Containerized Service Router Simulator (SR-SIM)

Release 25.7.R1

SR-SIM Installation, Setup, and Deployment Guide

3HE 21883 AAAA TQZZA 01
Edition: 01
July 2025

Nokia is committed to diversity and inclusion. We are continuously reviewing our customer documentation and consulting with standards bodies to ensure that terminology is inclusive and aligned with the industry. Our future customer documentation will be updated accordingly.

Table of contents

List of tables

List of figures

1 Getting started

1.1 About this guide

This guide describes how to install, deploy, and setup the Nokia Containerized Service Router Simulator (SR-SIM) for the 7250 IXR, 7450 ESS, 7750 SR, and 7950 XRS router.

Note: Unless otherwise indicated, command line interface (CLI) commands, contexts, and configuration examples in this guide apply for both the MD-CLI and the classic CLI.

This guide is organized into functional chapters and includes:

- a functional overview of the SR-SIM tool
- a description of the SR-SIM system architecture
- requirements for containerization infrastructure supporting the SR-SIM
- initial commissioning procedures to bring up the SR-SIM in various environments and modelling various hardware platforms

Command outputs shown in this guide are examples only; actual outputs may differ depending on supported functionality and user configuration.

Deployment examples provided in this guide assume the Linux host operating system. Commands and outputs may vary on other host operating systems.

Note: This guide generically covers 25.*x*.R*x* content and may contain some content that will be released in later maintenance loads. For information about features supported in each load of the Release 25.*x*.R*x* software or for a list of unsupported features by platform and chassis, see the *SR OS R25.x.Rx Software Release Notes*, part number 3HE 21562 000*x* TQZZA.

1.1.1 Audience

This guide is intended for anyone who is creating SR-SIMs in a qualified lab environment. It is assumed that the reader has an understanding of the following:

- x86 hardware architecture
- Linux system installation, configuration, and administration methods
- basic YAML syntax
- 7250 IXR, 7450 ESS, 7750 SR, and 7950 XRS router chassis components
- SR OS CLI (Classic and Model-Driven)
- UNIX/Linux system administration principles
- networking principles and configurations, including virtualized I/O techniques in a containerized environment

1.1.2 List of technical publications

After the SR-SIM is successfully deployed, see the product-specific *Guide to Documentation* for a list of applicable SR OS user guides for the 7250 IXR, 7450 ESS, 7750 SR, or 7950 XRS. These guides contain information about the software configuration and CLI that is used to configure network parameters and services on that router.

1.1.3 Conventions

This section describes the general conventions used in this guide.

1.1.3.1 Precautionary and information messages

The following information symbols are used in the documentation.

DANGER: Danger warns that the described activity or situation may result in serious personal injury or death. An electric shock hazard could exist. Before you begin work on this equipment, be aware of hazards involving electrical circuitry, be familiar with networking environments, and implement accident prevention procedures.

WARNING: Warning indicates that the described activity or situation may, or will, cause equipment damage, serious performance problems, or loss of data.

Caution: Caution indicates that the described activity or situation may reduce your component or system performance.

Note: Note provides additional operational information.

Tip: Tip provides suggestions for use or best practices.

1.1.3.2 Options or substeps in procedures and sequential workflows

Options in a procedure or a sequential workflow are indicated by a bulleted list. In the following example, at step 1, the user must perform the described action. At step 2, the user must perform one of the listed options to complete the step.

Example: Options in a procedure

1. User must perform this step.
2. This step offers three options. User must perform one option to complete this step.
   - This is one option.
   - This is another option.
   - This is yet another option.

Substeps in a procedure or a sequential workflow are indicated by letters. In the following example, at step 1, the user must perform the described action. At step 2, the user must perform two substeps (a. and b.) to complete the step.

Example: Substeps in a procedure

1. User must perform this step.
2. User must perform all substeps to complete this action.
   a. This is one substep.
   b. This is another substep.

2 SR-SIM overview

2.1 SR-SIM overview

The Containerized Service Router Simulator, known as the SR-SIM, is a simulation tool available to Nokia customers who have an active SR-SIM subscription, to assist with:

- labs
- training
- education
- automation development
- network simulation
- emulating the control and management plane of a device under test (DUT) in preparation for deployment into a production network

The SR-SIM does not provide identical functionality to the hardware platforms as some functionality is provided by the hardware datapath implementation.

2.1.1 SR-SIM concept

The SR-SIM provides a simulation tool to emulate a 7250 IXR, 7450 ESS, 7750 SR, or 7950 XRS router. This tool is provided as a container and designed to run on an x86 system within common containerization systems, such as Docker and Kubernetes.

The SR-SIM is a containerized version of the SR OS software that simulates the software that runs on the hardware platforms. Configuration of hardware elements (such as provisioning line cards) and software elements (such as interfaces, network protocols, and services) is performed the same way as on the physical SR OS platforms.

2.1.2 Containerization benefits

The ability to use the Containerized Network Functions (CNFs) on commodity hardware (which previously depended on custom hardware), using standard IT containerization technologies, provides significant benefits for network operators, including:

- reduced CAPEX by using industry-standard hardware that is potentially easier to upgrade
- reduced OPEX (space, power, cooling) by consolidation of multiple functions on fewer physical platforms
- faster and simpler testing and rollout of new services
- flexibility to scale capacity up or down, as needed

- flexibility to create specific network topologies for a given scenario
- decoupling the operator from the physical lab location allowing engineers and developers to create SR-SIM in multiple locations

2.1.3 SR-SIM instance

The SR-SIM is delivered as a native container, rather than a virtual machine wrapped inside a container. A single SR-SIM instance comprises one or more containers coupled together.

Some functionality of the SR-SIM may be offloaded to the host machine (as is common practice in containerized solutions).

Note: Care must be taken to prevent over-subscription of host resources. SR-SIM containers do not need dedicated CPU cores or dedicated vRAM memory as this is arbitrated by the containerization system, but not providing enough resources will render the container unstable. In addition, combining SR-SIM containers with any other program on the host machine with intensive memory access requirements should be avoided for stability reasons. Recommended minimum memory requirements are provided to start the container, but may not meet the minimum memory required for your specific use case.

3 Obtaining the SR-SIM software

The SR-SIM container software is delivered as a single compressed file on the Nokia online customer portal.

To use the SR-SIM container image, it must be made available to the container management system. See Making the SR-SIM image available to the container management system for detailed information.

4 SR-SIM software licensing

Access to the SR-SIM tool is provided through a subscription license, valid for one year from the date the license is issued.

SR-SIM containers will fail to boot unless a valid license file is available. This license file is provided by Nokia and made available to the container.

The license file must be provided to each container by mounting the license in the following location within the container: `/nokia/license/license.txt`. To provide the license file to the SR-SIM container, use the file/directory mounting techniques or a Kubernetes a `ConfigMap` or `Secret`.

Note: The SR-SIM does not support FTP, TFTP, SCP, and SFTP license provisioning.

When the license expires, the SR-SIM will reboot.

5 Host machine requirements

The SR-SIM tool is a containerized solution that relies on CPU and memory resources shared with the host machine and other containers deployed on the host.

The SR-SIM can operate across multiple host machines as long as the containers have the required connectivity.

5.1 CPU

The SR-SIM is designed to run on an Intel x86 host CPU; the tool is not tested on machines powered by AMD. The SR-SIM will not boot on ARM CPUs.

5.2 Memory

The SR-SIM requires 2 GB minimum memory in order to boot. As line cards and functionality is added the memory requirements will grow. The platform matrix will provide the recommended minimum for the default router and card layouts for each device. These should be used as a guideline only. The memory requirements on the host system may vary.

5.3 Storage

The SR-SIM image is approximately 2 GB in size once uncompressed.

Each SR-SIM container requires a small amount of host machine storage for the container image itself.

Each SR-SIM also provides emulated `cf1:`, `cf2:`, and `cf3:` drives inside the emulated router. The content stored on these devices consumes the available space in the container. These devices can be specifically mounted to host directories, ephemeral volumes, or persistent volumes, as required.

To use a local storage location, or an ephemeral/persistent volume, mount it to one of the following locations in the container:

- `/cf1`
- `/cf2`
- `/cf3`

5.4 Network Interface Cards

The SR-SIM does not rely on the presence of specific network interface cards (NICs) in the underlying host machine. Any NIC may be used, as long as it is supported by the container management system. The SR-SIM is not optimized for throughput performance.

5.5 Software requirements

This section describes the software requirements for the SR-SIM.

5.5.1 Host operating system

The SR-SIM does not rely on a specific host operating system (OS).

Due to the number of OSs (and versions) available, the SR-SIM containers are not tested against any specific host OS. The host OS may potentially interact with the container management system or the SR-SIM containers in a way that affects the optimal functioning of the SR-SIM container.

5.5.2 Container management systems

The following table lists the container management systems that are tested to work with the SR-SIM.

Table 1: Tested container management systems

Solution	Tested version
Docker	28.2.2 (community)
Docker compose	2.36.2
Kubernetes (k3s)	v1.32.5+k3s1
Kubernetes (minikube)	1.36.0
Containerlab	0.69

6 SR-SIM functional models

The SR-SIM tool emulates a number of hardware routers. These routers are either pizza-box systems with integrated linecards, or chassis-based systems with multiple linecards per chassis. The operator can model both types of devices.

The pizza-box systems can be emulated using the integrated model.

The chassis-based systems can be emulated using the distributed model.

See Appendix A: SR-SIM supported hardware for information about the specific model implementation for the desired simulated system.

6.1 Integrated model

The integrated SR-SIM model allows operators to emulate pizza-box systems. A single container instance provides a self-contained router with control, management, and traffic-forwarding capabilities included.

The container has a single management interface available, as shown in the following figure.

Figure 1: Container instance

An integrated SR-SIM is created when the configured chassis type is `sr-1`, `sr-1s`, `ixr-r6`, `ixr-ec`, `ixr-e2`, or `ixr-e2c`. All other chassis types require a distributed model of deployment.

Note: The `sr1` and `sr-1s` chassis types do not refer to the following chassis:

- 7750 SR-1x-48D
- 7750 SR-1-24D
- 7750 SR-1-48D
- 7750 SR-1-92S
- 7750 SR-1-46S
- 7750 SR-1x-92S
- 7750 SR-1se

All FP5 small, fixed platforms require a distributed model of deployment.

While `sr-1`, `sr-1s`, `ixr-ec`, `ixr-e2`, and `ixr-e2c` chassis types are single-container combined systems without redundancy support, the `ixr-r6` chassis type can have two combined containers to support redundancy. Otherwise, the `ixr-r6` behaves as an integrated model, as both containers have combined CPM/IOM components.

6.1.1 Management interfaces

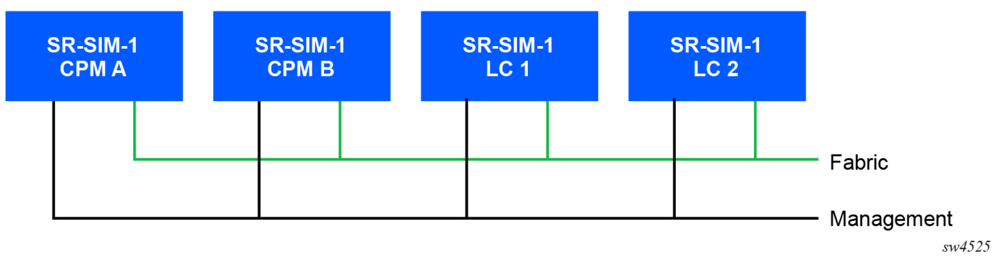
Integrated SR-SIM systems provide one management interface. The management interface is automatically assigned an IPv4 address by the container management system, if it is not explicitly set using the environment variable `NOKIA_SROS_ADDRESS_IPV4_ACTIVE`.

6.2 Distributed model

The distributed SR-SIM model allows operators to emulate chassis-based systems. Routers emulated in distributed mode require one container per linecard, with each container being specifically identified as supporting control plane processing (CPM) or datapath functions (IOM or XCM). For example, a system comprising two CPMs and two linecards requires four containers to be created.

A distributed SR-SIM is created when the configured chassis type is anything other than the `sr-1`, `sr-1s`, `ixr-r6`, `ixr-ec`, `ixr-e2`, or `ixr-e2c` chassis type.

Figure 2: Distributed model



6.2.1 Management interfaces

Distributed SR-SIM systems provide one management interface per CPM. The management interface is automatically assigned an IPv4 address by the container management system if it is not explicitly set using the environment variable `NOKIA_SROS_ADDRESS_IPV4_ACTIVE`.

Linecard containers (excluding CPM) should have a management interface attached to the container. However, any dynamically or statically assigned management IP addresses are not used by those containers.

6.2.2 Fabric interfaces

Distributed SR-SIM systems must have their containers connected together with a Layer 2 bridge interface. This may be between standalone containers, or in the case of Kubernetes, by deploying all containers in the same Kubernetes pod. This interface emulates the integrated backplane switch fabric of a hardware router.

Note: If all containers are deployed in the same Kubernetes pod, the pod will restart if any container is restarted.

The MTU of interfaces associated with SR-SIM internal fabric interfaces must be set to 9000 bytes. Packets sent over the fabric by each IOM/XCM or CPM are Ethernet encapsulated (without 802.1Q VLAN tags), and frames with a multicast or broadcast destination MAC address must be delivered to all containers of the SR-SIM instance.

7 Datapath interfaces

The SR-SIM allows the connection of underlying container Ethernet interfaces into SR OS ports, following a standardized naming convention.

The naming convention for SR OS ports typically takes one of the following formats: `L/X/M/C/P`, `L/M/C/P`, or `L/M/P` where:

- **L**: Linecard number
- **X**: XIOM number (when present)
- **M**: MDA position
- **C**: Cage or connector number
- **P**: Breakout port inside the connector

To ensure automatic connection of datapath interfaces from the container to their respective SR OS ports, use the following interface naming convention when attaching networks to the deployed containers: `eL-xX-M-cC-P`, `eL-M-cC-P`, or `eL-M-P`.

Note: The prefix e is added at the beginning of the port, and the forward slash `/` is replaced with a hyphen `-`. Some examples are listed in the following table.

Table 2: Naming convention examples

Container interface name	SR OS port name	Expanded description
e1-2-3	1/2/3	card 1, mda 2, port 3
e1-2-c3-1	1/2/c3/1	card 1, mda 2, connector 3, port 1
e2-2-c3-4	2/2/c3/4	card 2, mda 2, connector 3, port 4
e1-x2-3-4	1/x2/3/4	card 1, xiom 2, mda 3, port 4
e1-x2-3-c4-5	1/x2/3/c4/5	card 1, xiom 2, mda 3, connector 4, port 5

8 Environment variables

Environment variables are used to select the different SR-SIM components. The variables are a way of passing information from the container management system to the container itself. This information is then used to emulate the required hardware and provide BOF-specific initial configuration information.

It is not mandatory for users to configure environment variables. If environment variables are not provided, the SR-SIM automatically uses either statically defined or dynamically adjusted values for the variables.

The environment variable field names must be provided in uppercase letters. The value of the environment variable is case insensitive.

The following table lists the available environment variables and their default values.

Table 3: Environment variables and their default values

Environment variable	Default	Description
NOKIA_SROS_CHASSIS	sr-1	Chassis type
NOKIA_SROS_CHASSIS_TOPOLOGY	Unset	If the user requires an XRS-40, the chassis topology must be set to xrs40. This only applies when the NOKIA_SROS_CHASSIS variable is set to xrs-20 or xrs-20e.
NOKIA_SROS_SLOT	A	Identifies the slot the container relates to
NOKIA_SROS_SFM	*dynamic*	The SFM type (if required)
NOKIA_SROS_CARD	*dynamic*	The card type
NOKIA_SROS_XIOM_<M>	*dynamic*	The XIOM type (if required). <M> represents the XIOM position.
NOKIA_SROS_MDA_[<M>_]<N>	*dynamic*	The MDA type (if required). <M> and <N> represent the MDA position.
NOKIA_SROS_ADDRESS_IPV4_ACTIVE	*dynamic*	Lead CPM management IPv4 address. Dynamically assigned by the container management system if not set.
NOKIA_SROS_ADDRESS_IPV6_ACTIVE	*dynamic*	Lead CPM management IPv6 address. Dynamically assigned by the container management system if not set.
NOKIA_SROS_STATIC_ROUTE_<N>	0.0.0.0/0@<gateway_ip>[1]	Static route in the format of subnet/bits@nexthop, for example: 192.168.40.0/24@192.168.10.1.

[1] Where <gateway_ip> is replaced by the default gateway provided by the container management system. If no default gateway is provided, this will remain unset.

Environment variable	Default	Description
		<N> is an integer starting from 1, and should be incremented for each static route required.
NOKIA_SROS_SYSTEM_BASE_MAC	*dynamic*	The system base MAC address in the format of aa:bb:cc:dd:ee:ff, for example: de:ff:ab:c9:43:fe.
NOKIA_SROS_MGMT_IF	eth0	Management interface name inside the container
NOKIA_SROS_FABRIC_IF	eth1	Fabric interface name inside the container

9 Privileged mode

The SR-SIM should be run in `privileged` mode as shown in the examples in this document as it requires specific permissions from the host machine.

If needed, the SR-SIM may be run in non-privileged mode. However, the following operational capabilities must be provided specifically for the containers:

- **CHOWN**: to ensure the contents of cfX: are owned by the user “sros”
- **SYS_CHROOT**: to ensure the contents of cfX: are owned by the user “sros”
- **IPC_LOCK**: to facilitate memory management (mmap)
- **NET_ADMIN**: to allow for network interface control
- **NET_BIND_SERVICES**: to enable the ability to open ports < 1024
- **NET_RAW**: to allow for network socket control
- **SYS_RESOURCE**: to ensure access to message queues and file descriptors
- **SYS_TIME**: to manage clock and timing functions

Additionally, the following SYSCTL settings should be provided:

- **net.ipv4.conf.all.rp_filter**: Set to `0` to disable reverse-path filtering
- **net.ipv4.conf.default.rp_filter**: Set to `0` to disable reverse-path filtering
- **net.ipv6.conf.all.accept_ra**: Set to `0` to prevent sending router advertisement solicitations
- **net.ipv6.conf.default.accept_ra**: Set to `0` to prevent sending router advertisement solicitations

10 SR-SIM deployment

This chapter describes the SR-SIM deployment options.

10.1 Making the SR-SIM image available to the container management system

To use the SR-SIM container image in a containerized environment, an operator must import the downloaded image into a local container registry.

A container registry is a storage area made available to the container management system (such as Docker) to place copies of container images that are used to deploy and start containers.

Container registries can be of the following types:

- local/internal - stored on a customer authorized site and available to container management systems specifically authorized to obtain container images from the registry
- public - anyone on the internet may obtain a container image from the registry, with or without authorization

Note: Operators must not place the image on public container registries as this may breach their end-user license agreement (EULA) or export control laws.

10.1.1 Example: Importing the SR-SIM image into a local Docker container registry

The following is an example of the process to place the downloaded SR-SIM container image `srsim.tar.xz` (example filename) into a local container registry provided in the Docker container management system.

```
bash# docker load -i srsim.tar.xz
Loaded image: localhost/nokia/srsim:25.10.R1

bash# docker image ls
docker image ls localhost/nokia/srsim:25.10.R1
REPOSITORY              TAG         IMAGE ID       CREATED       SIZE
localhost/nokia/srsim   25.10.R1    7deaabababc0   10 days ago   2.1GB

bash# docker image ls
docker tag localhost/nokia/srsim:25.10.R1 srsim:25.10.R1
```

The image is now available for use, named `srsim:25.10.R1`.

10.1.2 Example: Importing the SR-SIM image into a local Kubernetes container registry

The following is an example of the process to place the downloaded SR-SIM container image `srsim.tar.xz` (example filename) into a local container registry available on `localhost:32000` that has been deployed into a Kubernetes environment.

Note: This example does not address the deployment of a local Kubernetes container registry. See the documentation of your Kubernetes management system for deployment instructions.

```
bash# docker load -i srsim.tar.xz
Loaded image: localhost/nokia/srsim:25.10.R1

bash# docker image ls
docker image ls localhost/nokia/srsim:25.10.R1
REPOSITORY             TAG          IMAGE ID       CREATED       SIZE
localhost/nokia/srsim   25.10.R1    7deaabababc0   10 days ago   2.1GB

bash# docker tag localhost/nokia/srsim:25.10.R1 localhost:32000/nokia/srsim:25.10.R1

bash# docker image ls
docker image ls localhost:32000/nokia/srsim:25.10.R1
REPOSITORY             TAG          IMAGE ID       CREATED       SIZE
localhost/nokia/srsim   25.10.R1    7deaabababc0   10 days ago   2.1GB

bash# docker push localhost:32000/nokia/srsim: 25.10.R1
The push refers to repository [localhost:32000/nokia/srsim]
01a1febdbeca: Mounted from srsim
25.10.R1: digest: sha256:7deaabababc08555ccea1ff26a7b5776ee8445e63c0e51bddbcc826544b966a6 size:
 52
```

The image is now available for use, named `localhost:32000/nokia/srsim:25.10.R1`.

10.2 Deploying the SR-SIM

This section describes how to deploy and configure the SR-SIM for use in some common container management systems.

An example is provided for each container management solution for an integrated model SR-SIM and a distributed model SR-SIM.

Note: The examples provided in the section assume that:

- the container image has been imported into a local registry and has been tagged appropriately
- the license subscription file provided by Nokia is stored on the local filesystem in `/tmp/license.txt`

10.2.1 Docker

Docker is one of the most common container management systems.

To correctly forward packets to interfaces facing the SR-SIM, disable the IP generic checksums on the bridge interfaces used. This bridge interface should also support an MTU of 9000.

You can use the default docker bridge interface or create SR-SIM bridges specifically for the management network.

Use the following UNIX command to disable TX generic IP checksums for the interface `<interface>`.

```
sudo ethtool -K <interface> tx-checksum-ip-generic off
```

By default, unless the environment variables provided override it, the first network attached to a container is treated as the management interface, and the second, as the fabric interface (where required).

10.2.1.1 Using Docker to deploy the SR-SIM

This section describes how to deploy the SR-SIM in integrated and distributed modes, using Docker.

In the deployment examples that follow, a management network is created as a Docker bridge network with the appropriate MTU and other required settings. This network is named `srsim_mgmt`.

Use the following commands to create and configure the management network `srsim_mgmt` in Docker.

```
docker network create --driver bridge --subnet 10.77.140.0/24 --gateway 10.77.140.1 --opt
 com.docker.network.driver.mtu=9000 --opt com.docker.network.bridge.name=srsim_mgmt srsim_mgmt
```

In the preceding case, the `srsim_mgmt` network is configured to automatically allocate a management IPv4 address to each SR-SIM container as it is started from the range `10.77.140.0/24`.

Use the following command to disable TX generic IP checksums for the `srsim_mgmt` interface.

```
sudo ethtool -K srsim_mgmt tx-checksum-ip-generic off
```

Use the following command to remove the management network at any time.

```
docker network rm srsim_mgmt
```

10.2.1.1.1 Integrated

Deploying the SR-SIM in integrated mode using Docker is straightforward. In the deployment example that follows, the command is used to create an SR-1 simulator named `srsim1` with the SSH protocol bound to port 2222 of the localhost.

This deployment (optionally) connects the current TTY and STDIN to this device, essentially providing console on the SR-SIM.

Note: This example is an unusual deployment in a container environment; most frequently, the SR-SIMs are deployed to run in the background.

```
docker run --privileged --rm --name srsim1 -t -i --network srsim_mgmt -p 2222:22 -v /tmp/
license.txt:/nokia/license/license.txt localhost:32000/srsim:latest
```

The command consists of the following component parts:

- `--priviledged` runs the container with additional privileged on the host machine. This is required in order to operate as a router with it's own networking stack, rather than running on top of another network stack.
- `--rm` (optional) deletes the container from the container management system (Docker) when the container stops. If this is not provided, when the container is stopped it will remain in a stopped state but will not be removed.
- `--name` (optional) is the name of the container. A name will be dynamically allocated by the container management system (Docker) if this is not provided.
- `-t` (optional) allocates a TTY to the container in order to allow console access to the router.
- `-i` (optional) ensures STDIN is redirected to the container in order to allow console access to the router.
- `-p 2222:22` redirects TCP port 2222 on the local system to port 22 inside the container. Port 22 is the SSH service on SR OS. This flag can be provided more than once to redirect additional ports following the format `-p <source port>:<destination port>` for a TCP port and `-p <source port>/udp:<destination port>/udp` for a UDP port.
- `-v /tmp/license.txt:/nokia/license/license.txt` mounts the local file `/tmp/license.txt` as the file `/nokia/license/license.txt` inside the container. This is the license file, however, the same approach may be taken to mount other files and directories inside the SR-SIM container.

To stop the created container, which was named `srsim1`, use the following command in another shell session:

```
docker container stop srsim1
```

As the `--rm` flag was used to create the container, the container is both stopped and removed.

The more common deployment of the SR-SIM is to run it in the background, so that it remains running, and use SSH to connect to the device. This more accurately mimics operational deployments.

To deploy the integrated mode SR-SIM in this way, remove the `-t` and `-i` flags from the command syntax, and add the `-d` flag. This detaches the container from the session, allowing it to run independently.

The deployment command is as follows.

```
docker run --privileged --rm --name srsim1 -d --network srsim_mgmt -p 2222:22 -v /tmp/
license.txt:/nokia/license/license.txt localhost:32000/srsim:latest
```

Use the following command to confirm the container is running.

```
docker container ls
```

Use the following `top` style tool to check your container.

```
docker stats
```

Use the following command to view the console and look at the boot sequence of the SR-SIM.

```
docker logs -f srsim1
```

Use the following command to identify the IP address allocated to the SR-SIM.

```
docker inspect srsim1 | grep "10.77.140" | grep "IPAddress"
```

To connect to the newly created SR-SIM, SSH to the local host machine (by using its IP address or name).

For example `ssh -l admin -p 2222 localhost` or `ssh -l admin 10.77.140.2` (assuming your SR-SIM was allocated the `10.77.140.2` address).

Detached containers are stopped in the same way as interactive containers, using this command.

```
docker container stop srsim1
```

As the `--rm` flag was used to create the container, the container is both stopped and removed.

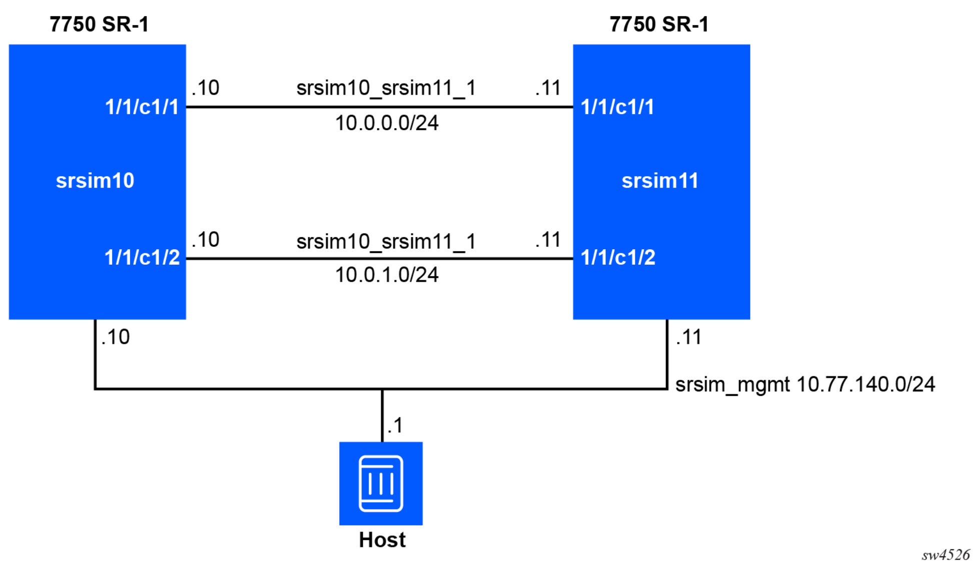
Use the following command to deploy an SR-SIM that is emulating an IXR-e2.

```
docker run --privileged --rm --name srsim1 -d --network srsim_mgmt -p 2222:22 -v /tmp/
license.txt:/nokia/license/license.txt -e NOKIA_SROS_CHASSIS=ixr-e2 srsim:latest
```

The additional `-e` flag is used in the preceding command. This sets an environment variable and it can be provided multiple times. As described in Environment variables, the SR-SIM uses environment variables to identify the hardware to emulate and perform some initial configuration.

The following topology can be deployed with the following Docker commands.

Figure 3: Example: Integrated mode network diagram



In the preceding example, while the management network `srsim_mgmt` is already set up, the user needs to create datapath networks `srsim10_srsim11_1` and `srsim10_srsim11_2` between the two routers. Use the following commands to create each datapath network.

```
docker network create srsim10_srsim11_1
```

```
docker network create srsim10_srsim11_2
```

```
docker run --privileged --rm --name srsim10 -d \
    --network srsim_mgmt --ip 10.77.140.10 \
    --network=name=srsim10_srsim11_1,driver-opt=com.docker.network.endpoint.ifname=e1-1-c1-1 \
    --network=name=srsim10_srsim11_2,driver-opt=com.docker.network.endpoint.ifname=e1-1-c1-2 \
    -v /tmp/license.txt:/nokia/license/license.txt \
    -v ./srsim10.cfg:/nokia/config/config.cfg \
    localhost:32000/srsim:25.10.R1

docker run --privileged --rm --name srsim11 -d \
    --network srsim_mgmt --ip 10.77.140.11 \
    --network=name=srsim10_srsim11_1,driver-opt=com.docker.network.endpoint.ifname=e1-1-c1-1 \
    --network=name=srsim10_srsim11_2,driver-opt=com.docker.network.endpoint.ifname=e1-1-c1-2 \
    -v /tmp/license.txt:/nokia/license/license.txt \
    -v ./srsim11.cfg:/nokia/config/config.cfg \
    localhost:32000/srsim:25.10.R1
```

The following important considerations apply to the preceding commands:

- The order of the `network` statements is important. The SR-SIM expects the management interface will be attached to `eth0` inside the container. To ensure this occurs automatically, specify the management network (in this example `srsim_mgmt`) as the first network in this configuration.
- The `--ip` option provisions a static IP address to the container. This IP address must be within the range configured on the network.
- The statement `--network=name=srsim10_srsim11_1,driver-opt=com.docker.network.endpoint.ifname=e1-1-c1-1` attaches a datapath interface to the container. Datapath interfaces in the SR-SIM are automatically bound to the SR OS ports. To achieve this binding, the container interface name must follow the matching convention for the ports. In this example, `e1-1-c1-1` will be bound to the SR OS port `1/1/c1/1`.
- The statement `-v ./srsim10.cfg:/nokia/config/config.cfg` mounts a valid SR OS configuration file into the container, which the SR-SIM uses to boot. If no configuration file is provided, the default configuration is used. If a configuration file is provided, it is used only if valid. The `-v` option takes the form `source:destination`. In this example, the `srsim10.cfg` file from the current directory is mounted inside the container as `/nokia/config/config.cfg`.

Use the following command to destroy the SR-SIM instances `srsim10` and `srsim11`.

```
docker container stop srsim10 srsim11
```

10.2.1.1.2 Distributed

Starting the SR-SIM in distributed mode using Docker requires slightly more preparation than the integrated mode, however it remains simple to deploy.

To deploy the SR-SIM in a distributed mode inside Docker requires consideration of the following additional items:

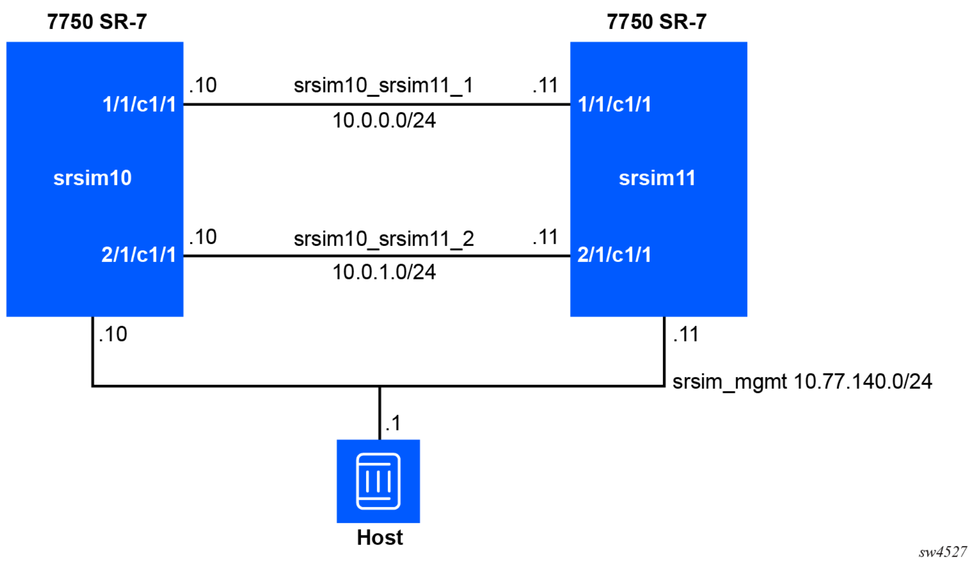
- each slot of the system will be its own container (including slots a and b, as well as the numbered slots `1` onward)
- each container requires the license file to be provided
- each container requires a management interface (even if it is not a CPM)

- each container requires access to the fabric bridge
- the fabric bridge should not be shared between SR-SIM devices

The following command creates an SR-7 simulator that comprises of two CPMs and two linecards. The SSH protocol is exposed on the primary and secondary CPMs (on different ports as Docker cannot expose the same local port to multiple containers). This deployment example runs all containers in the background.

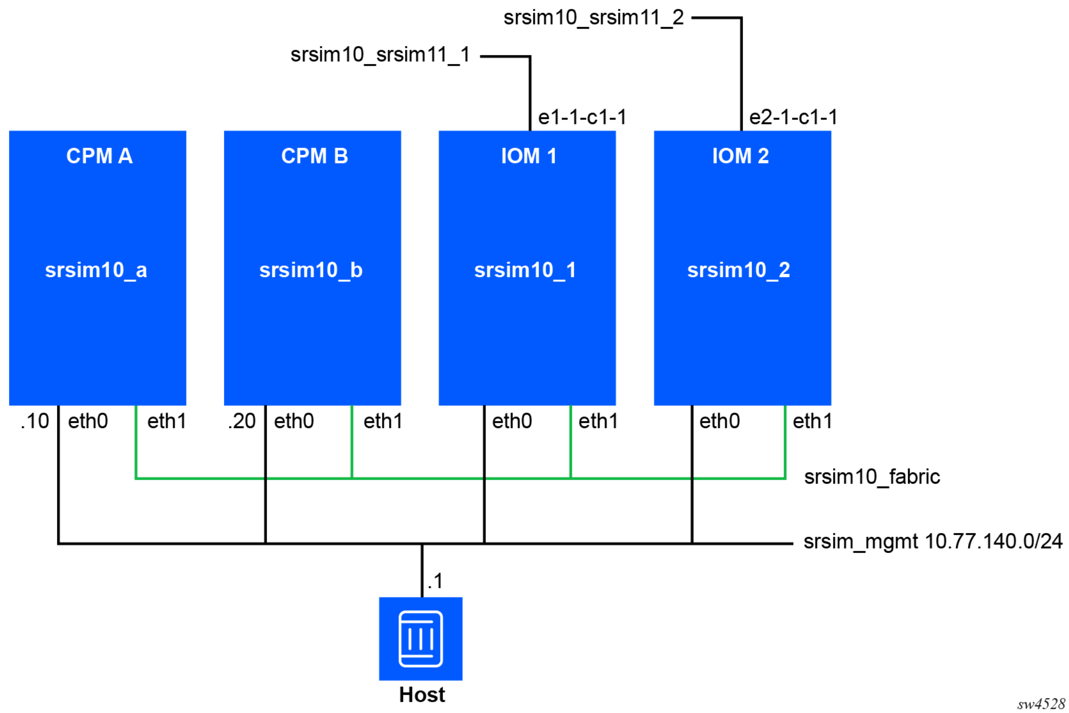
In the distributed mode, each linecard of each router requires its own container. The following figure shows the network diagram for the example network that will be created.

Figure 4: Example: Distributed mode network diagram



Each router consists of a number of containers. The `srsim10` router container layout is shown here for illustrative purposes. Each box is a separate container.

Figure 5: Example: Router container layout



Each container has a management connection, which is also the first interface in the container. The primary and standby CPM must have an assigned IP address on the management network. On both CPMs, the `NOKIA_SROS_ADDRESS_IPV4_ACTIVE` environment variable should be set to the chosen static IP address for the active CPM. This address is used for redundancy switchover.

Each container needs to have a fabric interface to connect them together. In the container layout example above, this is named `srsim10_fabric`. No IP addressing is required on this interface, it is a layer-2 bridge interface. By ensuring this interface is named `eth1` inside the container it will be used as the fabric interface.

The datapath networks are attached to their specific linecards and have interface names in the container that match the SR OS interface naming format. For example, `e2-1-c1-1` will be connected to port `2/1/c1/1` in the `srsim10_2` container (which is IOM 2).

The `srsim_mgmt` network has already been created. The following commands create the two datapath networks (srsim10_srsim11_1 and srsim10_srsim11_2) and the two fabric networks used to interconnect the containers acting as device linecards (`srsim10_fabric` and `srsim11_fabric`).

```
docker network create srsim10_srsim11_1
docker network create srsim10_srsim11_2
docker network create srsim10_fabric
docker network create srsim11_fabric
```

Use the following commands to create the eight containers required to deploy two 7750 SR-7 simulated devices using the SR-SIM in distributed mode.

```
docker run --privileged --rm --name srsim10_a -d \
    --network srsim_mgmt --ip 10.77.140.10 \
```

```
    --network=name=srsim10_fabric,driver-opt=com.docker.network.endpoint.ifname=eth1 \
    -e NOKIA_SROS_CHASSIS=sr-7 \
    -e NOKIA_SROS_SLOT=a \
    -e NOKIA_SROS_SYSTEM_BASE_MAC=de:ff:ab:c9:bb:aa \
    -e NOKIA_SROS_ADDRESS_IPV4_ACTIVE=10.77.140.10/24 \
    -v /tmp/license.txt:/nokia/license/license.txt \
    -v ./srsim10.cfg:/nokia/config/config.cfg \
    localhost:32000/srsim:25.10.R1

docker run --privileged --rm --name srsim10_b -d \
    --network srsim_mgmt --ip 10.77.140.20 \
    --network=name=srsim10_fabric,driver-opt=com.docker.network.endpoint.ifname=eth1 \
    -e NOKIA_SROS_CHASSIS=sr-7 \
    -e NOKIA_SROS_SLOT=b \
    -e NOKIA_SROS_SYSTEM_BASE_MAC=de:ff:ab:c9:bb:aa \
    -e NOKIA_SROS_ADDRESS_IPV4_ACTIVE=10.77.140.10/24 \
    -v /tmp/license.txt:/nokia/license/license.txt \
    -v ./srsim10.cfg:/nokia/config/config.cfg \
    localhost:32000/srsim:25.10.R1

docker run --privileged --rm --name srsim10_1 -d \
    --network srsim_mgmt \
    --network=name=srsim10_fabric,driver-opt=com.docker.network.endpoint.ifname=eth1 \
    --network=name=srsim10_srsim11_1,driver-opt=com.docker.network.endpoint.ifname=e1-1-c1-1 \
    -e NOKIA_SROS_CHASSIS=sr-7 \
    -e NOKIA_SROS_SLOT=1 \
    -v /tmp/license.txt:/nokia/license/license.txt \
    localhost:32000/srsim:25.10.R1

docker run --privileged --rm --name srsim10_2 -d \
    --network srsim_mgmt \
    --network=name=srsim10_fabric,driver-opt=com.docker.network.endpoint.ifname=eth1 \
    --network=name=srsim10_srsim11_2,driver-opt=com.docker.network.endpoint.ifname=e2-1-c1-1 \
    -e NOKIA_SROS_CHASSIS=sr-7 \
    -e NOKIA_SROS_SLOT=2 \
    -v /tmp/license.txt:/nokia/license/license.txt \
    localhost:32000/srsim:25.10.R1

docker run --privileged --rm --name srsim11_a -d \
    --network srsim_mgmt --ip 10.77.140.11 \
    --network=name=srsim11_fabric,driver-opt=com.docker.network.endpoint.ifname=eth1 \
    -e NOKIA_SROS_CHASSIS=sr-7 \
    -e NOKIA_SROS_SLOT=a \
    -e NOKIA_SROS_SYSTEM_BASE_MAC=de:ff:ab:c9:bb:bb \
    -e NOKIA_SROS_ADDRESS_IPV4_ACTIVE=10.77.140.11/24 \
    -v /tmp/license.txt:/nokia/license/license.txt \
    -v ./srsim11.cfg:/nokia/config/config.cfg \
    localhost:32000/srsim:25.10.R1

docker run --privileged --rm --name srsim11_b -d \
    --network srsim_mgmt --ip 10.77.140.21 \
    --network=name=srsim11_fabric,driver-opt=com.docker.network.endpoint.ifname=eth1 \
    -e NOKIA_SROS_CHASSIS=sr-7 \
    -e NOKIA_SROS_SLOT=b \
    -e NOKIA_SROS_SYSTEM_BASE_MAC=de:ff:ab:c9:bb:bb \
    -e NOKIA_SROS_ADDRESS_IPV4_ACTIVE=10.77.140.11/24 \
    -v /tmp/license.txt:/nokia/license/license.txt \
    -v ./srsim11.cfg:/nokia/config/config.cfg \
    localhost:32000/srsim:25.10.R1

docker run --privileged --rm --name srsim11_1 -d \
    --network srsim_mgmt \
    --network=name=srsim11_fabric,driver-opt=com.docker.network.endpoint.ifname=eth1 \
    --network=name=srsim10_srsim11_1,driver-opt=com.docker.network.endpoint.ifname=e1-1-c1-1 \
```

```
    -e NOKIA_SROS_CHASSIS=sr-7 \
    -e NOKIA_SROS_SLOT=1 \
    -v /tmp/license.txt:/nokia/license/license.txt \
    localhost:32000/srsim:25.10.R1

docker run --privileged --rm --name srsim11_2 -d \
    --network srsim_mgmt \
    --network=name=srsim11_fabric,driver-opt=com.docker.network.endpoint.ifname=eth1 \
    --network=name=srsim10_srsim11_2,driver-opt=com.docker.network.endpoint.ifname=e2-1-c1-1 \
    -e NOKIA_SROS_CHASSIS=sr-7 \
    -e NOKIA_SROS_SLOT=2 \
    -v /tmp/license.txt:/nokia/license/license.txt \
    localhost:32000/srsim:25.10.R1
```

The containers are named `srsim<number>_<slot_number>` to clearly identify which containers are performing. The container names also clearly indicate their functional role on the applicable router.

Use the following commands to stop the containers and remove all networks.

```
docker container stop srsim10_a srsim10_b srsim10_1 srsim10_2 srsim11_a srsim11_b srsim11_1
 srsim11_2
docker network rm srsim_mgmt srsim10_fabric srsim11_fabric srsim10_srsim11_1 srsim10_srsim11_2
```

10.2.2 Docker compose

Docker compose extends the provisioning options available with Docker to a single provisioning manifest. This manifest is named `docker-compose.yml` and defines the deployment of one (or more) SR-SIM instances (in either integrated or distributed mode) in terms of containers (referred to as `services` in Docker compose), `networks`, and storage (referred to as `volumes` in Docker compose).

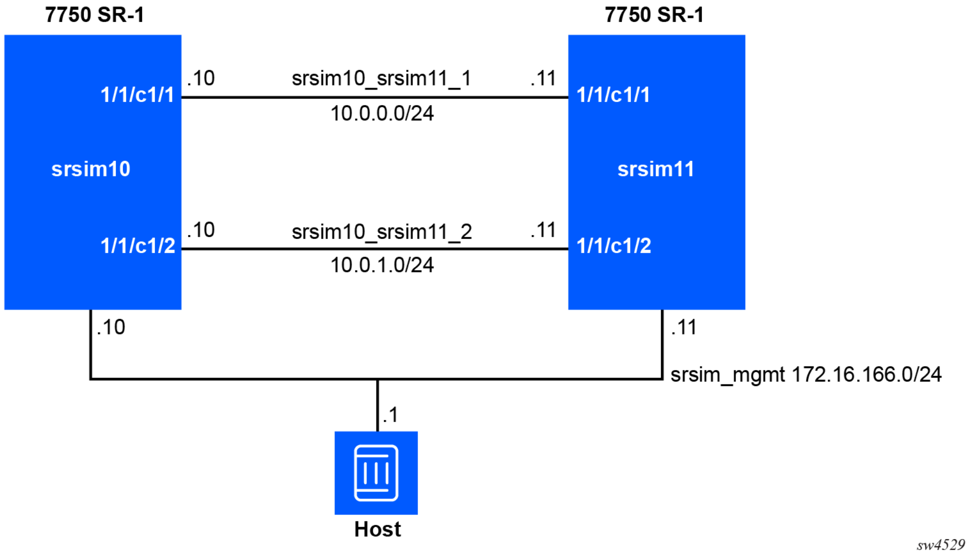
To deploy the instances defined in the `docker-compose.yml` manifest, navigate to the directory where the file is located and run the `docker compose` command.

Note: These deployment examples assume the SR-SIM software image has been placed in a local registry named `localhost:32000/srsim:25.10.R1`.

10.2.2.1 Integrated

The following figure shows an example manifest to provision a network comprising two SR-SIM instances connected using bridge interfaces.

Figure 6: Example: Network diagram for integrated mode manifest



The following `docker-compose.yml` manifest is used to deploy these two SR-SIM instances and the network connectivity between them.

```
services:
  srsim10:
    image: localhost:32000/srsim:25.10.R1
    privileged: true
    volumes:
      - /tmp/license.txt:/nokia/license/license.txt
      - ./srsim10.cfg:/nokia/config/config.cfg
    networks:
      srsim_mgmt:
        ipv4_address: 172.16.166.10
      srsim10_srsim11_1:
        interface_name: e1-1-c1-1
      srsim10_srsim11_2:
        interface_name: e1-1-c1-2

  srsim11:
    image: localhost:32000/srsim:25.10.R1
    privileged: true
    volumes:
      - /tmp/license.txt:/nokia/license/license.txt
      - ./srsim11.cfg:/nokia/config/config.cfg
    networks:
      srsim_mgmt:
        ipv4_address: 172.16.166.11
      srsim10_srsim11_1:
        interface_name: e1-1-c1-1
      srsim10_srsim11_2:
        interface_name: e1-1-c1-2

networks:
  srsim_mgmt:
    driver: bridge
    driver_opts:
      com.docker.network.driver.mtu: "9000"
```

```
      com.docker.network.bridge.name: "srsim_mgmt"
    ipam:
      driver: default
      config:
        - subnet: 172.16.166.0/24
          gateway: 172.16.166.1
  srsim10_srsim11_1:
  srsim10_srsim11_2:
```

This section provides detailed description of the key sections of this manifest.

```
networks:
  srsim_mgmt:
    driver: bridge
    driver_opts:
      com.docker.network.driver.mtu: "9000"
      com.docker.network.bridge.name: "srsim_mgmt"
    ipam:
      driver: default
      config:
        - subnet: 172.16.166.0/24
          gateway: 172.16.166.1
  srsim10_srsim11_1:
  srsim10_srsim11_2:
```

This part of the manifest creates three networks. Networks are created within the `networks` top-level section of the manifest. The first network is called `srsim_mgmt`. This network is a bridge network that will create the `srsim_mgmt` network on the host machine with the MTU set to `9000`. The `srsim_mgmt` bridge interface will be deployed with the `172.16.166.0/24` subnet assigned to it, from which Docker will allocate IP addresses to containers that join this network (either dynamically [by default] or statically as shown in this example). When the deployment is removed, this interface will remain on the host unless the following command is issued manually `sudo ifconfig srsim_mgmt down`.

The other two networks, `srsim10_srsim11_1` and `srsim10_srsim11_2`, are internal Docker networks. They are created within Docker and are not visible on the host machine once the deployment is removed. In this example, each link is created as its own network, which can also be deployed as a single shared broadcast network. Although it is not described in this section, when these networks are applied to the containers (SR-SIM instances), additional configuration will be applied.

The `services` section of the manifest describes the containers that will be created. In the following section, a single example `srsim10` container is considered.

```
services:
  srsim10:
    image: localhost:32000/srsim:25.10.R1
    privileged: true
    volumes:
      - /tmp/license.txt:/nokia/license/license.txt
      - ./srsim10.cfg:/nokia/config/config.cfg
    networks:
      srsim_mgmt:
        ipv4_address: 172.16.166.10
      srsim10_srsim11_1:
        interface_name: e1-1-c1-1
      srsim10_srsim11_2:
        interface_name: e1-1-c1-2
```

The preceding service (SR-SIM container) comprises the following component parts:

- The container name is the name of the service specified in the Docker compose manifest. In this example, the container will be named `srsim10`.
- The `image` statement specifies the source location of the SR-SIM software image. In this example, the `localhost:32000/srsim:25.10.R1` software image will be used.
- The `privileged: true` line grants the container additional permissions on the host machine, allowing it to manipulate the networking interfaces.
- The `volumes` section defines how files and directories are mounted inside the container. Two files are mounted in this example: the license file and the router configuration file. The SR-SIM will not boot without a valid license file. If the configuration file is not provided, the SR OSSR OS default configuration is used.
  - The `/tmp/license.txt:/nokia/license/license.txt` should be treated as `source:destination`. This line mounts the local `/tmp/license.txt` file inside the container as `/nokia/license/license.txt`.
  - The `./srsim10.cfg:/nokia/config/config.cfg` line mounts the `srsim10.cfg` in the same directory as the `docker-compose.yml` file inside the container as `/nokia/config/config.cfg`.
- The `networks` section defines the network interfaces for the container. Within each container, this section references the networks defined in the top-level `networks` section. The order in which these networks are used defines the order (and name) of the interfaces inside the container.
  - The `srsim_mgmt` entry adds the previously defined `srsim_mgmt` bridge network. The `ipv4_address` entry assigns a static IPv4 address to the network interface inside the container. This static IP address must be within the subnet defined in the bridge network definition. As the `srsim_mgmt` is defined first, it is attached inside the container as the `eth0` network interface.
  - The `srsim10_srsim11_1` network is attached next into the container, with the interface name `e1-1-c1-1`. The name assigned to the network interface is crucial, as the SR-SIM uses this name to determine where to attach the networks inside SR OS. In this example, the `srsim10_srsim11_1` network is attached into the container as the `e1-1-c1-1` interface, and SR OS knows that this interface is attached to port `1/1/c1/1`.

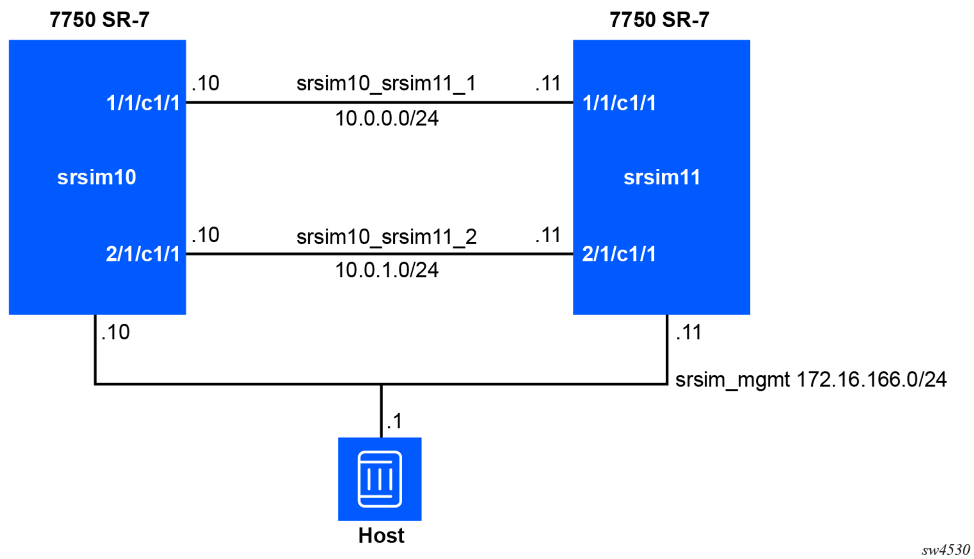
Use the `docker compose` command to deploy the manifest. The preferred method of deployment is to start the containers and run them in the background. The `docker compose up -d` command deploys the network described in the preceding example, allowing it to run in the background.

Use the `docker compose down --remove-orphans` command to destroy (undeploy) the network, and remove containers and associated networking. The `--remove-orphans` flag is optional, but useful to remove unused or stale Docker containers, networks, or volumes from the system.

10.2.2.2 Distributed

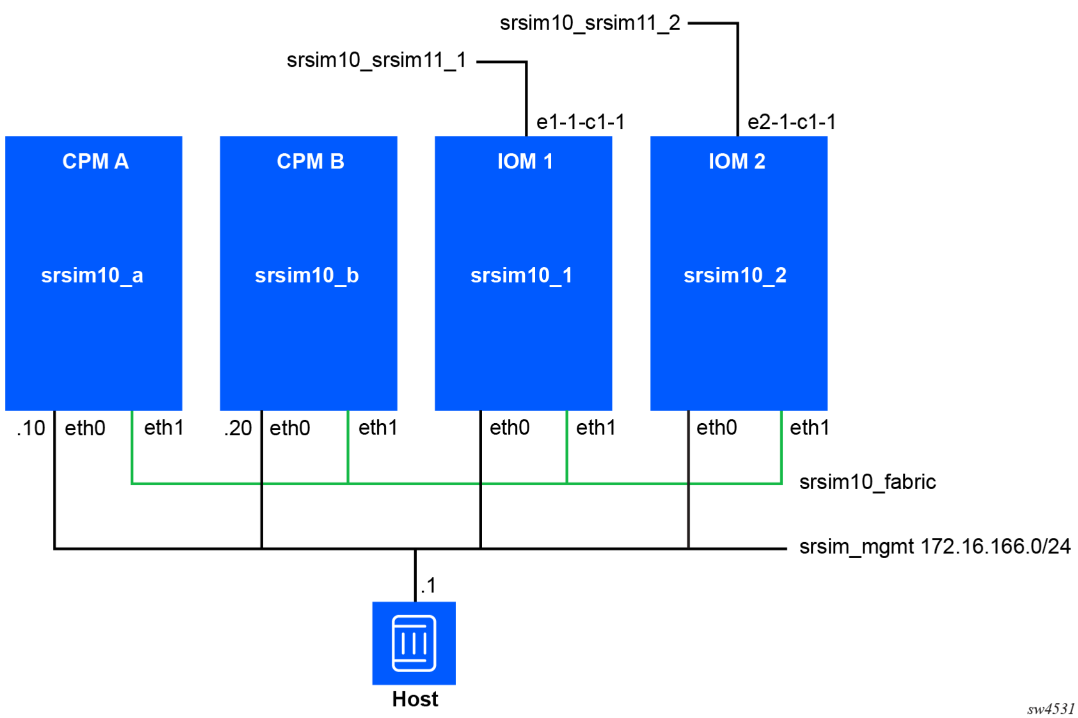
In the distributed mode, each linecard of each router requires its own container. The following figure shows the network diagram for this `docker-compose.yml` manifest.

Figure 7: Example: Network diagram for distributed mode



Each router consists of a number of containers. The following figure shows an example `srsim10` router container layout, where each box represents a separate container.

Figure 8: Example: Router container layout



Each container has a management connection, which is also the first interface in the container. It is required to have an IP address on the management network for the primary CPM and standby CPM. On both the active and standby CPM, the NOKIA_SROS_ADDRESS_IPV4_ACTIVE environment variable should be set to the chosen static IP address for the active CPM. This address is used for redundancy switchover.

Each container requires a fabric interface to connect them together. In the preceding container layout example, this interface is named srsim10_fabric. No IP addressing is required because it is a layer-2 bridge interface. Ensure that this interface is named eth1 inside the container, which designates it as the fabric interface.

The datapath networks are attached to their specific linecards and the interface names within the container that follow the SR OS interface naming format. For example, e2-1-c1-1 will be connected to port 2/1/c1/1 in the srsim10_2 container (which is IOM 2).

The following is an example manifest to provision the network comprising two SR-SIM instances connected together using bridge interfaces.

```
services:
  srsim10_a:
    image: localhost:32000/srsim:25.10.R1
    privileged: true
    volumes:
      - /tmp/license.txt:/nokia/license/license.txt
      - ./srsim10.cfg:/nokia/config/config.cfg
    environment:
      - NOKIA_SROS_CHASSIS=sr-7
      - NOKIA_SROS_SLOT=a
      - NOKIA_SROS_SYSTEM_BASE_MAC=de:ff:ab:c9:bb:aa
      - NOKIA_SROS_ADDRESS_IPV4_ACTIVE=172.16.166.10/24
    networks:
      srsim_mgmt:
        ipv4_address: 172.16.166.10
      srsim10_fabric:
        interface_name: eth1

  srsim10_b:
    image: localhost:32000/srsim:25.10.R1
    privileged: true
    volumes:
      - /tmp/license.txt:/nokia/license/license.txt
      - ./srsim10.cfg:/nokia/config/config.cfg
    environment:
      - NOKIA_SROS_CHASSIS=sr-7
      - NOKIA_SROS_SLOT=b
      - NOKIA_SROS_SYSTEM_BASE_MAC=de:ff:ab:c9:bb:aa
      - NOKIA_SROS_ADDRESS_IPV4_ACTIVE=172.16.166.10/24
    networks:
      srsim_mgmt:
        ipv4_address: 172.16.166.20
      srsim10_fabric:
        interface_name: eth1

  srsim10_1:
    image: localhost:32000/srsim:25.10.R1
    privileged: true
    volumes:
      - /tmp/license.txt:/nokia/license/license.txt
    environment:
      - NOKIA_SROS_CHASSIS=sr-7
      - NOKIA_SROS_SLOT=1
    networks:
      srsim_mgmt:
      srsim10_fabric:
```

```
        interface_name: eth1
      srsim10_srsim11_1:
        interface_name: e1-1-c1-1

  srsim10_2:
    image: localhost:32000/srsim:25.10.R1
    privileged: true
    volumes:
      - /tmp/license.txt:/nokia/license/license.txt
    environment:
      - NOKIA_SROS_CHASSIS=sr-7
      - NOKIA_SROS_SLOT=2
    networks:
      srsim_mgmt:
      srsim10_fabric:
        interface_name: eth1
      srsim10_srsim11_2:
        interface_name: e2-1-c1-1

  srsim11_a:
    image: localhost:32000/srsim:25.10.R1
    privileged: true
    volumes:
      - /tmp/license.txt:/nokia/license/license.txt
      - ./srsim11.cfg:/nokia/config/config.cfg
    environment:
      - NOKIA_SROS_CHASSIS=sr-7
      - NOKIA_SROS_SLOT=a
      - NOKIA_SROS_SYSTEM_BASE_MAC=de:ff:ab:c9:bb:bb
      - NOKIA_SROS_ADDRESS_IPV4_ACTIVE=172.16.166.11/24
    networks:
      srsim_mgmt:
        ipv4_address: 172.16.166.11
      srsim11_fabric:
        interface_name: eth1

  srsim11_b:
    image: localhost:32000/srsim:25.10.R1
    privileged: true
    volumes:
      - /tmp/license.txt:/nokia/license/license.txt
      - ./srsim11.cfg:/nokia/config/config.cfg
    environment:
      - NOKIA_SROS_CHASSIS=sr-7
      - NOKIA_SROS_SLOT=b
      - NOKIA_SROS_SYSTEM_BASE_MAC=de:ff:ab:c9:bb:bb
      - NOKIA_SROS_ADDRESS_IPV4_ACTIVE=172.16.166.11/24
    networks:
      srsim_mgmt:
        ipv4_address: 172.16.166.21
      srsim11_fabric:
        interface_name: eth1

  srsim11_1:
    image: localhost:32000/srsim:25.10.R1
    privileged: true
    volumes:
      - /tmp/license.txt:/nokia/license/license.txt
    environment:
      - NOKIA_SROS_CHASSIS=sr-7
      - NOKIA_SROS_SLOT=1
    networks:
      srsim_mgmt:
      srsim11_fabric:
```

```
        interface_name: eth1
      srsim10_srsim11_1:
        interface_name: e1-1-c1-1

  srsim11_2:
    image: localhost:32000/srsim:25.10.R1
    privileged: true
    volumes:
      - /tmp/license.txt:/nokia/license/license.txt
    environment:
      - NOKIA_SROS_CHASSIS=sr-7
      - NOKIA_SROS_SLOT=2
    networks:
      srsim_mgmt:
      srsim11_fabric:
        interface_name: eth1
      srsim10_srsim11_2:
        interface_name: e2-1-c1-1

networks:
  srsim_mgmt:
    driver: bridge
    driver_opts:
      com.docker.network.driver.mtu: "9000"
      com.docker.network.bridge.name: "srsim_mgmt"
    ipam:
      driver: default
      config:
        - subnet: 172.16.166.0/24
          gateway: 172.16.166.1
  srsim10_srsim11_1:
  srsim10_srsim11_2:
  srsim10_fabric:
  srsim11_fabric:
```

See the Integrated section for a detailed explanation of the `docker-compose.yml` manifest. In addition to the details used in that manifest, the following specific items are required to create the two SR-SIM instances in integrated mode (as 7750 SR-7 routers in this example).

In the `networks` top-level section, define a separate fabric network for each router. In this example, the `srsim10_fabric` and `srsim11_fabric` networks are defined.

The `srsim_mgmt` host bridge network and the networks to interconnect the routers (`srsim10_srsim11_1` and `srsim10_srsim11_2`) remain the same as shown in the integrated mode example manifest.

The following is an example of the CPM definitions (using the `srsim10`).

```
services:
  srsim10_a:
    image: localhost:32000/srsim:25.10.R1
    privileged: true
    volumes:
      - /tmp/license.txt:/nokia/license/license.txt
      - ./srsim10.cfg:/nokia/config/config.cfg
    environment:
      - NOKIA_SROS_CHASSIS=sr-7
      - NOKIA_SROS_SLOT=a
      - NOKIA_SROS_SYSTEM_BASE_MAC=de:ff:ab:c9:bb:aa
      - NOKIA_SROS_ADDRESS_IPV4_ACTIVE=172.16.166.10/24
    networks:
      srsim_mgmt:
        ipv4_address: 172.16.166.10
      srsim10_fabric:
```

```
        interface_name: eth1

  srsim10_b:
    image: localhost:32000/srsim:25.10.R1
    privileged: true
    volumes:
      - /tmp/license.txt:/nokia/license/license.txt
      - ./srsim10.cfg:/nokia/config/config.cfg
    environment:
      - NOKIA_SROS_CHASSIS=sr-7
      - NOKIA_SROS_SLOT=b
      - NOKIA_SROS_SYSTEM_BASE_MAC=de:ff:ab:c9:bb:aa
      - NOKIA_SROS_ADDRESS_IPV4_ACTIVE=172.16.166.10/24
    networks:
      srsim_mgmt:
        ipv4_address: 172.16.166.20
      srsim10_fabric:
        interface_name: eth1
```

The definitions now include environment variables. These environment variables are used to inform SR OS of specific provisioning requirements. In this case, each CPM defines the following:

- The router chassis type using the `NOKIA_SROS_CHASSIS` option, setting it to `sr-7` to tell the SR-SIM to mimic the 7750 SR-7.
- The slot number (or letter), the two CPM are provisioned as slot a and slot b respectively using the `NOKIA_SROS_SLOT` environment variable.
- In a dual CPM system, each CPM must have a system MAC address that is statically defined using the `NOKIA_SROS_SYSTEM_BASE_MAC` environment variable. This ensures that both CPMs act as a single redundant pair for the same router.
- The IP address of the active CPM must be provisioned into both SR OS CPMs using the `NOKIA_SROS_ADDRESS_IPV4_ACTIVE` environment variable.

Note: This variable requires an IP address value in CIDR notation (for example, `172.16.166.10/24`).

Looking at the IOM definitions (still using `srsim10` as the example) as shown.

```
srsim10_1:
    image: localhost:32000/srsim:25.10.R1
    privileged: true
    volumes:
      - /tmp/license.txt:/nokia/license/license.txt
    environment:
      - NOKIA_SROS_CHASSIS=sr-7
      - NOKIA_SROS_SLOT=1
    networks:
      srsim_mgmt:
      srsim10_fabric:
        interface_name: eth1
      srsim10_srsim11_1:
        interface_name: e1-1-c1-1

  srsim10_2:
    image: localhost:32000/srsim:25.10.R1
    privileged: true
    volumes:
      - /tmp/license.txt:/nokia/license/license.txt
    environment:
```

```
      - NOKIA_SROS_CHASSIS=sr-7
      - NOKIA_SROS_SLOT=2
    networks:
      srsim_mgmt:
      srsim10_fabric:
        interface_name: eth1
      srsim10_srsim11_2:
        interface_name: e2-1-c1-1
```

There are now specific containers for each datapath linecard (1 and 2 in this example), and each of these use the `NOKIA_SROS_SLOT` environment variable to inform SR OS of their position in the system.

Each datapath container also has a connection to the management network as its first interface, but without a statically assigned IP address. Docker automatically assigns an IP address to each container. However, as this address is not used to establish an SSH connection to the linecard, it is not relevant to the user.

Each container connects to the required fabric Docker network `srsim10_fabric` as the `eth1` interface, ensuring the SR OS uses it as the fabric interface.

As in the integrated mode, the datapath interfaces are automatically bound to SR OS ports if they are attached into the containers with the correct naming convention. In this example, each linecard has a single datapath interface. Specifically, IOM 1 has the `e1-1-c1-1` interface, which will be bound to the `1/1/c1/1` port, and IOM 2 has the `e2-1-c1-1` interface, which will be bound to the `2/1/c1/1` port.

Deploy the manifest using the `docker compose` command. The preferred method of deployment is to start the containers and run them in the background. Use the `docker compose up -d` command to deploy the network described in the preceding example, allowing it to run in the background.

Use the `docker compose down --remove-orphans` command to destroy (undeploy) the network, and remove the containers and associated networking. The `--remove-orphans` flag is optional, but useful to clean up any unused or stale Docker containers, networks, or volumes on the system.

10.2.3 Containerlab

Containerlab is a modern lab management solution for network engineers. It simplifies the creation of multi-vendor labs from a simple command-line tool. For information about Containerlab and the use of SR-SIM with it, review the containerlab documentation.

SR-SIM is supported on Containerlab using the `nokia_srsim` kind. Customization of the SR-SIM instances within Containerlab is achieved using the environment variables described in this document.

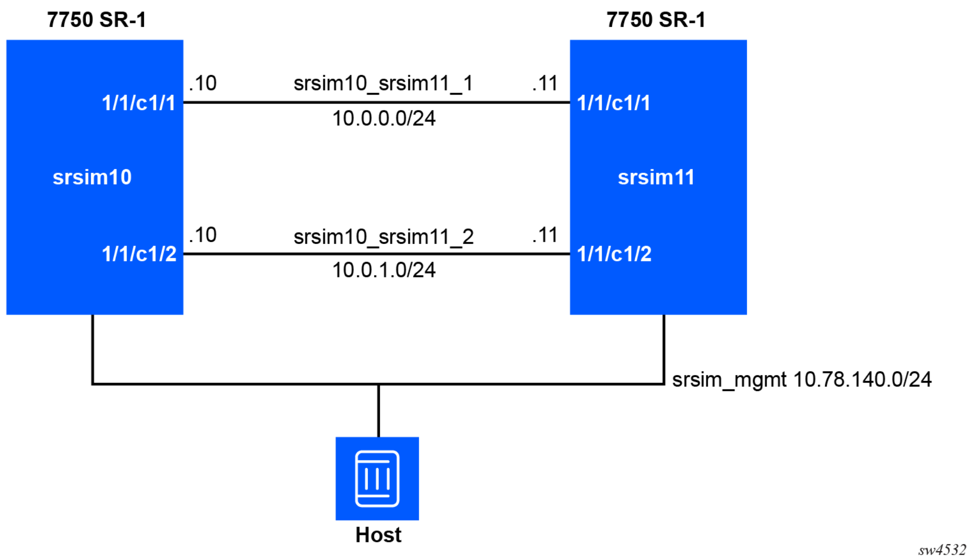
Containerlab supports the use of complete or partial configurations at startup with the SR-SIM.

Containerlab supports both integrated and distributed deployment models for the SR-SIM.

10.2.3.1 Integrated

The following figure shows an example Containerlab topology file to provision a network comprising two SR-SIM instances connected using bridge interfaces.

Figure 9: Example: Container topology provisioning SR-SIM instances with bridge interfaces



To deploy the preceding network in Containerlab, create the Containerlab topology file. In this example, the file is named `srsim.clab.yml`.

srsim.clab.yml:

```
name: "srsrim"
mgmt:
  network: srsim_mgmt
  ipv4-subnet: 10.78.140.0/24
topology:
  kinds:
    nokia_srsim:
      license: /tmp/license.txt
      image: localhost:32000/srsim:25.10.R1
  nodes:
    srsim10:
      kind: nokia_srsim
      type: SR-1 # Implicit default
      startup-config: srsim10.partial.cfg # Example partial config
    srsim11:
      kind: nokia_srsim
      startup-config: srsim11.cfg # Example complete config
  links:
    # Datapath interfaces
    - endpoints: ["srsim10:e1-1-c1-1", "srsim11:e1-1-c1-1"]
    - endpoints: ["srsim10:e1-1-c1-2", "srsim11:e1-1-c1-2"]
```

To deploy this lab, use the `containerlab` or `clab` command.

```
clab deploy -t srsim.clab.yml
```

When the nodes are booted and connected, output similar to the following is displayed:

Name	Kind/Image	State	IPv4/6 Address

```
| clab-srsrim-srsim10 | nokia_srsim                          | running | 10.78.140.2 |
|                     | localhost:32000/srsim:25.10.R1       |         | N/A         |
| clab-srsrim-srsim11 | nokia_srsim                          | running | 10.78.140.3 |
|                     | localhost:32000/srsim:25.10.R1       |         | N/A         |
```

To remove the deployed network, use the following command.

```
clab destroy -t srsim.clab.yml
```

The output displayed is similar to following:

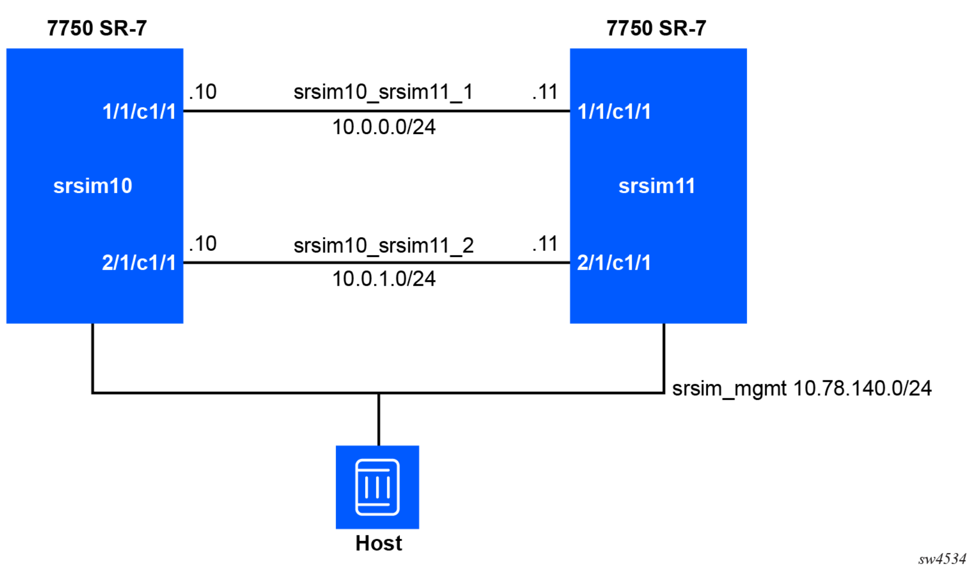
```
22:32:23 INFO Parsing & checking topology file=srsim.clab.yml
22:32:23 INFO Parsing & checking topology file=srsim.clab.yml
22:32:23 INFO Destroying lab name=srsrim
22:32:24 INFO Removed container name=clab-srsrim-srsim10
22:32:24 INFO Removed container name=clab-srsrim-srsim11
22:32:24 INFO Removing host entries path=/etc/hosts
22:32:24 INFO Removing SSH config path=/etc/ssh/ssh_config.d/clab-srsrim.conf
```

To also remove the filesystems created by the router instances, use the `--cleanup` option.

10.2.3.2 Distributed

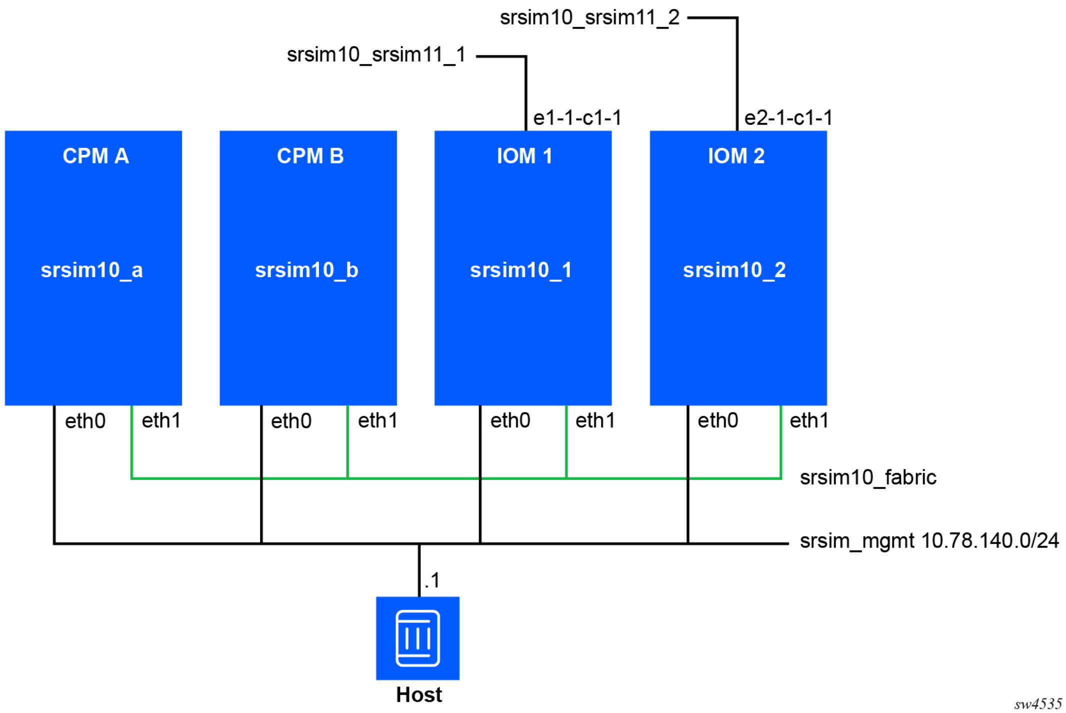
The following figure shows an example Containerlab topology file to provision a network comprising two SR-SIM instances connected using bridge interfaces.

Figure 10: Example: Containerlab topology provisioning two instances with bridge interfaces



Each router consists of a number of containers. The following figure shows an example `srsim10` router container layout, where each box represents a separate container. In Containerlab, it is important to ensure that all containers in a distributed mode deployment are configured to share the same network namespace. This is achieved using the `network_mode` option.

Figure 11: Example: Router container layout



To deploy the preceding network in Containerlab, create the Containerlab topology file. In this example, the file is named `srsim.clab.yml`.

```
name: "srsrim"
mgmt:
  network: srsim_mgmt
  ipv4-subnet: 10.78.140.0/24
topology:
  kinds:
    nokia_srsim:
      license: /tmp/license.txt
      image: localhost:32000/srsim:25.10.R1
  nodes:
    srsim10_fabric:
      kind: bridge
    srsim11_fabric:
      kind: bridge
    # srsim10
    srsim10-a:
      kind: nokia_srsim
      type: SR-7
      env:
        NOKIA_SROS_SLOT: A
        NOKIA_SROS_SYSTEM_BASE_MAC: 1c:58:07:00:03:01
      startup-config: srsim10.cfg
    srsim10-b:
      kind: nokia_srsim
      type: SR-7
      network-mode: container:srsim10-a
      env:
        NOKIA_SROS_SLOT: B
```

```
      NOKIA_SROS_SYSTEM_BASE_MAC: 1c:58:07:00:03:01
    startup-config: srsim10.cfg
  srsim10-1:
    kind: nokia_srsim
    type: SR-7
    network-mode: container:srsim10-a
    env:
      NOKIA_SROS_SLOT: 1
  srsim10-2:
    kind: nokia_srsim
    type: SR-7
    network-mode: container:srsim10-a
    env:
      NOKIA_SROS_SLOT: 2
  # srsim11
  srsim11-a:
    kind: nokia_srsim
    type: SR-7
    env:
      NOKIA_SROS_SLOT: A
      NOKIA_SROS_SYSTEM_BASE_MAC: 1c:58:07:00:03:02
    startup-config: srsim11.cfg
  srsim11-b:
    kind: nokia_srsim
    type: SR-7
    network-mode: container:srsim11-a
    env:
      NOKIA_SROS_SLOT: B
      NOKIA_SROS_SYSTEM_BASE_MAC: 1c:58:07:00:03:02
    startup-config: srsim11.cfg
  srsim11-1:
    kind: nokia_srsim
    type: SR-7
    network-mode: container:srsim11-a
    env:
      NOKIA_SROS_SLOT: 1
  srsim11-2:
    kind: nokia_srsim
    type: SR-7
    network-mode: container:srsim11-a
    env:
      NOKIA_SROS_SLOT: 2
links:
  # Datapath links
  - endpoints: ["srsim10-1:e1-1-c1-1", "srsim11-1:e1-1-c1-1"]
  - endpoints: ["srsim10-2:e2-1-c1-1", "srsim11-2:e2-1-c2-1"]
```

Note: The standby CPM and all linecards in a SR-SIM distributed instance must have their interfaces in the same underlying network namespace (netns) to facilitate the inter-card connectivity. This is achieved using the `network-mode` statement and setting its value to the name of the active CPM container.

To remove the deployed network, use the following command.

```
clab destroy -t srsim.clab.yml
```

The displayed output is similar to the following:

```
22:30:11 INFO Parsing & checking topology file=srsim.clab.yml
22:30:11 INFO Parsing & checking topology file=srsim.clab.yml
22:30:11 INFO Destroying lab name=srsrim
```

```
22:30:11 INFO Removed container name=clab-srsrim-srsim10-a
22:30:13 INFO Removed container name=clab-srsrim-srsim11-b
22:30:13 INFO Removed container name=clab-srsrim-srsim10-b
22:30:13 INFO Removed container name=clab-srsrim-srsim10-2
22:30:13 INFO Removed container name=clab-srsrim-srsim11-2
22:30:13 INFO Removed container name=clab-srsrim-srsim11-1
22:30:13 INFO Removed container name=clab-srsrim-srsim10-1
22:30:13 INFO Removed container name=clab-srsrim-srsim11-a
22:30:13 INFO Removing host entries path=/etc/hosts
22:30:13 INFO Removing SSH config path=/etc/ssh/ssh_config.d/clab-srsrim.conf
```

To also remove the filesystems created by the router instances, use the `--cleanup` option.

10.2.4 Kubernetes

10.2.4.1 Kubernetes overview

Kubernetes is a container management platform that allows for the detailed management of large-scale compute clusters, with built-in scheduling and resiliency functionality. Kubernetes is designed to facilitate highly customizable application microservices that can scale vertically by increasing or decreasing memory and CPU resources dynamically, and horizontally by adding and removing container instances to meet the requirements of the overall application or service.

Kubernetes is a declarative system, meaning that the user instructs Kubernetes through the use of resource manifest files, specifying how the end system should look. The Kubernetes management system then determines how to reconcile the current state of the Kubernetes cluster to the desired (declared) state.

Deploying the SR-SIM on a Kubernetes cluster enables the creation of large-scale labs distributed over multiple compute nodes, optimizing the available infrastructure.

The following Kubernetes-specific terminology is used throughout this document:

- **node**: A physical compute machine (server).
- **cluster**: A Kubernetes cluster is a collection of *nodes* joined together to function as one pool of compute.
- **container**: The simplest building block within Kubernetes, referring to the actual software image running on the compute. A container creates an instance of the software from a given container registry.
- **registry**: A container registry is a storage environment that contains copies of the physical software in a library that can be queried by Kubernetes. When a container is started, it queries the registry for the software image and then downloads the file. Registries may be public or private.

  Note: Nokia recommends using a private container registry; the SR-SIM image should not be stored in a publicly accessible registry.

- **volume**: A volume represents an element of data that can be presented to a container as a file within its filesystem. A volume may come from a file, a directory, or from other Kubernetes elements, such as *ConfigMaps* or *Secrets*.
- **configmap**: A ConfigMap is a set of constants that can be used with other Kubernetes elements. ConfigMaps can be created statically, or generated from *Kustomization* or files.

- **secret**: A secret is similar to a ConfigMap, but the stored data is encoded (obfuscated). **Secrets** are often used to store passwords.
- **pod**: A pod is an important element in Kubernetes. It is a collection of containers and volumes connected together to form a deployable instance. One SR-SIM can be considered to be delivered in one pod. A pod may contain multiple containers and volumes.
- **deployment**: A deployment is a logical grouping that provides the intended deployment of a pod. A deployment enables an implementation to scale horizontally to create and remove pod instances as required. The example in this document shows a *Deployment* as a method to instantiate the SR-SIM.
- **service**: When a Kubernetes *pod* is deployed, it does not have any external connectivity or internal cluster connectivity. By default, a pod cannot communicate with another pod, although containers within a pod can communicate with each other. A service exposes specific pods to customers and external entities. A service may expose ports locally on the host machine (not recommended), to the cluster itself (called **ClusterIP**), or to external entities via a Kubernetes load balancer (called **LoadBalancerIP**).
- **cni**: A Container Network Interface (CNI) is a plugin that enables connectivity between containers and between nodes in a Kubernetes cluster. There are many CNI available for Kubernetes. The choice of a CNI is important in the context of the SR-SIM, as the SR-SIM requires each container to be provided with an IP address within a routable range and a default route to a reachable IP address. Some CNI do not support this capability.
- **network attachment definition**: A network attachment definition is a declarative way of describing network interfaces used to connect pods.
- **kustomize**: Kustomize provides a way to automatically generate specific Kubernetes elements upon deployment. It also provides a convenient way to link a number of Kubernetes resources (elements) together to deploy them all at once. The examples in this document use *Kustomize* to instantiate SR-SIM laboratory networks.
- **namespace**: Kubernetes namespaces keep Kubernetes resources (elements) separated, ensuring security and reducing the blast radius of any failing deployments.

10.2.4.2 Kubernetes distributions

There are many Kubernetes distributions available. Common distributions include (but are not limited to):

- CNCF Kubernetes
- Canonical Microk8s
- MiniKube
- k3s

Note: Each distribution requires deployment and configuration to suit the user's environment. Deployment and configuration details are beyond the scope of this document. Nokia does not specifically recommend any distribution. The infrastructure and versions used to test the SR-SIM are described earlier in this document. However, deployment is not limited to these versions. It is worth noting that Microk8s uses the Calico CNI, which by default does not allocate routable IP addresses or gateway addresses.

10.2.4.3 License file

The SR-SIM license file must be mounted in each SR-SIM container as the `/nokia/license/license.txt` file. Kubernetes provides several ways to achieve this, including Volumes, ConfigMaps, and Secrets.

Examples in this section use *Kustomize* to automatically generate *ConfigMaps* from the license file stored locally on disk. This approach eliminates the need to distribute the license file to hosts in the Kubernetes cluster.

10.2.4.4 SR OS configuration file

The SR OS configuration file may, optionally, be provided to a CPM container by mounting it as the `/nokia/config/config.cfg` file on the container filesystem. Kubernetes provides several ways to achieve this, including Volumes, ConfigMaps, and Secrets.

Examples in this section use *Kustomize* to automatically generate *ConfigMaps* from the license file stored locally on disk. This approach eliminates the need to distribute the license file to hosts in the Kubernetes cluster.

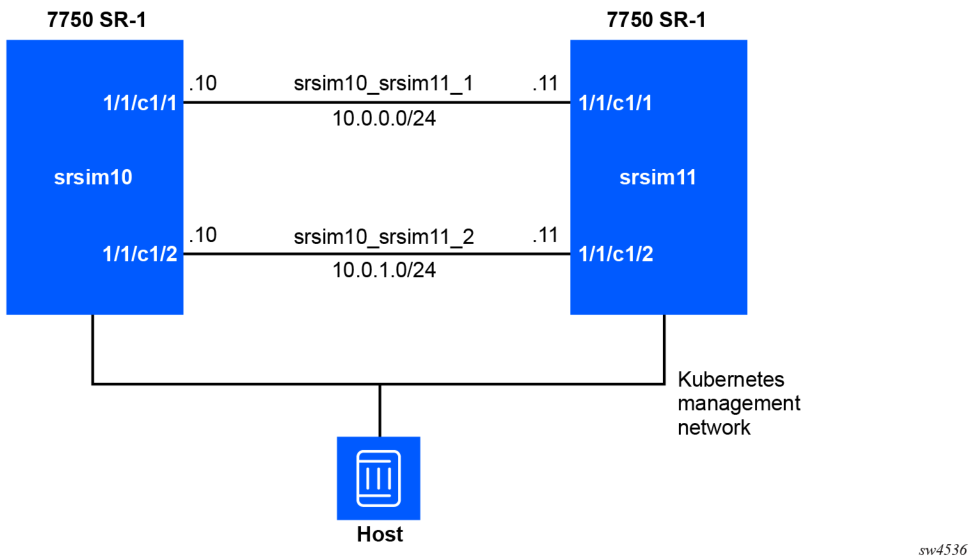
10.2.4.5 Kubernetes characteristics used in examples

The following examples are deployed on a k3s Kubernetes cluster running two Container Network Interfaces (CNI): Flannel and Multus. Flannel provides simple networking for the cluster and Multus provides inter-pod connectivity with `NetworkAttachmentDefinition` resources. The cluster is deployed using Ubuntu Linux version 24.04.

10.2.4.5.1 Integrated mode deployment

The following figure shows an example network comprising of two SR-SIM instances running in integrated mode. Each SR-SIM instance has a connection to the default Kubernetes management network and two connections to the other SR-SIM instance.

Figure 12: Example: Integrated mode network



To create this network, deploy the following resources to the Kubernetes cluster:

- A `Namespace` called `srsim` that will contain each of the Kubernetes resources, keeping them separate from any other workloads on the Kubernetes cluster
- A `NetworkAttachmentDefinition` for each of the `srsim10_srsim11_1` and `srsim10_srsim11_2` networks
- A `ConfigMap` containing the two SR OS configuration files `srsim10.cfg` and `srsim11.cfg`, along with the SR OS license file `license.txt`
- A `Deployment` for each SR-SIM instance (`srsim10` and `srsim11`)
- A `Service` for each SR-SIM instance exposing the SSH port on each router

Each of these resources must be defined in YAML files and created and deployed using *Kustomize* and the Kustomization manifest file `kustomization.yml`.

Create a directory to store the manifest files. Navigate to make this your working directory for the rest of this deployment.

Create the namespace manifest file for the `srsim` namespace.

srsim-namespace.yml:

```
kind: Namespace
apiVersion: v1
metadata:
  name: srsim
  labels:
    name: srsim
```

This resource creates the `srsim` namespace, which will contain all other resources in this example.

Create the `NetworkAttachmentDefinition` manifest files for the `srsim10_srsim11_1` and `srsim10_srsim11_2` networks.

srsim10_srsim11_1_nad.yml:

```
apiVersion: 'k8s.cni.cncf.io/v1'
kind: NetworkAttachmentDefinition
metadata:
  name: srsim10-srsim11-1
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "srsim10_srsim11_1",
    "type": "macvlan",
    "mode": "bridge",
    "mtu": 9000
  }'
```

srsim10_srsim11_2_nad.yml:

```
apiVersion: 'k8s.cni.cncf.io/v1'
kind: NetworkAttachmentDefinition
metadata:
  name: srsim10-srsim11-2
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "srsim10_srsim11_2",
    "type": "macvlan",
    "mode": "bridge",
    "mtu": 9000
  }'
```

These `NetworkAttachmentDefinition` resources are used by the Multus CNI plugin to interconnect the SR-SIM instances. The Multus CNI plugin is used because it allows the operator to attach multiple network interfaces to a single container or pod.

In the preceding `NetworkAttachmentDefinition`, each network is created as a MACVLAN network in bridge mode. The MTU on each link is set at creation.

Create the `Deployment` manifest for the first SR-SIM instance, `srsim10`.

srsim10-deployment.yml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: srsim10
spec:
  selector:
    matchLabels:
      app: srsim10
  template:
    metadata:
      annotations:
        k8s.v1.cni.cncf.io/networks: srsim10-srsim11-1@e1-1-c1-1, srsim10-srsim11-2@e1-1-c1-2
      labels:
        app: srsim10
    spec:
      volumes:
        - name: files-vol
          configMap:
            name: files-configmap
            items:
              - key: license.txt
```

```
                path: license.txt
              - key: srsim10.cfg
                path: srsim10.cfg
      containers:
        - name: srsim10
          image: localhost:32000/srsim:25.10.R1
          volumeMounts:
            - name: files-vol
              mountPath: /nokia/license/license.txt
              subPath: license.txt
            - name: files-vol
              mountPath: /nokia/config/config.cfg
              subPath: srsim10.cfg
          securityContext:
            privileged: true
```

Note: This is a critical manifest file. It defines the Pod that will be created and specifies any infrastructure requirements and constraints for its creation.

The component parts of this `Deployment` are as follows:

- `name: srsim10`: This line sets the name of the deployment. The names of the Pods created as a result will be prefixed with this value.
- `k8s.v1.cni.cncf.io/networks: srsim10-srsim11-1@e1-1-c1-1, srsim10-srsim11-2@e1-1-c1-2`: This line creates the datapath interfaces inside the container. There are two interfaces created between `srsim10` and `srsim11`, as shown in the figure earlier in this section. Each interface created here is defined as `<NetworkAttachmentDefinition name>@<Interface name inside the container>`. The `<NetworkAttachmentDefinition name>` refers to the definition in the earlier manifests. One interface is added on the first bridge and the other on the second. The `<Interface name inside the container>` defines the name of the interface that will be created inside the container. This must use the matching format to align with the SR OS port names. For example, `e1-1-c1-1` will be connected to port `1/1/c1/1` in SR OS.
- The `volumes` section creates a `files-vol` volume, which uses a `ConfigMap` called `files-configmap`. The `key` and `path` fields within this `ConfigMap` reference specific portions of the `ConfigMap`. This information will be described in detail in the `kustomize.yml` file definition. This section defines files as being available for mounting, but does not mount any files into containers.
- The `containers` section defines the containers that will make up this Pod. In this example, only one container is defined, named `srsim10`.
- The `image: localhost:32000/srsim:25.10.R1` defines the location of the SR-SIM image and its tag in the registry. Similar to other examples in this document, the SR-SIM image is tagged as `25.10.R1` in the `localhost:32000/srsim` registry.
- The `volumeMounts` section uses the previously defined `volume` called `files-vol` and mounts the chosen data from the volume as specific files in the container filesystem. In this example, the `subPath: license.txt` references the `license.txt` key in the volume and mounts its contents as the `/nokia/license/license.txt` file. A second `volumeMounts` entry uses the same `volume`, but this time mounts the `srsim10.cfg` key as the `/nokia/config/config.cfg` file inside the container.
- The `privileged: true` option grants the container extended privileges on the host machine, allowing it to manipulate network interfaces.

Create the `Deployment` manifest for the second SR-SIM instance, `srsim11`.

srsim11-deployment.yml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: srsim11
spec:
  selector:
    matchLabels:
      app: srsim11
  template:
    metadata:
      annotations:
        k8s.v1.cni.cncf.io/networks: srsim10-srsim11-1@e1-1-c1-1, srsim10-srsim11-2@e1-1-c1-2
      labels:
        app: srsim11
    spec:
      volumes:
        - name: files-vol
          configMap:
            name: files-configmap
            items:
              - key: license.txt
                path: license.txt
              - key: srsim11.cfg
                path: srsim11.cfg
      containers:
        - name: srsim11
          image: localhost:32000/srsim:25.10.R1
          volumeMounts:
            - name: files-vol
              mountPath: /nokia/license/license.txt
              subPath: license.txt
            - name: files-vol
              mountPath: /nokia/config/config.cfg
              subPath: srsim11.cfg
          securityContext:
            privileged: true
```

Create the `Service` manifests for both SR-SIM instances (`srsim10` and `srsim11`). These manifests will expose the SSH port of each SR-SIM instance so they are accessible.

srsim10-service.yml:

```
apiVersion: v1
kind: Service
metadata:
  name: srsim10-service
spec:
  type: ClusterIP
  selector:
    app: srsim10
  ports:
  - port: 22
    targetPort: 22
    protocol: TCP
```

srsim11-service.yml:

```
apiVersion: v1
kind: Service
metadata:
```

```
  name: srsim11-service
spec:
  type: ClusterIP
  selector:
    app: srsim11
  ports:
  - port: 22
    targetPort: 22
    protocol: TCP
```

You should have the following **nine** files in your directory now:

- `srsim-namespace.yml`
- `srsim10_srsim11_1_nad.yml`
- `srsim10_srsim11_2_nad.yml`
- `srsim10-deployment.yml`
- `srsim11-deployment.yml`
- `srsim10-service.yml`
- `srsim11-service.yml`
- `srsim10.cfg` (This is a valid SR OS configuration file. You may omit this and comment out the appropriate `VolumeMount` if the default config is desired.)
- `srsim11.cfg` (This is a valid SR OS configuration file. You may omit this and comment out the appropriate `VolumeMount` if the default config is desired.)
- `license.txt` (This is a valid SR-SIM license file. It is required to boot the SR-SIM.)

The component parts are now ready for deployment, but they need to be tied together. Use **Kustomize** to achieve this. Create the following file.

kustomization.yml:

```
namespace: srsim
resources:
  - srsim-namespace.yml
  - srsim10-deployment.yml
  - srsim10-service.yml
  - srsim11-deployment.yml
  - srsim11-service.yml
  - srsim10_srsim11_1_nad.yml
  - srsim10_srsim11_2_nad.yml
configMapGenerator:
  - name: files-configmap
    files:
      - license.txt
      - srsim10.cfg
      - srsim11.cfg
```

This file does more that just reference other files (`resources`), it also dynamically creates other resource manifests that are used automatically. In this example, the file dynamically creates the `ConfigMap` called `files-configmap`, which is referenced later in the deployment manifests. It dynamically creates keys in the `ConfigMap` with the same name as the filename, and then includes the file data into the `ConfigMap`.

The SR-SIM network is now ready to be deployed. Use the `kubectl` command to deploy the network.

```
kubectl apply -k .
```

The `apply` keyword makes Kubernetes deploy the resources. The `-k` tells `kubectl` to use *Kustomize*. The output will look similar to the following:

```
namespace/srsim created
configmap/files-configmap-tcc8f7tg55 created
service/srsim10-service created
service/srsim11-service created
deployment.apps/srsim10 created
deployment.apps/srsim11 created
networkattachmentdefinition.k8s.cni.cncf.io/srsim10-srsim11-1 created
networkattachmentdefinition.k8s.cni.cncf.io/srsim10-srsim11-2 created
```

The `kubectl` command can be used to check that the SR-SIM instances are deployed correctly. Use the following command to show all `Deployment` and `Pod` resources in the `srsim` namespace.

```
kubectl get -n srsim all
```

The output will look similar to this:

```
NAME                           READY   STATUS    RESTARTS   AGE
pod/srsim10-67c8cccfdd-rv47t   1/1     Running   0          110s
pod/srsim11-7bdb4cb6fc-cg7xh   1/1     Running   0          110s

NAME                      TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/srsim10-service   ClusterIP   10.43.229.81   <none>        22/TCP    111s
service/srsim11-service   ClusterIP   10.43.168.67   <none>        22/TCP    111s

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/srsim10   1/1     1            1           111s
deployment.apps/srsim11   1/1     1            1           110s

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/srsim10-67c8cccfdd   1         1         1       111s
replicaset.apps/srsim11-7bdb4cb6fc   1         1         1       110s
```

If the deployments and pods show `1/1` deployed, the SR-SIM instances are booted correctly.

If they are not started correctly, *describing* the `Pod` will provide useful information. *Describing* the deployed pods provides a detail list of events involved when each one is created:

```
kubectl describe -n srsim pod/srsim10-67c8cccfdd-rv47t
```

The output will look similar to the following.

```
Name:             srsim10-67c8cccfdd-rv47t
Namespace:        srsim
Priority:         0
Service Account:  default
Node:             myserver/192.168.184.53
Start Time:       Fri, 20 Jun 2025 16:12:42 +0000
Labels:           app=srsim10
                  pod-template-hash=67c8cccfdd
Annotations:      k8s.v1.cni.cncf.io/network-status:
                    [{
                        "name": "cbr0",
```

```
                         "interface": "eth0",
                         "ips": [
                             "10.42.1.62"
                         ],
                         "mac": "2a:58:b9:fd:d6:22",
                         "default": true,
                         "dns": {},
                         "gateway": [
                             "10.42.1.1"
                         ]
                     },{
                         "name": "srsim/srsim10-srsim11-1",
                         "interface": "e1-1-c1-1",
                         "mac": "46:40:86:46:3d:a2",
                         "dns": {}
                     },{
                         "name": "srsim/srsim10-srsim11-2",
                         "interface": "e1-1-c1-2",
                         "mac": "fe:91:a9:4e:6c:87",
                         "dns": {}
                     }]
                   k8s.v1.cni.cncf.io/networks: srsim10-srsim11-1@e1-1-c1-1, srsim10-srsim11-
2@e1-1-c1-2
Status:           Running
IP:               10.42.1.62
IPs:
  IP:           10.42.1.62
Controlled By:  ReplicaSet/srsim10-67c8cccfdd
Containers:
  srsim10:
    Container ID:   containerd://
9c12adfaa86c6ec27e770e8496ab8e5dbd3b962decf423d72f80c1adc3322d85
    Image:          localhost:32000/srsim:25.10.R1
    Image ID:       localhost:32000/
srsim@sha256:4d43f27f833940a9e53538138604af30cfde74d99b538714df067992efbf6986
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Fri, 20 Jun 2025 16:12:44 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /nokia/config/config.cfg from files-vol (rw,path="srsim10.cfg")
      /nokia/license/license.txt from files-vol (rw,path="license.txt")
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-v6t5n (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  files-vol:
    Type:      ConfigMap (a volume populated by a ConfigMap)
    Name:      files-configmap-tcc8f7tg55
    Optional:  false
  kube-api-access-v6t5n:
    Type:                    Projected (a volume that contains injected data from multiple
 sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
```

```
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason          Age        From               Message
  ----    ------          ----       ----               -------
  Normal  Scheduled       2m12s      default-scheduler  Successfully assigned srsim/srsim10-
67c8cccfdd-rv47t to myserver
  Normal  AddedInterface  <invalid>  multus             Add eth0 [10.42.1.62/24] from cbr0
  Normal  AddedInterface  <invalid>  multus             Add e1-1-c1-1 [] from srsim/srsim10-
srsim11-1
  Normal  AddedInterface  <invalid>  multus             Add e1-1-c1-2 [] from srsim/srsim10-
srsim11-2
  Normal  Pulled          <invalid>  kubelet            Container image "localhost:32000/
srsim:25.10.R1" already present on machine
  Normal  Created         <invalid>  kubelet            Created container: srsim10
  Normal  Started         <invalid>  kubelet            Started container srsim10
```

The last section is the most useful, providing a concise, chronological list of events, including errors.

The console logs from an SR-SIM instance can also be displayed using the `kubectl` command.

For a one-off view of the logs, use the following command:

```
kubectl logs -n srsim srsim10-67c8cccfdd-rv47t
```

To continuously stream logs to the screen, use the following command:

```
kubectl logs -n srsim -f srsim10-67c8cccfdd-rv47t
```

To connect to the SR-SIM instances, identify the management IP address of each instance. This was allocated by the `Service` resource when it was created. Use the `kubectl` command to obtain this information:

```
kubectl get -n srsim service
```

The output should look similar to the following:

```
NAME              TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
srsim10-service   ClusterIP   10.43.229.81   <none>        22/TCP    24m
srsim11-service   ClusterIP   10.43.168.67   <none>        22/TCP    24m
```

The management IP addresses for each router are shown here under the "CLUSTER-IP" column. Use these IP addresses to SSH into the devices.

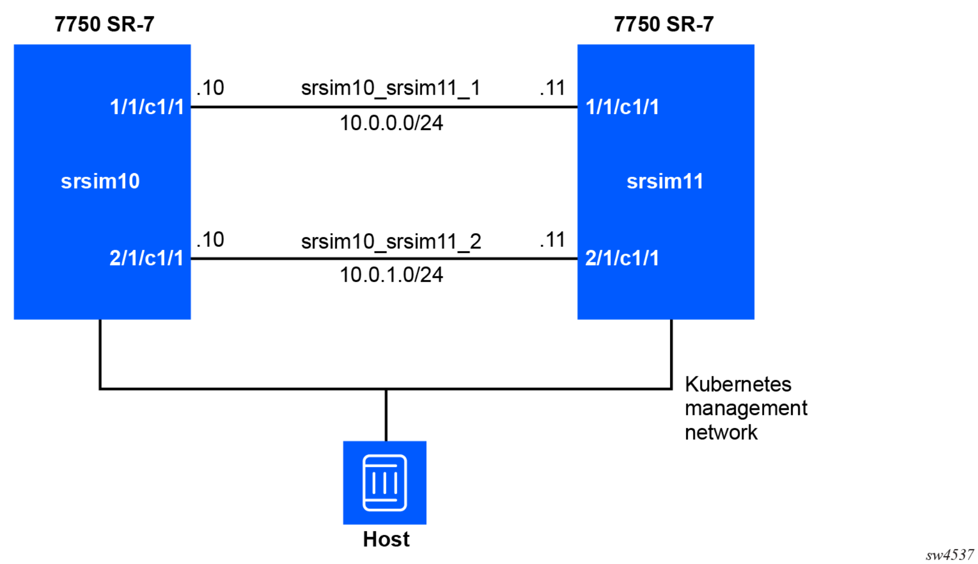
10.2.4.5.2 Distributed mode deployment

See the Integrated mode deployment section before progressing to this section as some of the concepts will be reused and extended here.

To deploy a distributed mode SR-SIM in Kubernetes we will deploy each line card as a separate *container*, however, all linecards for a given router will be deployed within a single *pod*. Using this method of deployment the SR-SIM can manage the inter-card fabric interface itself and the active and standby CPM can be handled by the same Kubernetes *service* for access.

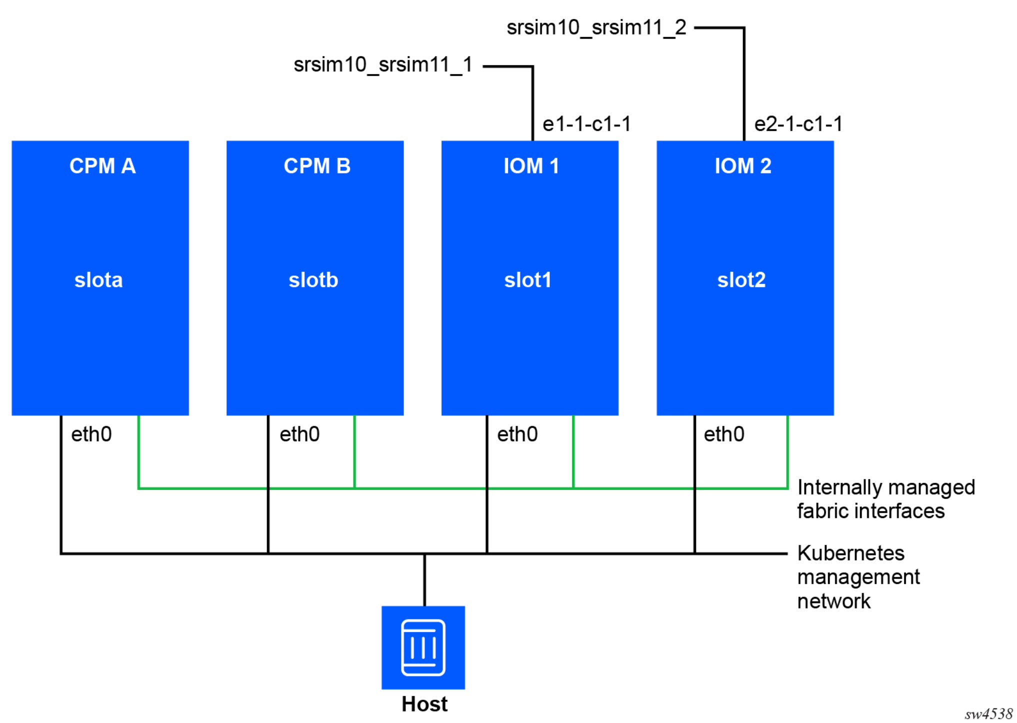
This is the example network that will be deployed:

Figure 13: Example: Distributed mode network deployment



Each router consists of a number of containers. The following figure shows an example `srsim10` router container layout, where each box represents a separate container.

Figure 14: Example: Router container layout



By using the *Kustomize* feature of Kubernetes this network can be deployed using a hierarchical template with minimal customization between routers.

The directory layout to create should look like this:

```
.
+-- kustomization.yml
+-- license.txt
+-- srsim10-srsim11-1-nad.yml
+-- srsim10-srsim11-2-nad.yml
+-- srsim-namespace.yml
+-- srsim
|   +-- srsim-deployment.yml
|   +-- srsim-service.yml
+-- srsim10
|   +-- kustomization.yml
|   +-- srsim10.cfg
+-- srsim11
    +-- kustomization.yml
    +-- srsim11.cfg
```

Create each file in turn using the following information.

./kustomization.yml:

```
namespace: srsim
resources:
  - srsim-namespace.yml
  - srsim10-srsim11-1-nad.yml
  - srsim10-srsim11-2-nad.yml
  - srsim10/
  - srsim11/
configMapGenerator:
  - name: license
    files:
      - license.txt
```

This `kustomization.yml` file imports `kustomization.yml` files from other directories (`./srsim10/kustomization.yml` and `./srsim11/kustomization.yml`) by providing the directory name in the `resources` section.

As explained in Integrated mode deployment, this `kustomization.yml` file creates a `ConfigMap` named `license` from the `license.txt` file. This `ConfigMap` can then be used inside each `Deployment` manifest.

./license.txt: This is the SR-SIM license file associated with your subscription.

./srsim10-srsim11-1-nad.yml:

```
apiVersion: 'k8s.cni.cncf.io/v1'
kind: NetworkAttachmentDefinition
metadata:
  name: srsim10-srsim11-1
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "srsim10_srsim11_1",
    "type": "macvlan",
    "mode": "bridge",
    "mtu": 9000
  }'
```

./srsim10-srsim11-2-nad.yml:

```
apiVersion: 'k8s.cni.cncf.io/v1'
kind: NetworkAttachmentDefinition
metadata:
  name: srsim10-srsim11-2
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "srsim10_srsim11_2",
    "type": "macvlan",
    "mode": "bridge",
    "mtu": 9000
  }'
```

srsim-namespace.yml:

```
kind: Namespace
apiVersion: v1
metadata:
  name: srsim
  labels:
    name: srsim
```

./srsim/kustomization.yml:

```
resources:
  - srsim-deployment.yml
  - srsim-service.yml
```

./srsim/srsim-deployment.yml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: srsim
spec:
  selector:
    matchLabels:
      app: srsim
  template:
    metadata:
      annotations:
        k8s.v1.cni.cncf.io/networks: srsim10-srsim11-1@e1-1-c1-1, srsim10-srsim11-2@e1-1-c1-2
      labels:
        app: srsim
    spec:
      volumes:
        - name: config-vol
          configMap:
            name: config
            items:
              - key: config
                path: config
        - name: license-vol
          configMap:
            name: license
            items:
              - key: license.txt
                path: license.txt
      containers:
```

```
        - name: slota
          image: localhost:32000/srsim:25.10.R1
          volumeMounts:
            - name: license-vol
              mountPath: /nokia/license/license.txt
              subPath: license.txt
            - name: config-vol
              mountPath: /nokia/config/config.cfg
              subPath: config
          securityContext:
            privileged: true
          envFrom:
            - configMapRef:
                name: router
          env:
            - name: NOKIA_SROS_SLOT
              value: "a"
        - name: slotb
          image: localhost:32000/srsim:25.10.R1
          volumeMounts:
            - name: license-vol
              mountPath: /nokia/license/license.txt
              subPath: license.txt
            - name: config-vol
              mountPath: /nokia/config/config.cfg
              subPath: config
          securityContext:
            privileged: true
          envFrom:
            - configMapRef:
                name: router
          env:
            - name: NOKIA_SROS_SLOT
              value: "b"
        - name: slot1
          image: localhost:32000/srsim:25.10.R1
          volumeMounts:
            - name: license-vol
              mountPath: /nokia/license/license.txt
              subPath: license.txt
          securityContext:
            privileged: true
          envFrom:
            - configMapRef:
                name: router
          env:
            - name: NOKIA_SROS_SLOT
              value: "1"
        - name: slot2
          image: localhost:32000/srsim:25.10.R1
          volumeMounts:
            - name: license-vol
              mountPath: /nokia/license/license.txt
              subPath: license.txt
          securityContext:
            privileged: true
          envFrom:
            - configMapRef:
                name: router
          env:
            - name: NOKIA_SROS_SLOT
              value: "2"
```

The majority of the distributed deployment setup is achieved in this manifest resource file. It is similar to that used in the integrated SR-SIM deployment discussed in Integrated mode deployment; however, this resource manifest creates four containers per *Pod*.

The containers in this *Deployment* manifest are named `slota`, `slotb`, `slot1` and `slot2` to align with the SR OS numbering for CPM A, CPM B, IOM 1 and IOM 2.

Each container requires a valid license file. The `volumes` section defines a `license-vol` that is created using the `license.txt` key from the `license` *ConfigMap*.

Each container needs to have the chassis set to the correct type. In this the router is a 7750 SR-7 and therefore the `router` *configmap* includes the environment variable `NOKIA_SROS_CHASSIS=SR-7`.

The CPM containers must each have the chassis MAC address set. This must be set to the same value in both CPM containers. The `router` *configmap* includes the environment variable `NOKIA_SROS_SYSTEM_BASE_MAC` to achieve this.

Each container must known which slot it is being deployed in. This is achieved on a per-container basis by setting the environment variable `NOKIA_SROS_SLOT`. This is set on each container rather than as part of a *ConfigMap* as it will not change between deployments and is different on each container so cannot be reused.

./srsim/srsim-service.yml:

```
apiVersion: v1
kind: Service
metadata:
  name: srsim
spec:
  type: ClusterIP
  selector:
    app: srsim
  ports:
  - port: 22
    targetPort: 22
    protocol: TCP
```

./srsim10/kustomization.yml:

```
resources:
  - ../srsim
nameSuffix: "10"
labels:
  - pairs:
      app: srsim10
    includeSelectors: true
configMapGenerator:
  - name: config
    files:
      - config=srsim10.cfg
  - name: router
    literals:
      - NOKIA_SROS_CHASSIS=SR-7
      - NOKIA_SROS_SYSTEM_BASE_MAC=de:ff:ab:c9:bb:aa
```

This `kustomization.yml` file imports the `kustomization.yml` file from the `../srsim` directory, which in turn imports the deployment manifest `srsim-deployment.yml` and the service manifest `srsim-service.yml`.

The `nameSuffix: "10"` is important. This will suffix the *Service*, *Deployment* and *Pod* resources with the number 10 which is used to differentiate between the two routers. The `labels` section also updates the labels applied to each deployment and pod in a similar way so that the service knows which element to link to.

./srsim10/srsim10.cfg: A valid SR OS configuration file for the `srsim10` router.

./srsim11/kustomization.yml:

```
resources:
  - ../srsim
nameSuffix: "11"
labels:
  - pairs:
      app: srsim11
    includeSelectors: true
configMapGenerator:
  - name: config
    files:
      - config=srsim11.cfg
  - name: router
    literals:
      - NOKIA_SROS_CHASSIS=SR-7
      - NOKIA_SROS_SYSTEM_BASE_MAC=de:ff:ab:c9:bb:bb
```

This `kustomization.yml` file imports the `kustomization.yml` file from the `../srsim` directory, which in turn imports the deployment manifest `srsim-deployment.yml` and the service manifest `srsim-service.yml`.

The `nameSuffix: "11"` is important. This will suffix the *Service*, *Deployment* and *Pod* resources with the number 11 which is used to differentiate between the two routers. The `labels` section also updates the labels applied to each deployment and pod in a similar way so that the service knows which element to link to.

./srsim11/srsim11.cfg: A valid SR OS configuration file for the `srsim11` router.

To deploy the network use the `kubectl` command:

```
kubectl apply -k .
```

The output will look similar to this:

```
namespace/srsim created
configmap/config10-d52dfh957b created
configmap/config11-m8kb4kfh4c created
configmap/license-55hfghh822 created
configmap/router10-h5tchkdcb5 created
configmap/router11-bkg7tg27ft created
service/srsim10 created
service/srsim11 created
deployment.apps/srsim10 created
deployment.apps/srsim11 created
networkattachmentdefinition.k8s.cni.cncf.io/srsim10-srsim11-1 created
networkattachmentdefinition.k8s.cni.cncf.io/srsim10-srsim11-2 created
```

Although the `Deployment`, `Service` and `ConfigMap` manifests are named `srsim` the created instances include the `nameSuffix` from each devices `kustomization.yml` file. The Kubernetes system knows how to link these dynamic names together so the operator does not need to manage this.

The deployed items are created in the `srsim` namespace and can be viewed using the `kubectl` command:

```
kubectl get -n srsim all
```

The output will look similar to this:

```
NAME                           READY   STATUS    RESTARTS   AGE
pod/srsim10-75b96d94cd-qjmg8   4/4     Running   0          3m22s
pod/srsim11-75bdd76d8b-vrfz6   4/4     Running   0          3m22s

NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/srsim10   ClusterIP   10.43.210.100   <none>        22/TCP    3m22s
service/srsim11   ClusterIP   10.43.121.107   <none>        22/TCP    3m22s

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/srsim10   1/1     1            1           3m22s
deployment.apps/srsim11   1/1     1            1           3m22s

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/srsim10-75b96d94cd   1         1         1       3m22s
replicaset.apps/srsim11-75bdd76d8b   1         1         1       3m22s
```

The *ConfigMaps* created may be viewed with this command:

```
kubectl get -n srsim configmap
```

The output will look similar to this:

```
NAME                    DATA   AGE
config10-d52dfh957b     1      4m27s
config11-m8kb4kfh4c     1      4m27s
kube-root-ca.crt        1      4m27s
license-55hfghh822      1      4m27s
router10-h5tchkdcb5     2      4m27s
router11-bkg7tg27ft     2      4m27s
```

To identify the IP address of the deployed routers use the `kubectl` command to query to deployed *Services*:

```
kubectl get -n srsim service
```

The output will show the IP addresses that can be used to SSH to the devices. The output will be similar to this:

```
NAME      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
srsim10   ClusterIP   10.43.210.100   <none>        22/TCP    6m59s
srsim11   ClusterIP   10.43.121.107   <none>        22/TCP    6m59s
```

11 Appendixes

- Appendix A: SR-SIM supported hardware
- Appendix B: Known limitations
- Appendix C: Example Kubernetes configuration using k3s on Ubuntu 24.04

11.1 Appendix A: SR-SIM supported hardware

This appendix provides tables that list supported hardware for the following chassis types:

- 7250 IXR
- 7450 ESS
- 7750 SR
- 7950 XRS

11.1.1 7250 IXR

The following tables list the default system layout and supported hardware for the 7250 IXR chassis type.

11.1.1.1 7250 IXR-6

Table 4: 7250 IXR-6 default system layout

7250 IXR-6 default system layout					
Chassis	**Slot**	**Recommended memory per container**	**SFM**	**Card**	**MDA**
IXR-6	A	6 GB	sfm-ixr-6	cpm-ixr	—
	1			imm36-100g-qsfp28	—

Table 5: 7250 IXR-6 supported hardware

7250 IXR-6 supported hardware			
Chassis	**SFM**	**Card**	**MDA**
IXR-6	sfm-ixr-6	cpm-ixr	—

7250 IXR-6 supported hardware			
Chassis	**SFM**	**Card**	**MDA**
		imm36-100g-qsfp28	m36-100g-qsfp28
		imm48-sfp+2-qsfp28	m48-sfp+2-qsfp28

11.1.1.2 7250 IXR-10

Table 6: 7250 IXR-10 default system layout

7250 IXR-10 default system layout					
Chassis	**Slot**	**Recommended memory per container**	**SFM**	**Card**	**MDA**
IXR-10	A	4 GB	sfm-ixr-10	cpm-ixr	—
	1	6 GB		imm36-100g-qsfp28	—

Table 7: 7250 IXR-10 supported hardware

7250 IXR-10 supported hardware			
Chassis	**SFM**	**Card**	**MDA**
IXR-10	sfm-ixr-10	cpm-ixr	—
		imm36-100g-qsfp28	m36-100g-qsfp28
		imm48-sfp+2-qsfp28	m48-sfp+2-qsfp28

11.1.1.3 7250 IXR-e

Table 8: 7250 IXR-e default system layout

7250 IXR-e default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
IXR-e	A	4 GB	cpm-ixr-e	—
	1		imm24-sfp++8-sfp28+2-qsfp28	—

Table 9: 7250 IXR-e supported hardware

7250 IXR-e supported hardware		
Chassis	**Card**	**MDA**
IXR-e	cpm-ixr-e/imm24-sfp++8-sfp28+2-qsfp28 cpm-ixr-e-gnss/imm24-sfp++8-sfp28+2-qsfp28	m24-sfp++8-sfp28+2-qsfp28 [2]
	cpm-ixr-e/imm14-10g-sfp++4-1g-tx cpm-ixr-e-gnss/imm14-10g-sfp++4-1g-tx	m14-10g-sfp++4-1g-tx [2]

11.1.1.4 7250 IXR-e2

Table 10: 7250 IXR-e2 default system layout

7250 IXR-e2 default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
IXR-e2	A	4 GB	cpm-ixr-e2	—

Table 11: 7250 IXR-e2 supported hardware

7250 IXR-e2 supported hardware		
Chassis	**Card**	**MDA**
IXR-e2	cpm-ixr-e2	m2-qsfpdd+2-qsfp28+24-sfp28

11.1.1.5 7250 IXR-e2c

Table 12: 7250 IXR-e2c default system layout

7250 IXR-e2c default system layout				
Chassis		**Recommended memory per container**	**Card**	**MDA**
IXR-e2c	A	4 GB	cpm-ixr-e2c	—

2 The MDA entry is not required on the CPM VM in slot A.

Table 13: 7250 IXR-e2c supported hardware

7250 IXR-e2c supported hardware		
Chassis	**Card**	**MDA**
IXR-e2c	cpm-ixr-e2c	m12-sfp28+2-qsfp28

11.1.1.6 7250 IXR-ec

Table 14: 7250 IXR-ec IXR-ec default system layout

7250 IXR-ec default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
IXR-ec	A	4 GB	cpm-ixr-ec	—

Table 15: 7250 IXR-ec supported hardware

7250 IXR-ec supported hardware		
Chassis	**Card**	**MDA**
IXR-ec	cpm-ixr-ec	m4-1g-tx+20-1g-sfp+6-10g-sfp+

11.1.1.7 7250 IXR-R4

Table 16: 7250 IXR-R4 default system layout

7250 IXR-R4 default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
IXR-R4	A	4 GB	cpm-ixr-r4	—
	1		iom-ixr-r4	m6-10g-sfp++1-100g-qsfp28

Table 17: 7250 IXR-R4 supported hardware

7250 IXR-R4 supported hardware		
Chassis	**Card**	**MDA**
IXR-R4	cpm-ixr-r4	—
	iom-ixr-r4	m6-10g-sfp++1-100g-qsfp28
		m20-1g-csfp [3]
		m10-10g-sfp+
		m4-10g-sfp++1-100g-cfp2
		m6-10g-sfp++4-25g-sfp28
		m10-1g-sfp+2-10g-sfp+ [4]

11.1.1.8 7250 IXR-R6

Table 18: 7250 IXR-R6 default system layout

7250 IXR-R6 default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
IXR-R6	A	6 GB	cpiom-ixr-r6	m6-10g-sfp++1-100g-qsfp28

Table 19: 7250 IXR-R6 supported hardware

7250 IXR-R6 supported hardware		
Chassis	**Card**	**MDA**
IXR-R6	cpiom-ixr-r6	a32-chds1v2 [5]
		m4-10g-sfp++1-100g-cfp2
		m6-10g-sfp++1-100g-qsfp28
		m6-10g-sfp++4-25g-sfp28

3 This MDA must use slots 1, 2, or 3.

4 The integrated MDA must be specified as mda/5.

5 This MDA has slot restrictions for slot 5 or 6.

7250 IXR-R6 supported hardware		
Chassis	**Card**	**MDA**
		m10-10g-sfp+
		m20-1g-csfp [6]

11.1.1.9 7250 IXR-R6d

Table 20: 7250 IXR-R6d default system layout

7250 IXR-R6d default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
IXR-R6d	A	4 GB	cpm-ixr-r6d	—
	1		iom-ixr-r6d	m1-400g-qsfpdd+1-100g-qsfp28

Table 21: 7250 IXR-R6d supported hardware

7250 IXR-R6d supported hardware		
Chassis	**Card**	**MDA**
IXR-R6d	cpm-ixr-r6d/iom-ixr-r6d	m1-400g-qsfpdd+1-100g-qsfp28 [2] m5-100g-qsfp28 [2] m18-25g-sfp28[2] m2-cfp2[2] m20-10g-sfp+[2] m10-50g-sfp56[2] m32-1g-csfp[2] m2-100g-qsfp28+16-10g-sfp+[2]

6 This MDA must use either slot 3 or 4.

11.1.1.10 7250 IXR-R6dl

Table 22: 7250 IXR-R6dl default system layout

7250 IXR-R6dl default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
IXR-R6dl	A	4 GB	cpm-ixr-r6d	—
	1		iom-ixr-r6d	m1-400g-qsfpdd+1-100g-qsfp28

Table 23: 7250 IXR-R6dl supported hardware

7250 IXR-R6dl supported hardware		
Chassis	**Card**	**MDA**
IXR-R6dl	cpm-ixr-r6d/iom-ixr-r6d	m1-400g-qsfpdd+1-100g-qsfp28 [2] m5-100g-qsfp28 [2] m18-25g-sfp28[2] m2-cfp2[2] m20-10g-sfp+[2] m10-50g-sfp56[2] m46-10g-sfp+[2] m32-1g-csfp[2] m80-1g-csfp[2] m2-100g-qsfp28+16-10g-sfp+[2]

11.1.1.11 7250 IXR-s

Table 24: 7250 IXR-s default system layout

7250 IXR-s default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
IXR-s	A	4 GB	cpm-ixr-s/imm48-sfp++6-qsfp28	—
	1		cpm-ixr-s/imm48-sfp++6-qsfp28	—

Table 25: 7250 IXR-s supported hardware

7250 IXR-s supported hardware		
Chassis	**Card**	**MDA**
IXR-s	cpm-ixr-s/imm48-sfp++6-qsfp28	m48-sfp++6-qsfp28 [2]

11.1.1.12 7250 IXR-X

Table 26: 7250 IXR-X default system layout

7250 IXR-X default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
IXR-X	A	4 GB	cpm-ixr-x	—
	1		imm6-qsfpdd+48-sfp56	—

Table 27: 7250 IXR-X supported hardware

7250 IXR-X supported hardware		
Chassis	**Card**	**MDA**
IXR-X	cpm-ixr-x/imm32-qsfp28+4-qsfpdd	m32-qsfp28+4-qsfpdd [2]
	cpm-ixr-x/imm6-qsfpdd+48-sfp56	m6-qsfpdd+48-sfp56 [2]

11.1.1.13 7250 IXR-X3

Table 28: 7250 IXR-X3 default system layout

7250 IXR-X3 default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
IXR-X3	A	6 GB	cpm-ixr-x	—
	1		imm36-qsfpdd	—

Table 29: 7250 IXR-X3 supported hardware

7250 IXR-X3 supported hardware		
Chassis	**Card**	**MDA**
IXR-X3	cpm-ixr-x/imm36-qsfpdd	m36-qsfpdd [2]

11.1.2 7450 ESS

The following tables list the default system layout and supported hardware for the 7450 ESS chassis type.

11.1.2.1 7450 ESS-7

Table 30: 7450 ESS-7 default system layout

7450 ESS-7 default system layout					
Chassis	**Slot**	**Recommended memory per container**	**SFM**	**Card**	**MDA**
ESS-7	A	4 GB	m-sfm6-7/12	cpm5	—
	1			iom5-e	me6-100gb-qsfp28

Table 31: 7450 ESS-7 supported hardware

7450 ESS-7 supported hardware			
Chassis	**SFM**	**Card**	**MDA**
ESS-7	m-sfm6-7/12	cpm5	—

7450 ESS-7 supported hardware			
Chassis	SFM	Card	MDA
		iom5-e	me6-100gb-qsfp28

11.1.2.2 7450 ESS-12

Table 32: 7450 ESS-12 default system layout

7450 ESS-12 default system layout					
Chassis	Slot	Recommended memory per container	SFM	Card	MDA
ESS-12	A	4 GB	m-sfm6-7/12	cpm5	—
	1			iom5-e	me6-100gb-qsfp28

Table 33: 7450 ESS-12 supported hardware

7450 ESS-12 supported hardware			
Chassis	SFM	Card	MDA
ESS-12	m-sfm6-7/12	cpm5	—
		iom5-e	me6-100gb-qsfp28

11.1.3 7750 SR

The following tables list the default system layout and supported hardware for the 7750 SR chassis type.

11.1.3.1 7750 SR-1

Table 34: 7750 SR-1 default system layout

7750 SR-1 default system layout					
Chassis	Slot	Recommended memory per container	Card	MDA/1	MDA/ 2
SR-1	A	4 GB	cpm-1	me6-100gb-qsfp28	me12-100gb-qsfp28

Table 35: 7750 SR-1 supported hardware

7750 SR-1 supported hardware		
Chassis	**Card**	**MDA**
SR-1	cpm-1	me12-100gb-qsfp28
		me3-200gb-cfp2-dco
		me6-100gb-qsfp28
		me6-400gb-qsfpdd
		me16-25gb-sfp28+2-100gb-qsfp28
		me3-400gb-qsfpdd
		me16-25gb-sfp28+2-100gb-qsfp-b

11.1.3.2 7750 SR-1-24D

Table 36: 7750 SR-1-24D default system layout

7750 SR-1-24D default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
SR-1-24D	A	4 GB	cpm-1x/i24-800g-qsfpdd-1	—
	1		cpm-1x/i24-800g-qsfpdd-1	—

Table 37: 7750 SR-1-24D supported hardware

7750 SR-1-24D supported hardware		
Chassis	**Card**	**MDA**
SR-1-24D	cpm-1x/i24-800g-qsfpdd-1	m24-800g-qsfpdd-1[2]

11.1.3.3 7750 SR-1-46S

Table 38: 7750 SR-1-46S default system layout

7750 SR-1-46S default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
SR-1-46S	A	4 GB	cpm-1x/i40-200g-sfpdd+6-800g-qsfpdd-1	—
	1		cpm-1x/i40-200g-sfpdd+6-800g-qsfpdd-1	—

Table 39: 7750 SR-1-46S supported hardware

7750 SR-1-46S supported hardware		
Chassis	**Card**	**MDA**
SR-1-46S	cpm-1x/i40-200g-sfpdd+6-800g-qsfpdd-1	m40-200g-sfpdd+6-800g-qsfpdd-1[2]

11.1.3.4 7750 SR-1-48D

Table 40: 7750 SR-1-48D default system layout

7750 SR-1-48D default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
SR-1-48D	A	4 GB	cpm-1x/i48-400g-qsfpdd-1	—
	1		cpm-1x/i48-400g-qsfpdd-1	—

Table 41: 7750 SR-1-48D supported hardware

7750 SR-1-48D supported hardware		
Chassis	**Card**	**MDA**
SR-1-48D	cpm-1x/i48-400g-qsfpdd-1	m48-400g-qsfpdd-1[2]

11.1.3.5 7750 SR-1-92S

Table 42: 7750 SR-1-92S default system layout

7750 SR-1-92S default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
SR-1-92S	A	4 GB	cpm-1x/i80-200g-sfpdd+12-400g-qsfpdd-1	—
	1		cpm-1x/i80-200g-sfpdd+12-400g-qsfpdd-1	—

Table 43: 7750 SR-1-92S supported hardware

7750 SR-1-92S supported hardware		
Chassis	**Card**	**MDA**
SR-1-92S	cpm-1x/i80-200g-sfpdd+12-400g-qsfpdd-1	m80-200g-sfpdd+12-400g-qsfpdd-1[2]

11.1.3.6 7750 SR-1e/2e/3e

Table 44: 7750 SR-1e default system layout

7750 SR-1e default system layout							
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA/ 1**	**MDA/ 2**	**MDA/ 3**	**MDA/ 4**
SR-1e	A	4 GB	cpm-e	—	—	—	—
	1		iom-e	me1-100gb-cfp2	me10-10gb-sfp+	me6-10gb-sfp+	isa2-aa

Table 45: 7750 SR-2e default system layout

7750 SR-2e default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
SR-2e	A	4 GB	cpm-e	—
	1		iom-e	me10-10gb-sfp+

Table 46: 7750 SR-3e default system layout

7750 SR-3e default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
SR-3e	A	4 GB	cpm-e	—
	1		ION: iom-e	me1-100gb-cfp2

Table 47: 7750 SR-1e/2e/3e supported hardware

7750 SR-1e/2e/3e supported hardware		
Chassis	**Card**	**MDA**
SR-1e SR-2e SR-3e	cpm-e	—
	iom-e	isa2-aa
		isa2-bb
		isa2-tunnel
		me10-10gb-sfp+
		me1-100gb-cfp2
		me12-10/1gb-sfp+
		me2-100gb-cfp4
		me2-100gb-ms-qsfp28
		me2-100gb-qsfp28
		me40-1gb-csfp
		me6-10gb-sfp+

7750 SR-1e/2e/3e supported hardware		
Chassis	**Card**	**MDA**
		me8-10/25gb-sfp28

11.1.3.7 7750 SR-1s

Table 48: 7750 SR-1s default system layout

7750 SR-1s default system layout					
Chassis	**Slot**	**Recommended memory per container**	**Card**	**XIOM**	**MDA**
SR-1s	A	6 GB	cpm-1s	—	s36-100gb-qsfp28

Table 49: 7750 SR-1s supported hardware

7750 SR-1s supported hardware			
Chassis	**Card**	**XIOM**	**MDA**
SR-1s	cpm-1s	—	s18-100gb-qsfp28
			s36-100gb-qsfp28
			s36-400gb-qsfpdd
			s36-100gb-qsfp28-3.6t
		iom-s-3.0t	ms2-400gb-qsfpdd+2-100gb-qsfp28
			ms3-200gb-cfp2-dco
			ms4-400gb-qsfpdd+4-100gb-qsfp28
			ms6-300gb-cfp2-dco
			ms8-100gb-sfpdd+2-100gb-qsfp28
			ms18-100gb-qsfp28
			ms16-100gb-sfpdd+4-100gb-qsfp28
			ms24-10/100gb-sfpdd

7750 SR-1s supported hardware			
Chassis	**Card**	**XIOM**	**MDA**
			ms16-sdd+4-qsfp28-b
			ms8-sdd+2-qsfp28-b

11.1.3.8 7750 SR-1se

Table 50: 7750 SR-1se default system layout

7750 SR-1se default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
SR-1se	A	6 GB	cpm-1se/imm36-800g-qsfpdd	—
	1		cpm-1se/imm36-800g-qsfpdd	—

Table 51: 7750 SR-1se supported hardware

7750 SR-1se supported hardware		
Chassis	**Card**	**MDA**
SR-1se	cpm-1se/imm36-800g-qsfpdd	ms36-800g-qsfpdd[2]

11.1.3.9 7750 SR-1x-48D

Table 52: 7750 SR-1x-48D default system layout

7750 SR-1x-48D default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
SR-1x-48D	A	4 GB	cpm-1x/i48-800g-qsfpdd-1x	—
	1		cpm-1x/i48-800g-qsfpdd-1x	—

Table 53: 7750 SR-1x-48D supported hardware

7750 SR-1x-48D supported hardware		
Chassis	**Card**	**MDA**
SR-1x-48D	cpm-1x/i48-800g-qsfpdd-1x	m48-800g-qsfpdd-1x[2]

11.1.3.10 7750 SR-1x-92S

Table 54: 7750 SR-1x-92S default system layout

7750 SR-1x-92S default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
SR-1x-92S	A	4 GB	cpm-1x/i80-200g-sfpdd+12-800g-qsfpdd-1x	—
	1		cpm-1x/i80-200g-sfpdd+12-800g-qsfpdd-1x	—

Table 55: 7750 SR-1x-92S supported hardware

7750 SR-1x-92S supported hardware		
Chassis	**Card**	**MDA**
SR-1x-92S	cpm-1x/i80-200g-sfpdd+12-800g-qsfpdd-1x	m80-200g-sfpdd+12-800g-qsfpdd-1x[2]

11.1.3.11 7750 SR-2s

Table 56: 7750 SR-2s default system layout

7750 SR-2s default system layout						
Chassis	**Slot**	**Recommended memory per container**	**SFM**	**Card**	**XIOM**	**MDA**
SR-2s	A	6 GB	sfm-2s	cpm-2s	—	—

7750 SR-2s default system layout						
Chassis	**Slot**	**Recommended memory per container**	**SFM**	**Card**	**XIOM**	**MDA**
	1			xcm-2s mda/1=s36-100gb-qsfp28	—	—

Table 57: 7750 SR-2s supported hardware

7750 SR-2s supported hardware				
	SFM	**Card**	**XIOM**	**MDA**
SR-2s	sfm-2s	cpm-2s	—	—
		xcm-2s	—	s18-100gb-qsfp28
				s36-100gb-qsfp28
				s36-400gb-qsfpdd
				s36-100gb-qsfp28-3.6t
			iom-s-1.5t iom-s-3.0t	ms2-400gb-qsfpdd+2-100gb-qsfp28
				ms3-200gb-cfp2-dco
				ms4-400gb-qsfpdd+4-100gb-qsfp28
				ms6-300gb-cfp2-dco
				ms8-100gb-sfpdd+2-100gb-qsfp28
				ms16-100gb-sfpdd+4-100gb-qsfp28
				ms18-100gb-qsfp28
				ms24-10/100gb-sfpdd
				ms16-sdd+4-qsfp28-b
				ms8-sdd+2-qsfp28-b

11.1.3.12 7750 SR-2se

Table 58: 7750 SR-2se default system layout

7750 SR-2se default system layout						
Chassis	**Slot**	**Recommended memory per container**	**SFM**	**Card**	**XIOM**	**MDA**
SR-2se	A	8 GB	sfm-2se	cpm-2se	—	—
	1			xcm-2se	—	me6-100gb-qsfp28

Table 59: 7750 SR-2se supported hardware

7750 SR-2se supported hardware				
Chassis	**SFM**	**Card**	**XIOM**	**MDA**
SR-2se	sfm-2se	cpm-2se	—	—
		xcm-2se	—	x2-s36-800g-qsfpdd-18.0t
				x2-s36-800g-qsfpdd-12.0t
			iom2-se-3.0t iom2-se-6.0t	mse24-200g-sfpdd
				mse14-800g+4-400g
				mse6-800g-qsfpdd
				mse6-800g-cfp2-dco
			x2-s36-800g-qsfpdd-6.0t	m36-800g-qsfpdd
			x2-s36-400g-qsfp112-3.0t	m36-400g-qsfp112
		xcmc-2se	iom2-se-3.0t iom2-se-6.0t	mse24-200g-sfpdd
				mse14-800g+4-400g
				mse6-800g-qsfpdd
				mse6-800g-cfp2-dco

7750 SR-2se supported hardware				
Chassis	SFM	Card	XIOM	MDA
			x2-s36-800g-qsfpdd-6.0t	m36-800g-qsfpdd
			x2-s36-400g-qsfp112-3.0t	m36-400g-qsfp112

11.1.3.13 7750 SR-7

Table 60: 7750 SR-7 default system layout

7750 SR-7 default system layout					
Chassis	Slot	Recommended memory per container	SFM	Card	MDA
SR-7	A	4 GB	m-sfm6-7/12	cpm5	—
	1			iom5-e	me6-100gb-qsfp28

Table 61: 7750 SR-7 supported hardware

7750 SR-7 supported hardware			
Chassis	SFM	Card	MDA
SR-7	m-sfm5-7 or m-sfm6-7/12	cpm5	—
	m-sfm5-7 or m-sfm6-7/12	imm-2pac-fp3	isa2-aa
			isa2-bb
			isa2-tunnel
			p10-10g-sfp
			p1-100g-cfp
	m-sfm5-7 or m-sfm6-7/12	imm48-1gb-sfp-c	imm24-1gb-xp-sfp
	m-sfm5-7 or m-sfm6-7/12	iom4-e	isa2-aa
			isa2-bb
			isa2-tunnel
			me10-10gb-sfp+

7750 SR-7 supported hardware			
Chassis	**SFM**	**Card**	**MDA**
			me1-100gb-cfp2
			me12-10/1gb-sfp+
			me2-100gb-cfp4
			me2-100gb-ms-qsfp28
			me2-100gb-qsfp28
			me40-1gb-csfp
			me6-10gb-sfp+
			me8-10/25gb-sfp28
	m-sfm5-7 or m-sfm6-7/12	iom4-e-b	isa2-aa
			isa2-bb
			isa2-tunnel
			me10-10gb-sfp+
			me1-100gb-cfp2
			me12-10/1gb-sfp+
			me2-100gb-cfp4
			me2-100gb-ms-qsfp28
			me2-100gb-qsfp28
			me40-1gb-csfp
			me6-10gb-sfp+
			me8-10/25gb-sfp28
	m-sfm5-12 or m-sfm6-7/12	iom4-e-hs	me10-10gb-sfp+
			me1-100gb-cfp2
			me12-10/1gb-sfp+
			me2-100gb-cfp4
			me2-100gb-ms-qsfp28
			me2-100gb-qsfp28
			me40-1gb-csfp

7750 SR-7 supported hardware			
Chassis	**SFM**	**Card**	**MDA**
			me6-10gb-sfp+
			me8-10/25gb-sfp28
	m-sfm6-7/12	iom5-e	me3-200gb-cfp2-dco
			me6-100gb-qsfp28
			me16-25gb-sfp28+2-100gb-qsfp28
			me3-400gb-qsfpdd
			me16-25gb-sfp28+2-100gb-qsfp-b

11.1.3.14 7750 SR-7s

Table 62: 7750 SR-7s default system layout

7750 SR-7s default system layout						
Chassis	**Slot**	**Recommended memory per container**	**SFM**	**Card**	**XIOM**	**MDA**
SR-7s	A	4 GB	sfm2-s	cpm2-s	—	—
	1	6 GB		xcm2-7s	—	x2-s36-800g-qsfpdd-18.0t

Table 63: 7750 SR-7s supported hardware

7750 SR-7s supported hardware				
Chassis	**SFM**	**Card**	**XIOM**	**MDA**
SR-7s	sfm-s	cpm-s cpm2-s	—	—
		xcm-7s	—	s18-100gb-qsfp28
				s36-100gb-qsfp28
				s36-400gb-qsfpdd
				s36-100gb-qsfp28-3.6t

7750 SR-7s supported hardware				
Chassis	SFM	Card	XIOM	MDA
			iom-s-1.5t iom-s-3.0t	ms2-400gb-qsfpdd +2-100gb-qsfp28
				ms3-200gb-cfp2-dco
				ms4-400gb-qsfpdd +4-100gb-qsfp28
				ms6-300gb-cfp2-dco
				ms8-100gb-sfpdd +2-100gb-qsfp28
				ms16-100gb-sfpdd +4-100gb-qsfp28
				ms18-100gb-qsfp28
				ms24-10/100gb-sfpdd
				ms16-sdd+4-qsfp28-b
				ms8-sdd+2-qsfp28-b
	sfm2-s	cpm2-s	—	—
		xcm2-7s	—	x2-s36-800g-qsfpdd-18.0t
				x2-s36-800g-qsfpdd-12.0t
			iom2-se-3.0t iom2-se-6.0t	mse24-200g-sfpdd
				mse14-800g+4-400g
				mse6-800g-qsfpdd
				mse6-800g-cfp2-dco
			x2-s36-800g-qsfpdd-6.0t	m36-800g-qsfpdd
			x2-s36-400g-qsfp112-3.0t	m36-400g-qsfp112
		xcm-7s-b	—	s18-100gb-qsfp28

7750 SR-7s supported hardware				
Chassis	SFM	Card	XIOM	MDA
				s36-100gb-qsfp28
				s36-400gb-qsfpdd
				s36-100gb-qsfp28-3.6t
			iom-s-1.5t iom-s-3.0t	ms2-400gb-qsfpdd +2-100gb-qsfp28
				ms3-200gb-cfp2-dco
				ms4-400gb-qsfpdd +4-100gb-qsfp28
				ms6-300gb-cfp2-dco
				ms8-100gb-sfpdd +2-100gb-qsfp28
				ms16-100gb-sfpdd +4-100gb-qsfp28
				ms18-100gb-qsfp28
				ms24-10/100gb-sfpdd
				ms16-sdd+4-qsfp28-b
				ms8-sdd+2-qsfp28-b

11.1.3.15 7750 SR-12

Table 64: 7750 SR-12 default system layout

7750 SR-12 default system layout					
Chassis	Slot	Recommended memory per container	SFM	Card	MDA
SR-12	A	4 GB	m-sfm6-7/12	cpm5	—
	1			iom5-e	me6-100gb-qsfp28

Table 65: 7750 SR-12 supported hardware

7750 SR-12 supported hardware			
Chassis	**SFM**	**Card**	**MDA**
SR-12	m-sfm5-12 or m-sfm6-7/12	cpm5	—
	m-sfm5-12 or m-sfm6-7/12	imm-2pac-fp3	isa2-aa
			isa2-bb
			isa2-tunnel
			p10-10g-sfp
			p1-100g-cfp
	m-sfm5-12 or m-sfm6-7/12	imm48-1gb-sfp-c	imm24-1gb-xp-sfp
	m-sfm5-12 or m-sfm6-7/12	iom4-e	isa2-aa
			isa2-bb
			isa2-tunnel
			me10-10gb-sfp+
			me1-100gb-cfp2
			me12-10/1gb-sfp+
			me2-100gb-cfp4
			me2-100gb-ms-qsfp28
			me2-100gb-qsfp28
			me40-1gb-csfp
			me6-10gb-sfp+
			me8-10/25gb-sfp28
	m-sfm5-12 or m-sfm6-7/12	iom4-e-b	isa2-aa
			isa2-bb
			isa2-tunnel
			me10-10gb-sfp+
			me1-100gb-cfp2

7750 SR-12 supported hardware			
Chassis	**SFM**	**Card**	**MDA**
			me12-10/1gb-sfp+
			me2-100gb-cfp4
			me2-100gb-ms-qsfp28
			me2-100gb-qsfp28
			me40-1gb-csfp
			me6-10gb-sfp+
			me8-10/25gb-sfp28
	m-sfm5-12 or m-sfm6-7/12	iom4-e-hs	me10-10gb-sfp+
			me1-100gb-cfp2
			me12-10/1gb-sfp+
			me2-100gb-cfp4
			me2-100gb-ms-qsfp28
			me2-100gb-qsfp28
			me40-1gb-csfp
			me6-10gb-sfp+
			me8-10/25gb-sfp28
	m-sfm6-7/12	iom5-e	me3-200gb-cfp2-dco
			me6-100gb-qsfp28
			me16-25gb-sfp28+2-100gb-qsfp28
			me3-400gb-qsfpdd
			me16-25gb-sfp28+2-100gb-qsfp-b

11.1.3.16 7750 SR-12e

Table 66: 7750 SR-12e default system layout

7750 SR-12e default system layout					
Chassis	**Slot**	**Recommended memory per container**	**SFM**	**Card**	**MDA**
SR-12e	A	4 GB	m-sfm6-12e	cpm5	—
	1			iom5-e	me6-100gb-qsfp28

Table 67: 7750 SR-12e supported hardware

7750 SR-12e supported hardware			
Chassis	**SFM**	**Card**	**MDA**
SR-12e	m-sfm5-12e or m-sfm6-12e	cpm5	—
	m-sfm5-12e or m-sfm6-12e	imm-2pac-fp3	isa2-aa
			isa2-bb
			isa2-tunnel
			p10-10g-sfp
			p1-100g-cfp
	m-sfm5-12e or m-sfm6-12e	imm40-10gb-sfp	m40-10g-sfp
	m-sfm5-12e or m-sfm6-12e	imm4-100gb-cfp4	m4-100g-cfp4
	m-sfm5-12e or m-sfm6-12e	iom4-e	isa2-aa
			isa2-bb
			isa2-tunnel
			me10-10gb-sfp+
			me1-100gb-cfp2
			me12-10/1gb-sfp+
			me2-100gb-cfp4

7750 SR-12e supported hardware			
Chassis	**SFM**	**Card**	**MDA**
			me2-100gb-ms-qsfp28
			me2-100gb-qsfp28
			me40-1gb-csfp
			me6-10gb-sfp+
			me8-10/25gb-sfp28
	m-sfm5-12e or m-sfm6-12e	iom4-e-b	isa2-aa
			isa2-bb
			isa2-tunnel
			me10-10gb-sfp+
			me1-100gb-cfp2
			me12-10/1gb-sfp+
			me2-100gb-cfp4
			me2-100gb-ms-qsfp28
			me2-100gb-qsfp28
			me40-1gb-csfp
			me6-10gb-sfp+
			me8-10/25gb-sfp28
	m-sfm5-12e or m-sfm6-12e	iom4-e-hs	me10-10gb-sfp+
			me1-100gb-cfp2
			me12-10/1gb-sfp+
			me2-100gb-cfp4
			me2-100gb-ms-qsfp28
			me2-100gb-qsfp28
			me40-1gb-csfp
			me6-10gb-sfp+
			me8-10/25gb-sfp28
	m-sfm6-12e	iom5-e	me12-100gb-qsfp28

7750 SR-12e supported hardware			
Chassis	SFM	Card	MDA
			me3-200gb-cfp2-dco
			me6-100gb-qsfp28
			me6-400gb-qsfpdd
			me16-25gb-sfp28+2-100gb-qsfp28
			me3-400gb-qsfpdd
			me16-25gb-sfp28+2-100gb-qsfp-b

11.1.3.17 7750 SR-14s

Table 68: 7750 SR-14s default system layout

7750 SR-14s default system layout						
Chassis	Slot	Recommended memory per container	SFM	Card	XIOM	MDA
SR-14s	A	4 GB	sfm2-s	cpm2-s	—	—
	1	8 GB		xcm2-14s		x2-s36-800g-qsfpdd-18.0t

Table 69: 7750 SR-14s supported hardware

7750 SR-14s supported hardware				
Chassis	SFM	Card	XIOM	MDA
SR-14s	sfm-s	cpm-s cpm2-s	—	—
		xcm-14s	—	s18-100gb-qsfp28
				s36-100gb-qsfp28
				s36-400gb-qsfpdd
				s36-100gb-qsfp28-3.6t

7750 SR-14s supported hardware				
Chassis	**SFM**	**Card**	**XIOM**	**MDA**
			iom-s-1.5t iom-s-3.0t	ms2-400gb-qsfpdd +2-100gb-qsfp28
				ms3-200gb-cfp2-dco
				ms4-400gb-qsfpdd +4-100gb-qsfp28
				ms6-300gb-cfp2-dco
				ms8-100gb-sfpdd +2-100gb-qsfp28
				ms16-100gb-sfpdd +4-100gb-qsfp28
				ms18-100gb-qsfp28
				ms24-10/100gb-sfpdd
				ms16-sdd+4-qsfp28-b
				ms8-sdd+2-qsfp28-b
	sfm2-s	cpm2-s	—	—
		xcm2-14s	—	x2-s36-800g-qsfpdd-18.0t
				x2-s36-800g-qsfpdd-12.0t
			iom2-se-3.0t iom2-se-6.0t	mse24-200g-sfpdd
				mse14-800g+4-400g
				mse6-800g-qsfpdd
				mse6-800g-cfp2-dco
			x2-s36-800g-qsfpdd-6.0t	m36-800g-qsfpdd
			x2-s36-400g-qsfp112-3.0t	m36-400g-qsfp112
		xcm-14s-b	—	s18-100gb-qsfp28

7750 SR-14s supported hardware				
Chassis	SFM	Card	XIOM	MDA
				s36-100gb-qsfp28
				s36-400gb-qsfpdd
				s36-100gb-qsfp28-3.6t
			iom-s-1.5t iom-s-3.0t	ms2-400gb-qsfpdd+2-100gb-qsfp28
				ms3-200gb-cfp2-dco
				ms4-400gb-qsfpdd+4-100gb-qsfp28
				ms6-300gb-cfp2-dco
				ms8-100gb-sfpdd+2-100gb-qsfp28
				ms16-100gb-sfpdd+4-100gb-qsfp28
				ms18-100gb-qsfp28
				ms24-10/100gb-sfpdd
				ms16-sdd+4-qsfp28-b
				ms8-sdd+2-qsfp28-b

11.1.3.18 7750 SR-a4/a8

Table 70: 7750 SR-a4 default system layout

7750 SR-a4 default system layout							
Chassis	Slot	Recommended memory per container	Card	MDA/ 1	MDA/ 2	MDA/ 3	MDA/ 4
SR-a4	A	4 GB	cpm-a	—			
	1		iom-a	maxp1-100gb-cfp	ma44-1gb-csfp	maxp10-10gb-sfp+	ma2-10gb-sfp+12-1gb-sfp

Table 71: 7750 SR-a8 default system layout

7750 SR-a8 default system layout				
Chassis	**Slot**	**Recommended memory per container**	**Card**	**MDA**
SR-a8	A	4 GB	cpm-a	—
	1		iom-a	maxp1-100gb-cfp

Table 72: 7750 SR-a4/a8 supported hardware

7750 SR-a4/a8 supported hardware		
Chassis	**Card**	**MDA**
SR-a4 SR-a8	cpm-a	—
	iom-a	ma20-1gb-tx
		ma2-10gb-sfp+12-1gb-sfp
		ma4-10gb-sfp+
		ma44-1gb-csfp
		maxp10-10/1gb-msec-sfp+
		maxp10-10gb-sfp+
		maxp1-100gb-cfp
		maxp1-100gb-cfp2
		maxp1-100gb-cfp4
		maxp6-10gb-sfp+1-40gb-qsfp+

11.1.4 7950 XRS

The following tables list the default system layout and supported hardware for the 7950 XRS chassis type.

11.1.4.1 7950 XRS-20/20e

Table 73: 7950 XRS-20/20e default system layout

7950 XRS-20/20e default system layout					
Chassis	**Slot**	**Recommended memory per container**	**SFM**	**Card**	**MDA**
XRS-20 XRS-20e	A	4 GB	sfm2-x20s	cpm2-x20	—
	1			xcm2-x20	x24-100g-qsfp28

Table 74: 7950 XRS-20/20e supported hardware

7950 XRS-20/20e supported hardware			
Chassis	**SFM**	**Card**	**MDA**
XRS-20 XRS-20e	sfm-x20 or sfm-x20-b or sfm-x20s-b	cpm-x20	—
	sfm2-x20s	cpm-x20 cpm2-x20	—
	sfm-x20	xcm-x20	x2-100g-tun
	sfm-x20-b or sfm-x20s-b or sfm2-x20s	xcm-x20	x2-100g-tun
			x40-10g-sfp
			x4-100g-cfp2
	sfm2-x20s	xcm2-x20	x12-400g-qsfpdd
			x24-100g-qsfp28
			x6-200g-cfp2-dco
			x6-400g-cfp8

11.2 Appendix B: Known limitations

The SR-SIM has the following limitations (this is not an exhaustive list):

- The SR-SIM is designed to replicate control and management plane operations, and port throughput is limited to 1000 pps/port as a result. However, overall throughput is dependent on the resources allocated.
- The control-plane performance is restricted.
- The filesystem on the SR-SIM is case-sensitive.
- In-Service software upgrade (ISSU) is not supported.

11.3 Appendix C: Example Kubernetes configuration using k3s on Ubuntu 24.04

11.3.1 Prerequisites

Ensure all host interfaces are configured with an MTU of 9000 before installation.

11.3.2 Install k3s

Replace `<my_secret_token>` in the following command with a private key specific to your installation.

```
curl -sfL https://get.k3s.io | INSTALL_K3S_EXEC="server" sh -s - --agent-token <my_secret_
token> --write-kubeconfig-mode "0666"
```

The `--write-kubeconfig-mode "0666"` flag is optional, allowing non-root users to issue `kubectl` deployment commands.

11.3.3 Ensuring the cni0 interface is set to an increased MTU

Issue the following command.

```
ifconfig cni0
```

The output should look similar to this.

```
cni0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 8950
        inet 10.42.0.1  netmask 255.255.255.0  broadcast 10.42.0.255
        inet6 fe80::181f:44ff:fe3f:484d  prefixlen 64  scopeid 0x20<link>
        ether 1a:1f:44:3f:48:4d  txqueuelen 1000  (Ethernet)
        RX packets 89118  bytes 687051420 (687.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 104066  bytes 697894863 (697.8 MB)
```

```
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

11.3.4 Ensuring the flannel.1 interface is set to an increased MTU

Issue the following command.

```
ifconfig flannel.1
```

The output should look similar to this.

```
flannel.1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 8950
        inet 10.42.0.0  netmask 255.255.255.255  broadcast 0.0.0.0
        inet6 fe80::7415:7cff:fe68:a94e  prefixlen 64  scopeid 0x20<link>
        ether 76:15:7c:68:a9:4e  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 7 overruns 0  carrier 0  collisions 0
```

If the MTU on the `flannel.1` interface is still set to a lower MTU, you can edit the `/etc/systemd/system/k3s.service` file to ensure the system is started based on the `cni0` interface (which has a larger MTU) replacing this line:

```
ExecStart=/usr/local/bin/k3s \
    server \
        '--agent-token' \
        '<my_secret_token>' \
        '--write-kubeconfig-mode' \
        '0666' \
```

With this:

```
ExecStart=/usr/local/bin/k3s \
    server \
        '--agent-token' \
        '<my_secret_token>' \
        '--write-kubeconfig-mode' \
        '0666' \
        '--flannel-iface' \
        'cni0' \
```

Now restart the `k3s` daemon.

```
sudo systemctl daemon-reload
sudo systemctl restart k3s
```

11.3.5 Ensuring transmit checksum offloading is disabled for the cni0 interface

Issue the following command.

```
sudo ethtool -K cni0 tx-checksum-ip-generic off
```

11.3.6 Create aliases for kuebctl

Create the following aliases in your shell. If you are using bash, add the following lines to the `~/.bash_aliases` file.

```
alias kubectl="k3s kubectl"
alias k="kubectl"
```

Then reread the file by issuing the following command.

```
source ~/.bash_aliases
```

11.3.7 Install a private container registry

Create the following files in a new directory dedicated to this purpose. Call it `container-registry`.

container-registry-ns.yml

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    kubernetes.io/metadata.name: container-registry
  name: container-registry
spec:
```

container-registry-pvc.yml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: registry-claim
  labels:
    app: container-registry
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: local-path
  resources:
    requests:
      storage: 10Gi
```

container-registry-deployment.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  labels:
    app: registry
  name: registry
  namespace: container-registry
spec:
  progressDeadlineSeconds: 600
```

```
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: registry
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: registry
    spec:
      containers:
      - env:
        - name: REGISTRY_HTTP_ADDR
          value: :5000
        - name: REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY
          value: /var/lib/registry
        - name: REGISTRY_STORAGE_DELETE_ENABLED
          value: "yes"
        image: registry:2.8.1
        imagePullPolicy: IfNotPresent
        name: registry
        ports:
        - containerPort: 5000
          name: registry
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        volumeMounts:
        - mountPath: /var/lib/registry
          name: registry-data
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
      volumes:
      - name: registry-data
        persistentVolumeClaim:
          claimName: registry-claim
```

container-registry-service.yml

```
apiVersion: v1
kind: Service
metadata:
  annotations:
  labels:
    app: registry
  name: registry
  namespace: container-registry
spec:
  externalTrafficPolicy: Cluster
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
```

```
  ports:
  - name: registry
    port: 32000
    protocol: TCP
    targetPort: 5000
  selector:
    app: registry
  sessionAffinity: None
  type: LoadBalancer
```

kustomization.yml

```
namespace: container-registry
resources:
  - container-registry-ns.yml
  - container-registry-deployment.yml
  - container-registry-service.yml
  - container-registry-pvc.yml
```

From the same directory, issue the following command to deploy the container registry.

```
kubectl apply -k .
```

If it deploys correctly, you should see output similar to the following when the `kubectl get all -n container-registry` command is issued.

```
NAME                            READY   STATUS    RESTARTS   AGE
pod/registry-579865c76c-75p5z   1/1     Running   0          15m

NAME               TYPE           CLUSTER-IP     EXTERNAL-IP                     PORT(S)
    AGE
service/registry   LoadBalancer   10.43.225.17   192.168.184.52,192.168.184.53   32000:31650/
TCP   15m

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/registry   1/1     1            1           15m

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/registry-579865c76c   1         1         1       15m
```

11.3.8 Install the Multus CNI

Install the Multus CNI, which provides the ability to add multiple interfaces to a container.

Create the following file in a different directory from the `container-registry` directory.

multus-k3s.yml

```
apiVersion: helm.cattle.io/v1
kind: HelmChart
metadata:
  name: multus
  namespace: kube-system
spec:
  repo: https://rke2-charts.rancher.io
  chart: rke2-multus
  targetNamespace: kube-system
  valuesContent: |-
```

```
    config:
      fullnameOverride: multus
      cni_conf:
        confDir: /var/lib/rancher/k3s/agent/etc/cni/net.d
        binDir: /var/lib/rancher/k3s/data/cni/
        kubeconfig: /var/lib/rancher/k3s/agent/etc/cni/net.d/multus.d/multus.kubeconfig
```

Deploy this manifest to the Kubernetes cluster using the following command.

```
kubectl apply -f multus-k3s.yml
```

If it deploys correctly, you should see output similar to the following when the `kubectl get all -n kube-system --show-labels -l app=rke2-multus` command is issued.

```
NAME                READY   STATUS    RESTARTS   AGE   LABELS
pod/multus-lmd8n   1/1     Running   0          74m   app=rke2-multus,controller-revision-hash=
7f7588f4bd,pod-template-generation=1,tier=node

NAME                    DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
       AGE   LABELS
daemonset.apps/multus   1         1         1       1            1           kubernetes.io/os=
linux   74m   app.kubernetes.io/managed-by=Helm,app=rke2-multus,tier=node
```

Customer document and product support

Customer documentation
Customer documentation welcome page

Technical support
Product support portal

Documentation feedback
Customer documentation feedback