

cSRX Container Firewall Deployment Guide for Bare-Metal Linux Server

Published
2024-06-16

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

Juniper Networks, the Juniper Networks logo, Juniper, and Junos are registered trademarks of Juniper Networks, Inc. in the United States and other countries. All other trademarks, service marks, registered marks, or registered service marks are the property of their respective owners.

Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.

cSRX Container Firewall Deployment Guide for Bare-Metal Linux Server
Copyright © 2024 Juniper Networks, Inc. All rights reserved.

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 | v

1

Overview

Understanding cSRX Container Firewall with a Bare-Metal Linux Server | 2

Junos OS Features Supported on cSRX Container Firewall | 9

2

Installing cSRX Container Firewall

Requirements for Deploying cSRX Container Firewall on a Bare-Metal Linux Server | 26

Installing cSRX Container Firewall in a Bare-Metal Linux Server | 29

Before You Deploy | 29

Confirming Docker Installation | 30

Loading the cSRX Container Firewall Image | 31

Creating the Linux Bridge Network for the cSRX Container Firewall | 33

Launching the cSRX Container Firewall Container | 34

3

cSRX Managing Container Firewall Containers

cSRX Container Firewall Environment Variables Overview | 39

Changing the Size of a cSRX Container Firewall Container | 41

Configuring Traffic Forwarding on a cSRX Container Firewall Container | 42

Configuring Routing Mode | 43

Configuring Secure-Wire Mode | 47

Configuring CPU Affinity for a cSRX Container Firewall Container | 48

Enabling Persistent Log File Storage to a Linux Host Directory | 48

Managing cSRX Container Firewall Containers | 49

Pausing/Resuming Processes within a cSRX Container Firewall Container | 49

Viewing Container Processes on a Running cSRX Container Firewall Container | 50

Removing a cSRX Container Firewall Container or Image | 50

Configuring cSRX Container Firewall

cSRX Container Firewall Configuration and Management Tools | 53

Configuring cSRX Container Firewall Using the Junos OS CLI | 54

About This Guide

Use this guide to install the cSRX Container Firewall 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 Firewall 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.

1

CHAPTER

Overview

[Understanding cSRX Container Firewall with a Bare-Metal Linux Server | 2](#)

[Junos OS Features Supported on cSRX Container Firewall | 9](#)

Understanding cSRX Container Firewall with a Bare-Metal Linux Server

IN THIS SECTION

- [cSRX Container Firewall Overview | 2](#)
- [cSRX Container Firewall Benefits and Uses | 6](#)
- [Docker Overview | 7](#)
- [cSRX Container Firewall Scale-Up Performance | 8](#)

The cSRX Container Firewall Container Firewall is a containerized version of the SRX Series Services Gateway with a low memory footprint. cSRX Container Firewall provides advanced security services, including content security, AppSecure, and Content Security in a container form factor. By using a Docker container in a bare-metal Linux server, the cSRX Container Firewall 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 Container Firewall enables agile, high-density security service deployment.

This section includes the following topics:

cSRX Container Firewall Overview

The cSRX Container Firewall 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 Firewall 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 Firewall container. cSRX Container Firewall 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 Firewall container runs, there are several daemons inside the Docker container that launch automatically when the cSRX Container Firewall 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, Content Security, IDP, AppID, and so on). srxpfe is the data-plane daemon that receives and sends packets from the revenue ports of a cSRX Container Firewall container. The cSRX Container Firewall 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 Container Firewall enables advanced security at the network edge in a multitenant virtualized environment. cSRX Container Firewall provides Layer 4 through 7 advanced security features such as firewall, IPS, and AppSecure. The cSRX Container Firewall container also provides an additional interface to manage the cSRX Container Firewall. When cSRX Container Firewall 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 Container Firewall services. cSRX Container Firewall then sends the frames out of the other interface.

Launch the cSRX Container Firewall 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 Firewall 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 Container Firewall in a Bare-Metal Linux Server" on page 29](#).

[Figure 1 on page 4](#) illustrates the cSRX Container Firewall operating in secure-wire mode. It is an example of how a cSRX Container Firewall container is bridged with an external network. In this illustration, cSRX Container Firewall eth1 is bridged with host physical NIC eth1 and cSRX Container Firewall eth2 is bridged with host physical NIC eth2.

Figure 1: cSRX Container Firewall in Secure-Wire Mode

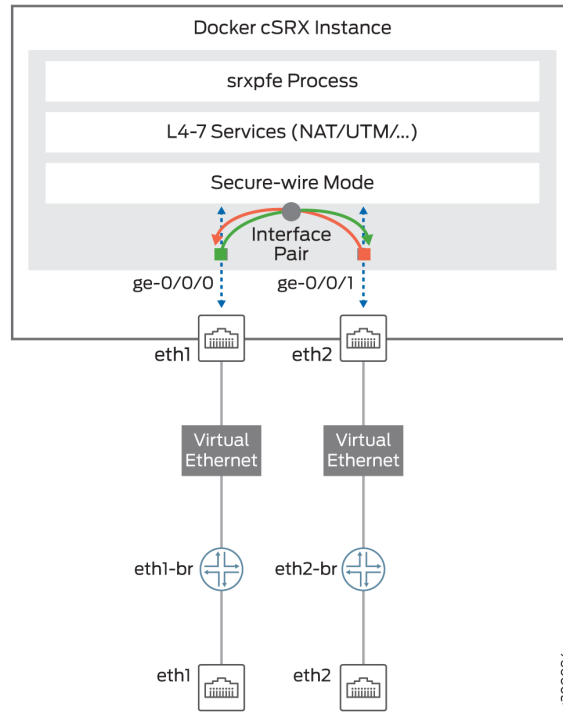
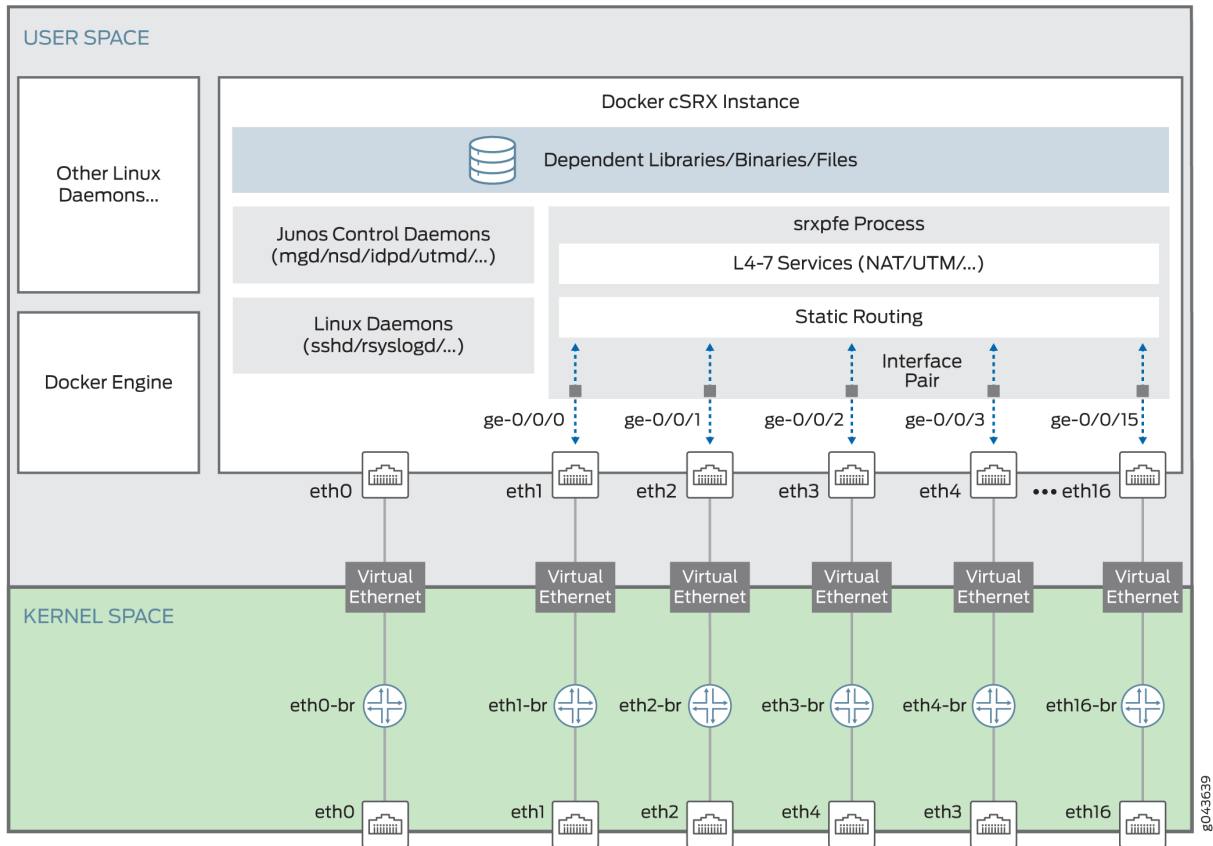


Figure 2 on page 5 illustrates the cSRX Container Firewall operating in routing mode.

Figure 2: cSRX Container Firewall 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 Container Firewall 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 Firewall 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 Firewall 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 Firewall container. The cSRX Container Firewall is ideal for public, private, and hybrid cloud environments.

Some of the key benefits of cSRX Container Firewall 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 Container Firewall 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 Container Firewall 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 Container Firewall can be deployed as a large scale security service.
- Application security features (including IPS and AppSecure).
- Content Security 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 Container Firewall and vSRX Virtual Firewall are similar, there are scenarios in which each product is the optimal option in your environment. For example, the cSRX Container Firewall 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 Virtual Firewall VM

provides the best feature set. For environments focused on security services in a Docker containerized deployment, cSRX Container Firewall is a better fit.

See ["Junos OS Features Supported on cSRX Container Firewall" on page 9](#) for a summary of the feature categories supported on cSRX Container Firewall, and also for a summary of features not supported on cSRX Container Firewall.

You can deploy the cSRX Container Firewall 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 Container Firewall can provide security for Layer 4 through 7 traffic, managed by Security Director.
- Private clouds—cSRX Container Firewall can provide security services in a private cloud running containerized workloads and can include Contrail integration.

Docker Overview

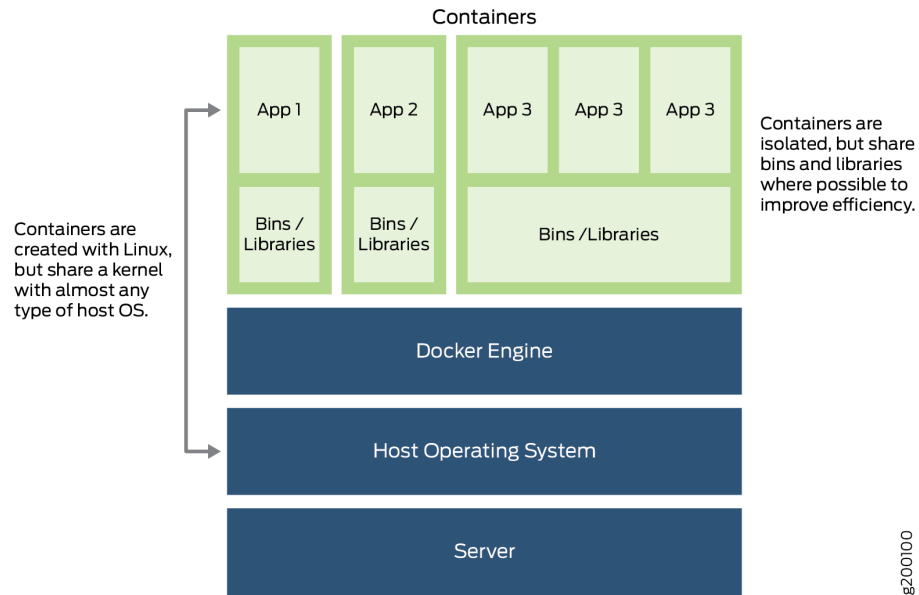
Docker is an open source software platform that simplifies the creation, management, and teardown of a virtual container that can run on any Linux server. A Docker container is an open source software development platform, with its main benefit being to package applications in “containers” to allow them 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 8](#) provides an overview of a typical Docker container environment.

Figure 3: Docker Container Environment



cSRX Container Firewall Scale-Up Performance

You can scale the performance and capacity of a cSRX Container Firewall container by increasing the allocated amount of virtual memory or the number of flow sessions. [Table 1 on page 8](#) shows the cSRX Container Firewall scale-up performance applied to a cSRX Container Firewall container based on its supported sizes: small, medium, and large. The default size for a cSRX Container Firewall container is large.

NOTE: See [Changing the Size of a cSRX Container](#) for the procedure on how to scale the performance and capacity of a cSRX Container Firewall container by changing the container size.

Table 1: cSRX Container Firewall Scale Up Performance

cSRX Container Firewall Size	Physical Memory Overhead	Number of Flow Sessions	Release Introduced
Small	256M	8K	Junos OS Release 18.1R1

Table 1: cSRX Container Firewall Scale Up Performance *(Continued)*

cSRX Container Firewall Size	Physical Memory Overhead	Number of Flow Sessions	Release Introduced
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 Container Firewall

IN THIS SECTION

- Supported SRX Series Features on cSRX Container Firewall | 10
- SRX Series Features Not Supported on cSRX Container Firewall | 17

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

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

Supported SRX Series Features on cSRX Container Firewall

[Table 2 on page 10](#) provides a high-level summary of the feature categories supported on cSRX Container Firewall and any feature considerations.

To determine the Junos OS features supported on cSRX Container Firewall, 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 2: SRX Series Features Supported on cSRX Container Firewall

Feature	Considerations
Application Firewall (AppFW)	Application Firewall Overview
Application Identification (AppID)	Understanding Application Identification Techniques
Application Tracking (AppTrack)	Understanding AppTrack
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	<p>A cSRX Container Firewall container supports 17 interfaces:</p> <ul style="list-style-type: none"> • 1 Out-of-band management Interface (eth0) • 16 In-band interfaces (ge-0/0/0 to ge-0/0/15). <p>Network Interfaces</p>

Table 2: SRX Series Features Supported on cSRX Container Firewall *(Continued)*

Feature	Considerations
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 Container Firewall platform, such as: <ul style="list-style-type: none"> • 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
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

Table 2: SRX Series Features Supported on cSRX Container Firewall (Continued)

Feature	Considerations
System Logs and Real-Time Logs	Starting in Junos OS Release 20.1R1, you can monitor traffic using system logs and RTlogs.
User Firewall	<p>Includes support for all user firewall functionality on the cSRX Container Firewall platform, such as:</p> <ul style="list-style-type: none"> • Policy enforcement with matching source identity criteria • Logging with source identity information • Integrated user firewall with active directory • Local authentication <p>For SRX Series user firewall configuration details, see:</p> <p>Overview of Integrated User Firewall</p>
Content Security	<p>Includes support for all Content Security functionality on the cSRX Container Firewall platform, such as:</p> <ul style="list-style-type: none"> • Antispam • Sophos Antivirus • Web filtering • Content filtering <p>For SRX Series Content Security configuration details, see:</p> <p>Unified Threat Management Overview</p> <p>For SRX Series Content Security antispam configuration details, see:</p> <p>Antispam Filtering Overview</p>
Zones and zone-based IP spoofing	Understanding IP Spoofing

Table 3: IKE and IPsec features

Feature		Supported on cSRX
IKE Features	Pre-shared key	Yes
	Certificate authentication	Yes
	IKEv1 (main mode/aggressive mode)	No
	IKEv2	Yes
	Route-based VPN	Yes
	Site-to-site VPN	Yes
	Auto VPN	Yes
	Dynamic endpoint VPN	Yes
	Point-to-point tunnel interfaces	Yes
	Point-to-multipoint tunnel interfaces	No
	Numbered tunnel interfaces	No
	Unnumbered tunnel interface	Yes
	Hub-and-spoke scenario for site-to-site VPNs	Yes
	Unicast static and dynamic (RIP, OSPF, BGP) routing over st0 interface	No

Table 3: IKE and IPsec features (*Continued*)

Feature		Supported on cSRX
	Virtual router	No
	IKED crash recovery	Yes
	Chassis Cluster	No
	HA Link Encryption	No
	Local address selection	Yes
	Loopback address termination	No
	DNS name as IKE gateway address	Yes
	NAT-Traversal (NAT-T) for IPv4 IKE peers	Yes
	Dead Peer Detection (DPD)	Yes
	Generic proposals and policies for IPv4 and IPv6	Yes
	General IKE ID	Yes
	Single proxy ID pairs	No
	Multiple traffic selector pairs	Yes
	Dual-stack (parallel IPv4 and IPv6 tunnels) over a single physical interface	Yes

Table 3: IKE and IPsec features *(Continued)*

Feature		Supported on cSRX
	Authentication Algorithms - md5, sha1, sha-256, sha-384, sha-512	Yes
	Encryption Algorithms - des-cbc, 3des-cbc, aes-128-cbc, aes-128-gcm, aes-192-cbc, aes-256-cbc, aes-256-gcm	Yes
	IKE Proposal Sets - basic, compatible, standard, prime-128, prime-256, suiteb-gcm-128, suiteb-gcm-256	Yes
	DH groups - 1,2,5,14,15,16,19,20,21,24	Yes
	Local Identity - distinguished-name, hostname, ipv4/v6 address, user-at-hostname, key-id	Yes
	Remote Identity - distinguished-name, hostname, ipv4/v6 address, user-at-hostname, key-id	Yes
	IKE Reauthentication (initiator and responder)	Yes
	Configuration payload	No
	EAP	No
	Remote Access - NCP/Licensing	No

Table 3: IKE and IPsec features *(Continued)*

Feature		Supported on cSRX
	Tunnel establishment - immediately, on-traffic, responder-only and responder-only-no-rekey mode	Yes
	Distribution-Profile	No
	Tunnel re-distribution	No
	IKEv2 Fragmentation	Yes
	SNMP MIB	No
	Statistics, logs, per-tunnel debugging	Yes
	IKE termination on lo0 interface	No
IPsec/Dataplane Features	ESP and AH tunnel modes	Yes
	Extended sequence number	Yes
	Lifetime of IKE or IPsec SA, in seconds	Yes
	Encryption Algorithms – des-cbc, 3des-cbc, aes-128-cbc, aes-192-cbc, aes-256-cbc, aes-gcm-128, aes-gcm-256	Yes

Table 3: IKE and IPsec features *(Continued)*

Feature		Supported on cSRX
	Authentication-algorithm - hmac-sha1-96, hmac-md5-96, hmac-sha-256-128, hmac-sha-384, hmac-sha-512	Yes
	Don't Fragment bit	Yes
	IPv6 extension headers	Yes
	IPsec fragmentation and reassembly	Yes
	Session affinity	No
	Power mode IPsec	Yes
	Configurable anti-replay window	Yes
	DSCP Copy	Yes
	Configurable delay installation of rekeyed outbound SAs	Yes
	Cos on st0	No

SRX Series Features Not Supported on cSRX Container Firewall

Table 4 on page 18 lists SRX Series features that are not applicable in a containerized environment, that are not currently supported, or that have qualified support on cSRX Container Firewall.

Table 4: SRX Series Features Not Supported on cSRX Container Firewall

	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-Content Security 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	

Table 4: SRX Series Features Not Supported on cSRX Container Firewall *(Continued)*

	SRX Series Feature
	LACP in standalone or chassis cluster mode
	Layer 3 LAG on routed ports
	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	

Table 4: SRX Series Features Not Supported on cSRX Container Firewall *(Continued)*

	SRX Series Feature
	Aggregated Ethernet interface
	IEEE 802.1X dynamic VLAN assignment
	IEEE 802.1X MAC bypass
	IEEE 802.1X port-based authentication control with multisuppliant support
	Interleaving using MLFR
	PoE
	PPP interface
	PPPoE-based radio-to-router protocol
	PPPoE interface
	Promiscuous mode on interfaces
VPNs	
	Acadia - Clientless VPN
	DVPN
	Multicast for AutoVPN
IPv6 Support	

Table 4: SRX Series Features Not Supported on cSRX Container Firewall *(Continued)*

	SRX Series Feature
	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
	High availability
	J-Web
	Logical systems
	MPLS
	Outbound SSH
	Remote instance access

Table 4: SRX Series Features Not Supported on cSRX Container Firewall (*Continued*)

	SRX Series Feature
	RESTCONF
	ATP Cloud
	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 <i>gr</i> , <i>ip</i> , and <i>st0</i> . Packet capture is not supported on a redundant Ethernet interface (<i>reth</i>).
Routing	

Table 4: SRX Series Features Not Supported on cSRX Container Firewall (*Continued*)

	SRX Series Feature
	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	<p>cSRX Container Firewall does not support all the log functions supported on other SRX Series Firewalls or vSRX Virtual Firewall instances due to limited CPU power and disk capacity.</p> <p>Unsupported system logs and real-time log functions on cSRX Container Firewall are:</p> <ul style="list-style-type: none"> • 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	
	Content Security

Table 4: SRX Series Features Not Supported on cSRX Container Firewall *(Continued)*

	SRX Series Feature
Content Security	
	Express AV
	Kaspersky AV
Upgrading and Rebooting	
	Autorecovery
	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 Container Firewall

Requirements for Deploying cSRX Container Firewall on a Bare-Metal Linux Server | 26

Installing cSRX Container Firewall in a Bare-Metal Linux Server | 29

Launching the cSRX Container Firewall Container | 34

Requirements for Deploying cSRX Container Firewall on a Bare-Metal Linux Server

IN THIS SECTION

- [Host Requirements | 26](#)
- [cSRX Container Firewall Basic Configuration Settings | 27](#)
- [Interface Naming and Mapping | 27](#)

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

Host Requirements

[Table 5 on page 26](#) lists the Linux host requirement specifications for deploying a cSRX Container Firewall container on a bare-metal Linux server.

NOTE: The cSRX Container Firewall 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 Firewall container.

Table 5: Host Requirement Specifications for cSRX Container Firewall

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	

Table 5: Host Requirement Specifications for cSRX Container Firewall (Continued)

Component	Specification	Release Introduced
Docker Engine	Docker Engine 1.9 or later installed on a Linux host	
Contrail Cloud Platform	Contrail 3.2 with OpenStack Liberty or OpenStack Mitaka	
vCPUs	2 CPU cores	
Memory	4 GB	
Disk space	40 GB hard drive	
Host processor type	x86_64 multicore CPU	
Network interface	1 Ethernet port (minimum)	

cSRX Container Firewall Basic Configuration Settings

The cSRX Container Firewall 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 Firewall 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 6 on page 28](#) lists the cSRX Container Firewall interface assignments with Docker.

Table 6: cSRX Container Firewall Interface Assignment

Interface Number	cSRX Container Firewall Interfaces	Docker Interfaces
1	eth0	eth0
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

Table 6: cSRX Container Firewall Interface Assignment (*Continued*)

Interface Number	cSRX Container Firewall Interfaces	Docker Interfaces
17	ge-0/0/15	eth16

Installing cSRX Container Firewall in a Bare-Metal Linux Server

IN THIS SECTION

- [Before You Deploy | 29](#)
- [Confirming Docker Installation | 30](#)
- [Loading the cSRX Container Firewall Image | 31](#)
- [Creating the Linux Bridge Network for the cSRX Container Firewall | 33](#)

This section outlines the steps to install the cSRX Container Firewall container in a Linux bare-metal server environment that is running Ubuntu, Red Hat Enterprise Linux (RHEL), or CentOS. The cSRX Container Firewall 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 Container Firewall as an advanced security service in a Linux container environment, ensure that you:

- Review "[Requirements for Deploying cSRX Container Firewall on a Bare-Metal Linux Server](#)" on page 26 to verify the system software requirement specifications for the Linux server required to deploy the cSRX Container Firewall 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/>

Confirming Docker Installation

Before you load the cSRX Container Firewall 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 Jun 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 Jun 23 21:17:13 2017
```

```
OS/Arch: linux/amd64
```

```
Experimental: False
```

Loading the cSRX Container Firewall Image

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

1. Download the cSRX Container Firewall software image from the [Juniper Networks website](#). The filename of the downloaded cSRX Container Firewall software image must not be changed to continue with the installation.
2. You can either download the cSRX Container Firewall image file normally using the browser or use the URL to download the image directly on your device as in the following example:

Run the following command to download images to a local registry using curl command or any other http utility. The syntax for curl commands is:

```
root@csrx-ubuntu3:~csrx# curl -o <file destination path> <Download link url>
```

```
root@csrx-ubuntu3:/var/tmp# curl -o /var/tmp/images/junos-csrx-docker-20.2R1.10.img "https://cdn.juniper.net/software/csrx/20.2R1.10/junos-csrx-docker-20.2R1.10.img?SM_USER=user=1595350694_5dbf6e62442de6bf14079d05a72464d4"
```

% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current
			Dload Upload	Total	Spent	Left	Speed
100	160M	100	160M	0	0	1090k	0
				0:02:30	0:02:30	--:--:--	1230k

3. Locate the cSRX Container Firewall image by using the ls Linux shell command.

```
root@csrx-ubuntu3:/var/tmp/images# ls
```

4. Load the downloaded cSRX Container Firewall image to the local registry.

```
root@csrx-ubuntu3:/var/tmp/images# docker image load -i /var/tmp/images/junos-csrx-docker-20.2R1.10.img
```

```
e758932b9168: Loading layer [=====>] 263MB/263MB
23f7a9961879: Loading layer [=====>] 14.51MB/14.51MB
1e4139e6fa81: Loading layer [=====>] 270.3MB/270.3MB
10334b424f86: Loading layer [=====>] 16.9kB/16.9kB
202ebb2f1137: Loading layer [=====>] 2.56kB/2.56kB
bc4a16173327: Loading layer [=====>] 1.536kB/1.536kB
8f9a9945544a: Loading layer [=====>] 2.048kB/2.048kB
Loaded image: csrx:20.2R1.10
```

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

```
root@csrx-ubuntu3:/var/tmp/images# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
SIZE			

csrx	20.2R1.10	88597d2d4940	2 weeks ago
534MB			

Creating the Linux Bridge Network for the cSRX Container Firewall

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 Firewall 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 Firewall container that includes: mgt_bridge (eth0), left_bridge (eth1), and right_bridge (eth2). The mgt_bridge is used by the cSRX Container Firewall for out-of-band management to accept management sessions and traffic, and the left_bridge and right_bridge are both used by the cSRX Container Firewall as the revenue ports to process in-band data traffic.

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. cSRX Container Firewall must be bound with the Linux bridge to pass traffic.

To create a three-bridge network for a cSRX Container Firewall 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 Firewall Container

You are now ready to launch the cSRX Container Firewall container that is running in Docker on the Linux bare-metal server. When you start the cSRX Container Firewall image, you have a running container of the image. You can stop and restart the cSRX Container Firewall container (see ["Managing cSRX Container Firewall Containers" on page 49](#)), and the container will retain all settings and file system changes unless those changes are explicitly deleted. However, the cSRX Container Firewall will lose anything in memory and all processes will be restarted.

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

- When you deploy cSRX Container Firewall you must enable the SSH service and SSH option for root-login. SSH service is not enabled by default.

To enable SSH service run the `set system services ssh` command and for root user login run the `set system services ssh root-login allow` command.

- Traffic forwarding mode (static route or secure-wire)
- cSRX Container Firewall container size (small, medium, or large)
- Packet I/O driver (polled or interrupt)
- CPU affinity for cSRX Container Firewall 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 Firewall container; most environment variables have a default value as shown in ["cSRX Container Firewall Environment Variables Overview" on page 39](#). You can launch the cSRX Container Firewall using the default environment variable settings.

To launch the cSRX Container Firewall container:

1. Use the `docker run` command to launch the cSRX Container Firewall container. You include the `mgt_bridge` management bridge to connect the cSRX Container Firewall to a network.

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

For example, to launch `csrx2` using cSRX Container Firewall software image `csrx:18.21R1.9` enter:

```
root@csrx-ubuntu3:~/csrx# docker run -d --privileged --network=mgt_bridge -e --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 Firewall 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 Firewall 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 Firewall 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 Firewall 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 Firewall 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

cSRX Managing Container Firewall Containers

[cSRX Container Firewall Environment Variables Overview | 39](#)

[Changing the Size of a cSRX Container Firewall Container | 41](#)

[Configuring Traffic Forwarding on a cSRX Container Firewall Container | 42](#)

[Configuring CPU Affinity for a cSRX Container Firewall Container | 48](#)

[Enabling Persistent Log File Storage to a Linux Host Directory | 48](#)

[Managing cSRX Container Firewall Containers | 49](#)

cSRX Container Firewall 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 Container Firewall image launches. You set an environment variable by specifying the `docker run -e VARIABLE=VALUE ... key`.

A series of cSRX Container Firewall environment variables enables you to modify the characteristics of the cSRX Container Firewall 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 7 on page 39](#). If desired, you can launch the cSRX Container Firewall using the default environment variable settings.

For example, to launch a cSRX Container Firewall instance in secure-wire forwarding mode, and using the middle size cSRX Container Firewall configuration:

```
root@csrx-ubuntu3:~/csrx# docker run -d --privileged --network=mgt_bridge -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 Firewall container to run in privileged mode.

[Table 7 on page 39](#) summarizes the list of available cSRX Container Firewall environment variables along with a link to the topic that outlines its usage.

Table 7: Summary of cSRX Container Firewall Environment Variables

Variable	Description	Values	Default	Topic
CSRX_FORWARD_MODE	Traffic forwarding mode.	"routing" "wire"	"routing"	"Configuring Traffic Forwarding on a cSRX Container Firewall Container" on page 42

Table 7: Summary of cSRX Container Firewall 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
CSRX_CTRL_CPU	CPU mask, indicating which CPU is running the cSRX Container Firewall control plane daemons (such as nsd, mgd, nstraced, utmd, and so on).	<i>hex value</i>	No CPU affinity	Configuring CPU Affinity for a cSRX Container
CSRX_DATA_CPU	CPU mask, indicating which CPU is running the cSRX Container Firewall data plane daemon (srxpfe).	<i>hex value</i>	No CPU affinity	Configuring CPU Affinity for a cSRX Container
CSRX_ARP_TIMEOUT	ARP entry timeout value for the control plane ARP learning or response.	<i>decimal value</i>	Same as the Linux host	"Configuring Traffic Forwarding on a cSRX Container Firewall Container" on page 42
CSRX_NDP_TIMEOUT	NDP entry timeout value for the control plane NDP learning or response.	<i>decimal value</i>	Same as the Linux host	"Configuring Traffic Forwarding on a cSRX Container Firewall Container" on page 42

Table 7: Summary of cSRX Container Firewall Environment Variables (Continued)

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

Changing the Size of a cSRX Container Firewall Container

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

[Table 8 on page 41](#) compares the scale requirements of a cSRX Container Firewall instance depending on the specified size.

Table 8: cSRX Container Firewall Size Comparison

Specification	cSRX Container Firewall: Small Size	cSRX Container Firewall: Middle Size	cSRX Container Firewall: Large Size (Default)
Physical Memory Overhead	256M	1G	4G

Table 8: cSRX Container Firewall Size Comparison (Continued)

Specification	cSRX Container Firewall: Small Size	cSRX Container Firewall: Middle Size	cSRX Container Firewall: Large Size (Default)
Number of Flow Sessions	8K	64K	512K

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

For example, to launch a cSRX Container Firewall 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 Firewall Container

IN THIS SECTION

- [Configuring Routing Mode | 43](#)
- [Configuring Secure-Wire Mode | 47](#)

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

NOTE: The cSRX Container Firewall 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 Firewall container in routing mode, the cSRX Container Firewall 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 Firewall 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 Container Firewall instance in routing mode with 17 interfaces:

```
root@csrx-ubuntu3:~/csrx# docker run -d --privileged --net=none -e CSRX_PORT_NUM=17
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 Container Firewall. Unless all the interfaces are added to the bridge or the macvlan networks, the PFE will not be launched on the cSRX Container Firewall, 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 Container Firewall to run in static route forwarding mode.

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

1. Launch the cSRX Container Firewall 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. Log into cSRX Container Firewall instance and start configuration mode.

```
root@csrx# cli
root@csrx> configure
[edit]
```

3. Configure interfaces.

Starting from 19.2R1.8, each cSRX Container Firewall can be configured with up to 15 revenue interfaces: eth1, eth2, and so on, until eth15. The number of interfaces can be predefined while booting up a cSRX Container Firewall. Usually, management IP on a cSRX Container Firewall is assigned by docker based on network settings while spinning the cSRX Container Firewall(--network=mgt_bridge). If you don't specify this variable, docker is going to assign IP from default docker network bridge.

The eth0 is used by the cSRX Container Firewall for out-of-band management to the accept management sessions and traffic, and eth1 and eth2 are both used by the cSRX Container Firewall as the two revenue ports to process in-band data traffic (the ge-0/0/0 and ge-0/0/1 interfaces).

For this example, assume that the docker default or the custom network management bridge is 172.31.21.0/24, docker assigns one IP address from this network. If your cSRX Container Firewall is the first container on the system, cSRX Container Firewall will get assigned with 172.31.21.2 and default gateway for the cSRX Container Firewall management plane going to be 172.31.21.1.

Table 9: IP Address Assignment for Interfaces

Interface	IP Address
Management Interface eth0 (fxp0)	172.31.21.1
Default gateway for the cSRX Container Firewall management plane	172.31.21.2
Eth1 (ge-0/0/0)	172.19.0.2/24
Eth2 (ge-0/0/1)	172.20.0.2/24
External Server	10.10.10.0

```
root@csrx# show | display set
```

```
root@csrx# set interfaces ge-0/0/0 unit 0 family inet address 172.19.0.2/24
```

```
root@csrx# set interfaces ge-0/0/1 unit 0 family inet address 172.20.0.2/24
```

4. Configure static routes.

Configure static route and specify next-hop address.

```
root@csrx# set routing-options static route 0.0.0.0/0 next-hop 172.19.0.2/24
```

5. 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
172.19.0.2       perm    0 172.19.0.10      locl   2006    1
172.19.0.10      perm    0 172.19.0.10      ucast  5501    1
1.255.255.255    perm    0              bcst   2007    1
1/8              perm    0              rslv   2009    1172.20.0.2
perm    0 172.20.0.2      locl   2001    1
172.20.0.10      perm    0 172.20.0.10      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
172.31.21.2/28    perm    0 172.20.0.10      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
```

6. 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 172.31.21.1
```

```
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
172.19.0.2	0.0.0.0	255.0.0.0	U	0	0	0 tap1
172.20.0.2	0.0.0.0	255.0.0.0	U	0	0	0 tap0
172.31.21.2	1.0.0.10	255.255.255.240	UG	0	0	0 tap1
10.10.10.0	172.31.21.1	255.255.255.0	UG	0	0	0 eth0
172.21.0.0	0.0.0.0	255.255.0.0	U	0	0	0 eth0

7. If required for your network environment, you can configure an IPv6 static route for the cSRX Container Firewall using the `set routing-options rib inet6.0 static route 3000::0/64 next-hop 1000::10/128` 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;
```

```
}
```

8. 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 srpxpfe. You can view ARP entries by using the Linux `arp` shell command.

NOTE: While there are multiple interfaces created inside the cSRX Container Firewall container, only two interfaces, `ge-0/0/0` and `ge-0/0/1`, are visible in `srpxpfe`.

```
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 Firewall 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 Container Firewall 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 Container Firewall to run in secure-wire forwarding mode.

NOTE: When you launch the cSRX Container Firewall container in secure-wire mode, the cSRX Container Firewall 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 Container Firewall 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>
```

Configuring CPU Affinity for a cSRX Container Firewall Container

A cSRX Container Firewall instance requires two CPU cores in the Linux server. To help schedule the Linux server tasks and adjust performance of the cSRX Container Firewall container running on a Linux host, you can launch the cSRX Container Firewall container and assign its control and data processes (or daemons) to a specific CPU. In a cSRX Container Firewall 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 Container Firewall control and data plane daemons are pinned to a specific physical CPU, which can improve the cSRX Container Firewall container performance by using the CPU cache efficiently. By default, there is not a defined CPU affinity for the cSRX Container Firewall 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 Firewall 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 Firewall 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 Firewall 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 Firewall container. If, at a future point, the cSRX Container Firewall 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 Firewall container when the cSRX Container Firewall is launched.

To configure the cSRX Container Firewall 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 Container Firewall Containers

IN THIS SECTION

- [Pausing/Resuming Processes within a cSRX Container Firewall Container | 49](#)
- [Viewing Container Processes on a Running cSRX Container Firewall Container | 50](#)
- [Removing a cSRX Container Firewall Container or Image | 50](#)

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

Pausing/Resuming Processes within a cSRX Container Firewall Container

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

To pause and restart a cSRX Container Firewall container:

1. Use the `docker pause` command to suspend all processes in a cSRX Container Firewall container.

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

2. Use the `docker unpause` command to resume all processes in the cSRX Container Firewall container.

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

Viewing Container Processes on a Running cSRX Container Firewall Container

Use the `docker exec` command to view the details of the processes (applications, services and status) running on a cSRX Container Firewall 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 Firewall Container or Image

To remove a cSRX Container Firewall container or image:

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

1. Use the `docker stop` command to stop the cSRX Container Firewall container.

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

2. Use the `docker rm` command to remove the cSRX Container Firewall container.

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

NOTE: Include `--force` to force the removal of a running cSRX Container Firewall container.

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

NOTE: Include `--force` to force the removal a cSRX Container Firewall image.

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

RELATED DOCUMENTATION

[Docker Engine User Guide](#)

[Docker commands](#)

4

CHAPTER

Configuring cSRX Container Firewall

cSRX Container Firewall Configuration and Management Tools | 53

Configuring cSRX Container Firewall Using the Junos OS CLI | 54

cSRX Container Firewall Configuration and Management Tools

IN THIS SECTION

- [Understanding the Junos OS CLI and Junos Scripts | 53](#)
- [Understanding cSRX Container Firewall with Contrail and Openstack Orchestration | 53](#)

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 Firewall container.

Understanding cSRX Container Firewall with Contrail and Openstack Orchestration

The cSRX Container Firewall 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 Container Firewall 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 Container Firewall 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 Firewall container as a large-scale security service in a multicloud environment, and configure the cSRX Container Firewall 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 Container Firewall with Juniper Networks Contrail.

RELATED DOCUMENTATION

[Introducing the Junos OS Command-Line Interface](#)

[Contrail Networks](#)

[Mastering Junos Automation Programming](#)

Configuring cSRX Container Firewall Using the Junos OS CLI

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

To configure the cSRX Container Firewall container using the Junos OS CLI:

1. Launch the cSRX Container Firewall container. Use the `docker run` command to launch the cSRX Container Firewall container. You include the `mgt_bridge` management bridge to connect the cSRX Container Firewall to a network.

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

For example, to launch `csrx2` using cSRX Container Firewall software image `csrx:18.21R1.9` enter:

```
root@csrx-ubuntu3:~/csrx# docker run -d --privileged --network=mgt_bridge -e --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 Firewall container to run in privileged mode.

2. Log in to the cSRX Container Firewall container using SSH which is accessed by cSRX Container Firewall exposed service port.

```
root@csrx-ubuntu3:~/csrx#ssh -p 30122 root@192.168.42.81
```

3. Start the CLI as root user.

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

4. 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
```

5. Enter configuration mode.

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

6. 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
```

7. Configure the hostname.

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

8. Configure the two traffic interfaces.

```
[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
```

9. 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
```

10. Verify the configuration.

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

11. Commit the configuration to activate it on the cSRX Container Firewall instance.

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

12. (Optional) Use the `show` command to display the configuration for verification.

RELATED DOCUMENTATION

[Junos OS for SRX Series](#)

[Introducing the Junos OS Command-Line Interface](#)