Juniper Networks | Engineering Simplicity

cSRX Deployment Guide for Bare-Metal Linux Server

Published
2020-03-30

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

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 the Documentation

IN THIS SECTION

Use this guide to install the cSRX Container Firewall in a Linux bare-metal server environment that is running Ubuntu, Red Hat Enterprise Linux (RHEL), or CentOS. This guide also includes basic cSRX container configuration and management procedures.

After completing the installation, management, and basic configuration procedures covered in this guide, refer to the Junos OS documentation for information about further software configuration.

Documentation and Release Notes

To obtain the most current version of all Juniper Networks® technical documentation, see the product documentation page on the Juniper Networks website at https://www.juniper.net/documentation/.

If the information in the latest release notes differs from the information in the documentation, follow the product Release Notes.

Juniper Networks Books publishes books by Juniper Networks engineers and subject matter experts. These books go beyond the technical documentation to explore the nuances of network architecture, deployment, and administration. The current list can be viewed at https://www.juniper.net/books.

Documentation Conventions

Table 1 on page vi defines notice icons used in this guide.

Table 1: Notice Icons

Icon	Meaning	Description
	Informational note	Indicates important features or instructions.
	Caution	Indicates a situation that might result in loss of data or hardware damage.
	Warning	Alerts you to the risk of personal injury or death.
	Laser warning	Alerts you to the risk of personal injury from a laser.
	Tip	Indicates helpful information.
	Best practice	Alerts you to a recommended use or implementation.

Table 2 on page vi defines the text and syntax conventions used in this guide.

Table 2: Text and Syntax Conventions

Convention	Description	Examples
Bold text like this	Represents text that you type.	To enter configuration mode, type the **configure** command: user@host> **configure**
`Fixed-width text like this`	Represents output that appears on the terminal screen.	`user@host>` **show chassis alarms** `No alarms currently active`
Italic text like this	• Introduces or emphasizes important new terms. • Identifies guide names. • Identifies RFC and Internet draft titles.	• A policy *term* is a named structure that defines match conditions and actions. • *Junos OS CLI User Guide* • RFC 1997, *BGP Communities Attribute*

Table 2: Text and Syntax Conventions *(continued)*

<table>
<tr><th>Convention</th><th>Description</th><th>Examples</th></tr>
<tr><td>Italic text like this</td><td>Represents variables (options for which you substitute a value) in commands or configuration statements.</td><td>Configure the machine’s domain name:
[edit]
root@# set system domain-name domain-name</td></tr>
<tr><td>Text like this</td><td>Represents names of configuration statements, commands, files, and directories; configuration hierarchy levels; or labels on routing platform components.</td><td>• To configure a stub area, include the stub statement at the [edit protocols ospf area area-id] hierarchy level.
• The console port is labeled CONSOLE.</td></tr>
<tr><td>< > (angle brackets)</td><td>Encloses optional keywords or variables.</td><td>stub <default-metric <i>metric</i>>;</td></tr>
<tr><td>| (pipe symbol)</td><td>Indicates a choice between the mutually exclusive keywords or variables on either side of the symbol. The set of choices is often enclosed in parentheses for clarity.</td><td>broadcast | multicast
(string1 | string2 | string3)</td></tr>
<tr><td># (pound sign)</td><td>Indicates a comment specified on the same line as the configuration statement to which it applies.</td><td>rsvp { # Required for dynamic MPLS only</td></tr>
<tr><td>[ ] (square brackets)</td><td>Encloses a variable for which you can substitute one or more values.</td><td>community name members [community-ids]</td></tr>
<tr><td>Indention and braces ({ })</td><td>Identifies a level in the configuration hierarchy.</td><td rowspan="2">[edit]
routing-options {
  static {
    route default {
      nexthop address;
      retain;
    }
  }
}</td></tr>
<tr><td>; (semicolon)</td><td>Identifies a leaf statement at a configuration hierarchy level.</td></tr>
<tr><td colspan="3">GUI Conventions</td></tr>
</table>

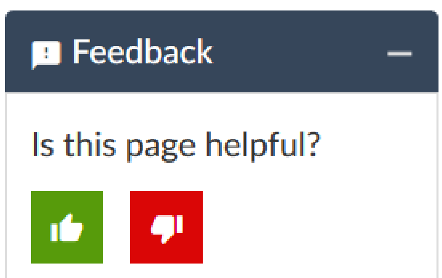
Table 2: Text and Syntax Conventions *(continued)*

Convention	Description	Examples
Bold text like this	Represents graphical user interface (GUI) items you click or select.	• In the Logical Interfaces box, select **All Interfaces**. • To cancel the configuration, click **Cancel**.
> (bold right angle bracket)	Separates levels in a hierarchy of menu selections.	In the configuration editor hierarchy, select **Protocols>Ospf**.

Documentation Feedback

We encourage you to provide feedback so that we can improve our documentation. You can use either of the following methods:

- Online feedback system—Click TechLibrary Feedback, on the lower right of any page on the Juniper Networks TechLibrary site, and do one of the following:



  - Click the thumbs-up icon if the information on the page was helpful to you.
  - Click the thumbs-down icon if the information on the page was not helpful to you or if you have suggestions for improvement, and use the pop-up form to provide feedback.

- E-mail—Send your comments to techpubs-comments@juniper.net. Include the document or topic name, URL or page number, and software version (if applicable).

Requesting Technical Support

Technical product support is available through the Juniper Networks Technical Assistance Center (JTAC). If you are a customer with an active Juniper Care or Partner Support Services support contract, or are covered under warranty, and need post-sales technical support, you can access our tools and resources online or open a case with JTAC.

- JTAC policies—For a complete understanding of our JTAC procedures and policies, review the *JTAC User Guide* located at https://www.juniper.net/us/en/local/pdf/resource-guides/7100059-en.pdf.
- Product warranties—For product warranty information, visit https://www.juniper.net/support/warranty/.
- JTAC hours of operation—The JTAC centers have resources available 24 hours a day, 7 days a week, 365 days a year.

Self-Help Online Tools and Resources

For quick and easy problem resolution, Juniper Networks has designed an online self-service portal called the Customer Support Center (CSC) that provides you with the following features:

- Find CSC offerings: https://www.juniper.net/customers/support/
- Search for known bugs: https://prsearch.juniper.net/
- Find product documentation: https://www.juniper.net/documentation/
- Find solutions and answer questions using our Knowledge Base: https://kb.juniper.net/
- Download the latest versions of software and review release notes: https://www.juniper.net/customers/csc/software/
- Search technical bulletins for relevant hardware and software notifications: https://kb.juniper.net/InfoCenter/
- Join and participate in the Juniper Networks Community Forum: https://www.juniper.net/company/communities/
- Create a service request online: https://myjuniper.juniper.net

To verify service entitlement by product serial number, use our Serial Number Entitlement (SNE) Tool: https://entitlementsearch.juniper.net/entitlementsearch/

Creating a Service Request with JTAC

You can create a service request with JTAC on the Web or by telephone.

- Visit https://myjuniper.juniper.net.
- Call 1-888-314-JTAC (1-888-314-5822 toll-free in the USA, Canada, and Mexico).

For international or direct-dial options in countries without toll-free numbers, see https://support.juniper.net/support/requesting-support/.

1

CHAPTER

Overview

Understanding cSRX with a Bare-Metal Linux Server

IN THIS SECTION

The cSRX Container Firewall is a containerized version of the SRX Series Services Gateway with a low memory footprint. cSRX provides advanced security services, including content security, AppSecure, and unified threat management in a container form factor. By using a Docker container in a bare-metal Linux server, the cSRX can substantially reduce overhead because each container shares the Linux host's OS kernel. Regardless of how many containers a Linux server hosts, only one OS instance is in use. And because of the containers' lightweight quality, a server can host many more container instances than it can virtual machines (VMs), yielding tremendous improvements in utilization. With its small footprint and Docker as a container management system, the cSRX Container Firewall enables agile, high-density security service deployment.

This section includes the following topics:

cSRX Overview

The cSRX Container Firewall runs as a single container on a Linux bare-metal server. It uses a Linux bare-metal server as the hosting platform for the Docker container environment. The cSRX container packages all of the dependent processes (daemons) and libraries to support the different Linux host distribution methods (Ubuntu, Red Hat Enterprise Linux, or CentOS). You use standard Docker commands to manage the cSRX container. cSRX is built on the Junos operating system (Junos OS) and delivers networking and security features similar to those available on the software releases for the SRX Series.

When the cSRX container runs, there are several daemons inside the Docker container that launch automatically when the cSRX becomes active. Some daemons support Linux features, providing the same service as if they are running on a Linux host (for example, sshd, rsyslogd, monit, and so on). Other daemons are compiled and ported from Junos OS to perform configuration and control jobs for security service (for example, MGD, NSD, UTM, IDP, AppID, and so on). srxpfe is the data-plane daemon that receives and sends packets from the revenue ports of a cSRX container. The cSRX uses srxpfe for Layer 2 through 3 forwarding functions (secure-wire forwarding or static routing forwarding) as well as for Layer 4 through 7 network security services.

The cSRX Container Firewall enables advanced security at the network edge in a multitenant virtualized environment. cSRX provides Layer 4 through 7 advanced security features such as firewall, IPS, and AppSecure. The cSRX container also provides an additional interface to manage the cSRX. When cSRX is operating in Layer 2 secure wire mode, incoming Layer 2 frames from one interface go through Layer 4 through 7 processing based on the configured cSRX services. cSRX then sends the frames out of the other interface.

Launch the cSRX instance in secure-wire mode using the following command:

root@csrx-ubuntu3:~/csrx# docker run -d --privileged --network=mgt_bridge -e CSRX_FORWARD_MODE="wire" --name=<csrx-container-name> <csrx-image-name>

> **NOTE:** As part of your Docker container configuration, you must connect the cSRX container to three virtual networks: one virtual network for out-of-band management sessions, the other two virtual networks to receive and transmit data traffic. See "Installing cSRX in a Bare-Metal Linux Server" on page 31.

Figure 1 on page 13 illustrates the cSRX operating in secure-wire mode. It is an example of how a cSRX container is bridged with an external network. In this illustration, cSRX eth1 is bridged with host physical NIC eth1 and cSRX eth2 is bridged with host physical NIC eth2.

Figure 1: cSRX in Secure-Wire Mode

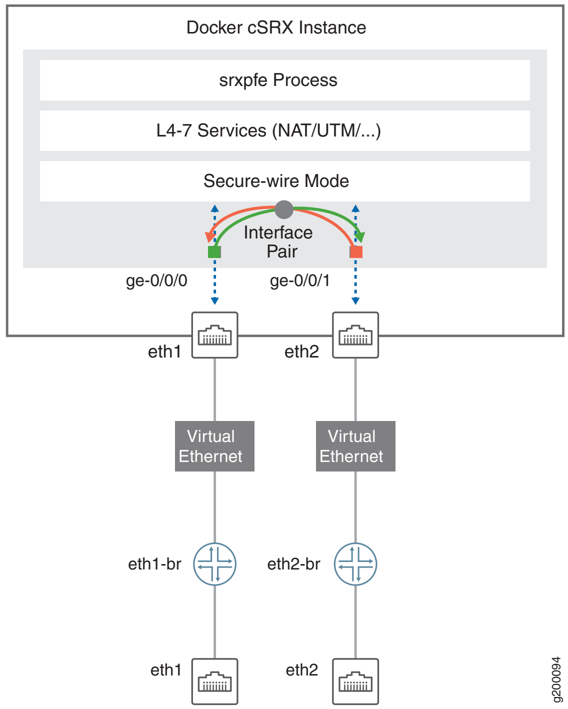
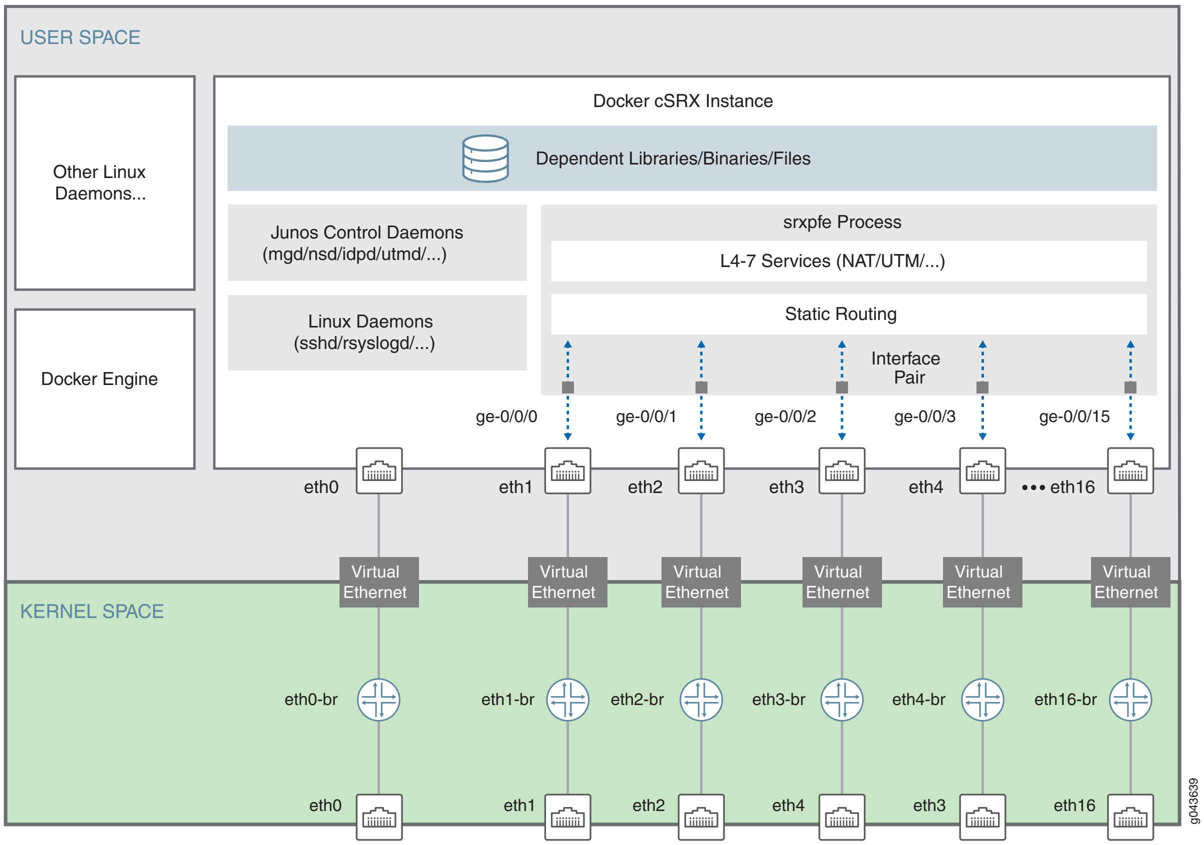


Figure 2 on page 14 illustrates the cSRX operating in routing mode.

Figure 2: cSRX Container in Routing Mode



Starting in Junos OS Release 19.2R1, in routing mode, the default number of interfaces supported are three and maximum of 17 interfaces (1 management and 16 data interfaces).

Prior to Junos OS Release 19.2R1, in routing mode, eth0 was mapped as out of band management interface, eth1 as ge-0/0/1, and eth2 as ge-0/0/0.

Starting in Junos OS Release 19.2R1, in routing mode, with this increase in the number of supported interfaces, the mapping of ge interfaces are reordered as:

- eth0 - out of band management interface
- eth1 - ge-0/0/0
- eth2 - ge-0/0/1
- eth3 - ge-0/0/2
- eth4 - ge-0/0/3 and so on

cSRX Benefits and Uses

The cSRX Container Firewall enables you to quickly introduce new firewall services, deliver customized services to customers, and scale security services based on dynamic needs. The cSRX container differs from VMs in several important ways. It runs with no guest OS overhead, has a notably smaller footprint, and is easier to migrate or download. The cSRX container uses less memory, and its spin-up time measures in subseconds—all leading to higher density at a lower cost. The boot time is reduced from several minutes with a VM-based environment to less than a few seconds for the cSRX container. The cSRX is ideal for public, private, and hybrid cloud environments.

Some of the key benefits of cSRX in a containerized private or public cloud multitenant environment include:

- Stateful firewall protection at the tenant edge.
- Faster deployment of containerized firewall services into new sites.
- With a small footprint and minimum resource reservation requirements, the cSRX can easily scale to keep up with customers' peak demand.
- Provides significantly higher density without requiring resource reservation on the host than what is offered by VM-based firewall solutions.
- Flexibility to run on a bare-metal Linux server or Juniper Networks Contrail.
  - In the Contrail Networking cloud platform, cSRX can be used to provide differentiated Layer 4 through 7 security services for multiple tenants as part of a service chain.
  - With the Contrail orchestrator, cSRX can be deployed as a large scale security service.
- Application security features (including IPS and AppSecure).
- UTM content security features (including antispam, Sophos Antivirus, web filtering, and content filtering).
- Authentication and integrated user firewall features.

> **NOTE:** While the security services features between cSRX and vSRX are similar, there are scenarios in which each product is the optimal option in your environment. For example, the cSRX does not support routing instances and protocols, switching features, MPLS LSPs and MPLS applications, chassis cluster, and software upgrade features. For environments that require routing or switching, a vSRX VM provides the best feature set. For environments focused on security services in a Docker containerized deployment, cSRX is a better fit.
>
> See "Junos OS Features Supported on cSRX" on page 18 for a summary of the feature categories supported on cSRX, and also for a summary of features not supported on cSRX.

You can deploy the cSRX Container Firewall in the following scenarios:

- Cloud CPE–For service providers (SPs) and managed security service providers (MSSPs) where there is a large subscriber base of branch offices or residential subscribers. MSSPs can offer differentiated services to individual subscribers.
- Contrail microsegmentation–Within a Contrail environment running mixed workloads of VMs and containers, cSRX can provide security for Layer 4 through 7 traffic, managed by Security Director.
- Private clouds–cSRX can provide security services in a private cloud running containerized workloads and can include Contrail integration.

Docker Overview

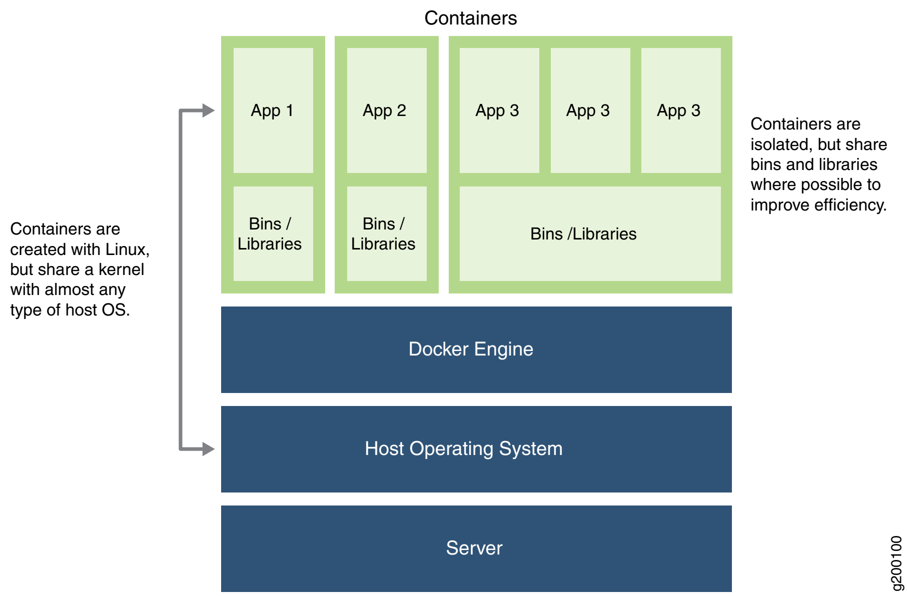
Docker is an open source software platform that simplifies the creation, management, and teardown of a virtual container that can run on any Linux server. A Docker container is an open source software development platform, with its main benefit being to package applications in “containers” to allow them to be portable among any system running the Linux operating system (OS). A container provides an OS-level virtualization approach for an application and associated dependencies that allow the application to run on a specific platform. Containers are not VMs, rather they are isolated virtual environments with dedicated CPU, memory, I/O, and networking.

A container image is a lightweight, standalone, executable package of a piece of software that includes everything required to run it: code, runtime, system tools, system libraries, settings, and so on. Because containers include all dependencies for an application, multiple containers with conflicting dependencies can run on the same Linux distribution. Containers use the host OS Linux kernel features, such as groups and namespace isolation, to allow multiple containers to run in isolation on the same Linux host OS. An application in a container can have a small memory footprint because the container does not require a guest OS, which is required with VMs, because it shares the kernel of its Linux host’s OS.

Containers have a high spin-up speed and can take much less time to boot up as compared to VMs. This enables you to install, run, and upgrade applications quickly and efficiently.

Figure 3 on page 17 provides an overview of a typical Docker container environment.

Figure 3: Docker Container Environment



cSRX Scale-Up Performance

You can scale the performance and capacity of a cSRX container by increasing the allocated amount of virtual memory or the number of flow sessions. Table 3 on page 17 shows the cSRX scale-up performance applied to a cSRX container based on its supported sizes: small, medium, and large. The default size for a cSRX container is large.

NOTE: See "Changing the Size of a cSRX Container" on page 44 for the procedure on how to scale the performance and capacity of a cSRX container by changing the container size.

Table 3: cSRX Scale Up Performance

cSRX Size	Physical Memory Overhead	Number of Flow Sessions	Release Introduced
Small	256M	8K	Junos OS Release 18.1R1
Medium	1G	64K	
Large	4G	512K	

RELATED DOCUMENTATION

- Docker Overview
- What is Docker?
- What is a Container?
- Get Started With Docker

Junos OS Features Supported on cSRX

IN THIS SECTION

cSRX provides Layer 4 through 7 secure services in a containerized environment.

This section presents an overview of the Junos OS features on cSRX.

Supported SRX Series Features on cSRX

Table 4 on page 18 provides a high-level summary of the feature categories supported on cSRX and any feature considerations.

To determine the Junos OS features supported on cSRX, use the Juniper Networks Feature Explorer, a Web-based application that helps you to explore and compare Junos OS feature information to find the right software release and hardware platform for your network. See Feature Explorer.

Table 4: SRX Series Features Supported on cSRX

Feature	Considerations
Application Firewall (AppFW)	Application Firewall Overview
Application Identification (AppID)	Understanding Application Identification Techniques
Application Tracking (AppTrack)	Understanding AppTrack

Table 4: SRX Series Features Supported on cSRX *(continued)*

Feature	Considerations
Basic firewall policy	Understanding Security Basics
Brute force attack mitigation	
Central management	CLI only. No J-Web support.
DDoS protection	DoS Attack Overview
DoS protection	DoS Attack Overview
Interfaces	A cSRX container supports 17 interfaces: • 1 Out-of-band management Interface (eth0) • 16 In-band interfaces (ge-0/0/0 to ge-0/0/15). Network Interfaces
Intrusion Detection and Prevention (IDP)	For SRX Series IPS configuration details, see: Understanding Intrusion Detection and Prevention for SRX Series
IPv4 and IPv6	Understanding IPv4 Addressing Understanding IPv6 Address Space
Jumbo frames	Understanding Jumbo Frames Support for Ethernet Interfaces
Malformed packet protection	
Network Address Translation (NAT)	Includes support for all NAT functionality on the cSRX platform, such as: • Source NAT • Destination NAT • Static NAT • Persistent NAT and NAT64 • NAT hairpinning • NAT for multicast flows For SRX Series NAT configuration details, see: Introduction to NAT

Table 4: SRX Series Features Supported on cSRX *(continued)*

Feature	Considerations
Routing	Basic Layer 3 forwarding with VLANs. Layer 2 through 3 forwarding functions: secure-wire forwarding or static routing forwarding
SYN cookie protection	Understanding SYN Cookie Protection
System Logs and Real-Time Logs	Starting in Junos OS Release 20.1R1, you can monitor traffic using system logs and RTlogs.
User Firewall	Includes support for all user firewall functionality on the cSRX platform, such as: • Policy enforcement with matching source identity criteria • Logging with source identity information • Integrated user firewall with active directory • Local authentication For SRX Series user firewall configuration details, see: Overview of Integrated User Firewall
Unified Threat Management (UTM)	Includes support for all UTM functionality on the cSRX platform, such as: • Antispam • Sophos Antivirus • Web filtering • Content filtering For SRX Series UTM configuration details, see: Unified Threat Management Overview For SRX Series UTM antispam configuration details, see: Antispam Filtering Overview
Zones and zone-based IP spoofing	Understanding IP Spoofing

SRX Series Features Not Supported on cSRX

Table 5 on page 21 lists SRX Series features that are not applicable in a containerized environment, that are not currently supported, or that have qualified support on cSRX.

Table 5: SRX Series Features Not Supported on cSRX

	SRX Series Feature
Application Layer Gateways	
	Avaya H.323
Authentication with IC Series Devices	
	Layer 2 enforcement in UAC deployments **NOTE:** UAC-IDP and UAC-UTM also are not supported.
Class of Service	
	High-priority queue on SPC
	Tunnels
Data Plane Security Log Messages (Stream Mode)	
	TLS protocol
Diagnostics Tools	
	Flow monitoring cflowd version 9
	Ping Ethernet (CFM)
	Traceroute Ethernet (CFM)
DNS Proxy	
	Dynamic DNS
Ethernet Link Aggregation	
	LACP in standalone or chassis cluster mode
	Layer 3 LAG on routed ports

Table 5: SRX Series Features Not Supported on cSRX *(continued)*

	SRX Series Feature
	Static LAG in standalone or chassis cluster mode
Ethernet Link Fault Management	
	Physical interface (encapsulations)
	ethernet-ccc **ethernet-tcc**
	extended-vlan-ccc **extended-vlan-tcc**
	Interface family
	ccc, tcc
	ethernet-switching
Flow-Based and Packet-Based Processing	
	End-to-end packet debugging
	Network processor bundling
	Services offloading
Interfaces	
	Aggregated Ethernet interface
	IEEE 802.1X dynamic VLAN assignment
	IEEE 802.1X MAC bypass
	IEEE 802.1X port-based authentication control with multisupplicant support
	Interleaving using MLFR
	PoE
	PPP interface

Table 5: SRX Series Features Not Supported on cSRX *(continued)*

	SRX Series Feature
	PPPoE-based radio-to-router protocol
	PPPoE interface
	Promiscuous mode on interfaces
IP Security and VPNs	
	Acadia - Clientless VPN
	DVPN
	Hardware IPsec (bulk crypto) Cavium/RMI
	IPsec tunnel termination in routing instances
	Multicast for AutoVPN
	Suite B implementation for IPsec VPN
IPv6 Support	
	DS-Lite concentrator (also known as AFTR)
	DS-Lite initiator (also known as B4)
Log File Formats for System (Control Plane) Logs	
	Binary format (binary)
	WELF
Miscellaneous	
	AppQoS
	Chassis cluster
	GPRS
	Hardware acceleration

Table 5: SRX Series Features Not Supported on cSRX *(continued)*

	SRX Series Feature
	High availability
	J-Web
	Logical systems
	MPLS
	Outbound SSH
	Remote instance access
	RESTCONF
	Sky ATP
	SNMP
	Spotlight Secure integration
	USB modem
	Wireless LAN
MPLS	
	CCC and TCC
	Layer 2 VPNs for Ethernet connections
Network Address Translation	
	Maximize persistent NAT bindings
Packet Capture	
	Packet capture **NOTE:** Only supported on physical interfaces and tunnel interfaces, such as *gr*, *ip*, and *st0*. Packet capture is not supported on a redundant Ethernet interface (*reth*).

Table 5: SRX Series Features Not Supported on cSRX *(continued)*

	SRX Series Feature
Routing	
	BGP extensions for IPv6
	BGP Flowspec
	BGP route reflector
	Bidirectional Forwarding Detection (BFD) for BGP
	CRTP
Switching	
	Layer 3 Q-in-Q VLAN tagging
Unsupported System Logs and Real-Time log functions	cSRX does not support all the log functions supported on other SRX devices or vSRX instances due to limited CPU power and disk capacity. Unsupported system logs and real-time log functions on cSRX are: • The binary log • On box logs (the LLMD daemon is not ported.) • On box reports (the LLMD daemon is not ported.) • TLS is not supported for sending stream mode security log to remote log server. • LSYS and Tenant related functions.
Transparent Mode	
	UTM
Unified Threat Management	
	Express AV
	Kaspersky AV
Upgrading and Rebooting	
	Autorecovery

Table 5: SRX Series Features Not Supported on cSRX *(continued)*

	SRX Series Feature
	Boot instance configuration
	Boot instance recovery
	Dual-root partitioning
	OS rollback
User Interfaces	
	NSM
	SRC application
	Junos Space Virtual Director
Application Security	
	SSL proxy

2

CHAPTER

Installing cSRX

Requirements for Deploying cSRX on a Bare-Metal Linux Server

IN THIS SECTION

This section presents an overview of requirements for deploying a cSRX container on a bare-metal Linux server:

Host Requirements

Table 6 on page 28 lists the Linux host requirement specifications for deploying a cSRX container on a bare-metal Linux server.

> **NOTE:** The cSRX can run either on a physical server or virtual machine. For scalability and availability reasons, we recommended using a physical server to deploy the cSRX container.

Table 6: Host Requirement Specifications for cSRX

Component	Specification	Release Introduced
Linux OS support	CentOS 6.5 or later	Junos OS Release 18.1R1
	Red Hat Enterprise Linux (RHEL) 7.0 or later	
	Ubuntu 14.04.2 or later	
Docker Engine	Docker Engine 1.9 or later installed on a Linux host	

Table 6: Host Requirement Specifications for cSRX *(continued)*

Component	Specification	Release Introduced
Contrail Cloud Platform	Contrail 3.2 with OpenStack Liberty or OpenStack Mitaka	
vCPUs	2 CPU cores	
Memory	8 GB	
Disk space	40 GB hard drive	
Host processor type	x86_64 multicore CPU	
Network interface	1 Ethernet port (minimum)	

cSRX Basic Configuration Settings

The cSRX container requires the following basic configuration settings:

- Interfaces must be assigned IP addresses.
- Policies must be configured between zones to permit or deny traffic.

Interface Naming and Mapping

A cSRX container supports 17 interfaces:

- 1 Out-of-band management Interface (eth0)
- 16 In-band interfaces (ge-0/0/0 to ge-0/0/15).

Table 7 on page 29 lists the cSRX interface assignments with Docker.

Table 7: cSRX Interface Assignment

Interface Number	cSRX Interfaces	Docker Interfaces
1	eth0	eth0

Table 7: cSRX Interface Assignment *(continued)*

Interface Number	cSRX Interfaces	Docker Interfaces
2	ge-0/0/0	eth1
3	ge-0/0/1	eth2
4	ge-0/0/2	eth3
6	ge-0/0/4	eth5
7	ge-0/0/5	eth6
8	ge-0/0/6	eth7
9	ge-0/0/7	eth8
10	ge-0/0/8	eth9
11	ge-0/0/9	eth10
12	ge-0/0/10	eth11
13	ge-0/0/11	eth12
14	ge-0/0/12	eth13
15	ge-0/0/13	eth14
16	ge-0/0/14	eth15
17	ge-0/0/15	eth16

Installing cSRX in a Bare-Metal Linux Server

IN THIS SECTION

This section outlines the steps to install the cSRX container in a Linux bare-metal server environment that is running Ubuntu, Red Hat Enterprise Linux (RHEL) , or CentOS. The cSRX container is packaged in a Docker image and runs in the Docker Engine on the Linux host.

This section includes the following topics:

Before You Deploy

Before you deploy the cSRX Container Firewall as an advanced security service in a Linux container environment, ensure that you:

- Review "Requirements for Deploying cSRX on a Bare-Metal Linux Server" on page 28 to verify the system software requirement specifications for the Linux server required to deploy the cSRX container.
- Install and configure Docker on your Linux host platform to implement the Linux container environment. Docker installation requirements vary based on the platform and the host OS (Ubuntu, Red Hat Enterprise Linux (RHEL), or CentOS). Install Docker. You can also use the script at: https://get.docker.com/ to install docker easily. You need to execute this script on shell.

  For docker installation instructions on the different supported Linux host operating systems, see:

  - **Centos/Redhat**—https://docs.docker.com/install/linux/docker-ce/centos/
  - **Debian**—https://docs.docker.com/install/linux/docker-ce/debian/
  - **Fedora**—https://docs.docker.com/install/linux/docker-ce/fedora/
  - **Ubuntu**—https://docs.docker.com/install/linux/docker-ce/ubuntu/
  - **Centos/Redhat**—https://docs.docker.com/install/linux/docker-ce/centos/
  - **Centos/Redhat**—https://docs.docker.com/install/linux/docker-ce/centos/

Confirming Docker Installation

Before you load the cSRX image, confirm that Docker is properly installed on the Linux host and that the Docker Engine is running.

To confirm Docker installation:

1. Confirm that Docker is installed and running on the Linux server by using the **service docker status** command.

   root@csrx-ubuntu3:~# service docker status

   docker start/running, process 701

   You should also be able to run **docker run hello-world** and see a similar response.

   root@csrx-ubuntu3:~# docker run hello-world

   ```
   Hello from Docker!
    This message shows that your installation appears to be working correctly.
   ```

   - If Docker is not installed, see Install Docker for installation instructions.
   - If Docker is not running, see Configure and troubleshoot the Docker daemon.

2. Verify the installed Docker Engine version by using the **docker version** command.

   > **NOTE:** Ensure that Docker version 1.9.0 or later is installed on the Linux host.

   root@csrx-ubuntu3:~# docker version

   Client:

   Docker version 17.05.0-ce-rc1, build 2878a85

   API Version: 1.30

   Go version: go1.8.3

   Git commit: 02cid87

   Built: Fri June 23 21:17:13 2017

OS/Arch: linux/amd64

Server:

Docker version 17.05.0-ce-rc1, build 2878a85

API Version: 1.30 (minimum version 1.12)

Go version: go1.8.3

Git commit: 02cid87

Built: Fri June 23 21:17:13 2017

OS/Arch: linux/amd64

Experimental: False

Loading the cSRX Image

The cSRX image is available as a cSRX Docker file from the Juniper Internal Docker registry.

Once the Docker Engine has been installed on the host, perform the following to download and start using the cSRX image:

1. Login to the Juniper Internal Docker registry using the login name and password that you received as part of the sales fulfillment process when ordering cSRX.

   root@csrx-ubuntu3:~csrx# docker login hub.juniper.net -u < username> -p < password>

2. To browse the existing images from the Juniper Internal Docker registry for a cSRX image, use the **curl** CLI command:

   root@csrx-ubuntu3:~csrx# curl -u < < username>> -X GET https://hub.juniper.net/v2/security/csrx/tags/list

   Enter host password for user '<<username>>:

   { "name":"security/csrx","tags":["18.1R1.9","18.2R1.9"]}

   root@csrx-ubuntu3:~csrx#

NOTE: To browse the existing images from the Juniper Internal Docker registry by using a Web browser instead of using the **curl** CLI command, you can launch a Web browser with https://hub.juniper.net/v2/security/csrx/tags/list. Use the login name and password that you received as part of the sales fulfillment process.

3. Pull the cSRX image from the Juniper Internal Docker registry.

   root@csrx-ubuntu3:~csrx# docker pull hub.juniper.net/security/csrx:< version>

   For example, to pull cSRX image version 18.2R1.9:

   root@csrx-ubuntu3:~csrx# docker pull hub.juniper.net/security/csrx:18.2R1.9

4. After the cSRX image loads, confirm that it is listed in the repository of Docker images.

   root@csrx-ubuntu3:~/csrx# docker images

   REPOSITORY TAG IMAGE ID CREATED SIZE

   csrx 18.1R1.0 6fcdebe006e4 Less than a second ago 585MB

   root@csrx-ubuntu3:~/csrx#

Creating the Linux Bridge Network for the cSRX

A Linux bridge is a virtual switch implemented as a kernel module. This Linux bridge is used within a Linux host to emulate a hardware bridge. Docker allows you to create a Linux bridge network and connect the cSRX container to this network to implement management and data processing sessions. The interfaces are created with the Linux VETH driver and are used to communicate with the Linux kernel.

This procedure describes how to create a three-bridge network for the cSRX container that includes: mgt_bridge (eth0), left_bridge (eth1), and right_bridge (eth2). The mgt_bridge is used by the cSRX for out-of-band management to accept management sessions and traffic, and the left_bridge and right_bridge are both used by the cSRX as the revenue ports to process in-band data traffic.

The trusted and untrusted interfaces required by a cSRX connector are connected to this Linux bridge on eth1 and eth2. In this example, the untrusted interface (eth1) is connected to Linux bridge Br1 and the trusted interface (eth2) is connected to Linux bridge Br2. By default, the cSRX boots in Layer 3 mode where it performs forwarding between the trusted and untrusted interfaces.

> **NOTE:** Docker automatically connects the management interface (eth0) to the Linux bridge and assigns an IP address. Interfaces eth1 and eth2 are for the inband traffic and you must assign a trusted and untrusted interface to the two revenue interfaces. cSRX must be bound with the Linux bridge to pass traffic.

To create a three-bridge network for a cSRX in the Linux host:

1. Create the management bridge in the network.

   root@csrx-ubuntu3::~/csrx# docker network create --driver bridge mgt_bridge

   3228844986eae1d1a8d367b34b54b31b130842be072b9dcdf7da3601c95b7130

2. Create the left bridge in the network (untrusted interface (eth1)).

   root@csrx-ubuntu3::~/csrx# docker network create --driver bridge left_bridge

   f1324b0a9072c55ababbcc51d83c83658084b67513811e13829172cccbc08e5d

3. Create the right bridge in the network (trusted interface (eth2)).

   root@csrx-ubuntu3::~/csrx# docker network create --driver bridge right_bridge

   196bd039f7c2401df4c117ea684114548a3df0b9d406cf3cf8f17338fab96774

RELATED DOCUMENTATION

Docker commands

Launching the cSRX Container

You are now ready to launch the cSRX container that is running in Docker on the Linux bare-metal server. When you start the cSRX image, you have a running container of the image. You can stop and restart the cSRX container (see “Managing cSRX Containers” on page 53), and the container will retain all settings and file system changes unless those changes are explicitly deleted. However, the cSRX will lose anything in memory and all processes will be restarted.

You have a series of cSRX environment variables that enable you to modify operating characteristics of the cSRX container when it is launched. You can modify:

- Initial root account password to log in to the cSRX container using SSH
- Traffic forwarding mode (static route or secure-wire)
- cSRX container size (small, medium, or large)
- Packet I/O driver (polled or interrupt)
- CPU affinity for cSRX control and data daemons
- Address Resolution Protocol (ARP) and Neighbor Discovery Protocol (NDP) entry timeout values
- Number of interfaces you need to add to container. Default is 3 and maximum is 17 (which means 1 management interfaces and 16 data interfaces).

NOTE: Specification of an environment variable is not mandatory when launching the cSRX container; most environment variables have a default value as shown in “cSRX Environment Variables Overview” on page 41. You can launch the cSRX using the default environment variable settings.

To launch the cSRX container:

1. Use the **docker run** command to launch the cSRX container. You include the **mgt_bridge** management bridge to connect the cSRX to a network. If you intend to log into the cSRX container using SSH, you must specify an initial root password when launching the cSRX.

   root@csrx-ubuntu3:~/csrx# docker run -d --privileged --network=mgt_bridge -e CSRX_ROOT_PASSWORD=<password> --name=<csrx-container-name> hub.juniper.net/security/**<csrx-image-name>**

   For example, to launch **csrx2** using cSRX software image **csrx:18.21R1.9** and root password **root123** enter:

   root@csrx-ubuntu3:~/csrx# docker run -d --privileged --network=mgt_bridge -e CSRX_ROOT_PASSWORD=root123 —name=csrx2 hub.juniper.net/security/csrx:18.2R1.9

   > **NOTE:** You must include the **--privileged** flag in the **docker run** command to enable the cSRX container to run in privileged mode.

2. Connect the left and right bridges to the Docker network.

   root@csrx-ubuntu3:~/csrx# docker network connect left_bridge csrx2

   root@csrx-ubuntu3:~/csrx#

   root@csrx-ubuntu3:~/csrx# docker network connect right_bridge csrx2

   root@csrx-ubuntu3:~/csrx#

3. Confirm that the three-bridge network has been created for the cSRX container.

   root@csrx-ubuntu3:~/csrx# docker network ls

   NETWORK ID NAME DRIVER SCOPE

   80bea9207560 bridge bridge local

   619da6736359 host host local

   112ab00aab1a left_bridge bridge local

   1484998f41bb mgt_bridge bridge local

   daf7a5a477bd none null local

   e409a4f54237 right_bridge bridge local

4. Confirm that the cSRX container is listed as a running Docker container.

root@csrx-ubuntu3:~/csrx# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

35e33e8aa4af csrx "/etc/rc.local init" 7 minutes ago Up 7 minutes 22/tcp, 830/tcp csrx2

5. Confirm that the cSRX container is up and running. You should see the expected Junos OS processes, such as nsd, srxpfe, and mgd.

root@csrx-ubuntu3:~/csrx# docker top csrx2

```
UID                 PID                 PPID                C
  STIME               TTY                 TIME                CMD
root                318                 305                 0
  09:13               pts/1               00:00:00            bash
root                27423               27407               0
  Mar30               pts/0               00:00:00            /bin/bash -e
/etc/rc.local init
root                27867               27423               0
  Mar30              ?                   00:08:16           /usr/sbin/rsyslogd
 -M/usr/lib/rsyslog
root                27880               27423               0
  Mar30               ?                   00:00:00            /usr/sbin/sshd
root                27882               27423               0
  Mar30              ?                   00:00:00           /usr/sbin/nstraced
root                27907               27423               0
  Mar30               ?                   00:00:08            /usr/sbin/mgd
root                27963               27423               0
  Mar30              pts/0               00:34:50           /usr/bin/monit -I
root                27979               27423               0
  Mar30               ?                   00:01:10            /usr/sbin/nsd
root                27989               27423               0
  Mar30               ?                   00:00:02            /usr/sbin/appidd
 -N
root                28023               27423               0
  Mar30              ?                   00:00:21           /usr/sbin/idpd -N
root                28040               27423               0
  Mar30              ?                   00:09:21           /usr/sbin/wmic -N
root                28048               27423               0
  Mar30              ?                   00:52:50           /usr/sbin/useridd
 -N
root                28126               27423               2
  Mar30               ?                   1-05:21:47          /usr/sbin/srxpfe
 -a -d
```

```
root                    28186                   27423                   0
  Mar30                   ?                       00:01:37                /usr/sbin/utmd -N
root                    28348                   27423                   0
  Mar30                   ?                        00:02:44                /usr/sbin/kmd
```

6. Confirm the IP address of the management interface of the cSRX container.

   root@csrx-ubuntu3:~/csrx# docker inspect csrx2 | grep IPAddress

```
              "SecondaryIPAddresses": null,
              "IPAddress": "",
                      "IPAddress": "172.19.0.2",
                      "IPAddress": "172.18.0.2",
                      "IPAddress": "172.20.0.2",
```

RELATED DOCUMENTATION

Docker commands

3
CHAPTER

Managing cSRX Containers

cSRX Environment Variables Overview

Docker allows you to store data such as configuration settings as environment variables. At runtime, the environment variables are exposed to the application inside the container. You can set any number of parameters to take effect when the cSRX image launches. You set an environment variable by specifying the **docker run -e VARIABLE=VALUE ...** key.

A series of cSRX environment variables enables you to modify the characteristics of the cSRX instance when it is launched. The specification of an environment variable is not mandatory; most environment variables have a default value as shown in Table 8 on page 41. If desired, you can launch the cSRX using the default environment variable settings.

For example, to launch a cSRX instance with an initial root account password, in secure-wire forwarding mode, and using the middle size cSRX configuration:

root@csrx-ubuntu3:~/csrx# docker run -d --privileged --network=mgt_bridge -e CSRX_ROOT_PASSWORD=<password> -e CSRX_SIZE="middle" -e CSRX_FORWARD_MODE="wire" --name=<csrx-container-name> <csrx-image-name>

NOTE: You must include the **--privileged** flag in the **docker run** command to enable the cSRX container to run in privileged mode.

Table 8 on page 41 summarizes the list of available cSRX environment variables along with a link to the topic that outlines its usage.

Table 8: Summary of cSRX Environment Variables

Variable	Description	Values	Default	Topic
CSRX_SIZE	cSRX size.	**"small"** | **"middle"** | **"large"**	**"large"**	"Changing the Size of a cSRX Container" on page 44
CSRX_FORWARD_MODE	Traffic forwarding mode.	**"routing"** | **"wire"**	**"routing"**	"Configuring Traffic Forwarding on a cSRX Container" on page 45

Table 8: Summary of cSRX Environment Variables *(continued)*

Variable	Description	Values	Default	Topic
CSRX_PACKET_DRIVER	Packet I/O driver.	**"poll"** | **"interrupt"**	**"poll"**	"Specifying the Packet I/O Driver for a cSRX Container" on page 50
CSRX_ROOT_PASSWORD	Initial root account password to log in to the cSRX container using SSH.	***string***	No default root password	"Specifying an Initial Root Password for Logging into a cSRX Container in a Linux Docker Environment" on page 43
CSRX_CTRL_CPU	CPU mask, indicating which CPU is running the cSRX control plane daemons (such as nsd, mgd, nstraced, utmd, and so on).	***hex value***	No CPU affinity	"Configuring CPU Affinity for a cSRX Container" on page 52
CSRX_DATA_CPU	CPU mask, indicating which CPU is running the cSRX data plane daemon (srxpfe).	***hex value***	No CPU affinity	"Configuring CPU Affinity for a cSRX Container" on page 52
CSRX_ARP_TIMEOUT	ARP entry timeout value for the control plane ARP learning or response.	***decimal value***	Same as the Linux host	"Configuring Traffic Forwarding on a cSRX Container" on page 45
CSRX_NDP_TIMEOUT	NDP entry timeout value for the control plane NDP learning or response.	***decimal value***	Same as the Linux host	"Configuring Traffic Forwarding on a cSRX Container" on page 45

Table 8: Summary of cSRX Environment Variables *(continued)*

Variable	Description	Values	Default	Topic
CSRX_PORT_NUM	Number of interfaces you need to add to container. Example: docker run -d --privileged --net=none -e CSRX_PORT_NUM=17 -e CSRX_ROOT_PASSWORD=<password> -e CSRX_SIZE=large -e CSRX_HUGEPAGES=no -e CSRX_PACKET_DRIVER=interrupt -e CSRX_FORWARD_MODE=routing --name=<csrx-container-name> <csrx-image-name>	Default is 3, maximum is 17 (which means 1 management interfaces and 16 data interfaces)	3	

Specifying an Initial Root Password for Logging into a cSRX Container in a Linux Docker Environment

If you intend to log into the cSRX container using SSH, specify an initial root password when launching the cSRX. When a cSRX container is launched, remote access using SSH will be enforced with username and password.

NOTE: After the cSRX container is started, change the password and, if desired, the authentication method for the root-level user.

To specify an initial root password for logging into the cSRX container:

root@csrx-ubuntu3:~/csrx# docker run -d --privileged --network=mgt_bridge -e CSRX_ROOT_PASSWORD=<password> --name=<csrx-container-name> <csrx-image-name>

Changing the Size of a cSRX Container

Based on your specific cSRX deployment requirements, scale requirements, and resource availability, you can scale the performance and capacity of a cSRX instance by specifying a specific size (small, middle, or large). Each cSRX size has certain characteristics and can be applicable to certain deployments. By default, the cSRX container launches using the large size configuration.

Table 9 on page 44 compares the scale requirements of a cSRX instance depending on the specified size.

Table 9: cSRX Size Comparison

Specification	cSRX: Small Size	cSRX: Middle Size	cSRX: Large Size (Default)
Physical Memory Overhead	256M	1G	4G
Number of Flow Sessions	8K	64K	512K

To assign a specific size for a cSRX instance, include the **CSRX_SIZE** environment variable in the **docker run** command.

For example, to launch a cSRX instance using the middle size configuration to scale performance and capacity:

root@csrx-ubuntu3:~/csrx# docker run -d --privileged --network=mgt_bridge -e CSRX_SIZE="middle" --name=<csrx-container-name> <csrx-image-name>

Configuring Traffic Forwarding on a cSRX Container

IN THIS SECTION

You can change the traffic forwarding mode of the cSRX container as a means to facilitate security service provisioning when running the cSRX. For example, if you deploy a cSRX container inline of protected segments, the cSRX should be transparent to avoid changing the virtual network topology. In other deployments, the cSRX container should be able to specify the next-hop address of egress traffic. To address variations in cSRX network deployment, you can configure the traffic forwarding mode of the cSRX to operate in routing mode (static routing only) or secure-wire mode.

NOTE: The cSRX uses **routing** as the default environment variable for traffic forwarding mode.

This section includes the following topics:

Configuring Routing Mode

When running the cSRX container in routing mode, the cSRX uses a static route to forward traffic for routes destined to interfaces ge-0/0/0 and ge-0/0/1. You will need to create a static route and specify the next-hop address.

When you start the cSRX container, you need to specify port number in the environment using the variable CSRX_PORT_NUM to define the number of interfaces you need to add to container in routing mode.

For example, to launch cSRX instance in routing mode with 17 interfaces:

root@csrx-ubuntu3:~/csrx# docker run -d --privileged --net=none -e CSRX_PORT_NUM=17 -e CSRX_ROOT_PASSWORD=<password> -e CSRX_SIZE=large -e CSRX_HUGEPAGES=no -e CSRX_PACKET_DRIVER=interrupt -e CSRX_FORWARD_MODE=routing --name=<srx-container-name> <csrx-image-name>

> **NOTE:** The interfaces specified in the CSRX_PORT_NUM environment variable (default value is 3) must be added to a network after instantiation of the cSRX. Unless all the interfaces are added to the bridge or the macvlan networks, the PFE will not be launched on the cSRX, and the ge-x/y/z interfaces will remain down.

Include the **-e CSRX_FORWARD_MODE=routing** environment variable in the **docker run** command to instruct the cSRX to run in static route forwarding mode.

To configure the cSRX container to run in static routing mode:

1. Launch the cSRX container in routing forwarding mode:

   root@csrx-ubuntu3:~/csrx# docker run -d --privileged --network=mgt_bridge -e CSRX_FORWARD_MODE="routing" --name=<csrx-container-name> <csrx-image-name>

2. After you start the cSRX container, log in to it and configure static routes.

   root@csrx# cli

   root@csrx> configure

   [edit]

root@csrx# show | display set

root@csrx# set interfaces ge-0/0/0 unit 0 family inet address 1.0.0.1/8

root@csrx# set interfaces ge-0/0/1 unit 0 family inet address 2.0.0.1/8

root@csrx# set routing-options static route 3.0.0.0/28 next-hop 1.0.0.10/32

3. View the forwarding table to verify the static routes.

   root@csrx> show route forwarding-table

```
Routing table: default.inet
Internet:
Destination        Type RtRef Next hop           Type Index    NhRef Netif
0.0.0.0            perm     0                    dscd      517     1
1.0.0.1            perm     0 1.0.0.1            locl     2006     1
1.0.0.10           perm     0 1.0.0.10          ucast     5501     1
1.255.255.255      perm     0                    bcst     2007     1
1/8                perm     0                    rslv     2009     1
2.0.0.1            perm     0 2.0.0.1            locl     2001     1
2.0.0.10           perm     0 2.0.0.10          ucast     5500     1
2.255.255.255      perm     0                    bcst     2002     1
2/8                perm     0                    rslv     2004     1
224.0.0.1          perm     0                    mcst      515     1
224/4              perm     0                    mdsc      516     1
3.0.0.0/28         perm     0 1.0.0.10          ucast     5501     1

Routing table: default.inet6
Internet6:
Destination        Type RtRef Next hop           Type Index    NhRef Netif
::                 perm     0                    dscd      527     1
ff00::/8           perm     0                    mdsc      526     1
ff02::1            perm     0                    mcst      525     1
```

4. Specify a route for the management interface. Static routes can only configure routes destined for interfaces ge-0/0/0 and ge-0/0/1. The route destined for the management interfaces (eth0) must be added by using the Linux **route** shell command.

   root@csrx% route add -net 10.10.10.0/24 gw 172.31.12.1

   root@csrx% route -n

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         0.0.0.0         0.0.0.0         U     0      0        0 pfe_tun
1.0.0.0         0.0.0.0         255.0.0.0       U     0      0        0 tap1
2.0.0.0         0.0.0.0         255.0.0.0       U     0      0        0 tap0
3.0.0.0         1.0.0.10        255.255.255.240 UG    0      0        0 tap1
10.10.10.0      172.31.12.1     255.255.255.0   UG    0      0        0 eth0
172.31.0.0      0.0.0.0         255.255.0.0     U     0      0        0 eth0
```

5. If required for your network environment, you can configure an IPv6 static route for the cSRX using the **set routing-options rib inet6.0 static route** command.

   [edit routing-options]

   root@csrx# set routing-options rib inet6.0 static route 3000::0/64 next-hop 1000::10/128

   [edit interfaces]

   root@csrx# commit

   root@csrx# show routing-options rib inet6.0

   static {

   route 3000::0/64 next-hop 1000::10/128;

   }

6. Under routing mode, the control plane ARP/NDP learning/response is provided by the Linux kernel through the TAP 0 and TAP 1 interfaces created to host the traffic for eth1 and eth2 through srxpfe. You can view ARP entries by using the Linux **arp** shell command.

   > **NOTE:** While there are multiple interfaces created inside the cSRX container, only two interfaces, ge-0/0/0 and ge-0/0/1, are visible in srxpfe.

   root@csrx% arp -a

```
? (2.0.0.10) at 6e:81:38:41:5e:0e [ether] on tap0
? (1.0.0.10) at 96:33:66:a1:e5:03 [ether] on tap1
? (172.31.12.1) at 02:c4:39:fa:0a:0d [ether] on eth0
```

The default ARP/NDP entries timeout is set to 1200 seconds. You can adjust this value by modifying either the **ARP_TIMEOUT** or **NDP_TIMEOUT** environment variable when launching the cSRX container. For example:

root@csrx-ubuntu3:~/csrx# docker run -d --privileged --network=mgt_bridge -e CSRX_FORWARD_MODE="routing" -e CSRX_ARP_TIMEOUT=<seconds> -e CSRX_NDP_TIMEOUT=<seconds> --name=<csrx-container-name> <csrx-image-name>

The maximum ARP entry number is controlled by the Linux host kernel. If there are a large number of neighbors, you might need to adjust the ARP or NDP entry limitations on the Linux host. There are options in the **sysctl** command on the Linux host to adjust the ARP or NDP entry limitations.

For example, to adjust the maximum ARP entries to 4096:

 # sysctl -w net.ipv4.neigh.default.gc_thresh1=1024

 # sysctl -w net.ipv4.neigh.default.gc_thresh2=2048

 # sysctl -w net.ipv4.neigh.default.gc_thresh3=4096

For example, to adjust the maximum NDP entries to 4096:

 # sysctl -w net.ipv6.neigh.default.gc_thresh1=1024

 # sysctl -w net.ipv6.neigh.default.gc_thresh1=2048

 # sysctl -w net.ipv6.neigh.default.gc_thresh1=4096

Configuring Secure-Wire Mode

When operating in secure-wire mode, all traffic that arrives on a specific interface, ge-0/0/0 or ge-0/0/1, will be forwarded unchanged through the interface. This mapping of interfaces, called *secure wire*, allows the cSRX to be deployed in the path of network traffic without requiring a change to routing tables or a reconfiguration of neighboring devices. A cross-connection is set up between interface pairs ge-0/0/0 and ge-0/0/1 to steer traffic from one port to the other port based on the Interworking and Interoperability Function (IIF) as the input key.

Include the **-e CSRX_FORWARD_MODE=wire** environment variable in the **docker run** command to instruct the cSRX to run in secure-wire forwarding mode.

> **NOTE:** When you launch the cSRX container in secure-wire mode, the cSRX instance automatically creates a default secure-wire named *csrx_sw* in the srxpfe process, and the ge-0/0/0 and ge-0/0/1 interface pair are added into the secure-wire.

Launch the cSRX instance in secure-wire mode using the following command:

root@csrx-ubuntu3:~/csrx# docker run -d --privileged --network=mgt_bridge -e CSRX_FORWARD_MODE="wire" --name=<csrx-container-name> <csrx-image-name>

Specifying the Packet I/O Driver for a cSRX Container

IN THIS SECTION

The cSRX container exchanges packets by using the Linux host user space driver over the VETH interface. The setting of the packet I/O driver can impact the forwarding performance and scalability of a cSRX container. You can launch a cSRX to use either the poll mode driver (default seting) or interrupt mode driver to define how packets are exchanged.

> **NOTE:** Poll mode is the default setting for the CSRX_PACKET_DRIVER environment variable.

Table 10 on page 50 compares the two packet I/O drivers supported by cSRX.

Table 10: cSRX Poll and Interrupt Mode Driver Comparison

Specification	Poll Mode Driver	Interrupt Mode Driver
Performance	Higher forwarding performance per cSRX.	Lower forwarding performance per cSRX.

Table 10: cSRX Poll and Interrupt Mode Driver Comparison ***(continued)***

Specification	Poll Mode Driver	Interrupt Mode Driver
Scalability	Reduced scalability; support for a single cSRX per vCPU.	Improved scalability; support for multiple cSRX containers per vCPU.
Scenario	Deployment of a cSRX supporting a virtualized network function (VNF).	Deployment of a cSRX supporting a large number of concurrent security services.

This section includes the following topics:

Specifying the Poll Mode Driver

The poll mode driver uses a PCAP-based DPDK driver to poll packets from the Linux VETH driver. Packets are exchanged between user and kernel space by using a Berkeley Packet Filter (BPF). The poll mode driver can obtain the best performance for a single cSRX container (for example, as a VNF).

> **NOTE:** When using the poll mode driver, the srxpfe process will always keep a CPU core at 100% utilization, even when the cSRX has no traffic to process.

To configure the cSRX container to use the poll mode driver, include the **-e CSRX_PACKET_DRIVER="poll"** environment variable in the **docker run** command.

```
root@csrx-ubuntu3:~/csrx# docker run -d --privileged --network=mgt_bridge -e
CSRX_FORWARD_MODE="routing" -e CSRX_PACKET_DRIVER="poll" -e CSRX_CTRL_CPU="0x1" -e
CSRX_DATA_CPU="0x6" --name=<csrx-container-name> <csrx-image-name>
```

Specifying the Interrupt Mode Driver

The interrupt mode driver receives and transmits packets using the packet socket on user space. By using the epoll mechanism provided by the Linux operating system, the interrupt mode driver can aid the srxpfe process in waiting until packets arrive on the VETH interfaces. If no packets load on the revenue ports of a cSRX instance, the srxpfe process remains in a sleep state as a means to help preserve CPU resources. With the support of the epoll mechanism, the Linux server can then sustain a large number of cSRX instances, in particular when there are multiple cSRX instances per CPU. In this case, the scheduler keeps track of which srxpfe process is busy and allocates CPU resources to that srxpfe process.

When you launch start a cSRX instance, you can include the CSRX_CTRL_CPU and CSRX_DATA_CPU environmental variables to specify a specific CPU to run control plane and data plane tasks. The CPU will schedule the srxpfe process among those CPUs according to their CPU status. See"Configuring CPU Affinity for a cSRX Container" on page 52 for details on the CSRX_CTRL_CPU and CSRX_DATA_CPU environmental variables.

To configure the cSRX container to use the interrupt mode driver, include the **-e CSRX_PACKET_DRIVER="interrupt"** environment variable in the **docker run** command.

root@csrx-ubuntu3:~/csrx# docker run -d --privileged --network=mgt_bridge -e CSRX_FORWARD_MODE="routing" -e CSRX_PACKET_DRIVER="interrupt" -e CSRX_CTRL_CPU="0x1" -e CSRX_DATA_CPU="0x6" --name=<csrx-container-name> <csrx-image-name>

Configuring CPU Affinity for a cSRX Container

A cSRX instance requires two CPU cores in the Linux server. To help schedule the Linux server tasks and adjust performance of the cSRX container running on a Linux host, you can launch the cSRX container and assign its control and data processes (or daemons) to a specific CPU. In a cSRX container, srxpfe is the data plane daemon and all other daemons (such as nsd, mgd, nstraced, utmd, and so on) are control plane daemons.

CPU affinity ensures that the cSRX control and data plane daemons are pinned to a specific physical CPU, which can improve the cSRX container performance by using the CPU cache efficiently. By default, there is not a defined CPU affinity for the cSRX control and data plane daemons; the CPU on which the control and data plane daemons run depends on Linux kernel scheduling.

To assign cSRX container control and data daemons to a specific CPU, include the environment variables **CSRX_CTRL_CPU** and **CSRX_DATA_CPU** in the **docker run** command.

For example, to configure the cSRX container to launch the control plane daemons on CPU 1 and the data plane daemon on CPU 2:

root@csrx-ubuntu3:~/csrx# docker run -d --privileged --network=mgt_bridge -e CSRX_CTRL_CPU="0x1" -e CSRX_DATA_CPU="0x2" --name=<csrx-container-name> <csrx-image-name>

Enabling Persistent Log File Storage to a Linux Host Directory

In a cSRX container, log files are stored in the /var/log directory. By default, if there are no external volumes mounted for the /var/log directory, the log files will be maintained only for this cSRX container. If, at a future point, the cSRX container is deleted, those log files will be lost. You can enable persistent log file storage to a Linux host directory as a means to directly mount a directory from a Linux host to the cSRX container when the cSRX is launched.

To configure the cSRX container to enable persistent log file storage to a Linux host directory, use the following command.

root@csrx-ubuntu3:~/csrx# docker run -d --privileged --network=mgt_bridge -e CSRX_FORWARD_MODE="routing" -e CSRX_PACKET_DRIVER="poll" –e CSRX_CTRL_CPU="0x1" –e CSRX_DATA_CPU="0x6" -v <path-log-directory-on-host>:/var/log --name=<csrx-container-name> <csrx-image-name>

Managing cSRX Containers

IN THIS SECTION

This section outlines basic Docker commands that you can use with a running cSRX container. It includes the following topics:

Pausing/Resuming Processes within a cSRX Container

You can suspend or resume all processes within one or more cSRX containers. On Linux, this task is performed using the **cgroups freezer** process.

To pause and restart a cSRX container:

1. Use the **docker pause** command to suspend all processes in a cSRX container.

   ```
   hostOS# docker pause <csrx-container-name>
   ```

2. Use the **docker unpause** command to resume all processes in the cSRX container.

   ```
   hostOS# docker unpause <csrx-container-name>
   ```

Viewing Container Processes on a Running cSRX Container

Use the **docker exec** command to view the details of the processes (applications, services and status) running on a cSRX container.

hostOS# **docker exec <csrx-container-name> ps aux**

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  18048  1648 pts/8    Ss   May15   0:00 /bin/bash -e
/etc/rc.local init
root        78  0.0  0.0 260072   968 ?        Ssl  May15   0:09 /usr/sbin/rsyslogd
 -M/usr/lib/rsyslog
root        97  0.0  0.0  61376  1304 ?        Ss   May15   0:00 /usr/sbin/sshd
root       118  0.0  0.0 108552  1304 ?        Sl   May15  34:12 /usr/bin/monit
root       124  0.0  0.0 723392  1516 ?        Ss   May15   0:00 /usr/sbin/nstraced
root       133  0.0  0.0 734084  4388 ?        Ss   May15   1:18 /usr/sbin/nsd
root       135  0.0  0.0   4440   644 ?        S    May15   0:00 /bin/sh
/etc/init.d/appidd start
root       141  0.0  0.2 752132 21184 ?        Sl   May15   0:02 /usr/sbin/appidd
 -N &
root       147  0.0  0.0   4440   652 ?        S    May15   0:00 /bin/sh
/etc/init.d/idpd start
root       153  0.0  0.0 730520  2768 ?        S    May15   0:25 /usr/sbin/idpd
-N &
root       170  0.0  0.1 1001088 12528 ?       Sl   May15  29:22 /usr/sbin/useridd
 -N
root       211  0.0  0.0 728448  2104 ?        Ss   May15   0:07 /usr/sbin/mgd
root       222  3.5  1.8 3943936 152920 ?      Sl   May15 1416:22 /usr/sbin/srxpfe
 -a -d
root       250  0.0  0.0   4440   648 ?        S    May15   0:00 /bin/sh
/etc/init.d/utmd start
root       256  0.0  0.0 725092  3880 ?        S    May15   1:36 /usr/sbin/utmd
-N &
root       267  0.0  0.0 731556  2472 ?        Ss   May15   2:39 /usr/sbin/kmd
root       301  0.0  0.0  18160  1916 pts/8    S+   May15   0:00 /bin/bash
root       324  0.0  0.0 853708  3324 ?        Sl   May15   6:13 /usr/sbin/wmic
-N
```

Removing a cSRX Container or Image

To remove a cSRX container or image:

NOTE: You must first stop and remove a cSRX container before you can remove a cSRX image.

1. Use the **docker stop** command to stop the cSRX container.

   hostOS# **docker stop <csrx-container-name>**

2. Use the **docker rm** command to remove the cSRX container.

   hostOS# **docker rm <csrx-container-name>**

   NOTE: Include **--force** to force the removal of a running cSRX container.

3. Use the **docker rmi** command to remove one or more cSRX images from the Docker Engine.

   NOTE: Include **--force** to force the removal a cSRX image.

   hostOS# **docker rmi <csrx-container-name>**

RELATED DOCUMENTATION

Docker Engine User Guide

Docker commands

4

CHAPTER

Configuring cSRX

cSRX Configuration and Management Tools

IN THIS SECTION

Understanding the Junos OS CLI and Junos Scripts

The Junos operating system command-line interface (Junos OS CLI) is a Juniper Networks specific command shell that runs on top of a UNIX-based operating system kernel.

Built into Junos OS, Junos script automation is an onboard toolset available on all Junos OS platforms, including routers, switches, and security instances.

You can use the Junos OS CLI and the Junos OS scripts to configure, manage, administer, and troubleshoot the cSRX container.

Understanding cSRX with Contrail and Openstack Orchestration

The cSRX Container Firewall can provide security services in a software-defined networking (SDN) environment. Juniper Networks Contrail is an open, standards-based software-defined networking (SDN) platform that delivers network virtualization and service automation for federated cloud networks. You use the Contrail Cloud Platform with open cloud orchestration systems such as OpenStack or CloudStack to instantiate instances of cSRX in a containerized environment. Contrail Cloud Platform automates the orchestration of compute, storage, and networking resources to create and scale open, intelligent, and reliable OpenStack clouds that seamlessly merge and hybridize through highly intelligent secure networks.

cSRX can be deployed as a dedicated firewall compute node in a Contrail Cloud platform environment to provide differentiated Layer 4 through 7 security services for multiple tenants as part of a service chain in the Contrail cloud platform. In the Contrail networking environment, you can deploy the cSRX container as a large-scale security service in a multicloud environment, and configure the cSRX to steer traffic from a vRouter with vRouter interface (VIF). Traffic and health statistics are monitored by the Contrail service orchestrator.

See cSRX Guide for Contrail for details on using cSRX with Juniper Networks Contrail.

RELATED DOCUMENTATION

Introducing the Junos OS Command-Line Interface

Contrail Networks

Mastering Junos Automation Programming

Configuring cSRX Using the Junos OS CLI

This section provides basic CLI configurations that can be used for configuring cSRX containers. For more details see, Introducing the Junos OS Command-Line Interface.

To configure the cSRX container using the Junos OS CLI:

1. Log in to the cSRX container using SSH.

```
root@csrx-ubuntu3:~/csrx#ssh 192.168.42.81
```

2. Start the CLI as root user.

> **NOTE:** When a cSRX container is launched, if you specified to log into the cSRX container with an initial root password, access to the cSRX container using SSH will be enforced with user name and password.

```
root#cli
root@>
```

3. Verify the interfaces.

root@> show interfaces

```
Physical interface: ge-0/0/1, Enabled, Physical link is Up
  Interface index: 100
  Link-level type: Ethernet, MTU: 1514
  Current address: 02:42:ac:13:00:02, Hardware address: 02:42:ac:13:00:02
Physical interface: ge-0/0/0, Enabled, Physical link is Up
  Interface index: 200
```

```
Link-level type: Ethernet, MTU: 1514
Current address: 02:42:ac:14:00:02, Hardware address: 02:42:ac:14:00:02
```

4. Enter configuration mode.

```
configure
[edit]
root@#
```

5. Set the root authentication password by entering a cleartext password, an encrypted password, or an SSH public key string (DSA or RSA).

```
[edit]
root@# set system root-authentication plain-text-password
New password: password
Retype new password: password
```

6. Configure the hostname.

```
[edit]
root@# set system host-name host-name
```

7. Configure the two traffic interfaces.

> **NOTE:** Docker automatically connects the fxp0 management interface (eth0) to the Linux bridge and automatically assigns an IP address. If is not necessary for you to configure the management interface for the cSRX container.

```
[edit]
root@# set interfaces ge-0/0/0 unit 0 family inet address 192.168.20.2/24
root@# set interfaces ge-0/0/1 unit 0 family inet address 192.168.10.2/24
```

8. Configure basic security zones for the public and private interfaces and bind them to traffic interfaces.

```
[edit]
root@# set security zones security-zone untrust interfaces ge-0/0/0.0
```

```
root@# set security zones security-zone trust interfaces ge-0/0/1.0
root@# set security policies default-policy permit-all
```

9. Verify the configuration.

```
[edit]
root@# commit check
configuration check succeeds
```

10. Commit the configuration to activate it on the cSRX instance.

```
[edit]
root@# commit
commit complete
```

11. (Optional) Use the **show** command to display the configuration to verify that it is correct.

RELATED DOCUMENTATION

Junos OS for SRX Series

Introducing the Junos OS Command-Line Interface