SUSE

Deploying and Installing SUSE AI

Publication Date: 2026-01-08

Contents

WHAT?

This document provides a comprehensive, step-by-step guide for the SUSE AI deployment.

WHY?

To help users successfully complete the deployment process.

EFFORT

Less than one hour of reading and an advanced knowledge of Linux deployment.

GOAL

To learn enough information to deploy SUSE AI in both testing and production environments.

SUSE AI is a versatile product consisting of multiple software layers and components. This document outlines the complete workflow for deployment and installation of all SUSE AI dependencies, as well as SUSE AI itself. You can also find references to recommended hardware and software requirements, as well as steps to take after the product installation.

Tip: Hardware and software requirements

For hardware, software and application-specific requirements, refer to SUSE AIrequirements (https://documentation.suse.com/suse-ai/1.0/html/AI-requirements/index.html) ↗.

1 Installation overview

The following chart illustrates the installation process of SUSE AI. It outlines the following possible scenarios:

- You have clean cluster nodes prepared without a supported Linux operating system installed.
- You have a supported Linux operating system and Kubernetes distribution installed on cluster nodes.
- You have SUSE Rancher Prime and all supportive components installed on the Kubernetes cluster and are prepared to install the required applications from the AI Library.

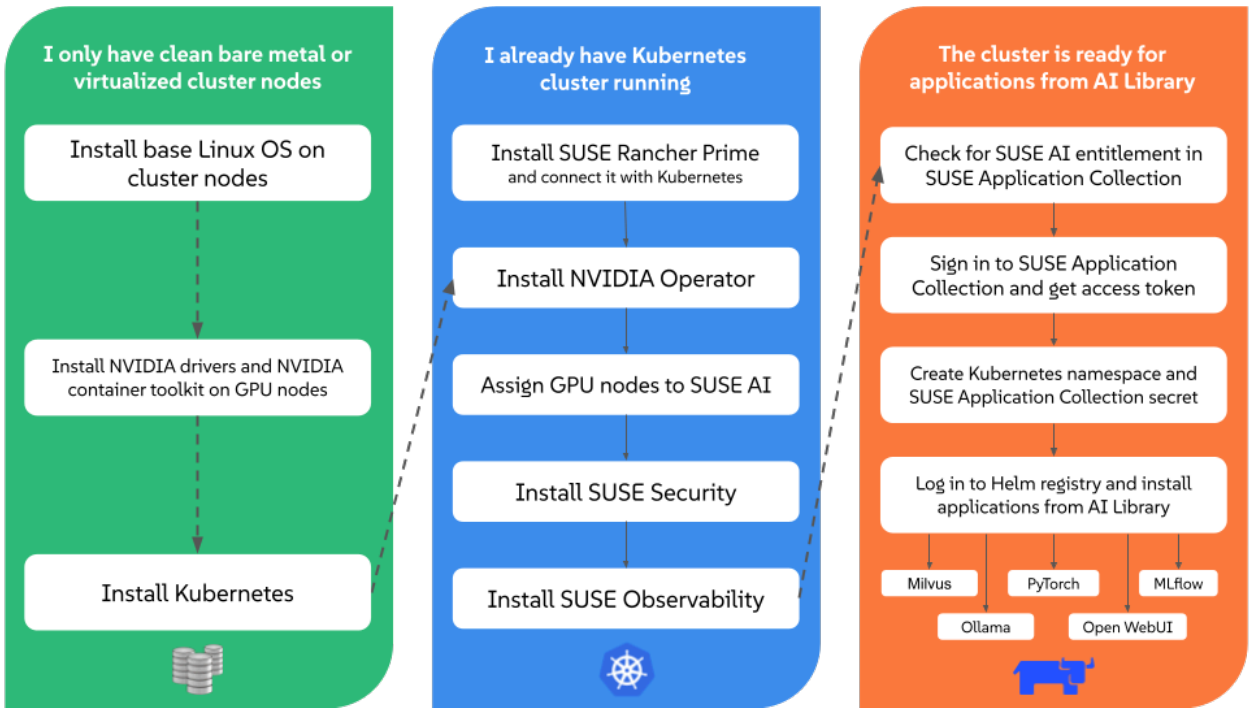

I only have clean bare metal or virtualized cluster nodes
Install base Linux OS on cluster nodes
Install NVIDIA drivers and NVIDIA container toolkit on GPU nodes
Install Kubernetes
I already have Kubernetes cluster running
Install SUSE Rancher Prime and connect it with Kubernetes
Install NVIDIA Operator
Assign GPU nodes to SUSE AI
Install SUSE Security
Install SUSE Observability
The cluster is ready for applications from AI Library
Check for SUSE AI entitlement in SUSE Application Collection
Sign in to SUSE Application Collection and get access token
Create Kubernetes namespace and SUSE Application Collection secret
Log in to Helm registry and install applications from AI Library
Milvus
PyTorch
MLflow
Ollama
Open WebUI

FIGURE 1: SUSE AI INSTALLATION PROCESS

2 Installing the Linux and Kubernetes distribution

This procedure includes the steps to install the base Linux operating system and a Kubernetes distribution for users who start deploying on cluster nodes from scratch. If you already have a Kubernetes cluster installed and running, you can skip this procedure and continue with *Section 4.1, "Installation procedure"*.

1. Install and register a supported Linux operating system on each cluster node. We recommend using one of the following operating systems:

   - SUSE Linux Enterprise Server 15 SP6 for a traditional non-transactional operating system. For more information, see *Section 2.1, "Installing SUSE Linux Enterprise Server"*.

   - SUSE Linux Micro 6.1 for an immutable transactional operating system. For more information, see SUSE Linux Micro 6.1 documentation (https://documentation.suse.com/sle-micro/6.1/) ↗.

For a list of supported operating systems, refer to https://www.suse.com/suse-rancher/support-matrix/all-supported-versions/ ↗.

2. Install the NVIDIA GPU driver on cluster nodes with GPUs. Refer to *Section 2.2, “Installing NVIDIA GPU drivers”* for details.

3. Install Kubernetes on cluster nodes. We recommend using the supported SUSE Rancher Prime: RKE2 distribution. Refer to *Section 2.3, “Installing SUSE Rancher Prime: RKE2”* for details. For a list of supported Kubernetes platforms, refer to https://www.suse.com/suse-rancher/support-matrix/all-supported-versions/ ↗.

2.1 Installing SUSE Linux Enterprise Server

Use the following procedures to install SLES on all supported hardware platforms. They assume you have successfully booted into the installation system. For more detailed installation instructions and deployment strategies, refer to SUSE Linux Enterprise Server Deployment Guide (https://documentation.suse.com/sles/15-SP6/html/SLES-all/book-deployment.html) ↗.

2.1.1 The Unified Installer

Starting with SLES 15, the installation medium consists only of the Unified Installer, a minimal system for installing, updating and registering all SLE base products. During the installation, you can add functionality by selecting modules and extensions to be installed on top of the Unified Installer.

2.1.2 Installing offline or without registration

The default installation medium 15 SP6-Online-ARCH-GM-media1.iso is optimized for size and does not contain any modules and extensions. Therefore, the installation requires network access to register your product and retrieve repository data for the modules and extensions.

For installation without registering the system, use the `15 SP6-Full-ARCH-GM-media1.iso` image from https://www.suse.com/download/sles/ ↗ and refer to Installing without registration (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-install.html#sec-yast-install-scc-registration-none) ↗.

Tip: Copying the installation media image to a removable flash disk

Use the following command to copy the contents of the installation image to a removable flash disk.

```
> sudo dd if=IMAGE of=FLASH_DISK bs=4M && sync
```

`IMAGE` needs to be replaced with the path to the `15 SP6-Online-ARCH-GM-media1.iso` or `15 SP6-Full-ARCH-GM-media1.iso` image file. `FLASH_DISK` needs to be replaced with the flash device. To identify the device, insert it and run:

```
{prompt_root}grep -Ff <(hwinfo --disk --short) <(hwinfo --usb --short)
disk:
  /dev/sdc             General USB Flash Disk
```

Make sure the size of the device is sufficient for the desired image. You can check the size of the device with:

```
{prompt_root}fdisk -l /dev/sdc | grep -e "^/dev"
     /dev/sdc1   *      2048 31490047 31488000  15G 83 Linux
```

In this example, the device has a capacity of 15 GB. The command to use for the `15 SP6-Full-ARCH-GM-media1.iso` would be:

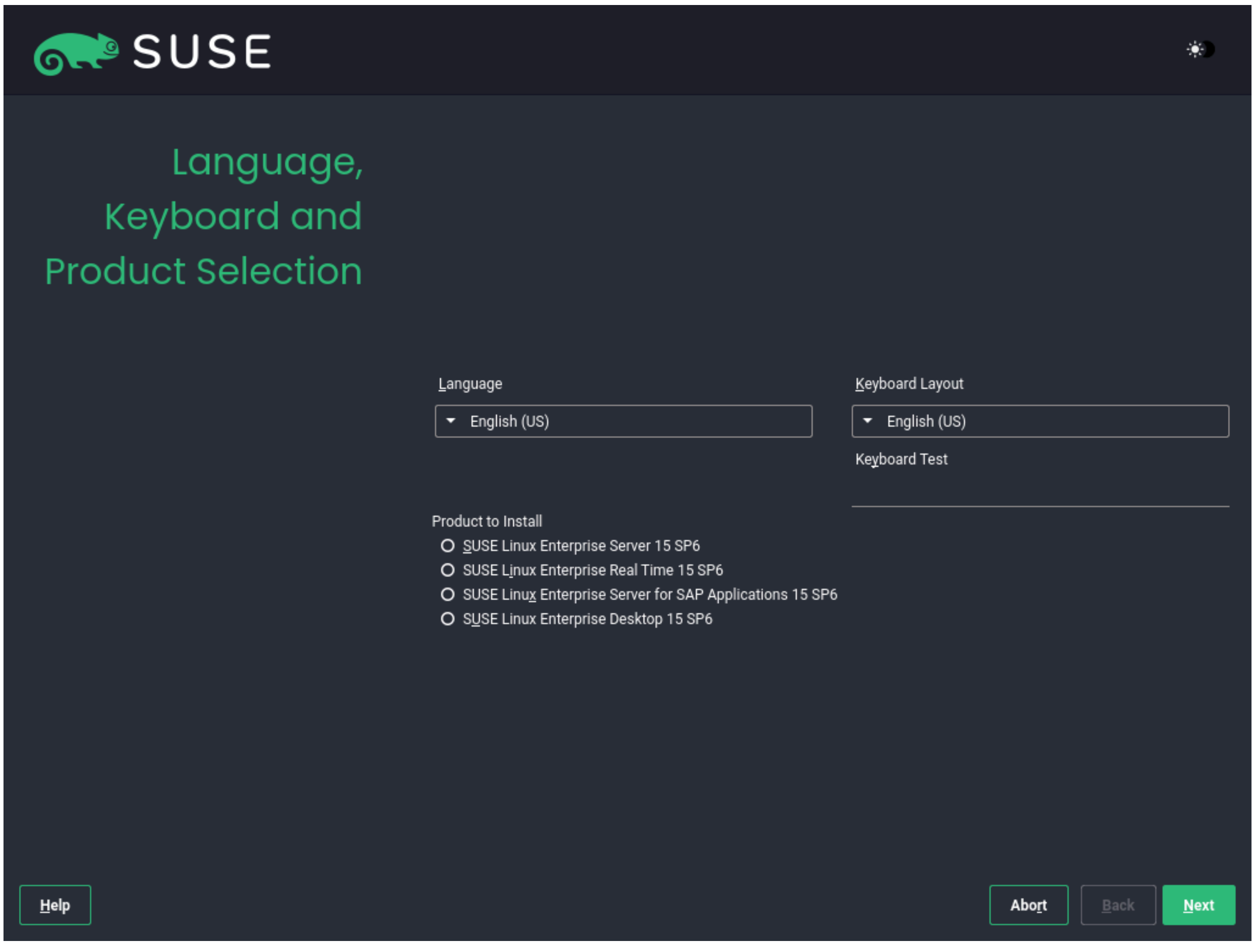
```
dd if=15 SP6-Full-ARCH-GM-media1.iso of=/dev/sdc bs=4M && sync
```

The device must not be mounted when running the `dd` command. Note that all data on the partition will be erased.

2.1.3 The installation procedure

To install SLES, boot or IPL into the installer from the Unified Installer medium and start the installation.

2.1.3.1 Language, keyboard and product selection



Language
English (US)

Keyboard Layout
English (US)

Keyboard Test

Product to Install
- SUSE Linux Enterprise Server 15 SP6
- SUSE Linux Enterprise Real Time 15 SP6
- SUSE Linux Enterprise Server for SAP Applications 15 SP6
- SUSE Linux Enterprise Desktop 15 SP6

FIGURE 2: LANGUAGE, KEYBOARD AND PRODUCT SELECTION

The *Language* and *Keyboard Layout* settings are initialized with the language you chose on the boot screen. If you do not change the default, it remains English (US). Change the settings here, if necessary. Use the *Keyboard Test* text box to test the layout.

Select SUSE Linux Enterprise Server 15 SP6 for installation. You need to have a registration code for the product. Proceed with *Next*.

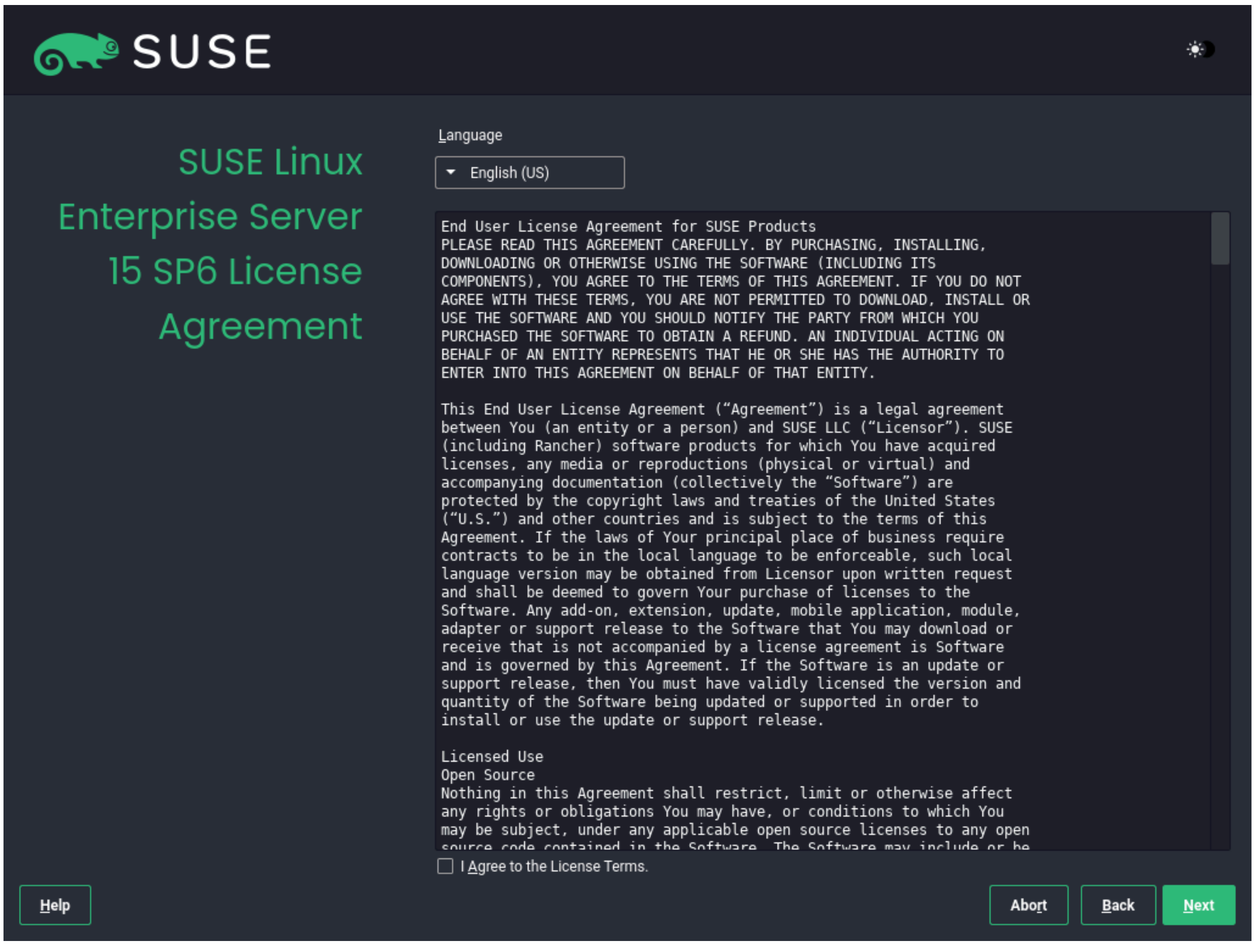
Tip: Light and high-contrast themes

If you have difficulty reading the labels in the installer, you can change the widget colors and theme.

Click the ☀◗ button or press `Shift`–`F3` to open a theme selection dialog. Select a theme from the list and *Close* the dialog.

`Shift` – `F4` switches to the color scheme for vision-impaired users. Press the buttons again to switch back to the default scheme.

2.1.3.2 License agreement

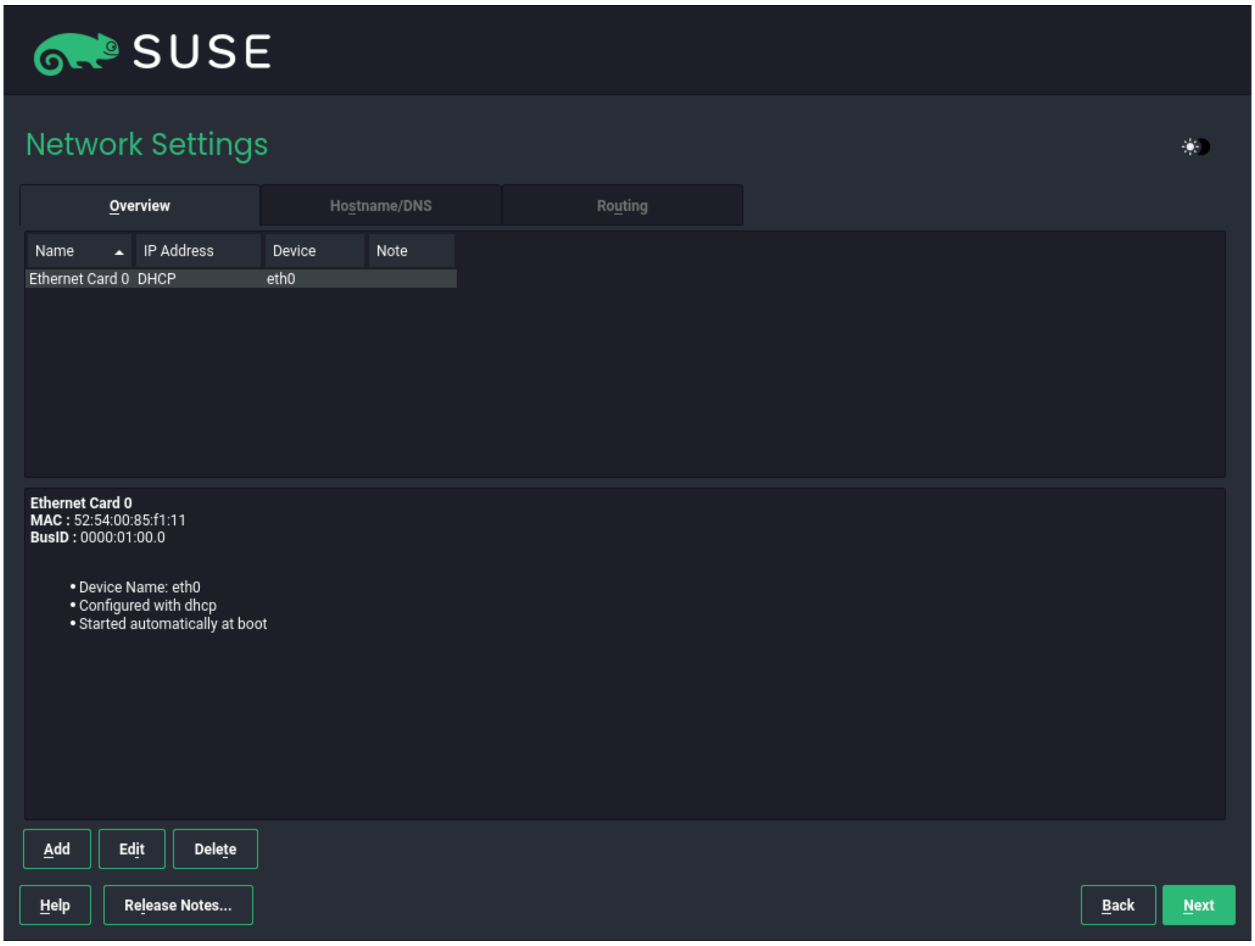

SUSE
SUSE Linux Enterprise Server 15 SP6 License Agreement
Language
English (US)
I Agree to the License Terms.
Help
Abort
Back
Next

```
End User License Agreement for SUSE Products
PLEASE READ THIS AGREEMENT CAREFULLY. BY PURCHASING, INSTALLING,
DOWNLOADING OR OTHERWISE USING THE SOFTWARE (INCLUDING ITS
COMPONENTS), YOU AGREE TO THE TERMS OF THIS AGREEMENT. IF YOU DO NOT
AGREE WITH THESE TERMS, YOU ARE NOT PERMITTED TO DOWNLOAD, INSTALL OR
USE THE SOFTWARE AND YOU SHOULD NOTIFY THE PARTY FROM WHICH YOU
PURCHASED THE SOFTWARE TO OBTAIN A REFUND. AN INDIVIDUAL ACTING ON
BEHALF OF AN ENTITY REPRESENTS THAT HE OR SHE HAS THE AUTHORITY TO
ENTER INTO THIS AGREEMENT ON BEHALF OF THAT ENTITY.

This End User License Agreement (“Agreement”) is a legal agreement
between You (an entity or a person) and SUSE LLC (“Licensor”). SUSE
(including Rancher) software products for which You have acquired
licenses, any media or reproductions (physical or virtual) and
accompanying documentation (collectively the “Software”) are
protected by the copyright laws and treaties of the United States
(“U.S.”) and other countries and is subject to the terms of this
Agreement. If the laws of Your principal place of business require
contracts to be in the local language to be enforceable, such local
language version may be obtained from Licensor upon written request
and shall be deemed to govern Your purchase of licenses to the
Software. Any add-on, extension, update, mobile application, module,
adapter or support release to the Software that You may download or
receive that is not accompanied by a license agreement is Software
and is governed by this Agreement. If the Software is an update or
support release, then You must have validly licensed the version and
quantity of the Software being updated or supported in order to
install or use the update or support release.

Licensed Use
Open Source
Nothing in this Agreement shall restrict, limit or otherwise affect
any rights or obligations You may have, or conditions to which You
may be subject, under any applicable open source licenses to any open
source code contained in the Software. The Software may include or be
```

FIGURE 3: LICENSE AGREEMENT

Read the License Agreement. It is presented in the language you have chosen on the boot screen. Translations are available via the *License Language* drop-down list. You need to accept the agreement by checking *I Agree to the License Terms* to install SLES. Proceed with *Next*.

2.1.3.3 Network settings



Name	IP Address	Device	Note
Ethernet Card 0	DHCP	eth0	

Ethernet Card 0
MAC : 52:54:00:85:f1:11
BusID : 0000:01:00.0

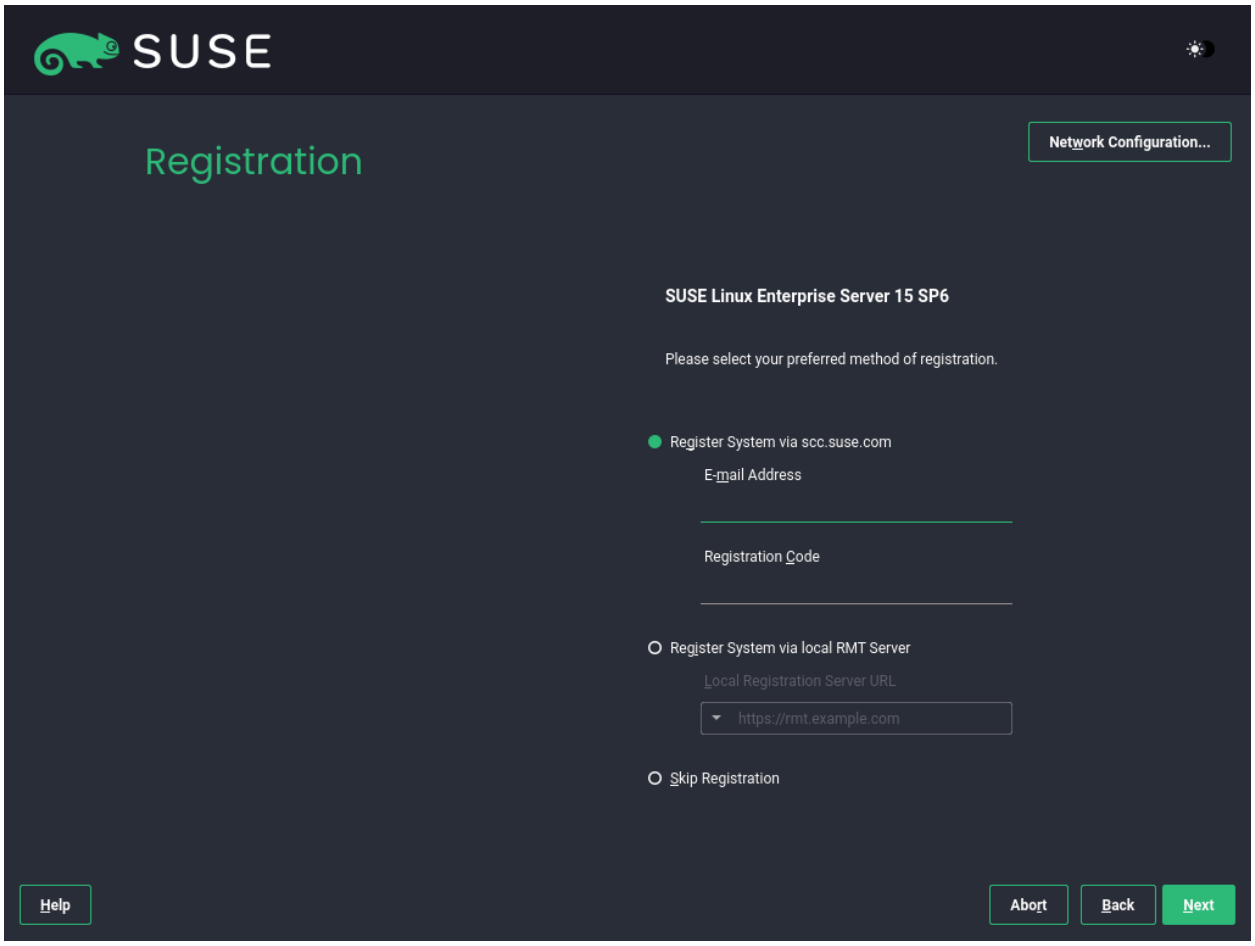
- Device Name: eth0
- Configured with dhcp
- Started automatically at boot

FIGURE 4: NETWORK SETTINGS

A system analysis is performed, where the installer probes for storage devices and tries to find other installed systems. If the network was automatically configured via DHCP during the start of the installation, you are presented the registration step.

If the network is not yet configured, the *Network Settings* dialog opens. Choose a network interface from the list and configure it with *Edit*. Alternatively, *Add* an interface manually. See the sections on installer network settings (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-install.html#sec-yast-install-network) and configuring a network connection with YaST (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-network.html#sec-network-yast) for more information. If you prefer to do an installation without network access, skip this step without making any changes and proceed with *Next*.

2.1.3.4 Registration



SUSE Linux Enterprise Server 15 SP6

Please select your preferred method of registration.

● Register System via scc.suse.com

E-mail Address

Registration Code

○ Register System via local RMT Server

Local Registration Server URL

https://rmt.example.com

○ Skip Registration

FIGURE 5: REGISTRATION

To get technical support and product updates, you need to register and activate SLES with the SUSE Customer Center or a local registration server. Registering your product at this stage also grants you immediate access to the update repository. This enables you to install the system with the latest updates and patches available.

When registering, repositories and dependencies for modules and extensions are loaded from the registration server.

Register system at scc.suse.com

To register at the SUSE Customer Center, enter the *E-mail Address* associated with your SUSE Customer Center account and the *Registration Code* for SLES. Proceed with *Next*.

Register system via local RMT server

If your organization provides a local registration server, you may alternatively register to it. Activate *Register System via local RMT Server* and either choose a URL from the drop-down list or type in an address. Proceed with *Next*.

Skip registration

If you are offline or want to skip registration, activate *Skip Registration*. Accept the warning with *OK* and proceed with *Next*.

Important: Skipping the registration

Your system and extensions need to be registered to retrieve updates and to be eligible for support. Skipping the registration is only possible when installing from the `15 SP6-Full-ARCH-GM-media1.iso` image.

If you do not register during the installation, you can do so at any time later from the running system. To do so, run *YaST* › *Product Registration* or the command-line tool `SUSE-Connect`.

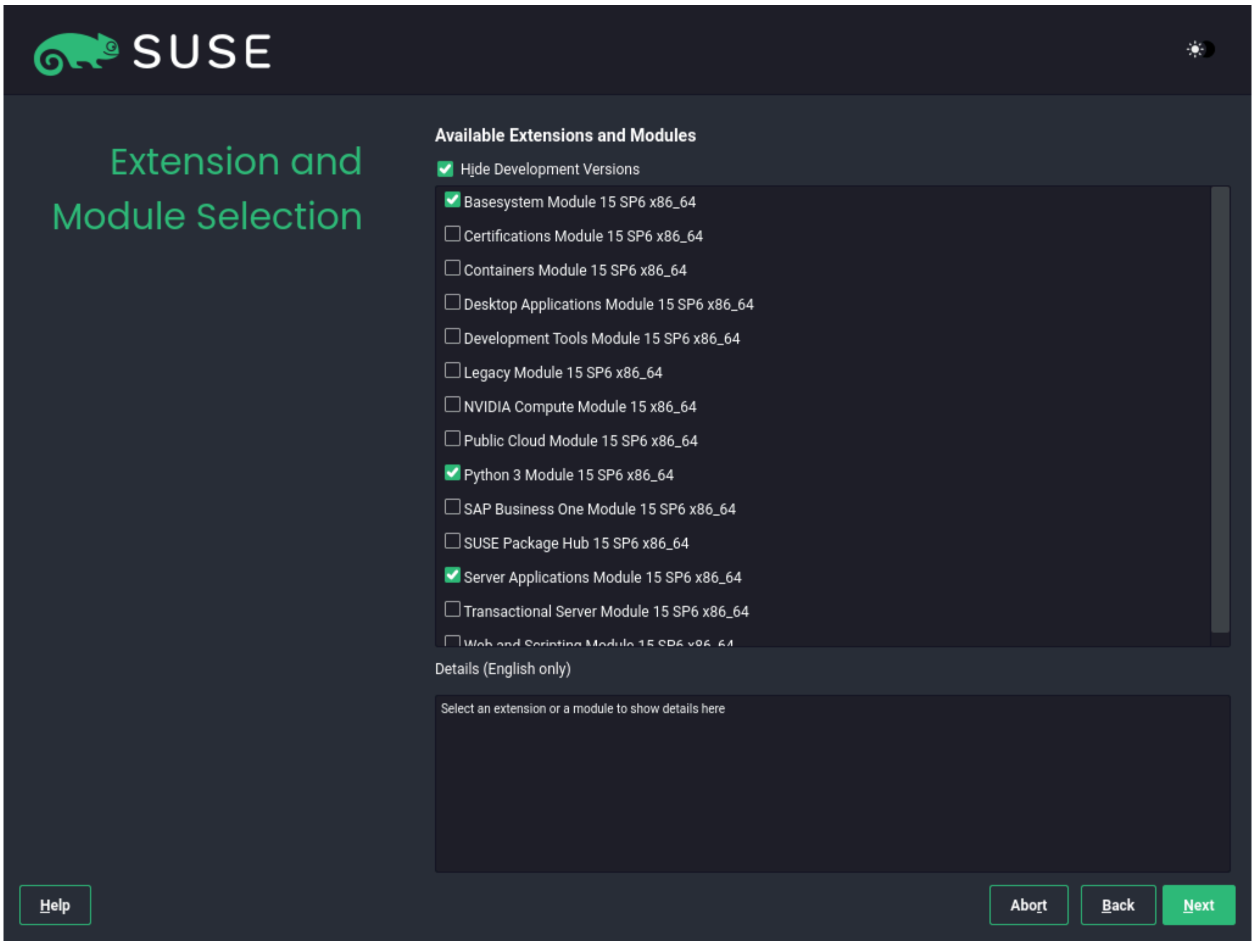
Tip: Installing product patches at installation time

After SLES has been successfully registered, you are asked whether to install the latest available online updates during the installation. If choosing *Yes*, the system will be installed with the most current packages without having to apply the updates after installation. Activating this option is recommended.

Note: Firewall settings for receiving updates

By default, the firewall on SUSE AI only blocks incoming connections. If your system is behind another firewall that blocks outgoing traffic, make sure to allow connections to `https://scc.suse.com/` and `https://updates.suse.com` on ports 80 and 443 to receive updates.

2.1.3.5 Extension and module selection



Available Extensions and Modules

- [✓] Hide Development Versions
- [✓] Basesystem Module 15 SP6 x86_64
- [ ] Certifications Module 15 SP6 x86_64
- [ ] Containers Module 15 SP6 x86_64
- [ ] Desktop Applications Module 15 SP6 x86_64
- [ ] Development Tools Module 15 SP6 x86_64
- [ ] Legacy Module 15 SP6 x86_64
- [ ] NVIDIA Compute Module 15 x86_64
- [ ] Public Cloud Module 15 SP6 x86_64
- [✓] Python 3 Module 15 SP6 x86_64
- [ ] SAP Business One Module 15 SP6 x86_64
- [ ] SUSE Package Hub 15 SP6 x86_64
- [✓] Server Applications Module 15 SP6 x86_64
- [ ] Transactional Server Module 15 SP6 x86_64
- [ ] Web and Scripting Module 15 SP6 x86_64

Details (English only)

Select an extension or a module to show details here

FIGURE 6: EXTENSION AND MODULE SELECTION

After the system is successfully registered, the installer lists modules and extensions that are available for SLES. Modules are components that allow you to customize the product according to your needs. They are included in your SLES subscription. Extensions add functionality to your product. They must be purchased separately.

The availability of certain modules or extensions depends on the product selected in the first step of the installation. For a description of the modules and their lifecycles, select a module to see the accompanying text. More detailed information is available in the Modules and extensins quick start guide (https://documentation.suse.com/sles-15/html/SLES-all/article-modules.html) ↗.

The selection of modules indirectly affects the scope of the installation, because it defines which software sources (repositories) are available for installation and in the running system.

The following modules and extensions are available for SUSE Linux Enterprise Server:

Basesystem Module

This module adds a basic system on top of the Unified Installer. It is required by all other modules and extensions. The scope of an installation that only contains the base system is comparable to the installation pattern `minimal system` of previous SLES versions. This module is selected for installation by default and should not be deselected.

Dependencies: None

Certifications Module

Contains the FIPS certification packages.

Dependencies: Server Applications

Confidential Computing Technical Preview

Contains packages related to confidential computing.

Dependencies: Basesystem

Containers Module

Contains support and tools for containers.

Dependencies: Basesystem

Desktop Applications Module

Adds a graphical user interface and essential desktop applications to the system.

Dependencies: Basesystem

Development Tools Module

Contains the compilers (including `gcc`) and libraries required for compiling and debugging applications. Replaces the former Software Development Kit (SDK).

Dependencies: Basesystem, Desktop Applications

High Performance Computing (HPC) Module

Provides specific tools commonly used for high performance, numerically intensive workloads.

Dependencies: Basesystem

Legacy Module

Helps you with migrating applications from earlier versions of SLES and other systems to SLES 15 SP6 by providing packages which are discontinued on SLE. Packages in this module are selected based on the requirements for migration and the level of complexity of configuration.

This module is recommended when migrating from a previous product version.

Dependencies: Basesystem, Server Applications

NVIDIA Compute Module

Contains the NVIDIA CUDA (Compute Unified Device Architecture) drivers.

The software in this module is provided by NVIDIA under the CUDA End User License Agreement (http://docs.nvidia.com/cuda/eula/) and is not supported by SUSE.

Dependencies: Basesystem

Public Cloud Module

Contains all tools required to create images for deploying SLES in cloud environments such as Amazon Web Services (AWS), Microsoft Azure, Google Compute Platform, or OpenStack.

Dependencies: Basesystem, Server Applications

Python 3 Module

This module contains the most recent versions of the selected Python 3 packages.

Dependencies: Basesystem

SAP Business One Server

This module contains packages and system configurations specific to SAP Business One Server. It is maintained and supported under the SUSE Linux Enterprise Server product subscription.

Dependencies: Basesystem, Server Applications, Desktop Applications, Development Tools

Server Applications Module

Adds server functionality by providing network services such as DHCP server, name server, or Web server. This module is selected for installation by default. Deselecting it is not recommended.

Dependencies: Basesystem

SLE High Availability

Adds clustering support for mission-critical setups to SLES. This extension requires a separate license key.

Dependencies: Basesystem, Server Applications

SLE Live Patching

Adds support for performing critical patching without having to shut down the system. This extension requires a separate license key.

Dependencies: Basesystem, Server Applications

SUSE Linux Enterprise Workstation Extension

Extends the functionality of SLES with packages from SUSE Linux Enterprise Desktop, like additional desktop applications (office suite, e-mail client, graphical editor, etc.) and libraries. It allows combining both products to create a fully featured workstation. This extension requires a separate license key.
Dependencies: Basesystem, Desktop Applications

SUSE Package Hub

Provides access to packages for SLES maintained by the openSUSE community. These packages are delivered without L3 support and do not interfere with the supportability of SLES. For more information, refer to https://packagehub.suse.com/ ↗.
Dependencies: Basesystem

Transactional Server Module

Adds support for transactional updates. Updates are either applied to the system as a single transaction or not applied at all. This happens without influencing the running system. If an update fails, or if the successful update is deemed to be incompatible or otherwise incorrect, it can be discarded to immediately return the system to its previous functioning state.
Dependencies: Basesystem

Web and Scripting Module

Contains packages intended for a running Web server.
Dependencies: Basesystem, Server Applications

Certain modules depend on the installation of other modules. Therefore, when selecting a module, other modules may be selected automatically to fulfill dependencies.

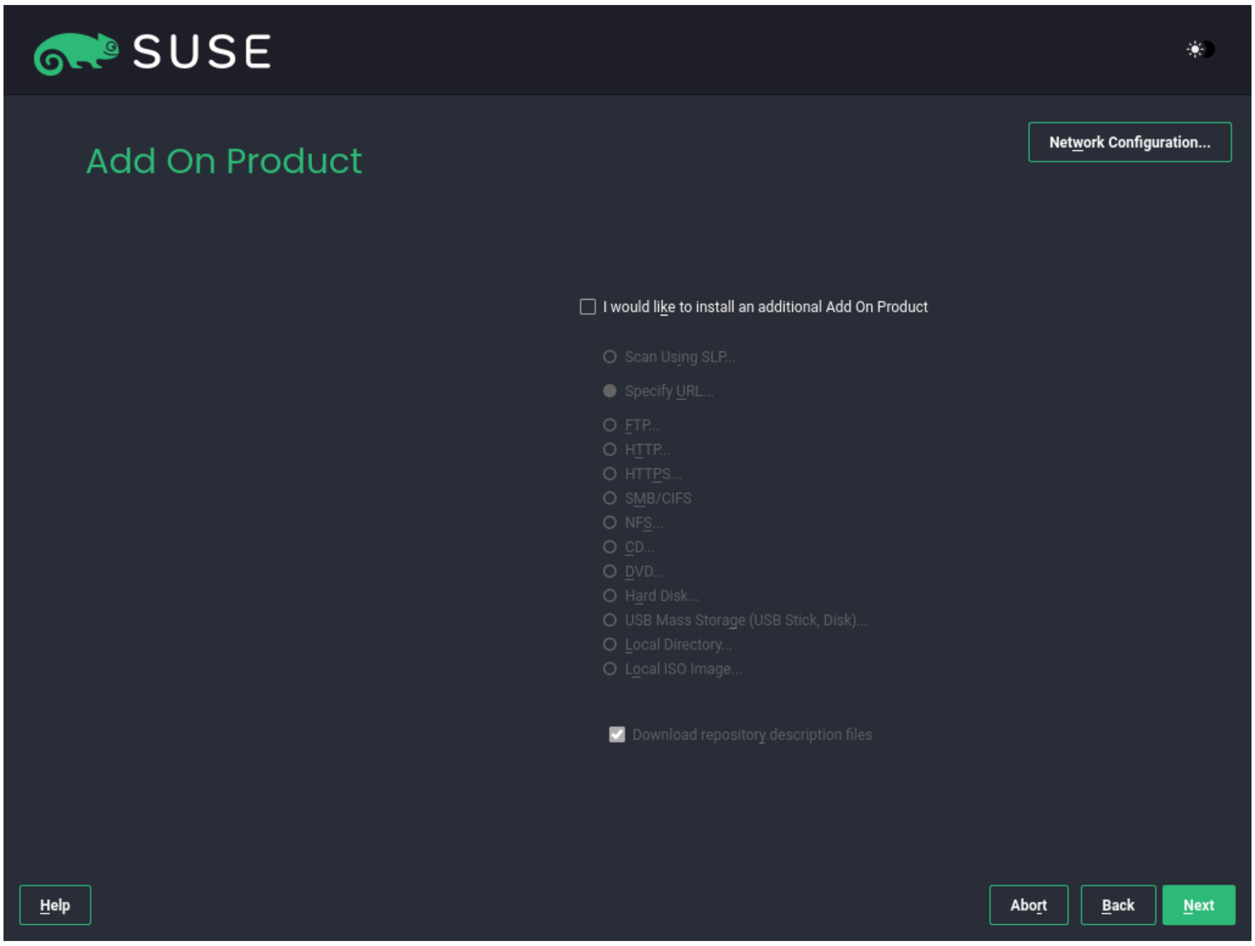
Depending on the product, the registration server can mark modules and extensions as recommended. Recommended modules and extensions are preselected for registration and installation. To avoid installing these recommendations, deselect them manually.

Select the modules and extensions you want to install and proceed with *Next*. In case you have chosen one or more extensions, you will be prompted to provide the respective registration codes. Depending on your choice, it may also be necessary to accept additional license agreements.

Important: Default modules for offline installation

When performing an offline installation from the 15 SP6-Full-ARCH-GM-media1.iso, only the *Basesystem Module* is selected by default. To install the complete default package set of SUSE Linux Enterprise Server, additionally select the *Server Applications Module* and the *Python 3 Module*.

2.1.3.6 Add-on product



☐ I would like to install an additional Add On Product

○ Scan Using SLP...
● Specify URL...
○ FTP...
○ HTTP...
○ HTTPS...
○ SMB/CIFS
○ NFS...
○ CD...
○ DVD...
○ Hard Disk...
○ USB Mass Storage (USB Stick, Disk)...
○ Local Directory...
○ Local ISO Image...

☑ Download repository description files

FIGURE 7: ADD-ON PRODUCT

The *Add-On Product* dialog allows you to add additional software sources (called "repositories") to SLES that are not provided by the SUSE Customer Center. Add-on products may include third-party products and drivers as well as additional software for your system.

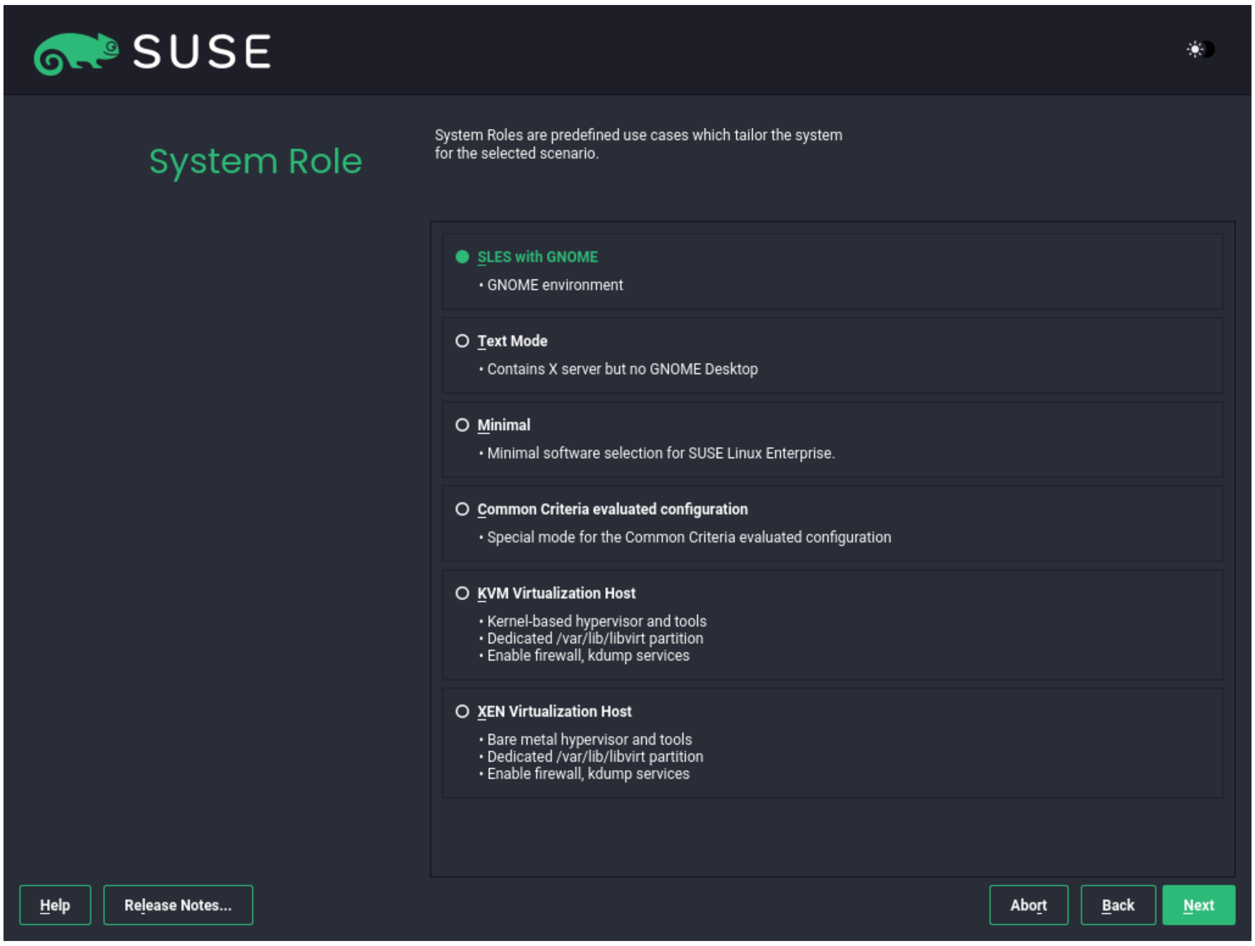
Tip: Adding drivers during the installation

You can also add driver update repositories via the *Add-On Product* dialog. Driver updates for SLE are provided at https://drivers.suse.com/ . These drivers have been created through the SUSE SolidDriver Program.

To skip this step, proceed with *Next*. Otherwise, activate *I would like to install an additional Add On Product*. Specify a media type, a local path, or a network resource hosting the repository and follow the on-screen instructions.

Check *Download Repository Description Files* to download the files describing the repository now. If deactivated, they will be downloaded after the installation has started. Proceed with *Next* and insert a medium if required. Depending on the content of the product, it may be necessary to accept additional license agreements. Proceed with *Next*. If you have chosen an add-on product requiring a registration key, you will be asked to enter it before proceeding to the next step.

2.1.3.7 System role


System Role

System Roles are predefined use cases which tailor the system for the selected scenario.

● SLES with GNOME
- GNOME environment

○ Text Mode
- Contains X server but no GNOME Desktop

○ Minimal
- Minimal software selection for SUSE Linux Enterprise.

○ Common Criteria evaluated configuration
- Special mode for the Common Criteria evaluated configuration

○ KVM Virtualization Host
- Kernel-based hypervisor and tools
- Dedicated /var/lib/libvirt partition
- Enable firewall, kdump services

○ XEN Virtualization Host
- Bare metal hypervisor and tools
- Dedicated /var/lib/libvirt partition
- Enable firewall, kdump services

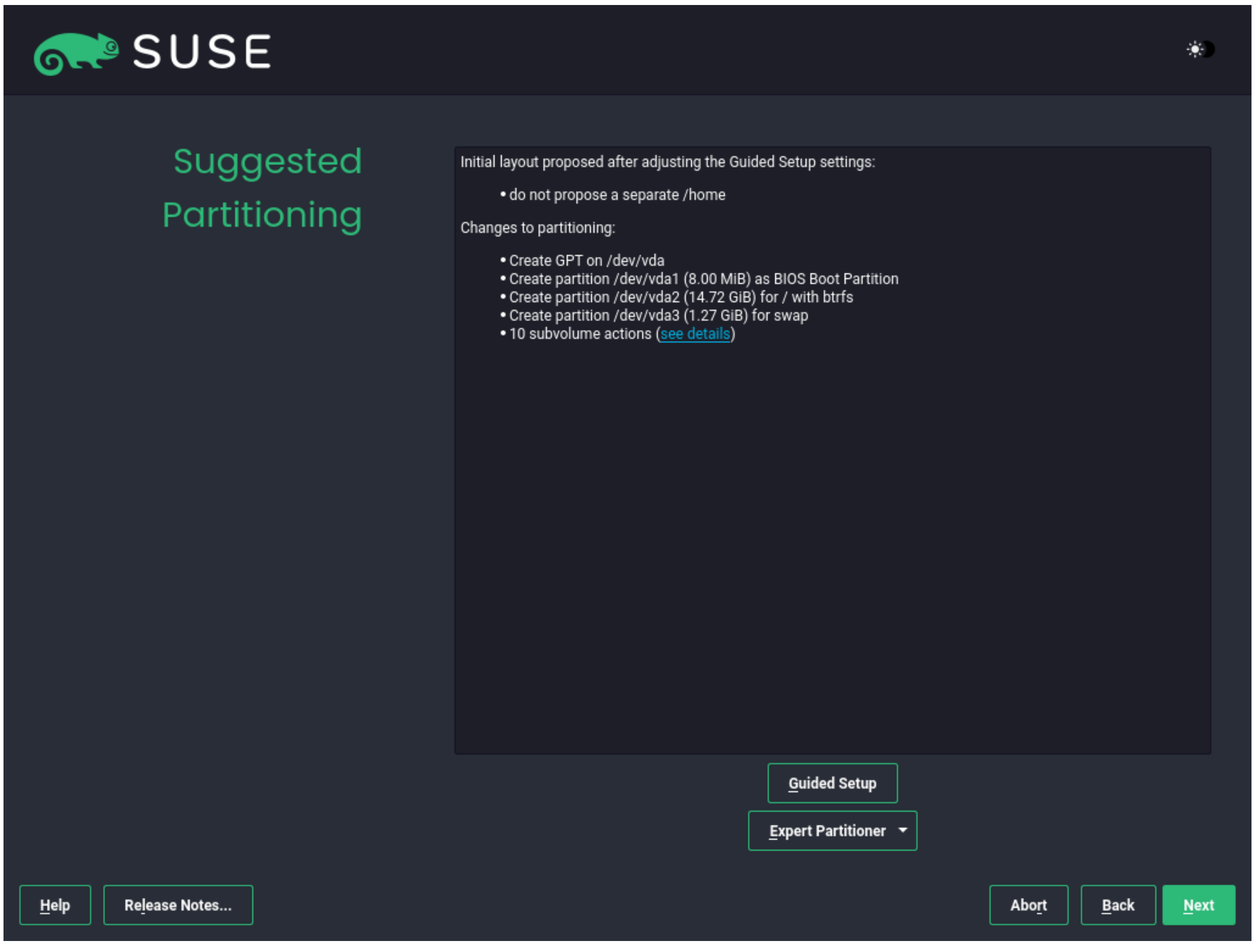
FIGURE 8: SYSTEM ROLE

The availability of system roles depends on your selection of modules and extensions. System roles define, for example, the set of software patterns that are preselected for the installation. Refer to the description on the screen to make your choice. Select a role and proceed with *Next*. If from the enabled modules only one role or no role is suitable for the respective base product, the *System Role* dialog is omitted.

Tip: Release notes

From this point on, the Release Notes can be viewed from any screen during the installation process by selecting *Release Notes*.

2.1.3.8 Suggested partitioning



Initial layout proposed after adjusting the Guided Setup settings:

- do not propose a separate /home

Changes to partitioning:

- Create GPT on /dev/vda
- Create partition /dev/vda1 (8.00 MiB) as BIOS Boot Partition
- Create partition /dev/vda2 (14.72 GiB) for / with btrfs
- Create partition /dev/vda3 (1.27 GiB) for swap
- 10 subvolume actions (see details)

FIGURE 9: SUGGESTED PARTITIONING

Review the partition setup proposed by the system. If necessary, change it. You have the following options:

Guided setup

Starts a wizard that lets you refine the partitioning proposal. The options available here depend on your system setup. If it contains more than a single hard disk, you can choose which disk or disks to use and where to place the root partition. If the disks already contain partitions, decide whether to remove or resize them.

In subsequent steps, you may also add LVM support and disk encryption. You can change the file system for the root partition and decide whether or not to have a separate home partition.

Expert partitioner

Opens the *Expert Partitioner*. This gives you full control over the partitioning setup and lets you create a custom setup. This option is intended for experts. For details, see the Expert Partitioner (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-expert-partitioner.html#sec-expert-partitioner) chapter.

Warning: Disk space units

For partitioning purposes, disk space is measured in binary units rather than in decimal units. For example, if you enter sizes of `1GB`, `1GiB` or `1G`, they all signify 1 GiB (Gibibyte), as opposed to 1 GB (Gigabyte).

Binary

1 GiB = 1,073,741,824 bytes.

Decimal

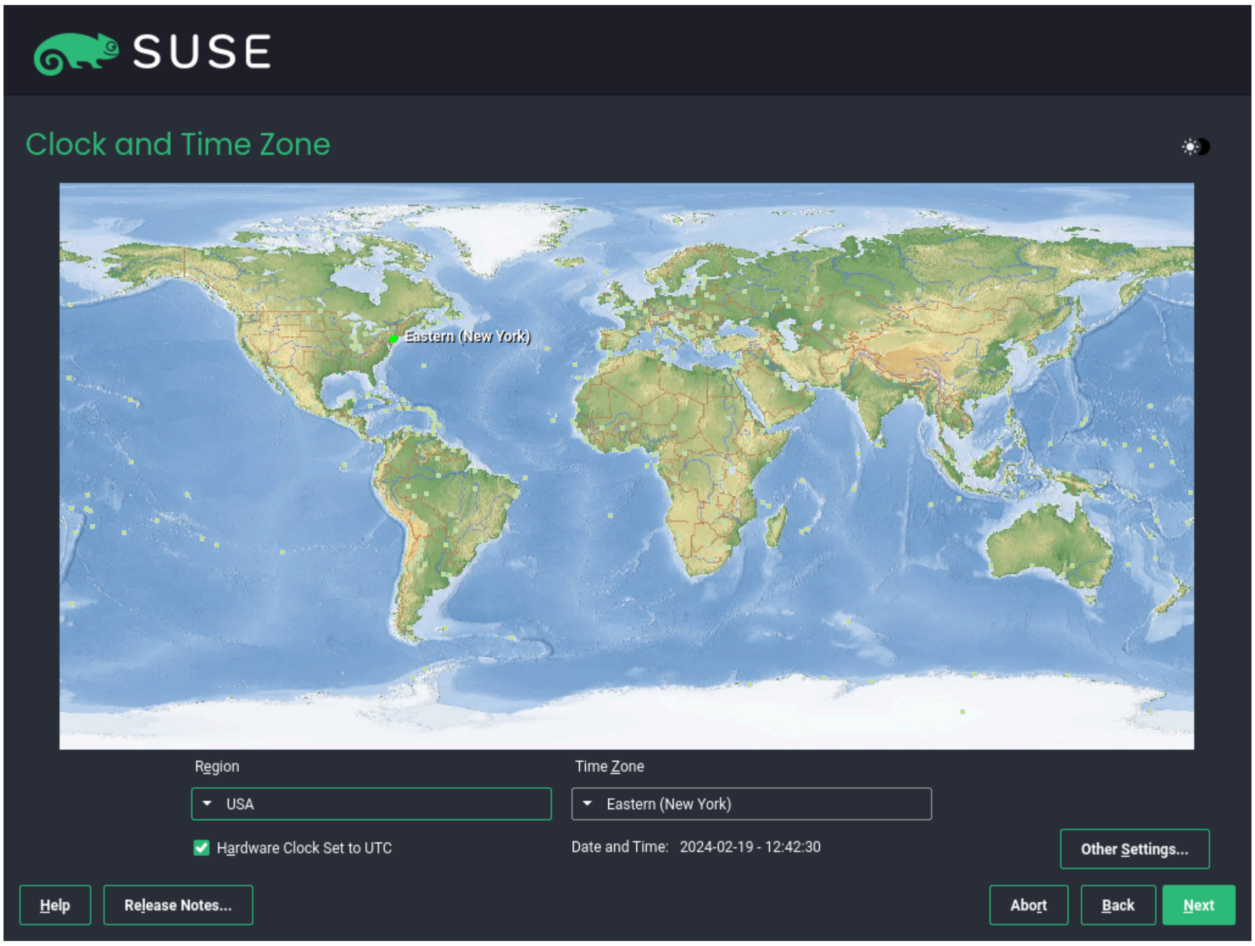
1 GB = 1,000,000,000 bytes.

Difference

1 GiB ≈ 1.07 GB.

To accept the proposed setup without any changes, choose *Next* to proceed.

2.1.3.9 Clock and time zone



Region: USA

Time Zone: Eastern (New York)

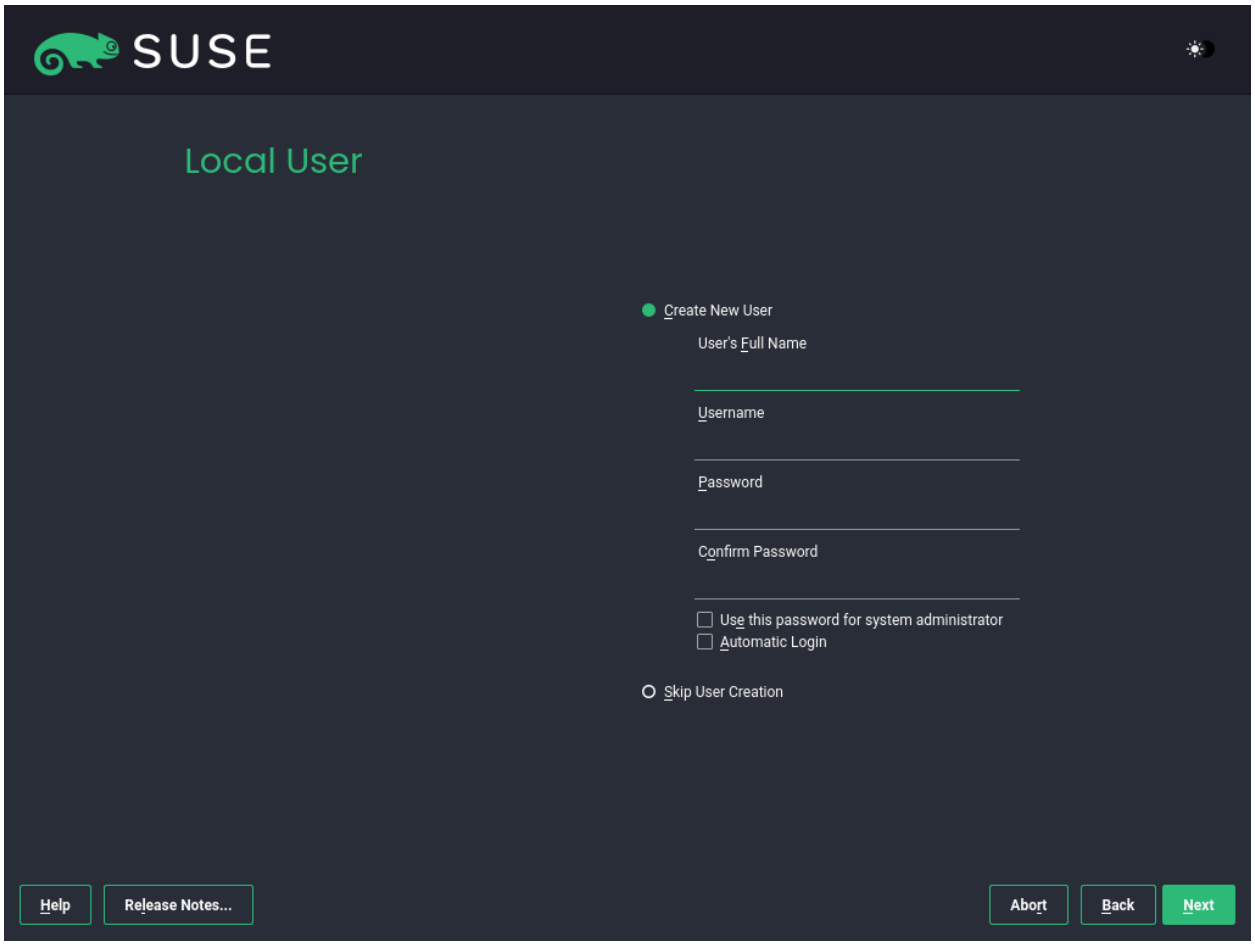
✓ Hardware Clock Set to UTC

Date and Time: 2024-02-19 - 12:42:30

FIGURE 10: CLOCK AND TIME ZONE

Select the clock and time zone to use in your system. To manually adjust the time or to configure an NTP server for time synchronization, choose *Other Settings*. See the section on Clock and Time Zone (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-install.html#sec-yast-install-date-time) for detailed information. Proceed with *Next*.

2.1.3.10 Local user



Local User

● Create New User

User's Full Name

Username

Password

Confirm Password

☐ Use this password for system administrator

☐ Automatic Login

○ Skip User Creation

FIGURE 11: LOCAL USER CREATION

To create a local user, type the first and last name in the *User's Full Name* field, the login name in the *Username* field, and the password in the *Password* field.

The password should be at least eight characters long and should contain both uppercase and lowercase letters and numbers. The maximum length for passwords is 72 characters, and passwords are case-sensitive.

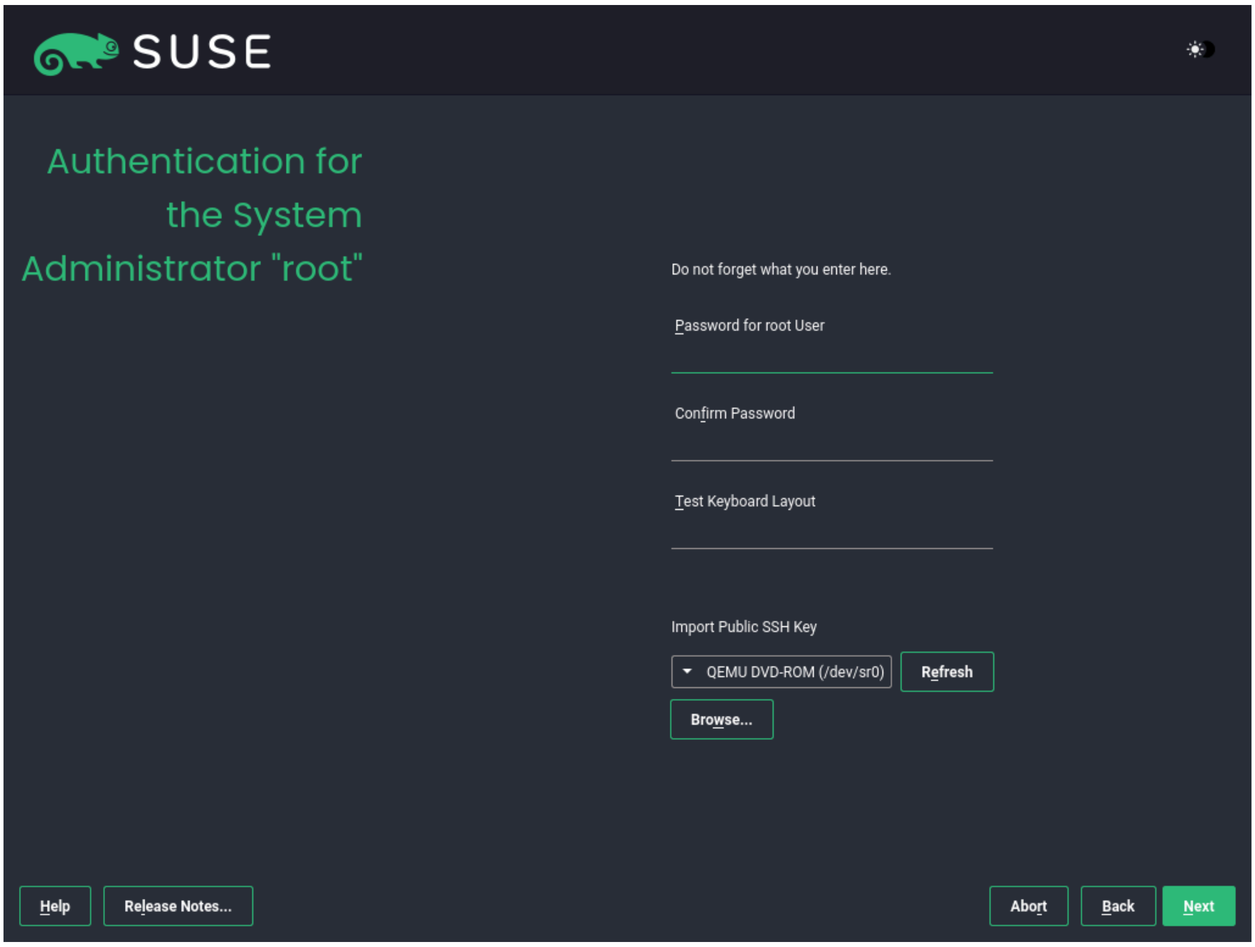
For security reasons, it is also strongly recommended **not** to enable *Automatic Login*. You should also **not** *Use this Password for the System Administrator* but provide a separate root password in the next installation step.

If you install on a system where a previous Linux installation was found, you may *Import User Data from a Previous Installation*. Click *Choose User* for a list of available user accounts. Select one or more users.

In an environment where users are centrally managed (for example, by NIS or LDAP), you can skip the creation of local users. Select *Skip User Creation* in this case.

Proceed with *Next*.

2.1.3.11 Authentication for the system administrator `root`



Do not forget what you enter here.

FIGURE 12: PASSWORD FOR THE SYSTEM ADMINISTRATOR `root`

Type a password for the system administrator (called the root user) or provide a public SSH key. If you want, you can use both.

Because the root user is equipped with extensive permissions, the password should be chosen carefully. You should never forget the root password. After you entered it here, the password cannot be retrieved.

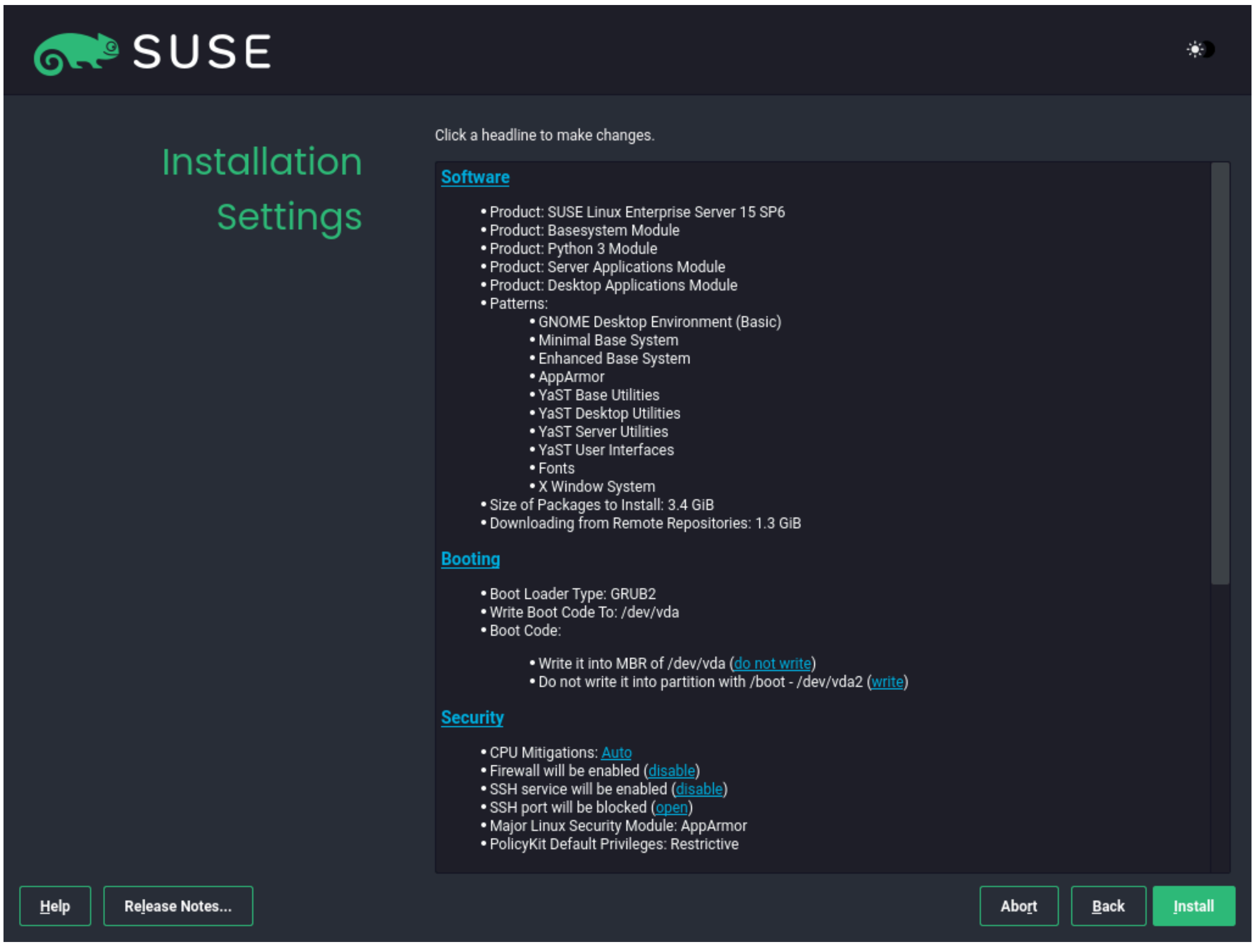
Tip: Passwords and keyboard layout

It is recommended to use only US ASCII characters. In the event of a system error or when you need to start your system in rescue mode, the keyboard may not be localized.

To access the system remotely via SSH using a public key, import a key from removable media or an existing partition. See the section on Authentication for the system administrator root (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-install.html#sec-yast-install-user-root) for more information.

Proceed with *Next*.

2.1.3.12 Installation settings



Click a headline to make changes.

Software

- Product: SUSE Linux Enterprise Server 15 SP6
- Product: Basesystem Module
- Product: Python 3 Module
- Product: Server Applications Module
- Product: Desktop Applications Module
- Patterns:
  - GNOME Desktop Environment (Basic)
  - Minimal Base System
  - Enhanced Base System
  - AppArmor
  - YaST Base Utilities
  - YaST Desktop Utilities
  - YaST Server Utilities
  - YaST User Interfaces
  - Fonts
  - X Window System
- Size of Packages to Install: 3.4 GiB
- Downloading from Remote Repositories: 1.3 GiB

Booting

- Boot Loader Type: GRUB2
- Write Boot Code To: /dev/vda
- Boot Code:
  - Write it into MBR of /dev/vda (do not write)
  - Do not write it into partition with /boot - /dev/vda2 (write)

Security

- CPU Mitigations: Auto
- Firewall will be enabled (disable)
- SSH service will be enabled (disable)
- SSH port will be blocked (open)
- Major Linux Security Module: AppArmor
- PolicyKit Default Privileges: Restrictive

FIGURE 13: INSTALLATION SETTINGS

Use the *Installation Settings* screen to review and—if necessary—change several proposed installation settings. The current configuration is listed for each setting. To change it, click the headline. Certain settings, such as firewall or SSH, can be changed directly by clicking the respective links.

Important: Remote access

Changes you can make here can also be made later at any time from the installed system. However, if you need remote access right after the installation, you may need to open the SSH port in the *Security* settings.

Software

The scope of the installation is defined by the modules and extensions you have chosen for this installation. However, depending on your selection, not all packages available in a module are selected for installation.

Clicking *Software* opens the *Software Selection and System Tasks* screen, where you can change the software selection by selecting or deselecting patterns. Each pattern contains several software packages needed for specific functions (for example, *KVM Host Server*). For a more detailed selection based on software packages to install, select *Details* to switch to the YaST *Software Manager*. See Installing or removing software (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-yast-software.html) ↗ for more information.

Booting

This section shows the boot loader configuration. Changing the defaults is recommended only if really needed. Refer to The boot loader GRUB 2 (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-grub2.html) ↗ for details.

Security

The *CPU Mitigations* refer to kernel boot command-line parameters for software mitigations that have been deployed to prevent CPU side-channel attacks. Click the selected entry to choose a different option. For details, see the section on CPU Mitigations (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-grub2.html#vle-grub2-yast2-cpu-mitigations) ↗.

By default, the *Firewall* is enabled on all configured network interfaces. To disable firewalld, click *disable* (not recommended). Refer to the Masquerading and Firewalls (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-security-firewall.html) ↗ chapter for configuration details.

Note: Firewall settings for receiving updates

By default, the firewall on SUSE AI only blocks incoming connections. If your system is behind another firewall that blocks outgoing traffic, make sure to allow connections to `https://scc.suse.com/` and `https://updates.suse.com` on ports 80 and 443 to receive updates.

The *SSH service* is enabled by default, but its port (22) is closed in the firewall. Click *open* to open the port or *disable* to disable the service. If SSH is disabled, remote logins will not be possible. Refer to Securing network operations with OpenSSH (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-ssh.html) ↗ for more information.
The default *Major Linux Security Module* is *AppArmor*. To disable it, select *None* as the module in the *Security* settings.

Security Policies

Click to *enable* the `Defense Information Systems Agency STIG` security policy. If any installation settings are incompatible with the policy, you will be prompted to modify them accordingly. Certain settings can be adjusted automatically while others require user input. Enabling a security profile enables a full SCAP remediation on first boot. You can also perform a *scan only* or *do nothing* and manually remediate the system later with OpenSCAP. For more information, refer to the section on Security Profiles (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-install.html#sec-yast-install-proposal-security-profile) ↗.

Network configuration

Displays the current network configuration. By default, `wicked` is used for server installations and NetworkManager for desktop workloads. Click *Network Configuration* to change the settings. For details, see the section on Configuring a network connection with YaST (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-network.html#sec-network-yast) ↗.

Important: Support for NetworkManager

SUSE only supports NetworkManager for desktop workloads with SLED or the Workstation extension. All server certifications are done with `wicked` as the network configuration tool, and using NetworkManager may invalidate them. NetworkManager is not supported by SUSE for server workloads.

Kdump

Kdump saves the memory image ("core dump") to the file system in case the kernel crashes. This enables you to find the cause of the crash by debugging the dump file. Kdump is preconfigured and enabled by default. See the Basic Kdump configuration (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-tuning-kexec.html#cha-tuning-kdump-basic) ↗ for more information.

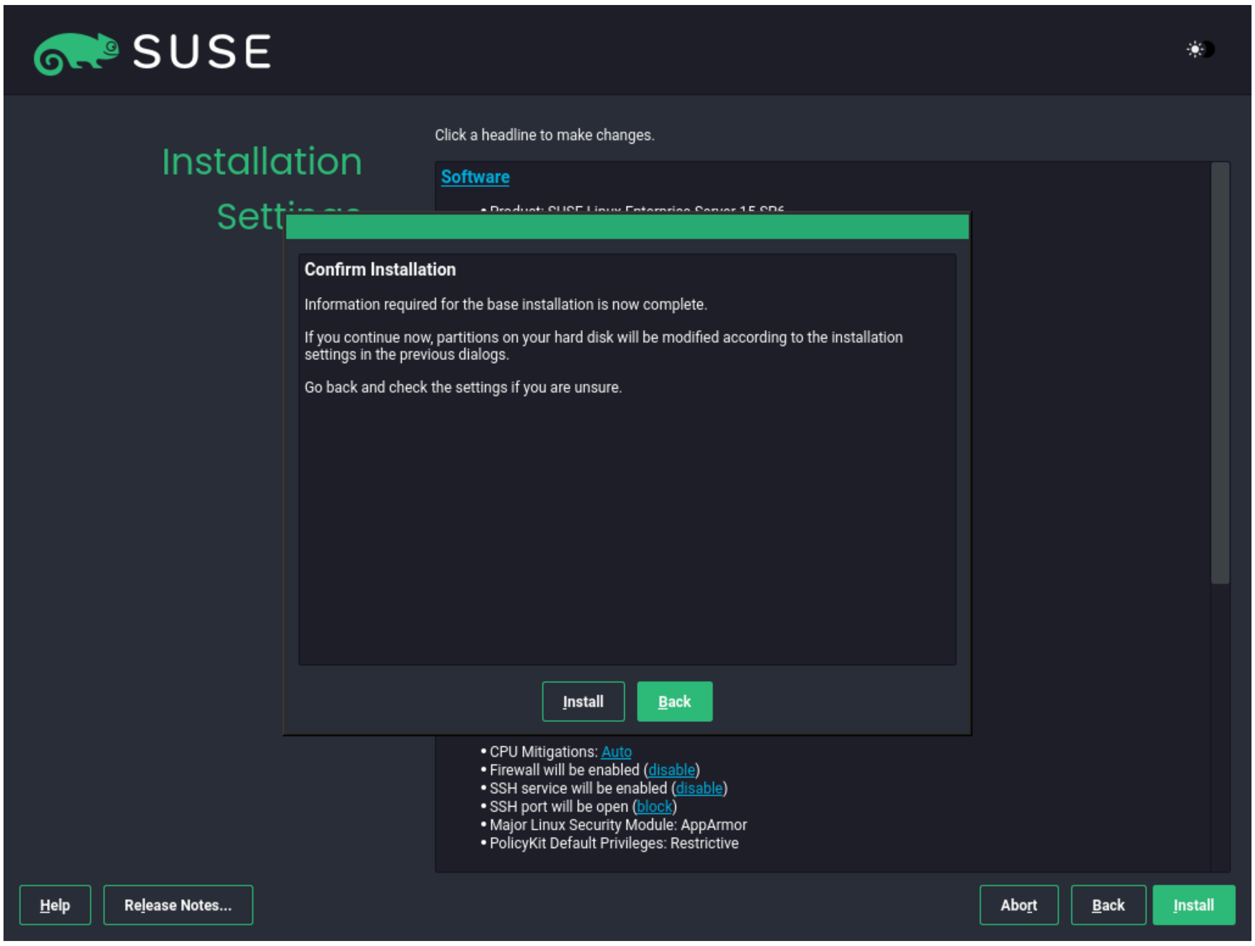
Default systemd target

If you have installed the desktop applications module, the system boots into the *graphical* target, with network, multi-user and display manager support. Switch to *multi-user* if you do not need to log in via a display manager.

System

View detailed hardware information by clicking *System*. In the resulting screen, you can also change *Kernel Settings*—see the section on System Information (https://documentation.suse.com/sles/15-SP6/html/SLES-all/cha-install.html#sec-yast-install-proposal-system) ↗ for more information.

2.1.3.13 Start the installation



Click a headline to make changes.

Confirm Installation

Information required for the base installation is now complete.

If you continue now, partitions on your hard disk will be modified according to the installation settings in the previous dialogs.

Go back and check the settings if you are unsure.

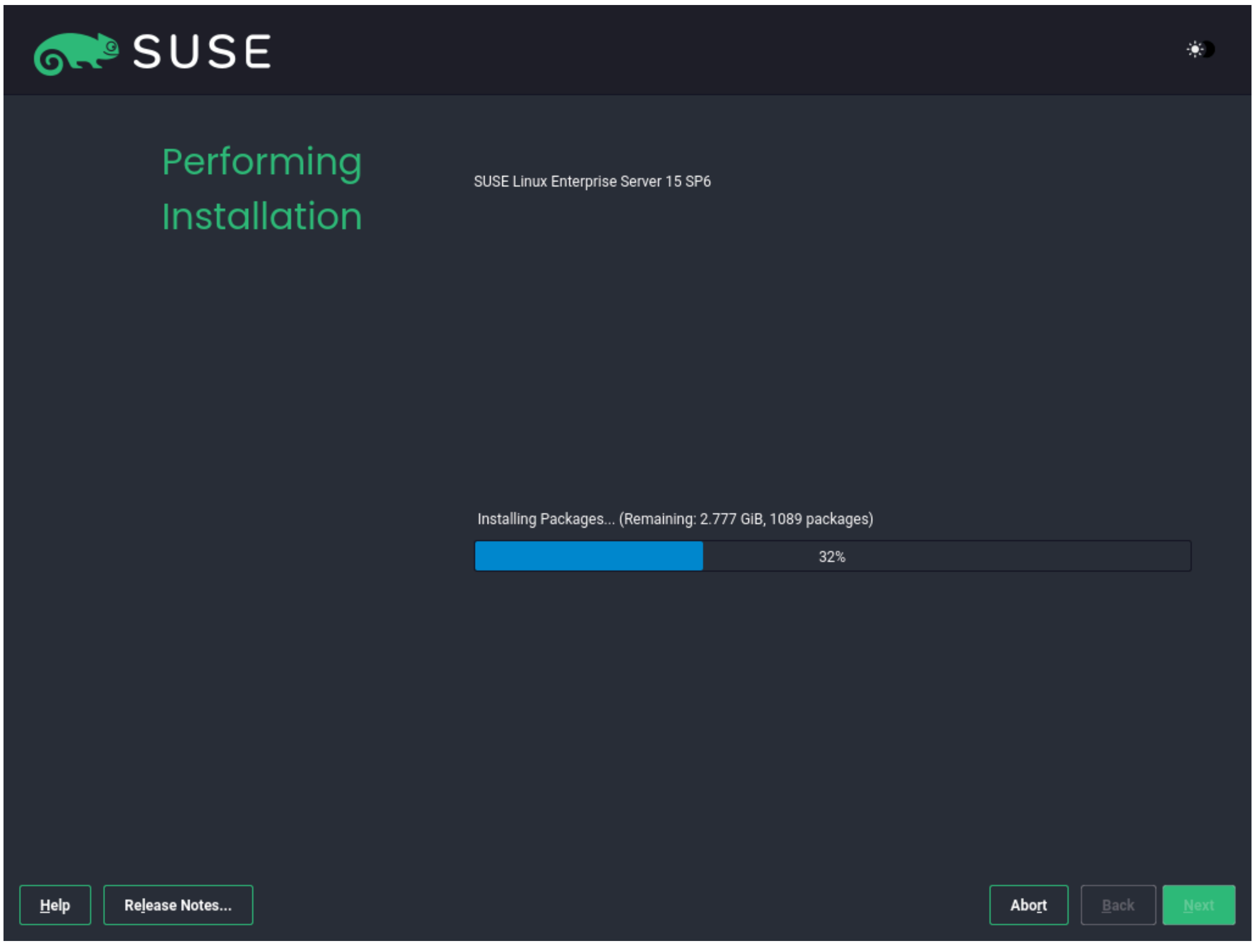
Install | Back

- CPU Mitigations: Auto
- Firewall will be enabled (disable)
- SSH service will be enabled (disable)
- SSH port will be open (block)
- Major Linux Security Module: AppArmor
- PolicyKit Default Privileges: Restrictive

FIGURE 14: CONFIRM INSTALLATION

After you have finalized the system configuration on the *Installation Settings* screen, click *Install*. Depending on your software selection, you may need to agree to license agreements before the installation confirmation screen pops up. Up to this point, no changes have been made to your system. After you click *Install* a second time, the installation process starts.

2.1.3.14 The installation process



SUSE Linux Enterprise Server 15 SP6

Installing Packages... (Remaining: 2.777 GiB, 1089 packages)

32%

FIGURE 15: PERFORMING THE INSTALLATION

During the installation, the progress is shown. After the installation routine has finished, the computer is rebooted into the installed system.

2.2 Installing NVIDIA GPU drivers

This article demonstrates how to implement host-level NVIDIA GPU support via the `open-driver`. The `open-driver`is part of the core package repositories. Therefore, there is no need to compile it or download executable packages. This driver is built into the operating system rather than dynamically loaded by the NVIDIA GPU Operator. This configuration is desirable for customers who want to pre-build all artifacts required for deployment into the image, and where the dynamic selection of the driver version via Kubernetes is not a requirement.

2.2.1 Installing NVIDIA GPU drivers on SUSE Linux Enterprise Server

2.2.1.1 Requirements

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

- At least one host with SLES 15 SP6 installed, physical or virtual.
- Your hosts are attached to a subscription as this is required for package access.
- A compatible NVIDIA GPU (https://github.com/NVIDIA/open-gpu-kernel-modules#compatible-gpus) ↗ installed or fully passed through to the virtual machine in which SLES is running.
- Access to the root user—these instructions assume you are the root user, and not escalating your privileges via `sudo`.

2.2.1.2 Considerations before the installation

2.2.1.2.1 Select the driver generation

You must verify the driver generation for the NVIDIA GPU that your system has. For modern GPUs, the `G06` driver is the most common choice. Find more details in the support database (https://en.opensuse.org/SDB:NVIDIA_drivers#Install) ↗.

This section details the installation of the `G06` generation of the driver.

2.2.1.2.2 Additional NVIDIA components

Besides the NVIDIA open-driver provided by SUSE as part of SLES, you might also need additional NVIDIA components. These could include OpenGL libraries, CUDA toolkits, command-line utilities such as `nvidia-smi`, and container-integration components such as nvidia-container-toolkit. Many of these components are not shipped by SUSE as they are proprietary NVIDIA software. This section describes how to configure additional repositories that give you access to these components and provides examples of using these tools to achieve a fully functional system.

2.2.1.3 The installation procedure

1. Add a package repository from NVIDIA. This allows pulling in additional utilities, for example, `nvidia-smi`.

   For the AMD64/Intel 64 architecture, run:

   ```
   # zypper ar \
     https://developer.download.nvidia.com/compute/cuda/repos/sles15/x86_64/ \
     cuda-sle15
   # zypper --gpg-auto-import-keys refresh
   ```

   For the Arm AArch64 architecture, run:

   ```
   # zypper ar \
     https://developer.download.nvidia.com/compute/cuda/repos/sles15/sbsa/ \
     cuda-sle15
   transactional update # zypper --gpg-auto-import-keys refresh
   ```

2. Install the Open Kernel driver KMP and detect the driver version.

   ```
   {prompt_root}zypper install -y --auto-agree-with-licenses \
     nv-prefer-signed-open-driver
   {prompt_root}version=$(rpm -qa --queryformat '%{VERSION}\n' \
     nv-prefer-signed-open-driver | cut -d "_" -f1 | sort -u | tail -n 1)
   ```

3. You can then install the appropriate packages for additional utilities that are useful for testing purposes.

   ```
   {prompt_root}zypper install -y --auto-agree-with-licenses \
   nvidia-compute-utils-G06=${version} \
   nvidia-persistenced=${version}
   ```

4. Reboot the host to make the changes effective.

   ```
   # reboot
   ```

5. Log back in and use the `nvidia-smi` tool to verify that the driver is loaded successfully and that it can both access and enumerate your GPUs.

   ```
   # nvidia-smi
   ```

   The output of this command should show you something similar to the following output. In the example below, the system has one GPU.

   ```
   Fri Aug  1 15:32:10 2025
   ```

```
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 580.82.07      Driver Version: 580.82.07    CUDA Version: 13.0    |
|-----------------------------+----------------------+----------------------+
| 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  Tesla T4            On  |   00000000:00:1E.0 Off |                    0 |
| N/A   33C    P8    13W /   70W |       0MiB /  15360MiB |      0%      Default |
|                              |                        |                  N/A |
+-----------------------------+----------------------+----------------------+


+-----------------------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------------------+
```

2.2.1.4 Validation of the driver installation

Running the `nvidia-smi` command has verified that, at the host level, the NVIDIA device can be accessed and that the drivers are loading successfully. To validate that it is functioning, you need to validate that the GPU can take instructions from a user-space application, ideally via a container and through the CUDA library, as that is typically what a real workload would use. For this, we can make a further modification to the host OS by installing `nvidia-container-toolkit`.

1. Install the `nvidia-container-toolkit` package from the NVIDIA Container Toolkit repository.

```
# zypper ar \
"https://nvidia.github.io/libnvidia-container/stable/rpm/"\
nvidia-container-toolkit.repo
# zypper --gpg-auto-import-keys install \
  -y nvidia-container-toolkit
```

The `nvidia-container-toolkit.repo` file contains a stable repository `nvidia-container-toolkit` and an experimental repository `nvidia-container-toolkit-experimental`. Use the stable repository for production use. The experimental repository is disabled by default.

2. Verify that the system can successfully enumerate the devices using the NVIDIA Container Toolkit. The output should be verbose, with INFO and WARN messages, but no ERROR messages.

   ```
   # nvidia-ctk cdi generate --output=/etc/cdi/nvidia.yaml
   ```

   This ensures that any container started on the machine can employ discovered NVIDIA GPU devices.

3. You can then run a Podman-based container. Doing this via `podman` gives you a good way of validating access to the NVIDIA device from within a container, which should give confidence for doing the same with Kubernetes at a later stage.
   Give Podman access to the labeled NVIDIA devices that were taken care of by the previous command and simply run the `bash` command.

   ```
   # podman run --rm --device nvidia.com/gpu=all \
     --security-opt=label=disable \
     -it registry.suse.com/bci/bci-base:latest bash
   ```

   You can now execute commands from within a temporary Podman container. It does not have access to your underlying system and is **ephemeral**—whatever you change in the container does not persist. Also, you cannot break anything on the underlying host.

4. Inside the container, install the required CUDA libraries. Identify their version from the output of the `nvidia-smi` command. From the above example, we are installing CUDA version 13.0 with many examples, demos and development kits to fully validate the GPU.

   ```
   # zypper ar \
     http://developer.download.nvidia.com/compute/cuda/repos/sles15/x86_64/ \
     cuda-sle15-sp6
   # zypper --gpg-auto-import-keys refresh
   # zypper install -y cuda-libraries-13-0 cuda-demo-suite-12-9
   ```

5. Inside the container, run the `deviceQuery` CUDA example of the same version, which comprehensively validates GPU access via CUDA and from within the container itself.

   ```
   # /usr/local/cuda-12.9/extras/demo_suite/deviceQuery Starting...
   ```

```
 CUDA Device Query (Runtime API)

Detected 1 CUDA Capable device(s)

Device 0: "Tesla T4"
  CUDA Driver Version / Runtime Version          13.0/ 13.0
  CUDA Capability Major/Minor version number:    7.5
  Total amount of global memory:                 14913 MBytes (15637086208 bytes)
  (40) Multiprocessors, ( 64) CUDA Cores/MP:     2560 CUDA Cores
  GPU Max Clock rate:                            1590 MHz (1.59 GHz)
  Memory Clock rate:                             5001 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 4194304 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:  1024
  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 / 0 / 30
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device
 simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 13.0, CUDA Runtime Version
 = 13.0, NumDevs = 1, Device0 = Tesla T4
Result = PASS
```

From inside the container, you can continue to run any other CUDA workload—such as compilers—to run further tests. When finished, you can exit the container.

```
# exit
```

Important

Changes you have made in the container and packages you have installed inside will be lost and will not impact the underlying operating system.

2.2.2 Installing NVIDIA GPU drivers on SUSE Linux Enterprise Micro

2.2.2.1 Requirements

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

- At least one host with SUSE Linux Enterprise Micro 6.1 installed, physical or virtual.
- Your hosts are attached to a subscription as this is required for package access.
- A compatible NVIDIA GPU (https://github.com/NVIDIA/open-gpu-kernel-modules#compatible-gpus) installed or fully passed through to the virtual machine in which SUSE Linux Enterprise Micro is running.
- Access to the root user—these instructions assume you are the root user, and not escalating your privileges via `sudo`.

2.2.2.2 Considerations before the installation

2.2.2.2.1 Select the driver generation

You must verify the driver generation for the NVIDIA GPU that your system has. For modern GPUs, the `G06` driver is the most common choice. Find more details in the support database (https://en.opensuse.org/SDB:NVIDIA_drivers#Install).

This section details the installation of the `G06` generation of the driver.

2.2.2.2.2 Additional NVIDIA components

Besides the NVIDIA open-driver provided by SUSE as part of SUSE Linux Enterprise Micro, you might also need additional NVIDIA components. These could include OpenGL libraries, CUDA toolkits, command-line utilities such as `nvidia-smi`, and container-integration components such as nvidia-container-toolkit. Many of these components are not shipped by SUSE as they are proprietary NVIDIA software. This section describes how to configure additional repositories that give you access to these components and provides examples of using these tools to achieve a fully functional system.

2.2.2.3 The installation procedure

1. On each (local) GPU-enabled host, open up a transactional-update shell session to create a new read/write snapshot of the underlying operating system so that we can make changes to the immutable platform.

```
# transactional-update shell
```

2. When you are in the transactional-update shell session, add a package repository from NVIDIA. This allows pulling in additional utilities, for example, `nvidia-smi`.
For the AMD64/Intel 64 architecture, run:

```
transactional update # zypper ar \
  https://developer.download.nvidia.com/compute/cuda/repos/sles15/x86_64/ \
  cuda-sle15
transactional update # zypper --gpg-auto-import-keys refresh
```

For the Arm AArch64 architecture, run:

```
transactional update # zypper ar \
  https://developer.download.nvidia.com/compute/cuda/repos/sles15/sbsa/ \
  cuda-sle15
transactional update # zypper --gpg-auto-import-keys refresh
```

3. Install the Open Kernel driver KMP and detect the driver version.

```
transactional update # zypper install -y --auto-agree-with-licenses \
  nvidia-open-driver-G06-signed-cuda-kmp-default
transactional update # version=$(rpm -qa --queryformat '%{VERSION}\n' \
  nvidia-open-driver-G06-signed-cuda-kmp-default \
  | cut -d "_" -f1 | sort -u | tail -n 1)
```

4. You can then install the appropriate packages for additional utilities that are useful for testing purposes.

```
transactional update # zypper install -y --auto-agree-with-licenses \
nvidia-compute-utils-G06=${version} \
nvidia-persistenced=${version}
```

5. Exit the transactional-update session and reboot to the new snapshot that contains the changes you have made.

```
transactional update # exit
# reboot
```

6. After the system has rebooted, log back in and use the `nvidia-smi` tool to verify that the driver is loaded successfully and that it can both access and enumerate your GPUs.

```
# nvidia-smi
```

The output of this command should show you something similar to the following output. In the example below, the system has one GPU.

```
Fri Aug  1 14:53:26 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 580.82.07     Driver Version: 580.82.07     CUDA Version: 13.0    |
|---------------------------------+--------------------+-----------------------+
| 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  Tesla T4              On   |00000000:00:1E.0 Off |                    0 |
| N/A   34C    P8     10W /   70W |    0MiB /  15360MiB |      0%      Default |
|                                 |                     |                  N/A |
+---------------------------------+---------------------+----------------------+


+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI         PID   Type   Process name                 GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

2.2.2.4 Validation of the driver installation

Running the `nvidia-smi` command has verified that, at the host level, the NVIDIA device can be accessed and that the drivers are loading successfully. To validate that it is functioning, you need to validate that the GPU can take instructions from a user-space application, ideally via a container and through the CUDA library, as that is typically what a real workload would use. For this, we can make a further modification to the host OS by installing `nvidia-container-toolkit`.

1. Open another transactional-update shell.

   ```
   {prompt_root} {tr-up-shell}
   ```

2. Install the `nvidia-container-toolkit` package from the NVIDIA Container Toolkit repository.

   ```
   transactional update # zypper ar \
   "https://nvidia.github.io/libnvidia-container/stable/rpm/"\
   nvidia-container-toolkit.repo
   transactional update # zypper --gpg-auto-import-keys install \
     -y nvidia-container-toolkit
   ```

   The `nvidia-container-toolkit.repo` file contains a stable repository `nvidia-container-toolkit` and an experimental repository `nvidia-container-toolkit-experimental`. Use the stable repository for production use. The experimental repository is disabled by default.

3. Exit the transactional-update session and reboot to the new snapshot that contains the changes you have made.

   ```
   transactional update # exit
   # reboot
   ```

4. Verify that the system can successfully enumerate the devices using the NVIDIA Container Toolkit. The output should be verbose, with INFO and WARN messages, but no ERROR messages.

   ```
   # nvidia-ctk cdi generate --output=/etc/cdi/nvidia.yaml
   ```

   This ensures that any container started on the machine can employ discovered NVIDIA GPU devices.

5. You can then run a Podman-based container. Doing this via `podman` gives you a good way of validating access to the NVIDIA device from within a container, which should give confidence for doing the same with Kubernetes at a later stage.
   Give Podman access to the labeled NVIDIA devices that were taken care of by the previous command and simply run the `bash` command.

```
# podman run --rm --device nvidia.com/gpu=all \
  --security-opt=label=disable \
  -it registry.suse.com/bci/bci-base:latest bash
```

   You can now execute commands from within a temporary Podman container. It does not have access to your underlying system and is **ephemeral**—whatever you change in the container does not persist. Also, you cannot break anything on the underlying host.

6. Inside the container, install the required CUDA libraries. Identify their version from the output of the `nvidia-smi` command. From the above example, we are installing CUDA version 13.0 with many examples, demos and development kits to fully validate the GPU.

```
# zypper ar \
  http://developer.download.nvidia.com/compute/cuda/repos/sles15/x86_64/ \
  cuda-sle15-sp6
# zypper --gpg-auto-import-keys refresh
# zypper install -y cuda-libraries-13-0 cuda-demo-suite-12-9
```

7. Inside the container, run the `deviceQuery` CUDA example of the same version, which comprehensively validates GPU access via CUDA and from within the container itself.

```
# /usr/local/cuda-12.9/extras/demo_suite/deviceQuery Starting...

 CUDA Device Query (Runtime API)

Detected 1 CUDA Capable device(s)

Device 0: "Tesla T4"
  CUDA Driver Version / Runtime Version          13.0 / 13.0
  CUDA Capability Major/Minor version number:    7.5
  Total amount of global memory:                 14914 MBytes (15638134784 bytes)
  (40) Multiprocessors, ( 64) CUDA Cores/MP:     2560 CUDA Cores
  GPU Max Clock rate:                            1590 MHz (1.59 GHz)
  Memory Clock rate:                             5001 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 4194304 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:  1024
   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 / 0 / 30
   Compute Mode:
      < Default (multiple host threads can use ::cudaSetDevice() with device
  simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 13.0, CUDA Runtime Version
 = 13.0, NumDevs = 1, Device0 = Tesla T4
Result = PASS
```

From inside the container, you can continue to run any other CUDA workload—such as compilers—to run further tests. When finished, you can exit the container.

```
# exit
```

Important

Changes you have made in the container and packages you have installed inside will be lost and will not impact the underlying operating system.

2.3 Installing SUSE Rancher Prime: RKE2

This guide will help you quickly launch a cluster with default options.

Tip

New to Kubernetes? The official Kubernetes docs already have great tutorials (https://kubernetes.io/docs/tutorials/kubernetes-basics/) ↗ outlining the basics.

Important

You can use any RKE2 Prime version listed on the Prime Artifacts URL for the assets mentioned in these steps. To learn more about the Prime Artifacts URL, see our Prime-only documentation (https://scc.suse.com/rancher-docs/rancherprime/latest/en/reference-guide.html#prime-artifacts-url) ↗. Authentication is required. Use your SUSE Customer Center (SCC) (https://scc.suse.com/home) ↗ credentials to log in.

2.3.1 Prerequisites

- Make sure your environment fulfills the requirements (https://documentation.suse.com/cloudnative/rke2/latest/en/install/requirements.html) ↗. If NetworkManager is installed and enabled on your hosts, ensure that it is configured to ignore CNI-managed interfaces (https://documentation.suse.com/cloudnative/rke2/latest/en/known_issues.html#_networkmanager) ↗.
- If the host kernel supports AppArmor (https://apparmor.net/) ↗, the AppArmor tools (usually available via the `apparmor-parser` package) must also be present before installing RKE2.
- The RKE2 installation process must be run as the root user or through `sudo`.

2.3.2 Server node installation

SUSE Rancher Prime: RKE2 provides an installation script that is a convenient way to install it as a service on systemd-based systems. This script is available at https://get.rke2.io. To install RKE2 using this method, do the following:

1. Run the installer, where `INSTALL_RKE2_ARTIFACT_URL` is the Prime Artifacts URL and `INSTALL_RKE2_CHANNEL` is a release channel you can subscribe to and defaults to `stable`. In this example, `INSTALL_RKE2_CHANNEL="latest"` gives you the latest version of RKE2.

   ```
   > sudocurl -sfL https://get.rke2.io/ | \
     sudo INSTALL_RKE2_ARTIFACT_URL=_PRIME-ARTIFACTS-URL_/rke2 \
     INSTALL_RKE2_CHANNEL="latest" sh -
   ```

   To specify a version, set the `INSTALL_RKE2_VERSION` environment variable.

   ```
   > sudocurl -sfL https://get.rke2.io/ | \
   sudo INSTALL_RKE2_ARTIFACT_URL=_PRIME-ARTIFACTS-URL_/rke2 \
     INSTALL_RKE2_VERSION="_VERSION_" ./install.sh
   ```

   This will install the `rke2-server` service and the `rke2` binary onto your machine. Due to its nature, it will fail unless it runs as the root user or through `sudo`.

2. Enable the `rke2-server` service.

   ```
   > sudosystemctl enable rke2-server.service
   ```

3. To pull images from the Rancher Prime registry, set the following value in `etc/rancher/rke2/config.yaml`:

   ```
   system-default-registry: registry.rancher.com
   ```

   This configuration tells RKE2 to use registry.rancher.com as the default location for all container images it needs to deploy within the cluster.

4. Start the service.

   ```
   > sudosystemctl start rke2-server.service
   ```

5. Follow the logs with the following command:

   ```
   > sudojournalctl -u rke2-server -f
   ```

After running this installation:

- The `rke2-server` service will be installed. The `rke2-server` service will be configured to automatically restart after node reboots or if the process crashes or is killed.
- Additional utilities will be installed at `/var/lib/rancher/rke2/bin/`. They include: `kubectl`, `crictl`, and `ctr`. Note that these are not on your path by default.
- Two cleanup scripts, `rke2-killall.sh` and `rke2-uninstall.sh`, will be installed to the path at:
  - `/usr/local/bin` for regular file systems
  - `/opt/rke2/bin` for read-only and Btrfs file systems
  - *`INSTALL_RKE2_TAR_PREFIX`*`/bin` if `INSTALL_RKE2_TAR_PREFIX` is set
- A kubeconfig (https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/) ↗ file will be written to `/etc/rancher/rke2/rke2.yaml`.
- A token that can be used to register other server or agent nodes will be created at `/var/lib/rancher/rke2/server/node-token`.

Note

If you are adding additional server nodes, you must have an odd number in total. An odd number is needed to maintain a quorum. See the High Availability documentation (https://documentation.suse.com/cloudnative/rke2/latest/en/install/ha.html) ↗ for more details.

2.3.3 Linux agent (worker) node installation

The steps on this section requires root-level access or `sudo` to work.

1. Run the installer.

   ```
   > sudocurl -sfL https://get.rke2.io | INSTALL_RKE2_TYPE="agent" sh -
   ```

   This will install the `rke2-agent` service and the `rke2` binary onto your machine. Due to its nature, it will fail unless it runs as the root user or through `sudo`.

2. Enable the `rke2-agent` service.

```
> sudosystemctl enable rke2-agent.service
```

3. Configure the `rke2-agent` service.

```
> sudomkdir -p /etc/rancher/rke2/
vim /etc/rancher/rke2/config.yaml
```

   Content for `config.yaml`:

```
> sudoserver: https://_SERVER_IP_OR_DNS_:9345
token: _TOKEN_FROM_SERVER_NODE_
```

   Note

   The `rke2 server` process listens on port `9345` for new nodes to register. The Kubernetes API is still served on port `6443`, as normal.

4. Start the service.

```
> sudosystemctl start rke2-agent.service
```

5. Follow the logs with the following command:

```
> sudojournalctl -u rke2-agent -f
```

Note

Each machine must have a unique host name. If your machines do not have unique host names, set the `node-name` parameter in the `config.yaml` file and provide a value with a valid and unique host name for each node. To learn more about the config.yaml file, refer to the Configuration options documentation (https://documentation.suse.com/cloud-native/rke2/latest/en/install/configuration.html#_configuration-file) ↗.

2.3.4 Microsoft Windows agent (worker) node installation

Windows Support works with Calico or Flannel as the CNI for the RKE2 cluster.

Note

The Windows Server Containers feature needs to be enabled for the RKE2 agent to work.

PREPARE THE MICROSOFT WINDOWS AGENT NODE

1. Open a new PowerShell window with administrator privileges.

   ```
   powershell -Command "Start-Process PowerShell -Verb RunAs"
   ```

2. In the new PowerShell window, run the following command to install the containers feature.

   ```
   Enable-WindowsOptionalFeature -Online -FeatureName containers –All
   ```

   This will require a reboot for the `Containers` feature to function properly.

3. Download the install script.

   ```
   Invoke-WebRequest -Uri https://raw.githubusercontent.com/rancher/rke2/master/
   install.ps1 -Outfile install.ps1
   ```

   This script will download the `rke2.exe` Windows binary onto your machine.

4. Configure the rke2-agent for Windows.

   ```
   > sudoNew-Item -Type Directory c:/etc/rancher/rke2 -Force
   Set-Content -Path c:/etc/rancher/rke2/config.yaml -Value @"
   server: https://_SERVER_IP_OR_DNS_:9345
   token: _TOKEN_FROM_SERVER_NODE_
   "@
   ```

   To learn more about the `config.yaml` file, refer to the Configuration options documentation (https://documentation.suse.com/cloudnative/rke2/latest/en/install/configuration.html#_configuration-file).

5. Configure the PATH.

   ```
   > sudo$env:PATH+=";c:\var\lib\rancher\rke2\bin;c:\usr\local\bin"

   [Environment]::SetEnvironmentVariable(
       "Path",
       [Environment]::GetEnvironmentVariable("Path",
    [EnvironmentVariableTarget]::Machine) + ";c:\var\lib\rancher\rke2\bin;c:\usr\local
   \bin",
   ```

```
    [EnvironmentVariableTarget]::Machine)
```

6. Run the installer.

```
> sudo./install.ps1
```

7. Start the Windows RKE2 Service.

```
> sudorke2.exe agent service --add
```

Note

Each machine must have a unique host name.

Do not forget to start the RKE2 service with:

```
Start-Service rke2
```

If you would prefer to use CLI parameters only instead, run the binary with the desired parameters.

```
rke2.exe agent --token _TOKEN_ --server _SERVER_URL_
```

3 Preparing the cluster for AI Library

This procedure assumes that you already have the base operating system installed on cluster nodes as well as the SUSE Rancher Prime: RKE2 Kubernetes distribution installed and operational. If you are installing from scratch, refer to *Section 2, "Installing the Linux and Kubernetes distribution"* first.

1. Install SUSE Rancher Prime (*Section 3.1, "Installing SUSE Rancher Prime on a Kubernetes cluster"*) on the cluster.

2. Install the NVIDIA GPU Operator on the cluster as described in *Section 3.2, "Installing the NVIDIA GPU Operator on the SUSE Rancher Prime: RKE2 cluster"*.

3. Connect the Kubernetes cluster to SUSE Rancher Prime as described in *Section 3.3, "Registering existing clusters"*.

4. Configure the GPU-enabled nodes so that the SUSE AI containers are assigned to Pods that run on nodes equipped with NVIDIA GPU hardware. Find more details about assigning Pods to nodes in *Section 3.4, "Assigning GPU nodes to applications"*.

5. _ *(Optional)*_ Install SUSE Security as described in *Section 3.5, "Installing SUSE Security"*. Although this step is not required, we strongly encourage it to ensure data security in the production environment.

6. Install and configure SUSE Observability to observe the nodes used for SUSE AI application. Refer to *Section 3.6, "Setting up SUSE Observability for SUSE AI"* for more details.

3.1 Installing SUSE Rancher Prime on a Kubernetes cluster

In this section, you will learn how to deploy SUSE Rancher Prime on a Kubernetes cluster using the Helm CLI.

3.1.1 Prerequisites

- Kubernetes cluster (*Section 3.1.1.1, "Kubernetes cluster"*)
- Ingress controller (*Section 3.1.1.2, "Ingress controller"*)
- CLI tools (*Section 3.1.1.3, "CLI tools"*)

3.1.1.1 Kubernetes cluster

Set up the SUSE Rancher Prime server's local Kubernetes cluster.

SUSE Rancher Prime can be installed on any Kubernetes cluster. This cluster can use upstream Kubernetes, or it can use one of SUSE Rancher Prime's Kubernetes distributions, or it can be a managed Kubernetes cluster from a provider such as Amazon EKS.

Tip

For help setting up a RKE2 cluster, refer to *Section 2.3, "Installing SUSE Rancher Prime: RKE2"*.

3.1.1.2 Ingress controller

The SUSE Rancher Prime UI and API are exposed through an Ingress. This means the Kubernetes cluster that you install SUSE Rancher Prime in must contain an Ingress controller.

For SUSE Rancher Prime: RKE2 and K3s installations, you do not have to install the Ingress controller manually because one is installed by default.

3.1.1.3 CLI tools

The following CLI tools are required for setting up the Kubernetes cluster. Make sure these tools are installed and available in your `$PATH`.

- `kubectl` (https://kubernetes.io/docs/tasks/tools/install-kubectl/#install-kubectl) - Kubernetes command-line tool.
- `helm` (https://docs.helm.sh/using_helm/#installing-helm) - Package management for Kubernetes. Refer to the Helm version requirements (https://docs.ranchermanager.rancher.io/getting-started/installation-and-upgrade/requirements/helm-version-requirements) to choose a version of Helm to install SUSE Rancher Prime. Refer to the instructions provided by the Helm project (https://helm.sh/docs/intro/install/) for your specific platform.

3.1.2 Install the Rancher Helm chart

SUSE Rancher Prime is installed using the Helm (https://helm.sh/) package manager for Kubernetes. Helm charts provide templating syntax for Kubernetes YAML manifest documents. With Helm, we can create configurable deployments instead of just using static files.

To choose a SUSE Rancher Prime version to install, refer to Choosing a SUSE Rancher Prime version (https://docs.ranchermanager.rancher.io/getting-started/installation-and-upgrade/resources/choose-a-rancher-version).

To choose a version of Helm to install SUSE Rancher Prime with, refer to the Helm version requirements (https://docs.ranchermanager.rancher.io/getting-started/installation-and-upgrade/requirements/helm-version-requirements).

Note

The installation instructions assume you are using Helm version 3.

To set up SUSE Rancher Prime,

1. Add the Helm chart repository (*Section 3.1.3, "Add the Helm chart repository"*)
2. Create a namespace for Rancher (*Section 3.1.4, "Create a namespace for Rancher"*)
3. Choose your SSL configuration (*Section 3.1.5, "Choose your SSL configuration"*)
4. Install cert-manager (*Section 3.1.6, "Install cert-manager"*)
5. Install Rancher with Helm and your chosen certificate option (*Section 3.1.7, "Install Rancher with Helm and your chosen certificate option"*)
6. Verify that the Rancher server is successfully deployed (*Section 3.1.8, "Verify that the Rancher server is successfully deployed"*)
7. Save your options (*Section 3.1.9, "Save your options"*)

3.1.3 Add the Helm chart repository

Use the `helm repo add` command to add the Helm chart repository that contains charts to install SUSE Rancher Prime.

```
> helm repo add rancher-prime <helm-chart-repo-url>
```

For information on the Helm chart repository URL, refer to the SUSE Rancher Prime documentation.

3.1.4 Create a namespace for Rancher

Define a Kubernetes namespace where the resources created by the chart will be installed named `cattle-system`:

```
> kubectl create namespace cattle-system
```

3.1.5 Choose your SSL configuration

The SUSE Rancher Prime management server is designed to be secure by default and requires SSL/TLS configuration.

Note

To externally cancel SSL/TLS, see TLS termination on an External Load Balancer (https://docs.ranchermanager.rancher.io/getting-started/installation-and-upgrade/references/helm-chart-options#external-tls-termination) ↗. As outlined on that page, this option does have additional requirements for TLS verification.

There are three recommended options for the source of the certificate used for TLS termination at the SUSE Rancher Prime server:

- **SUSE Rancher Prime-generated TLS certificate:** In this case, you need to install `cert-manager` into the cluster. SUSE Rancher Prime utilizes `cert-manager` to issue and maintain its certificates. SUSE Rancher Prime generates a CA certificate of its own, and sign a certificate using that CA. `cert-manager` is then responsible for managing that certificate. No extra action is needed when `agent-tls-mode` is set to strict. More information can be found on this setting in Agent TLS Enforcement (https://docs.ranchermanager.rancher.io/getting-started/installation-and-upgrade/references/tls-settings#agent-tls-enforcement) ↗.

- **Let's Encrypt:** The Let's Encrypt option also uses `cert-manager`. However, in this case, cert-manager is combined with a special Issuer for Let's Encrypt that performs all actions (including request and validation) necessary for getting a Let's Encrypt issued cert. This configuration uses HTTP validation (`HTTP-01`), so the load balancer must have a public DNS record and be accessible from the internet. When setting `agent-tls-mode` to `strict`, you must also specify `--privateCA=true` and upload the Let's Encrypt CA as described in Adding TLS Secrets (https://docs.ranchermanager.rancher.io/getting-started/installation-and-upgrade/resources/tls-secrets) ↗. Find more information on this setting in Agent TLS Enforcement (https://docs.ranchermanager.rancher.io/getting-started/installation-and-upgrade/references/tls-settings#agent-tls-enforcement) ↗.

- **Bring your own certificate:** This option allows you to bring your own public- or private-CA signed certificate. SUSE Rancher Prime will use that certificate to secure websocket and HTTPS traffic. In this case, you must upload this certificate (and associated key) as PEM-encoded files with the name `tls.crt` and `tls.key`. If you are using a private CA, you must also upload that certificate. This is because this private CA may not be trusted by your nodes. SUSE Rancher Prime will take that CA certificate, and generate a checksum from it, which the various SUSE Rancher Prime components use to validate their connection to SUSE Rancher Prime. If `agent-tls-mode` is set to `strict`, the

CA must be uploaded, so that downstream clusters can successfully connect. Find more information in Agent TLS Enforcement (https://docs.ranchermanager.rancher.io/getting-started/installation-and-upgrade/references/tls-settings#agent-tls-enforcement) ↗.

TABLE 1: SSL CONFIGURATION OPTIONS

Configuration	Helm Chart Option	Requires cert-manager
SUSE Rancher Prime Generated Certificates (Default)	`ingress.tls.source=rancher`	yes (*Section 3.1.6, "Install cert-manager"*)
Let's Encrypt	`ingress.tls.source=letsEncrypt`	*Section 3.1.6, "Install cert-manager"*
Certificates from Files	`ingress.tls.source=secret`	no

3.1.6 Install cert-manager

You should skip this step if you are bringing your own certificate files (option `ingress.tls.source=secret`), or if you use TLS termination on an external load balancer (https://docs.ranchermanager.rancher.io/getting-started/installation-and-upgrade/references/helm-chart-options#external-tls-termination) ↗.

This step is only required to use certificates issued by SUSE Rancher Prime's generated CA (`ingress.tls.source=rancher`) or to request Let's Encrypt issued certificates (`ingress.tls.source=letsEncrypt`).

Important

Recent changes to cert-manager require an upgrade. If you are upgrading SUSE Rancher Prime and using a version of cert-manager older than v0.11.0, please see our upgrade documentation (https://docs.ranchermanager.rancher.io/getting-started/installation-and-upgrade/resources/upgrade-cert-manager) ↗.

These instructions are adapted from the official cert-manager documentation (https://cert-manager.io/docs/installation/kubernetes/#installing-with-helm) ↗.

Note

To see options on how to customize the cert-manager install including for cases where your cluster uses PodSecurityPolicies, see the cert-manager docs (https://artifacthub.io/packages/helm/cert-manager/cert-manager#configuration).

```
# If you have installed the CRDs manually, instead of setting --set installCRDs=true \
  or --set crds.enabled=true in your Helm install command, \
  you should upgrade your CRD resources before upgrading the Helm chart:
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/
<VERSION>/cert-manager.crds.yaml

# Add the Jetstack Helm repository
helm repo add jetstack https://charts.jetstack.io

# Update your local Helm chart repository cache
helm repo update

# Install the cert-manager Helm chart
helm install cert-manager jetstack/cert-manager \
  --namespace cert-manager \
  --create-namespace \
  --set crds.enabled=true
```

Once you have installed cert-manager, you can verify it is deployed correctly by checking the cert-manager namespace for running pods:

```
kubectl get pods --namespace cert-manager

NAME                                       READY   STATUS    RESTARTS   AGE
cert-manager-5c6866597-zw7kh               1/1     Running   0          2m
cert-manager-cainjector-577f6d9fd7-tr77l   1/1     Running   0          2m
cert-manager-webhook-787858fcdb-nlzsq      1/1     Running   0          2m
```

3.1.7 Install Rancher with Helm and your chosen certificate option

The exact command to install SUSE Rancher Prime differs depending on the certificate configuration.

However, irrespective of the certificate configuration, the name of the SUSE Rancher Prime installation in the `cattle-system` namespace should always be `rancher`.

Tip

Testing and development. This final command to install SUSE Rancher Prime requires a domain name that forwards traffic to SUSE Rancher Prime. If you are using the Helm CLI to set up a proof-of-concept, you can use a fake domain name when passing the `hostname` option. An example of a fake domain name would be `<IP_OF_LINUX_NODE>.sslip.io`, which would expose SUSE Rancher Prime on an IP where it is running. Production installs would require a real domain name.

3.1.7.1 Rancher-generated certificates

The default is for SUSE Rancher Prime to generate a CA and uses `cert-manager` to issue the certificate for access to the SUSE Rancher Prime server interface.

Because `rancher` is the default option for `ingress.tls.source`, we are not specifying `ingress.tls.source` when running the `helm install` command.

- Set the `hostname` to the DNS name you pointed at your load balancer.
- Set the `bootstrapPassword` to something unique for the `admin` user.
- To install a specific SUSE Rancher Prime version, use the `--version` flag, example: `--version 2.7.0`

```
> helm install rancher rancher-prime/rancher \
  --namespace cattle-system \
  --set hostname=rancher.my.org \
  --set bootstrapPassword=admin
```

Wait for SUSE Rancher Prime to be rolled out:

```
> kubectl -n cattle-system rollout status deploy/rancher
Waiting for deployment "rancher" rollout to finish: 0 of 3 updated replicas are
 available...
deployment "rancher" successfully rolled out
```

3.1.7.2 Let's Encrypt

This option uses `cert-manager` to automatically request and renew Let's Encrypt certificates (https://letsencrypt.org/) ↗. This is a free service that provides you with a valid certificate as Let's Encrypt is a trusted CA.

Note

You need to have port 80 open as the HTTP-01 challenge can only be done on port 80.

In the following command,

- `hostname` is set to the public DNS record,
- Set the `bootstrapPassword` to something unique for the `admin` user.
- `ingress.tls.source` is set to `letsEncrypt`
- `letsEncrypt.email` is set to the email address used for communication about your certificate (for example, expiry notices)
- Set `letsEncrypt.ingress.class` to whatever your Ingress controller is, for example, `traefik`, `nginx`, `haproxy`, etc.

Important

When `agent-tls-mode` is set to `strict` (the default value for new installs of SUSE Rancher Prime starting from v2.9.0), you must supply the `privateCA=true` chart value (e.x. through `--set privateCA=true`) and upload the Let's Encrypt Certificate Authority as outlined in Adding TLS Secrets (https://docs.ranchermanager.rancher.io/getting-started/installation-and-upgrade/resources/tls-secrets). Information on identifying the Let's Encrypt Root CA can be found in the Let's Encrypt docs (https://letsencrypt.org/certificates/). If you do not upload the CA, then SUSE Rancher Prime may fail to connect to new or existing downstream clusters.

```
> helm install rancher rancher-prime/rancher \
  --namespace cattle-system \
  --set hostname=rancher.my.org \
  --set bootstrapPassword=admin \
  --set ingress.tls.source=letsEncrypt \
  --set letsEncrypt.email=me@example.org \
  --set letsEncrypt.ingress.class=nginx
```

Wait for SUSE Rancher Prime to be rolled out:

```
> kubectl -n cattle-system rollout status deploy/rancher
```

```
Waiting for deployment "rancher" rollout to finish: 0 of 3 updated replicas are
 available...
deployment "rancher" successfully rolled out
```

3.1.7.3 Certificates from files

In this option, Kubernetes secrets are created from your own certificates for SUSE Rancher Prime to use.

When you run this command, the `hostname` option must match the `Common Name` or a `Subject Alternative Names` entry in the server certificate or the Ingress controller will fail to configure correctly.

Although an entry in the `Subject Alternative Names` is technically required, having a matching `Common Name` maximizes compatibility with older browsers and applications.

Note

To check if your certificates are correct, see How do I check Common Name and Subject Alternative Names in my server certificate? (https://docs.ranchermanager.rancher.io/faq/technical-items#how-do-i-check-common-name-and-subject-alternative-names-in-my-server-certificate) ↗

- Set the `hostname`.
- Set the `bootstrapPassword` to something unique for the `admin` user.
- Set `ingress.tls.source` to `secret`.

```
> helm install rancher rancher-prime/rancher \
  --namespace cattle-system \
  --set hostname=rancher.my.org \
  --set bootstrapPassword=admin \
  --set ingress.tls.source=secret
```

If you are using a Private CA signed certificate , add `--set privateCA=true` to the command:

```
> helm install rancher rancher-prime/rancher \
  --namespace cattle-system \
  --set hostname=rancher.my.org \
  --set bootstrapPassword=admin \
  --set ingress.tls.source=secret \
  --set privateCA=true
```

Now that SUSE Rancher Prime is deployed, see Adding TLS Secrets to publish the certificate files so SUSE Rancher Prime and the Ingress controller can use them (https://docs.ranchermanager.rancher.io/getting-started/installation-and-upgrade/resources/tls-secrets) ↗.

3.1.7.4 Advanced options

The SUSE Rancher Prime chart configuration has many options for customizing the installation to suit your specific environment. Here are common advanced scenarios.

- HTTP Proxy (https://docs.ranchermanager.rancher.io/getting-started/installation-and-upgrade/references/helm-chart-options#http-proxy) ↗
- Private Container Image Registry (https://docs.ranchermanager.rancher.io/getting-started/installation-and-upgrade/references/helm-chart-options#private-registry-and-air-gap-installs) ↗
- TLS Termination on an External Load Balancer (https://docs.ranchermanager.rancher.io/getting-started/installation-and-upgrade/references/helm-chart-options#external-tls-termination) ↗

See the Chart Options (https://docs.ranchermanager.rancher.io/getting-started/installation-and-upgrade/references/helm-chart-options) ↗ for the full list of options.

3.1.8 Verify that the Rancher server is successfully deployed

After adding the secrets, check if SUSE Rancher Prime was rolled out successfully:

```
> kubectl -n cattle-system rollout status deploy/rancher
Waiting for deployment "rancher" rollout to finish: 0 of 3 updated replicas are
 available...
deployment "rancher" successfully rolled out
```

If you see the following error: `error: deployment "rancher" exceeded its progress deadline`, you can check the status of the deployment by running the following command:

```
> kubectl -n cattle-system get deploy rancher
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
rancher   3         3         3            3           3m
```

It should show the same count for `DESIRED` and `AVAILABLE`.

3.1.9 Save your options

Make sure you save the `--set` options you used. You will need to use the same options when you upgrade SUSE Rancher Prime to new versions with Helm.

3.1.10 Finishing up

Now you should have a functional SUSE Rancher Prime server.

In a Web browser, go to the DNS name that forwards traffic to your load balancer. Then you should be greeted by the colorful login page.

3.2 Installing the NVIDIA GPU Operator on the SUSE Rancher Prime: RKE2 cluster

The NVIDIA operator allows administrators of Kubernetes clusters to manage GPUs just like CPUs. It includes everything needed for pods to be able to operate GPUs.

3.2.1 Host OS requirements

To expose the GPU to the pod correctly, the NVIDIA kernel drivers and the `libnvidia-ml` library must be correctly installed in the host OS. The NVIDIA Operator can automatically install drivers and libraries on specific operating systems. Check the NVIDIA documentation for information on supported operating system releases (https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/platform-support.html#supported-operating-systems-and-kubernetes-platforms). Installation of the NVIDIA components on your host OS is out of the scope of this document. Refer to the NVIDIA documentation for instructions.

The following three commands should return a correct output if the kernel driver is correctly installed.

1. `lsmod | grep nvidia` returns a list of NVIDIA kernel modules. For example:

   ```
   nvidia_uvm             2129920  0
   nvidia_drm              131072  0
   ```

```
nvidia_modeset       1572864  1 nvidia_drm
video                  77824  1 nvidia_modeset
nvidia               9965568  2 nvidia_uvm,nvidia_modeset
ecc                    45056  1 nvidia
```

2. `cat /proc/driver/nvidia/version` returns the NVRM and GCC versions of the driver. For example:

   ```
   NVRM version: NVIDIA UNIX Open Kernel Module for x86_64  555.42.06
     Release Build  (abuild@host)  Thu Jul 11 12:00:00 UTC 2024
     GCC version:  gcc version 7.5.0 (SUSE Linux)
   ```

3. `find /usr/ -iname libnvidia-ml.so` returns a path to the `libnvidia-ml.so` library. For example:

   ```
   /usr/lib64/libnvidia-ml.so
   ```

   This library is used by Kubernetes components to interact with the kernel driver.

3.2.2 Operator installation

Once the OS is ready and RKE2 is running, adjust the RKE2 nodes:

1. On the agent nodes of RKE2, run the following command:

   ```
   # echo PATH=$PATH:/usr/local/nvidia/toolkit >> /etc/default/rke2-agent
   ```

2. On the server nodes of RKE2, run the following command:

   ```
   # echo PATH=$PATH:/usr/local/nvidia/toolkit >> /etc/default/rke2-server
   ```

Then, install the NVIDIA GPU Operator using the following YAML manifest.

```
apiVersion: helm.cattle.io/v1
kind: HelmChart
metadata:
  name: gpu-operator
  namespace: kube-system
spec:
  repo: https://helm.ngc.nvidia.com/nvidia
  chart: gpu-operator
  targetNamespace: gpu-operator
  createNamespace: true
```

```
valuesContent: |-
  toolkit:
    env:
    - name: CONTAINERD_SOCKET
      value: /run/k3s/containerd/containerd.sock
```

Warning

The NVIDIA operator restarts containerd with a hangup call, which restarts RKE2.

After approximately one minute, you can make the following checks to verify that everything works as expected.

1. Assuming the drivers and `libnvidia-ml.so` library are installed, check if the operator detects them correctly.

   ```
   > kubectl get node <NODENAME> \
     -o jsonpath='{.metadata.labels}' | grep "nvidia.com/gpu.deploy.driver"
   ```

   You should see the value `pre-installed`. If you see `true`, the drivers are not installed correctly. If the requirements in *Section 3.2.1, "Host OS requirements"* are met, you may have forgotten to reboot the node after installing all packages.
   You can also check other driver labels:

   ```
   > kubectl get node <NODENAME> \
     -o jsonpath='{.metadata.labels}' | jq | grep "nvidia.com"
   ```

   You should see labels specifying driver and GPU, for example, `nvidia.com/gpu.machine` or `nvidia.com/cuda.driver.major`.

2. Check if the GPU was added by `nvidia-device-plugin-daemonset` as an allocatable resource in the node.

   ```
   > kubectl get node <NODENAME> \
     -o jsonpath='{.status.allocatable}' | jq
   ```

   You should see `"nvidia.com/gpu":` followed by the number of GPUs in the node.

3. Check that the container runtime binary was installed by the operator (in particular, by the `nvidia-container-toolkit-daemonset`):

   ```
   > ls /usr/local/nvidia/toolkit/nvidia-container-runtime
   ```

4. Verify whether the containerd configuration was updated to include the NVIDIA container runtime.

```
> grep nvidia /var/lib/rancher/rke2/agent/etc/containerd/config.toml
```

5. Run a pod to verify that the GPU resource can successfully be scheduled on a pod and the pod can detect it.

```
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: compute,utility
```

Note: Version gate

Available as of October 2024 releases: v1.28.15+rke2r1, v1.29.10+rke2r1, v1.30.6+rke2r1, v1.31.2+rke2r1.

RKE2 will now use `PATH` to find alternative container runtimes, in addition to checking the default paths used by the container runtime packages. To use this feature, you must modify the RKE2 service's `PATH` environment variable to add the directories containing the container runtime binaries.

We recommend modifying one of these two environment files:

- `/etc/default/rke2-server` # or rke2-agent
- `/etc/sysconfig/rke2-server` # or rke2-agent

This example adds the `PATH` in `/etc/default/rke2-server`:

```
> echo PATH=$PATH >> /etc/default/rke2-server
```

Warning

`PATH` changes should be done with care to avoid placing untrusted binaries in the path of services that run as root.

3.3 Registering existing clusters

In this section, you will learn how to register existing RKE2 clusters in SUSE Rancher Prime (Rancher).

The cluster registration feature replaced the feature for importing clusters.

The control that Rancher has to manage a registered cluster depends on the type of cluster. For details, see *Section 3.3.3, "Management capabilities for registered clusters"*.

3.3.1 Prerequisites

3.3.1.1 Kubernetes node roles

Registered RKE Kubernetes clusters must have all three node roles: `etcd`, `controlplane` and `worker`. A cluster with only `controlplane` components cannot be registered in Rancher.

For more information on RKE node roles, see the best practices (https://ranchermanager.docs.rancher.com/getting-started/installation-and-upgrade/installation-requirements/production-checklist#cluster-architecture).

3.3.1.2 Permissions

To register a cluster in Rancher, you must have `cluster-admin` privileges within that cluster. If you do not, grant these privileges to your user by running:

```
> kubectl create clusterrolebinding cluster-admin-binding \
  --clusterrole cluster-admin \
--user <USER_ACCOUNT>
```

3.3.2 Registering a cluster

1. Click # > *Cluster Management*.
2. On the *Clusters* page, click *Import Existing*.
3. Choose the type of cluster.
4. Use *Member Roles* to configure user authorization for the cluster. Click *Add Member* to add users who can access the cluster. Use the *Role* drop-down list to set permissions for each user.
5. If you are importing a generic Kubernetes cluster in Rancher, perform the following steps for setup:
   a. Click *Agent Environment Variables* under *Cluster Options* to set environment variables for the Rancher cluster agent (https://ranchermanager.docs.rancher.com/getting-started/installation-and-upgrade/advanced-options/about-rancher-agents). The environment variables can be set using key-value pairs. If the Rancher agent requires the use of a proxy to communicate with the Rancher server, `HTTP_PROXY`, `HTTP_PROXY`, `HTTPS_PROXY` and `NO_PROXY` environment variables can be set using agent environment variables.
   b. Enable *Project Network Isolation* to ensure the cluster supports Kubernetes `NetworkPolicy` resources. Users can select the *Project Network Isolation* option under the *Advanced Options* drop-down list to do so.
   c. Configure the version management feature for imported RKE2 and K3s clusters. (*Section 3.3.4, "Configuring version management for RKE2 and SUSE Rancher Prime: K3s clusters"*)
6. Click *Create*.
7. The requirements for `cluster-admin` privileges are shown (see *Section 3.3.1, "Prerequisites"*), including an example command to fulfill them.
8. Copy the `kubectl` command to your clipboard and run it on a node where `kubeconfig` is configured to point to the cluster you want to import. If you are unsure it is configured correctly, run `kubectl get nodes` to verify before running the command shown in Rancher.

9. If you are using self-signed certificates, you will receive the message `certificate signed by unknown authority`. To work around this validation, copy the command starting with `curl` displayed in Rancher to your clipboard. Then run the command on a node where `kubeconfig` is configured to point to the cluster you want to import.

10. After you finish running the command(s) on your node, click *Done*.

Important

The `NO_PROXY` environment variable is not standardized, and the accepted format of the value can differ between applications. When configuring the `NO_PROXY` variable for Rancher, the value must adhere to the format expected by Golang.

Specifically, the value should be a comma-delimited string that contains only IP addresses, CIDR notation, domain names or special DNS labels (such as `*`). For a full description of the expected value format, refer to the upstream Golang documentation (https://pkg.go.dev/golang.org/x/net/http/httpproxy#Config) ↗.

Note: Expected results

- Your cluster is registered and assigned a state of `Pending`. Rancher is deploying resources to manage your cluster.
- You can access your cluster after its state is updated to `Active`.
- `Active` clusters are assigned two projects: `Default` (containing the namespace `default`) and `System` (containing the namespaces `cattle-system`, `ingress-nginx`, `kube-public` and `kube-system`, if present).

Note

You cannot re-register a cluster that is currently active in a Rancher setup.

3.3.3 Management capabilities for registered clusters

The control that Rancher has to manage a registered cluster depends on the type of cluster.

- Features for all registered clusters (*Section 3.3.3.1, “Features for all registered clusters”*)
- Additional features for registered RKE2 and SUSE Rancher Prime: K3s clusters (*Section 3.3.3.2, “Additional features for registered RKE2 and SUSE Rancher Prime: K3s clusters”*)

3.3.3.1 Features for all registered clusters

After registering a cluster, the cluster owner can:

- Manage cluster access (https://ranchermanager.docs.rancher.com/how-to-guides/new-user-guides/authentication-permissions-and-global-configuration/manage-role-based-access-control-rbac/cluster-and-project-roles) ↗ through role-based access control
- Enable monitoring, alerts and notifiers (https://ranchermanager.docs.rancher.com/how-to-guides/new-user-guides/monitoring-alerting-and-logging/rancher-ui-monitoring) ↗
- Enable logging (https://ranchermanager.docs.rancher.com/how-to-guides/new-user-guides/monitoring-alerting-and-logging/logging) ↗
- Enable Istio (https://ranchermanager.docs.rancher.com/how-to-guides/new-user-guides/istio-setup-guide) ↗
- Manage projects and workloads

3.3.3.2 Additional features for registered RKE2 and SUSE Rancher Prime: K3s clusters

SUSE Rancher Prime: K3s (https://documentation.suse.com/cloudnative/k3s/latest/en/introduction.html) ↗ is a lightweight, fully compliant Kubernetes distribution for edge installations.

RKE2 (https://documentation.suse.com/cloudnative/rke2/latest/en/introduction.html) ↗ is Rancher’s next-generation Kubernetes distribution for data center and cloud installations.

When an RKE2 or SUSE Rancher Prime: K3s cluster is registered in Rancher, Rancher will recognize it. The Rancher UI will expose features available to all registered clusters (*Section 3.3.3.1, "Features for all registered clusters"*), along with the following options for editing and upgrading the cluster:

- Enable or disable version management (*Section 3.3.4, "Configuring version management for RKE2 and SUSE Rancher Prime: K3s clusters"*).
- Upgrade the Kubernetes version (https://ranchermanager.docs.rancher.com/how-to-guides/new-user-guides/manage-clusters/back-up-restore-and-disaster-recovery/back-up-rancher-launched-kubernetes-clusters) ↗ when version management is enabled.
- Configure the upgrade strategy (*Section 3.3.5, "Configuring RKE2 and SUSE Rancher Prime: K3s cluster upgrades"*).
- View a read-only version of the cluster's configuration arguments and environment variables used to launch each node.

3.3.4 Configuring version management for RKE2 and SUSE Rancher Prime: K3s clusters

Warning

When version management is enabled for an imported cluster, upgrading it outside of Rancher may lead to unexpected consequences.

The version management feature for imported RKE2 and SUSE Rancher Prime: K3s clusters can be configured using one of the following options:

- **Global default** (default): Inherits behavior from the global `imported-cluster-version-management` setting.
- **True**: Enables version management, allowing users to control the Kubernetes version and upgrade strategy of the cluster through Rancher.
- **False**: Disables version management, enabling users to manage the cluster's Kubernetes version independently, outside of Rancher.

You can define the default behavior for newly created clusters or existing ones set to 'Global default' by modifying the `imported-cluster-version-management` setting.

Changes to the global `imported-cluster-version-management` setting take effect during the cluster's next reconciliation cycle.

Note

If version management is enabled for a cluster, Rancher will deploy the `system-upgrade-controller` app, along with the associated plans and other required Kubernetes resources, to the cluster. If version management is disabled, Rancher will remove these components from the cluster.

3.3.5 Configuring RKE2 and SUSE Rancher Prime: K3s cluster upgrades

Tip

It is a Kubernetes best practice to back up the cluster before upgrading. When upgrading a high-availability SUSE Rancher Prime: K3s cluster with an external database, back up the database in whichever way is recommended by the relational database provider.

The **concurrency** is the maximum number of nodes that are permitted to be unavailable during an upgrade. If the number of unavailable nodes is larger than the **concurrency**, the upgrade will fail. If an upgrade fails, you may need to repair or remove failed nodes before the upgrade can succeed.

- **Control plane concurrency:** the maximum number of server nodes to upgrade at a single time; also the maximum unavailable server nodes
- **Worker concurrency:** the maximum number of worker nodes to upgrade at the same time; also the maximum unavailable worker nodes

In the RKE2 and SUSE Rancher Prime: K3s documentation, control plane nodes are called server nodes. These nodes run the Kubernetes master, which maintains the desired state of the cluster. By default, these control plane nodes can have workloads scheduled to them by default.

Also in the RKE2 and SUSE Rancher Prime: K3s documentation, nodes with the worker role are called agent nodes. Any workloads or pods that are deployed in the cluster can be scheduled to these nodes by default.

3.3.6 Debug logging and troubleshooting for registered RKE2 and SUSE Rancher Prime: K3s clusters

Nodes are upgraded by the system upgrade controller running in the downstream cluster. Based on the cluster configuration, Rancher deploys two plans (https://github.com/rancher/system-upgrade-controller#example-upgrade-plan) ↗ to upgrade nodes: one for control plane nodes and one for workers. The system upgrade controller follows the plans and upgrades the nodes.

To enable debug logging on the system upgrade controller deployment, edit the configmap (https://github.com/rancher/system-upgrade-controller/blob/50a4c8975543d75f1d76a8290001d87dc298bdb4/manifests/system-upgrade-controller.yaml#L32) ↗ to set the debug environment variable to true. Then restart the `system-upgrade-controller` pod.

Logs created by the `system-upgrade-controller` can be viewed by running this command:

```
> kubectl logs -n cattle-system system-upgrade-controller
```

The current status of the plans can be viewed with this command:

```
> kubectl get plans -A -o yaml
```

Tip

If the cluster becomes stuck during upgrading, restart the `system-upgrade-controller`.

To prevent issues when upgrading, the Kubernetes upgrade best practices (https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade/) ↗ should be followed.

3.3.7 Authorized cluster endpoint support for RKE2 and SUSE Rancher Prime: K3s clusters

Rancher supports Authorized Cluster Endpoints (ACE) for registered RKE2 and SUSE Rancher Prime: K3s clusters. This support includes manual steps you will perform on the downstream cluster to enable the ACE. For additional information on the authorized cluster endpoint, refer to How the Authorized Cluster Endpoint Works (https://ranchermanager.docs.rancher.com/how-to-guides/new-user-guides/manage-clusters/access-clusters/authorized-cluster-endpoint) ↗.

Note: Notes

- These steps only need to be performed on the control plane nodes of the downstream cluster. You must configure each control plane node individually.
- The following steps will work on both RKE2 and SUSE Rancher Prime: K3s clusters registered in v2.6.x as well as those registered (or imported) from a previous version of Rancher with an upgrade to v2.6.x.
- These steps will alter the configuration of the downstream RKE2 and SUSE Rancher Prime: K3s clusters and deploy the `kube-api-authn-webhook`. If a future implementation of the ACE requires an update to the `kube-api-authn-webhook`, then this would also have to be done manually. For more information on this webhook, see Authentication webhook documentation (https://ranchermanager.docs.rancher.com/how-to-guides/new-user-guides/manage-clusters/access-clusters/authorized-cluster-endpoint#about-the-kube-api-auth-authentication-webhook) ↗.

MANUAL STEPS TO BE TAKEN ON THE CONTROL PLANE OF EACH DOWNSTREAM CLUSTER TO ENABLE ACE

1. Create a file at `/var/lib/rancher/{rke2,k3s}/kube-api-authn-webhook.yaml` with the following contents:

```
apiVersion: v1
kind: Config
clusters:
- name: Default
  cluster:
    insecure-skip-tls-verify: true
    server: http://127.0.0.1:6440/v1/authenticate
users:
- name: Default
  user:
    insecure-skip-tls-verify: true
current-context: webhook
contexts:
- name: webhook
  context:
    user: Default
    cluster: Default
```

2. Add the following to the configuration file (or create one if it does not exist). Note that the default location is `/etc/rancher/{rke2,k3s}/config.yaml`:

```
kube-apiserver-arg:
  - authentication-token-webhook-config-file=/var/lib/rancher/{rke2,k3s}/kube-api-
authn-webhook.yaml
```

3. Run the following commands:

```
> sudo systemctl stop {rke2,k3s}-server
> sudo systemctl start {rke2,k3s}-server
```

4. Finally, you **must** go back to the Rancher UI and edit the imported cluster there to complete the ACE enablement. Click on # > *Edit Config*, then click the *Networking* tab under *Cluster Configuration*. Finally, click the *Enabled* button for *Authorized Endpoint*. Once the ACE is enabled, you then have the option of entering a fully qualified domain name (FQDN) and certificate information.

Note

The *FQDN* field is optional, and if one is entered, it should point to the downstream cluster. Certificate information is only needed if there is a load balancer in front of the downstream cluster that is using an untrusted certificate. If you have a valid certificate, then nothing needs to be added to the *CA Certificates* field.

3.3.8 Annotating registered clusters

For all types of registered Kubernetes clusters except for RKE2 and SUSE Rancher Prime: K3s Kubernetes clusters, Rancher does not have any information about how the cluster is provisioned or configured.

Therefore, when Rancher registers a cluster, it assumes that several capabilities are disabled by default. Rancher assumes this to avoid exposing UI options to the user even when the capabilities are not enabled in the registered cluster.

However, if the cluster has a certain capability, a user of that cluster might still want to select the capability for the cluster in the Rancher UI. To do that, the user will need to manually indicate to Rancher that certain capabilities are enabled for the cluster.

By annotating a registered cluster, it is possible to indicate to Rancher that a cluster was given additional capabilities outside of Rancher.

The following annotation indicates Ingress capabilities. Note that the values of non-primitive objects need to be JSON-encoded, with quotations escaped.

```
"capabilities.cattle.io/ingressCapabilities": "[
  {
    \"customDefaultBackend\":true,
    \"ingressProvider\":\"asdf\"
  }
]"
```

These capabilities can be annotated for the cluster:

- `ingressCapabilities`
- `loadBalancerCapabilities`
- `nodePoolScalingSupported`
- `nodePortRange`
- `taintSupport`

All the capabilities and their type definitions can be viewed in the Rancher API view, at `<RANCHER_SERVER_URL>/v3/schemas/capabilities`.

To annotate a registered cluster,

1. Click # > *Cluster Management*.
2. On the *Clusters* page, go to the custom cluster you want to annotate and click # > *Edit Config*.
3. Expand the *Labels & Annotations* section.
4. Click *Add Annotation*.
5. Add an annotation to the cluster with the format `capabilities/<capability>: <value>` where `value` is the cluster capability that will be overridden by the annotation. In this scenario, Rancher is not aware of any capabilities of the cluster until you add the annotation.
6. Click *Save*.

Tip

The annotation does not give the capabilities to the cluster, but it does indicate to Rancher that the cluster has those capabilities.

3.4 Assigning GPU nodes to applications

When deploying a containerized application to Kubernetes, you need to ensure that containers requiring GPU resources are run on appropriate worker nodes. For example, Ollama, a core component of SUSE AI, can deeply benefit from the use of GPU acceleration. This topic describes how to satisfy this requirement by explicitly requesting GPU resources and labeling worker nodes for configuring the node selector.

REQUIREMENTS

- Kubernetes cluster—such as SUSE Rancher Prime: RKE2—must be available and configured with more than one worker node in which certain nodes have NVIDIA GPU resources and others do not.
- This document assumes that any kind of deployment to the Kubernetes cluster is done using Helm charts.

3.4.1 Labeling GPU nodes

To distinguish nodes with the GPU support from non-GPU nodes, Kubernetes uses **labels**. Labels are used for relevant metadata and should not be confused with annotations that provide simple information about a resource. It is possible to manipulate labels with the `kubectl` command, as well as by tweaking configuration files from the nodes. If an IaC tool such as Terraform is used, labels can be inserted in the node resource configuration files.

To label a single node, use the following command:

```
> kubectl label node <GPU_NODE_NAME> accelerator=nvidia-gpu
```

To achieve the same result by tweaking the `node.yaml` node configuration, add the following content and apply the changes with `kubectl apply -f node.yaml`:

```
apiVersion: v1
kind: Node
metadata:
  name: node-name
  labels:
    accelerator: nvidia-gpu
```

Tip: Labeling multiple nodes

To label multiple nodes, use the following command:

```
> kubectl label node \
  <GPU_NODE_NAME1> \
  <GPU_NODE_NAME2> ... \
  accelerator=nvidia-gpu
```

Tip

If Terraform is being used as an IaC tool, you can add labels to a group of nodes by editing the `.tf` files and adding the following values to a resource:

```
resource "node_group" "example" {
  labels = {
    "accelerator" = "nvidia-gpu"
  }
}
```

To check if the labels are correctly applied, use the following command:

```
> kubectl get nodes --show-labels
```

3.4.2 Assigning GPU nodes

The matching between a container and a node is configured by the explicit resource allocation and the use of labels and node selectors. The use cases described below focus on NVIDIA GPUs.

3.4.2.1 Enable GPU passthrough

Containers are isolated from the host environment by default. For the containers that rely on the allocation of GPU resources, their Helm charts must enable GPU passthrough so that the container can access and use the GPU resource. Without enabling the GPU passthrough, the container may still run, but it can only use the main CPU for all computations. Refer to Ollama Helm chart (https://documentation.suse.com/suse-ai/1.0/html/AI-deployment-intro/index.html#ollama-helmchart) ↗ for an example of the configuration required for GPU acceleration.

3.4.2.2 Assignment by resource request

After the NVIDIA GPU Operator is configured on a node, you can instantiate applications requesting the resource `nvidia.com/gpu` provided by the operator. Add the following content to your `values.yaml` file. Specify the number of GPUs according to your setup.

```
resources:
  requests:
    nvidia.com/gpu: 1
  limits:
    nvidia.com/gpu: 1
```

3.4.2.3 Assignment by labels and node selectors

If affected cluster nodes are labeled with a label such as `accelerator=nvidia-gpu`, you can configure the node selector to check for the label. In this case, use the following values in your `values.yaml` file.

```
nodeSelector:
  accelerator: nvidia-gpu
```

3.4.3 Verifying Ollama GPU assignment

If the GPU is correctly detected, the Ollama container logs this event:

```
| [...] source=routes.go:1172 msg="Listening on :11434 (version 0.0.0)"
                            |
| [...] source=payload.go:30 msg="extracting embedded files" dir=/tmp/ollama2502346830/
runners                       |
| [...] source=payload.go:44 msg="Dynamic LLM libraries [cuda_v12 cpu cpu_avx cpu_avx2]"
                            |
| [...] source=gpu.go:204 msg="looking for compatible GPUs"
                            |
| [...] source=types.go:105 msg="inference compute" id=GPU-c9ad37d0-d304-5d2a-c2e6-
d3788cd733a7 library=cuda compute |
```

3.5 Installing SUSE Security

This chapter describes how to install SUSE Security to scan SUSE AI nodes for vulnerabilities and improve data protection. You can install it either using SUSE Rancher Prime (*Section 3.5.1, “Installing and managing SUSE Security through Rancher Extensions or Apps & Marketplace”*) or on any Kubernetes cluster (*Section 3.5.2, “Installing SUSE Security using Kubernetes”*).

3.5.1 Installing and managing SUSE Security through Rancher Extensions or Apps & Marketplace

SUSE Security can be deployed easily either through Rancher Extensions for Prime customers, or Rancher Apps and Marketplace. The default (Helm-based) deployment deploys SUSE Security containers into the `cattle-neuvector-system` namespace.

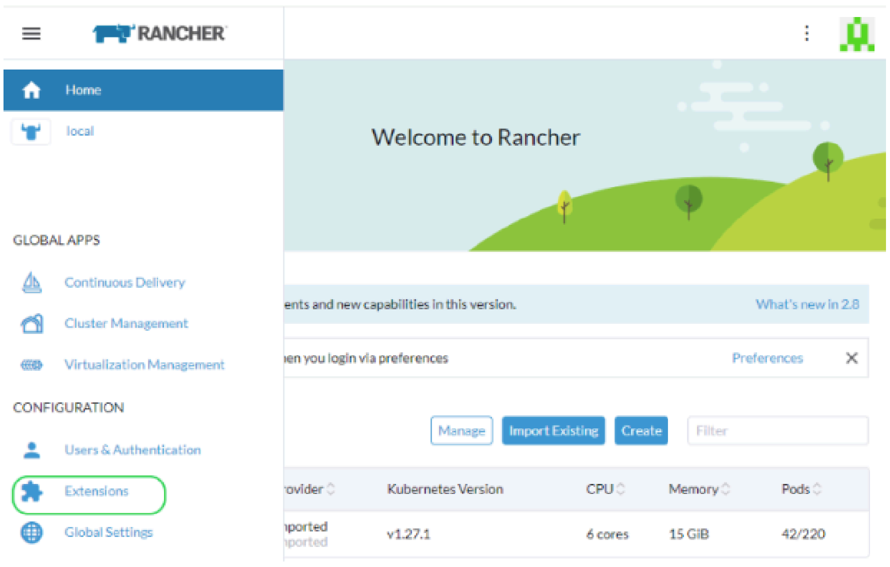
Note

Only SUSE Security deployments through Rancher Extensions (SUSE Security) of Rancher version 2.7.0+, or Apps & Marketplace of Rancher version 2.6.5+ can be managed directly (single sign-on to the SUSE Security console) through Rancher. If adding clusters to Rancher with SUSE Security already deployed, or where SUSE Security has been deployed directly onto the cluster, these clusters will not be enabled for SSO integration.

3.5.1.1 SUSE Security UI extension for Rancher

SUSE Rancher Prime customers are able to easily deploy SUSE Security and the SUSE Security UI Extension for Rancher. This will enable Prime users to monitor and manage certain SUSE Security functions and events directly through the Rancher UI. For community users, please see the Deploy SUSE Security section below to deploy from Rancher Apps and Marketplace.

1. The first step is to enable the Rancher Extensions capability globally if it is not already enabled.

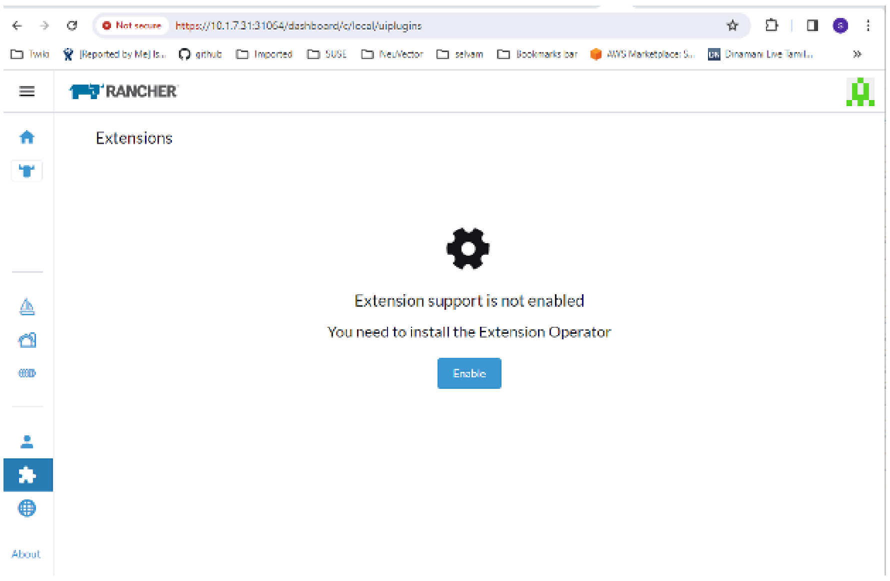


ents and new capabilities in this version.

en you login via preferences

rovider	Kubernetes Version	CPU	Memory	Pods
nported nported	v1.27.1	6 cores	15 GiB	42/220

FIGURE 16: RANCHER EXTENSIONS



Extension support is not enabled

You need to install the Extension Operator

FIGURE 17: ENABLE EXTENSIONS

2. Install the SUSE Security-UI-Ext from the *Available* list.

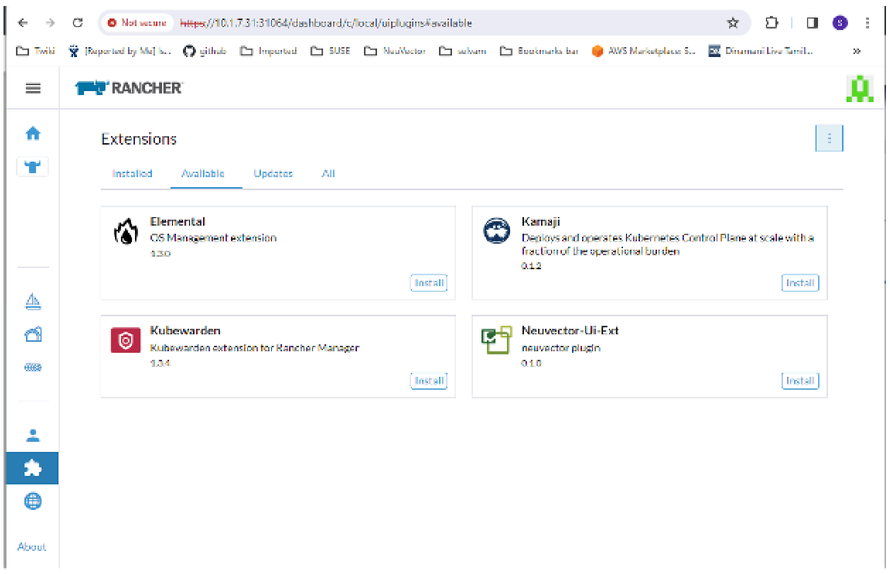


FIGURE 18: INSTALL UI EXTENSION

3. Reload the extension after installation is complete.

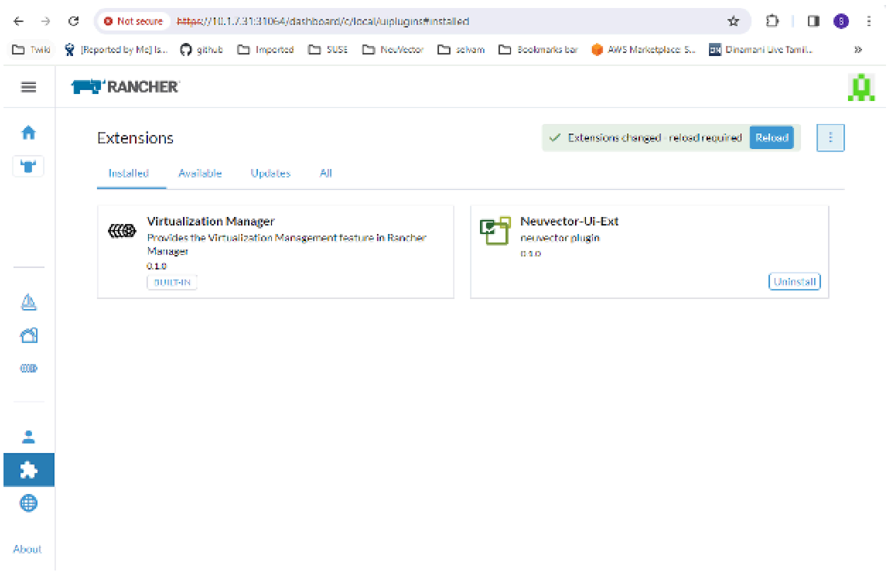
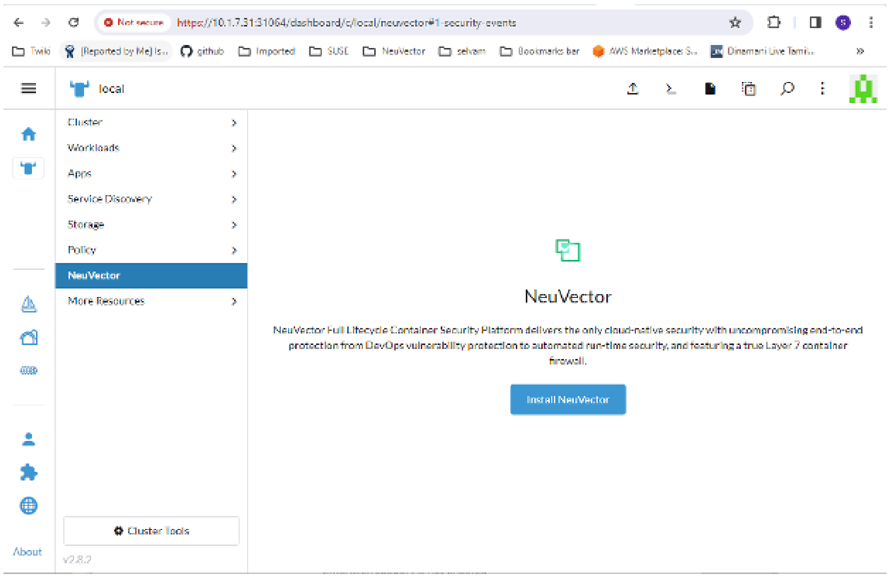


FIGURE 19: RELOAD EXTENSION

4. On your selected cluster, install the SUSE Security application from the SUSE Security tab if the SUSE Security app is not already installed. This should take you to the application installation steps. For more details on this installation process, see *Section 3.5.1.2, “Deploy SUSE Security”*.



NeuVector Full Lifecycle Container Security Platform delivers the only cloud-native security with uncompromising end-to-end protection from DevOps vulnerability protection to automated run-time security, and featuring a true Layer 7 container firewall.

FIGURE 20: INSTALL SUSE SECURITY APPLICATION

5. The SUSE Security dashboard should now be shown from the SUSE Security menu for that cluster. From this dashboard, the security health of the cluster can be monitored. There are interactive elements in the dashboard, such as invoking a wizard to improve your Security Risk Score, including being able to turn on automated scanning for vulnerabilities if it is not enabled.

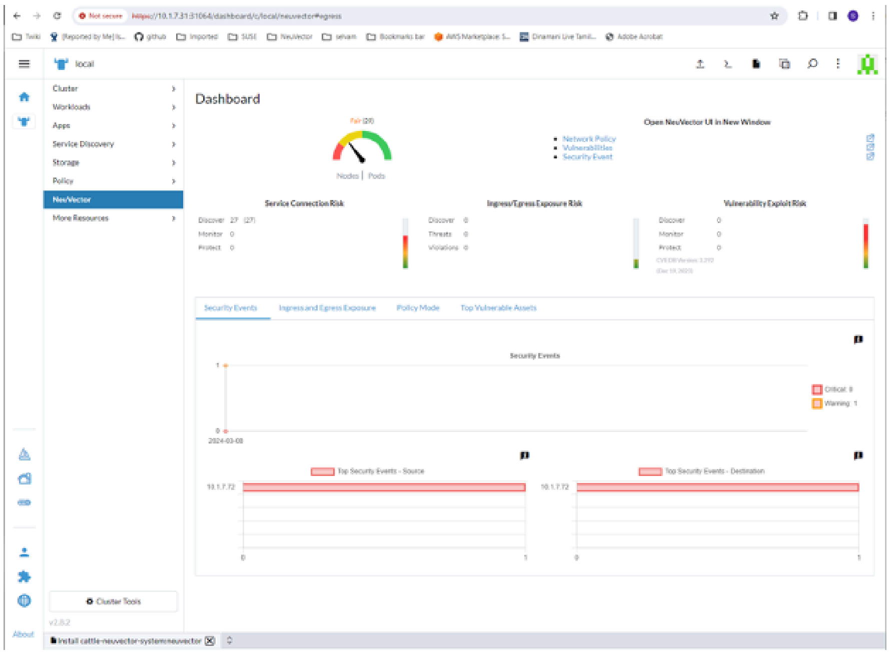

https://10.1.7.31:31064/dashboard/c/local/neuvector#egress
local
Cluster
Workloads
Apps
Service Discovery
Storage
Policy
NeuVector
More Resources
Cluster Tools
Dashboard
Open NeuVector UI in New Window
Network Policy
Vulnerabilities
Security Event
Service Connection Risk
Ingress/Egress Exposure Risk
Vulnerability Exploit Risk
Security Events
Ingress and Egress Exposure
Policy Mode
Top Vulnerable Assets
Top Security Events - Source
Top Security Events - Destination
Install cattle-neuvector-system:neuvector

FIGURE 21: SUSE SECURITY DASBOARD

The links in the upper right of the dashboard provide convenient single sign-on (SSO) links to the full SUSE Security console for more detailed analysis and configuration.

6. To uninstall the extension, go back to the Extensions page.

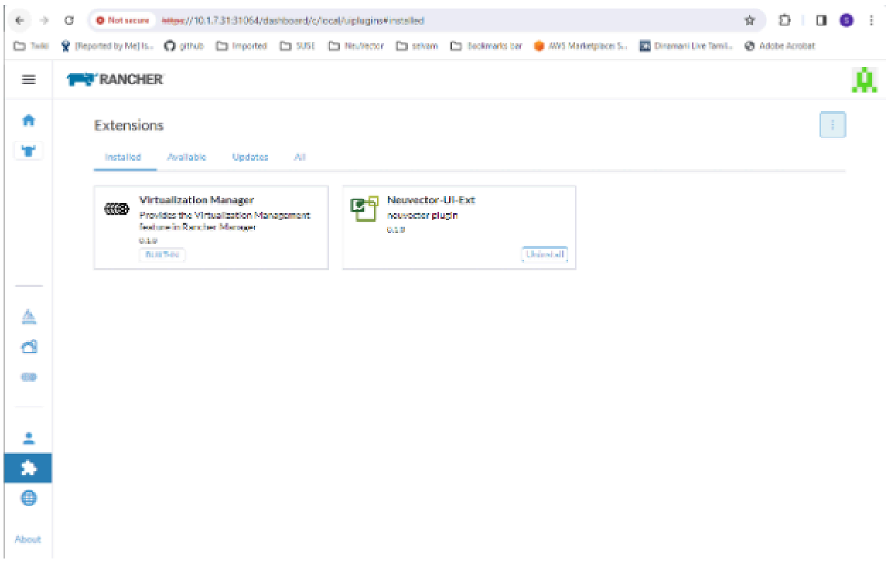


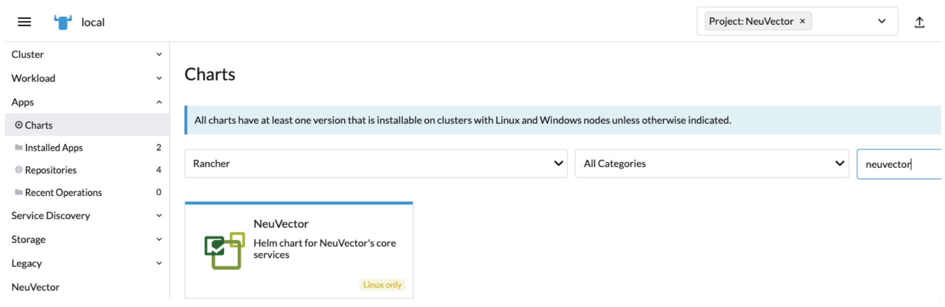
FIGURE 22: UNINSTALLING EXTENSION

> **Note**
>
> Uninstalling the SUSE Security UI extension does not uninstall the SUSE Security app from each cluster. The SUSE Security menu will revert to providing an SSO link into the SUSE Security console.

3.5.1.2 Deploy SUSE Security

First, find the SUSE Security chart in Rancher charts, select it and review the instructions and configuration values. Optionally, create a project to deploy into if desired, for example, SUSE Security.

> **Note**
>
> If you see more than one SUSE Security chart, do not select the one that is for upgrading legacy SUSE Security 4.x Helm chart deployments.



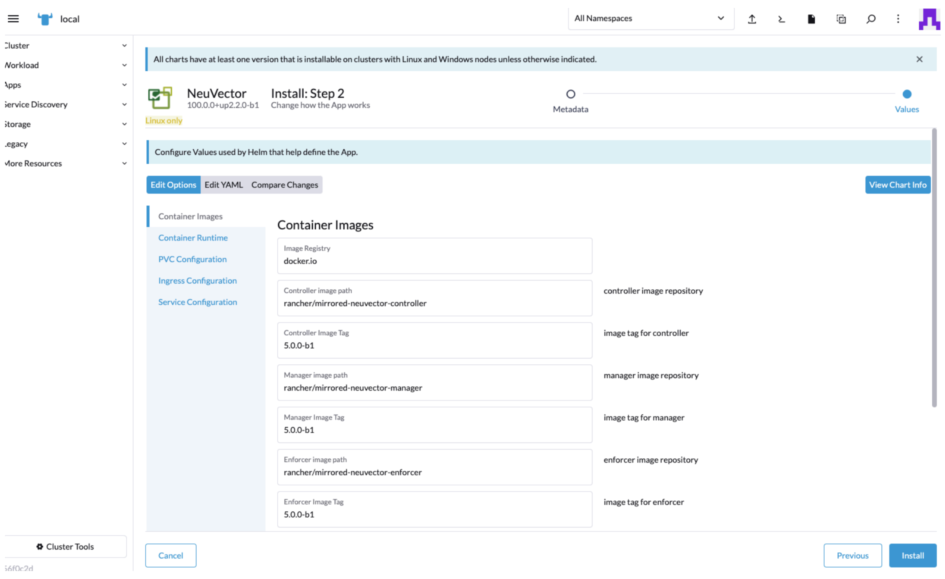
All charts have at least one version that is installable on clusters with Linux and Windows nodes unless otherwise indicated.

NeuVector
Helm chart for NeuVector's core services

FIGURE 23: RANCHER CHART

Deploy the SUSE Security chart, first configuring appropriate values for a Rancher deployment, such as:

- Container runtime, such as Docker for RKE and containerd for RKE2, or select the K3s value if using K3s.
- Manager service type: change to LoadBalancer if available on public cloud deployments. If access is only desired through Rancher, any allowed value will work here. See the Important note below about changing the default administration password in SUSE Security.
- Indicate whether this cluster will be a multi-cluster federated primary or remote (or select both if either option is desired).
- Persistent volume for configuration backups



All charts have at least one version that is installable on clusters with Linux and Windows nodes unless otherwise indicated.

NeuVector 100.0.0+up2.2.0-b1 Install: Step 2 Change how the App works

Configure Values used by Helm that help define the App.

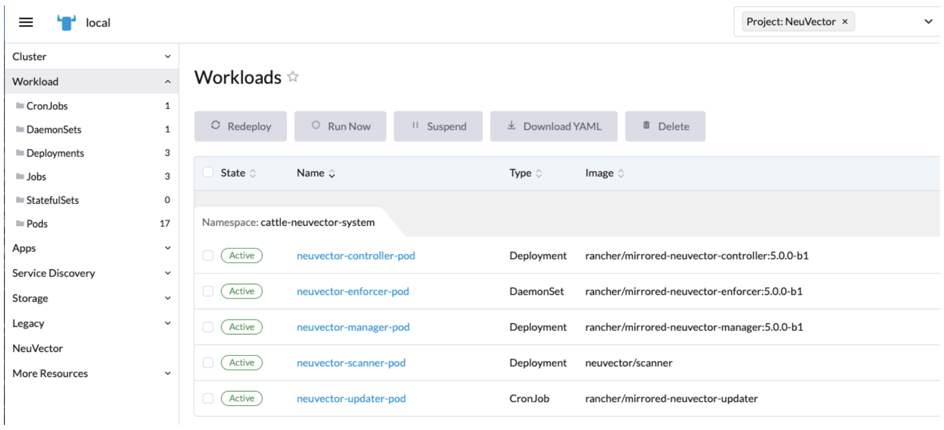
Container Images

Image Registry: docker.io	
Controller image path: rancher/mirrored-neuvector-controller	controller image repository
Controller Image Tag: 5.0.0-b1	image tag for controller
Manager image path: rancher/mirrored-neuvector-manager	manager image repository
Manager Image Tag: 5.0.0-b1	image tag for manager
Enforcer image path: rancher/mirrored-neuvector-enforcer	enforcer image repository
Enforcer Image Tag: 5.0.0-b1	image tag for enforcer

FIGURE 24: NEUVECTOR VALUES

Click *Install* after you have reviewed and updated any chart values.

After a successful SUSE Security deployment, you will see a summary of the deployments, daemon sets, and cron jobs for SUSE Security. You will also be able to see the services deployed in the Services Discovery menu on the left.



Namespace: cattle-neuvector-system

State	Name	Type	Image
Active	neuvector-controller-pod	Deployment	rancher/mirrored-neuvector-controller:5.0.0-b1
Active	neuvector-enforcer-pod	DaemonSet	rancher/mirrored-neuvector-enforcer:5.0.0-b1
Active	neuvector-manager-pod	Deployment	rancher/mirrored-neuvector-manager:5.0.0-b1
Active	neuvector-scanner-pod	Deployment	neuvector/scanner
Active	neuvector-updater-pod	CronJob	rancher/mirrored-neuvector-updater

FIGURE 25: NEUVECTOR DEPLOYED

3.5.1.3 Manage SUSE Security

You will now see a SUSE Security menu item in the left, and selecting that will show a SUSE Security tile/button, which when clicked will take you to the SUSE Security console, in a new tab.

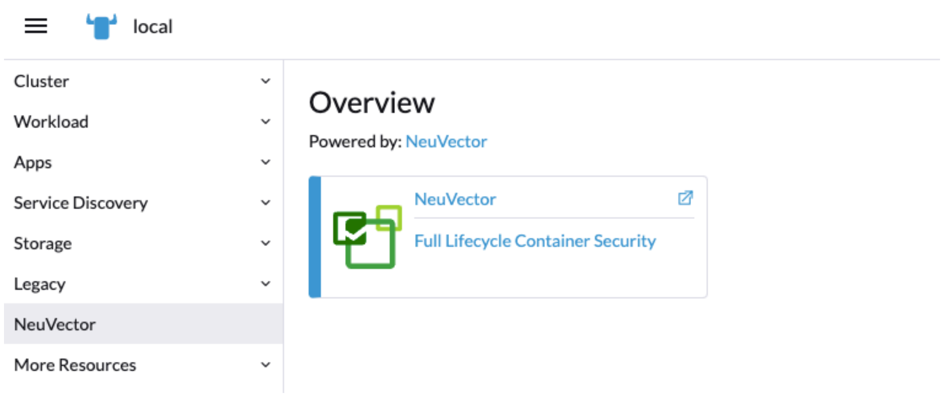
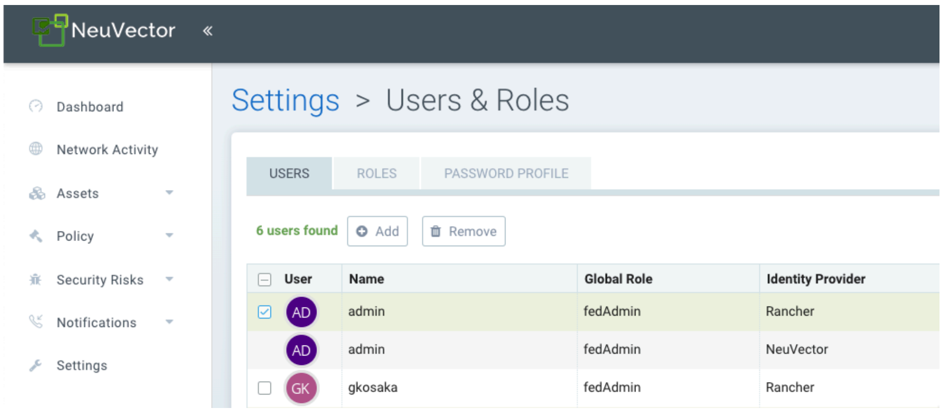


FIGURE 26: NEUVECTOR CONSOLE ACCESS

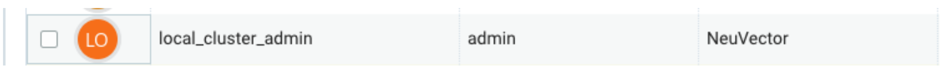
When this Single Sign-On (SSO) access method is used for the first time, a corresponding user in the SUSE Security cluster is created for the Rancher user login. The same user name as the Rancher logged-in user will be created in SUSE Security, with a role of either admin or fedAdmin, and Identity provider as Rancher.



User	Name	Global Role	Identity Provider
☑ AD	admin	fedAdmin	Rancher
AD	admin	fedAdmin	NeuVector
☐ GK	gkosaka	fedAdmin	Rancher

FIGURE 27: NEUVECTOR ADMINISTRATOR USERS

In the above screenshot, two Rancher users--`admin` and `gkosaka`--have been automatically created for SSO. If another user is created manually in SUSE Security, the identity provider would be listed as SUSE Security, as shown below. This local user can log in directly to the SUSE Security console without going through Rancher.



☐ LO	local_cluster_admin	admin	NeuVector

FIGURE 28: LOCAL ADMIN

Important

It is recommended to log in directly to the SUSE Security console as admin/admin to manually change the administrator password to a strong password. This will only change the SUSE Security identity provider administrator user password (you may see another administrator user whose identify provider is Rancher). Alternatively, include a ConfigMap as a secret (https://ranchermanager.docs.rancher.com/integrations-in-rancher/neuvector/configuration/configmaps#protect-sensitive-data-using-a-secret) in the initial deployment from Rancher (see chart values for ConfigMap settings) to set the default admin password to a strong password.

3.5.1.4 Neuvector/Rancher SSO permission resources

The Rancher v2.9.2 UI provides for selecting Neuvector permission resources when creating `Global/Cluster/Project/Namespaces` roles. When a Rancher user is assigned a role with a Neuvector permission resource, the user's Neuvector SSO session is assigned the respective Neuvector permission accordingly. This is to provide SSO users with custom roles other than the reserved `admin/reader/fedAdmin/fedReader` roles.

Below are the mapped permission resources used with applicable `Global/Cluster/Project/Namespaces` roles.

3.5.1.4.1 Mapped permission resources for `Global/Cluster` role

Note

Users will need to manually add * (Verbs) / services/proxy (Resource) to Neuvector-related `Global/Cluster` Roles.

API Groups:

`permission.neuvector.com`

Verbs:

```
get    // for read-only(i.e. view)
*      // for read/write(i.e. modify)
```

Resources:

Neuvector, Cluster Scoped

```
AdmissionControl
Authentication
CI Scan
Cluster
Federation
Vulnerability
```

Neuvector, Namespaced

```
AuditEvents
Authorization
Compliance
Events
```

```
Namespace
RegistryScan
RuntimePolicy
RuntimeScan
SecurityEvents
SystemConfig
```

3.5.1.4.2 Mapped permission resources for Project/Namespace role

Note

You will need to manually add * (Verbs) / services/proxy (Resource) to Neuvector-related `Project/Namespace` Roles.

API Groups:

`permission.neuvector.com`

Verbs:

```
get    // for read-only(i.e. view)
*      // for read/write(i.e. modify)
```

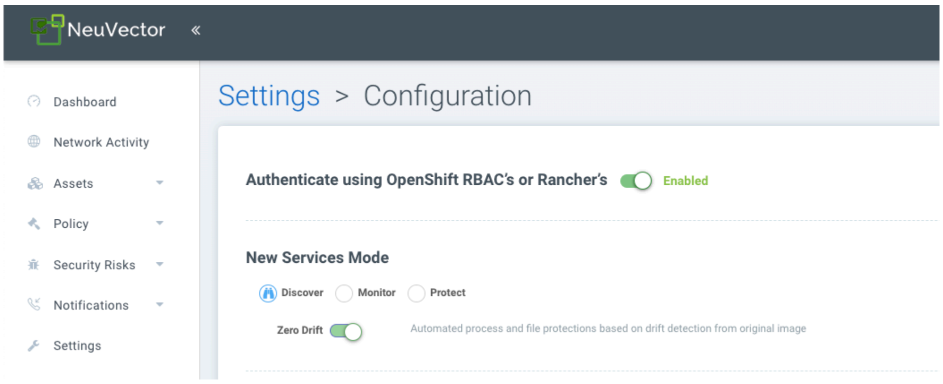
Resources:

Neuvector, Namespaced

```
AuditEvents
Authorization
Compliance
Events
Namespace
RegistryScan
RuntimePolicy
RuntimeScan
SecurityEvents
SystemConfig
```

3.5.1.5 Disabling SUSE Security/Rancher SSO

To disable the ability to log in to SUSE Security from SUSE Rancher Prime, go to Settings → Configuration.



Authenticate using OpenShift RBAC's or Rancher's: Enabled

New Services Mode: [✓ Discover] Monitor Protect

Zero Drift: Automated process and file protections based on drift detection from original image

FIGURE 29: RANCHER SSO

3.5.1.6 Rancher legacy deployments

The sample file will deploy one manager and 3 controllers. It will deploy an enforcer on every node. See the bottom section for specifying dedicated manager or controller nodes using node labels.

Note

We do not recommend deploying or scaling more than one manager behind a load balancer due to potential session state issues.

Note

Deployment on Rancher 2.x/Kubernetes should follow the Kubernetes reference section and/or Helm-based deployment.

1. Deploy the catalog `docker-compose-dist.yml`. Controllers will be deployed on the labeled nodes; enforcers will be deployed on the rest of the nodes. (The sample file can be modified so that enforcers are only deployed to the specified nodes.)
2. Pick one controller for the manager to connect to. Modify the manager's catalog file `docker-compose-manager.yml`, set `CTRL_SERVER_IP` to the controller's IP, then deploy the manager catalog.

Here are the sample compose files. If you wish to deploy only one or two of the components, just use that section of the file.

SUSE Rancher Prime/Controller/Enforcer Compose Sample File:

```
manager:
   scale: 1
   image: neuvector/manager
   restart: always
   environment:
     - CTRL_SERVER_IP=controller
   ports:
     - 8443:8443
controller:
   scale: 3
   image: neuvector/controller
   restart: always
   privileged: true
   environment:
     - CLUSTER_JOIN_ADDR=controller
   volumes:
     - /var/run/docker.sock:/var/run/docker.sock
     - /proc:/host/proc:ro
     - /sys/fs/cgroup:/host/cgroup:ro
     - /var/neuvector:/var/neuvector
enforcer:
   image: neuvector/enforcer
   pid: host
   restart: always
   privileged: true
   environment:
     - CLUSTER_JOIN_ADDR=controller
   volumes:
     - /lib/modules:/lib/modules
     - /var/run/docker.sock:/var/run/docker.sock
     - /proc:/host/proc:ro
     - /sys/fs/cgroup/:/host/cgroup/:ro
   labels:
     io.rancher.scheduler.global: true
```

3.5.1.7 Deploy without privileged mode

On certain systems, deployment without using privileged mode is supported. These systems must support the ability to add capabilities using the cap_add setting and to set the AppArmor profile.

Here is a sample Rancher compose file for deployment without privileged mode:

```
manager:
   scale: 1
   image: neuvector/manager
   restart: always
   environment:
     - CTRL_SERVER_IP=controller
   ports:
     - 8443:8443
controller:
   scale: 3
   image: neuvector/controller
   pid: host
   restart: always
   cap_add:
     - SYS_ADMIN
     - NET_ADMIN
     - SYS_PTRACE
   security_opt:
     - apparmor=unconfined
     - seccomp=unconfined
     - label=disable
   environment:
     - CLUSTER_JOIN_ADDR=controller
   volumes:
     - /var/run/docker.sock:/var/run/docker.sock
     - /proc:/host/proc:ro
     - /sys/fs/cgroup:/host/cgroup:ro
     - /var/neuvector:/var/neuvector
enforcer:
   image: neuvector/enforcer
   pid: host
   restart: always
   cap_add:
     - SYS_ADMIN
     - NET_ADMIN
     - SYS_PTRACE
     - IPC_LOCK
   security_opt:
     - apparmor=unconfined
     - seccomp=unconfined
     - label=disable
   environment:
     - CLUSTER_JOIN_ADDR=controller
   volumes:
     - /lib/modules:/lib/modules
     - /var/run/docker.sock:/var/run/docker.sock
     - /proc:/host/proc:ro
     - /sys/fs/cgroup/:/host/cgroup/:ro
   labels:
    io.rancher.scheduler.global: true
```

3.5.1.8 Using node labels for manager and controller nodes

To control which nodes the Manager and Controller are deployed on, label each node. Pick the nodes where the controllers are to be deployed. Label them with 'nvcontroller=true'. With the current sample file, no more than one controller can run on the same node.

For the manager node, label it 'nvmanager=true'.

Add labels to the YAML file. For example, for the manager:

```
  labels:
    io.rancher.scheduler.global: true
    io.rancher.scheduler.affinity:host_label: "nvmanager=true"
```

For the controller:

```
  labels:
    io.rancher.scheduler.global: true
    io.rancher.scheduler.affinity:host_label: "nvcontroller=true"
```

For the enforcer, to prevent it from running on a controller node (if desired):

```
 labels:
    io.rancher.scheduler.global: true
    io.rancher.scheduler.affinity:host_label_ne: "nvcontroller=true"
```

3.5.2 Installing SUSE Security using Kubernetes

You can use Kubernetes to deploy separate manager, controller and enforcer containers and make sure that all new nodes have an enforcer deployed. SUSE Security requires and supports Kubernetes network plug-ins such as flannel, weave and calico.

The sample file will deploy one manager and 3 controllers. It will deploy an enforcer on every node as a daemonset. By default, the sample below will deploy to the Master node as well.

Refer to *Section 3.5.2.3, “Using node labels for manager and controller nodes”* for specifying dedicated manager or controller nodes using node labels.

Note

It is not recommended to deploy (scale) more than one manager behind a load balancer due to potential session state issues. If you plan to use a PersistentVolume claim to store the backup of SUSE Security configuration files, please see the general Backup/Persistent Data section in the Deploying SUSE Security (https://ranchermanager.docs.rancher.com/integrations-in-rancher/neuvector/production-deployment-considerations#backups-and-persistent-data) overview.

If your deployment supports an integrated load balancer, change type `NodePort` to `LoadBalancer` for the console in the YAML file below.

SUSE Security supports Helm-based deployment with a Helm chart at https://github.com/neuvector/neuvector-helm.

There is a separate section for OpenShift instructions, and EE on Kubernetes has some special steps described in the Docker section.

3.5.2.1 SUSE Security images on Docker Hub

The images are on the SUSE Security Docker Hub registry. Use the appropriate version tag for the manager, controller and enforcer, and leave the version as 'latest' for scanner and updater. For example:

- `neuvector/manager:5.4.3`
- `neuvector/controller:5.4.3`
- `neuvector/enforcer:5.4.3`
- `neuvector/scanner:latest`
- `neuvector/updater:latest`

Be sure to update the image references in the appropriate YAML files.

If deploying with the current SUSE Security Helm chart (v1.8.9+), the following changes should be made to `values.yml`:

- Update the registry to `docker.io`.
- Update image names and tags to the current version on Docker Hub, as shown above.
- Leave `imagePullSecrets` empty.

Note

If deploying from the SUSE Rancher Prime 2.6.5+ SUSE Security chart, images are pulled automatically from the Rancher Registry mirrored image repo, and deployed into the `cattle-neuvector-system` namespace.

3.5.2.2 Deploy SUSE Security

1. Create the SUSE Security namespace and the required service accounts:

```
> kubectl create namespace neuvector
> kubectl create sa controller -n neuvector
> kubectl create sa enforcer -n neuvector
> kubectl create sa basic -n neuvector
> kubectl create sa updater -n neuvector
> kubectl create sa scanner -n neuvector
> kubectl create sa registry-adapter -n neuvector
> kubectl create sa cert-upgrader -n neuvector
```

2. *(Optional)* Create the SUSE Security Pod Security Admission (PSA) or Pod Security Policy (PSP). If you have enabled Pod Security Admission (aka Pod Security Standards) in Kubernetes 1.25+, or Pod Security Policies (prior to 1.25) in your Kubernetes cluster, add the following for SUSE Security (for example, `nv_psp.yaml`).

Note

- PSP is deprecated in Kubernetes 1.21 and will be removed in 1.25.
- The Manager and Scanner pods run without a UID. If your PSP has a rule `Run As User: Rule: MustRunAsNonRoot` then add the following into the sample YAML below with the appropriate value for `#`:

```
securityContext:
    runAsUser: ###
```

For PSA in Kubernetes 1.25+, label the SUSE Security namespace with the privileged profile for deploying on a PSA-enabled cluster.

```
> kubectl label namespace neuvector \
  "pod-security.kubernetes.io/enforce=privileged"
```

3. Create the custom resources (CRD) for SUSE Security rules. For Kubernetes 1.19+:

Note

If you are upgrading to version `5.4.6` using YAML, you must deploy the `responserules-crd-k8s.yaml` file. If you are using Helm charts, this step is handled automatically, and no action is required.

```
> kubectl apply -f https://raw.githubusercontent.com/neuvector/manifests/main/
kubernetes/5.4.0/crd-k8s-1.19.yaml
> kubectl apply -f https://raw.githubusercontent.com/neuvector/manifests/main/
kubernetes/5.4.0/waf-crd-k8s-1.19.yaml
> kubectl apply -f https://raw.githubusercontent.com/neuvector/manifests/main/
kubernetes/5.4.0/dlp-crd-k8s-1.19.yaml
> kubectl apply -f https://raw.githubusercontent.com/neuvector/manifests/main/
kubernetes/5.4.0/com-crd-k8s-1.19.yaml
> kubectl apply -f https://raw.githubusercontent.com/neuvector/manifests/main/
kubernetes/5.4.0/vul-crd-k8s-1.19.yaml
> kubectl apply -f https://raw.githubusercontent.com/neuvector/manifests/main/
kubernetes/5.4.0/admission-crd-k8s-1.19.yaml
> kubectl apply -f https://raw.githubusercontent.com/neuvector/manifests/main/
kubernetes/5.4.0/5.4.3_group-definition-k8s.yaml
> kubectl apply -f https://raw.githubusercontent.com/neuvector/manifests/main/
kubernetes/5.4.0/5.4.3_group-definition-k8s
> kubectl apply -f https://raw.githubusercontent.com/neuvector/manifests/main/
kubernetes/5.4.0/responserules-crd-k8s.yaml
```

4. Add read permission to access the Kubernetes API.

Important

The standard SUSE Security 5.2+ deployment uses least-privileged service accounts instead of the default. See below if upgrading from a version prior to 5.3.

Warning

If you are upgrading to 5.3.0+, run the following commands based on your current version:

Version 5.2.0:

```
> kubectl delete clusterrole neuvector-binding-nvsecurityrules \
  neuvector-binding-nvadmissioncontrolsecurityrules \
  neuvector-binding-nvdlpsecurityrules \
  neuvector-binding-nvwafsecurityrules
```

Versions prior to 5.2.0:

```
> kubectl delete clusterrolebinding \
  neuvector-binding-app neuvector-binding-rbac \
  neuvector-binding-admission \
  neuvector-binding-customresourcedefinition \
  neuvector-binding-nvsecurityrules \
  neuvector-binding-view \
  neuvector-binding-nvwafsecurityrules \
  neuvector-binding-nvadmissioncontrolsecurityrules \
  neuvector-binding-nvdlpsecurityrules
> kubectl delete rolebinding neuvector-admin -n neuvector
```

Apply the read permissions via the following `create clusterrole` commands:

```
> kubectl create clusterrole neuvector-binding-app --verb=get,list,watch,update --
resource=nodes,pods,services,namespaces
> kubectl create clusterrole neuvector-binding-rbac --verb=get,list,watch --
resource=rolebindings.rbac.authorization.k8s.io,roles.rbac.authorization.k8s.io,clusterrolebinding
> kubectl create clusterrolebinding neuvector-binding-app --clusterrole=neuvector-
binding-app --serviceaccount=neuvector:controller
> kubectl create clusterrolebinding neuvector-binding-rbac --clusterrole=neuvector-
binding-rbac --serviceaccount=neuvector:controller
> kubectl create clusterrole neuvector-binding-
admission --verb=get,list,watch,create,update,delete --
resource=validatingwebhookconfigurations,mutatingwebhookconfigurations
> kubectl create clusterrolebinding neuvector-binding-admission --
clusterrole=neuvector-binding-admission --serviceaccount=neuvector:controller
> kubectl create clusterrole neuvector-binding-customresourcedefinition --
verb=watch,create,get,update --resource=customresourcedefinitions
> kubectl create clusterrolebinding neuvector-binding-customresourcedefinition
 --clusterrole=neuvector-binding-customresourcedefinition --
serviceaccount=neuvector:controller
> kubectl create clusterrole neuvector-binding-nvsecurityrules --
verb=get,list,delete --resource=nvsecurityrules,nvclustersecurityrules
> kubectl create clusterrole neuvector-binding-nvadmissioncontrolsecurityrules --
verb=get,list,delete --resource=nvadmissioncontrolsecurityrules
```

```
> kubectl create clusterrole neuvector-binding-nvdlpsecurityrules --
verb=get,list,delete --resource=nvdlpsecurityrules
> kubectl create clusterrole neuvector-binding-nvwafsecurityrules --
verb=get,list,delete --resource=nvwafsecurityrules
> kubectl create clusterrolebinding neuvector-binding-nvsecurityrules --
clusterrole=neuvector-binding-nvsecurityrules --serviceaccount=neuvector:controller
> kubectl create clusterrolebinding neuvector-binding-view --clusterrole=view --
serviceaccount=neuvector:controller
> kubectl create clusterrolebinding neuvector-binding-nvwafsecurityrules
 --clusterrole=neuvector-binding-nvwafsecurityrules --
serviceaccount=neuvector:controller
> kubectl create clusterrolebinding neuvector-binding-
nvadmissioncontrolsecurityrules --clusterrole=neuvector-binding-
nvadmissioncontrolsecurityrules --serviceaccount=neuvector:controller
> kubectl create clusterrolebinding neuvector-binding-nvdlpsecurityrules
 --clusterrole=neuvector-binding-nvdlpsecurityrules --
serviceaccount=neuvector:controller
> kubectl create role neuvector-binding-scanner --verb=get,patch,update,watch --
resource=deployments -n neuvector
> kubectl create rolebinding neuvector-binding-scanner --role=neuvector-binding-
scanner --serviceaccount=neuvector:updater --serviceaccount=neuvector:controller -n
 neuvector
> kubectl create role neuvector-binding-secret --verb=get --resource=secrets -n
 neuvector
> kubectl create rolebinding neuvector-binding-secret --role=neuvector-binding-
secret --serviceaccount=neuvector:controller -n neuvector
> kubectl create role neuvector-binding-secret --verb=get,list,watch --
resource=secrets -n neuvector
> kubectl create rolebinding neuvector-binding-secret --role=neuvector-binding-
secret --serviceaccount=neuvector:controller --serviceaccount=neuvector:enforcer
 --serviceaccount=neuvector:scanner --serviceaccount=neuvector:registry-adapter -n
 neuvector
> kubectl create clusterrole neuvector-binding-nvcomplianceprofiles --
verb=get,list,delete --resource=nvcomplianceprofiles
> kubectl create clusterrolebinding neuvector-binding-nvcomplianceprofiles
 --clusterrole=neuvector-binding-nvcomplianceprofiles --
serviceaccount=neuvector:controller
> kubectl create clusterrole neuvector-binding-nvvulnerabilityprofiles --
verb=get,list,delete --resource=nvvulnerabilityprofiles
> kubectl create clusterrolebinding neuvector-binding-nvvulnerabilityprofiles
 --clusterrole=neuvector-binding-nvvulnerabilityprofiles --
serviceaccount=neuvector:controller
> kubectl apply -f https://raw.githubusercontent.com/neuvector/manifests/main/
kubernetes/5.4.0/neuvector-roles-k8s.yaml
> kubectl create role neuvector-binding-lease --verb=create,get,update --
resource=leases -n neuvector
```

```
> kubectl create rolebinding neuvector-binding-cert-upgrader --role=neuvector-
binding-cert-upgrader --serviceaccount=neuvector:cert-upgrader -n neuvector
> kubectl create rolebinding neuvector-binding-job-creation --role=neuvector-
binding-job-creation --serviceaccount=neuvector:controller -n neuvector
> kubectl create rolebinding neuvector-binding-lease --role=neuvector-binding-lease
 --serviceaccount=neuvector:controller --serviceaccount=neuvector:cert-upgrader -n
 neuvector
> kubectl create clusterrole neuvector-binding-nvgroupdefinitions --
verb=list,get,delete --resource=nvgroupdefinitions
> kubectl create clusterrolebinding neuvector-binding-nvgroupdefinitions
 --clusterrole=neuvector-binding-nvgroupdefinitions --
serviceaccount=neuvector:controller
> kubectl create role neuvector-binding-secret-controller --verb=create,patch,update
 --resource=secrets -n neuvector
> kubectl create rolebinding neuvector-binding-secret-controller --
role=neuvector-binding-secret-controller --serviceaccount=neuvector:controller --
serviceaccount=neuvector:default -n neuvector
> kubectl create clusterrole neuvector-binding-nvresponserulesecurityrules --
verb=get,list,delete --resource=nvresponserulesecurityrules
> kubectl create clusterrolebinding neuvector-binding-nvresponserulesecurityrules
 --clusterrole=neuvector-binding-nvresponserulesecurityrules --
serviceaccount=neuvector:controller
```

5. Run the following commands to check if the neuvector/controller and neuvector/updater service accounts are added successfully.

```
> kubectl get ClusterRoleBinding \
  neuvector-binding-app neuvector-binding-rbac \
  neuvector-binding-admission \
  neuvector-binding-customresourcedefinition \
  neuvector-binding-nvsecurityrules \
  neuvector-binding-view \
  neuvector-binding-nvwafsecurityrules \
  neuvector-binding-nvadmissioncontrolsecurityrules \
  neuvector-binding-nvdlpsecurityrules \
  neuvector-binding-nvgroupdefinitions \
  neuvector-binding-nvresponserulesecurityrules -o wide
```

Sample output:

```
NAME                                                ROLE
                                AGE   USERS   GROUPS   SERVICEACCOUNTS
neuvector-binding-app                               ClusterRole/neuvector-binding-
app                               66d                    neuvector/controller
neuvector-binding-rbac                              ClusterRole/neuvector-binding-
rbac                              66d                    neuvector/controller
```

```
neuvector-binding-admission                         ClusterRole/neuvector-binding-
admission                         66d                    neuvector/controller
neuvector-binding-customresourcedefinition          ClusterRole/neuvector-binding-
customresourcedefinition          66d                    neuvector/controller
neuvector-binding-nvsecurityrules                   ClusterRole/neuvector-binding-
nvsecurityrules                   66d                    neuvector/controller
neuvector-binding-view                              ClusterRole/view
                                66d                    neuvector/controller
neuvector-binding-nvwafsecurityrules                ClusterRole/neuvector-binding-
nvwafsecurityrules                66d                    neuvector/controller
neuvector-binding-nvadmissioncontrolsecurityrules   ClusterRole/neuvector-binding-
nvadmissioncontrolsecurityrules   66d                    neuvector/controller
neuvector-binding-nvdlpsecurityrules                ClusterRole/neuvector-binding-
nvdlpsecurityrules                66d                    neuvector/controller
neuvector-binding-nvgroupdefinitions                ClusterRole/neuvector-binding-
nvgroupdefinitions                66d                    neuvector/controller
```

And this command:

```
> kubectl get RoleBinding neuvector-binding-scanner \
  neuvector-binding-cert-upgrader \
  neuvector-binding-job-creation \
  neuvector-binding-lease \
  neuvector-binding-secret -n neuvector -o wide
```

Sample output:

```
NAME                              ROLE                                       AGE
 USERS   GROUPS   SERVICEACCOUNTS
neuvector-binding-scanner         Role/neuvector-binding-scanner         8m8s
             neuvector/controller, neuvector/updater
neuvector-binding-cert-upgrader   Role/neuvector-binding-cert-upgrader   8m8s
             neuvector/cert-upgrader
neuvector-binding-job-creation    Role/neuvector-binding-job-creation    8m8s
             neuvector/controller
neuvector-binding-lease           Role/neuvector-binding-lease           8m8s
             neuvector/controller, neuvector/cert-upgrader
neuvector-binding-secret          Role/neuvector-binding-secret          8m8s
             neuvector/controller, neuvector/enforcer, neuvector/scanner, neuvector/
registry-adapter
```

6. *(Optional)* Create the Federation Master and/or Remote Multi-Cluster Management Services. If you plan to use the multi-cluster management functions in SUSE Security, one cluster must have the Federation Master service deployed, and each remote cluster must have the Federation Worker service. For flexibility, you may choose to deploy both Master and Worker services on each cluster so any cluster can be a master or remote.

```
apiVersion: v1
kind: Service
metadata:
  name: neuvector-service-controller-fed-master
  namespace: neuvector
spec:
  ports:
  - port: 11443
    name: fed
    protocol: TCP
  type: LoadBalancer
  selector:
    app: neuvector-controller-pod

---

apiVersion: v1
kind: Service
metadata:
  name: neuvector-service-controller-fed-worker
  namespace: neuvector
spec:
  ports:
  - port: 10443
    name: fed
    protocol: TCP
  type: LoadBalancer
  selector:
    app: neuvector-controller-pod
```

   Then create the appropriate service(s):

```
> kubectl create -f nv_master_worker.yaml
```

7. Create the primary SUSE Security services and pods using the preset version commands or modify the sample YAML below. The preset version invokes a LoadBalancer for the SUSE Security Console. If using the sample YAML file below, replace the image names and `VERSION` tags for the manager, controller and enforcer image references in the YAML

file. Also, make any other modifications required for your deployment environment (such as LoadBalancer/NodePort/Ingress for manager access). The YAML below needs to be changed for internal certificate changes if deployed from v5.4.2 or above. Refer to this *Section 3.5.2.7, "Kubernetes deployment YAML for v5.4.2 onwards"*.

```
> kubectl apply -f https://raw.githubusercontent.com/neuvector/manifests/main/
kubernetes/5.4.0/neuvector-k8s.yaml
```

Or, if modifying any of the above YAML or samples from below:

```
> kubectl create -f neuvector.yaml
```

Now you should be able to connect to the SUSE Security console and log in with admin:admin, for example: `https://<PUBLIC-IP>:8443`

Note

The `nodeport` service specified in the `neuvector.yaml` file will open a random port on all Kubernetes nodes for the SUSE Security Management Web console port. Alternatively, you can use a LoadBalancer or Ingress, using a public IP and default port 8443. For `nodeport`, be sure to open access through firewall rules for that port, if needed. To see which port is open on the host nodes, please run the following commands:

```
> kubectl get svc -n neuvector
```

And you will see something like:

```
NAME                            CLUSTER-IP      EXTERNAL-IP   PORT(S)
                    AGE
neuvector-service-webui       10.100.195.99     <nodes>       8443:30257/TCP
                    15m
```

PKS Change

Note

PKS is field-tested and requires enabling privileged containers to the plan/tile, and changing the YAML hostPath as follows for All-in-One, Controller and Enforcer:

```
hostPath:
  path: /var/vcap/sys/run/docker/docker.sock
```

Master node taints and tolerations

All taint info must match to schedule Enforcers on nodes. To check the taint info on a node (such as Master):

```
> kubectl get node taintnodename -o yaml
```

Sample output:

```
spec:
  taints:
  - effect: NoSchedule
    key: node-role.kubernetes.io/master
  # there may be an extra info for taint as below
  - effect: NoSchedule
    key: mykey
    value: myvalue
```

If there are additional taints as above, add these to the sample YAML tolerations section:

```
spec:
  template:
    spec:
      tolerations:
        - effect: NoSchedule
          key: node-role.kubernetes.io/master
        - effect: NoSchedule
          key: node-role.kubernetes.io/control-plane
        # if there is an extra info for taints as above, please add it here.
        # This is required to match all the taint info defined on the taint
        # node. Otherwise, the Enforcer won't deploy on the taint node
        - effect: NoSchedule
          key: mykey
          value: myvalue
```

3.5.2.3 Using node labels for manager and controller nodes

To control which nodes the Manager and Controller are deployed on, label each node. Replace `NODE_NAME` with the appropriate node name (“kubectl get nodes”). Note: By default, Kubernetes will not schedule pods on the master node.

```
> kubectl label nodes <NODE_NAME> nvcontroller=true
```

Then add a nodeSelector to the YAML file for the Manager and Controller deployment sections. For example:

```
          - mountPath: /host/cgroup
              name: cgroup-vol
              readOnly: true
      nodeSelector:
        nvcontroller: "true"
      restartPolicy: Always
```

To prevent the enforcer from being deployed on a controller node, if it is a dedicated management node (without application containers to be monitored), add a nodeAffinity to the Enforcer YAML section. For example:

```
  app: neuvector-enforcer-pod
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                - key: nvcontroller
                  operator: NotIn
                  values: ["true"]
      imagePullSecrets:
```

3.5.2.4 Rolling updates

Orchestration tools such as Kubernetes, Red Hat OpenShift, and Rancher support rolling updates with configurable policies. You can use this feature to update the SUSE Security containers. The most important thing will be to ensure that there is at least one Controller (or All-in-One) running so that policies, logs and connection data are not lost. Make sure that there is a minimum of 120 seconds between container updates so that a new leader can be elected and the data synchronized between controllers.

The provided sample deployment YAMLs already configure the rolling update policy. If you are updating via the SUSE Security Helm chart, please pull the latest chart to properly configure new features such as admission control, and delete the old cluster role and cluster role binding for SUSE Security. If you are updating via Kubernetes, you can manually update to a new version with the sample commands below.

3.5.2.4.1 Sample Kubernetes rolling update

For upgrades that just need to update to a new image version, you can use this simple approach.

If your Deployment or DaemonSet is already running, you can change the YAML file to the new version, then apply the update:

```
> kubectl apply -f <YAML_FILE>
```

This will update to a new version of SUSE Security from the command line.

For the controller as a Deployment (also do the same for the manager):

```
> kubectl set image deployment/neuvector-controller-pod \
  neuvector-controller-pod=neuvector/controller:<VERSION> -n neuvector
```

For any container as a DaemonSet:

```
> kubectl set image -n neuvector \
  ds/neuvector-enforcer-pod neuvector-enforcer-pod=neuvector/enforcer:<VERSION>
```

To check the status of the rolling update:

```
> kubectl rollout status -n neuvector ds/neuvector-enforcer-pod
> kubectl rollout status -n neuvector deployment/neuvector-controller-pod
```

To roll back the update:

```
> kubectl rollout undo -n neuvector ds/neuvector-enforcer-pod
> kubectl rollout undo -n neuvector deployment/neuvector-controller-pod
```

3.5.2.5 Expose REST API in Kubernetes

To expose the REST API for access from outside of the Kubernetes cluster, here is a sample YAML file:

```
apiVersion: v1
kind: Service
metadata:
  name: neuvector-service-rest
  namespace: neuvector
spec:
  ports:
    - port: 10443
      name: controller
      protocol: TCP
  type: LoadBalancer
  selector:
    app: neuvector-controller-pod
```

Please see the Automation section for more info on the REST API.

3.5.2.6 Kubernetes deployment in non-privileged mode

The following instructions can be used to deploy SUSE Security without using privileged mode containers. The controller is already in non-privileged mode and enforcer deployment should be changed, which is shown in the excerpted snippets below.

Enforcer:

```
spec:
  template:
    metadata:
      annotations:
        container.apparmor.security.beta.kubernetes.io/neuvector-enforcer-pod: unconfined
        # this line is required to be added if k8s version is pre-v1.19
        # container.seccomp.security.alpha.kubernetes.io/neuvector-enforcer-pod:
 unconfined
    spec:
      containers:
          securityContext:
            # the following two lines are required for k8s v1.19+.
            # Comment out both lines if version is pre-1.19.
            # Otherwise, a validating data error message will show
            seccompProfile:
              type: Unconfined
            capabilities:
              add:
              - SYS_ADMIN
              - NET_ADMIN
              - SYS_PTRACE
              - IPC_LOCK
```

3.5.2.7 Kubernetes deployment YAML for v5.4.2 onwards

The following sample YAML is for versions 5.4.2 and onwards where we need to mount the internal certificates on Controller, Enforcer and Scanner pods since we do not support hard-coded certificates anymore. Create the internal-certificate secret from the given link before deploying: Replacing Internal Certificates (https://ranchermanager.docs.rancher.com/integrations-in-rancher/neuvector/custom-certs#replacing-internal-certificates) ↗.

```
apiVersion: v1
```

```
kind: Service
metadata:
  name: neuvector-svc-crd-webhook
  namespace: neuvector
spec:
  ports:
  - port: 443
    targetPort: 30443
    protocol: TCP
    name: crd-webhook
  type: ClusterIP
  selector:
    app: neuvector-controller-pod

---

apiVersion: v1
kind: Service
metadata:
  name: neuvector-svc-admission-webhook
  namespace: neuvector
spec:
  ports:
  - port: 443
    targetPort: 20443
    protocol: TCP
    name: admission-webhook
  type: ClusterIP
  selector:
    app: neuvector-controller-pod

---

apiVersion: v1
kind: Service
metadata:
  name: neuvector-service-webui
  namespace: neuvector
spec:
  ports:
    - port: 8443
      name: manager
      protocol: TCP
  type: LoadBalancer
  selector:
    app: neuvector-manager-pod
```

```
---

apiVersion: v1
kind: Service
metadata:
  name: neuvector-svc-controller
  namespace: neuvector
spec:
  ports:
  - port: 18300
    protocol: "TCP"
    name: "cluster-tcp-18300"
  - port: 18301
    protocol: "TCP"
    name: "cluster-tcp-18301"
  - port: 18301
    protocol: "UDP"
    name: "cluster-udp-18301"
  clusterIP: None
  selector:
    app: neuvector-controller-pod

---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: neuvector-manager-pod
  namespace: neuvector
spec:
  selector:
    matchLabels:
      app: neuvector-manager-pod
  replicas: 1
  template:
    metadata:
      labels:
        app: neuvector-manager-pod
    spec:
      serviceAccountName: basic
      serviceAccount: basic
      containers:
        - name: neuvector-manager-pod
          image: neuvector/manager:5.4.3
          env:
            - name: CTRL_SERVER_IP
              value: neuvector-svc-controller.neuvector
      restartPolicy: Always

---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: neuvector-controller-pod
  namespace: neuvector
spec:
  selector:
    matchLabels:
      app: neuvector-controller-pod
  minReadySeconds: 60
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  replicas: 3
  template:
    metadata:
      labels:
        app: neuvector-controller-pod
    spec:
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                  - neuvector-controller-pod
              topologyKey: "kubernetes.io/hostname"
      serviceAccountName: controller
      serviceAccount: controller
      containers:
        - name: neuvector-controller-pod
          image: neuvector/controller:5.4.3
          securityContext:
            runAsUser: 0
          readinessProbe:
            exec:
              command:
```

```
            - cat
            - /tmp/ready
          failureThreshold: 3
          initialDelaySeconds: 5
          periodSeconds: 5
          successThreshold: 1
          timeoutSeconds: 1
        env:
          - name: CLUSTER_JOIN_ADDR
            value: neuvector-svc-controller.neuvector
          - name: CLUSTER_ADVERTISED_ADDR
            valueFrom:
              fieldRef:
                fieldPath: status.podIP
          - name: CLUSTER_BIND_ADDR
            valueFrom:
              fieldRef:
                fieldPath: status.podIP
        volumeMounts:
          - mountPath: /etc/config
            name: config-volume
            readOnly: true
          - mountPath: /etc/neuvector/certs/internal/cert.key
            name: internal-cert
            readOnly: true
            subPath: tls.key
          - mountPath: /etc/neuvector/certs/internal/cert.pem
            name: internal-cert
            readOnly: true
            subPath: tls.crt
          - mountPath: /etc/neuvector/certs/internal/ca.cert
            name: internal-cert
            readOnly: true
            subPath: ca.crt
    terminationGracePeriodSeconds: 300
    restartPolicy: Always
    volumes:
      - name: config-volume
        projected:
          sources:
            - configMap:
                name: neuvector-init
                optional: true
            - secret:
                name: neuvector-init
                optional: true
            - secret:
                  name: neuvector-secret
                  optional: true
        - name: internal-cert
          secret:
            defaultMode: 420
            secretName: internal-cert

---

apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: neuvector-enforcer-pod
  namespace: neuvector
spec:
  selector:
    matchLabels:
      app: neuvector-enforcer-pod
  updateStrategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: neuvector-enforcer-pod
    spec:
      tolerations:
        - effect: NoSchedule
          key: node-role.kubernetes.io/master
        - effect: NoSchedule
          key: node-role.kubernetes.io/control-plane
      hostPID: true
      serviceAccountName: enforcer
      serviceAccount: enforcer
      containers:
        - name: neuvector-enforcer-pod
          image: neuvector/enforcer:5.4.3
          securityContext:
            privileged: true
          env:
            - name: CLUSTER_JOIN_ADDR
              value: neuvector-svc-controller.neuvector
            - name: CLUSTER_ADVERTISED_ADDR
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
            - name: CLUSTER_BIND_ADDR
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
          volumeMounts:
            - mountPath: /lib/modules
              name: modules-vol
              readOnly: true
            - mountPath: /var/nv_debug
              name: nv-debug
              readOnly: false
            - mountPath: /etc/neuvector/certs/internal/cert.key
              name: internal-cert
              readOnly: true
              subPath: tls.key
            - mountPath: /etc/neuvector/certs/internal/cert.pem
              name: internal-cert
              readOnly: true
              subPath: tls.crt
            - mountPath: /etc/neuvector/certs/internal/ca.cert
              name: internal-cert
              readOnly: true
              subPath: ca.crt
      terminationGracePeriodSeconds: 1200
      restartPolicy: Always
      volumes:
        - name: modules-vol
          hostPath:
            path: /lib/modules
        - name: nv-debug
          hostPath:
            path: /var/nv_debug
        - name: internal-cert
          secret:
            defaultMode: 420
            secretName: internal-cert

---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: neuvector-scanner-pod
  namespace: neuvector
spec:
  selector:
    matchLabels:
      app: neuvector-scanner-pod
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  replicas: 2
  template:
    metadata:
      labels:
        app: neuvector-scanner-pod
    spec:
      serviceAccountName: scanner
      serviceAccount: scanner
      containers:
        - name: neuvector-scanner-pod
          image: neuvector/scanner:latest
          imagePullPolicy: Always
          env:
            - name: CLUSTER_JOIN_ADDR
              value: neuvector-svc-controller.neuvector
          volumeMounts:
            - mountPath: /etc/neuvector/certs/internal/cert.key
              name: internal-cert
              readOnly: true
              subPath: tls.key
            - mountPath: /etc/neuvector/certs/internal/cert.pem
              name: internal-cert
              readOnly: true
              subPath: tls.crt
            - mountPath: /etc/neuvector/certs/internal/ca.cert
              name: internal-cert
              readOnly: true
              subPath: ca.crt
      restartPolicy: Always
      volumes:
        - name: internal-cert
          secret:
            defaultMode: 420
            secretName: internal-cert
---

apiVersion: batch/v1
kind: CronJob
metadata:
  name: neuvector-updater-pod
  namespace: neuvector
spec:
  schedule: "0 0 * * *"
  jobTemplate:
    spec:
      template:
        metadata:
          labels:
            app: neuvector-updater-pod
        spec:
          serviceAccountName: updater
          serviceAccount: updater
          containers:
          - name: neuvector-updater-pod
            image: neuvector/updater:latest
            imagePullPolicy: Always
            command:
            - /bin/sh
            - -c
            - TOKEN=`cat /var/run/secrets/kubernetes.io/serviceaccount/token`; /usr/
bin/curl -kv -X PATCH -H "Authorization:Bearer $TOKEN" -H "Content-Type:application/
strategic-merge-patch+json" -d '{"spec":{"template":{"metadata":{"annotations":
{"kubectl.kubernetes.io/restartedAt":"'`date +%Y-%m-%dT%H:%M:%S%z`'"}}}}}' 'https://
kubernetes.default/apis/apps/v1/namespaces/neuvector/deployments/neuvector-scanner-pod'
          restartPolicy: Never
```

The following sample is a complete deployment reference (Kubernetes 1.19+).

```
apiVersion: v1
kind: Service
metadata:
  name: neuvector-svc-crd-webhook
  namespace: neuvector
spec:
  ports:
  - port: 443
    targetPort: 30443
    protocol: TCP
    name: crd-webhook
  type: ClusterIP
  selector:
    app: neuvector-controller-pod

---

apiVersion: v1
kind: Service
metadata:
  name: neuvector-svc-admission-webhook
  namespace: neuvector
spec:
  ports:
  - port: 443
    targetPort: 20443
    protocol: TCP
    name: admission-webhook
  type: ClusterIP
  selector:
    app: neuvector-controller-pod

---

apiVersion: v1
kind: Service
metadata:
  name: neuvector-service-webui
  namespace: neuvector
spec:
  ports:
    - port: 8443
      name: manager
      protocol: TCP
  type: LoadBalancer
  selector:
    app: neuvector-manager-pod

---

apiVersion: v1
kind: Service
metadata:
  name: neuvector-svc-controller
  namespace: neuvector
spec:
  ports:
  - port: 18300
    protocol: "TCP"
    name: "cluster-tcp-18300"
  - port: 18301
    protocol: "TCP"
    name: "cluster-tcp-18301"
  - port: 18301
    protocol: "UDP"
    name: "cluster-udp-18301"
  clusterIP: None
  selector:
    app: neuvector-controller-pod
```

```
---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: neuvector-manager-pod
  namespace: neuvector
spec:
  selector:
    matchLabels:
      app: neuvector-manager-pod
  replicas: 1
  template:
    metadata:
      labels:
        app: neuvector-manager-pod
    spec:
      serviceAccountName: basic
      serviceAccount: basic
      containers:
        - name: neuvector-manager-pod
          image: neuvector/manager:5.4.3
          env:
            - name: CTRL_SERVER_IP
              value: neuvector-svc-controller.neuvector
      restartPolicy: Always

---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: neuvector-controller-pod
  namespace: neuvector
spec:
  selector:
    matchLabels:
      app: neuvector-controller-pod
  minReadySeconds: 60
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  replicas: 3
  template:
    metadata:
      labels:
        app: neuvector-controller-pod
    spec:
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                  - neuvector-controller-pod
              topologyKey: "kubernetes.io/hostname"
      serviceAccountName: controller
      serviceAccount: controller
      containers:
        - name: neuvector-controller-pod
          image: neuvector/controller:5.4.3
          securityContext:
            runAsUser: 0
          readinessProbe:
            exec:
              command:
              - cat
              - /tmp/ready
            initialDelaySeconds: 5
            periodSeconds: 5
          env:
            - name: CLUSTER_JOIN_ADDR
              value: neuvector-svc-controller.neuvector
            - name: CLUSTER_ADVERTISED_ADDR
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
            - name: CLUSTER_BIND_ADDR
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
          volumeMounts:
            - mountPath: /etc/config
              name: config-volume
              readOnly: true
      terminationGracePeriodSeconds: 300
      restartPolicy: Always
      volumes:
        - name: config-volume
          projected:
            sources:
              - configMap:
                  name: neuvector-init
                  optional: true
              - secret:
                  name: neuvector-init
                  optional: true
              - secret:
                  name: neuvector-secret
                  optional: true

---

apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: neuvector-enforcer-pod
  namespace: neuvector
spec:
  selector:
    matchLabels:
      app: neuvector-enforcer-pod
  updateStrategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: neuvector-enforcer-pod
      annotations:
        container.apparmor.security.beta.kubernetes.io/neuvector-enforcer-pod: unconfined
      # Add the following for pre-v1.19
      # container.seccomp.security.alpha.kubernetes.io/neuvector-enforcer-pod: unconfined
    spec:
      tolerations:
        - effect: NoSchedule
          key: node-role.kubernetes.io/master
        - effect: NoSchedule
          key: node-role.kubernetes.io/control-plane
      hostPID: true
      serviceAccountName: enforcer
      serviceAccount: enforcer
      containers:
        - name: neuvector-enforcer-pod
          image: neuvector/enforcer:5.4.3
          securityContext:
            # the following two lines are required for k8s v1.19+. pls comment out both
 lines if version is pre-1.19. Otherwise, a validating data error message will show
            seccompProfile:
              type: Unconfined
            capabilities:
              add:
              - SYS_ADMIN
              - NET_ADMIN
              - SYS_PTRACE
              - IPC_LOCK
          env:
            - name: CLUSTER_JOIN_ADDR
              value: neuvector-svc-controller.neuvector
            - name: CLUSTER_ADVERTISED_ADDR
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
            - name: CLUSTER_BIND_ADDR
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
          volumeMounts:
            - mountPath: /lib/modules
              name: modules-vol
              readOnly: true
            - mountPath: /var/nv_debug
              name: nv-debug
              readOnly: false
      terminationGracePeriodSeconds: 1200
      restartPolicy: Always
      volumes:
        - name: modules-vol
          hostPath:
            path: /lib/modules
        - name: nv-debug
          hostPath:
            path: /var/nv_debug

---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: neuvector-scanner-pod
  namespace: neuvector
spec:
  selector:
    matchLabels:
      app: neuvector-scanner-pod
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  replicas: 2
  template:
    metadata:
      labels:
        app: neuvector-scanner-pod
    spec:
      serviceAccountName: scanner
      serviceAccount: scanner
      containers:
        - name: neuvector-scanner-pod
          image: neuvector/scanner:latest
          imagePullPolicy: Always
          env:
            - name: CLUSTER_JOIN_ADDR
              value: neuvector-svc-controller.neuvector
      restartPolicy: Always

---

apiVersion: batch/v1
kind: CronJob
metadata:
  name: neuvector-updater-pod
  namespace: neuvector
spec:
  schedule: "0 0 * * *"
  jobTemplate:
    spec:
      template:
        metadata:
          labels:
            app: neuvector-updater-pod
        spec:
          serviceAccountName: updater
          serviceAccount: updater
          containers:
          - name: neuvector-updater-pod
            image: neuvector/updater:latest
            imagePullPolicy: Always
            command:
```

```
          - TOKEN=`cat /var/run/secrets/kubernetes.io/serviceaccount/token`; /usr/
bin/curl -kv -X PATCH -H "Authorization:Bearer $TOKEN" -H "Content-Type:application/
strategic-merge-patch+json" -d '{"spec":{"template":{"metadata":{"annotations":
{"kubectl.kubernetes.io/restartedAt":"'`date +%Y-%m-%dT%H:%M:%S%z`'"}}}}}' 'https://
kubernetes.default/apis/apps/v1/namespaces/neuvector/deployments/neuvector-scanner-pod'
          restartPolicy: Never
```

3.5.2.8 PKS change

Note

PKS is field-tested and requires enabling privileged containers to the plan/tile, and changing the YAML hostPath as follows for All-in-One and Enforcer:

```
    hostPath:
          path: /var/vcap/sys/run/docker/docker.sock
```

3.6 Setting up SUSE Observability for SUSE AI

SUSE Observability provides comprehensive monitoring and insights into your infrastructure and applications. It enables efficient tracking of metrics, logs and traces, helping you maintain optimal performance and troubleshoot issues effectively. This procedure guides you through setting up SUSE Observability for the SUSE AI environment using the SUSE AI Observability Extension.

3.6.1 Deployment scenarios

You can deploy SUSE Observability and SUSE AI in two different ways:

- **Single-Cluster setup:** Both SUSE AI and SUSE Observability are installed in the same Kubernetes cluster. This is a simpler approach ideal for testing and proof-of-concept deployments. Communication between components can use internal cluster DNS.
- **Multi-Cluster setup:** SUSE AI and SUSE Observability are installed on separate, dedicated Kubernetes clusters. This setup is recommended for production environments because it isolates workloads. Communication requires exposing the SUSE Observability endpoints externally, for example, via an Ingress.

This section provides instructions for both scenarios.

3.6.2 Requirements

To set up SUSE Observability for SUSE AI, you need to meet the following requirements:

- Have access to SUSE Application Collection
- Have a valid SUSE AI subscription
- Have a valid license for SUSE Observability in SUSE Customer Center
- Instrument your applications for telemetry data acquisition with OpenTelemetry.

For details on how to collect traces and metrics from SUSE AI components and user-developed applications, refer to Monitoring SUSE AI with OpenTelemetry and SUSE Observability (https://documentation.suse.com/suse-ai/1.0/html/AI-monitoring/index.html). It includes configurations that are essential for full observability.

Important: SUSE Application Collection not instrumented by default

Applications from the SUSE Application Collection are not instrumented by default. If you want to monitor your AI applications, you need to follow the instrumentation guidelines that we provide in the document Monitoring SUSE AI with OpenTelemetry and SUSE Observability (https://documentation.suse.com/suse-ai/1.0/html/AI-monitoring/index.html).

3.6.3 Setup process overview

The following chart shows the high-level steps for the setup procedure. You will first set up the SUSE Observability cluster, then configure the SUSE AI cluster, and finally instrument your applications. Execute the steps in each column from left to right and top to bottom.

- *Blue steps* are related to Helm chart installations.
- *Gray steps* represent another type of interaction, such as coding.

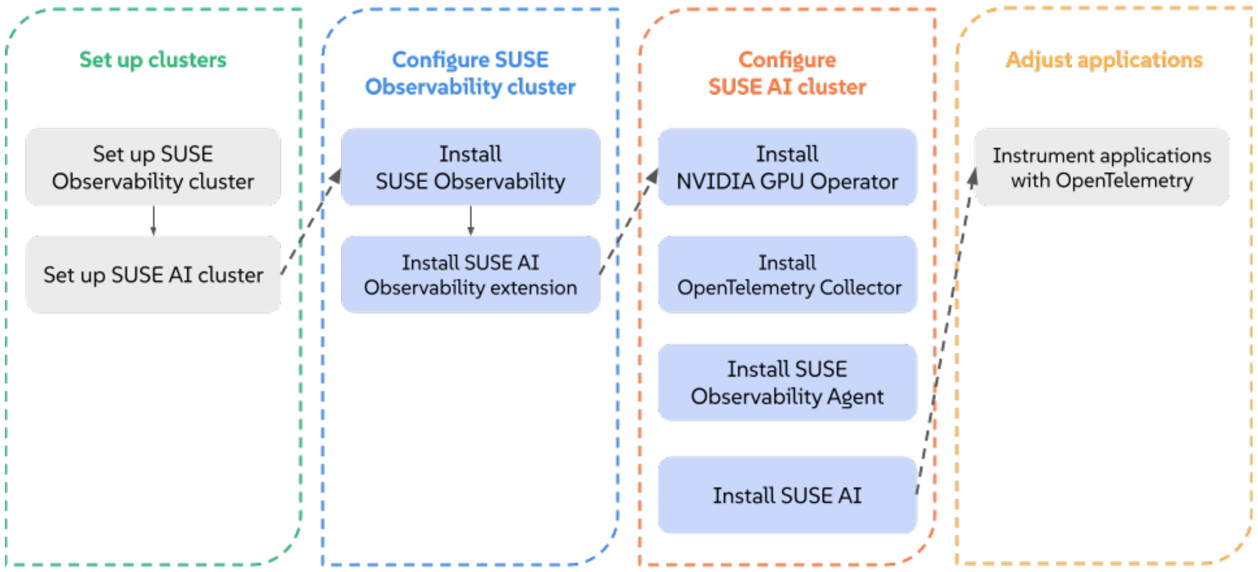

Set up clusters
Set up SUSE Observability cluster
Set up SUSE AI cluster
Configure SUSE Observability cluster
Install SUSE Observability
Install SUSE AI Observability extension
Configure SUSE AI cluster
Install NVIDIA GPU Operator
Install OpenTelemetry Collector
Install SUSE Observability Agent
Install SUSE AI
Adjust applications
Instrument applications with OpenTelemetry

FIGURE 30: HIGH-LEVEL OVERVIEW OF THE SUSE OBSERVABILITY SETUP

Tip: Setup clusters

You can create and configure Kubernetes clusters for SUSE AI and SUSE Observability as you prefer. If you are using SUSE Rancher Prime, check its documentation (https://ranchermanager.docs.rancher.com/how-to-guides/new-user-guides/kubernetes-clusters-in-rancher-setup) ↗. For testing purposes, you can even share one cluster for both deployments. You can skip instructions on setting up a specific cluster if you already have one configured.

The diagram below shows the result of the above steps. There are two clusters represented, one for the SUSE Observability workload and another one for SUSE AI. You may use identical setup or customize it for your environment.

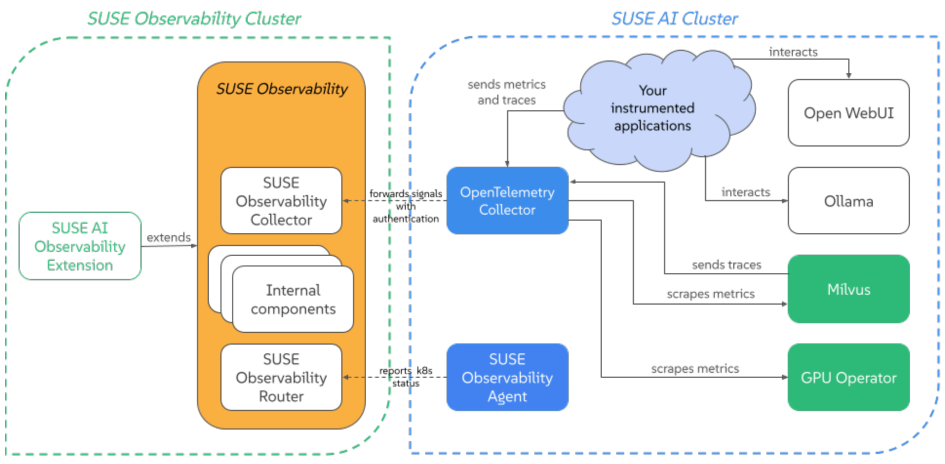


FIGURE 31: SEPARATE CLUSTERS FOR SUSE AI AND SUSE OBSERVABILITY

POINTS TO NOTICE

- You can install SUSE AI Observability Extension alongside SUSE Observability. It means that you can confidently use the internal Kubernetes DNS.
- SUSE Observability contains several components and the following two of them need to be accessible by the AI Cluster:
  - The Collector endpoint. Refer to Exposing SUSE Observability outside of the cluster (https://documentation.suse.com/cloudnative/suse-observability/next/en/setup/install-stackstate/kubernetes_openshift/ingress.html) or Self-hosted SUSE Observability (https://documentation.suse.com/cloudnative/suse-observability/next/en/setup/otel/otlp-apis.html#_self_hosted_suse_observability) for details about exposing it.
  - The SUSE Observability API. Refer to Exposing SUSE Observability outside of the cluster (https://documentation.suse.com/cloudnative/suse-observability/next/en/setup/install-stackstate/kubernetes_openshift/ingress.html) for details about exposing it.
  - Milvus metrics and traces can be scraped by the OpenTelemetry Collector with simple configurations, provided below. The same is true for GPU metrics.
  - To get information from Open WebUI, Ollama or vLLM, you must have a specific instrumentation set. It can be an application instrumented with the OpenLIT SDK or other form of instrumentation following the same patterns.

Important

Remember that in multi-cluster setups, it is **critical** to properly expose your endpoints. Configure TLS, be careful with the configuration, and make sure to provide the right keys and tokens. More details are provided in the respective instructions.

3.6.4 Setting up the SUSE Observability cluster

This initial step is identical for both single-cluster and multi-cluster deployments.

1. **Install SUSE Observability.** You can follow the official SUSE Observability installation documentation (https://documentation.suse.com/cloudnative/suse-observability/latest/en/classic.html) for all installation instructions. Remember to expose your APIs (https://documentation.suse.com/cloudnative/suse-observability/latest/en/setup/install-stackstate/kubernetes_openshift/ingress.html) and collector endpoints to your SUSE AI cluster.

   Important: Multi-cluster setup

   For multi-cluster setups, you must expose the SUSE Observability API and collector endpoints so that the SUSE AI cluster can reach them. Refer to the guide on exposing SUSE Observability outside of the cluster (https://documentation.suse.com/cloudnative/suse-observability/latest/en/setup/install-stackstate/kubernetes_openshift/ingress.html).

2. **Install the SUSE Observability extension.** Create a new Helm values file named `genai_values.yaml`. Before creating the file, review the placeholders below.

   SUSE_OBSERVABILITY_API_URL

   The URL of the SUSE Observability API. For multi-cluster deployments, this is the external URL. For single-cluster deployments, this can be the internal service URL. Example: `http://suse-observability-api.your-domain.com`

   SUSE_OBSERVABILITY_API_KEY

   The API key from the `baseConfig_values.yaml` file used during the SUSE Observability installation.

SUSE_OBSERVABILITY_API_TOKEN_TYPE

Can be `api` for a token from the Web UI or `service` for a Service Token.

SUSE_OBSERVABILITY_TOKEN

The API or Service token itself.

OBSERVED_SERVER_NAME

The name of the cluster to observe. It must match the name used in the Kubernetes StackPack configuration. Example: `suse-ai-cluster`.

a. Create the `genai_values.yaml` file with the following content:

```
global:
  imagePullSecrets:
  - application-collection 1
  ifdef::deployment_airgap[]
  imageRegistry: <LOCAL_DOCKER_REGISTRY_URL>:5043
  endif::[]
serverUrl: <SUSE_OBSERVABILITY_API_URL>
apiKey: <SUSE_OBSERVABILITY_API_KEY>
tokenType: <SUSE_OBSERVABILITY_API_TOKEN_TYPE>
apiToken: <SUSE_OBSERVABILITY_TOKEN>
clusterName: <OBSERVED_SERVER_NAME>
```

1 Instructs Helm to use credentials from the SUSE Application Collection. For instructions on how to configure the image pull secrets for the SUSE Application Collection, refer to the official documentation (https://docs.apps.rancher.io/get-started/authentication/).

b. Run the install command.

```
> helm upgrade --install ai-obs \
  oci://dp.apps.rancher.io/charts/suse-ai-observability-extension \
  -f genai_values.yaml --namespace so-extensions --create-namespace
```

Note: Self-signed certificates not supported

Self-signed certificates are not supported. Consider running the extension in the same cluster as SUSE Observability and then use the internal Kubernetes address.

After the installation is complete, a new menu called *GenAI* is added to the Web interface and also a Kubernetes cron job is created that synchronizes the topology view with the components found in the SUSE AI cluster.

3. **Verify SUSE Observability extension.** After the installation, you can verify that a new lateral menu appears:

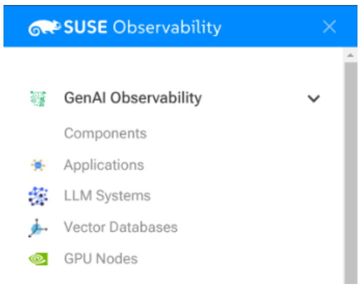


FIGURE 32: NEW GENAI OBSERVABILITY MENU ITEM

3.6.5 Setting up the SUSE AI cluster

Follow the instructions for your deployment scenario.

Single-cluster deployment

In this setup, the SUSE AI components are installed in the same cluster as SUSE Observability and can communicate using internal service DNS.

Multi-cluster deployment

In this setup, the SUSE AI cluster is separate. Communication relies on externally exposed endpoints of the SUSE Observability cluster.

The difference between deployment scenarios affects the **OTEL Collector exporter configuration** and the **SUSE Observability Agent URL** as described in the following list.

SUSE_OBSERVABILITY_API_URL

The URL of the SUSE Observability API.

Single-cluster example: http://suse-observability-otel-collector.suse-observability.svc.cluster.local:4317

Multi-cluster example: https://suse-observability-api.your-domain.com

SUSE_OBSERVABILITY_COLLECTOR_ENDPOINT

The endpoint of the SUSE Observability Collector.

Single-cluster example: http://suse-observability-router.suse-observability.svc.cluster.local:8080/receiver/stsAgent

Multi-cluster example: https://suse-observability-router.your-domain.com/receiver/stsAgent

1. **Install NVIDIA GPU Operator.** Follow the instructions in https://documentation.suse.com/cloudnative/rke2/latest/en/advanced.html#_deploy_nvidia_operator.

2. **Install OpenTelemetry collector.** Create a secret with your SUSE Observability API key in the namespace where you want to install the collector. Retrieve the API key using the Web UI or from the `baseConfig_values.yaml` file that you used during the SUSE Observability installation. If the namespace does not exist yet, create it.

   ```
   kubectl create namespace observability
   kubectl create secret generic open-telemetry-collector \
     --namespace observability \
     --from-literal=API_KEY='<SUSE_OBSERVABILITY_API_KEY>'
   ```

   Create a new file named `otel-values.yaml` with the following content.

   ```
   global:
     imagePullSecrets:
     - application-collection
     ifdef::deployment_airgap[]
     repository: <LOCAL_DOCKER_REGISTRY_URL>:5043/opentelemetry-collector-k8s
     endif::[]
   extraEnvsFrom:
     - secretRef:
         name: open-telemetry-collector
   mode: deployment
   ports:
     metrics:
       enabled: true
   presets:
     kubernetesAttributes:
       enabled: true
       extractAllPodLabels: true
   config:
     receivers:
       prometheus:
         config:
           scrape_configs:
             - job_name: 'gpu-metrics'
               scrape_interval: 10s
               scheme: http
   ```

```
            kubernetes_sd_configs:
              - role: endpoints
                namespaces:
                  names:
                    - gpu-operator
          - job_name: 'milvus'
            scrape_interval: 15s
            metrics_path: '/metrics'
            static_configs:
              - targets:
 ['<MILVUS_SERVICE_NAME>.<SUSE_AI_NAMESPACE>.svc.cluster.local:9091'] (1)
          - job_name: 'vllm'
            scrape_interval: 10s
            scheme: http
            kubernetes_sd_configs:
              - role: service
            relabel_configs:
              - source_labels: [__meta_kubernetes_namespace]
                action: keep
                regex: '<VLLM_NAMESPACE>' (2)

              - source_labels: [__meta_kubernetes_service_name]
                action: keep
                regex: '.*<VLLM_RELEASE_NAME>.*' (3)
  exporters:
    otlp:
      endpoint: https://<OPEN_TELEMETRY_COLLECTOR_NAME>.suse-
observability.svc.cluster.local:4317 (4)
      headers:
        Authorization: "SUSEObservability ${env:API_KEY}"
      tls:
        insecure: true
  processors:
    tail_sampling:
      decision_wait: 10s
      policies:
      - name: rate-limited-composite
        type: composite
        composite:
          max_total_spans_per_second: 500
          policy_order: [errors, slow-traces, rest]
          composite_sub_policy:
          - name: errors
            type: status_code
            status_code:
              status_codes: [ ERROR ]
          - name: slow-traces
```

```
     type: latency
     latency:
       threshold_ms: 1000
    - name: rest
     type: always_sample
    rate_allocation:
    - policy: errors
     percent: 33
    - policy: slow-traces
     percent: 33
    - policy: rest
     percent: 34
 resource:
  attributes:
  - key: k8s.cluster.name
   action: upsert
   value: <CLUSTER_NAME> 5
  - key: service.instance.id
   from_attribute: k8s.pod.uid
   action: insert
 filter/dropMissingK8sAttributes:
  error_mode: ignore
  traces:
   span:
    - resource.attributes["k8s.node.name"] == nil
    - resource.attributes["k8s.pod.uid"] == nil
    - resource.attributes["k8s.namespace.name"] == nil
    - resource.attributes["k8s.pod.name"] == nil
connectors:
 spanmetrics:
  metrics_expiration: 5m
  namespace: otel_span
 routing/traces:
  error_mode: ignore
  table:
  - statement: route()
   pipelines: [traces/sampling, traces/spanmetrics]
service:
 extensions:
  - health_check
 pipelines:
  traces:
   receivers: [otlp, jaeger]
   processors: [filter/dropMissingK8sAttributes, memory_limiter, resource]
   exporters: [routing/traces]
  traces/spanmetrics:
   receivers: [routing/traces]
```

```
        processors: []
        exporters: [spanmetrics]
      traces/sampling:
        receivers: [routing/traces]
        processors: [tail_sampling, batch]
        exporters: [debug, otlp]
      metrics:
        receivers: [otlp, spanmetrics, prometheus]
        processors: [memory_limiter, resource, batch]
        exporters: [debug, otlp]
```

1. Configure the Milvus service and namespace for the Prometheus scraper. Because Milvus will be installed in subsequent steps, you can return to this step and edit the endpoint if necessary.
2. Update to match the values in the vLLM deployment section.
3. Update to match the values in the vLLM deployment section.
4. Set the exporter to your exposed SUSE Observability collector. Remember that the value can be distinct, depending on the deployment pattern. For production usage, we recommend using TLS communication.
5. Replace `<CLUSTER_NAME>` with the cluster's name.

Finally, run the installation command.

```
> helm upgrade --install opentelemetry-collector \
  oci://dp.apps.rancher.io/charts/opentelemetry-collector \
  -f otel-values.yaml --namespace observability
```

Verify the installation by checking the existence of a new deployment and service in the observability namespace.

3. The GPU metrics scraper that we configure in the OTEL Collector requires custom RBAC rules. Create a file named `otel-rbac.yaml` with the following content:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: suse-observability-otel-scraper
rules:
  - apiGroups:
      - ""
    resources:
      - services
      - endpoints
    verbs:
      - list
      - watch

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: suse-observability-otel-scraper
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: suse-observability-otel-scraper
subjects:
  - kind: ServiceAccount
    name: opentelemetry-collector
    namespace: observability
```

Then apply the configuration by running the following command.

```
> kubectl apply -n gpu-operator -f otel-rbac.yaml
```

4. **Install the SUSE Observability Agent.**

```
> helm upgrade --install \
  --namespace suse-observability --create-namespace \
  --set-string 'stackstate.apiKey'='<YOUR_API_KEY>' \ ❶
  --set-string 'stackstate.cluster.name'='<CLUSTER_NAME>' \ ❷
  --set-string 'stackstate.url'='http://suse-observability-router.suse-
observability.svc.cluster.local:8080/receiver/stsAgent' \ ❸
  --set 'nodeAgent.skipKubeletTLSVerify'=true suse-observability-agent \
  suse-observability/suse-observability-agent
```

❶ Retrieve the API key using the Web UI or from the `baseConfig_values.yaml` file that you used during the SUSE Observability installation.

❷ Replace `<CLUSTER_NAME>` with the cluster's name.

❸ Replace with your SUSE Observability server URL.

5. **Install SUSE AI.** Refer to *Section 4, "Installing applications from AI Library"* for the complete procedure.

Warning: SUSE Observability version 2.6.2 and above

With SUSE Observability version 2.6.2, a change of the standard behavior broke the vLLM monitoring performed by the extension. To fix it, update `otel-values.yaml` to include the following additions. No changes are required for people using SUSE Observability version 2.6.1 and below.

- Add a new processor.

```
config:
  processors:
    ... # same as before
    transform:
      metric_statements:
        - context: metric
          statements:
            - replace_pattern(name, "^vllm:", "vllm_")
```

- Modify the metrics pipeline to perform the transformation defined above:

```
config:
  service:
    pipelines:
      ... # same as before
      metrics:
        receivers: [otlp, spanmetrics, prometheus]
        processors: [transform, memory_limiter, resource, batch]
        exporters: [debug, otlp]
```

3.6.6 Instrument applications

Instrumentation is the act of configuring your applications for telemetry data acquisition. Our stack employs OpenTelemetry standards as a vendor-neutral and open base for our telemetry. For a comprehensive guide on how to set up your instrumentation, please refer to Monitoring SUSE AI with OpenTelemetry and SUSE Observability (https://documentation.suse.com/suse-ai/1.0/html/AI-monitoring/index.html)↗.

By following the instructions in the document referenced above, you will be able to retrieve all relevant telemetry data from Open WebUI, Ollama, Milvus and vLLM by simply applying specific configuration to their Helm chart values. You can find links for advanced use cases (auto-instrumentation with the OTEL Operator) at the end of the document.

4 Installing applications from AI Library

SUSE AI is delivered as a set of components that you can combine to meet specific use cases. To enable the full integrated stack, you need to deploy multiple applications in sequence. Applications with the fewest dependencies must be installed first, followed by dependent applications once their required dependencies are in place within the cluster.

You can either install required AI Library components manually using their Helm charts, or use SUSE Deployer to include all the dependencies in one step.

4.1 Installation procedure

This procedure includes steps to install AI Library applications.

1. Purchase the SUSE AI entitlement. It is a separate entitlement from SUSE Rancher Prime.
2. Access SUSE AI via the SUSE Application Collection at https://apps.rancher.io/ ↗ to perform the check for the SUSE AI entitlement.
3. If the entitlement check is successful, you are given access to the SUSE AI-related Helm charts and container images, and can deploy directly from the SUSE Application Collection.
4. Visit the SUSE Application Collection, sign in and get the user access token as described in https://docs.apps.rancher.io/get-started/authentication/ ↗.
5. Create a Kubernetes namespace if it does not already exist. The steps in this procedure assume that all containers are deployed into the same namespace referred to as `SUSE_AI_NAMESPACE`. Replace its name to match your preferences.

   ```
   > kubectl create namespace <SUSE_AI_NAMESPACE>
   ```

6. Create the SUSE Application Collection secret.

   ```
   > kubectl create secret docker-registry application-collection \
     --docker-server=dp.apps.rancher.io \
     --docker-username=<APPCO_USERNAME> \
     --docker-password=<APPCO_USER_TOKEN> \
     -n <SUSE_AI_NAMESPACE>
   ```

7. Log in to the Helm registry.

   ```
   > helm registry login dp.apps.rancher.io/charts \
     -u <APPCO_USERNAME> \
   ```

```
  -p <APPCO_USER_TOKEN>
```

8. Install cert-manager as described in *Section 4.2, "Installing cert-manager"*.
9. Install AI Library components. You can either install each component separately, or use the SUSE Deployer chart to install the components together as described in *Section 4.11, "Installing AI Library components using SUSE Deployer"*.
   a. Install an application with vector database capabilities. Open WebUI supports either OpenSearch (*Section 4.3, "Installing OpenSearch"*) or Milvus (*Section 4.4, "Installing Milvus"*).
   b. *(Optional)* Install Ollama as described in *Section 4.5, "Installing Ollama"*.
   c. Install Open WebUI as described in *Section 4.6, "Installing Open WebUI"*.
   d. Install vLLM as described in *Section 4.7, "Installing vLLM"*.
   e. Install mcpo as described in *Section 4.8, "Installing mcpo"*.

4.2 Installing cert-manager

cert-manager is an extensible X.509 certificate controller for Kubernetes workloads. It supports certificates from popular public issuers as well as private issuers. cert-manager ensures that the certificates are valid and up-to-date, and attempts to renew certificates at a configured time before expiry.

In previous releases, cert-manager was automatically installed together with Open WebUI. Currently, cert-manager is no longer part of the Open WebUI Helm chart and you need to install it separately.

4.2.1 Details about the cert-manager application

Before deploying cert-manager, it is important to know more about the supported configurations and documentation. The following command provides the corresponding details:

```
helm show values oci://dp.apps.rancher.io/charts/cert-manager
```

Alternatively, you can also refer to the cert-manager Helm chart page on the SUSE Application Collection site at https://apps.rancher.io/applications/cert-manager ↗. It contains available versions and the link to pull the cert-manager container image.

4.2.2 cert-manager installation procedure

Tip

Before the installation, you need to get user access to the SUSE Application Collection, create a Kubernetes namespace, and log in to the Helm registry as described in *Section 4.1, “Installation procedure”*.

- Install the cert-manager chart.

```
> helm upgrade --install cert-manager \
  oci://dp.apps.rancher.io/charts/cert-manager \
  -n <CERT_MANAGER_NAMESPACE> \
  --set crds.enabled=true \
  --set 'global.imagePullSecrets[0].name'=application-collection
```

4.2.3 Upgrading cert-manager

To upgrade cert-manager to a specific new version, run the following command:

```
> helm upgrade --install cert-manager \
  oci://dp.apps.rancher.io/charts/cert-manager \
  -n <CERT_MANAGER_NAMESPACE> \
  --version <VERSION_NUMBER>
```

To upgrade cert-manager to the latest version, run the following command:

```
> helm upgrade --install cert-manager \
  oci://dp.apps.rancher.io/charts/cert-manager \
  -n <CERT_MANAGER_NAMESPACE>
```

4.2.4 Uninstalling cert-manager

To uninstall cert-manager, run the following command:

```
> helm uninstall cert-manager -n <CERT_MANAGER_NAMESPACE>
```

4.3 Installing OpenSearch

OpenSearch is a community-driven, open source search and analytics suite. It is used to search, visualize and analyze data. OpenSearch consists of a data store and search engine (OpenSearch), a visualization and user interface (OpenSearch Dashboards), and a server-side data collector (Data Prepper). Its functionality can be extended by plug-ins that enhance features like search, analytics, observability, security or machine learning.

4.3.1 Details about the OpenSearch application

Before deploying OpenSearch, it is important to know more about the supported configurations and documentation. The following command provides the corresponding details:

```
helm show values oci://dp.apps.rancher.io/charts/opensearch
```

Alternatively, you can also refer to the OpenSearch Helm chart page on the SUSE Application Collection site at https://apps.rancher.io/applications/opensearch. It contains OpenSearch dependencies, available versions and the link to pull the OpenSearch container image.

4.3.2 OpenSearch installation procedure

OpenSearch can operate as a single-node or multi-node cluster. The following override file examples outline both scenarios.

Important

Both scenarios require increasing the value of the `vm.max_map_count` to at least 262144. To check the current value, run the following command:

```
> cat /proc/sys/vm/max_map_count
```

To increase the value, add the following to `/etc/sysctl.conf`:

```
vm.max_map_count=262144
```

Then run `sudo sysctl -p` to reload.

Tip

Before the installation, you need to get user access to the SUSE Application Collection, create a Kubernetes namespace, and log in to the Helm registry as described in *Section 4.1, “Installation procedure”*.

1. Create an `opensearch_custom_overrides.yaml` file to override the default values of the Helm chart.
   For a single-node cluster, use the following template file:

   ```
   # opensearch_custom_overrides.yaml
   global:
     imagePullSecrets:
       - application-collection
   singleNode: true
   replicas: 1
   persistence:
     enabled: true
     storageClass: local-path

   extraEnvs:
     - name: OPENSEARCH_INITIAL_ADMIN_PASSWORD
       value: "MySecurePass123"

   service:
     type: NodePort
     httpPort: 9200
     transportPort: 9300
     metricsPort: 9600

   resources:
     limits:
       memory: "6Gi"
       cpu: "2"

   config:
     opensearch.yml: |
       plugins.security.disabled: true
   ```

   For a multi-node cluster, use the following template file:

   ```
   # opensearch_custom_overrides.yaml
   global:
     imagePullSecrets:
   ```

```
    - application-collection
singleNode: false
replicas: 3
persistence:
  enabled: true
  storageClass: local-path

extraEnvs:
  - name: OPENSEARCH_INITIAL_ADMIN_PASSWORD
    value: "MySecurePass123"
  - name: ES_JAVA_OPTS
    value: "-Xms3g -Xmx3g"

service:
  type: NodePort
  httpPort: 9200
  transportPort: 9300
  metricsPort: 9600

resources:
  limits:
    memory: "6Gi"
    cpu: "2"
  requests:
    memory: "4Gi"
    cpu: "1"

startupProbe:
  tcpSocket:
    port: 9200
  initialDelaySeconds: 60
  periodSeconds: 10
  timeoutSeconds: 5
  failureThreshold: 12

readinessProbe:
  tcpSocket:
    port: 9200
  initialDelaySeconds: 60
  periodSeconds: 10
  timeoutSeconds: 5
  failureThreshold: 6

livenessProbe:
  tcpSocket:
    port: 9200
  initialDelaySeconds: 120
  periodSeconds: 20
  timeoutSeconds: 5
  failureThreshold: 3

config:
  opensearch.yml: |
    plugins.security.disabled: true
```

2. After saving the override file as `opensearch_custom_overrides.yaml`, apply its configuration with the following command.

```
> helm upgrade --install \
  opensearch oci://dp.apps.rancher.io/charts/opensearch \
  -n <SUSE_AI_NAMESPACE> \
  -f <opensearch_custom_overrides.yaml>
```

3. Check that the pods and services are running.

```
> kubectl get pods -n <SUSE_AI_NAMESPACE> | grep "opensearch"
opensearch-cluster-master-0           1/1     Running   0          5h34m
```

A multi-node cluster configuration shows that the replicas are distributed across multiple nodes.

```
> kubectl get pods -n <SUSE_AI_NAMESPACE> | grep "opensearch"
opensearch-cluster-master-0 1/1 Running 0 2m30s 10.42.1.32 mgmt-rancher-wkrgpu1
opensearch-cluster-master-1 1/1 Running 0 2m30s 10.42.1.33 mgmt-rancher-wkrgpu1
opensearch-cluster-master-2 1/1 Running 0 2m30s 10.42.0.27 mgmt-rancher
```

4.3.3 Integrating OpenSearch with Open WebUI

To integrate OpenSearch with Open WebUI, follow these steps:

1. Edit the override file for Open WebUI, `owui_custom_overrides.yaml`, and update the `extraEnvVars` section as follows.

   - Change the `VECTOR_DB` value to `opensearch`.
   - Remove the `MILVUS_URI` variable.
   - Add all the OpenSearch-related environment variables.

   ```
   extraEnvVars:
   - name: DEFAULT_MODELS
   ```

```
  value: "gemma:2b"
- name: DEFAULT_USER_ROLE
  value: "pending"
- name: ENABLE_SIGNUP
  value: "true"
- name: GLOBAL_LOG_LEVEL
  value: INFO
- name: RAG_EMBEDDING_MODEL
  value: "sentence-transformers/all-MiniLM-L6-v2"
- name: INSTALL_NLTK_DATASETS
  value: "true"
- name: VECTOR_DB
  value: "opensearch"
#- name: MILVUS_URI
#  value: http://milvus.<SUSE_AI_NAMESPACE>.svc.cluster.local:19530
- name: OPENAI_API_KEY
  value: "0p3n-w3bu!"
- name: OPENSEARCH_SSL
  value: "false"
- name: OPENSEARCH_URI
  value: http://opensearch-cluster-
master.<SUSE_AI_NAMESPACE>.svc.cluster.local:9200
- name: OPENSEARCH_USERNAME
  value: admin
- name: OPENSEARCH_PASSWORD
  value: MySecurePass123
- name: OPENSEARCH_CERT_VERIFY
  value: "false"
```

2. Redeploy Open WebUI.

```
> helm upgrade --install \
  open-webui oci://dp.apps.rancher.io/charts/open-webui \
  -n <SUSE_AI_NAMESPACE> \
  -f <owui_custom_overrides.yaml>
```

3. Verify that `VECTOR_DB` is set to `opensearch`.

```
> kubectl exec -it open-webui-0 -n <SUSE_AI_NAMESPACE> \
  -- sh -c 'echo "VECTOR_DB=$VECTOR_DB"'

Defaulted container "open-webui" out of: open-webui, copy-app-data (init)
VECTOR_DB=opensearch
```

4.3.4 Upgrading OpenSearch

The OpenSearch chart receives application updates and updates of the Helm chart templates. New versions may include changes that require manual steps. These steps are listed in the corresponding `README` file. All OpenSearch dependencies are updated automatically during an OpenSearch upgrade.

To upgrade OpenSearch, identify the new version number and run the following command below:

```
> helm upgrade --install \
  opensearch oci://dp.apps.rancher.io/charts/opensearch \
  -n <SUSE_AI_NAMESPACE> \
  --version <VERSION_NUMBER> \
  -f <opensearch_custom_overrides.yaml>
```

Tip

If you omit the `--version` option, OpenSearch gets upgraded to the latest available version.

4.3.5 Uninstalling OpenSearch

To uninstall OpenSearch, run the following command:

```
> helm uninstall opensearch -n <SUSE_AI_NAMESPACE>
```

4.4 Installing Milvus

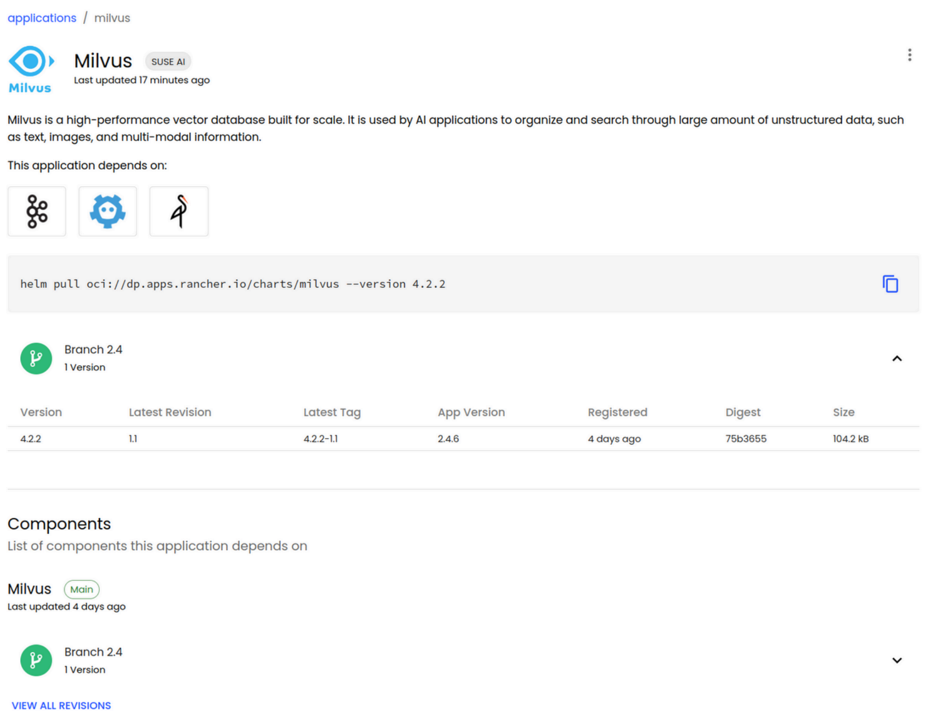
Milvus is a scalable, high-performance vector database designed for AI applications. It enables efficient organization and searching of massive unstructured datasets, including text, images and multi-modal content. This procedure walks you through the installation of Milvus and its dependencies.

4.4.1 Details about the Milvus application

Before deploying Milvus, it is important to know more about the supported configurations and documentation. The following command provides the corresponding details:

```
helm show values oci://dp.apps.rancher.io/charts/milvus
```

Alternatively, you can also refer to the Milvus Helm chart page on the SUSE Application Collection site at https://apps.rancher.io/applications/milvus. It contains Milvus dependencies, available versions and the link to pull the Milvus container image.



Milvus is a high-performance vector database built for scale. It is used by AI applications to organize and search through large amount of unstructured data, such as text, images, and multi-modal information.

This application depends on:

```
helm pull oci://dp.apps.rancher.io/charts/milvus --version 4.2.2
```

Branch 2.4
1 Version

Version	Latest Revision	Latest Tag	App Version	Registered	Digest	Size
4.2.2	1.1	4.2.2-1.1	2.4.6	4 days ago	75b3655	104.2 kB

Components
List of components this application depends on

Milvus Main
Last updated 4 days ago

Branch 2.4
1 Version

FIGURE 33: MILVUS PAGE IN THE SUSE APPLICATION COLLECTION

4.4.2 Milvus installation procedure

Tip

Before the installation, you need to get user access to the SUSE Application Collection, create a Kubernetes namespace, and log in to the Helm registry as described in *Section 4.1, “Installation procedure”*.

1. When installed as part of SUSE AI, Milvus depends on etcd, MinIO and Apache Kafka. Because the Milvus chart uses a non-default configuration, create an override file `milvus_custom_overrides.yaml` with the following content.

   Tip

   As a template, you can download the Milvus Helm chart that includes the `values.yaml` file with the default configuration by running the following command:

   ```
   > helm pull oci://dp.apps.rancher.io/charts/milvus --version 4.2.2
   ```

   ```
   global:
     imagePullSecrets:
     - application-collection
     ifdef::deployment_airgap[]
     imageRegistry: <LOCAL_DOCKER_REGISTRY_URL>:5043
     endif::[]
   cluster:
     enabled: True
   standalone:
     persistence:
       persistentVolumeClaim:
         storageClassName: "local-path"
   etcd:
     replicaCount: 1
     persistence:
       storageClassName: "local-path"
   minio:
     mode: distributed
     replicas: 4
     rootUser: "admin"
     rootPassword: "adminminio"
     persistence:
       storageClass: "local-path"
     resources:
       requests:
         memory: 1024Mi
   kafka:
     enabled: true
     name: kafka
     replicaCount: 3
     broker:
       enabled: true
   ```

```
  cluster:
    listeners:
      client:
        protocol: 'PLAINTEXT'
      controller:
        protocol: 'PLAINTEXT'
  persistence:
    enabled: true
    annotations: {}
    labels: {}
    existingClaim: ""
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 8Gi
    storageClassName: "local-path"
extraConfigFiles: (1)
  user.yaml: |+
    trace:
      exporter: jaeger
      sampleFraction: 1
      jaeger:
        url: "http://opentelemetry-collector.observability.svc.cluster.local:14268/
api/traces" (2)
```

(1) The `extraConfigFiles` section is optional, required only to receive telemetry data from Open WebUI.

(2) The URL of the OpenTelemetry Collector installed by the user.

Tip

The above example uses local storage. For production environments, we recommend using an enterprise class storage solution such as SUSE Storage in which case the `storageClassName` option must be set to `longhorn`.

2. Install the Milvus Helm chart using the `milvus_custom_overrides.yaml` override file.

```
> helm upgrade --install \
  milvus oci://dp.apps.rancher.io/charts/milvus \
  -n <SUSE_AI_NAMESPACE> \
  --version 4.2.2 -f <milvus_custom_overrides.yaml>
```

4.4.2.1 Using Apache Kafka with SUSE Storage

When Milvus is deployed in cluster mode, it uses Apache Kafka as a message queue. If Apache Kafka uses SUSE Storage as a storage back-end, you need to create an XFS storage class and make it available for the Apache Kafka deployment. Otherwise deploying Apache Kafka with a storage class of an Ext4 file system fails with the following error:

```
"Found directory /mnt/kafka/logs/lost+found, 'lost+found' is not
  in the form of topic-partition or topic-partition.uniqueId-delete
  (if marked for deletion)"
```

To introduce the XFS storage class, follow these steps:

1. Create a file named `longhorn-xfs.yaml` with the following content:

   ```
   apiVersion: storage.k8s.io/v1
   kind: StorageClass
   metadata:
     name: longhorn-xfs
   provisioner: driver.longhorn.io
   allowVolumeExpansion: true
   reclaimPolicy: Delete
   volumeBindingMode: Immediate
   parameters:
     numberOfReplicas: "3"
     staleReplicaTimeout: "30"
     fromBackup: ""
     fsType: "xfs"
     dataLocality: "disabled"
     unmapMarkSnapChainRemoved: "ignored"
   ```

2. Create the new storage class using the `kubectl` command.

   ```
   > kubectl apply -f longhorn-xfs.yaml
   ```

3. Update the Milvus overrides YAML file to reference the Apache Kafka storage class, as in the following example:

   ```
     [...]
       kafka:
       enabled: true
       persistence:
         storageClassName: longhorn-xfs
   ```

4.4.3 Upgrading Milvus

The Milvus chart receives application updates and updates of the Helm chart templates. New versions may include changes that require manual steps. These steps are listed in the corresponding README file. All Milvus dependencies are updated automatically during Milvus upgrade.

To upgrade Milvus, identify the new version number and run the following command below:

```
> helm upgrade --install \
  milvus oci://dp.apps.rancher.io/charts/milvus \
  -n <SUSE_AI_NAMESPACE> \
  --version <VERSION_NUMBER> \
  -f <milvus_custom_overrides.yaml>
```

4.4.4 Uninstalling Milvus

To uninstall Milvus, run the following command:

```
> helm uninstall milvus -n <SUSE_AI_NAMESPACE>
```

4.5 Installing Ollama

Ollama is a tool for running and managing language models locally on your computer. It offers a simple interface to download, run and interact with models without relying on cloud resources.

Tip

When installing SUSE AI, Ollama is installed by the Open WebUI installation by default. If you decide to install Ollama separately, disable its installation during the installation of Open WebUI as outlined in *Example 6, "Open WebUI override file with Ollama installed separately"*.

4.5.1 Details about the Ollama application

Before deploying Ollama, it is important to know more about the supported configurations and documentation. The following command provides the corresponding details:

```
helm show values oci://dp.apps.rancher.io/charts/ollama
```

Alternatively, you can also refer to the Ollama Helm chart page on the SUSE Application Collection site at https://apps.rancher.io/applications/ollama. It contains the available versions and a link to pull the Ollama container image.

4.5.2 Ollama installation procedure

Tip

Before the installation, you need to get user access to the SUSE Application Collection, create a Kubernetes namespace, and log in to the Helm registry as described in *Section 4.1, "Installation procedure"*.

1. Create the `ollama_custom_overrides.yaml` file to override the values of the parent Helm chart. Refer to *Section 4.5.5, "Values for the Ollama Helm chart"* for more details.
2. Install the Ollama Helm chart using the `ollama-custom-overrides.yaml` override file.

   ```
   > helm upgrade \
     --install ollama oci://dp.apps.rancher.io/charts/ollama \
     -n <SUSE_AI_NAMESPACE> \
     -f ollama_custom_overrides.yaml
   ```

   #### Tip: Hugging Face models

   Models downloaded from Hugging Face need to be converted before they can be used by Ollama. Refer to https://github.com/ollama/ollama/blob/main/docs/import.md for more details.

4.5.3 Uninstalling Ollama

To uninstall Ollama, run the following command:

```
> helm uninstall ollama -n <SUSE_AI_NAMESPACE>
```

4.5.4 Upgrading Ollama

You can upgrade Ollama to a specific version by running the following command:

```
> helm upgrade ollama oci://dp.apps.rancher.io/charts/ollama \
  -n <SUSE_AI_NAMESPACE> \
  --version <OLLAMA_VERSION_NUMBER> -f <ollama_custom_overrides.yaml>
```

If you omit the `--version` option, Ollama gets upgraded to the latest available version.

4.5.4.1 Upgrading from version 0.x.x to 1.x.x

The version 1.x.x introduces the ability to load models in memory at startup. To reflect this, change `ollama.models` to `ollama.models.pull` in the Ollama Helm chart to avoid errors before upgrading, for example:

EXAMPLE 1: OLLAMA HELM CHART VERSION 0.X.X

```
[...]
ollama:
  models:
    - "gemma:2b"
    - "llama3.1"
```

EXAMPLE 2: OLLAMA HELM CHART VERSION 1.X.X

```
[...]
ollama:
  models:
    pull:
      - "gemma:2b"
      - "llama3.1"
```

Without this change you may experience the following error when trying to upgrade from 0.x.x to 1.x.x.

```
coalesce.go:286: warning: cannot overwrite table with non table for
ollama.ollama.models (map[pull:[] run:[]])
Error: UPGRADE FAILED: template: ollama/templates/deployment.yaml:145:27:
executing "ollama/templates/deployment.yaml" at <.Values.ollama.models.pull>:
can't evaluate field pull in type interface {}
```

4.5.5 Values for the Ollama Helm chart

To override the default values during the Helm chart installation or update, you can create an override YAML file with custom values. Then, apply these values by specifying the path to the override file with the `-f` option of the `helm` command. Remember to replace <SUSE_AI_NAMESPACE> with your Kubernetes namespace.

Important: GPU section

Ollama can run optimized for NVIDIA GPUs if the following conditions are fulfilled:

- The NVIDIA driver and NVIDIA GPU Operator are installed as described in Installing NVIDIA GPU Drivers on SLES (https://documentation.suse.com/suse-ai/1.0/html/NVIDIA-GPU-driver-on-SLES/index.html) or Installing NVIDIA GPU Drivers on SUSE Linux Micro (https://documentation.suse.com/suse-ai/1.0/html/NVIDIA-GPU-driver-on-SL-Micro/index.html).
- The workloads are set to run on NVIDIA-enabled nodes as described in https://documentation.suse.com/suse-ai/1.0/html/AI-deployment-intro/index.html#ai-gpu-nodes-assigning.

If you do not want to use the NVIDIA GPU, remove the `gpu` section from `ollama_custom_overrides.yaml` or disable it.

```
ollama:
 [...]
 gpu:
   enabled: false
   type: 'nvidia'
   number: 1
```

EXAMPLE 3: BASIC OVERRIDE FILE WITH GPU AND TWO MODELS PULLED AT STARTUP

```
global:
  imagePullSecrets:
  - application-collection
ingress:
  enabled: false
defaultModel: "gemma:2b"
runtimeClassName: nvidia
ollama:
  models:
    pull:
      - "gemma:2b"
      - "llama3.1"
    run:
      - "gemma:2b"
      - "llama3.1"
  gpu:
    enabled: true
    type: 'nvidia'
    number: 1
    nvidiaResource: "nvidia.com/gpu"
persistentVolume: ①
  enabled: true
  storageClass: local-path ②
```

① Without the `persistentVolume` option enabled, changes made to Ollama—such as downloading other LLM-- are lost when the container is restarted.

② Use `local-path` storage only for testing purposes. For production use, we recommend using a storage solution suitable for persistent storage, such as SUSE Storage.

EXAMPLE 4: BASIC OVERRIDE FILE WITH INGRESS AND NO GPU

```
ollama:
  models:
    pull:
      - llama2
    run:
      - llama2
  persistentVolume:
    enabled: true
    storageClass: local-path ①
ingress:
  enabled: true
  hosts:
  - host: <OLLAMA_API_URL>
    paths:
      - path: /
        pathType: Prefix
```

① Use `local-path` storage (requires installing the corresponding provisioner) only for testing purposes. For production use, we recommend using a storage solution suitable for persistent storage, such as SUSE Storage.

TABLE 2: OVERRIDE FILE OPTIONS FOR THE OLLAMA HELM CHART

Key	Type	Default	Description
affinity	object	{}	Affinity for pod assignment
autoscaling.enabled	bool	false	Enable autoscaling
autoscaling.maxReplicas	int	100	Number of maximum replicas
autoscaling.minReplicas	int	1	Number of minimum replicas
autoscaling.targetCPUUtilizationPercentage	int	80	CPU usage to target replica
extraArgs	list	[]	Additional arguments on the output Deployment definition.
extraEnv	list	[]	Additional environment variables on the output Deployment definition.
fullnameOverride	string	""	String to fully override template
global.imagePullSecrets	list	[]	Global override for container image registry pull secrets
global.imageRegistry	string	""	Global override for container image registry

Key	Type	Default	Description
hostIPC	bool	false	Use the host’s IPC namespace
hostNetwork	bool	false	Use the host’s network namespace
hostPID	bool	false	Use the host’s PID namespace.
image.pullPolicy	string	"IfNotPresent"	Image pull policy to use for the Ollama container
image.registry	string	"dp.apps.rancher.io"	Image registry to use for the Ollama container
image.repository	string	"containers/ollama"	Image repository to use for the Ollama container
image.tag	string	"0.3.6"	Image tag to use for the Ollama container
imagePullSecrets	list	[]	Docker registry secret names as an array
ingress.annotations	object	{}	Additional annotations for the Ingress resource
ingress.className	string	""	IngressClass that is used to implement the Ingress (Kubernetes 1.18+)

Key	Type	Default	Description
ingress.enabled	bool	false	Enable Ingress controller resource
ingress.hosts[0].host	string	"ollama.local"	
ingress.hosts[0].paths[0].path	string	"/"	
ingress.hosts[0].paths[0].pathType	string	"Prefix"	
ingress.tls	list	[]	The TLS configuration for host names to be covered with this Ingress record
initContainers	list	[]	Init containers to add to the pod
knative.containerConcurrency	int	0	Knative service container concurrency
knative.enabled	bool	false	Enable Knative integration
knative.idleTimeoutSeconds	int	300	Knative service idle timeout seconds
knative.responseStartTimeoutSeconds	int	300	Knative service response start timeout seconds
knative.timeoutSeconds	int	300	Knative service timeout seconds
livenessProbe.enabled	bool	true	Enable livenessProbe

Key	Type	Default	Description
livenessProbe.failureThreshold	int	6	Failure threshold for livenessProbe
livenessProbe.initialDelaySeconds	int	60	Initial delay seconds for livenessProbe
livenessProbe.path	string	"/"	Request path for livenessProbe
livenessProbe.periodSeconds	int	10	Period seconds for livenessProbe
livenessProbe.successThreshold	int	1	Success threshold for livenessProbe
livenessProbe.timeoutSeconds	int	5	Timeout seconds for livenessProbe
nameOverride	string	""	String to partially override template (maintains the release name)
nodeSelector	object	{}	Node labels for pod assignment
ollama.gpu.enabled	bool	false	Enable GPU integration
ollama.gpu.number	int	1	Specify the number of GPUs
ollama.gpu.nvidiaResource	string	"nvidia.com/gpu"	Only for NVIDIA cards; change to `nvidia.com/mig-1g.10gb` to use MIG slice

Key	Type	Default	Description
ollama.gpu.type	string	"nvidia"	GPU type: 'nvidia' or 'amd.' If 'ollama.g-pu.enabled' is en-abled, the default value is 'nvidia.' If set to 'amd,' this adds the 'rocm' suffix to the image tag if 'im-age.tag' is not over-ride. This is because AMD and CPU/CUDA are different images.
ollama.insecure	bool	false	Add insecure flag for pulling at container startup
ollama.models	list	[]	List of models to pull at container startup. The more you add, the longer the con-tainer takes to start if models are not present models: - lla-ma2 - mistral
ollama.mountPath	string	""	Override ollama-data volume mount path, default: "/root/.olla-ma"
persistentVolume.ac-cessModes	list	["ReadWriteOnce"]	Ollama server da-ta Persistent Vol-ume access modes. Must match those of

Key	Type	Default	Description
			existing PV or dynamic provisioner, see https://kubernetes.io/docs/concepts/storage/persistent-volumes/ ↗.
persistentVolume.annotations	object	{}	Ollama server data Persistent Volume annotations
persistentVolume.enabled	bool	false	Enable persistence using PVC
persistentVolume.existingClaim	string	""	If you want to bring your own PVC for persisting Ollama state, pass the name of the created + ready PVC here. If set, this Chart does not create the default PVC. Requires `server.persistentVolume.enabled: true`
persistentVolume.size	string	"30Gi"	Ollama server data Persistent Volume size
persistentVolume.storageClass	string	""	If persistentVolume.storageClass is present, and is set to either a dash ('-') or empty string (''), dynamic provision-

Key	Type	Default	Description
			ing is disabled. Otherwise, the storageClassName for persistent volume claim is set to the given value specified by persistentVolume.storageClass. If persistentVolume.storageClass is absent, the default storage class is used for dynamic provisioning whenever possible. See https://kubernetes.io/docs/concepts/storage/storage-classes/ ↗ for more details.
persistentVolume.subPath	string	""	Subdirectory of Ollama server data Persistent Volume to mount. Useful if the volume's root directory is not empty.
persistentVolume.volumeMode	string	""	Ollama server data Persistent Volume Binding Mode. If empty (the default) or set to null, no volumeBindingMode

Key	Type	Default	Description
			specification is set, choosing the default mode.
persistentVolume.volumeName	string	""	Ollama server Persistent Volume name. It can be used to force-attach the created PVC to a specific PV.
podAnnotations	object	{}	Map of annotations to add to the pods
podLabels	object	{}	Map of labels to add to the pods
podSecurityContext	object	{}	Pod Security Context
readinessProbe.enabled	bool	true	Enable readinessProbe
readinessProbe.failureThreshold	int	6	Failure threshold for readinessProbe
readinessProbe.initialDelaySeconds	int	30	Initial delay seconds for readinessProbe
readinessProbe.path	string	"/"	Request path for readinessProbe
readinessProbe.periodSeconds	int	5	Period seconds for readinessProbe
readinessProbe.successThreshold	int	1	Success threshold for readinessProbe
readinessProbe.timeoutSeconds	int	3	Timeout seconds for readinessProbe

Key	Type	Default	Description
replicaCount	int	1	Number of replicas
resources.limits	object	{}	Pod limit
resources.requests	object	{}	Pod requests
runtimeClassName	string	""	Specify runtime class
securityContext	object	{}	Container Security Context
service.annotations	object	{}	Annotations to add to the service
service.nodePort	int	31434	Service node port when service type is 'NodePort'
service.port	int	11434	Service port
service.type	string	"ClusterIP"	Service type
serviceAccount.annotations	object	{}	Annotations to add to the service account
serviceAccount.automount	bool	true	Whether to automatically mount a ServiceAccount’s API credentials
serviceAccount.create	bool	true	Whether a service account should be created
serviceAccount.name	string	""	The name of the service account to use. If not set and 'create'

Key	Type	Default	Description
			is 'true', a name is generated using the full name template.
tolerations	list	[]	Tolerations for pod assignment
topologySpreadConstraints	object	{}	Topology Spread Constraints for pod assignment
updateStrategy	object	{"type":""}	How to replace existing pods.
updateStrategy.type	string	""	Can be 'Recreate' or 'RollingUpdate'; default is 'RollingUpdate'
volumeMounts	list	[]	Additional volumeMounts on the output Deployment definition
volumes	list	[]	Additional volumes on the output Deployment definition

4.6 Installing Open WebUI

Open WebUI is a user-friendly web interface for interacting with Large Language Models (LLMs). It supports various LLM runners, including Ollama and vLLM.

4.6.1 Details about the Open WebUI application

Before deploying Open WebUI, it is important to know more about the supported configurations and documentation. The following command provides the corresponding details:

```
helm show values oci://dp.apps.rancher.io/charts/open-webui
```

Alternatively, you can also refer to the Open WebUI Helm chart page on the SUSE Application Collection site at https://apps.rancher.io/applications/open-webui ↗. It contains available versions and the link to pull the Open WebUI container image.

4.6.2 Open WebUI installation procedure

Tip

Before the installation, you need to get user access to the SUSE Application Collection, create a Kubernetes namespace, and log in to the Helm registry as described in *Section 4.1, "Installation procedure"*.

REQUIREMENTS

- An installed cert-manager. If cert-manager is not installed from previous Open WebUI releases, install it by following the steps in *Section 4.2, "Installing cert-manager"*.

1. Create the `owui_custom_overrides.yaml` file to override the values of the parent Helm chart. The file contains URLs for Milvus and Ollama, and specifies whether a stand-alone Ollama deployment is used or whether Ollama is installed as part of the Open WebUI installation. Find more details in *Section 4.6.5, "Examples of Open WebUI Helm chart override files"*. For a list of all installation options with examples, refer to *Section 4.6.6, "Values for the Open WebUI Helm chart"*.

2. Install the Open WebUI Helm chart using the `owui_custom_overrides.yaml` override file.

   ```
   > helm upgrade --install \
     open-webui charts/open-webui-<X.Y.Z>.tgz \
     -n <SUSE_AI_NAMESPACE> \
     --version <X.Y.Z> -f <owui_custom_overrides.yaml>
   ```

4.6.3 Upgrading Open WebUI

To upgrade Open WebUI to a specific new version, run the following command:

```
> helm upgrade --install open-webui \
  oci://dp.apps.rancher.io/charts/open-webui \
  -n <SUSE_AI_NAMESPACE> \
  --version <VERSION_NUMBER> \
  -f <owui_custom_overrides.yaml>
```

To upgrade Open WebUI to the latest version, run the following command:

```
> helm upgrade --install open-webui \
  oci://dp.apps.rancher.io/charts/open-webui \
  -n <SUSE_AI_NAMESPACE> \
  -f <owui_custom_overrides.yaml>
```

4.6.4 Uninstalling Open WebUI

To uninstall Open WebUI, run the following command:

```
> helm uninstall open-webui -n <SUSE_AI_NAMESPACE>
```

4.6.5 Examples of Open WebUI Helm chart override files

To override the default values during the Helm chart installation or update, you can create an override YAML file with custom values. Then, apply these values by specifying the path to the override file with the `-f` option of the `helm` command. Remember to replace <SUSE_AI_NAMESPACE> with your Kubernetes namespace.

EXAMPLE 5: OPEN WEBUI OVERRIDE FILE WITH OLLAMA INCLUDED

The following override file installs Ollama during the Open WebUI installation.

```
global:
  imagePullSecrets:
  - application-collection
  ifdef::deployment_airgap[]
  imageRegistry: <LOCAL_DOCKER_REGISTRY_URL>:5043
  endif::[]
ollamaUrls:
- http://open-webui-ollama.<SUSE_AI_NAMESPACE>.svc.cluster.local:11434
persistence:
  enabled: true
  storageClass: local-path (1)
```

```
ollama:
  enabled: true
  ingress:
    enabled: false
  defaultModel: "gemma:2b"
  ollama:
    models: 2
      - "gemma:2b"
      - "llama3.1"
    gpu: 3
      enabled: true
      type: 'nvidia'
      number: 1
    persistentVolume: 4
      enabled: true
      storageClass: local-path
pipelines:
  enabled: true
  persistence:
    storageClass: local-path
  extraEnvVars: 5
    - name: PIPELINES_URLS 6
      value: "https://raw.githubusercontent.com/SUSE/suse-ai-observability-
extension/refs/heads/main/integrations/oi-filter/suse_ai_filter.py"
    - name: OTEL_SERVICE_NAME 7
      value: "Open WebUI"
    - name: OTEL_EXPORTER_HTTP_OTLP_ENDPONT 8
      value: "http://opentelemetry-collector.suse-
observability.svc.cluster.local:4318"
    - name: PRICING_JSON 9
      value: "https://raw.githubusercontent.com/SUSE/suse-ai-observability-
extension/refs/heads/main/integrations/oi-filter/pricing.json"
ingress:
  enabled: true
  class: ""
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    nginx.ingress.kubernetes.io/proxy-body-size: "1024m"
  host: suse-ollama-webui 10
  tls: true
extraEnvVars:
- name: DEFAULT_MODELS 11
  value: "gemma:2b"
- name: DEFAULT_USER_ROLE
  value: "user"
- name: WEBUI_NAME
  value: "SUSE AI"
```

```
- name: GLOBAL_LOG_LEVEL
  value: INFO
- name: RAG_EMBEDDING_MODEL
  value: "sentence-transformers/all-MiniLM-L6-v2"
- name: VECTOR_DB
  value: "milvus"
- name: MILVUS_URI
  value: http://milvus.<SUSE_AI_NAMESPACE>.svc.cluster.local:19530
- name: INSTALL_NLTK_DATASETS 12
  value: "true"
- name: OMP_NUM_THREADS
  value: "1"
- name: OPENAI_API_KEY 13
  value: "0p3n-w3bu!"
```

1 Use `local-path` storage only for testing purposes. For production use, we recommend using a storage solution more suitable for persistent storage. To use SUSE Storage, specify `longhorn`.

2 Specifies that two large language models (LLM) will be loaded in Ollama when the container starts.

3 Enables GPU support for Ollama. The `type` must be `nvidia` because NVIDIA GPUs are the only supported devices. `number` must be between 1 and the number of NVIDIA GPUs present on the system.

4 Without the `persistentVolume` option enabled, changes made to Ollama—such as downloading other LLM-- are lost when the container is restarted.

5 The environment variables that you are making available for the pipeline's runtime container.

6 A list of pipeline URLs to be downloaded and installed by default. Individual URLs are separated by a semicolon `;`.

7 The service name that appears in traces and topological representations in SUSE Observability.

8 The endpoint for the OpenTelemetry collector. Make sure to use the HTTP port of your collector.

9 A file for the model multipliers in cost estimation. You can customize it to match your actual infrastructure experimentally.

10 Specifies the default LLM for Ollama.

11 Specifies the host name for the Open WebUI Web UI.

(12) Installs the *natural language toolkit* (NLTK) datasets for Ollama. Refer to https://www.nltk.org/index.html for licensing information.

(13) API key value for communication between Open WebUI and Open WebUI Pipelines. The default value is '0p3n-w3bu!'.

EXAMPLE 6: OPEN WEBUI OVERRIDE FILE WITH OLLAMA INSTALLED SEPARATELY

The following override file installs Ollama separately from the Open WebUI installation.

```
global:
  imagePullSecrets:
  - application-collection
  ifdef::deployment_airgap[]
  imageRegistry: <LOCAL_DOCKER_REGISTRY_URL>:5043
  endif::[]
ollamaUrls:
- http://ollama.<SUSE_AI_NAMESPACE>.svc.cluster.local:11434
persistence:
  enabled: true
  storageClass: local-path (1)
ollama:
  enabled: false
pipelines:
  enabled: False
  persistence:
    storageClass: local-path (2)
ingress:
  enabled: true
  class: ""
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
  host: suse-ollama-webui
  tls: true
extraEnvVars:
- name: DEFAULT_MODELS (3)
  value: "gemma:2b"
- name: DEFAULT_USER_ROLE
  value: "user"
- name: WEBUI_NAME
  value: "SUSE AI"
- name: GLOBAL_LOG_LEVEL
  value: INFO
- name: RAG_EMBEDDING_MODEL
  value: "sentence-transformers/all-MiniLM-L6-v2"
- name: VECTOR_DB
  value: "milvus"
- name: MILVUS_URI
```

```
  value: http://milvus.<SUSE_AI_NAMESPACE>.svc.cluster.local:19530
- name: ENABLE_OTEL ❹
  value: "true"
- name: OTEL_EXPORTER_OTLP_ENDPOINT ❺
  value: http://opentelemetry-collector.observability.svc.cluster.local:4317 ❻
- name: OMP_NUM_THREADS
  value: "1"
```

❶ Use `local-path` storage only for testing purposes. For production use, we recommend using a storage solution suitable for persistent storage, such as SUSE Storage.

❷ Use `local-path` storage only for testing purposes. For production use, we recommend using a storage solution suitable for persistent storage, such as SUSE Storage.

❸ Specifies the default LLM for Ollama.

❹ These values are optional, required only to receive telemetry data from Open WebUI.

❺ These values are optional, required only to receive telemetry data from Open WebUI.

❻ The URL of the OpenTelemetry Collector installed by the user.

EXAMPLE 7: OPEN WEBUI OVERRIDE FILE WITH PIPELINES ENABLED

The following override file installs Ollama separately and enables Open WebUI pipelines. This simple filter adds a limit to the number of question and answer turns during the LLM chat.

Tip

Pipelines normally require additional configuration provided either via environment variables or specified in the Open WebUI Web UI.

```
global:
  imagePullSecrets:
  - application-collection
  ifdef::deployment_airgap[]
  imageRegistry: <LOCAL_DOCKER_REGISTRY_URL>:5043
  endif::[]
ollamaUrls:
- http://ollama.<SUSE_AI_NAMESPACE>.svc.cluster.local:11434
persistence:
  enabled: true
  storageClass: local-path
ollama:
  enabled: false
```

```
pipelines:
  enabled: true
  persistence:
    storageClass: local-path
  extraEnvVars:
  - name: PIPELINES_URLS ❶
    value: "https://raw.githubusercontent.com/SUSE/suse-ai-observability-extension/
refs/heads/main/integrations/oi-filter/conversation_turn_limit_filter.py"
ingress:
  enabled: true
  class: ""
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
  host: suse-ollama-webui
  tls: true
[...]
```

❶ A list of pipeline URLs to be downloaded and installed by default. Individual URLs are separated by a semicolon `;`.

EXAMPLE 8: OPEN WEBUI OVERRIDE FILE WITH A CONNECTION TO VLLM

The following example shows how to extend the `extraEnvVars` section of the Open WebUI override file to connect to vLLM. Replace `SUSE_AI_NAMESPACE` with your Kubernetes namespace.

Tip

Find more details about installing vLLM in *Section 4.7, “Installing vLLM”*.

```
extraEnvVars:
[...]
- name: OPENAI_API_BASE_URL
  value: "http://vllm-router-service.<SUSE_AI_NAMESPACE>.svc.cluster.local:80/v1"
- name: OPENAI_API_KEY
  value: "dummy" ❶
```

❶ Open WebUI will require you to provide the OpenAI API key.

If the Open WebUI installation has pipelines enabled besides the vLLM deployment, you can extend the `extraEnvVars` section as follows.

```
extraEnvVars:
[...]
- name: OPENAI_API_BASE_URLS
```

```
  value: "http://open-webui-
pipelines.<SUSE_AI_NAMESPACE>.svc.cluster.local:9099;http://vllm-router-
service.<SUSE_AI_NAMESPACE>.svc.cluster.local:80/v1"
- name: OPENAI_API_KEYS
  value: "0p3n-w3bu!;dummy"
```

EXAMPLE 9: STAND-ALONE DEPLOYMENT OF OPEN-WEBUI-PIPELINES

You can install the `open-webui-pipelines` service as a stand-alone deployment, independent of the Open WebUI chart. To install open-webui-pipelines as a stand-alone component, use the following command:

```
> helm upgrade --install open-webui-pipelines \
  oci://dp.apps.rancher.io/charts/open-webui-pipelines \
-n <SUSE_AI_NAMESPACE> \
-f open-webui-pipelines-values.yaml
```

Following is an example of the `open-webui-pipelines-values.yaml` override file.

```
runtimeClassName: nvidia
global:
  imagePullSecrets:
    - application-collection
image:
  registry: dp.apps.rancher.io
  repository: containers/open-webui-pipelines
  tag: <IMAGE_TAG>
  pullPolicy: IfNotPresent
persistence:
  enabled: true
  storageClass: local-path
  size: 10Gi
```

4.6.6 Values for the Open WebUI Helm chart

To override the default values during the Helm chart installation or update, you can create an override YAML file with custom values. Then, apply these values by specifying the path to the override file with the `-f` option of the `helm` command. Remember to replace <SUSE_AI_NAMESPACE> with your Kubernetes namespace.

TABLE 3: AVAILABLE OPTIONS FOR THE OPEN WEBUI HELM CHART

Key	**Type**	**Default**	**Description**
affinity	object	{}	Affinity for pod assignment

Key	Type	Default	Description
annotations	object	{}	
cert-manager.enabled	bool	true	
clusterDomain	string	"cluster.local"	Value of cluster domain
containerSecurityContext	object	{}	Configure container security context, see https://kubernetes.io/docs/tasks/configure-pod-container/security-context/#set-the-security-context-for-a-containe.
extraEnvVars	list	[{"name":"OPENAI_API_KEY", "value":"0p3n-w3bu!"}]	Environment variables added to the Open WebUI deployment. Most up-to-date environment variables can be found in https://docs.openwebui.com/getting-started/env-configuration/.
extraEnvVars[0]	object	{"name":"OPENAI_API_KEY","value":"0p3n-w3bu!"}	Default API key value for Pipelines. It should be updated in a production deployment and changed to the required API key if not using Pipelines.

Key	Type	Default	Description
global.imagePullSe-crets	list	[]	Global override for container image reg-istry pull secrets
global.imageRegistry	string	""	Global override for container image reg-istry
global.tls.additional-TrustedCAs	bool	false	
global.tls.issuerName	string	"suse-private-ai"	
global.tls.letsEncryp-t.email	string	"none@example.com (mailto:none@exam-ple.com) ↗"	
global.tls.letsEncryp-t.environment	string	"staging"	
global.tls.letsEncryp-t.ingress.class	string	""	
global.tls.source	string	"suse-private-ai"	The source of Open WebUI TLS keys, see *Section 4.6.6.1, "TLS sources"*.
image.pullPolicy	string	"IfNotPresent"	Image pull policy to use for the Open We-bUI container
image.registry	string	"dp.apps.rancher.io"	Image registry to use for the Open WebUI container

Key	Type	Default	Description
image.repository	string	"containers/open-webui"	Image repository to use for the Open WebUI container
image.tag	string	"0.3.32"	Image tag to use for the Open WebUI container
imagePullSecrets	list	[]	Configure imagePullSecrets to use private registry, see https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/ .
ingress.annotations	object	{"nginx.ingress.kubernetes.io/ssl-redirect":"true"}	Use appropriate annotations for your Ingress controller, such as `nginx.ingress.kubernetes.io/rewrite-target: /` for NGINX.
ingress.class	string	""	
ingress.enabled	bool	true	
ingress.existingSecret	string	""	
ingress.host	string	""	
ingress.tls	bool	true	

Key	Type	Default	Description
nameOverride	string	""	
nodeSelector	object	{}	Node labels for pod assignment
ollama.enabled	bool	true	Automatically install Ollama Helm chart from https://otwld.github.io/ollama-helm/ ↗. Configure the following Helm values (https://github.com/otwld/ollama-helm/#helm-values) ↗.
ollama.fullnameOverride	string	"open-webui-ollama"	If enabling embedded Ollama, update fullnameOverride to your desired Ollama name value, or else it will use the default ollama.name value from the Ollama chart.
ollamaUrls	list	[]	A list of Ollama API endpoints. These can be added instead of automatically installing the Ollama Helm chart, or in addition to it.

Key	Type	Default	Description
openaiBaseApiUrl	string	""	OpenAI base API URL to use. Defaults to the Pipelines service endpoint when Pipelines are enabled, or to `https://api.openai.com/v1` if Pipelines are not enabled and this value is blank.
persistence.accessModes	list	["ReadWriteOnce"]	If using multiple replicas, you must update accessModes to ReadWriteMany.
persistence.annotations	object	{}	
persistence.enabled	bool	true	
persistence.existingClaim	string	""	Use existingClaim to reuse an existing Open WebUI PVC instead of creating a new one.
persistence.selector	object	{}	
persistence.size	string	"2Gi"	
persistence.storageClass	string	""	

Key	Type	Default	Description
pipelines.enabled	bool	false	Automatically install Pipelines chart to extend Open WebUI functionality using Pipelines, see https://github.com/open-webui/pipelines/ ↗.
pipelines.extraEnvVars	list	[]	This section can be used to pass the required environment variables to your pipelines (such as the Langfuse host name).
podAnnotations	object	{}	
podSecurityContext	object	{}	Configure pod security context, see https://kubernetes.io/docs/tasks/configure-pod-container/security-context/#set-the-security-context-for-a-containe ↗.
replicaCount	int	1	
resources	object	{}	
service	object	{"annotations":{},"containerPort":8080, "labels":{},"loadBal-	Service values to expose Open WebUI pods to cluster

Key	Type	Default	Description
		ancerClass":"", "nodePort":"","port":80,"type":"ClusterIP"}	
tolerations	list	[]	Tolerations for pod assignment
topologySpreadConstraints	list	[]	Topology Spread Constraints for pod assignment

4.6.6.1 TLS sources

There are three recommended options where Open WebUI can obtain TLS certificates for secure communication.

Self-Signed TLS certificate

This is the default method. You need to install `cert-manager` on the cluster to issue and maintain the certificates. This method generates a CA and signs the Open WebUI certificate using the CA. `cert-manager` then manages the signed certificate. For this method, use the following Helm chart option:

```
global.tls.source=suse-private-ai
```

Let’s Encrypt

This method also uses `cert-manager`, but it is combined with a special issuer for Let’s Encrypt that performs all actions—including request and validation—to get the Let’s Encrypt certificate issued. This configuration uses HTTP validation (HTTP-01) and therefore the load balancer must have a public DNS record and be accessible from the Internet. For this method, use the following Helm chart option:

```
global.tls.source=letsEncrypt
```

Provide your own certificate

This method allows you to bring your own signed certificate to secure the HTTPS traffic. In this case, you must upload this certificate and associated key as PEM-encoded files named `tls.crt` and `tls.key`. For this method, use the following Helm chart option:

```
global.tls.source=secret
```

4.7 Installing vLLM

vLLM is an open-source high-performance inference and serving engine for large language models (LLMs). It is designed to maximize throughput and reduce latency by using an efficient memory management system that handles dynamic batching and streaming outputs. In short, vLLM makes running LLMs cheaper and faster in production.

Deploying vLLM on Kubernetes is a scalable and efficient way to serve machine learning models. This guide walks you through deploying vLLM using its Helm chart, which is part of AI Library. The Helm chart deploys the full vLLM production stack and enables you to run optimized LLM inference workloads on NVIDIA GPU in your Kubernetes cluster. It consists of the following components:

- **Serving Engine** runs the model inference.
- **Router** handles OpenAI-compatible API requests.
- **LMCache** *(Optional)* improves caching efficiency.
- **CacheServer** *(Optional)* is a distributed KV cache back-end.

4.7.1 Details about the vLLM application

Before deploying vLLM, it is important to know more about the supported configurations and documentation. The following command provides the corresponding details:

```
helm show values oci://dp.apps.rancher.io/charts/vllm
```

Alternatively, you can also refer to the vLLM Helm chart page on the SUSE Application Collection site at https://apps.rancher.io/applications/vllm ↗. It contains vLLM dependencies, available versions and the link to pull the vLLM container image.

4.7.2 vLLM installation procedure

Tip

Before the installation, you need to get user access to the SUSE Application Collection, create a Kubernetes namespace, and log in to the Helm registry as described in *Section 4.1, "Installation procedure"*.

Warning: NVIDIA GPUs required

NVIDIA GPUs must be available in your Kubernetes cluster to successfully deploy and run vLLM.

Important: Limitation

The current release of SUSE AI vLLM does not support Ray and LoraController.

1. Create a `vllm_custom_overrides.yaml` file to override the default values of the Helm chart. Find examples of override files in *Section 4.7.6, "Examples of vLLM Helm chart override files"*.
2. After saving the override file as `vllm_custom_overrides.yaml`, apply its configuration with the following command.

```
> helm upgrade --install \
  vllm oci://dp.apps.rancher.io/charts/vllm \
  -n <SUSE_AI_NAMESPACE> \
  -f <vllm_custom_overrides.yaml>
```

4.7.3 Integrating vLLM with Open WebUI

You can integrate vLLM in Open WebUI either using the Open WebUI Web user interface, or updating Open WebUI override file during Open WebUI deployment (see *Example 8, "Open WebUI override file with a connection to vLLM"*).

Integrating vLLM with Open WebUI via the Web user interface.

REQUIREMENTS

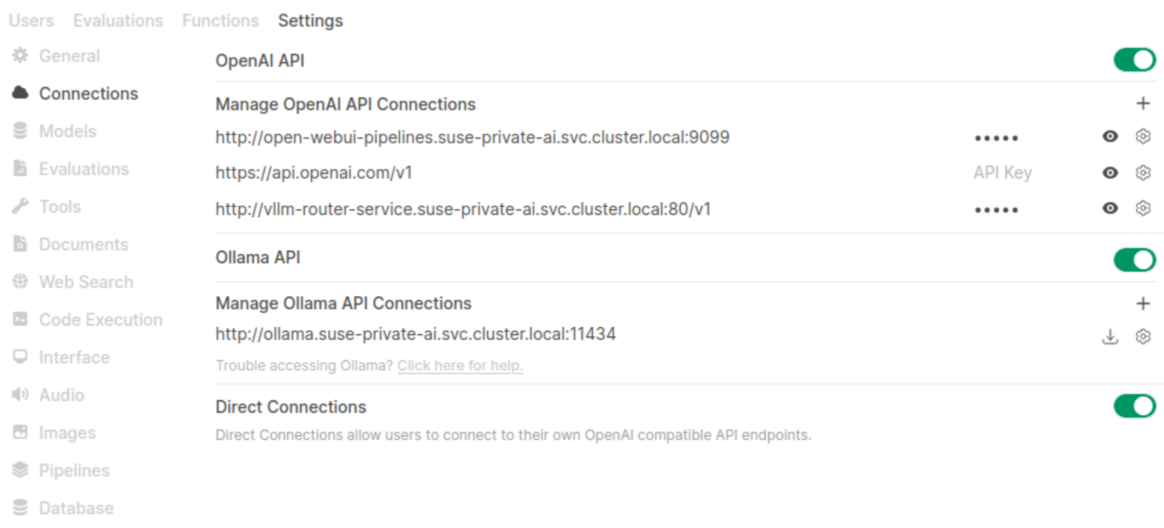
-

You must have Open WebUI administrator privileges to access configuration screens or settings mentioned in this section.

1. In the bottom left of the Open WebUI window, click your avatar icon to open the user menu and select *Admin Panel*.
2. Click the *Settings* tab and select *Connections* from the left menu.
3. In the *Manage OpenAI API Connections* section, add a new connection URL to the vLLM router service, for example:

   ```
   http://vllm-router-service.<SUSE_AI_NAMESPACE>.svc.cluster.local:80/v1
   ```

   Confirm with *Save*.



OpenAI API

Manage OpenAI API Connections

http://open-webui-pipelines.suse-private-ai.svc.cluster.local:9099 •••••

https://api.openai.com/v1 API Key

http://vllm-router-service.suse-private-ai.svc.cluster.local:80/v1 •••••

Ollama API

Manage Ollama API Connections

http://ollama.suse-private-ai.svc.cluster.local:11434

Trouble accessing Ollama? Click here for help.

Direct Connections

Direct Connections allow users to connect to their own OpenAI compatible API endpoints.

FIGURE 34: ADDING A VLLM CONNECTION TO OPEN WEBUI

4.7.4 Upgrading vLLM

The vLLM chart receives application updates and updates of the Helm chart templates. New versions may include changes that require manual steps. These steps are listed in the corresponding `README` file. All vLLM dependencies are updated automatically during a vLLM upgrade.

To upgrade vLLM, identify the new version number and run the following command below:

```
> helm upgrade --install \
  vllm oci://dp.apps.rancher.io/charts/vllm \
  -n <SUSE_AI_NAMESPACE> \
  --version <VERSION_NUMBER> \
```

```
  -f <vllm_custom_overrides.yaml>
```

Tip

If you omit the `--version` option, vLLM gets upgraded to the latest available version.

Note: Rolling update

The `helm upgrade` command performs a rolling update on Deployments or StatefulSets with the following conditions:

- The *old* pod stays running until the *new* pod passes readiness checks.
- If the cluster is already at GPU capacity, the new pod cannot start because there is no GPU left to schedule it. This requires patching the deployment using the *Recreate* update strategy. The following commands identify the vLLM deployment name and patch its deployment.

  ```
  > kubectl get deployments -n <SUSE_AI_NAMESPACE>
  > kubectl patch deployment <VLLM_DEPLOYMENT_NAME> \
    -n <SUSE_AI_NAMESPACE> \
    -p '{"spec": {"strategy": {"type": "Recreate", "rollingUpdate": null}}}'
  ```

4.7.5 Uninstalling vLLM

To uninstall vLLM, run the following command:

```
> helm uninstall vllm -n <SUSE_AI_NAMESPACE>
```

4.7.6 Examples of vLLM Helm chart override files

To override the default values during the Helm chart installation or update, you can create an override YAML file with custom values. Then, apply these values by specifying the path to the override file with the `-f` option of the `helm` command. Remember to replace <SUSE_AI_NAMES-PACE> with your Kubernetes namespace.

EXAMPLE 10: MINIMAL CONFIGURATION

The following override file installs vLLM using a model that is publicly available.

```
global:
  imagePullSecrets:
  - application-collection
servingEngineSpec:
  modelSpec:
  - name: "phi3-mini-4k"
    registry: "dp.apps.rancher.io"
    repository: "containers/vllm-openai"
    tag: "0.9.1"
    imagePullPolicy: "IfNotPresent"
    modelURL: "microsoft/Phi-3-mini-4k-instruct"
    replicaCount: 1
    requestCPU: 6
    requestMemory: "16Gi"
    requestGPU: 1
```

VALIDATING THE INSTALLATION

1. Pulling the images can take a long time. You can monitor the status of the vLLM installation by running the following command:

```
> kubectl get pods -n <SUSE_AI_NAMESPACE>

NAME                                            READY   STATUS     RESTARTS
 AGE
[...]
vllm-deployment-router-7588bf995c-5jbkf         1/1     Running    0
 8m9s
vllm-phi3-mini-4k-deployment-vllm-79d6fdc-tx7  1/1     Running    0
 8m9s
```

   Pods for the vLLM deployment should transition to the states `Ready` and `Running`.

VALIDATING THE STACK

1. Expose the `vllm-router-service` port to the host machine:

```
> kubectl port-forward svc/vllm-router-service \
  -n <SUSE_AI_NAMESPACE> 30080:80
```

2. Query the OpenAI-compatible API to list the available models:

```
> curl -o- http://localhost:30080/v1/models
```

3. Send a query to the OpenAI `/completion` endpoint to generate a completion for a prompt:

```
> curl -X POST http://localhost:30080/v1/completions \
  -H "Content-Type: application/json" \
  -d '{
    "model": "microsoft/Phi-3-mini-4k-instruct",
    "prompt": "Once upon a time,",
    "max_tokens": 10
  }'
```

```
# example output of generated completions
{
    "id": "cmpl-3dd11a3624654629a3828c37bac3edd2",
    "object": "text_completion",
    "created": 1757530703,
    "model": "microsoft/Phi-3-mini-4k-instruct",
    "choices": [
        {
            "index": 0,
            "text": " in a bustling city full of concrete and",
            "logprobs": null,
            "finish_reason": "length",
            "stop_reason": null,
            "prompt_logprobs": null
        }
    ],
    "usage": {
        "prompt_tokens": 5,
        "total_tokens": 15,
        "completion_tokens": 10,
        "prompt_tokens_details": null
    },
    "kv_transfer_params": null
}
```

EXAMPLE 11: BASIC CONFIGURATION

The following vLLM override file includes basic configuration options.

PREREQUISITES

- Access to a Hugging Face token (`HF_TOKEN`).

- The model `meta-llama/Llama-3.1-8B-Instruct` from this example is a gated model that requires you to accept the agreement to access it. For more information, see https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct.
- The `runtimeClassName` specified here is `nvidia`.
- Update the `storageClass:` entry for each `modelSpec`.

```
# vllm_custom_overrides.yaml
global:
  imagePullSecrets:
  - application-collection
servingEngineSpec:
  runtimeClassName: "nvidia"
  modelSpec:
  - name: "llama3" ①
    registry: "dp.apps.rancher.io" ②
    repository: "containers/vllm-openai" ③
    tag: "0.9.1" ④
    imagePullPolicy: "IfNotPresent"
    modelURL: "meta-llama/Llama-3.1-8B-Instruct" ⑤
    replicaCount: 1 ⑥
    requestCPU: 10 ⑦
    requestMemory: "16Gi" ⑧
    requestGPU: 1 ⑨
    storageClass: <STORAGE_CLASS>
    pvcStorage: "50Gi" ⑩
    pvcAccessMode:
      - ReadWriteOnce

    vllmConfig:
      enableChunkedPrefill: false ⑪
      enablePrefixCaching: false ⑫
      maxModelLen: 4096 ⑬
      dtype: "bfloat16" ⑭
      extraArgs: ["--disable-log-requests", "--gpu-memory-utilization", "0.8"] ⑮

    hf_token: <HF_TOKEN> ⑯
```

① The unique identifier for your model deployment.

② The Docker image registry containing the model's serving engine image.

③ The Docker image repository containing the model's serving engine image.

④ The version of the model image to use.

⑤ The URL pointing to the model on Hugging Face or another hosting service.

6 The number of replicas for the deployment, which allows scaling for load.

7 The amount of CPU resources requested per replica.

8 Memory allocation for the deployment. Sufficient memory is required to load the model.

9 The number of GPUs to allocate for the deployment.

10 The Persistent Volume Claim (PVC) size for model storage.

11 Optimizes performance by prefetching model chunks.

12 Enables caching of prompt prefixes to speed up inference for repeated prompts.

13 The maximum sequence length the model can handle.

14 The data type for model weights, such as `bfloat16` for mixed-precision inference and faster performance on modern GPUs.

15 Additional command-line arguments for vLLM, such as disabling request logging or setting GPU memory utilization.

16 Your Hugging Face token for accessing gated models. Replace `HF_TOKEN` with your actual token.

EXAMPLE 12: **LOADING PREFETCHED MODELS FROM PERSISTENT STORAGE**

Prefetching models to a Persistent Volume Claim (PVC) prevents repeated downloads from Hugging Face during pod startup. The process involves creating a PVC and a job to fetch the model. This PVC is mounted at `/models`, where the prefetch job stores the model weights. Subsequently, the vLLM `modelURL` is set to this path, which ensures that the model is loaded locally instead of being downloaded when the pod starts.

1. Define a PVC for model weights using the following YAML specification.

```
# pvc-models.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: models-pvc
  namespace: <SUSE_AI_NAMESPACE>
spec:
  accessModes: ["ReadWriteOnce"]
  resources:
    requests:
      storage: 50Gi # Adjust size based on your model
  storageClassName: <STORAGE_CLASS>
```

Save it as `pvc-models.yaml` and apply with `kubectl apply -f pvc-models.yaml`.

2. Create a secret resource for the Hugging Face token.

```
> kubectl create secret -n <SUSE_AI_NAMESPACE> \
  generic huggingface-credentials \
  --from-literal=HUGGING_FACE_HUB_TOKEN=<HF_TOKEN>
```

3. Create a YAML specification for prefetching the model and save it as `job-prefetch-llama3.1-8b.yaml`.

```
# job-prefetch-llama3.1-8b.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: prefetch-llama3.1-8b
  namespace: <SUSE_AI_NAMESPACE>
spec:
  template:
    spec:
      restartPolicy: OnFailure
      containers:
      - name: hf-download
        image: python:3.10-slim
        env:
        - name: HF_TOKEN
          valueFrom: { secretKeyRef: { name: huggingface-credentials, key:
 <HUGGING_FACE_HUB_TOKEN> } }
        - name: HF_HUB_ENABLE_HF_TRANSFER
          value: "1"
        - name: HF_HUB_DOWNLOAD_TIMEOUT
          value: "60"
        command: ["bash","-lc"]
        args:
        - |
          set -e
          echo "Logging in..."
          echo "Installing Hugging Face CLI..."
          pip install "huggingface_hub[cli]"
          pip install "hf_transfer"
          hf auth login --token "${HF_TOKEN}"
          echo "Downloading Llama 3.1 8B Instruct to /models/llama-3.1-8b-
it ..."
          hf download meta-llama/Llama-3.1-8B-Instruct --local-dir /models/
llama-3.1-8b-it
        volumeMounts:
```

```
        - name: models
          mountPath: /models
      volumes:
      - name: models
        persistentVolumeClaim:
          claimName: models-pvc
```

Apply the specification with the following commands:

```
> kubectl apply -f job-prefetch-llama3.1-8b.yaml
> kubectl -n <SUSE_AI_NAMESPACE> \
  wait --for=condition=complete job/prefetch-llama3.1-8b
```

4. Update the custom vLLM override file with support for PVC.

```
# vllm_custom_overrides.yaml
global:
  imagePullSecrets:
  - application-collection
servingEngineSpec:
  runtimeClassName: "nvidia"
  modelSpec:
  - name: "llama3"
    registry: "dp.apps.rancher.io"
    repository: "containers/vllm-openai"
    tag: "0.9.1"
    imagePullPolicy: "IfNotPresent"
    modelURL: "/models/llama-3.1-8b-it"
    replicaCount: 1

    requestCPU: 10
    requestMemory: "16Gi"
    requestGPU: 1

    extraVolumes:
      - name: models-pvc
        persistentVolumeClaim:
          claimName: models-pvc  1

    extraVolumeMounts:
      - name: models-pvc
        mountPath: /models  2

    vllmConfig:
      maxModelLen: 4096

    hf_token: <HF_TOKEN>
```

1. Specify your PVC name.
2. The mount path must match the base directory of the `servingEngineSpec.modelSpec.modeURL` value specified above.

Save it as `vllm_custom_overrides.yaml` and apply with `kubectl apply -f vllm_custom_overrides.yaml`.

5. The following example lists mounted PVCs for a pod.

```
> kubectl exec -it vllm-llama3-deployment-vllm-858bd967bd-w26f7 \
  -n <SUSE_AI_NAMESPACE> -- ls -l /models
drwxr-xr-x 1 root root 608 Aug 22 16:29 llama-3.1-8b-it
```

EXAMPLE 13: CONFIGURATION WITH MULTIPLE MODELS

This example shows how to configure multiple models to run on different GPUs. Remember to update the entries `hf_token` and `storageClass`.

Note: Ray is not supported

Ray is currently not supported. Therefore, sharding a single large model across multiple GPUs is not supported.

```
# vllm_custom_overrides.yaml
global:
  imagePullSecrets:
  - application-collection
servingEngineSpec:
  modelSpec:
  - name: "llama3"
    registry: "dp.apps.rancher.io"
    repository: "containers/vllm-openai"
    tag: "0.9.1"
    imagePullPolicy: "IfNotPresent"
    modelURL: "meta-llama/Llama-3.1-8B-Instruct"
    replicaCount: 1
    requestCPU: 10
    requestMemory: "16Gi"
    requestGPU: 1
    pvcStorage: "50Gi"
    storageClass: <STORAGE_CLASS>
    vllmConfig:
      maxModelLen: 4096
    hf_token: <HF_TOKEN_FOR_LLAMA_31>
  - name: "mistral"
    registry: "dp.apps.rancher.io"
    repository: "containers/vllm-openai"
    tag: "0.9.1"
    imagePullPolicy: "IfNotPresent"
    modelURL: "mistralai/Mistral-7B-Instruct-v0.2"
    replicaCount: 1
    requestCPU: 10
    requestMemory: "16Gi"
    requestGPU: 1
    pvcStorage: "50Gi"
    storageClass: <STORAGE_CLASS>
    vllmConfig:
      maxModelLen: 4096
    hf_token: <HF_TOKEN_FOR_MISTRAL>
```

EXAMPLE 14: CPU OFFLOADING

This example demonstrates how to enable KV cache offloading to the CPU using LMCache in a vLLM deployment. You can enable LMCache and set the CPU offloading buffer size using the `lmcacheConfig` field. In the following example, the buffer is set to 20 GB, but you can adjust this value based on your workload. Remember to update the entries `hf_token` and `storageClass`.

Warning: Experimental Features

Setting `lmcacheConfig.enabled` to `true` implicitly enables the `LMCACHE_USE_EXPERIMENTAL` flag for LMCache. These experimental features are only supported on newer GPU generations. It is not recommended to enable them without a compelling reason.

```
# vllm_custom_overrides.yaml
global:
  imagePullSecrets:
  - application-collection
servingEngineSpec:
  runtimeClassName: "nvidia"
  modelSpec:
  - name: "mistral"
    registry: "dp.apps.rancher.io"
    repository: "containers/lmcache-vllm-openai"
    tag: "0.3.2"
    imagePullPolicy: "IfNotPresent"
    modelURL: "mistralai/Mistral-7B-Instruct-v0.2"
    replicaCount: 1
    requestCPU: 10
    requestMemory: "40Gi"
    requestGPU: 1
    pvcStorage: "50Gi"
    storageClass: <STORAGE_CLASS>
    pvcAccessMode:
      - ReadWriteOnce
    vllmConfig:
      maxModelLen: 32000

    lmcacheConfig:
      enabled: false
      cpuOffloadingBufferSize: "20"

    hf_token: <HF_TOKEN>
```

EXAMPLE 15: SHARED REMOTE KV CACHE STORAGE WITH LMCACHE

This example shows how to enable remote KV cache storage using LMCache in a vLLM deployment. The configuration defines a `cacheserverSpec` and uses two replicas. Remember to replace the placeholder values for `hf_token` and `storageClass` before applying the configuration.

Warning: Experimental features

Setting `lmcacheConfig.enabled` to `true` implicitly enables the `LMCACHE_USE_EXPERIMENTAL` flag for LMCache. These experimental features are only supported on newer GPU generations. It is not recommended to enable them without a compelling reason.

```
# vllm_custom_overrides.yaml
global:
  imagePullSecrets:
  - application-collection
servingEngineSpec:
  runtimeClassName: "nvidia"
  modelSpec:
  - name: "mistral"
    registry: "dp.apps.rancher.io"
    repository: "containers/lmcache-vllm-openai"
    tag: "0.3.2"
    imagePullPolicy: "IfNotPresent"
    modelURL: "mistralai/Mistral-7B-Instruct-v0.2"
    replicaCount: 2
    requestCPU: 10
    requestMemory: "40Gi"
    requestGPU: 1
    pvcStorage: "50Gi"
    storageClass: <STORAGE_CLASS>
    vllmConfig:
      enablePrefixCaching: true
      maxModelLen: 16384
    lmcacheConfig:
      enabled: false
      cpuOffloadingBufferSize: "20"
    hf_token: <HF_TOKEN>
    initContainer:
      name: "wait-for-cache-server"
      image: "dp.apps.rancher.io/containers/lmcache-vllm-openai:0.3.2"
      command: ["/bin/sh", "-c"]
      args:
        - |
          timeout 60 bash -c '
          while true; do
            /opt/venv/bin/python3 /workspace/LMCache/examples/kubernetes/
health_probe.py $(RELEASE_NAME)-cache-server-service $(LMCACHE_SERVER_SERVICE_PORT)
 && exit 0
            echo "Waiting for LMCache server..."
            sleep 2
          done'
cacheserverSpec:
  replicaCount: 1
  containerPort: 8080
  servicePort: 81
  serde: "naive"
  registry: "dp.apps.rancher.io"
  repository: "containers/lmcache-vllm-openai"
  tag: "0.3.2"
  resources:
    requests:
      cpu: "4"
      memory: "8G"
    limits:
      cpu: "4"
      memory: "10G"
  labels:
    environment: "cacheserver"
    release: "cacheserver"
routerSpec:
  resources:
    requests:
      cpu: "1"
      memory: "2G"
    limits:
      cpu: "1"
      memory: "2G"
  routingLogic: "session"
  sessionKey: "x-user-id"
```

4.8 Installing mcpo

MCP (Model Context Protocol) is an open source standard for connecting AI applications—such as SUSE AI—to external systems. These external systems can include data sources like databases or local files, or tools like calculators or search engines.

mcpo is the MCP-to-OpenAPI proxy server provided by Open WebUI. It solves communication compatibility issues, enables cloud and UI integrations, and offers increased security and scalability.

4.8.1 Details about the mcpo application

Before deploying mcpo, it is important to know more about the supported configurations and documentation. The following command provides the corresponding details:

```
helm show values oci://dp.apps.rancher.io/charts/open-webui-mcpo
```

Alternatively, you can also refer to the mcpo Helm chart page on the SUSE Application Collection site at https://apps.rancher.io/applications/open-webui-mcpo ↗. It contains mcpo dependencies, available versions and the link to pull the mcpo container image.

4.8.2 mcpo installation procedure

Tip

Before the installation, you need to get user access to the SUSE Application Collection, create a Kubernetes namespace, and log in to the Helm registry as described in *Section 4.1, “Installation procedure”*.

1. Create a `mcpo_custom_overrides.yaml` file to override the default values of the Helm chart. The following file defines multiple MCP servers in the `config.mcpServers` section. These servers will be added to the mcpo configuration file `config.json`.

```
# mcpo_custom_overrides.yaml
global:
  imagePullSecrets:
  - application-collection
config:
  mcpServers:
    memory:
      command: npx
      args:
        - -y
        - "@modelcontextprotocol/server-memory"
    time:
      command: uvx
      args:
        - mcp-server-time
        - --local-timezone=America/New_York
    fetch:
      command: uvx
      args:
        - mcp-server-fetch
    weather:
      command: uvx
      args:
        - --from
        - git+https://github.com/adhikasp/mcp-weather.git
        - mcp-weather
      env:
        - ACCUWEATHER_API_KEY: your_api_key_here</screen>
```

2. After saving the override file as `mcpo_custom_overrides.yaml`, apply its configuration with the following command.

```
> helm upgrade --install \
  mcpo oci://dp.apps.rancher.io/charts/open-webui-mcpo \
  -n SUSE_AI_NAMESPACE \
  -f mcpo_custom_overrides.yaml
```

Tip

Installing MCP servers. You can add new MCP servers by including them in the mcpo configuration file following the Claude Desktop MCP format (https://github.com/modelcontextprotocol/servers). For detailed information on installing MCP servers with mcpo, refer to the mcpo Quick Usage guide (https://github.com/open-webui/mcpo?tab=readme-ov-file#-quick-usage).

4.8.3 Integrating mcpo with Open WebUI

To integrate mcpo with Open WebUI, follow these steps:

REQUIREMENTS

- You must have Open WebUI administrator privileges to access configuration screens or settings mentioned in this section.

1. In the bottom left of the Open WebUI window, click your avatar icon to open the user menu and select *Admin Panel*.
2. Click the *Settings* tab and select *Tools* from the left menu.
3. Under *Manage Tool Servers*, click the `plus` icon to add a new connection.
4. For each MCP server:
   - Provide the server URL, name and description.
   - Set the visibility to *Public* to make it available to all users.
   - Check if the connection is successful and confirm with *Save*.

   Tip

   The general URL format is: MCPO_URL/MCP_SERVER_NAME. For example, if mcpo was deployed as `mcpo` in the namespace `suse-ai` with the default port configuration, the URL is:

   ```
   http://mcpo-open-webui-mcpo.suse-ai.svc.cluster.local:8000/
   MCP_SERVER_NAME
   ```

After you have configured at least one MCP server, you can enable them from the Open WebUI chat input field to make answers more specific. For more information, see Selecting mcpo services from the chat input field (https://documentation.suse.com/suse-ai/1.0/html/openwebui-using/index.html#openwebui-input-field-specify-mcpo-tools).

Tip: Enabling MCP tools by default

To enable selected MCP tools by default for a model, refer to Enabling default MCP services (https://documentation.suse.com/suse-ai/1.0/html/openwebui-configuring/index.html#owui-enabling-default-mcp-services).

4.8.4 Upgrading mcpo

The mcpo chart receives application updates and updates of the Helm chart templates. New versions may include changes that require manual steps. These steps are listed in the corresponding `README` file. All mcpo dependencies are updated automatically during an mcpo upgrade.

To upgrade mcpo, identify the new version number and run the following command below:

```
> helm upgrade --install \
  mcpo oci://dp.apps.rancher.io/charts/open-webui-mcpo \
  -n SUSE_AI_NAMESPACE \
  --version VERSION_NUMBER \
  -f mcpo_custom_overrides.yaml
```

Tip

If you omit the `--version` option, mcpo gets upgraded to the latest available version.

4.8.5 Uninstalling mcpo

To uninstall mcpo, run the following command:

```
> helm uninstall mcpo -n SUSE_AI_NAMESPACE
```

4.9 Installing PyTorch

PyTorch is a widely used open-source deep-learning framework that supports both CPU and GPU acceleration. When deployed with the SUSE AI stack, the PyTorch Helm chart lets you inject your own training or inference code into the container and run it on NVIDIA GPUs available in your cluster.

4.9.1 Details about the PyTorch application

Before deploying PyTorch, it is important to know more about the supported configurations and documentation. The following command provides the corresponding details:

```
helm show values oci://dp.apps.rancher.io/charts/pytorch
```

Alternatively, you can also refer to the PyTorch Helm chart page (https://apps.rancher.io/applications/pytorch) ↗. It contains PyTorch dependencies, available versions and the link to pull the PyTorch container image.

4.9.2 PyTorch installation procedure

Tip

Before the installation, you need to get user access to the SUSE Application Collection, create a Kubernetes namespace, and log in to the Helm registry as described in *Section 4.1, "Installation procedure"*.

1. Create a `pytorch_custom_overrides.yaml` file to override the values of the parent Helm chart. Find examples of PyTorch override files in *Section 4.9.5, "Examples of PyTorch Helm chart override files"* and a list of all valid options and their values in *Section 4.9.6, "Values for the PyTorch Helm chart"*.

2. Install the PyTorch Helm chart using the `pytorch_custom_overrides.yaml` file using the following command.

```
> helm upgrade --install \
  pytorch oci://dp.apps.rancher.io/charts/pytorch \
  -n SUSE_AI_NAMESPACE \
  -f pytorch_custom_overrides.yaml
```

4.9.3 Upgrading PyTorch

You can upgrade PyTorch to a specific version by running the following command:

```
> helm upgrade \
  pytorch oci://dp.apps.rancher.io/charts/pytorch \
  -n SUSE_AI_NAMESPACE \
  --version VERSION_NUMBER \
  -f pytorch_custom_overrides.yaml
```

Tip

If you omit the `--version` option, PyTorch gets upgraded to the latest available version.

4.9.4 Uninstalling PyTorch

To uninstall PyTorch, run the following command:

```
> helm uninstall pytorch -n SUSE_AI_NAMESPACE
```

4.9.5 Examples of PyTorch Helm chart override files

To override the default values during the Helm chart installation or update, you can create an override YAML file with custom values. Then, apply these values by specifying the path to the override file with the `-f` option of the `helm` command. Remember to replace <SUSE_AI_NAMESPACE> with your Kubernetes namespace.

EXAMPLE 16: BASIC OVERRIDE FILE WITH GPU ENABLED

```
# pytorch_custom_overrides.yaml
runtimeClassName: nvidia
global:
  imagePullSecrets:
    - application-collection ❶
image:
  registry: dp.apps.rancher.io
  repository: containers/pytorch
  tag: "2.7.0-nvidia"
  pullPolicy: IfNotPresent
persistence:
  enabled: true
  storageClass: local-path ❷
gpu:
  enabled: true
  type: 'nvidia'
  number: 1
```

1. Instructs Helm to use credentials from the SUSE Application Collection. For instructions on how to configure the image pull secrets for the SUSE Application Collection, refer to the official documentation (https://docs.apps.rancher.io/get-started/authentication/).
2. Use `local-path` storage only for testing purposes. For production use, we recommend using a storage solution suitable for persistent storage, such as SUSE Storage.

EXAMPLE 17: CONFIGMAP-BASED UPLOAD

To create a ConfigMap, run the following command:

```
> kubectl describe configmap \
  MY_CONFIG_MAP -n SUSE_AI_NAMESPACE
```

```
# pytorch_custom_overrides.yaml
runtimeClassName: nvidia
global:
  imagePullSecrets:
    - application-collection
image:
  registry: dp.apps.rancher.io ①
  repository: containers/pytorch
  tag: "2.7.0-nvidia"
  pullPolicy: IfNotPresent
persistence:
  enabled: true
  storageClass: local-path ②
gpu:
  enabled: true
  type: 'nvidia'
  number: 1

configMapExtFiles: "my-config-files" ③
```

1. Instructs Helm to use credentials from the SUSE Application Collection. For instructions on how to configure the image pull secrets for the SUSE Application Collection, refer to the official documentation (https://docs.apps.rancher.io/get-started/authentication/).
2. Use `local-path` storage only for testing purposes. For production use, we recommend using a storage solution suitable for persistent storage, such as SUSE Storage.
3. Specifies ConfigMap files.

EXAMPLE 18: HOST-FOLDER WITH FILES BAKED INTO THE CHART

Move the `entrypoint.sh` file plus any helper files under the `scripts/` directory.

```
# pytorch_custom_overrides.yaml
runtimeClassName: nvidia
global:
  imagePullSecrets:
    - application-collection ❶
image:
  registry: dp.apps.rancher.io
  repository: containers/pytorch
  tag: "2.7.0-nvidia"
  pullPolicy: IfNotPresent
persistence:
  enabled: true
  storageClass: local-path ❷
gpu:
  enabled: true
  type: 'nvidia'
  number: 1

entrypointscript:
  filename: "entrypoint.sh" ❸
  arguments: [] ❹
```

❶ Instructs Helm to use credentials from the SUSE Application Collection. For instructions on how to configure the image pull secrets for the SUSE Application Collection, refer to the official documentation (https://docs.apps.rancher.io/get-started/authentication/) ↗.

❷ Use `local-path` storage only for testing purposes. For production use, we recommend using a storage solution suitable for persistent storage, such as SUSE Storage.

❸ The file will be mounted and accessible at `/workspace/entrypoint.sh`.

❹ Add custom command-line arguments if needed.

EXAMPLE 19: GIT REPOSITORY CLONE: PUBLIC WITH NO AUTHENTICATION

```
# pytorch_custom_overrides.yaml
runtimeClassName: nvidia
global:
  imagePullSecrets:
    - application-collection ❶
image:
  registry: dp.apps.rancher.io
  repository: containers/pytorch
  tag: "2.7.0-nvidia"
  pullPolicy: IfNotPresent
persistence:
  enabled: true
  storageClass: local-path ❷
gpu:
  enabled: true
  type: 'nvidia'
  number: 1

gitClone:
  enabled: true
  repository: "github.com/YOUR_ORGANIZATOIN/YOUR_REPO" ❸
  revision: "main" ❹
```

❶ Instructs Helm to use credentials from the SUSE Application Collection. For instructions on how to configure the image pull secrets for the SUSE Application Collection, refer to the official documentation (https://docs.apps.rancher.io/get-started/authentication/) ↗.

❷ Use `local-path` storage only for testing purposes. For production use, we recommend using a storage solution suitable for persistent storage, such as SUSE Storage.

❸ Do not specify the protocol, such as `https://`.

❹ Specify a branch name, a tag name or a commit.

EXAMPLE 20: GIT REPOSITORY CLONE: PRIVATE WITH AUTHENTICATION

```
# pytorch_custom_overrides.yaml
runtimeClassName: nvidia
global:
  imagePullSecrets:
    - application-collection ❶
image:
  registry: dp.apps.rancher.io
  repository: containers/pytorch
  tag: "2.7.0-nvidia"
  pullPolicy: IfNotPresent
persistence:
  enabled: true
  storageClass: local-path ❷
gpu:
  enabled: true
  type: 'nvidia'
  number: 1

gitClone:
  enabled: true
  repository: "github.com/YOUR_ORGANIZATOIN/YOUR_REPO" ❸
  revision: "main" ❹
  secretName: "MY_GIT_CREDENTIALS" ❺
```

❶ Instructs Helm to use credentials from the SUSE Application Collection. For instructions on how to configure the image pull secrets for the SUSE Application Collection, refer to the official documentation (https://docs.apps.rancher.io/get-started/authentication/) ↗.

❷ Use `local-path` storage only for testing purposes. For production use, we recommend using a storage solution suitable for persistent storage, such as SUSE Storage.

❸ Do not specify the protocol, such as `https://`.

❹ Specify a branch name, a tag name or a commit.

❺ Specify a preconfigured secret with username and password (or token).

4.9.6 Values for the PyTorch Helm chart

To override the default values during the Helm chart installation or update, you can create an override YAML file with custom values. Then, apply these values by specifying the path to the override file with the `-f` option of the `helm` command. Remember to replace <SUSE_AI_NAMESPACE> with your Kubernetes namespace.

TABLE 4: AVAILABLE OPTIONS FOR THE PYTORCH HELM CHART

Key	Type	Default	Description
`global.imageRegistry`	string	`""`	Global override for the container-image registry used by all chart images.
`global.imagePullSecrets`	list(string)	`[]`	Global list of image-pull secrets to attach to all pods.
`image.registry`	string	`dp.apps.rancher.io`	Registry that hosts the PyTorch container image.
`image.repository`	string	`containers/pytorch`	Repository name (path) of the PyTorch container image.
`image.tag`	string	`"2.5.0-nvidia"`	Image tag to deploy (CUDA/NVIDIA build by default).

Key	Type	Default	Description
`image.pullPolicy`	string	`IfNotPresent`	Kubernetes pull policy for the PyTorch image.
`imagePullSecrets`	list(string)	`[]`	Additional pull secrets (overrides global.imagePullSecrets).
`nameOverride`	string	`""`	Replace the chart name in resource names.
`fullnameOverride`	string	`""`	Fully override the generated release name.
`gpu.enabled`	bool	`false`	Enable GPU scheduling and automatically add device requests/limits.
`gpu.type`	string	`"nvidia"`	GPU vendor: `'nvidia'` or `'amd'`. If set to `'amd'`, a *rocm* image tag is inferred.
`gpu.number`	int	`1`	Number of full GPUs requested (ignored when MIG is used).
`gpu.nvidiaResource`	string	`nvidia.com/gpu`	Requested resource name; change to a MIG slice (e.g. `nvidia.com/mig-1g.10gb`) to schedule MIG devices.
`gpu.mig.enabled`	bool	`false`	Enable explicit specification of multiple MIG device types.
`gpu.mig.devices`	map	`{}`	Map of *MIG-slice-name* → *count* pairs (e.g. `1g.10gb: 1`).
`podAnnotations`	map	`{}`	Custom annotations added to the PyTorch pod.

Key	Type	Default	Description
`podLabels`	map	`{}`	Additional labels added to the PyTorch pod.
`podSecurityContext`	map	`{}`	Pod-level security context (e.g. `fsGroup`).
`securityContext`	map	`{}`	Container-level security context (capabilities, runAsUser, etc.).
`service.enabled`	bool	`false`	Create a ClusterIP/NodePort/LoadBalancer service for the PyTorch container.
`service.type`	string	`ClusterIP`	Service type when `service.enabled` is `true`.
`service.port`	int	(unset)	External service port.
`service.containerPort`	int	(unset)	Target container port inside the pod.
`service.nodePort`	int/string	`""`	Fixed nodePort value (for `type: NodePort`).
`service.loadBalancerIP`	string	(unset)	Requested load-balancer IP.
`service.loadBalancerClass`	string	(unset)	Load-balancer implementation class.
`service.annotations`	map	`{}`	Extra annotations applied to the Service object.
`serviceAccount.create`	bool	`false`	Whether to create a dedicated ServiceAccount.
`serviceAccount.automount`	bool	`true`	Auto-mount ServiceAccount token in the pod.

Key	Type	Default	Description
`serviceAccount.annotations`	map	`{}`	Annotations added to the ServiceAccount.
`serviceAccount.name`	string	`""`	Explicit ServiceAccount name (otherwise auto-generated).
`ingress.enabled`	bool	`false`	Create an Ingress exposing the service.
`ingress.className`	string	`""`	Explicit `IngressClass` to use.
`ingress.annotations`	map	`{}`	Extra annotations for the Ingress.
`ingress.hosts`	list	`[{host: chart-example.local, paths: [{/ , ImplementationSpecific}]}]`	Default host and path definitions.
`ingress.tls`	list	`[]`	TLS blocks for the Ingress resource.
`resources.requests`	map	`{}`	Pod resource requests (CPU / memory / GPU).
`resources.limits`	map	`{}`	Pod resource limits (CPU / memory / GPU).
`livenessProbe.enabled`	bool	`false`	Enable liveness probe.

Key	Type	Default	Description
`livenessProbe.initialDelaySeconds`	int	`5`	Delay before first liveness probe.
`livenessProbe.periodSeconds`	int	`5`	Interval between liveness probes.
`livenessProbe.timeoutSeconds`	int	`20`	Probe timeout.
`livenessProbe.failureThreshold`	int	`6`	Consecutive failures before restart.
`livenessProbe.successThreshold`	int	`1`	Successes needed to mark pod healthy.
`readinessProbe.enabled`	bool	`false`	Enable readiness probe.
`readinessProbe.initialDelaySeconds`	int	`5`	Delay before first readiness probe.
`readinessProbe.periodSeconds`	int	`5`	Interval between readiness probes.

Key	Type	Default	Description
`readinessProbe.timeoutSeconds`	int	`20`	Probe timeout.
`readinessProbe.failureThreshold`	int	`6`	Consecutive failures before marking pod unready.
`readinessProbe.successThreshold`	int	`1`	Successes required to mark pod ready.
`volumes`	list	`[]`	Extra Kubernetes volumes attached to the deployment.
`volumeMounts`	list	`[]`	Extra `volumeMounts` in the container spec.
`nodeSelector`	map	`{}`	Node-selector labels for pod scheduling.
`tolerations`	list	`[]`	Tolerations added to the pod spec.
`affinity`	map	`{}`	Affinity/anti-affinity rules for the pod.
`entrypointscript.filename`	string	`""`	Name (and path) of a startup script inside the container.
`entrypointscript.arguments`	list(string)	`[]`	CLI arguments passed to the entry point script.
`persistence.enabled`	bool	`false`	Provision a PVC to persist data (e.g. checkpoints).

Key	Type	Default	Description
`persistence.accessModes`	list(string)	`["ReadWriteOnce"]`	Access modes for the PVC.
`persistence.annotations`	map	`{}`	Annotations applied to the PVC.
`persistence.existingClaim`	string	`""`	Use an existing PVC instead of creating a new one.
`persistence.size`	string	`30Gi`	Requested storage size for the PVC.
`persistence.storageClass`	string	`""`	StorageClass used for dynamic provisioning (`""` → default).
`persistence.volumeMode`	string	`""`	Optional PV `volumeMode`.
`persistence.subPath`	string	`""`	Subdirectory within the PV to mount.
`persistence.volumeName`	string	`""`	Bind the PVC to a pre-existing PV by name.
`gitClone.enabled`	bool	`false`	Clone a Git repository into the container at startup.
`gitClone.repository`	string	`""`	Repository to clone (`github.com/org/repo`, no protocol).

Key	Type	Default	Description
`gitClone.revision`	string	`""`	Branch, tag, or commit to checkout.
`gitClone.secretName`	string	`""`	Name of a Secret containing Git credentials (username/password or token).
`configMapExtFiles`	string	`""`	Name of the ConfigMap whose files will be mounted into the container.

4.10 Installing MLflow

MLflow is an open-source platform for managing the end-to-end machine learning lifecycle. It provides a centralized model registry to track and manage the entire lifecycle of machine learning models. MLflow includes tools for experiment tracking, model packaging, versioning and deployment. This helps streamline the transition from development to production, ensuring reproducibility and collaboration among data science teams.

This section describes how to deploy MLflow using either Docker or Helm on a Kubernetes cluster.

4.10.1 Installing MLflow using Helm on a Kubernetes cluster

Tip

Before the installation, you need to get user access to the SUSE Application Collection, create a Kubernetes namespace, and log in to the Helm registry as described in *Section 4.1, “Installation procedure”*.

1. Create a skeleton for a new `MLflow` Helm chart.

```
> helm create mlflow
```

The command creates an `mlflow` directory with the basic file structure for a chart.

2. Replace `mlflow/values.yaml` with the following content. Replace `CONTAINER_VERSION` with the current chart version.

```
# values.yaml
replicaCount: 1
image:
  repository: dp.apps.rancher.io/containers/mlflow
  pullPolicy: IfNotPresent
  tag: "CONTAINER_VERSION"
imagePullSecrets:
  - name: application-collection
nameOverride: ""
fullnameOverride: ""
serviceAccount:
  # Specifies whether a service account should be created
  create: true
  # Automatically mount a ServiceAccount's API credentials?
  automount: true
  # Annotations to add to the service account
  annotations: {}
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the fullname template
  name: ""
podAnnotations: {}
podLabels: {}
podSecurityContext: {}
  # fsGroup: 2000
securityContext: {}
  # capabilities:
  #   drop:
  #   - ALL
  # readOnlyRootFilesystem: true
  # runAsNonRoot: true
  # runAsUser: 1000
service:
  type: ClusterIP
  port: 5000
ingress:
  enabled: true
  className: ""
  annotations: {}
    # kubernetes.io/ingress.class: nginx
    # kubernetes.io/tls-acme: "true"
  hosts:
    - host: suse-mlflow
      paths:
        - path: /
          pathType: ImplementationSpecific
  tls: []
  #  - secretName: chart-example-tls
  #    hosts:
  #      - chart-example.local
resources:
   limits:
     cpu: "2"
     memory: "2Gi"
   requests:
     cpu: "1"
     memory: "1Gi"
livenessProbe:
  httpGet:
    path: /health
    port: 5000
readinessProbe:
  httpGet:
    path: /health
    port: 5000
autoscaling:
  enabled: false
  minReplicas: 1
  maxReplicas: 100
  targetCPUUtilizationPercentage: 80
  # targetMemoryUtilizationPercentage: 80
# Additional volumes on the output Deployment definition.
volumes: []
# - name: foo
#   secret:
#     secretName: mysecret
#     optional: false
# Additional volumeMounts on the output Deployment definition.
volumeMounts: []
# - name: foo
#   mountPath: "/etc/foo"
#   readOnly: true
nodeSelector: {}
tolerations: []
affinity: {}</screen>
```

3. Replace `mlflow/template/deployment.yaml` with the following content:

```
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: {{ include "mlflow.fullname" . }}
  labels:
    {{- include "mlflow.labels" . | nindent 4 }}
spec:
  {{- if not .Values.autoscaling.enabled }}
  replicas: {{ .Values.replicaCount }}
  {{- end }}
  selector:
    matchLabels:
      {{- include "mlflow.selectorLabels" . | nindent 6 }}
  template:
    metadata:
      {{- with .Values.podAnnotations }}
      annotations:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      labels:
        {{- include "mlflow.labels" . | nindent 8 }}
        {{- with .Values.podLabels }}
        {{- toYaml . | nindent 8 }}
        {{- end }}
    spec:
      {{- with .Values.imagePullSecrets }}
      imagePullSecrets:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      serviceAccountName: {{ include "mlflow.serviceAccountName" . }}
      securityContext:
        {{- toYaml .Values.podSecurityContext | nindent 8 }}
      containers:
        - name: {{ .Chart.Name }}
          securityContext:
            {{- toYaml .Values.securityContext | nindent 12 }}
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag |
 default .Chart.AppVersion }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          command:
            - /usr/bin/mlflow
            - server
            - --host
            - "0.0.0.0"
            - --port
            - "5000"
          ports:
            - name: http
```

```
              containerPort: {{ .Values.service.port }}
              protocol: TCP
          livenessProbe:
            {{- toYaml .Values.livenessProbe | nindent 12 }}
          readinessProbe:
            {{- toYaml .Values.readinessProbe | nindent 12 }}
          resources:
            {{- toYaml .Values.resources | nindent 12 }}
          {{- with .Values.volumeMounts }}
          volumeMounts:
            {{- toYaml . | nindent 12 }}
          {{- end }}
      {{- with .Values.volumes }}
      volumes:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      {{- with .Values.nodeSelector }}
      nodeSelector:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      {{- with .Values.affinity }}
      affinity:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      {{- with .Values.tolerations }}
      tolerations:
        {{- toYaml . | nindent 8 }}
      {{- end }}
```

4. Install MLflow using the following command:

```
> helm install mlflow ./mlflow \
  -n SUSE_AI_NAMESPACE
```

5. Validate that Ingress is enabled for MLflow.

```
> kubectl get ingress --all-namespaces
NAMESPACE        AME       CLASS HOSTS                ADDRESS     PORTS     AGE
[...]
suse-private-ai  mlflow     nginx suse-mlflow         10.0.3.184  80        153m
suse-private-ai  pen-webui nginx suse-ollama-webui    10.0.3.184  80, 443   8h
```

4.10.1.1 Uninstalling MLflow

To uninstall MLflow, run the following command:

```
> helm uninstall mlflow -n SUSE_AI_NAMESPACE
```

4.10.2 Installing MLflow using Docker

There are two ways to install MLflow using Docker:

- By downloading and running the MLflow container (*Section 4.10.2.1, "Installing MLflow using a Docker container"*) directly.
- By creating an MLflow Docker Compose (*Section 4.10.2.2, "Installing MLflow using a Docker Compose YAML file"*) YAML file.

4.10.2.1 Installing MLflow using a Docker container

1. Download the MLflow container. Replace `CONTAINER_VERSION` with the current container version.

```
{prompt_user}docker pull dp.apps.rancher.io/containers/mlflow:CONTAINER_VERSION
```

1. *(Optional)* Verify the downloaded image.

```
{prompt_user}docker images
REPOSITORY                           TAG    IMAGE ID     CREATED      SIZE
dp.apps.rancher.io/containers/mlflow 3.6.0  d984124afc22 33 hours ago 715MB
```

1. Run the MLflow server by starting the container at port 5000. Replace `CONTAINER_VERSION` with the current container version.

```
{prompt_user} docker run -p 5000:5000 \
  dp.apps.rancher.io/containers/mlflow:CONTAINER_VERSION mlflow server \
  --host 0.0.0.0 --port 5000
[2025-11-27 18:34:15 +0000] [12] [INFO] Starting gunicorn 23.0.0
[2025-11-27 18:34:15 +0000] [12] [INFO] Listening at: http://0.0.0.0:5000 (12)
[2025-11-27 18:34:15 +0000] [12] [INFO] Using worker: sync
[2025-11-27 18:34:15 +0000] [13] [INFO] Booting worker with pid: 13
[2025-11-27 18:34:15 +0000] [14] [INFO] Booting worker with pid: 14
[2025-11-27 18:34:15 +0000] [15] [INFO] Booting worker with pid: 15
```

```
[2025-11-27 18:34:15 +0000] [16] [INFO] Booting worker with pid: 16
```

4.10.2.2 Installing MLflow using a Docker Compose YAML file

1. Create a `docker-compose.yaml` file with the following content:

```
services:
  mlflow:
    image: dp.apps.rancher.io/containers/mlflow:CONTAINER_VERSION
    container_name: mlflow
    restart: always
    ports:
      - "5000:5000"
    command:
      - /usr/bin/mlflow
      - server
      - --host
      - "0.0.0.0"
      - --port
      - "5000"
```

1. Run MLflow using the following command:

```
{prompt_user}docker-compose up -d
[...]
[+] Running 2/2
 \u2714 Network {exampleuser_plain}_default  Created                0.0s
 \u2714 Container mlflow      Started                         4s
```

1. *(Optional)* Verify that the container is running.

```
(venv) {exampleuser_plain}@localhost:~[] docker ps
CONTAINER ID IMAGE     ...     STATUS          PORTS                        NAMES
1e58723cb3d  mlflow:3.6.0      Up 23 seconds   0.0.0.0:5000->5000/tcp... mlflow
```

1. *(Optional)* Follow the logs to ensure that the MLflow server has started correctly.

```
{prompt_user}(venv) {exampleuser_plain}@localhost:~[] docker-compose logs -f
mlflow [2025-11-01 00:56:54 +0000] [3] [INFO] Starting gunicorn 23.0.0
mlflow [2025-11-01 00:56:54 +0000] [3] [INFO] Listening at: http://0.0.0.0:5000 (3)
mlflow [2025-11-01 00:56:54 +0000] [3] [INFO] Using worker: sync
mlflow [2025-11-01 00:56:54 +0000] [4] [INFO] Booting worker with pid: 4
mlflow [2025-11-01 00:56:54 +0000] [5] [INFO] Booting worker with pid: 5
mlflow [2025-11-01 00:56:55 +0000] [6] [INFO] Booting worker with pid: 6
mlflow [2025-11-01 00:56:55 +0000] [7] [INFO] Booting worker with pid: 7
```

4.10.3 Accessing MLflow Web UI

After the MLflow server is up and running, you can access it from a Web browser either on the local host or exposed via Ingress.

To access MLflow locally, point your Web browser to `http://localhost:5000`.

To access MLflow via Ingress, add a corresponding line to your `/etc/hosts`, for example:

```
10.0.3.184 suse-mflow
```

Then point your Web browser to `http://suse-mlflow`.

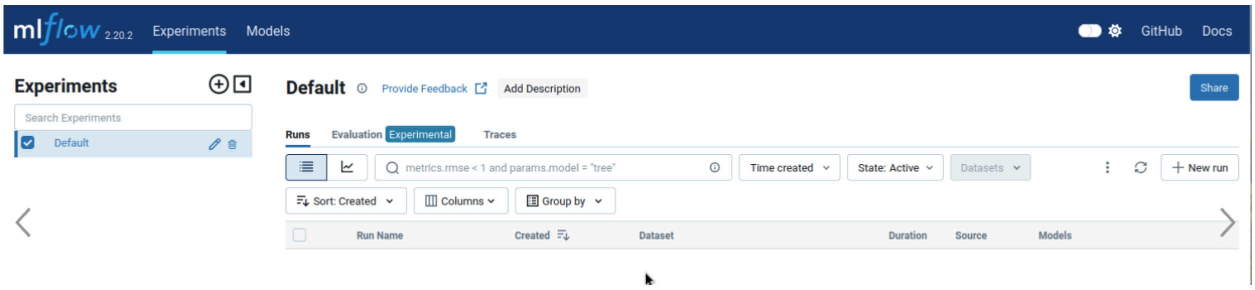


FIGURE 35: MLFLOW WEB UI

Tip: For more information

Explore MLflow core features, model training, tracing (observability) and more by following the official documentation (https://github.com/mlflow/mlflow?tab=readme-ov-file#-core-components) ↗.

4.11 Installing AI Library components using SUSE Deployer

SUSE Deployer consists of a meta Helm chart that takes care of downloading and installing individual AI Library components required by SUSE AI on a Kubernetes cluster.

The following procedure describes how to customize and use the SUSE Deployer to install AI Library components. It assumes that you already completed steps described in *Section 4.1, “Installation procedure”* including the installation of cert-manager.

1. Pull the SUSE Deployer Helm chart with the relevant chart version and untar it. You can find the latest of the chart on the SUSE Application Collection page at https://apps.rancher.io/applications/suse-ai-deployer.

   ```
   > helm pull oci://dp.apps.rancher.io/charts/suse-ai-deployer \
     --version 1.0.0 --untar
   > cd suse-ai-deployer
   ```

2. Inspect the downloaded chart and its default values.

   ```
   > helm show chart .
   > helm show values .
   ```

   ### Tip

   To see default values for the charts of the individual components within the meta chart, run the following commands.

   ```
   > helm show values charts/ollama/
   > helm show values charts/open-webui/
   > helm show values charts/milvus/
   > helm show values charts/pytorch
   ```

3. Explore downloaded example override files in the `suse-ai-deployer/examples` subdirectory. It typically includes the following files:

   `suse-gen-ai-minimal.yaml`

   Basic configuration to get started with GenAI. It deploys Ollama without GPU support, Open WebUI, and Milvus in stand-alone mode using local storage. PyTorch is disabled.

   `suse-gen-ai.yaml`

   Configuration optimized for production usage. It deploys Ollama with GPU support, Open WebUI, and Milvus in cluster mode using Longhorn storage. PyTorch is disabled.

   `suse-ml-stack.yaml`

   Basic configuration that enables deployment of PyTorch with no GPU support with Longhorn storage. It deploys PyTorch but disables Ollama, Open WebUI and Milvus.

4. Create `custom-overrides.yaml` override file based one of the above examples. The examples use self-signed certificates for TLS communication. To use other option (see *Section 4.6.6.1, "TLS sources"*), copy the `global` section from the `values.yaml` file into your `custom-overrides.yaml` and update its `tls` section as needed.

5. Install the SUSE Deployer Helm chart with while overriding values from the `custom-overrides.yaml` file. Use the appropriate `RELEASE_NAME` and `SUSE_AI_NAMESPACE` based the configuration in `custom-overrides.yaml`.

```
> helm upgrade --install \
  RELEASE_NAME \
  --namespace  SUSE_AI_NAMESPACE \
  --create-namespace \
  --values ./custom-overrides.yaml \
  --version 1.0.0 \
  oci://dp.apps.rancher.io/charts/suse-ai-deployer
```

5 Steps after the installation is complete

Once the SUSE AI installation is finished, follow these tasks to complete the initial setup and configuration.

1. Log in to SUSE AI Open WebUI using the default credentials.

2. After you have logged in, update the administrator password for SUSE AI.

3. From the available language models, configure the one you prefer. Optionally, install a custom language model. Refer to the section Setting base AI models (https://documentation.suse.com/suse-ai/1.0/html/openwebui-configuring/index.html#openwebui-setting-base-models) ↗ and Setting the default AI model (https://documentation.suse.com/suse-ai/1.0/html/openwebui-configuring/index.html#openwebui-setting-default-models) ↗ for more details.

4. Configure user management with *role-base access control* (RBAC) as described in https://documentation.suse.com/suse-ai/1.0/html/openwebui-configuring/index.html#openwebui-managing-user-roles ↗.

5. Integrate *single sign-on* authentication manager—such as Okta—with Open WebUI as described in https://documentation.suse.com/suse-ai/1.0/html/openwebui-configuring/index.html#openwebui-authentication-via-okta ↗.
6. Configure *retrieval-augmented generation* (RAG) to let the model process content relevant to the customer.

Glossary

AI, artificial intelligence

Refers to the simulation of human intelligence in machines that are designed to learn and solve problems like humans. Enables computers to understand language, make decisions and improve from experience.

Air gap

A security measure where a computer network is physically isolated from unsecured networks, including the public Internet.

Batch size

The number of samples processed simultaneously during model inference, affecting processing speed and resource utilization.

BYOC, bring your own certificate

A practice allowing users to provide their own SSL/TLS certificates for securing communications instead of using default or auto-generated ones.

CA, certification authority

An entity that issues digital certificates to verify the identity of certificate holders and ensure secure communications.

Chain-of-thought (CoT) prompting

A prompting technique that guides AI models to break down complex problems into step-by-step reasoning processes, improving response accuracy and transparency.

Chat template

A structured format for organizing conversations between users and AI models, defining how system prompts, user inputs, and AI responses are formatted and processed.

Context window

The maximum amount of text (tokens) that an AI model can process at once, including both the input prompt and generated response.

CRD, custom resource definitions

Extensions of the Kubernetes API that allow users to define custom resources and their controllers in a Kubernetes cluster.

CUDA, Compute Unified Device Architecture

NVIDIA's parallel computing platform and programming model used to accelerate AI workloads on GPU hardware.

Data leakage

The unintended exposure of sensitive information through AI model responses, potentially compromising data security and privacy.

Embeddings

Numerical representations of data (text, images, etc.) in a high-dimensional space that capture semantic relationships and enable AI models to process information effectively.

Fine-tuning

The process of further training a pre-trained AI model on specific data to adapt it for particular tasks or domains, improving its performance for targeted applications.

GenAI, generative AI

A type of artificial intelligence that can create new content such as text, images or music.

GPU, graphics processing unit

Specialized hardware designed for parallel processing. In AI applications, GPUs accelerate model training and inference tasks.

Hallucination

An AI behavior where the model generates false or unsupported information that appears plausible but has no basis in provided context or real facts.

Helm

A package manager for Kubernetes that helps install and manage applications. Helm uses charts to define, install and upgrade complex Kubernetes applications.

Helm chart
: A package format for Kubernetes applications that contains all resource definitions needed to deploy and configure application workloads.

IaC, infrastructure as code
: The practice of managing and provisioning infrastructure through machine-readable definition files rather than manual processes.

Inference
: The process of using a trained AI model to make predictions or generate outputs based on new input data.

Kubernetes pods
: The smallest deployable units in Kubernetes that can host one or more containers, sharing networking and storage resources.

LLM, large language model
: An advanced AI model trained on amounts of text data to understand and generate human-like text. Can perform tasks like translation, summarization and answering questions.

Model weights
: The learned parameters of an AI model that determine how it processes inputs and generates outputs. These weights are adjusted during training to optimize model performance.

NLG, natural language generation
: A process of automatically generating human-like text from structured data or other forms of input. Designed to convert raw data into coherent and meaningful language easily understood by humans.

NLU, natural language understanding
: A process AI uses to analyze and understand the meaning of the input query.

NVIDIA GPU driver
: Software that enables communication between the operating system and NVIDIA graphics hardware, essential for GPU-accelerated AI workloads.

NVIDIA GPU Operator
: A Kubernetes operator that automates the management of NVIDIA GPUs in container environments, handling driver deployment, runtime configuration, and monitoring.

Ollama

An open source framework for running and serving AI models locally. Ollama simplifies the process of downloading, running and managing large language models.

OpenGL

A cross-platform API for rendering 2D and 3D graphics, commonly used in visualization applications and GPU-accelerated computing.

Prompt Engineering

The practice of crafting effective input queries to AI models to obtain desired and accurate outputs. Good prompt engineering helps prevent hallucinations and improves response quality.

Prompt injection

A security vulnerability where malicious inputs attempt to override or bypass an AI model's system prompt or safety constraints.

Quantization

A technique to reduce AI model size and computational requirements by converting model parameters to lower precision formats while maintaining acceptable performance.

RAG, retrieval-augmented generation

A technique that enhances AI responses by retrieving relevant information from a knowledge base before generating answers, improving accuracy and reducing hallucinations.

RBAC, role-based access control

A security model that restricts system access based on roles assigned to users, managing permissions and authorization in Kubernetes clusters.

Semantic search

A search method using AI to understand the meaning and context of queries rather than just matching keywords, enabling more relevant results.

System prompt

Initial instructions given to an AI model that define its behavior, role and response parameters. System prompts help maintain consistent and appropriate AI responses.

Temperature

A parameter controlling the randomness in AI model outputs. Lower values produce more focused and deterministic responses, while higher values increase creativity and variability.

Token

The basic unit of text processing in AI models, representing parts of words, characters or symbols. Models process text by breaking it into tokens for analysis and generation.

Top-K

A parameter that limits token selection during text generation to the K most likely next tokens, helping control output quality and relevance.

Top-P

Also known as nucleus sampling, a parameter that selects from the smallest set of tokens whose cumulative probability exceeds P, providing dynamic control over text generation diversity.

Vector database

A specialized database designed to store and efficiently query high-dimensional vectors that represent data in AI applications, enabling similarity searches and semantic operations.

Vector store

A specialized storage system optimized for managing and querying vector embeddings, essential for semantic search and RAG implementations in AI applications.

A Copyright

B GNU Free Documentation License

B1 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

B2 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that

B3 2. VERBATIM COPYING

B4 3. COPYING IN QUANTITY

B5 4. MODIFICATIONS

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

B6 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

B7 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

B8 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

B9 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.