Juniper Networks | Engineering Simplicity

cJunosEvolved Deployment Guide

Published
2025-06-26

Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, California 94089
USA
408-745-2000
www.juniper.net

cJunosEvolved Deployment Guide

The information in this document is current as of the date on the title page.

YEAR 2000 NOTICE

Juniper Networks hardware and software products are Year 2000 compliant. Junos OS has no known time-related limitations through the year 2038. However, the NTP application is known to have some difficulty in the year 2036.

END USER LICENSE AGREEMENT

The Juniper Networks product that is the subject of this technical documentation consists of (or is intended for use with) Juniper Networks software. Use of such software is subject to the terms and conditions of the End User License Agreement ("EULA") posted at https://support.juniper.net/support/eula/. By downloading, installing or using such software, you agree to the terms and conditions of that EULA.

Table of Contents

About This Guide

Use this guide to install the containerized JunosEvolved (cJunosEvolved) on Docker. This guide includes basic cJunosEvolved configuration and management procedures.

cJunosEvolved is a containerized version of the Junos OS Evolved-based PTX platform that represents a Juniper Networks® router running Junos OS Evolved in a Linux environment.

After installing and configuring the cJunosEvolved platform as covered in this guide, refer to Junos® operating system Evolved (Junos OS Evolved) documentation for information about additional software configurations.

RELATED DOCUMENTATION

- Junos OS Evolved Documentation

PART 1

cJunosEvolved Deployment for Docker

CHAPTER 1

Overview

IN THIS CHAPTER

What Is cJunosEvolved

SUMMARY

This topic provides an overview, key features, benefits, and limitations of cJunosEvolved.

IN THIS SECTION

Overview

cJunosEvolved is a containerized version of the two Junos OS Evolved single form factor PTX platforms. It can run directly on an x86 server or within a VM running on an x86 server.

Either of the following PTX platforms can be emulated with cJunosEvolved:

- PTX10001-36MR–Simulates the Express 4 (BT) chipset
- PTX10002-36QDD–Simulates the Express 5 (BX) chipset

NOTE: In this document, wherever there is a difference in behavior between BT and BX platforms cJunosEvolved-BT will refer to PTX10001-36MR and cJunosEvolved-BX will

refer to PTX10002-36QDD. Generically, the product family is referred to as cJunosEvolved.

cJunosEvolved is intended only for lab use and not for commercial deployments. Being a containerized virtual product, it does not support the same bandwidth and speed as the physical PTX.

NOTE: You do not need feature licenses, and we do not provide any. You can ignore any license check messages if you receive an alert.

Key Features

This topic provides a list of key features that the cJunosEvolved platform supports.

The cJunosEvolved platforms support the following interface scaling:

- cJunosEvolved-BT supports up to 12 WAN interfaces and 72 channelized interfaces.
- cJunosEvolved-BX supports up to 36 WAN interfaces and 144 channelized interfaces.

For more information on the PTX platforms that cJunosEvolved is based on, please refer to the platform datasheets:

- **cJunosEvolved-BT :**PTX10001 and PTX10003 Fixed Configuration Routers Datasheet
- **cJunosEvolved-BX :**PTX10002-36QDD Fixed Configuration Router Datasheet

For details on configuration of these features, refer to Software Documentation.

Benefits and Uses

The benefits and use cases of cJunosEvolved on standard x86 servers are as follows:

- Reduced capital expenditure (Capex)—cJunosEvolved is available for free to build test labs reducing costs associated with physical routers.
- Reduced deployment time—You can use cJunosEvolved to build and to test topologies virtually without building expensive physical labs. Virtual labs can be built instantly. As a result, you can reduce costs and delays associated with physical hardware deployment.
- Eliminate lab hardware—cJunosEvolved helps you eliminate waiting time for lab hardware to arrive after procurement. cJunosEvolved is available for free and can be downloaded instantly.
- Education and training—Allows you to build labs for learning and education services for your employees.

- Automate, build and validate—You can validate various data center routing and switching topologies, pre-build configurations examples, and get automation ready.

Limitations

IN THIS SECTION

The cJunosEvolved has the following limitations:

Limitations Common to BT and BX Versions

- cJunosEvolved is intended only for lab use and not for commercial deployment.
- Fixed form factor: a single Routing Engine and single FPC architecture.
- The EVO RE and FPC image can be upgraded at run-time, but the data plane simulator can only be upgraded via installing a new container image. The release notes for every release will list the changes for that release and mention if there is a need to upgrade the data plane simulator in the list of fixed defects.
- The cJunosEvolved does not support in-service software upgrade (ISSU).
- You cannot attach or detach interfaces while cJunosEvolved is running.
- Supports a maximum bandwidth of 2 kpps or 3-5 Mbps over all the interfaces.
- Does not support Single Root I/0 Virtualization (SR-IOV)
- The BT or BX data simulators, also known as COSIM reliably work up to 2000 pps across 128B-1500B packet length. You do not need a bandwidth license.
- Routing Engine and PFE Resiliency support
- Timing and Synchronization
- QOS features that operate in real time e.g., shaping, policing BFD and other protocols with aggressive (sub-second) timers
- Fabric chip functionality (not to be confused with network fabrics)

- Hardware components like temperature, voltage sensors, etc.
- TPM
- For cJunosEvolved to function correctly as a Virtual Extensible LAN (VXLAN) Tunnel End Point (VTEP), you must configure tunnel termination using the set forwarding-options tunnel termination command. Otherwise, the traffic in the tunnel drops on the egress VTEP.

cJunosEvolved-BT Limitations and Unsupported Features

The following are the limitations on cJunosEvolved-BT platform:

- The only channelization factor supported is 8x25G or 4x25G.
- FPC restart is not supported.

The following features are the unsupported features on cJunosEvolved-BT platform:

- Storm control
- Multichassis link aggregation (MC-LAG)
- VXLAN seamless stitching for Data Center Interconnect (DCI)
- Virtual Router Redundancy Protocol (VRRP)
- Q-in-Q tunneling through an Ethernet VPN–Virtual Extensible LAN (EVPN-VXLAN) fabric
- Layer-2 egress filtering on EVPN-VXLAN-enabled integrated routing and bridging (IRB) interfaces
- IPv6 underlay and overlay
- MAC filtering in EVPN-VXLAN fabric using a MAC list

cJunosEvolved-BX Limitations and Unsupported Features

The following are the limitations on cJunosEvolved-BX platform:

- Only 4x100G channelization is supported

The following features are the unsupported features on cJunosEvolved-BX platform:

- Port mirroring with policers
- Firewall policer and Firewall filter flex offset match
- MACsec IFD support
- Inline CFM

- Optics and EM policy

cJunosEvolved Architecture

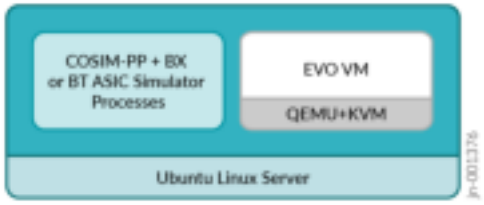
IN THIS SECTION

This topic provides details of cJunosEvolved-BT and cJunosEvolved-BT architecture.

cJunosEvolved is a KVM based Docker container. The embedded EVO VM within the container provides the Junos OS Evolved control and management plane functionality. The EVO VM includes the Routing Engine and Packet Forwarding Engine. The BT and BX data plane ASIC simulators reside in the Linux side of the container.

Figure 1 on page 6 shows the high-level architecture of cJunosEvolved and how it resides in an Ubuntu host server. cJunosEvolved can be deployed on a bare metal Ubuntu Linux server as well as from within an Ubuntu VM. The bare metal server is the preferred deployment model as the cJunosEvolved container embeds an EVO VM within it. Therefore, hosting it within another VM will use nested virtualization which inevitably slows the response of cJunosEvolved.

Figure 1: cJunosEvolved High Level Architecture



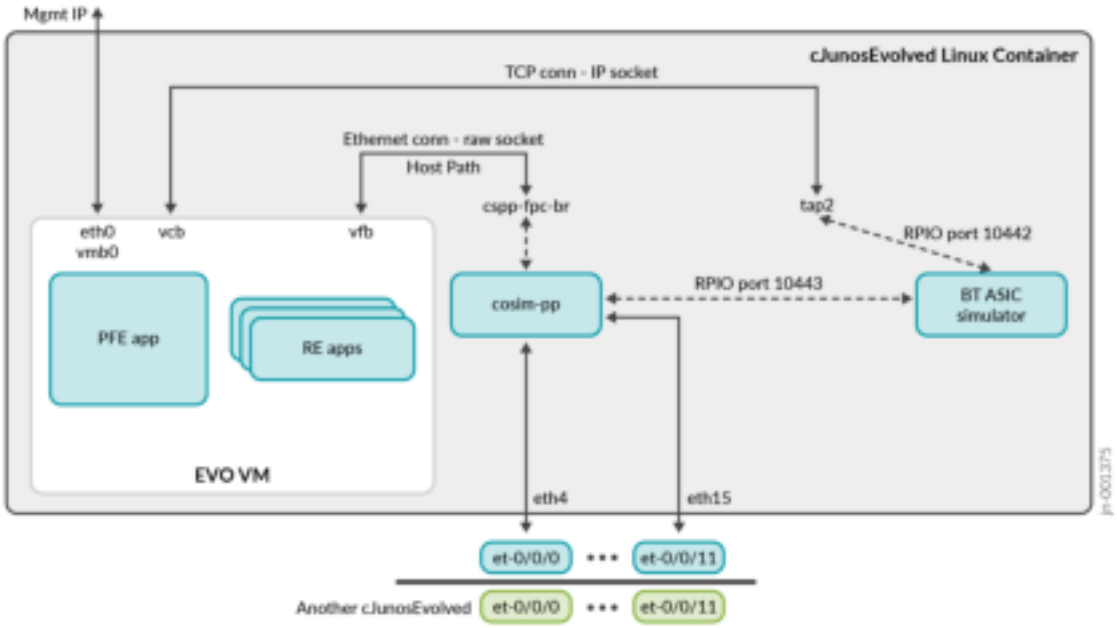
cJunosEvolved-BT Architecture

This section describes the detailed cJunosEvolved-BT Architecture.

The Linux side of the container includes a single instance of the COSIM packet processing (cosim-pp) plus the BT ASIC simulator process.

The COSIM is situated between the external virtual ether (veth) data ports, the BT and the EVO VM. It facilitates packet processing between them.

Figure 2: cJunosEvolved-BT Architecture



cJunosEvolved-BX Architecture

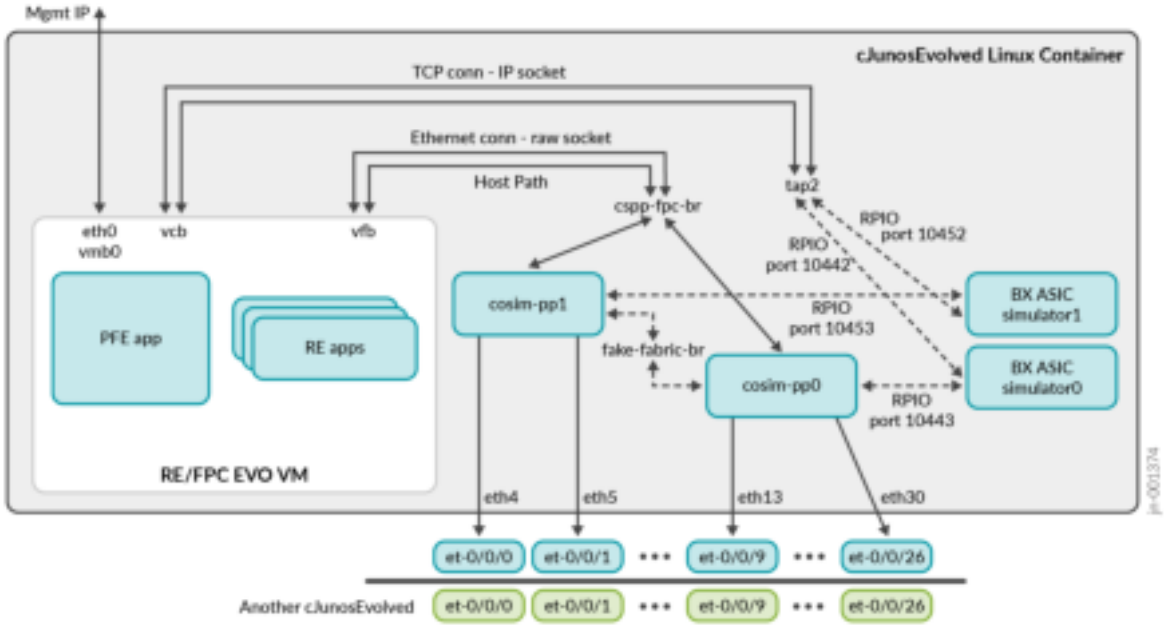
This section provides detailed information about cJunosEvolved-BX Architecture.

The Linux side of the container includes two instances of the COSIM packet processing (cosim-pp) plus the BX ASIC simulator process.

Each COSIM is situated between the external virtual ether (veth) data ports, it's corresponding BX and the EVO VM. It facilitates packet processing between them.

The BX ASIC is simulated in software. The Packet Forwarding Engine in EVO VM communicates with the data plane within the container for configuration, control and data packet processing.

Figure 3: cJunosEvolved-BX Architecture



The EVO VM also contains the Routing Engine applications as well as other processes such as MGD for router configuration and the control and management plane. The EVO VM is started at cJunosEvolved initialization through QEMU-KVM.

CHAPTER 2

cJunosEvolved Hardware and Software Requirements on Docker

IN THIS CHAPTER

Minimum Hardware and Software Requirements

This topic provides you a list of hardware and software requirements to start a cJunosEvolved instance.

Minimum Hardware Requirements for cJunosEvolved on page 9 lists the hardware requirements for cJunosEvolved.

Table 1: Minimum Hardware Requirements for cJunosEvolved

Description	Value
Sample system configuration	For lab simulation and low performance (less than 2Kpps) use cases, any x86 processor (Intel or AMD) with VT-x capability. Broadwell processors or preferably later Example of Broadwell processor: Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40 GHz
Number of cores	A minimum of 4 cores are required. This is the standard support with vJunosEvolved platform and can be adjusted in the configuration based on requirement.
Memory	A minimum of 8 GB is required.
Disk	20 G for container at runtime, can grow dynamically up to 40 G

Table 1: Minimum Hardware Requirements for cJunosEvolved *(Continued)*

Description	Value
Other requirements	• Intel VT-x capability. • Hyperthreading (recommended) • AES-NI

Table 2: Software Requirements for Ubuntu

Description	Value
Operating system **NOTE**: Only English localization is supported.	• Ubuntu 24.04 LTS • Ubuntu 22.04 LTS

Table 2: Software Requirements for Ubuntu *(Continued)*

Description	Value

Virtualization	• QEMU-KVM
	The default version for each Ubuntu version is sufficient.
	• 1. First install the following package:
	`# apt-get install cpu-checker`
	2. Then run the following command to verify the server supports qemu-kvm:
	`# kvm-ok`
	`INFO: /dev/kvm exists KVM acceleration can be used`
	3. Now run the following command to install qemu-kvm:
	`# apt-get install qemu-kvm`
	The following command will show the installed version of QEMU, the standard current version should be OK:
	`# /usr/bin/qemu-system-x86_64 -version`
	`QEMU emulator version 6.2.0 (Debian 1:6.2+dfsg-2ubuntu6.26)`
	`Copyright (c) 2003-2021 Fabrice Bellard and the QEMU Project developers`

Table 2: Software Requirements for Ubuntu *(Continued)*

Description	Value
Required packages	These Docker and Docker Compose versions have been verified with cJunosEvolved on Ubuntu 22.04 and 24.04 host servers. **CAUTION**: Do not use any earlier version of Docker Engine than 28.0.0, as the earlier versions can connect the interfaces to cJunosEvolved out of order, leading to traffic failing between connected containers. • Docker Engine - Community version 28.1.1 • Docker Compose version v2.35.1
Supported Deployment Environments	Ubuntu 22.04 or 24.04 host server with the software requirements specified above Multiple cJunosEvolved instances can be deployed on the same host server in a Docker Compose topology that connects them together. The Containerlab open source deployment is supported and will be documented on Containerlab
cJunosEvolved Images	You can access the images from the vJunos Labs download area of juniper.net at: Free Virtual Junos OS Download for Labs

CHAPTER 3

Install and Deploy cJunosEvolved on Docker

IN THIS CHAPTER

Install cJunosEvolved on Docker

SUMMARY

Read this topic to understand how to install the cJunosEvolved in a Docker environment.

IN THIS SECTION

Prepare the Linux Host Servers to Install cJunosEvolved

1. Install the standard package versions for your Ubuntu host server to ensure that the servers meet the minimum hardware and software requirements.
2. You can install the latest version of Docker and Docker Compose. These change frequently and occasionally have bugs. If you hit an issue with them, recommend that you change the Docker and Docker Compose packages to match the ones specified in Table 2. These are the versions verified during development and test of cJunosEvolved.

   Use the following link to install the Docker packages. If your server does not have Docker on it, the “Install using the apt repository” Section shows an easy way to do this:

   https://docs.docker.com/engine/install/ubuntu/

The Docker and Docker Compose versions which have been verified for this release of cJunosEvolved are:

```
# docker version
Client: Docker Engine - Community
 Version:           28.1.1
 API version:       1.49
 Go version:        go1.23.8
 Git commit:        4eba377
 Built:             Fri Apr 18 09:52:10 2025
 OS/Arch:           linux/amd64
 Context:           default


Server: Docker Engine - Community
 Engine:
  Version:          28.1.1
  API version:      1.49 (minimum version 1.24)
  Go version:       go1.23.8
  Git commit:       01f442b
  Built:            Fri Apr 18 09:52:10 2025
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.7.27
  GitCommit:        05044ec0a9a75232cad458027ca83437aae3f4da
 runc:
  Version:          1.2.5
  GitCommit:        v1.2.5-0-g59923ef
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0

# docker compose version
Docker Compose version v2.35.1
```

3. Verify that Intel VT-x technology is enabled. Run the **lscpu** command on your host server. The **Virtualization** field in the output of the lscpu command will display VT-x, if VT-x is enabled. If VTx is not enabled, then see your server documentation to learn how to enable it in BIOS.

```
# lscpu | grep Virtualization
```

```
Virtualization:                      VT-x
```

Deploy and Manage cJunosEvolved on Docker

SUMMARY

Read this topic to understand how to deploy and manage the cJunosEvolved instance after you install it on Docker.

IN THIS SECTION

This topic describes:

Overview

Docker Compose is used to start a topology of cJunosEvolved routers that are configured and connected according to the specifications in a YAML file. The YAML file greatly simplifies deployment, as it is a recipe that allows the containers in it to be configured and connected to each other accordingly.

This section describes the key features of Docker Compose as it pertains to cJunosEvolved deployment.

A sample layout of a Docker Compose file is shown below. For complete example files refer to the vJunos Labs download page on Juniper's website.

Docker Compose Lay Out Description

This section provides a high-level overview of the Docker Compose YAML file. The details are described in the deployment section.

The services section of the file contains the following fields for each container being deployed:

- The cJunosEvolved Docker image
- The container name

- The optional hostname
- A mandatory field indicating that privileged mode is used
- The environmental variables that can be passed into the container are shown in a subsequent table.
- An ordered and prioritized list of the Docker networks for each container, including the single management Ethernet port and whatever WAN (data) Ethernet ports Docker needs to supply to the container.

There is also an overall networks section in the YAML file. The IP subnet address for each Ethernet port is specified here. This allows Docker to connect the interfaces between the containers properly so that traffic can be carried between them. The management port subnet for the containers in the topology should be specified in the same section.

Environmental Variables

The environmental variables that can be specified for each container in the Docker Compose YAML file are specified below:

Table 3: Environmental Variables

Env Variable	Value	Description
CPTX_COSIM	BT	"CPTX_COSIM: BT" emulates the PTX10001-36MR platform which uses the BT chipset. This is the default setting.
CPTX_COSIM	BX	"CPTX_COSIM: BX" emulates the PTX10002-36QDD platform which uses the BX chipset.
CPTX_CHANNELIZED (optional)	1	"CPTX_CHANNELIZED: 1" places the interfaces for either the BT or BX types into channelized mode. If this environmental variable is not specified, the interfaces will be in unchannelized mode, which is the default mode.
CPTX_EVOVM_CPU (optional)	Number of cores for EVOVM	A minimum of 4 cores is required. There is no CPU pinning.

Table 3: Environmental Variables *(Continued)*

Env Variable	Value	Description
CPTX_EVOVM_MEM_MB (optional)	MB of memory to EVO VM	A minimum of 6292 MB is required. **NOTE**: A minimum of 8GB of memory is required for the entire cJunosEvolved container. Which also includes COSIM and QEMU in the container outside the EVO VM.
CPTX_AUTO_CONFIG (optional)	1	Set to "1" to enable auto provisioning configuration. Auto configuration takes precedence over passing in the configuration via a disk specified in Docker Compose.

WAN Interfaces Mapping

IN THIS SECTION

This topic provides the details of interface mapping between the Linux style "eth" WAN ports and the CLI interface naming convention for each type of cJunosEvolved.

cJunosEvolved-BT Unchannelized Interface Mapping

There are 12 unchannelized interfaces supported by the BT version of cJunosEvolved. These interfaces are numbered from et-0/0/0 to et-0/0/11 and are nominally rated at 400 G.

The Docker Compose Linux eth4 – eth15 interfaces correspond to the et-0/0/0 – et-0/0/11 interfaces in the CLI configuration.

cJunosEvolved-BT Channelized WAN Interface Mapping

By using channelization, the 12 unchannelized BT interfaces can be multiplexed into 72 channelized interfaces. The nominal speed for each channelized port is 8x25G. Some physical ports are shut down and not used as per this table once channelization is turned on.

Table 4: cJunosEvolved-BT Channelized WAN Interface Mapping

Channelized port in CLI Notation	Ports	Docker Compose (i.e., Linux) notation	Speed
et-0/0/0:0 to et-0/0/0:7	8	eth4 – eth11	8x25G
et-0/0/1:0 to et-0/0/1:7	8	eth12- eth19	8x25G
et-0/0/2:0 to et-0/0/2:7	8	eth20-eth27	8x25G
et-0/0/3:0 to et-0/0/3:7	8	eth28-eth35	8x25G
et-0/0/4:0 to et-0/0/4:3 (retimer)	4	eth36-eth39	4x25G
et-0/0/5	0	-	shutdown
et-0/0/6:0 to et-0/0/6:3 (retimer)	4	eth40 – eth43	4x25G
et-0/0/7	0	-	shutdown
et-0/0/8:0 to et-0/0/8:7	8	eth44- eth51	8x25G
et-0/0/9:0 to et-0/0/9:7	8	eth52 – eth59	8x25G

Table 4: cJunosEvolved-BT Channelized WAN Interface Mapping *(Continued)*

Channelized port in CLI Notation	Ports	Docker Compose (i.e., Linux) notation	Speed
et-0/0/10:0 to et-0/0/10:7	8	eth60 – eth67	8x25G
et-0/0/11:0 to et-0/0/11:7	8	eth68 – eth75	8x25G

cJunosEvolved-BX Unchannelized WAN Interface Mapping

This mode supports a total of 36 WAN interfaces. The nominal port speed for each interface is 800 G.

The Docker Compose Linux eth4-eth39 ports correspond to the CLI et-0/0/0- et-0/0/35 interfaces, respectively.

cJunosEvolved-BX Channelized WAN Interface Mapping

Each BX ASIC simulator supports 72 channelized interfaces at 100 G. This yields 144 total channelized interfaces.

Use a CLI command like below to channelize each unchannelized interface, et-0/0/0 is just an example and set the speed to 100 G.

```
set groups global interfaces et-0/0/0 number-of-sub-ports 4
```

```
set groups global interfaces et-0/0/0 speed 100g
```

The following tables show the channelization WAN port mapping. The channelization factor is 4x100G, as each “physical” WAN port is divided into 4 channels.

Table 5: BX Channelization WAN Port Mapping

Channelized Port in CLI Notation (Lower 72)	Docker Compose Linux interface
et-0/0/0:0 to et-0/0/0:3	eth4 – eth7

Table 5: BX Channelization WAN Port Mapping *(Continued)*

Channelized Port in CLI Notation (Lower 72)	Docker Compose Linux interface
et-0/0/1:0 to et-0/0/1:3	eth8- eth11
et-0/0/2:0 to et-0/0/2:3	eth12-eth15
et-0/0/3:0 to et-0/0/3:3	eth16-eth19
et-0/0/4:0 to et-0/0/4:3	eth20-eth23
et-0/0/5:0 to et-0/0/5:3	eth24-eth27
et-0/0/6:0 to et-0/0/6:3	eth28-eth31
et-0/0/7:0 to et-0/0/7:3	eth32-eth35
et-0/0/8:0 to et-0/0/8:3	eth36-eth39
et-0/0/9:0 to et-0/0/9:3	eth40-eth43
et-0/0/10:0 to et-0/0/10:3	eth44-eth47
et-0/0/11:0 to et-0/0/11:3	eth48-eth51
et-0/0/12:0 to et-0/0/12:3	eth52-eth55
et-0/0/13:0 to et-0/0/13:3	eth56-eth59
et-0/0/14:0 to et-0/0/14:3	eth60-eth63
et-0/0/15:0 to et-0/0/15:3	eth64-eth67

Table 5: BX Channelization WAN Port Mapping *(Continued)*

Channelized Port in CLI Notation (Lower 72)	Docker Compose Linux interface
et-0/0/16:0 to et-0/0/16:3	eth68-eth71
et-0/0/17:0 to et-0/0/17:3	eth72-eth75

Table 5: BX Channelization WAN Port Mapping *(Continued)*

Channelized Port (Upper 72)	Docker Compose Linux Notation
et-0/0/18:0 to et-0/0/18:3	eth76-eth79
et-0/0/19:0 to et-0/0/19:3	eth80-eth83
et-0/0/20:0 to et-0/0/20:3	eth84- eth87
et-0/0/21:0 to et-0/0/21:3	eth88-eth91
et-0/0/22:0 to et-0/0/22:3	eth92-eth95
et-0/0/23:0 to et-0/0/23:3	eth96-eth99
et-0/0/24:0 to et-0/0/24:3	eth100-eth103
et-0/0/25:0 to et-0/0/25:3	eth104-eth107
et-0/0/26:0 to eth-0/0/26:3	eth108-eth111
et-0/0/27:0 to et-0/0/27:3	eth112-eth115
et-0/0/28:0 to et-0/0/28:3	eth116-eth119
et-0/0/29:0 to et-0/0/29:3	eth120-eth123

Table 5: BX Channelization WAN Port Mapping *(Continued)*

Channelized Port (Upper 72)	Docker Compose Linux Notation
et-0/0/30:0 to et-0/0/30:3	eth124-eth127
et-0/0/31:0 to et-0/0/31:3	eth128-eth131
et-0/0/32:0 to et-0/0/32:3	eth132-eth135
et-0/0/33:0 to et-0/0/33:3	eth136-eth139
et-0/0/34:0 to et-0/0/34:3	eth140-eth143
et-0/0/35:0 to et-0/0/35:3	eth144-eth147

Deploy cJunosEvolved

Docker Compose and its associated YAML file is used for deployment.

NOTE: This topic highlights only a few sections of the YAML file for deploying cJunosEvolved through Docker Compose. Some sample cJunosEvolved Compose files are available for download along with the required container image and associated documentation from the lab download page.

1. Load the cJunosEvolved Docker Compose image onto Ubuntu host.

```
# docker image load -i cJunosEvolved-<release>-EVO.tar.gz
```

2. Run the command in this step to see the image name and its Docker tag.

   This is only an example image. The repository and tag names need to be provided in the Docker Compose YAML file.

```
#docker image ls
REPOSITORY         TAG            IMAGE ID      CREATED      SIZE
cjunosevolved      25.2R1.1-EVO   84b56d3b4090  5 hours ago  2.01GB
```

3. Edit the services section of the YAML file as described in the following steps. Do this for each container in your topology.
4. Set the image field in the YAML file, to <repository:tag> as shown below. Also, choose your container name and hostname. By convention, the hostname should be what you want to set as hostname in the CLI configuration. The **container_name** identifies the container to the host server and to the various Docker commands, such as `docker container ls`, `docker logs <container_name` and so on.

   NOTE: Ensure that the privileged mode must be set to **true** for each container in the YAML file.

   The image is tagged at build time. You can use that tag as shown below or retag it through Docker.

```
services:
  cJunosEvo1:
    image: 'cjunosevolved:25.2R1.1-EVO'
    container_name: cevo1
    hostname: cevo1
    privileged: true
```

5. Set the environmental variables in the YAML file. This is under the services section as well. Refer to the “Environmental Variables” section above.

```
CPTX_COSIM: BX
```

- **NOTE**: For BT set it to " CPTX_COSIM: BT"

  This only needs to be set if you want to assign more CPUs to EVOVM. 4 is the default (and minimum) value and hence is optional. Specify this only if you want to assign more MB of memory to the EVOVM. 6292 MB is the default value and hence does not need to be specified.

```
CPTX_EVOVM_CPU: 4
```

- Specify this only if you want to assign more MB of memory to the EVOVM. 6292 MB is the default value and hence does not need to be specified.

```
CPTX_EVOVM_MEM_MB: 6292
```

- Optional way to configure cJunosEvolved. Refer to the configuration section.

6. Edit the management Ethernet port to configure the IP address. This is under the services section for the specific container in the YAML file. eth0_mgmt designates the management port in the YAML file. The actual interface in the container is “eth0” and must be called in the driver_opts.

```
networks:
  eth0_mgmt:
    ipv4_address: 100.92.240.2
    driver_opts:
      com.docker.network.endpoint.ifname: eth0
```

CAUTION: To ensure proper connectivity of networks to interfaces, you must use the driver_opts stanza mentioned above for every interface of the container, including WAN (data) interfaces. For more information see https://github.com/docker/compose/issues/12776

- If you have not set the CPTX_AUTO_CONFIG environment option for the containers in the YAML file, do not specify any WAN interface IP in the YAML file as it will be ignored. Once the container is up and running, you need to pass in a CLI configuration using the `volume` command or manually configure the IP through CLI.

- Always set the IP address for the management interface in the YAML file and also specify its subnet IP in the overall network section of the file. This is so you have control over what IP address is assigned to the management interface. This is true whether CPTX_AUTO_CONFIG is used or not.

- If you are passing the EVO CLI configuration using the `volumes` command in the YAML file, configure the same management IP address and subnet in the passed in “juniper.conf” as you have in the YAML file.

7. Similarly, edit the other interfaces in the YAML file, under the **networks** subsection of the services clause for each container. These first 3 are reserved for internal cJunosEvolved use and should be left as is for proper operation.

```
eth1_reserved:
  driver_opts:
    com.docker.network.endpoint.ifname: eth1
eth2_reserved:
  driver_opts:
    com.docker.network.endpoint.ifname: eth2
eth3_reserved:
  driver_opts:
    com.docker.network.endpoint.ifname: eth3
```

The WAN (data) interfaces start from eth4 on Linux side. There will be multiple of these as per your needs. Only set the IP address for WAN ports if you are using CPTX_AUTO_CONFIG.

```
eth4:
  ipv4_address: 100.1.1.2
  driver_opts:
    com.docker.network.endpoint.ifname: eth4
```

8. You can pass in an entire JunosEvolved hierarchical CLI configuration through the following stanza in the Docker Compose file.

   The hierarchical configuration format is what is shown when the you type the `show configuration` command in the CLI mode. This is the most comprehensive way to configure cJunosEvolved at startup. If you choose this option, then do not specify WAN IP addresses and subnets in the Docker Compose file. cJunosEvolved will only use IP address/subnets specified in the YAML file when CPTX_AUTO_CONFIG is configured. Otherwise, you can use the Disk Configuration mode or login to the container and configure them manually.

9. Alternatively, you can use the “CPTX_AUTO_CONFIG: 1” environmental variable, as specified in the “Configure cJunosEvolved on Docker, Automatic Configuration” section.

10. You must also specify the overall “networks” section of the YAML file.

    This section is common for all the containers in the YAML file, which is common to all the containers. The **networks** section is listed after the **services** section for the various containers. This section lists all the WAN interfaces as well as the management interface for the topology. If you specified the IP addresses for interfaces in the matching network section of the YAML file for each container, specify the corresponding subnet IP for the same network in the overall networks

section. This allows the interfaces in the individual containers to exchange IP traffic between each other. For example, in previous step one of the containers had this stanza:

```
eth4:
  ipv4_address: 100.1.1.2
```

So, in the overall networks section common to the containers in the YAML file, you must specify a valid subnet IP for the same interface as:

```
networks:
  eth4:
    name: eth4
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 100.1.1.0/24
```

11. Create the cJunosEvolved containers specified in the YAML file using the following command from the host server.

```
# docker compose -f <docker-compose-filename>.yaml up -d
```

The **-d** option runs the containers in the background. This means control is returned to the terminal you run this command from, and you can use the terminal for other purposes.

12. Run the `docker network ls` command from the host server to verify that the order of the network interfaces matches the order specified in the YAML file. Otherwise, the connection between the containers specified in the YAML file may be “crossed up”.

```
# docker network ls
NETWORK ID     NAME            DRIVER    SCOPE
# this is a default Docker bridge, not created due to the YAML file
785df5e996d5   bridge          bridge    local
6222dbea40c6   eth0_mgmt       bridge    local
fb5a95cd7236   eth1_reserved   bridge    local
cec52e294667   eth2_reserved   bridge    local
4f6d181ab5ec   eth3_reserved   bridge    local
3a8ccd8d44dc   eth4            bridge    local
```

```
4e701a843d95   eth5            bridge    local
6e3035f60692   eth6            bridge    local
bf4f5b2583aa   eth7            bridge    local
0f8822dda3ab   eth8            bridge    local
507f0ad3feaf   eth9            bridge    local
4af981ed503e   eth10           bridge    local
7d280f0f348a   eth11           bridge    local
# These two are defaults bridges and not created due to the YAML file
2304e4ab7184   host            host      local
da25cbdc438c   none            null      local
```

13. Run the `docker container ls` command from the host server to verify that the containers specified in the Docker Compose file have been created:

```
user@host:/var/home/user# docker container ls
CONTAINER ID  IMAGE                        COMMAND           CREATED
STATUS            PORTS NAMES
4eccebc058a8  cjunosevolved:25.2R1.1-EVO   "/entrypoint.sh"  About a minute ago  Up About a
minute       cevo1
5ca6d865fd28  cjunosevolved:25.2R1.1-EVO   "/entrypoint.sh"  About a minute ago  Up About a
minute       cevo2
```

Configure cJunosEvolved on Docker

SUMMARY

Read this topic to understand the connections and configurations of the cJunosEvolved instances deployment on KVM

IN THIS SECTION

Connect to cJunosEvolved

You can SSH to the cJunosEvolved from the Ubuntu host as shown in this section using the admin credentials:

The Management IP address is what was specified in for “eth0_mgmt” under the networks section for the specific container in the Docker Compose YAML file.

Each cJunosEvolved is configured for SSH access using the admin credentials. The admin password is “admin@123”.

NOTE: You can set a root authentication plain-text password for cJunosEvolved if you prefer root access. This is done in the same way as for any JunosEvolved router. Also enable the SSH as root user if you prefer to login as root:

```
set system root-authentication plain-text-password <enter the password twice>

set system services ssh root-login allow
```

```
# ssh admin@<management IP address>
(admin@14.1.1.3) Password:
Last login: Thu Mar 20 21:06:48 2025 from 14.1.1.1
--- JUNOS cJunosEvolved-25.2R1.1-EVO Linux (none) 5.15.164-10.22.33.18-yocto-standard-
juniper-16971-g1c568856e6a0 #1 SMP PREEMPT Thu Feb 6 08:43:02 UTC 2025 x86_64 x86_64 x86_64 GNU/
Linux
admin@re0>
```

An alternative to ssh is to use the following command from the Host Server, to enter the CLI and configure the container. It is best to run this command after you have run “`docker logs -f cevo1` command and seen the container (called cevo1 in this example) has reached login prompt.

```
# docker exec -ti cevo1 cli

root@cevo1>
```

Configure cJunosEvolved

IN THIS SECTION

There are multiple ways to configure cJunosEvolved:

Manual Configuration

You can connect to the EVO VM within the cJunosEvolved container through either ssh or the Docker command mentioned in the “Connect to cJunosEvolved” section and proceed to configure the container.

Disk Configuration

Specify a hierarchical Junos Evolved CLI configuration through the “volumes” stanza of the Docker Compose YAML file. In the example below:

There needs to be a valid hierarchical configuration file, called juniper.conf in a directory on your Host Server for the specific container, such as the below directory, for example: **/root/cjunosevo/config/cevo1**

cJunosEvolved software expects the `juniper.conf` file to be placed in its `/home/evo/configdisk` directory in the Linux side of the container. See below:

```
services:
  cJunosEvo1:
    #[snip]
    privileged: true
    volumes:
      -'/root/cjunosevo/config/cevo1:/home/evo/configdisk'
```

Automatic Configuration

Another option is to have cJunosEvolved auto configured through the CPTX_AUTO_CONFIG environment variable being set to 1 as previously described in this manual.

NOTE: The autoconfigure option is a more minimal configuration tool than the Disk configuration. It is mutually exclusive with the Disk Configuration option and takes precedence over it.

Auto configuration does a startup configuration for cJunosEvolved based on extracting the information from the Docker Compose YAML file. Auto configuration can configure the following:

- **1.** The admin login credentials:
  - userid: admin
  - password: admin@123

2. To enable ssh run the `set system services ssh` command.
3. The default logging level is set as follows:

   ```
   set system syslog file interactive-commands interactive-commands any

   set system syslog file messages any notice

   set system syslog file messages authorization info
   ```

4. The management IP of cJunosEvolved based on the value specified for “eth0_managment”
5. The data interface “ipv4_address” values for interfaces starting from eth4 as specified in the Compose YAML file are configured in the startup configuration of the RE. The interface notation used in the CLI is automatically configured based on the environment variables indicating whether BX or BT, channelized or unchannelized notation should be used. See the WAN Interface tables in this manual for more details.

For example, in the case of a BX channelized cJunosEvolved being auto configured, if the Docker Compose YAML file has:

```
eth4:
  ipv4_address: 100.1.1.2
# and this in the “networks” section:
networks:
  eth4:
    name: eth4
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 100.1.1.0/24
```

Then run the `set interfaces et-0/0/0:0 unit 0 family inet address 100.1.1.2/24` command. This configuration will be set in the Routing Engine CLI.

In addition, depending on whether BX or BT, channelized or unchannelized, the corresponding general interface related configuration will be set in the Routing Engine CLI for all the data interfaces. The example below is for the first data interface (eth4) of a channelized BX:

```
set interfaces et-0/0/0 number-of-sub-ports 4

set interfaces et-0/0/0 speed 100g
```

NOTE: If the “CPTX_AUTO_CONFIG: 1” environmental variable is set in the Docker Compose YAML file and no IPv4 address is specified for a data interface in the same file, cJunosEvolved will assign an IP address based on what Docker provides it. For example, if a subnet IP is specified for the corresponding data interface, Docker will assign an IP address from that subnet, and this will be configured in the cJunosEvolved Routing Engine CLI for the corresponding interface. The recommendation is to provide the IP address for each interface in the services section of the YAML file and a matching subnet IP address for that network in the “networks” section of the YAML file when using CPTX_AUTO_CONFIG, so you have control over the IP addresses.

How to Stop, Start, and Restart cJunosEvolved

IN THIS SECTION

Stop

The cJunosEvolved containers in a Docker Compose file can be stopped simultaneously in a graceful manner so its disk is preserved for the next restart. Run the `stop` command from the host server:

```
# docker compose -f <docker-compose-filename>.yaml stop -t <secs>
```

The -t option asks Docker Compose to wait the specified number of seconds, before it sends a SIGTERM to the container. The value of the timeout to specify, depends on various factors including the size of logs and configuration stored in the container and the disk speed of the host server. In testing, a -t value of 240 secs has been sufficient and usually it is less than 60 secs. Please benchmark this value for your use case as follows:

- From the host server, start monitoring the EVO VM logs using the `docker logs -f <container-name>` command. This will print ongoing logs from EVO VM.
- Issue the `stop` command with a larger -t value as follows:

  ```
  # docker compose -f <docker-compose-filename>.yaml stop -t 240
  ```

- You will see the container logs as it is getting notified of the `stop` command and proceeds accordingly. For a successful shutdown, a message like below should be seen at the end of the logs:

  Fri Mar 28 22:40:14 UTC 2025: EVO VM shutdown successfully in [30secs]

Once cJunosEvolved is stopped, “docker container ls” will not show it but its networks are preserved for a future restart and can be seen as shown in the example below:

```
# docker container ls
CONTAINER ID   IMAGE                          COMMAND            CREATED      STATUS  PORTS
NAMES
171a361a6f86   cjunosevolved:25.2R1.1-EVO    "/entrypoint.sh"   3 days ago   Up      3 days
cevo2
740524cf95b3   cjunosevolved:25.2R1.1-EVO    "/entrypoint.sh"   3 days ago   Up      3 days
cevo1
# docker compose -f dual-bx-bt-chan-autocompose.yaml stop  -t 180
[+] Stopping 2/2
 ✓ Container cevo1
Stopped
        36.9s
 ✓ Container cevo2
Stopped
        36.9s
# docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
# docker network ls
NETWORK ID     NAME            DRIVER    SCOPE
785df5e996d5   bridge          bridge    local
6222dbea40c6   eth0_mgmt       bridge    local
fb5a95cd7236   eth1_reserved   bridge    local
cec52e294667   eth2_reserved   bridge    local
4f6d181ab5ec   eth3_reserved   bridge    local
3a8ccd8d44dc   eth4            bridge    local
4e701a843d95   eth5            bridge    local
6e3035f60692   eth6            bridge    local
bf4f5b2583aa   eth7            bridge    local
0f8822dda3ab   eth8            bridge    local
507f0ad3feaf   eth9            bridge    local
4af981ed503e   eth10           bridge    local
7d280f0f348a   eth11           bridge    local
2304e4ab7184   host            host      local
da25cbdc438c   none            null      local
```

Verify that both the containers are stopped successfully using the `docker logs` command:

```
# docker logs cevo1
[snip]
Tue Apr  1 00:00:24 UTC 2025: EVO VM  shutdown successfully in [30secs]
# docker logs cevo2
[snip]
Tue Apr  1 00:00:24 UTC 2025: EVO VM  shutdown successfully in [30secs]
```

Start

Stopped containers can be started again using the `docker compose start` command.

For a fresh installation, use the `docker compose up -d` command as previously shown in this document.

The following example illustrates the `docker compose start`.

```
# docker compose -f <docker-compose-file-name>
[+] Running 2/2
 ✓ Container cevo1
Started
         6.2s
 ✓ Container cevo2
Started
         6.2s
# docker container ls
CONTAINER ID   IMAGE                                        COMMAND               CREATED
STATUS     PORTS         NAMES
171a361a6f86   cjunosevolved:25.2R1.1-EVO   "/entrypoint.sh"      3 days ago      Up         54
seconds    cevo2
740524cf95b3   cjunosevolved:25.2R1.1-EVO   "/entrypoint.sh"      3 days ago      Up         54
seconds    cevo1

You  can monitor the console of EVOVM as it boots up via the following command. It should show
cJunosEvolved reach login prompt and shows in this case it took 66 seconds to do so.
#docker logs -f <container-name>

Tue Apr  1 00:47:41 UTC 2025: EVO VM boot Done in [66secs]
Tue Apr  1 00:47:41 UTC 2025: entrypoint.sh is done, enter wait loop..
```

```
re0 login:
```

Restart

You can restart running containers, using the `docker compose -f <docker-compose-file-name> restart -t 90` command.

Provide a “-t” you have benchmarked as previously mentioned in the stop section. A `restart` command is a `stop` and then a `start` command sequentially, so Docker Compose needs to be told how long to provide the container to stop gracefully to protect the container disk as mentioned in the Stop section above.

For example:

```
# docker compose -f <docker-compose-file-name>dual-bx-bt-chan-autocompose.yaml restart  -t 90
[+] Restarting 2/2
 ✓ Container cevo1
Started
        41.1s
 ✓ Container cevo2  Started
In this case, the container was stopped gracefully in 30 secs as shown in the “docker logs”
command. It was restarted as shown above.
Tue Apr  1 00:58:18 UTC 2025: EVO VM  shutdown successfully in [30secs]
```

CHAPTER 4

Verify and Troubleshoot cJunosEvolved on Docker

IN THIS CHAPTER

Verify and Troubleshoot cJunosEvolved on Docker

SUMMARY

Use this topic to verify your cJunosEvolved configurations and for any troubleshooting information.

IN THIS SECTION

Verify that cJunosEvolved Container is Running and Ready

1. The `docker container ls` command lists the containers running on your server. The status column should show cJunosEvolved as being UP.

```
# docker container ls
CONTAINER ID    IMAGE                        COMMAND            CREATED       STATUS
PORTS  NAMES

4eccebc058a8   cjunosevolved:25.2R1.1-EVO   "/entrypoint.sh"   6 days ago     Up 6
days         cevo1
5ca6d865fd28   cjunosevolved:25.2R1.1-EVO   "/entrypoint.sh"   6 days ago     Up 6
days         cevo2
```

2. You can use the command mentioned in this step to enter the CLI of cJunosEvolved. If the container is not yet completely up, a "not ready" message will print periodically on the host server console. You can also SSH to the management port IP.

   ```
   # docker exec -ti R1 cliroot@re0>
   ```

3. Run standard commands such as `show chassis fpc` and `show interfaces terse` in the CLI mode to verify if the FPC is online, and if interfaces are up.

```
user@host> show chassis fpc
                    Temp   CPU Utilization (%)   Memory Utilization (%)
Slot State            (C)    Total  Interrupt      DRAM (MB) Heap Buffer

0    Online          1min     5min                   15min
```

```
root@re0> show interfaces terse
Interface        Admin Link   Proto    Local         Remote
et-0/0/0:0       up           up
et-0/0/0:0.0     up           up       inet          210.1.1.2/24
                                       multiservice
et-0/0/0:1       up           up
et-0/0/0:1.0     up           up       inet          211.1.1.2/24
                                      multiservice
[snip]
```

Verify CPU Information

On the host server, use the `lscpu` command to display CPU information.

The output displays information such as the total number of CPUs, the number of cores per socket, and the number of CPU sockets.

For example, the following information is for an Ubuntu 22.04 LTS host server supporting a total of 56 CPUs (with hyperthreading enabled).

```
root@host-server# lscpu
    Architecture:          x86_64
    CPU op-mode(s):        32-bit, 64-bit
    Address sizes:         46 bits physical, 48 bits virtual
```

```
    Byte Order:            Little Endian
    CPU(s):                56
    On-line CPU(s) list:   0-55
    Vendor ID:             GenuineIntel
    Model name:            Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz
    CPU family:            6
    Model:                 79
    Thread(s) per core:    2
    Core(s) per socket:    14
    Socket(s):             2
[snip]
    Virtualization features:
        Virtualization:    VT-x
[snip]
```

Collect Log Files

IN THIS SECTION

Docker Logs

The `docker logs -f <container-name>` command provides cJunosEvolved logs starting with its bootup logs.

EVOVM System Logs

To view the system logs, login to the EVOVM as previously described. Then run the `root@re0 > show log ?` command to see the list of log files available.

For example, to view the EVO init logs, run the `root> show log evoinit.log command`

The log files can be viewed from the **/var/log** directory of the cJunosEvolved Routing Engine. These logs are the standard JunosEvolved log files that are also found on other Juniper Networks® products.

1. Use the `request system debug-info` command.

   This command places all the system traces in a file named **/var/tmp/debug_collector_<date_time>.tgz**

2. Run `scp` command on the host server to transfer the debug_collector files to the host server.

   Example: `scp admin@<management IP> :/var/tmp/debug_collector__<date_time>.tgz`

3. Use the `show system core-dumps` command to view the collected core files. You can transfer these core files to the host server for analysis through the management interface on the Routing Engine.

   The **/var/crash** directory of the cJunosEvolved stores all the core files. You can follow the standard procedures of the Junos OS to transfer the core files cJunosEvolved-BX to an external host.