectrlログ)。
 - Metal³/Elementalコンポーネントのログ。
 - EIB構築ログおよびその他のログ。
- **システム情報:**
 - uname -a
 - df -h
 - ip a
 - /etc/os-release
- **設定ファイル:** Elemental、Metal³、EIBに関連する設定ファイル(Helmチャート値、configmapなど)。
- **Kubernetes情報:** ノード、サービス、デプロイメントなど。
- **影響を受けるKubernetesオブジェクト:** BMH、MachineRegistrationなど。

収集方法

- **ログの場合:** コマンド出力をファイルにリダイレクトします(例: journalctl -u k3s > k3s_logs.txt)。

- **Kubernetesリソースの場合:** `kubectl get <resource> -o yaml > <resource_name>.yaml`を使用して、詳細なYAML定義を取得します。
- **システム情報の場合:** 上記のコマンドの出力を収集します。
- **SL Microの場合:** `supportconfig`を使用してサポートのためのシステム情報を収集する方法については、[SUSE Linux Microトラブルシューティングガイド \(https://documentation.suse.com/sle-micro/5.5/html/SLE-Micro-all/cha-adm-support-slemicro.html\)](https://documentation.suse.com/sle-micro/5.5/html/SLE-Micro-all/cha-adm-support-slemicro.html) のドキュメントを確認してください。
- **RKE2/Rancher の場合:** Rancher v2.x Linuxログコレクタスクリプトを実行するには、[Rancher v2.x Linuxログコレクタスクリプト \(https://www.suse.com/support/kb/doc/?id=000020191\)](https://www.suse.com/support/kb/doc/?id=000020191) の記事を確認してください。

サポートに問い合わせる。 SUSEサポートへの問い合わせ方法の詳細については、「[How-to effectively work with SUSE Technical Support \(SUSEテクニカルサポートを効果的に活用する方法\) \(https://www.suse.com/support/kb/doc/?id=000019452\)](https://www.suse.com/support/kb/doc/?id=000019452)」にある記事と、「[SUSEテクニカルサポートのハンドブック \(https://www.suse.com/support/handbook/\)](https://www.suse.com/support/handbook/)」にあるサポートハンドブックを参照してください。

IX 付録

52 リリースノート **551**

52 リリースノート

52.1 要約

SUSE Edge 3.4は、インフラストラクチャとクラウドネイティブなアプリケーションをエッジにデプロイするという独自の課題に対処することに特化した、緊密に統合されて包括的に検証されたエンドツーエンドのソリューションです。重点を置いているのは、独創的でありながら高い柔軟性とスケーラビリティを持つセキュアなプラットフォームを提供し、初期デプロイメントイメージの構築からノードのプロビジョニングとオンボーディング、アプリケーションのデプロイメント、可観測性、ライフサイクル管理にまで対応することです。

このソリューションは、顧客の要件や期待はさまざまであるため「万能」なエッジプラットフォームは存在しないという考え方に基づいて設計されています。エッジデプロイメントにより、実に困難な問題を解決し、継続的に進化させることが要求されます。たとえば、大規模なスケーラビリティ、ネットワークの可用性の制限、物理的なスペースの制約、新たなセキュリティの脅威と攻撃ベクトル、ハードウェアアーキテクチャとシステムリソースのバリエーション、レガシインフラストラクチャやレガシアプリケーションのデプロイとインタフェースの要件、耐用年数を延長している顧客ソリューションといった課題があります。

SUSE Edgeは、最良のオープンソースソフトウェアに基づいてゼロから構築されており、SUSEが持つ、30年にわたってセキュアで安定した定評あるSUSE Linuxプラットフォームを提供してきた歴史と、Rancherポートフォリオによって拡張性に優れ機能豊富なKubernetes管理を提供してきた経験の両方に合致するものです。SUSE Edgeは、これらの機能の上に構築されており、小売、医療、輸送、物流、通信、スマート製造、産業用IoTなど、さまざまな市場セグメントに対応できる機能を提供します。

SUSE Edgeの製品サポートライフサイクルの更新に関する詳細については、[製品サポートライフサイクル \(https://www.suse.com/lifecycle/#suse-edge-33\)](https://www.suse.com/lifecycle/#suse-edge-33) を参照してください。



注記

SUSE Telco Cloud (旧称SUSE Edge for Telco)はSUSE Edgeの派生製品にあたり、このプラットフォームを通信事業者の要件に対処可能にするための最適化とコンポーネントが追加されています。明記されていない限り、すべてのリリースノートはSUSE Edge 3.4とSUSE Telco Cloud 3.4の両方に適用されます。

52.2 概要

これらのリリースノートは、明示的に指定および説明されていない限り、すべてのアーキテクチャで同一です。また、最新バージョンは、その他すべてのSUSE製品のリリースノートとともに、常に<https://www.suse.com/releasesnotes> でオンラインで参照できます。

エントリが記載されるのは1回だけですが、そのエントリが重要で複数のセクションに属している場合は複数の場所で参照できます。リリースノートには通常、連続する2つのリリース間の変更のみが記載されます。特定の重要なエントリは、以前の製品バージョンのリリースノートから繰り返し記載される場合があります。このようなエントリを特定しやすくするために、該当するエントリにはその旨を示すメモが含まれています。

ただし、繰り返し記載されているエントリは厚意としてのみ提供されています。したがって、リリースを1つ以上スキップする場合は、スキップしたリリースのリリースノートも確認してください。現行リリースのリリースノートしか確認しないと、システムの動作に影響する可能性がある重要な変更を見逃す可能性があります。SUSE Edgeのバージョンはx.y.zで定義され、「x」はメジャーバージョン、「y」はマイナーバージョン、「z」は「z ストリーム」とも呼ばれるパッチバージョンを表します。SUSE Edgeの製品ライフサイクルは、「3.4」のようなマイナーリリースを中心に定義されますが、「3.4.1」のように、ライフサイクルを通じて後続のパッチアップデートが適用されます。



注記

SUSE Edge zストリームリリースは、バージョン管理されたスタックとして緊密に統合されていて、綿密にテストされています。個々のコンポーネントを上記のバージョンとは異なるバージョンにアップグレードすると、システムのダウンタイムが発生する可能性が高くなります。テストされていない設定でEdgeクラスタを実行することは可能ですが、推奨されません。また、サポートチャンネルを通じて解決策を提供するのに時間がかかる場合があります。

52.3 リリース3.4.0

公開日: 2025年9月24日

完全サポート終了日: 2026年3月20日

保守サポート終了日: 2027年9月20日

EOL: 2027年9月21日

概要: SUSE Edge 3.4.0はSUSE Edge 3.4リリースストリームの最初のリリースです。

52.3.1 新機能

- Kubernetes 1.33およびRancher Prime 2.12に更新
- Rancher Turtles、Cluster API、Metal3/Ironicのバージョンを更新
- SUSE Storage (Longhorn) 1.9.1に更新(リリースノート (<https://longhorn.io/docs/1.9.1/>) )
- AArch64ダウンストリームクラスタのより柔軟なデプロイメントが、ダイレクトネットワークプロビジョニングフローを通じて可能になりました。詳細については、[第42章「完全に自動化されたダイレクトネットワークプロビジョニング」](#)を参照してください。
- デュアルスタッククラスタのデプロイメントが完全にサポートされるようになりました(シングルスタックIPv6は[52.5項「技術プレビュー」](#)のままです)。
- MetalLBのBGPモードが技術プレビューとして利用可能になりました。詳細については、[52.5項「技術プレビュー」](#)および[第25章「K3s上のMetalLB \(レイヤ3モードを使用\)」](#)を参照してください。
- Edge Image Builderが1.3.0に更新されました。アップストリームリリースノート (https://github.com/suse-edge/edge-image-builder/blob/release-1.3/RELEASE_NOTES.md) を参照してください。

52.3.2 バグおよびセキュリティの修正

- Rancher Prime 2.12にはいくつかのバグ修正が含まれています(アップストリーム Rancherリリースノート (<https://github.com/rancher/rancher/releases/tag/v2.12.1>) )
- Rancher Prime 2.12には、拡張機能のアップグレードの利用可能性を判断する際の AppVersionに関連する問題の修正が含まれています。この問題はEdgeチャートでの動作に影響を与えていました(アップストリームの問題 (<https://github.com/rancher/dashboard/issues/14204>) )
- SUSE Storage (Longhorn) 1.9.1には複数のバグ修正が含まれています(アップストリームのLonghornバグ修正 (<https://github.com/longhorn/longhorn/releases/tag/v1.9.1>) )
- 更新されたMetal³チャートは、検査中にボンディングされたインタフェースに対して間違ったMACが収集される可能性のある問題を修正しました(アップストリームのIPAの問題 (<https://bugs.launchpad.net/ironic-python-agent/+bug/2103450>) )

- 更新されたMetal³チャートは、ConfigMapの更新時にデプロイメントが正しく再起動されない可能性のある問題を修正しました([アップストリームの問題 \(https://github.com/suse-edge/charts/issues/219\)](https://github.com/suse-edge/charts/issues/219) )
- Rancher Turtlesの更新には、MachineTemplate ownerReferencesがRKE2 CAPIプロバイダによって適用されなかった問題を解決する修正が含まれています([アップストリームの問題 \(https://github.com/rancher/cluster-api-provider-rke2/issues/500\)](https://github.com/rancher/cluster-api-provider-rke2/issues/500) )

52.3.3 既知の問題



警告

新しいクラスタをデプロイする場合は、[第28章「Kiwiを使用したSUSE Linux Microの更新イメージの構築」](#)に従って、まず新規イメージを構築してください。これはAMD64/Intel 64とAArch64アーキテクチャ両方のクラスタ、および管理クラスタとダウンストリームクラスタを作成するために必要な最初の手順です。

- Edge Image Builder経由でデプロイする場合、`HelmChartConfigs`マニフェストを`kubernetes/manifests`設定ディレクトリに配置すると失敗する可能性があります。代わりに、EIB os-filesインタフェースを使用して、`HelmChartConfigs`を`/var/lib/rancher/{rke2/k3s}/server/manifests/`に配置することをお勧めします。例については、[40.3.1項「ディレクトリ構造」](#)を参照してください。これを実行しないと、[#8357 RKE2の問題 \(https://github.com/rancher/rke2/issues/8357\)](#) で説明されているように、ノードが初期起動時に`NotReady`状態のままになる可能性があります。
- RKE2/K3s 1.31、1.32、1.33バージョンでは、CNI設定の保存に使用されるディレクトリ`/etc/cni`で、`overlayfs`に関連する特定の条件により、そこで書き込まれるファイルの通知を`containerd`にトリガされない場合があります([#8356 RKE2の問題 \(https://github.com/rancher/rke2/issues/8356\)](#) )。この結果、RKE2/K3sのデプロイメントがCNIが起動するのを待機した状態で停止し、RKE2/K3sノードが`NotReady`状態のままになります。これは、`kubectl describe node <affected_node>`を使用してノードレベルで確認できます。

```
Conditions:
  Type      Status  LastHeartbeatTime           LastTransitionTime          Reason
  ----      -
  Ready     False   Thu, 05 Jun 2025 17:41:28 +0000  Thu, 05 Jun 2025 14:38:16
+0000      KubeletNotReady  container runtime network not ready: NetworkReady=false
```

```
reason:NetworkPluginNotReady message:Network plugin returns error: cni plugin not initialized
```

回避策として、RKE2が起動する前に、tmpfsボリュームを`/etc/cni`ディレクトリにマウントできます。これにより、overlayfsの使用が回避され、containerdの通知が見つからない問題や、ノードが再起動されてPodのinitcontainerが再実行されるたびに設定が書き換えられる問題を回避できます。EIBを使用する場合、これは`custom/scripts`ディレクトリ内の`04-tmpfs-cni.sh`スクリプトになり(こちら[\[https://github.com/suse-edge/edge-image-builder/blob/release-1.2/docs/building-images.md#custom\]](https://github.com/suse-edge/edge-image-builder/blob/release-1.2/docs/building-images.md#custom)で説明)、次のようになります。

```
#!/bin/bash
mkdir -p /etc/cni
mount -t tmpfs -o mode=0700,size=5M tmpfs /etc/cni
echo "tmpfs /etc/cni tmpfs defaults,size=5M,mode=0700 0 0" >> /etc/fstab
```

- Elementalを使用してリモートホストをオンボーディングする際に、`dbus.service`と`elemental-system-agent.service`の間に競合状態が発生し、リモートホスト上の`rancher-system-agent.service`が以下のようなエラーで起動に失敗する可能性があります(詳細については、[#784 Edge Image Builderの問題](https://github.com/suse-edge/edge-image-builder/issues/784) (<https://github.com/suse-edge/edge-image-builder/issues/784>) を参照してください)。

```
Sep 19 19:38:07 elementalvm elemental-system-agent[3671]: time="2025-09-19T19:38:07Z"
level=info
msg="[6b20fe64c854da2639804884b34129bb8f718eb59578111da58d9de1509c24db_1:stderr]: Failed
to restart rancher-system-agent.service: Message recipient disconnected from message bus
without replying"
```

回避策として、systemdのオーバーライドファイルを以下のように作成できます。

```
[Unit]
Wants=dbus.service network-online.target
After=dbus.service network-online.target time-sync.target

[Service]
ExecStartPre=/bin/bash -c 'echo "Waiting for dbus to become active..." | systemd-cat -
p info -t elemental-system-agent; sleep 15; timeout 300 bash -c "while ! systemctl is-
active --quiet dbus.service; do sleep 15; done"'
```

また、`30a-copy-elemental-system-agent-override.sh`というカスタムスクリプトを使用して、EIBの`31-elemental-register.sh` (<https://github.com/suse-edge/edge-image-builder/blob/main/pkg/combustion/templates/31-elemental-register.sh.tpl>) スクリプトが燃焼フェーズ中に実行される前に、オーバーライドを`/etc/systemd/system/elemental-system-agent.service.d`に配置できます。

```
#!/bin/bash
```

```
/bin/mkdir -p /etc/systemd/system/elemental-system-agent.service.d
/bin/cp -f elemental-system-agent-override.conf /etc/systemd/system/elemental-system-agent.service.d/override.conf
```

52.3.4 コンポーネントバージョン

以下の表は、3.4.0リリースを構成する個々のコンポーネントを示します。これには、バージョン、Helmチャートバージョン(該当する場合)、およびリリースされたアーティファクトをバイナリ形式でプルできる場所が含まれています。使用法とデプロイメントの例について、関連するドキュメントを参照してください。

名前	バージョン	Helmチャートバージョン	アーティファクトの場所(URL/イメージ)
SUSE Linux Micro	6.1 (最新)	該当なし	SUSE Linux Micro ダウンロード ページ (https://www.suse.com/download/sle-micro/)  SL-Micro.x86_64-6.1-Base-SelfInstall-GM.install.iso (sha256 70b9be28f2d92bc3b228412e4fc2b) SL-Micro.x86_64-6.1-Base-RT-SelfInstall-GM.install.iso (sha256 9ce83e4545d4b36c7c6a44f7841dc) SL-Micro.x86_64-6.1-Base-GM.raw.xz (sha256 36e3efa55822113840dd76fdf6914c) SL-Micro.x86_64-6.1-Base-RT-GM.raw.xz (sha256 2ee66735da3e1da107b4878e73ae)

SUSE Multi-Linux Manager	5.0.5	該当なし	SUSE Multi-Linux Managerダウンロードページ (https://www.suse.com/download/suse-manager/) 
K3s	1.33.3	該当なし	アップストリームK3sリリース (https://github.com/k3s-io/k3s/releases/tag/v1.33.3%2Bk3s1) 
RKE2	1.33.3	該当なし	アップストリームRKE2リリース (https://github.com/rancher/rke2/releases/tag/v1.33.3%2Brke2r1) 
SUSE Rancher Prime	2.12.1	2.12.1	Rancher Prime Helm リポジトリ (https://charts.rancher.com/server-charts/prime/index.yaml)  Rancher 2.12.1 コンテナイメージ (https://github.com/rancher/rancher/releases/download/v2.12.1/rancher-images.txt) 
SUSE Storage (Longhorn)	1.9.1	107.0.0+up1.9.1	Rancher Charts Helm リポジトリ (https://charts.rancher.io/index.yaml) 

			registry.suse.com/ rancher/mirrored- longhornio-csi- attacher:v4.9.0-20250709
			registry.suse.com/ rancher/mirrored- longhornio-csi- provisioner:v5.3.0-20250709
			registry.suse.com/ rancher/mirrored- longhornio-csi- resizer:v1.14.0-20250709
			registry.suse.com/ rancher/mirrored- longhornio-csi- snapshotter:v8.3.0-20250709
			registry.suse.com/ rancher/mirrored- longhornio-csi- node-driver- registrar:v2.14.0-20250709
			registry.suse.com/ rancher/mirrored- longhornio- livenessprobe:v2.16.0-20250709
			registry.suse.com/ rancher/mirrored- longhornio- backing-image- manager:v1.9.1
			registry.suse.com/ rancher/mirrored- longhornio- longhorn- engine:v1.9.1

			registry.suse.com/ rancher/mirrored- longhornio- longhorn-instance- manager:v1.9.1 registry.suse.com/ rancher/mirrored- longhornio- longhorn- manager:v1.9.1 registry.suse.com/ rancher/mirrored- longhornio- longhorn-share- manager:v1.9.1 registry.suse.com/ rancher/mirrored- longhornio- longhorn-ui:v1.9.1 registry.suse.com/ rancher/mirrored- longhornio-support- bundle-kit:v0.0.61 registry.suse.com/ rancher/mirrored- longhornio- longhorn-cli:v1.9.1
SUSE Security	5.4.5	107.0.0+up2.8.7	Rancher Charts Helm リポジ トリ (https:// charts.rancher.io/ index.yaml)  registry.suse.com/ rancher/neuvector- controller:5.4.5 registry.suse.com/ rancher/neuvector- enforcer:5.4.5

			registry.suse.com/ rancher/neuvector- manager:5.4.5 registry.suse.com/ rancher/neuvector- compliance- config:1.0.6 registry.suse.com/ rancher/ neuvector-registry- adapter:0.1.8 registry.suse.com/ rancher/neuvector- scanner:6 registry.suse.com/ rancher/neuvector- updater:0.0.4
Rancher Turtles (CAPI)	0.24.0	304.0.6+up0.24.0	registry.suse.com/ edge/charts/ rancher- turtles:304.0.6_up0.24.0 registry.rancher.com/ rancher/rancher/ turtles:v0.24.0 registry.rancher.com/ rancher/cluster- api-metal3- controller:v1.10.2 registry.rancher.com/ rancher/cluster- api-metal3-ipam- controller:v1.10.2 registry.suse.com/ rancher/cluster-api- controller:v1.10.5

			registry.suse.com/ rancher/cluster- api-provider-rke2- bootstrap:v0.20.1 registry.suse.com/ rancher/cluster- api-provider-rke2- controlplane:v0.20.1
Rancher Turtlesエア ギャップリソース	0.24.0	304.0.6+up0.24.0	registry.suse.com/ edge/charts/ rancher- turtles-airgap- resources:304.0.6_up0.24.0
Metal ³	0.11.5	304.0.16+up0.12.6	registry.suse.com/ edge/charts/ metal3:304.0.16_up0.12.6 registry.suse.com/ edge/3.4/baremetal- operator:0.10.2.1 registry.suse.com/ edge/3.4/ ironic:29.0.4.3 registry.suse.com/ edge/3.4/ironic-ipa- downloader:3.0.9 registry.suse.com/ edge/ mariadb:10.6.15.1
MetalLB	0.14.9	304.0.0+up0.14.9	registry.suse.com/ edge/charts/ metallb:304.0.0_up0.14.9 registry.suse.com/ edge/3.4/metallb- controller:v0.14.8

			registry.suse.com/ edge/3.4/metallb- speaker:v0.14.8 registry.suse.com/ edge/3.4/frr:8.4 registry.suse.com/ edge/3.4/frr- k8s:v0.0.14
Elemental	1.7.3	1.7.3	registry.suse.com/ rancher/elemental- operator-chart:1.7.3 registry.suse.com/ rancher/elemental- operator-crds- chart:1.7.3 registry.suse.com/ rancher/elemental- operator:1.7.3
Elementalダッシュ ボード拡張機能	3.0.1	3.0.1	Elemental拡張機 能Helmチャート (https://github.com/ rancher/ui- plugin-charts/ tree/4.0.0/charts/ elemental/3.0.1) ↗
Edge Image Builder	1.3.0	該当なし	registry.suse.com/ edge/3.4/edge- image-builder:1.3.0
NM Configurator	0.3.3	該当なし	NMConfigurator アップストリーム リリース (https:// github.com/ suse-edge/nm-

			configurator/releases/tag/v0.3.3) 
KubeVirt	1.5.2	304.0.1+up0.6.0	registry.suse.com/ edge/charts/ kubevirt:304.0.1_up0.6.0 registry.suse.com/ suse/sles/15.7/virt- operator:1.5.2 registry.suse.com/ suse/sles/15.7/virt- api:1.5.2 registry.suse.com/ suse/sles/15.7/virt- controller:1.5.2 registry.suse.com/ suse/sles/15.7/virt- exportproxy:1.5.2 registry.suse.com/ suse/sles/15.7/virt- exportserver:1.5.2 registry.suse.com/ suse/sles/15.7/virt- handler:1.5.2 registry.suse.com/ suse/sles/15.7/virt- launcher:1.5.2
KubeVirtダッシュ ボード拡張機能	1.3.2	304.0.3+up1.3.2	registry.suse.com/ edge/charts/ kubevirt-dashboard- extension:304.0.3_up1.3.2
Containerized Data Importer	1.62.0	304.0.1+up0.6.0	registry.suse.com/ edge/charts/ cdi:304.0.1_up0.6.0

			registry.suse.com/ suse/sles/15.7/cdi- operator:1.62.0 registry.suse.com/ suse/sles/15.7/cdi- controller:1.62.0 registry.suse.com/ suse/sles/15.7/cdi- importer:1.62.0 registry.suse.com/ suse/sles/15.7/cdi- cloner:1.62.0 registry.suse.com/ suse/sles/15.7/cdi- apiserver:1.62.0 registry.suse.com/ suse/sles/15.7/cdi- uploadserver:1.62.0 registry.suse.com/ suse/sles/15.7/cdi- uploadproxy:1.62.0
Endpoint Copier Operator	0.3.0	304.0.1+up0.3.0	registry.suse.com/ edge/charts/ endpoint-copier- operator:304.0.1_up0.3.0 registry.suse.com/ edge/3.4/ endpoint-copier- operator:0.3.0
Akri (廃止)	0.12.20	304.0.0+up0.12.20	registry.suse.com/ edge/charts/ akri:304.0.0_up0.12.20 registry.suse.com/ edge/charts/ akri-dashboard- extension:304.0.0_up1.3.1

			registry.suse.com/ edge/3.4/akri- agent:v0.12.20 registry.suse.com/ edge/3.4/akri- controller:v0.12.20 registry.suse.com/ edge/3.4/akri- debug-echo- discovery- handler:v0.12.20 registry.suse.com/ edge/3.4/akri- onvif-discovery- handler:v0.12.20 registry.suse.com/ edge/3.4/akri- opcua-discovery- handler:v0.12.20 registry.suse.com/ edge/3.4/akri- udev-discovery- handler:v0.12.20 registry.suse.com/ edge/3.4/akri- webhook- configuration:v0.12.20
SR-IOV Network Operator	1.5.0	304.0.2+up1.5.0	registry.suse.com/ edge/charts/ sriov-network- operator:304.0.2_up1.5.0 registry.suse.com/ edge/charts/sriov- crd:304.0.2_up1.5.0

System Upgrade Controller	0.16.0	107.0.0	Rancher Charts Helm リポジトリ (https://charts.rancher.io/index.yaml)  registry.suse.com/ rancher/system- upgrade- controller:v0.16.0
Upgrade Controller	0.1.1	304.0.1+up0.1.1	registry.suse.com/ edge/charts/ upgrade- controller:304.0.1_up0.1.1 registry.suse.com/ edge/3.4/upgrade- controller:0.1.1 registry.suse.com/ edge/3.4/ kubectrl:1.33.4 registry.suse.com/ edge/3.4/release- manifest:3.4.0
Kiwi Builder	10.2.12.0	該当なし	registry.suse.com/ edge/3.4/kiwi- builder:10.2.12.0

52.4 廃止された機能

特に記載のない限り、これらは3.4.0リリースおよびそれ以降のすべてのzストリームバージョンに適用されます。

- Akriは、以前のEdgeリリースでは技術プレビュー製品でしたが、現在は廃止されています。今後のリリースで削除される予定です。

52.5 技術プレビュー

特に記載のない限り、これらは3.4.0リリースおよびそれ以降のすべてのzストリームバージョンに適用されます。

- シングルスタックIPv6デプロイメントは、技術プレビュー製品であり、標準的なサポート範囲の対象外です。
- ダウンストリームデプロイメントにおけるPrecision Time Protocol (PTP)は、技術プレビュー製品であり、標準的なサポート範囲の対象外です。
- MetalLBのBGPモードは、技術プレビュー製品であり、標準的なサポート範囲の対象外です。

52.6 コンポーネントの検証

上記のコンポーネントはSoftware Bill Of Materials (SBOM)のデータを使用して検証できます。たとえば、以下に説明するように`cosign`を使用します。

SUSE署名キーのソース (<https://www.suse.com/support/security/keys/>) からSUSE Edge Containerの公開鍵をダウンロードします。

```
> cat key.pem
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEA7N0S2d8LFKW4WU43bq7Z
IZT537xlKe170QEpYjNrdtqnSwA0/jLtK83m7bTzfYRK4wty/so0g3BGo+x6yDFt
SVXTPBqnYvabU/j7UKaybJtX3jc4SjaezeBqdi96h6yEsLvg4VTZDpy6TFP5ZHxZ
A0fX6m5kU2/RyHGXIt0eUmL5hZ+APYgYG4/455NBaZT2y0ywJ6+1zRgpR0cRAekI
0ZXl51k0ebsGV6ui/NGEC06MB5e3arAhszf8eHDE02FeNJw5cimXkgDh/1Lg3Kp0
dvUNm0EPWvnkNYeMCKR+687QG0bXqSVyCbY6+HG/HLkeBWkv6Hn41oeTSLrjYVGa
T3zxPQVM726sami6pgZ5vULy0leQuKBZr\FhFLbFyXqv1/DokUqEppm2Y3xZQv77
fMNogapp0qYz+nE3wSK4UHPd9z+2bq5WEKQSaLyxadyuq0zxqZgSoCNoX5iIuWte
Zf1RmHjiEndg/2UgxKUysVnyCpiWoGbaLM4dnWE24102050Gj6M4B5fe73hbaRlf
NBqP+97uznnRlS18FizhXzdzJiVPcRav1tDdRUyDE2XkNRXmGfD3aCmILhB27SOA
Lppkouw849PWBt9kDMvzelUYLpINYPHRi2+/eyhHNlufeyJ7e7d6N9VcvjR/6qWG
64iSkcF2DTW61CN5TrCe0k0CAwEAAQ==
-----END PUBLIC KEY-----
```

コンテナイメージのハッシュを検証します。たとえば、`crane`を使用します。

```
> crane digest registry.suse.com/edge/3.4/baremetal-operator:0.10.2.1 --platform linux/
amd64
sha256:310d939f8ae4b547710195b9671a4e9ff417420c0856103dd728b051788b5374
```



注記

マルチアーキテクチャイメージの場合、ダイジェストを取得する際にはプラットフォームを指定する必要があります。例: `--platform linux/amd64`または`--platform linux/arm64`。指定しないと、次の手順でエラーが発生します(`Error: no matching attestations` (エラー: 一致する認証がありません))。

`cosign`を使用して検証します。

```
> cosign verify-attestation --type spdxjson --key key.pem registry.suse.com/edge/3.4/
baremetal-operator@sha256:310d939f8ae4b547710195b9671a4e9ff417420c0856103dd728b051788b5374
> /dev/null
#
Verification for registry.suse.com/edge/3.4/baremetal-
operator@sha256:310d939f8ae4b547710195b9671a4e9ff417420c0856103dd728b051788b5374 --
The following checks were performed on each of these signatures:
- The cosign claims were validated
- Existence of the claims in the transparency log was verified offline
- The signatures were verified against the specified public key
```

SUSE SBOMドキュメント (<https://www.suse.com/support/security/sbom/>) の説明に従ってSBOMデータを抽出します。

```
> cosign verify-attestation --type spdxjson --key key.pem registry.suse.com/edge/3.4/
baremetal-operator@sha256:310d939f8ae4b547710195b9671a4e9ff417420c0856103dd728b051788b5374
| jq '.payload | @base64d | fromjson | .predicate'
```

52.7 アップグレード手順

新しいリリースにアップグレードする方法の詳細については、[パートVI「Day 2操作」](#)を参照してください。

52.8 製品サポートライフサイクル

SUSE Edgeは、SUSEが提供する定評あるサポートに支えられています。SUSEは、エンタープライズ品質のサポートサービスの提供において確固たる実績を誇るテクノロジーリーダーです。詳細については、<https://www.suse.com/lifecycle>、およびサポートポリシーのページ

ジ(<https://www.suse.com/support/policy.html>)を参照してください。サポートケースの作成、SUSEが重大度レベルを分類する方法、またはサポートの範囲について質問がある場合は、テクニカルサポートハンドブック(<https://www.suse.com/support/handbook/>)を参照してください。

SUSE Edge「3.4」は24か月間の運用サポートでサポートされ、最初の6か月間は「完全サポート」、その後の18か月間は「保守サポート」が提供されます。これらのサポートフェーズ後、製品は「サポート終了済み」(EOL)となり、サポートは終了となります。ライフサイクルフェーズの詳細については、以下の表を参照してください。

完全サポート(6か月)	緊急かつ選択された優先度の高いバグ修正は、完全サポート期間中にリリースされます。その他のすべてのパッチ(緊急でないもの、機能強化、新機能)は、通常のリリーススケジュールに従ってリリースされます。
保守サポート(18 か月)	この期間中は、重要な修正のみがパッチとしてリリースされます。その他のバグ修正はSUSEの裁量によりリリースされる可能性があります、必ずしも提供されるとは限りません。
サポート終了済み(EOL)	製品リリースがサポート終了済み日を迎えた場合、顧客は製品ライセンス契約の条件に従い、製品を引き続き使用することができます。SUSEのサポートプランは、EOL日を過ぎた製品リリースには適用されません。

明記されていない限り、記載されているコンポーネントはすべて一般提供(GA)とみなされ、SUSEの標準のサポート範囲の対象となります。一部のコンポーネントは「技術レビュー」として記載されている場合があります。この場合、SUSEは評価のためにGA前の機能への早期アクセスをお客様に提供しますが、これらの機能には標準のサポートポリシーが適用されず、運用ユースケースには推奨されません。SUSEでは、技術レビューのコンポーネントに関するフィードバックや、当該コンポーネントの改良についてのご提案を心からお待ちしております。しかし、機能がお客様のニーズを満たさない場合やSUSEが求める成熟度に達しない場合、一般提供になる前に技術レビューの機能を廃止する権利を留保します。

SUSEは場合により、機能の廃止やAPIの仕様変更を行わなければならないことがあることに注意してください。機能の廃止やAPIの変更の理由としては、機能が新しい実装によって更新または置き換えられた、新しい機能セットが導入された、アップストリームの技術が利用で

きなくなった、アップストリームコミュニティによって互換性のない変更が導入された、などが考えられます。これは特定のマイナーリリース(x.z)内で発生することは意図されていないため、すべてのzストリームリリースではAPIの互換性と機能が維持されます。SUSEは、廃止に関する警告をリリースノート内で十分に余裕をもって提供し、併せて回避策、推奨事項、サービスの中断を最小限に抑える軽減策も提供するように努めます。

SUSE Edgeチームはコミュニティからのフィードバックも歓迎しており、<https://www.github.com/suse-edge> の各コードリポジトリ内で問題を報告できます。

52.9 ソースコードの取得

このSUSE製品には、GNU General Public License (GPL)やその他のさまざまなオープンソースライセンスの下でSUSEにライセンスされた素材が含まれます。SUSEはGPLに従ってGPLでライセンスされた素材に対応するソースコードを提供する必要があるほか、その他すべてのオープンソースライセンスの要件にも準拠します。よって、SUSEはすべてのソースコードを利用可能にしておき、一般的にSUSE Edge GitHubリポジトリ(<https://www.github.com/suse-edge>)にあります。また、依存コンポーネントについてはSUSE Rancher GitHubリポジトリ(<https://www.github.com/rancher>)にあり、特にSUSE Linux Microについては<https://www.suse.com/download/sle-micro> (<https://www.suse.com/download/sle-micro/>)の「Medium 2」でソースコードをダウンロードできます。

52.10 法的通知

SUSEは、この文書の内容や使用に関していかなる表明や保証も行いません。特に、商品性または特定目的への適合性に関する明示的または暗黙的な保証は一切行いません。さらに、SUSEは本書を改訂し、その内容に随時変更を加える権利を留保しますが、いかなる個人または団体に対しても当該の改訂または変更を通知する義務を負いません。

SUSEは、いかなるソフトウェアに関しても、いかなる表明や保証も行いません。特に、商品性または特定目的への適合性に関する明示的または暗黙的な保証は一切行いません。さらに、SUSEはSUSEソフトウェアのあらゆる部分に随時変更を加える権利を留保しますが、いかなる個人または団体に対しても当該の変更を通知する義務を負いません。

本契約の下で提供されるいかなる製品または技術情報も、米国の輸出管理法規および他国の貿易法の対象となる場合があります。お客様はすべての輸出管理規制を遵守し、成果物の輸出、再輸出、または輸入のために必要なライセンスまたは分類を取得することに同意します。お客様は、現行の米国輸出禁止リストに記載されている団体や米国輸出法に規定された禁輸国やテロ支援国への輸出や再輸出を行わないことに同意します。また、成果物を禁止されている核、ミサイル、または化学/生物兵器の最終用途に使用しないことにも同意

します。SUSEソフトウェアの輸出に関する詳細情報については、<https://www.suse.com/company/legal/>  を参照してください。SUSEは、必要な輸出許可の取得を怠ったことに対する責任を一切負いません。

Copyright © 2024 SUSE LLC.

このリリースノート文書は、Creative Commons Attribution-NoDerivatives 4.0 International License (CC-BY-ND-4.0)の下でライセンスされています。お客様は、この文書と併せてライセンスのコピーを受け取っている必要があります。受け取っていない場合は、<https://creativecommons.org/licenses/by-nd/4.0/>  を参照してください。

SUSEは、本書で説明されている製品に組み込まれた技術に関連する知的財産権を有しています。これらの知的財産権には、特に、<https://www.suse.com/company/legal/>  に記載されている1つまたは複数の米国特許、ならびに米国およびその他の国における1つまたは複数のその他の特許または出願中の特許申請が含まれていることがあります。これらに限定されません。

SUSEの商標については、SUSEの商標とサービスマークのリスト(<https://www.suse.com/company/legal/> )を参照してください。第三者のすべての商標は各所有者の財産です。SUSEのブランド情報と使用要件については、<https://brand.suse.com/>  で公開されているガイドラインを参照してください。