

cSRX Deployment Guide for Contrail

Published
2020-03-27

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 Deployment Guide for Contrail
Copyright © 2020 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 the Documentation | vi

Documentation and Release Notes | vi

Documentation Conventions | vi

Documentation Feedback | ix

Requesting Technical Support | ix

Self-Help Online Tools and Resources | x

Creating a Service Request with JTAC | x

1

Overview

Understanding cSRX with Contrail | 12

cSRX Overview | 12

cSRX Benefits and Uses | 15

Docker Overview | 16

Juniper Networks Contrail Overview | 17

cSRX Scale-Up Performance | 20

Junos OS Features Supported on cSRX | 21

Supported SRX Series Features on cSRX | 21

SRX Series Features Not Supported on cSRX | 24

2

cSRX Service Chaining in Contrail

Requirements for Deploying cSRX on Contrail | 32

Platform and Server Requirements | 32

cSRX Basic Configuration Settings | 33

Service Chains Overview | 34

Understanding Service Chains | 34

Service Chain Modes | 35

Components of a Service Chain | 35

Service Templates | 35

Virtual Networks | 36

Service Instances | 36

Network Policies | 36

Preparing a Contrail Cluster | 36

Configuring cSRX in a Contrail Service Chain | 39

- Before You Begin | 39
- Configuring the Docker Registry and Compute Node | 40
- Creating an Availability Zone for the cSRX Container | 42
- Importing the cSRX Image | 45
- Creating Virtual Networks in Contrail | 48
- Launching the cSRX Container | 52
- Creating a Service Template for the cSRX | 54
- Creating and Launching the Service Instance | 56
- Creating a Network Policy (Optional) | 58
- Adding a Network Policy to a Virtual Network (Optional) | 59

3

Managing cSRX Containers in Contrail

cSRX Configuration Data File and Environment Variables | 62

- Openstack User Data File | 63
- Openstack Metadata | 65

Specifying an Initial Root Password for Logging into a cSRX Container | 65

Configuring cSRX for Routing Mode | 66

Changing the Size of a cSRX Container | 69

Specifying the Packet I/O Driver for a cSRX Container | 70

- Specifying the Poll Mode Driver | 71
- Specifying the Interrupt Mode Driver | 72

Configuring CPU Affinity for a cSRX Container | 72

Managing cSRX Containers | 73

- Powering On the cSRX Container from OpenStack CLI | 74
- Powering On the cSRX Container from OpenStack Dashboard | 74
- Pausing the cSRX Container from OpenStack CLI | 74
- Pausing the cSRX Container from OpenStack Dashboard | 75
- Restarting the cSRX Container from OpenStack CLI | 75

Restarting the cSRX Container from OpenStack Dashboard | 75

Deleting the cSRX Container from OpenStack CLI | 75

Deleting the cSRX Container from Contrail | 76

Monitoring Basic cSRX Statistics with the Contrail Monitor | 76

Configuring cSRX

Configuring cSRX Using the Junos OS CLI | 78

About the Documentation

IN THIS SECTION

- Documentation and Release Notes | vi
- Documentation Conventions | vi
- Documentation Feedback | ix
- Requesting Technical Support | ix

Use this guide to install and configure the cSRX Container Firewall as a dedicated compute node in a Contrail service chain. 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 vii 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 vii 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
<i>Italic text like this</i>	<ul style="list-style-type: none"> Introduces or emphasizes important new terms. Identifies guide names. Identifies RFC and Internet draft titles. 	<ul style="list-style-type: none"> A policy <i>term</i> is a named structure that defines match conditions and actions. <i>Junos OS CLI User Guide</i> RFC 1997, <i>BGP Communities Attribute</i>

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

Convention	Description	Examples
<i>Italic text like this</i>	Represents variables (options for which you substitute a value) in commands or configuration statements.	Configure the machine's domain name: [edit] root@# set system domain-name <i>domain-name</i>
Text like this	Represents names of configuration statements, commands, files, and directories; configuration hierarchy levels; or labels on routing platform components.	<ul style="list-style-type: none"> 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.
< > (angle brackets)	Encloses optional keywords or variables.	stub <default-metric <i>metric</i> >;
(pipe symbol)	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.	broadcast multicast (<i>string1</i> <i>string2</i> <i>string3</i>)
# (pound sign)	Indicates a comment specified on the same line as the configuration statement to which it applies.	rsvp { # Required for dynamic MPLS only
[] (square brackets)	Encloses a variable for which you can substitute one or more values.	community name members [<i>community-ids</i>]
Indentation and braces ({ })	Identifies a level in the configuration hierarchy.	[edit] routing-options { static { route default { nexthop <i>address</i> ; retain; } } }
;(semicolon)	Identifies a leaf statement at a configuration hierarchy level.	

GUI Conventions

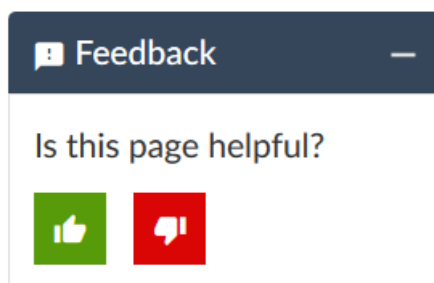
Table 2: Text and Syntax Conventions (*continued*)

Convention	Description	Examples
Bold text like this	Represents graphical user interface (GUI) items you click or select.	<ul style="list-style-type: none"> 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 Contrail | 12

Junos OS Features Supported on cSRX | 21

Understanding cSRX with Contrail

IN THIS SECTION

- [cSRX Overview | 12](#)
- [cSRX Benefits and Uses | 15](#)
- [Docker Overview | 16](#)
- [Juniper Networks Contrail Overview | 17](#)
- [cSRX Scale-Up Performance | 20](#)

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 Contrail 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 deploys as a single container on a Docker Engine compute node running in a Contrail cluster. It runs on a Linux bare-metal server as the hosting platform for the Docker container environment. The cSRX container packages all of the dependent processes (or daemons) and libraries to support the different Linux host distribution methods (Ubuntu, Red Hat Enterprise Linux, or CentOS). 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 processes (or 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, ApplID, and so on). srpxpfe is the data-plane daemon

that receives and sends packets from the two revenue ports of a cSRX container. The cSRX uses srxpfe for Layer 2 through 3 forwarding functions 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 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. The cSRX container either allows the frames to pass through unaltered or drops the frames, based on the configured security policies.

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>
```

Figure 1 on page 13 illustrates the cSRX operating in secure-wire mode.

Figure 1: cSRX in Secure-Wire Mode

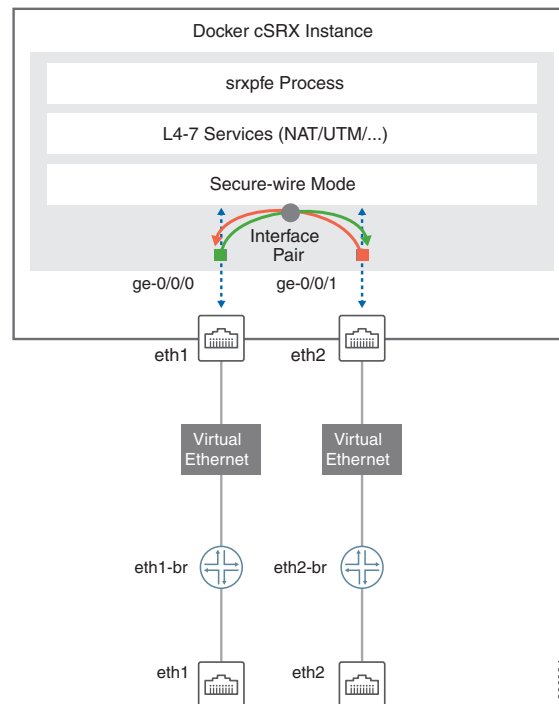
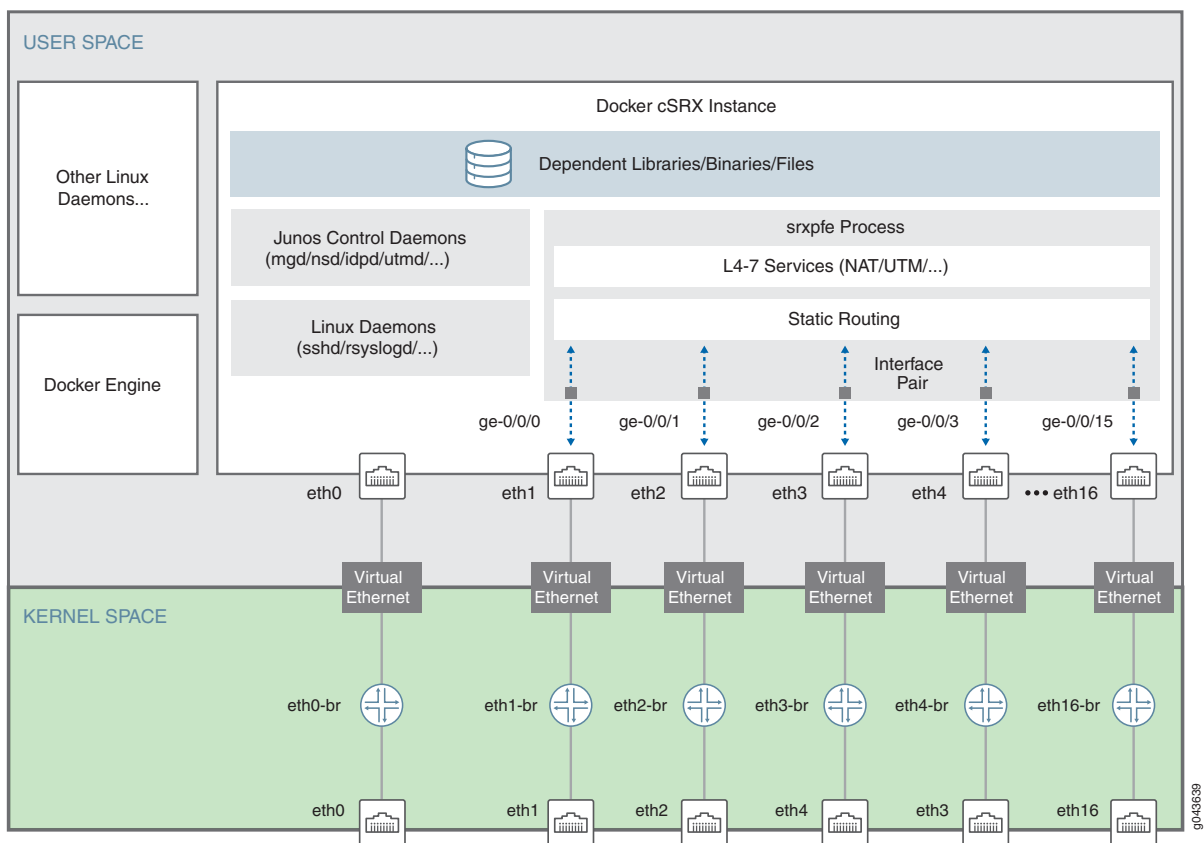


Figure 2 on page 14 illustrates a high-level view of a cSRX container instance in routing mode. It is an example of how a cSRX container is bridged with an external network. In this illustration, cSRX eth1 is bridged with host physical NIC eth1 and cSRX eth2 is bridged with host physical NIC eth2.

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, and the other two virtual networks to receive and transmit in-band data traffic. See [“Configuring cSRX in a Contrail Service Chain” on page 39](#).

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

Figure 2: cSRX Container Overview



Starting in Junos OS Release 19.2R1, in routing mode, the default number of interfaces supported are three and maximum of 16 interfaces (1 management and 16 data interfaces). Also, you must configure zones explicitly and add interfaces to zones, no zones or interfaces are be mapped statically. If the interfaces configured are less than what is supported, then the PFE will not be launched.

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 21](#) 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

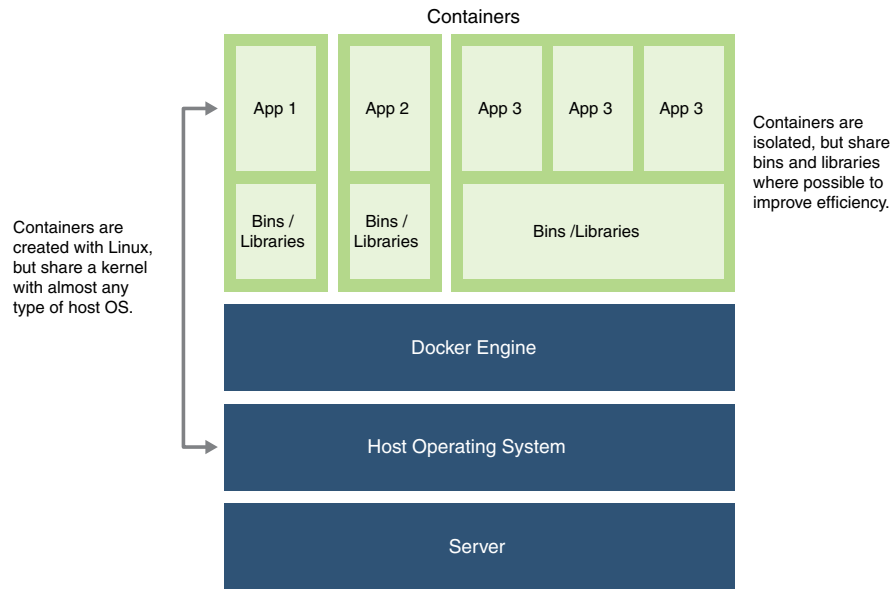
Docker is an open source software platform that simplifies the creation, management, and teardown of a virtual container that can run on any Linux server. A Docker container is an open source software development platform, with its main benefit being to package applications in “containers” to allow them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



Juniper Networks Contrail Overview

Juniper Networks Contrail is an open, standards-based software-defined networking (SDN) platform that delivers network virtualization and service automation for federated cloud networks. It provides self-service provisioning and improves network troubleshooting and diagnostics. It also enables service chaining for dynamic application environments across an enterprise virtual private cloud (VPC), managed Infrastructure as a Service (IaaS), and Network Functions Virtualization (NFV) use cases. You can use Contrail with open cloud orchestration systems such as OpenStack or CloudStack to instantiate instances of cSRX in a containerized environment.

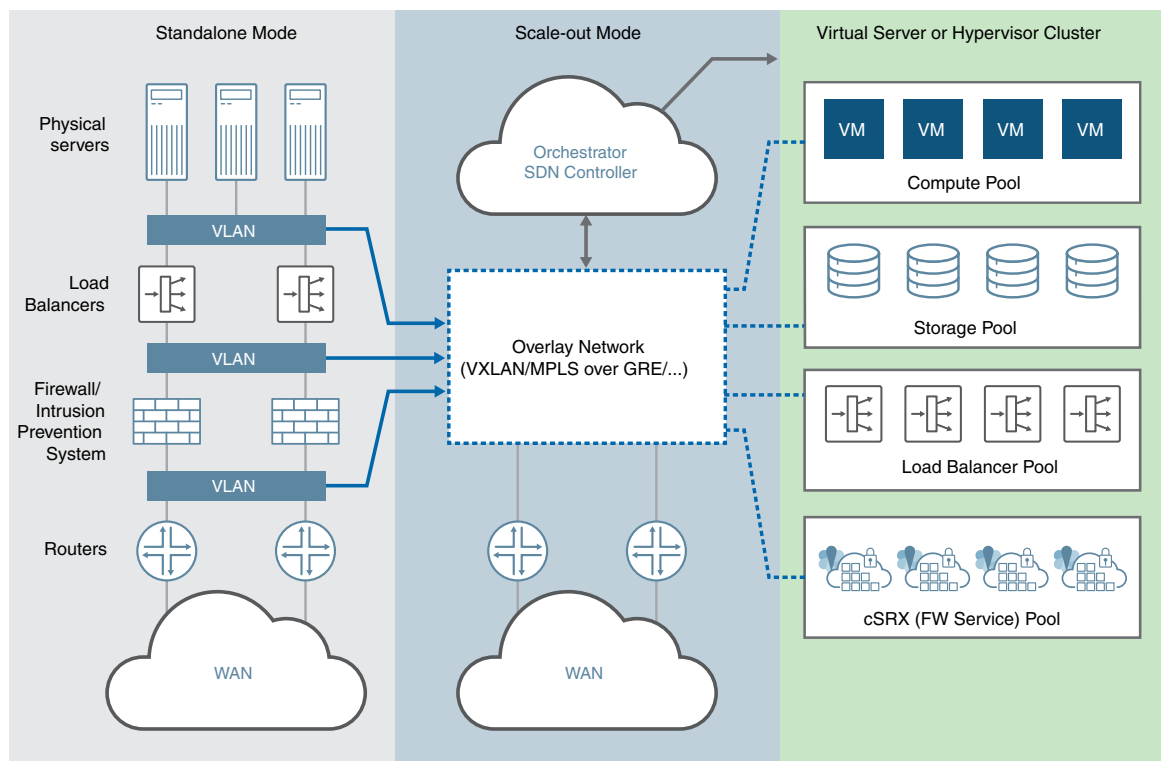
The cSRX Container Firewall can be deployed on a Docker Engine compute node as a dedicated firewall in the Contrail Networking cloud environment to provide differentiated Layer 4 through 7 security services for multiple tenants as part of a service chain. With the Contrail orchestrator, cSRX is deployed as a large scale security service, and is configured to steer traffic from vRouter with vRouter interface (VIF). Traffic and health statistics are monitored by the Contrail service orchestrator.

When you deploy the cSRX container in a Contrail Networking cloud environment, Contrail:

- Stores the cSRX image in an Openstack glance service so it can be used to launch new service instances.
- Distributes the cSRX image to different computer nodes from the Docker registry service.
- Launches the cSRX container with the Openstack Nova service.
- Spawns a cSRX service chain in the Contrail controller with a service template, virtual network, service instance, and network policy.
- Monitors cSRX container status and statistics in the Contrail service orchestrator.

Figure 4 on page 18 illustrates how the cSRX Container Firewall provides security services in a Contrail Networking cloud environment.

Figure 4: cSRX Container Firewall in an SDN Environment



Contrail is logically centralized and behaves as a single logical unit, despite the fact that it is implemented as a cluster of multiple nodes. Contrail Networking nodes include:

- **Contrail Control Nodes**—Control nodes implement the logically centralized portion of the control plane. Control nodes are responsible for the routing control plane, configuration management, analytics, and UI.
- **Contrail Compute Nodes**—Compute nodes are general-purpose virtualized servers that host VMs and containers (such as the cSRX). These VMs can be tenants running general applications, or these VMs can be service VMs running network services such as a virtual load balancer or virtual firewall. Each

compute node contains a vRouter that implements the forwarding plane and the distributed part of the control plane. Compute nodes are responsible for managing the data plane.

In the Contrail network, a compute node is a general-purpose x86 server that hosts VMs and containers running applications such as Web servers, database servers, enterprise applications or hosting virtualized services used to create service chains. You install and configure the Docker Engine on at least one compute node to implement the Linux container environment, and the cSRX container is installed on the compute node that is running the Docker Engine. A cSRX service pool can consist of multiple Docker Engine compute nodes.

The control node hosts the Docker registry, Openstack Controller, and Contrail Controller to orchestrate the virtual services. The cSRX image is automatically pulled from the Docker registry to the Docker Engine compute node when a cSRX instance is initially launched. The cSRX compute node communicates with the control node to receive instructions to start each cSRX container and to set up the service chain by inserting the cSRX network interface into the different vRouter virtual networks in the Contrail cluster. Virtual networks can be shared across different tenants.

Figure 5 on page 19 illustrates the role of the cSRX Container Firewall in a Contrail Networking cloud environment and how it provides multitenancy.

Figure 5: cSRX Service in Contrail Networking Cloud Environment

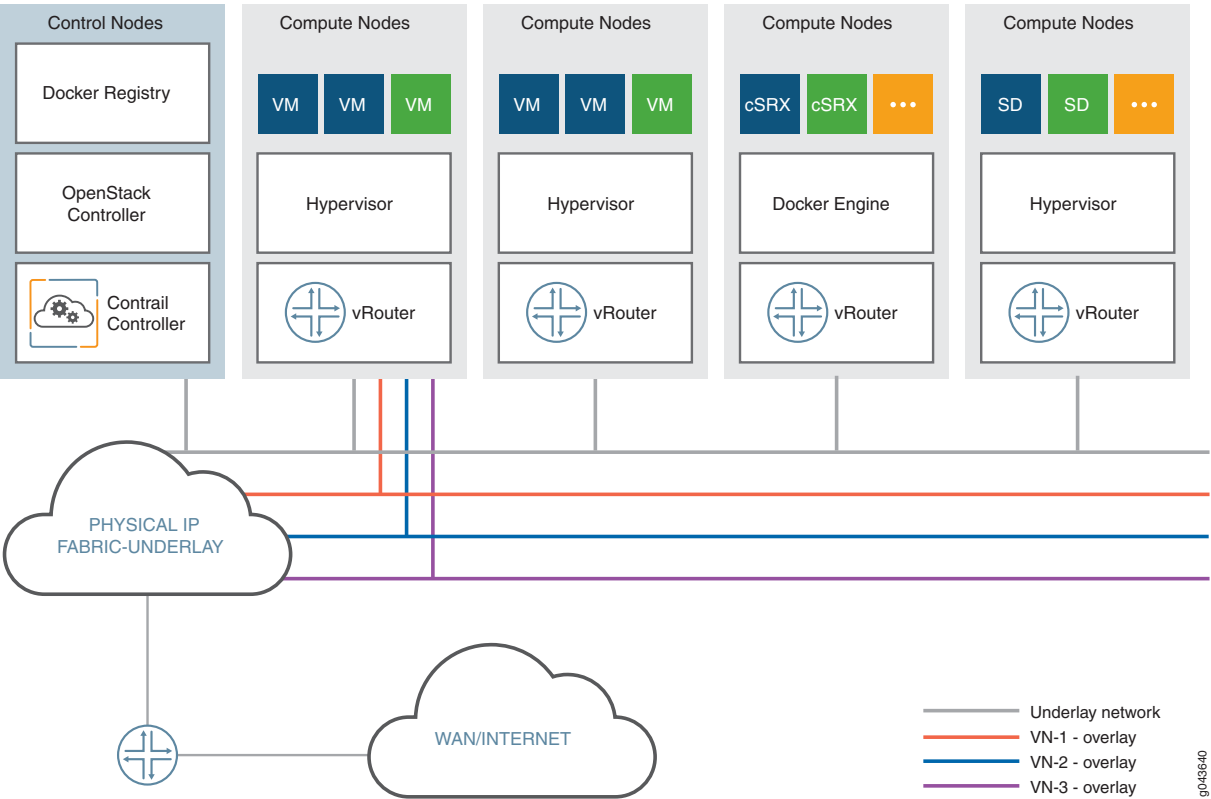
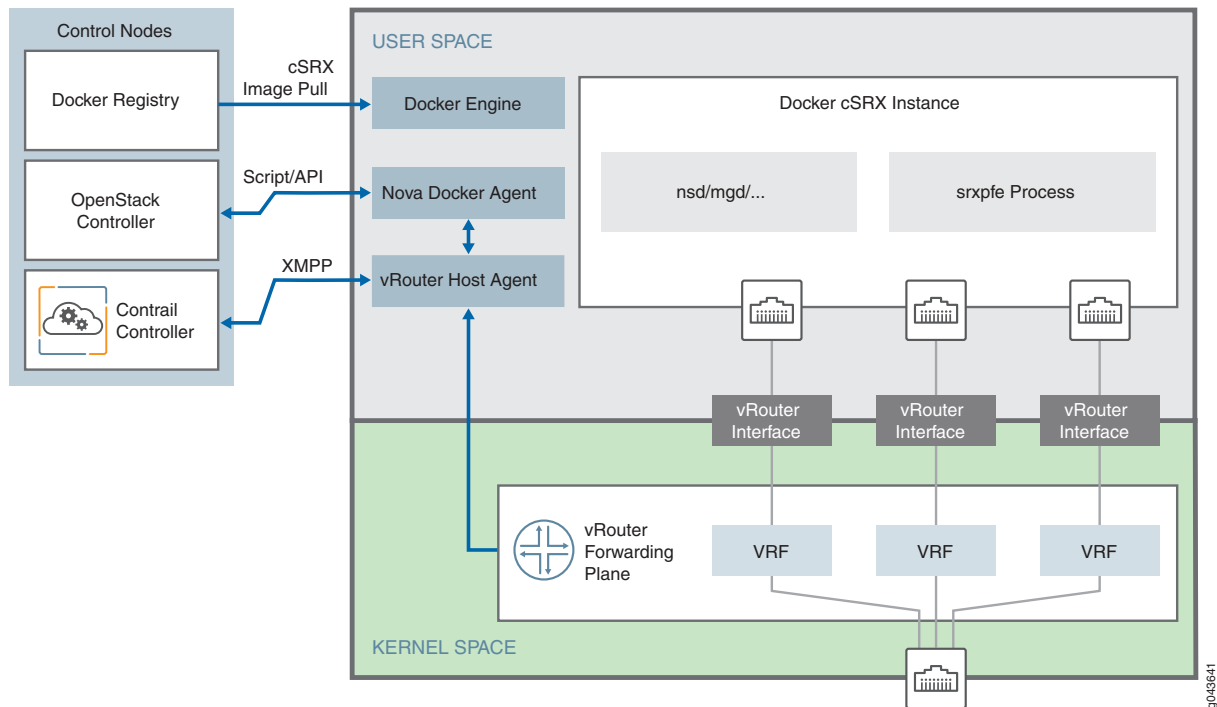


Figure 6 on page 20 illustrates the cSRX compute nodes in a Contrail Networking cloud environment.

A dedicated Nova Docker Agent runs on the cSRX compute node to receive instructions from the Openstack Controller and to act on behalf of the cSRX. When the Nova Docker Agent starts the cSRX container, the agent will first check if the cSRX image is located in the local host Docker Engine. If not, the agent will then attempt to pull the cSRX image from the remote Docker registry. Once a cSRX container starts, the Nova Docker Agent also creates a vRouter interface (VIF) for the cSRX container and “plugs” the VIF to the different virtual routing and forwarding (VRF) instances of the vRouter according to the service template’s virtual network configuration.

Figure 6: cSRX Compute Node



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 21](#) 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 69](#) 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

- Supported SRX Series Features on cSRX | 21
- SRX Series Features Not Supported on cSRX | 24

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 22 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
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 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). Network Interfaces
Intrusion Detection and Prevention (IDP)	<p>For SRX Series IPS configuration details, see:</p> 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	

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

Feature	Considerations
Network Address Translation (NAT)	<p>Includes support for all NAT functionality on the cSRX platform, such as:</p> <ul style="list-style-type: none"> • Source NAT • Destination NAT • Static NAT • Persistent NAT and NAT64 • NAT hairpinning • NAT for multicast flows <p>For SRX Series NAT configuration details, see:</p> <p>Introduction to NAT</p>
Routing	<p>Basic Layer 3 forwarding with VLANs.</p> <p>Layer 2 through 3 forwarding functions: secure-wire forwarding or static routing forwarding</p>
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	<p>Includes support for all user firewall functionality on the cSRX 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>

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

Feature	Considerations
Unified Threat Management (UTM)	<p>Includes support for all UTM functionality on the cSRX platform, such as:</p> <ul style="list-style-type: none"> • Antispam • Sophos Antivirus • Web filtering • Content filtering <p>For SRX Series UTM configuration details, see:</p> <p>Unified Threat Management Overview</p> <p>For SRX Series UTM antispam configuration details, see:</p> <p>Antispam Filtering Overview</p>
Zones and zone-based IP spoofing	Understanding IP Spoofing

SRX Series Features Not Supported on cSRX

Table 5 on page 24 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	
	<p>Layer 2 enforcement in UAC deployments</p> <p>NOTE: UAC-IDP and UAC-UTM also are not supported.</p>
Class of Service	
	High-priority queue on SPC
	Tunnels

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

	SRX Series Feature
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
	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	

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

	SRX Series Feature
	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 multisuppliant support
	Interleaving using MLFR
	PoE
	PPP interface
	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

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

	SRX Series Feature
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
	High availability
	J-Web
	Logical systems
	MPLS
	Outbound SSH
	Remote instance access
	RESTCONF
	Sky ATP
	SNMP
	Spotlight Secure integration

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

	SRX Series Feature
	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	
	BGP extensions for IPv6
	BGP Flowspec
	BGP route reflector
	Bidirectional Forwarding Detection (BFD) for BGP
	C RTP
Switching	
	Layer 3 Q-in-Q VLAN tagging

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

	SRX Series Feature
Unsupported System Logs and Real-Time log functions	<p>cSRX does not support all the log functions supported on other SRX devices or vSRX instances due to limited CPU power and disk capacity.</p> <p>Unsupported system logs and real-time log functions on cSRX 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	
	UTM
Unified Threat Management	
	Express AV
	Kaspersky AV
Upgrading and Rebooting	
	Autorecovery
	Boot instance configuration
	Boot instance recovery
	Dual-root partitioning
	OS rollback
User Interfaces	
	NSM
	SRC application

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

	SRX Series Feature
	Junos Space Virtual Director
Application Security	
	SSL proxy

2

CHAPTER

cSRX Service Chaining in Contrail

Requirements for Deploying cSRX on Contrail | **32**

Service Chains Overview | **34**

Preparing a Contrail Cluster | **36**

Configuring cSRX in a Contrail Service Chain | **39**

Requirements for Deploying cSRX on Contrail

IN THIS SECTION

- Platform and Server Requirements | 32
- cSRX Basic Configuration Settings | 33

This section presents an overview of requirements for deploying a cSRX container on Contrail:

Platform and Server Requirements

Table 6 on page 32 lists the Contrail platform requirement specifications and Table 7 on page 33 lists the server requirements for deploying a cSRX container in a compute node.

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

Table 6: Platform Requirements

Component	Operating System and Kernel Versions
Contrail Release	Contrail 3.2
OpenStack Release	
● OpenStack Liberty	<ul style="list-style-type: none">● CentOS 7.2—Linux Kernel version-3.10.0-327.10.1● Ubuntu 14.04.4—Linux kernel versions 3.13.0-85-generic and 4.4.0-34-generic● Red Hat 7.2—Linux Kernel version- 3.10.0-327.10.● VMware vCenter 5.5, 6.0—Ubuntu 14.04.4 kernel version 3.13.0-85-generic
● OpenStack Mitaka	<ul style="list-style-type: none">● CentOS 7.2—Linux kernel version 3.10.0-327.10.1● Ubuntu 14.04.4—Linux kernel version 3.13.0-85-generic

The cSRX container compute node requirements should be same as the other compute nodes running in Contrail cloud. Each server must have the minimum requirements outlined in Table 7 on page 33.

Table 7: Server Requirement Specifications

Component	Specification	Release Introduced
Docker Engine	Docker Engine 1.9 or later installed on the same compute node as the cSRX	Junos OS Release 18.1R1
vCPUs	2 CPU cores	
Memory	8 GB	
Disk space	40 GB hard drive	
Network interface	<p>1 Ethernet port (minimum)</p> <p>cSRX container includes 5 network interfaces (eth0, eth1, eth2, eth3, and eth4), and requires that you create 5 virtual interfaces and attach those interfaces to a virtual network.</p> <p>One interface is intended for out-of-band management to accept management sessions and traffic, and the other four interfaces are used by the cSRX as the revenue ports to process in-band data traffic.</p>	Support for additional revenue ports are eth 3 and eth4 is added in Junos OS Release 19.2R1.

cSRX Basic Configuration Settings

The cSRX container requires the following basic configuration settings:

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

By default, interface ge-0/0/0 is bound to the untrust security zone (eth2) and interface ge-0/0/1 is bound to the trust security zone (eth1).

Service Chains Overview

IN THIS SECTION

- [Understanding Service Chains | 34](#)
- [Service Chain Modes | 35](#)
- [Components of a Service Chain | 35](#)

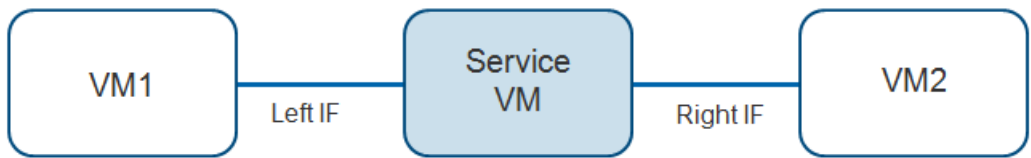
You can use Contrail to chain various Layer 2 through Layer 7 services such as firewall, NAT, and IDP through one or more cSRX containers. For example, you can insert a cSRX firewall container between two other virtual machines (VMs, containers, or both). By using cSRX and service chains, you can tailor your security needs to a targeted virtual network and container set. This provides agility and scalability in line with the fluidity of cloud network environments.

Understanding Service Chains

To create a service through cSRX, you instantiate one or more cSRX containers to dynamically apply single or multiple services to network traffic.

[Figure 7 on page 34](#) shows a basic service chain with a single cSRX container. The cSRX service container spawns a service, such as a firewall. The left interface (left IF) points to the internal end customer, who uses the service; and the right interface (right IF) points to the external network or Internet. You can also instantiate multiple cSRX containers to chain multiple services together. For example, you could add an IDP service after the firewall.

Figure 7: cSRX Service Chaining



When you create a service chain, Contrail creates tunnels across the underlay network that span all services in the chain.

Service Chain Modes

You can configure the following service modes:

- In-network or routed mode—Provides a gateway service that routes packets between the service instance interfaces. Examples include NAT, Layer 3 firewall, and load balancing.
- In-network-nat mode—Similar to in-network mode; however, packets from the left (private) network are not routed to the right (public) source network. In-network-nat mode is particularly useful for NAT services.

NOTE: Ensure that you define the service policy with the private network on the left and public on the right to get the public routes (usually the default) advertised into the left network.

Components of a Service Chain

Service chaining requires the following configuration components to build the chain:

- Service template
- Virtual networks
- Service instance
- Network policy

Service Templates

Service templates map out the basic configuration that Contrail uses to instantiate a service instance or container. Within Contrail, you configure service templates in the scope of a domain, and you can use the templates on all projects within a domain. You can use a template to launch multiple service instances of the same type in different projects within a domain. Within a service template, you select the service mode, a cSRX image name for the container that will provide the service, and an ordered list of interfaces for the service. cSRX service containers require the management interface to be the first interface in that ordered list. The service template launches the cSRX as part of the service chain. A dedicated Nova Docker Agent runs on the cSRX compute node to receive instructions from the Openstack Controller and to act on behalf

of the cSRX. When the Nova Docker Agent starts the cSRX container, the agent will first check if the cSRX image is located in the local host Docker Engine. If not, the agent will then attempt to pull the cSRX image from the remote Docker registry.

Virtual Networks

Virtual networks provide the link between the service instance and the network traffic in the containerized environment. You can create the virtual networks in Contrail or OpenStack and use those networks to direct traffic to or through the service instance.

Service Instances

A service instance is the instantiation of the selected service template to create one or more containers that provide the service (for example, a firewall). When you create a service instance, you select a service template that defines the instance. You also associate the interfaces in the service template with the virtual networks needed to direct traffic into and out of the service instance. If you enable service scaling in the selected service template, you can instantiate more than one container when you create the service instance.

Network Policies

By default, all traffic in a virtual network remains isolated. You configure a network policy to allow traffic between virtual networks and through the service instance. The network policy filters traffic to and from the service container based on the rules you configure. You select the service instance container and the virtual networks for the right and left interfaces of that container that the network policy applies to. As a final step, you associate the network policy with each virtual network the policy applies to.

RELATED DOCUMENTATION

| [Contrail - Service Chaining](#)

Preparing a Contrail Cluster

Before you can add the cSRX firewall service to Contrail, you must first add a control node and compute node to an existing Contrail cluster and install the Docker Engine on that compute node to support cSRX. This procedure outlines the steps in preparing the Contrail cluster to support cSRX.

If you have not done so already, install the operating system (Ubuntu, CentOS, or Red Hat Enterprise Linux (RHEL)) and Contrail software package (see [Contrail Getting Started Guide \(Contrail 3.2\)](#)).

Optionally, ensure OpenStack Glance is installed (see [OpenStack - Add the Image Service \(glance\)](#)).

To prepare the Contrail cluster:

1. Add a minimum of one control node and two compute nodes to the Contrail cluster. One compute node is running a KVM hypervisor and the other compute node is running the Docker Engine (see [Configuring the Control Node](#) and [Adding or Removing a Compute Node in an Existing Contrail Cluster](#)).

NOTE: The KVM hypervisor and Docker Engine cannot run on the same host.

If using Contrail 3.2, to deploy a compute node to work with Nova Docker in a Contrail cluster, ensure that the Nova DockerDriver is defined in place of the LibvirtDriver in the env.hypervisor dictionary in the **testbed.py** file. See [Configuring Open Stack Nova Docker with Contrail](#).

2. Install and configure the Docker Engine on at least one compute node to implement the Linux container environment. There must be at least one compute node configured with the Docker Engine. Docker installation requirements vary based on the platform and the host OS (Ubuntu, Red Hat Enterprise Linux (RHEL), or CentOS).

See [Install Docker](#) for installation instructions on the different supported Linux host operating systems.

3. Use **Monitor > Infrastructure > Dashboard** to get a view of the system infrastructure components for Contrail cluster status, including the numbers of virtual routers, control nodes, analytics nodes, and configuration nodes that are currently operational. Any of the control nodes, virtual routers, analytics nodes, and configuration nodes can be monitored individually and in detail from the Dashboard by clicking an associated box, and drilling down for more detail.
4. Specify the compute node that is to run the cSRX container as Docker type *Hypervisor* in the **testbed.py** file, under **OPTIONAL COMPUTE HYPERVISOR CHOICE**. See [Setting Up the Testbed Definitions File](#) for the procedure on how to edit the **testbed.py** file.

The following example shows the **testbed.py** file for a Contrail cluster configuration.

```
... ..
#Management ip addresses of hosts in the cluster
host1 = 'root@10.208.29.2'
host2 = 'root@10.208.29.1'
... ..
... ..
#Role definition of the hosts.
env.roledefs = {
```

```

    'all': [host1],
    'cfgm': [host1],
    'openstack': [host1],
    'control': [host1],
    'compute': [host1,host2],
    'collector': [host1],
    'webui': [host1],
    'database': [host1],
    'build': [host_build],
    'storage-master': [host1],
    'storage-compute': [host1],
}
... ..
... ..
#For reimage purpose
env.ostypes = {
    host1:'ubuntu',
    host2:'ubuntu',
}
#env.orchestrator = 'openstack' #other values are 'vcenter', 'none'
default:openstack

#ntp server the servers should point to
#env.ntp_server = 'ntp.juniper.net'

# OPTIONAL COMPUTE HYPERVISOR CHOICE:
#=====
# Compute Hypervisor
env.hypervisor = {
    host2: 'docker',
}
... ..

```

Configuring cSRX in a Contrail Service Chain

IN THIS SECTION

- [Before You Begin | 39](#)
- [Configuring the Docker Registry and Compute Node | 40](#)
- [Creating an Availability Zone for the cSRX Container | 42](#)
- [Importing the cSRX Image | 45](#)
- [Creating Virtual Networks in Contrail | 48](#)
- [Launching the cSRX Container | 52](#)
- [Creating a Service Template for the cSRX | 54](#)
- [Creating and Launching the Service Instance | 56](#)
- [Creating a Network Policy \(Optional\) | 58](#)
- [Adding a Network Policy to a Virtual Network \(Optional\) | 59](#)

This section outlines the steps to install and configure the cSRX Container Firewall as a dedicated compute node in a Contrail service chain. You use Contrail to chain various Layer 2 through Layer 7 services such as firewall, NAT, and IDP through the cSRX containers.

This section includes the following topics:

Before You Begin

Before you deploy the cSRX Container Firewall as an advanced security service in the Contrail Networking cloud environment, ensure that you:

- Review [“Requirements for Deploying cSRX on Contrail” on page 32](#) to verify the Contrail platform requirement specifications and server requirements for deploying a cSRX container in a compute node.
- Install Contrail and prepare the Contrail cluster to support the cSRX (see [“Preparing a Contrail Cluster” on page 36](#)).

Configuring the Docker Registry and Compute Node

This topic describes how to configure the Docker registry on the control node and install the cSRX-Contrail software package on the compute node. The cSRX-Contrail software package automates installation of the necessary software to the control node and compute node required to use the cSRX container.

To configure the Docker registry server and the compute node:

1. Install and configure the Docker Engine on the control node 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).
See [Install Docker](#) for installation instructions on the different supported Linux host operating systems.
2. Copy the cSRX-Contrail software package to the control node and extract it.

```
[root@eng--shell6 ~/contrail_csr]$ ls
```

```
container- srx- contrail.tgz
[root@eng--shell6 ~/contrail_csr]$ pwd
/homes/user/contrail_csr
```

3. On the control node, configure and install the cSRX-Contrail software package.

```
root@ubtvm02:~/container-srx-contrail# ./configure --with-registry --registry-addr 10.208.29.2
```

```
Maximum instances per node:      2
Registry server name:            csrx-registry
Registry server IP address:      10.208.29.2
Registry port number:            5050
Registry user name:              regress
Registry password:               MaRtInI
root@ubtvm02:~/container-srx-contrail# ./install
cp -rf ./etc/csr/* /etc/csr
cp -f bin/csrx-configure-compute /usr/bin/
cp -f bin/csrx-configure-control /usr/bin/
cp -f bin/csrx-gen-cert          /usr/bin/
cp -f bin/csrx-run-registry      /usr/bin/
root@ubtvm02:~/container-srx-contrail# csrx-
csrx-configure-compute csrx-configure-control csrx-gen-cert csrx-run-registry
```

4. Start the Docker registry service on the control node.


```
root@ubtvm02:~/container-srx-contrail# csrx-run-registry
```

```
Stop and remove existing registry ...
...
Starting registry server
..
root@ubtvm02:~/container-srx-contrail# csrx-configure-control
Start to configure docker registry client....
..
..
Login Succeeded
```

5. Configure the cSRX compute nodes.

NOTE: `csrx-configure-compute` enables the cSRX compute nodes to receive information from `/opt/contrail/utils/fabfile/testbed/testbed.py` and to configure them automatically.

```
root@ubtvm02:~/container-srx-contrail# csrx-configure-compute
```

```
[info][10.208.29.1]:Uploading file /etc/csr/novadocker.patch
[info][10.208.29.1]:Uploading file /etc/csr/profile
[info][10.208.29.1]:Uploading file /etc/csr/csr-configure-local
[info][10.208.29.1]:Uploading file /etc/csr/cert/csr-registry.crt
[info][10.208.29.1]:chmod u+x /usr/bin/csr-configure-local
[info][10.208.29.1]:/usr/bin/csr-configure-local

[info][10.208.29.1]:/usr/bin/csr-configure-local
Start to configure docker registry client....
Updating certificates in /etc/ssl/certs... 0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d....done.
docker stop/waiting
docker start/running, process 25081

[info][10.208.29.1]:Configure registry client done.
Now Login to csrx-registry:5050
docker login -u regress -p MaRtInI -e xxx@juniper.net csrx-registry:5050
WARNING: login credentials saved in /root/.dockercfg.
Login Succeeded
Start to configure huge page ...
```

```
...
...
Patching novadocker ...
patching file driver.py
/usr/bin/csr-x-configure-local done!
```

Creating an Availability Zone for the cSRX Container

Contrail creates a separate Nova availability zone (nova/docker) for compute nodes deployed with DockerDriver. An availability zone is an aggregate of the compute nodes running Docker services. You can add the compute nodes to availability zones that are running the Docker services. In this case, an availability zone is required for the cSRX container to start the cSRX services. When launching the cSRX, you specify it by the availability zone to start the container for the cSRX service.

NOTE: An availability zone is necessary only when your environment includes a mixture of KVM and Docker computer nodes.

This topic outlines how to create a new availability zone for the cSRX, and then to add the compute node to the availability zone. Availability zone can be created with the **nova** command or from the OpenStack Dashboard (Horizon).

To create an availability zone for cSRX using the **nova** commands:

1. Create a host aggregate that is exposed as the availability zone using the **nova aggregate-create** command.

```
root@ubtvm02:~# source /etc/contrail/openstackrc
```

```
root@ubtvm02:~# nova aggregate-create aggregate-docker az-docker
```

```
root@ubtvm02:~# .....
```

```
root@ubtvm02:~# nova aggregate-list
```

```
+-----+-----+-----+
| Id | Name | Availability Zone |
+-----+-----+-----+
```

```
| 1 | aggregate-docker | az-docker |
+---+-----+-----+-----+
```

2. Add a host to the host aggregate using the **nova aggregate-add-host** command.

```
root@ubtvm02:~# nova aggregate-add-host 1 ubtvm01
```

```
Aggregate 1 has been successfully updated.
```

3. Check the availability zone with the **nova aggregate-details** command.

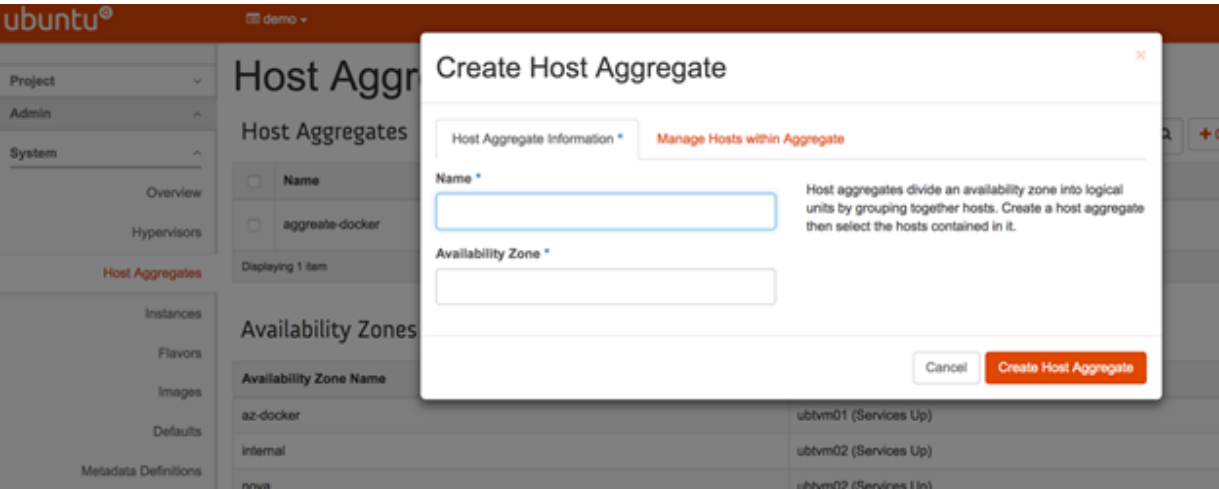
```
root@ubtvm02:~# nova aggregate-details 1
```

```
+---+-----+-----+-----+-----+
| Id | Name          | Availability Zone | Hosts      | Metadata |
|    |              |                  |            |          |
+---+-----+-----+-----+-----+
| 1  | aggregate-docker | az-docker        | 'ubtvm01' |          |
|    |                  |                  |            | 'availability_zone=az-docker' |
```

To create an availability zone for cSRX using the OpenStack Dashboard (Horizon):

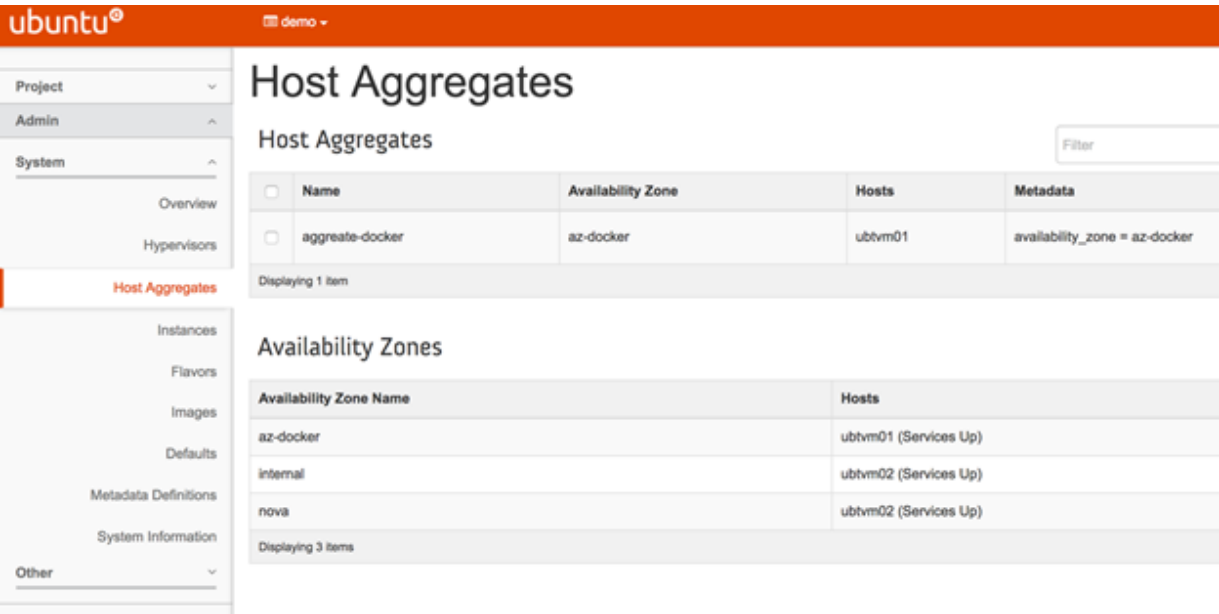
1. Log in to the Dashboard.
2. Open the System tab and click the **Host Aggregates** category.
3. In the Create Host Aggregate page (see [Figure 8 on page 44](#)), enter or select the following values in the Host Aggregate Information tab:
 - Name: The host aggregate name. When you create a host aggregate, you have the option of providing an availability zone name.
 - Availability Zone: The cloud provider defines the default availability zone, such as **us-west**.

Figure 8: Create Host Aggregate Dialog Box



- 4. Click **Create Host Aggregate** to create the host aggregate.
- 5. Check the availability zone on the Host Aggregates screen (see [Figure 9 on page 44](#)).

Figure 9: Host Aggregates Screen



Importing the cSRX Image

To launch a cSRX container based on the images stored in the Openstack Image service (or Glance), you must first add the cSRX image from the Juniper Internal Docker registry. Glance provides discovery, registration, and delivery services for disk and server images. The cSRX image is automatically pulled from the Docker registry to the compute node when a cSRX instance is initially launched.

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

To import the cSRX image file to the Openstack Glance image service:

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@ubtvm02:~# 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:

```
root@ubtvm02:~# curl -u <<username>> -X GET https://hub.juniper.net/v2/security/csr/containers/list
```

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

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

```
root@ubtvm02:~#
```

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/csr/containers/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@ubtvm02:~# docker pull hub.juniper.net/security/csr:<version>
```

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

```
root@ubtvm02:~# docker pull hub.juniper.net/security/csr:18.2R1.9
```

4. Create a tag target image of the cSRX source image and push it to the Docker registry on the control node. You tag the cSRX target image to the cSRX image in the Docker registry. The cSRX registry is a service installed on the control node to help automate the distribution of the cSRX image to other compute nodes in the Contrail Networking cloud environment.

```
root@ubtvm02:~# docker tag csrx:18.1R1.0 csrx-registry:5050/csrx:18.1R1.0
```

```
root@ubtvm02:~# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
SIZE			
csrx	18.1R1.0	918aa1636f22	27 hours ago
799 MB			
csrx-registry:5050/csrx	18.1R1.0	918aa1636f22	27 hours ago
799 MB			
... ..			

```
root@ubtvm02:~# docker push csrx-registry:5050/csrx:18.1R1.0
```

```
.....
096c5913d0b0: Pushed
```

5. Import the cSRX image to the Openstack Glance image service using the appropriate values for your Contrail environment and disk image.

```
root@ubtvm02:~# source /etc/contrail/openstackrc
```

```
root@ubtvm02:~# docker save csrx-registry:5050/csrx:18.1R1.0 | glance image-create
--container-format=docker --disk-format=raw --name csrx-registry:5050/csrx:18.1R1.0
```

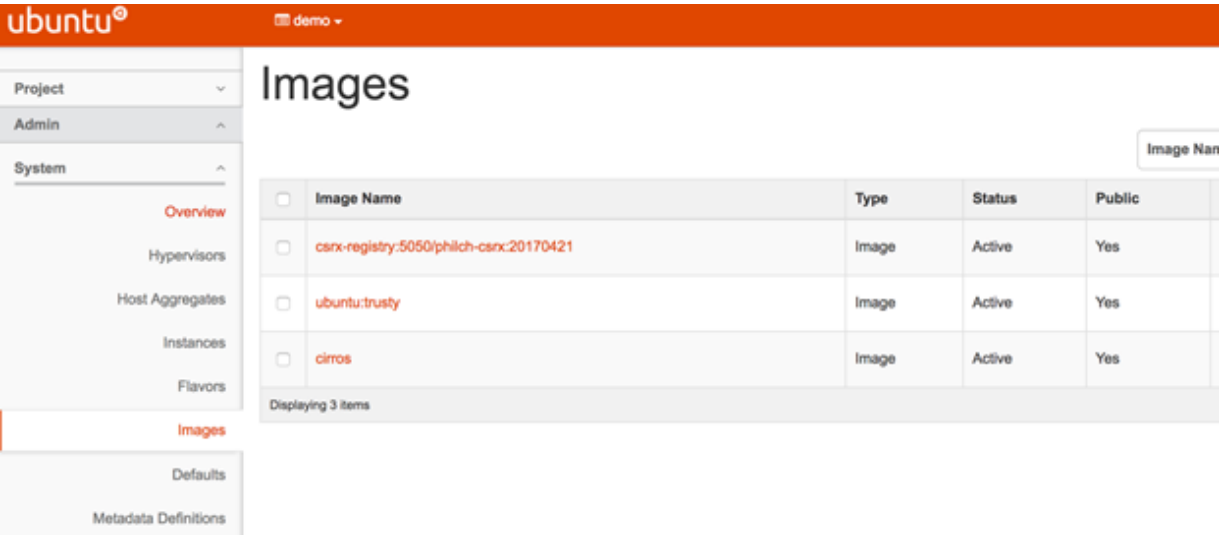
6. Check the Openstack Glance image using the **nova image-list** command.

```
root@ubtvm02:~# nova image-list
```

```
..
...
| | fd4532bc-edfe-4fe6-835f-20304f53c115 | csrx-registry:5050/csrx:18.1R1.0 |
ACTIVE |
```

You can also check the Glance image from the Images page (see [Figure 10 on page 47](#)) of the OpenStack Dashboard (Horizon).

Figure 10: Images Screen



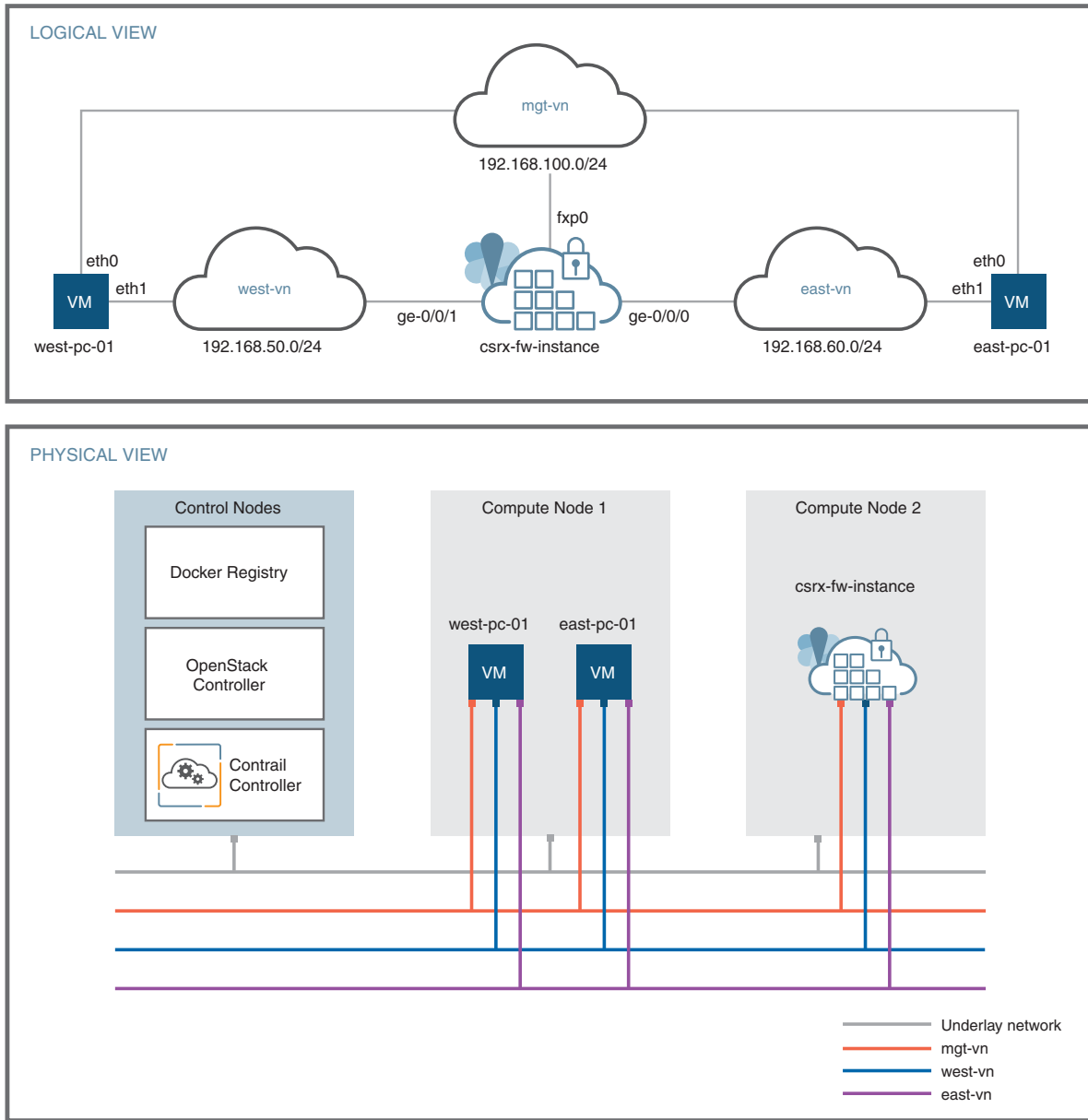
Creating Virtual Networks in Contrail

The cSRX container requires three virtual networks: one virtual network for out-of-band management sessions, and the other two virtual networks to receive and transmit in-band data traffic. You create a left, right, and management virtual network on Contrail, and then connect the cSRX to the virtual networks. You create networks and network policies at the user dashboard of Contrail, then associate policies with each network. The trusted and untrusted interfaces required by a cSRX connector are connected to eth1 and eth2.

NOTE: If there is already a virtual network created in your Contrail Networking cloud environment, the cSRX container can be launched and attached to the existing virtual networks. Virtual networks can be shared across different tenants.

This topic summarizes how to create the three virtual networks required by the cSRX container: mgt-vn (eth0), west-vn (eth1), and east-vn (eth2). mgt-vn is used by the cSRX for out-of-band management to the accept management sessions and traffic, and west-vn and east-vn are both used by the cSRX as the two revenue ports to process in-band data traffic (the ge-0/0/0 and ge-0/0/1 interfaces).

[Figure 11 on page 50](#) illustrates three virtual networks used by a cSRX in an East West firewall.



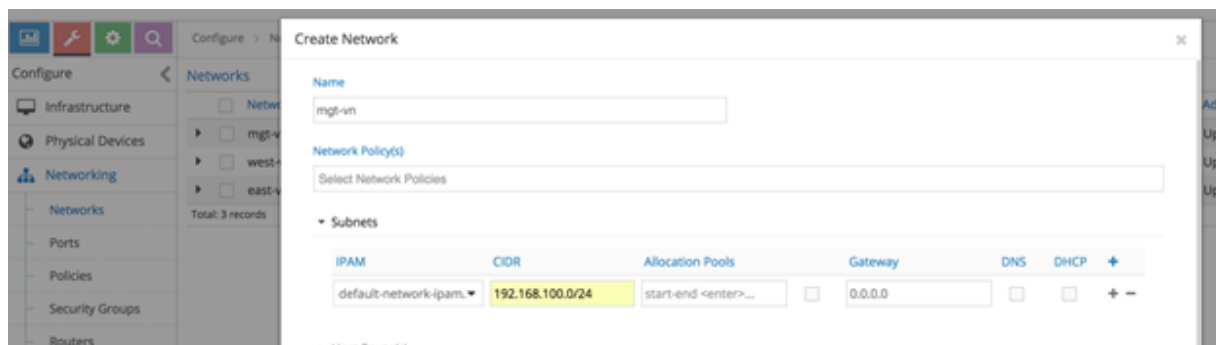
For the procedure on creating a virtual network in Contrail, see [Creating a Virtual Network with Juniper Networks Contrail](#).

NOTE: This procedure assumes that “left” and “right” VMs exist in your Contrail virtual network.

To create the virtual networks required by cSRX:

1. Before creating a virtual network, ensure that you have IP Address Management (IPAM) set up for your project. Select **Configure > Networking > IP Address Management**, and then click the **Create** button.
2. From the Contrail GUI, select **Configure > Networking > Networks** to access the Configure Network page. The list of existing networks appears.
3. Click the **Create Network (+)** icon. The Create Network page appears (see [Figure 12 on page 51](#)).

Figure 12: Create Network Page



4. Enter a name for the virtual network (mgt-vn (eth0), west-vn (eth1), or east-vn (eth1)).
Do not select a network policy yet. You create the network policy after you create the service instance and then you update this virtual network to add the policy.
5. Expand **Subnet** and click **+** to add IPAM to this virtual network.
6. Select the appropriate IPAM from the list.
7. Set the CIDR and Gateway fields. Depending on the virtual network you are creating, ensure that the eth0 network address is assigned to the mgt-vn, the ge-0/0/0 network address is assigned to the “left” network, and the ge-0/0/1 network address is assigned to the “right” network.
8. Expand **Advanced Options** and select appropriate options for your network.

NOTE: When creating a virtual network for west-vn (eth1) and east-vn (eth2), ensure that you enable **Advanced Options**. This is a requirement for Layer 2 forwarding.

9. Click **Save**. The new virtual network appears in the list of configured networks.

10. Repeat this procedure for the remaining virtual networks required by the cSRX container.
11. Verify the completed virtual networks for the cSRX container in the Networks page (see [Figure 13 on page 52](#)).

Figure 13: Completed Virtual Networks for cSRX

Network	Subnets	Attached Policies	Shared	Admin State
mgmt-vn	192.168.100.0/24	-	Enabled	Up
west-vn	192.168.50.0/24	-	Enabled	Up
east-vn	192.168.60.0/24	-	Enabled	Up

Total: 3 records 50 Records

Launching the cSRX Container

Launch the cSRX container in Openstack using the **nova boot** CLI command. 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
- 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

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 Configuration Data File and Environment Variables” on page 62](#). You can launch the cSRX using the default environment variable settings.

To launch the cSRX container:

1. Use the **nova boot** command to launch the cSRX container. If you intend to log into the cSRX container using SSH, you must specify an initial root password when launching the cSRX.

Metadata is the key value pair that can be specified when you launch a compute instance in Openstack. For the cSRX container, the metadata is used to pass one or more environment variables when you launch the cSRX. Any environment variable supported by the cSRX container can be passed to the cSRX by including the **-meta** option in the **nova boot** command.

[“cSRX Configuration Data File and Environment Variables” on page 62](#) summarizes the list of available cSRX environment variables along with a link to the topic that outlines its usage.

For example:

```
root@csrx-ubuntu3:~/csrx# nova boot --image csrx-registry:5050/csrx:18.1R1.0 --flavor m1.small
--availability-zone az-docker --nic net-id=039e73e4-6033-4851-8379-21e1cedf1a30 --nic
net-id=326eb329-1e66-46b7-8438-a8f41c88bec9 --nic
net-id=3e744a74-2579-455f-aea9-92e0655abec6 --meta CSRX_SIZE=middle --meta
CSRX_ROOT_PASSWORD=<password> --meta CSRX_PACKET_DRIVER=interrupt --meta csrx-fw
```

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

```
root@csrx-ubuntu3:~/csrx# nova list
```

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
```

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

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

UID	PID	PPID	C
STIME	TTY	TIME	CMD
root	1809	1788	0
07:51	pts/0	00:00:00	/bin/bash -e
/etc/rc.local init			
root	2271	1809	0
07:51	?	00:00:00	/usr/sbin/rsyslogd
-M/usr/lib/rsyslog			
root	2290	1809	0
07:51	?	00:00:00	/usr/sbin/sshd
root	2308	1809	0
07:51	?	00:00:00	/usr/bin/monit
root	2314	1809	0
07:51	?	00:00:00	/usr/sbin/nstraced

root	2325	1809	0
07:51	?	00:00:00	/usr/sbin/nsd
root	2335	1809	14
07:51	?	00:00:02	/usr/sbin/appidd
-N			
root	2349	1809	0
07:51	?	00:00:00	/usr/sbin/idpd -N
root	2358	1809	0
07:51	?	00:00:00	/usr/sbin/wmic -N
root	2366	1809	0
07:51	?	00:00:00	/usr/sbin/useridd
-N			
root	2380	1809	0
07:51	?	00:00:00	/usr/sbin/mgd
root	2439	1809	96
07:51	?	00:00:17	/usr/sbin/srxpfe
-a -d			
root	2467	1809	0
07:51	?	00:00:00	/usr/sbin/utmd -N
root	2488	1809	0
07:51	?	00:00:00	/usr/sbin/kmd
root	2623	1809	0
07:51	pts/0	00:00:00	/bin/bash

Creating a Service Template for the cSRX

Creation of a service template (version 2) in Contrail is a critical step in adding the cSRX container to a service chain. The Contrail service template is used in a service instance to launch the cSRX as part of a service chain.

To create a cSRX service template:

1. From the Contrail GUI, select **Configure > Services > Service Templates**. The list of existing service templates appears.
2. Click the **Create (+)** button on Service Templates. The Create Service Template page appears (see [Figure 14 on page 55](#)).

Figure 14: Create Service Template Page

Create

Service Template Permissions

Name
csrx-template-v2

Version
v2

Virtualization Type
Virtual Machine

Service Mode
In-Network

Service Type
Firewall

Interface(s) +

management + -

left + -

right

Cancel Save

3. Add a name for the service template in the Name box.
4. Select **v2** in the Version field
5. Select **Virtual Machine** as Virtualization Type from the list.
6. Select **In-Network** as Service Mode and **Firewall** as Service Type from the lists.
7. Under Interface(s), click + to add three interfaces. Select **Management** for the first interface type, **Left** for the second interface type, and **Right** for the third interface type. You associate the left and right interfaces with the left and right virtual networks when you create the service instance. Any additional interfaces must be of type Other.
8. Click **Save** to save the new service template. The cSRX service template appears on the Service Templates page.
9. Confirm the cSRX service template settings from the Service Templates page.

Creating and Launching the Service Instance

You are now ready to create and launch the service instance from the Docker registry.

To create and launch the service instance:

1. From the Contrail GUI, configure a service instance for an in-network service template. Navigate to **Configure > Services > Service Instances** and then click **Create** on the Service Templates window. The Create Service Instance page appears (see [Figure 15 on page 56](#)).

Figure 15: Create Service Instance Page

Create

Service Instance Permissions

Name Service Template

csrx-fw-instance csrx-template-v2 - [in-network (manageme...

Interface Type Virtual Network

management mgt_lab_subnet

left net_left

right net_right

Port Tuples

Tuple

port-tuple0

Cancel Save

2. Enter a name for the cSRX service instance.

NOTE: Do not use white space in the service instance name.

3. Select the service template you created for cSRX from the Service Template list.

4. Under Virtual Network. select the virtual network for the management, left, and right interfaces.
5. Under Port Tuples, select the port tuples from Tuples list. Ports for the cSRX container are created as part of the cSRX container launch in Openstack using the **nova boot** CLI command. With a port-tuple object, you can create ports and pass the port information when creating the service instance. The ports are linked to a port-tuple object that is a child of a service instance.
6. Click **Save** to save this service instance. Contrail launches the cSRX container for this service instance.
7. Confirm that the service instance status is **Active** .
8. Check the cSRX compute node. Confirm that the cSRX image was automatically pulled from the Docker registry and that the Docker instance is running.

NOTE: It might take longer for the first cSRX instance to launch because it has to pull the image from the Docker registry server.

root@ubtvm02:~# nova image list

REPOSITORY		TAG	IMAGE ID	CREATED
	VIRTUAL SIZE			
csrx- registry:5050/- csrx		18.1R1.0	4b7fcac7f30d	39 hours
ago	551.1 MB			
ubuntu		trusty	bec964527be1	7 weeks
ago	188 MB			
csrx- registry:5000/ubuntu- 14		1.0	a4c8a0f2f25f	16 months
ago	589 MB			

root@ubtvm02:~# nova list

CONTAINER ID	IMAGE	COMMAND
CREATED		
STATUS	PORTS	NAMES
b4002accalac	csrx- registry:5050/- csrx:18.1R1.0	"/etc/rc.local init"
11 minutes ago Up		
11 minutes	nova- 85d5f949- 97e7- 4f46- b18f--	
0ddf227fe4fe		

Creating a Network Policy (Optional)

(Optional) To create a network policy to allow traffic between virtual networks and the service instance:

1. From the Contrail GUI, select **Configure > Networking > Policies**. The table of policies appears.
2. Click **+** to create a new policy. The Create Policy page appears, as shown in [Figure 16 on page 58](#).

Figure 16: Creating a Network Policy in Contrail

Create Policy

Name

cSRXPolicy1

Policy Rules

Action	Protocol	Source	Ports	Direction	Destination	Ports	Services	Mirror	
PASS	ANY	↔ Left	ANY	<>	↔ Right	ANY	<input checked="" type="checkbox"/>	<input type="checkbox"/>	- +

Services

ixSvcTest x

Cancel Save

3. Name the policy.
4. Click **+** to create a new rule for this policy.
5. Select the left virtual network you created from the Source list and select the right virtual network from the Destination list.
6. Select the appropriate protocol from the Protocol list and select the source and destination ports for this policy.
7. Select **Services** and select the cSRX instance you want to apply this policy to.
8. Optionally, add more policy rules to this policy.
9. Click **Save** to create this policy.

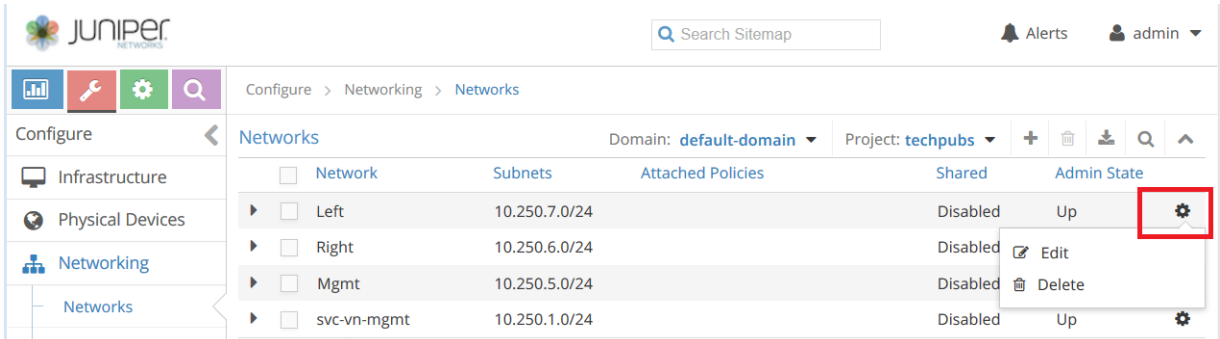
See [Creating a Network Policy—Juniper Networks Contrail](#) for more details.

Adding a Network Policy to a Virtual Network (Optional)

(Optional) To add a network policy to a virtual network:

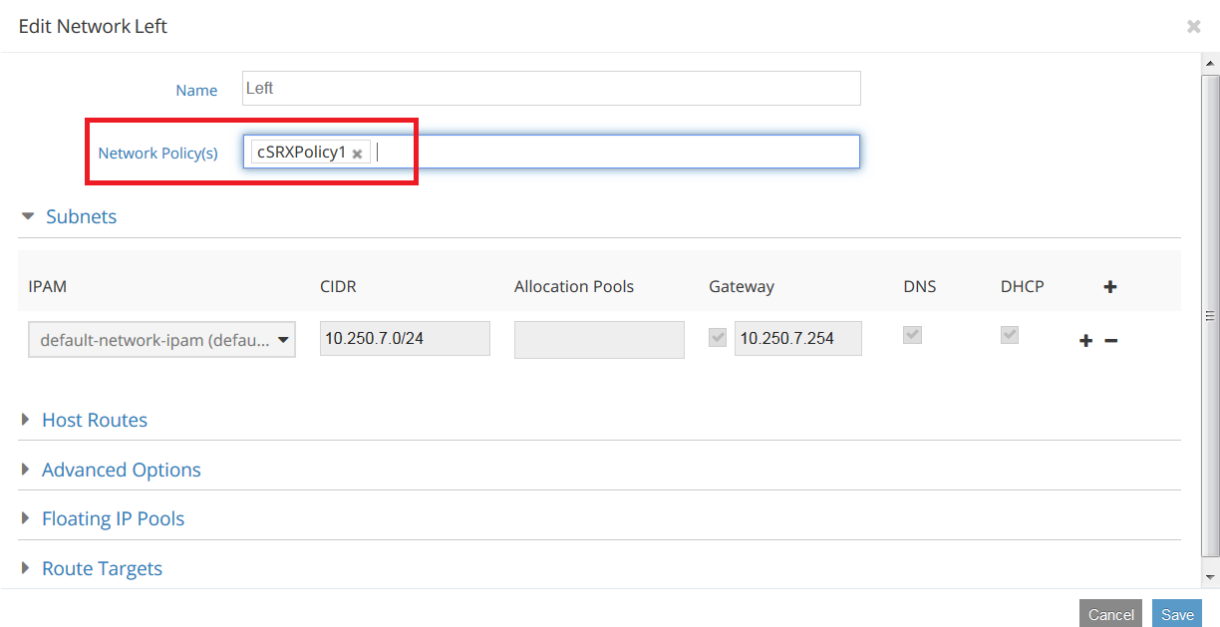
1. From the Contrail GUI, select **Configure > Networking**, and select the settings icon to the right of the virtual network you want to add a network policy to, as shown in [Figure 17 on page 59](#).

Figure 17: Networks Window Page



2. Click **Edit**. The Edit Networks page appears, as shown in [Figure 18 on page 59](#).

Figure 18: Adding a Network Policy to a Virtual Network



3. Select the appropriate policy from the Network Policy(s) list.

4. Click **Save** to save this change.
5. Repeat this procedure for the other virtual network in this service chain.

See [Associating a Network to a Policy—Juniper Networks Contrail](#) for more details.

3

CHAPTER

Managing cSRX Containers in Contrail

cSRX Configuration Data File and Environment Variables | **62**

Specifying an Initial Root Password for Logging into a cSRX Container | **65**

Configuring cSRX for Routing Mode | **66**

Changing the Size of a cSRX Container | **69**

Specifying the Packet I/O Driver for a cSRX Container | **70**

Configuring CPU Affinity for a cSRX Container | **72**

Managing cSRX Containers | **73**

cSRX Configuration Data File and Environment Variables

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. A series of cSRX environment variables enable you to modify the characteristics of the cSRX instance when it is launched. You can set any number of parameters to take effect when the cSRX image launches. You can pass configuration settings in the form of a configuration data file or environment variables to the cSRX when it launches at boot time.

NOTE: The specification of an environment variable is not mandatory; most environment variables have a default value as shown in [Table 8 on page 62](#). If desired, you can launch the cSRX using the default environment variable settings.

You pass configuration settings by using the following methods:

- Openstack user data file– Passes a validated Junos OS configuration file to automate the initialization of a cSRX instance when it is launched.
- Openstack metadata–Passes a series of cSRX environment variables to modify the characteristics of the cSRX instance when it is launched.

[Table 8 on page 62](#) 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

Environment Variable	Description	Values	Default	Topic
CSRX_SIZE	cSRX size.	small middle large	large	“Changing the Size of a cSRX Container” on page 69
CSRX_PACKET_DRIVER	Packet I/O driver.	poll interrupt	poll	“Specifying the Packet I/O Driver for a cSRX Container” on page 70
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” on page 65

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

Environment Variable	Description	Values	Default	Topic
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 72
CSRX_DATA_CPU	CPU mask, indicating which CPU is running the cSRX data plane daemon (srxfpe).	hex value	No CPU affinity	“Configuring CPU Affinity for a cSRX Container” on page 72
CSRX_ARP_TIMEOUT	ARP entry timeout value for the control plane ARP learning or response.	decimal value	Same as the Linux host	“Configuring cSRX for Routing Mode” on page 66
CSRX_NDP_TIMEOUT	NDP entry timeout value for the control plane NDP learning or response.	decimal value	Same as the Linux host	“Configuring cSRX for Routing Mode” on page 66

Openstack User Data File

User data enables you to pass a Junos OS configuration contained in a local file to a cSRX instance at launch time. A typical use case would be to pass something similar to a shell script or a configuration file as user data. The Openstack user data file for cSRX is organized as an XML formatted file, enclosed with the `<csrx_conf>` and `</csrx_conf>` tags. .

You can also specify the following subelements in an XML file:

- `conf`—Configuration to be pushed for the cSRX service after launching the cSRX container.
- `boot_script`—Script to be executed in the shell after launching the cSRX container.

The following example shows an Openstack user data file configured to pass a Junos OS configuration :

```
<csrx_conf>
  <conf>

edit
set system root-authentication encrypted-password
"$1$91fRgcxz$Vc8dgodJiPR61Rd59/Lza/"
```

```

set system root-authentication ssh-rsa "ssh-rsa <<<PUBLIC.SSH.KEY.HERE user
@juniper.net>>>"
set system name-server 8.8.8.8
set system host-name csrx33

set interfaces ge-0/0/0 unit 0 family inet address 192.168.60.33/24
set interfaces ge-0/0/1 unit 0 family inet address 192.168.50.33/24
set routing-options static route 0.0.0.0/0 next-hop 192.168.60.1
set routing-options static route 172.26.0.0/16 next-hop 10.0.109.1

set security policies from-zone trust to-zone untrust policy t2u match
source-address any
set security policies from-zone trust to-zone untrust policy t2u match
destination-address any
set security policies from-zone trust to-zone untrust policy t2u match application
any
set security policies from-zone trust to-zone untrust policy t2u then permit
set security policies from-zone trust to-zone untrust policy t2u then log
session-init
set security policies from-zone trust to-zone untrust policy t2u then log
session-close
set security policies default-policy deny-all

commit
    </conf>
    <boot_script>

#!/bin/sh
echo "my boot script"
    </boot_script>>
</csrx_conf>

```

After you create the user data file, you can then use it to pass a Junos OS configuration to the cSRX container when you launch the cSRX with the **nova boot** command. You send the user data file by including the **--user-data /path/to/filename** option.

For example:

```

root@csrx-ubuntu3:~/csrx# nova boot --image csrx-registry:5050/-csrx:20171214 --flavor m1.small
--availability-zone az-docker --nic net-id=039e73e4-6033-4851-8379-21e1cedf1a30 --nic
net-id=326eb329-1e66-46b7-8438-a8f41c88bec9 --nic net-id=3e744a74-2579-455f-aea9-92e0655abec6
--meta CSRX_USER_DATA=yes --user-data ./user-data.xml csrx-fw

```


Openstack Metadata

Metadata is the key value pair that can be specified when you launch a compute instance in Openstack. For the cSRX, the metadata is used to pass one or more environment variables to the cSRX container when you launch the cSRX with the **nova boot** command. Any environment variable supported by the cSRX container (see [Table 8 on page 62](#)) can be passed to the cSRX by including the **-meta** option in the **nova boot** command.

For example:

```
root@csrx-ubuntu3:~/csrx# nova boot --image csrx-registry:5050/csrx:20171214 --flavor m1.small
--availability-zone az-docker --nic net-id=039e73e4-6033-4851-8379-21e1cedf1a30 --nic
net-id=326eb329-1e66-46b7-8438-a8f41c88bec9 --nic net-id=3e744a74-2579-455f-aea9-92e0655abec6
--meta CSRX_SIZE=middle --meta CSRX_ROOT_PASSWORD=<password> --meta
CSRX_PACKET_DRIVER=interrupt csrx-fw
```

Specifying an Initial Root Password for Logging into a cSRX Container

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, include the **CSRX_ROOT_PASSWORD** environment variable in the **-meta** option as part of the **nova boot** command syntax. For example:

```
root@csrx-ubuntu3:~/csrx# nova boot --image csrx-registry:5050/csrx:20171214 --flavor m1.small
--availability-zone az-docker --nic net-id=039e73e4-6033-4851-8379-21e1cedf1a30 --nic
```

```
net-id=326eb329-1e66-46b7-8438-a8f41c88bec9 --nic net-id=3e744a74-2579-455f-aea9-92e0655abec6
--meta CSRX_ROOT_PASSWORD=<password> csrx-fw
```

Configuring cSRX for Routing Mode

With the cSRX container operating 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 of egress traffic.

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

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

1. Launch the cSRX container.

```
root@csrx-ubuntu3:~/csrx# nova boot --image csrx-registry:5050/csrx:20171214 --flavor m1.small
--availability-zone az-docker --nic net-id=039e73e4-6033-4851-8379-21e1cedf1a30 --nic
net-id=326eb329-1e66-46b7-8438-a8f41c88bec9 --nic
net-id=3e744a74-2579-455f-aea9-92e0655abec6 --meta CSRX_SIZE=middle --meta
CSRX_ROOT_PASSWORD=<password> csrx-fw
```

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

```

- 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

```

- 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;
```

```
}
```

- 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 and added to security zones by default.

```
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# nova boot --image csrx-registry:5050/csrx:20171214 --flavor m1.small
--availability-zone az-docker --nic net-id=039e73e4-6033-4851-8379-21e1cedf1a30 --nic
net-id=326eb329-1e66-46b7-8438-a8f41c88bec9 --nic
net-id=3e744a74-2579-455f-aea9-92e0655abec6 --meta CSRX_ARP_TIMEOUT=<seconds> --meta
CSRX_ROOT_PASSWORD=<password> csrx-fw
```

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
```

Changing the Size of a cSRX Container

Based on your specific cSRX container 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 69](#) 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 **-meta** option as part of the **nova boot** command syntax. For example, to launch a cSRX instance using the middle size configuration:

```
root@csrx-ubuntu3:~/csrx# nova boot --image csrx-registry:5050/csrx:20171214 --flavor m1.small
--availability-zone az-docker --nic net-id=039e73e4-6033-4851-8379-21e1cedf1a30 --nic
```


```
net-id=326eb329-1e66-46b7-8438-a8f41c88bec9 --nic net-id=3e744a74-2579-455f-aea9-92e0655abec6
--meta CSRX_SIZE=middle csrx-fw
```

Specifying the Packet I/O Driver for a cSRX Container

IN THIS SECTION

- [Specifying the Poll Mode Driver | 71](#)
- [Specifying the Interrupt Mode Driver | 72](#)

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 70](#) 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.
Scalability	Reduced scalability; support for a single cSRX per CPU.	Improved scalability; support for multiple cSRX containers per CPU.
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 **CSRX_PACKET_DRIVER=poll** environment variable in the **-meta** option as part of the **nova boot** command syntax.

```
root@csrx-ubuntu3:~/csrx# nova boot --image csrx-registry:5050/csrx:20171214 --flavor m1.small
--availability-zone az-docker --nic net-id=039e73e4-6033-4851-8379-21e1cedf1a30 --nic
net-id=326eb329-1e66-46b7-8438-a8f41c88bec9 --nic net-id=3e744a74-2579-455f-aea9-92e0655abec6
--meta CSRX_PACKET_DRIVER=poll --meta CSRX_ROOT_PASSWORD=<password> csrx-fw
```

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 srpxfe process in waiting until packets arrive on the VETH interfaces. If no packets load on the revenue ports of a cSRX instance, the srpxfe process remains in a sleep state 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 srpxfe process is busy and allocates CPU resources to that srpxfe process.

NOTE: When you launch 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 srpxfe process among those CPUs according to their CPU status. See [“Configuring CPU Affinity for a cSRX Container” on page 72](#) 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 **`CSRX_PACKET_DRIVER=interrupt`** environment variable in the **`-meta`** option as part of the **`nova boot`** command syntax.

```
root@csrx-ubuntu3:~/csrx# nova boot --image csrx-registry:5050/csrx:20171214 --flavor m1.small
--availability-zone az-docker --nic net-id=039e73e4-6033-4851-8379-21e1cedf1a30 --nic
net-id=326eb329-1e66-46b7-8438-a8f41c88bec9 --nic net-id=3e744a74-2579-455f-aea9-92e0655abec6
--meta CSRX_PACKET_DRIVER=interrupt --meta CSRX_ROOT_PASSWORD=<password> --meta
CSRX_CTRL_CPU=0x1 --meta CSRX_DATA_CPU=0x2 csrx-fw
```

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, srpxfe 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 **CSRX_CTRL_CPU** and **CSRX_DATA_CPU** environment variables in the **--meta** option as part of the **nova boot** command syntax.

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# nova boot --image csrx-registry:5050/csrx:20171214 --flavor m1.small
--availability-zone az-docker --nic net-id=039e73e4-6033-4851-8379-21e1cedf1a30 --nic
net-id=326eb329-1e66-46b7-8438-a8f41c88bec9 --nic net-id=3e744a74-2579-455f-aea9-92e0655abec6
--meta CSRX_CTRL_CPU=0x1 --meta CSRX_DATA_CPU=0x2 --meta
CSRX_ROOT_PASSWORD=<password> csrx-fw
```

Managing cSRX Containers

IN THIS SECTION

- [Powering On the cSRX Container from OpenStack CLI | 74](#)
- [Powering On the cSRX Container from OpenStack Dashboard | 74](#)
- [Pausing the cSRX Container from OpenStack CLI | 74](#)
- [Pausing the cSRX Container from OpenStack Dashboard | 75](#)
- [Restarting the cSRX Container from OpenStack CLI | 75](#)
- [Restarting the cSRX Container from OpenStack Dashboard | 75](#)
- [Deleting the cSRX Container from OpenStack CLI | 75](#)
- [Deleting the cSRX Container from Contrail | 76](#)
- [Monitoring Basic cSRX Statistics with the Contrail Monitor | 76](#)

Each cSRX instance is an independent container in Contrail that you can directly manage. You can also monitor basic statistics with the Contrail Monitor.

This section includes the following topics:

Powering On the cSRX Container from OpenStack CLI

To power on the cSRX container from the OpenStack CLI:

1. From the OpenStack CLI, enter **nova list**. The list of existing instances appears, including the cSRX container.
2. Enter **nova start<csrx_name>**.

Powering On the cSRX Container from OpenStack Dashboard

To power on the cSRX container from the OpenStack Dashboard:

1. From the OpenStack Dashboard, select **Compute > Instances**. The list of existing instances appears.
2. Check the cSRX container you want to power on.
3. From the Actions column, select **Start Instance** from the list.

Pausing the cSRX Container from OpenStack CLI

To pause and resume a cSRX container from the OpenStack CLI:

1. From the OpenStack CLI, enter **nova list**. The list of existing instances appears, including the cSRX container.
2. To pause the cSRX container, select **nova pause<csrx_name>**.
3. To resume the cSRX container, select **nova unpause<csrx_name>**.

Pausing the cSRX Container from OpenStack Dashboard

To pause the cSRX container from the OpenStack Dashboard:

1. From the OpenStack Dashboard, select **Compute > Instances**. The list of existing instances appears.
2. Check the cSRX container that you want to pause.
3. From the Actions column, select **Pause Instance** from the list.

Restarting the cSRX Container from OpenStack CLI

To restart the cSRX container from the OpenStack CLI:

1. From the OpenStack CLI, enter **nova list**. The list of existing instances appears, including the cSRX container.
2. Enter **nova reboot<csrx_name>** to perform a soft reboot of the cSRX container.

Restarting the cSRX Container from OpenStack Dashboard

To restart the cSRX container from the OpenStack Dashboard:

1. From the OpenStack Dashboard, select **Compute > Instances**. The list of existing instances appears.
2. Check the container that you want to reboot.
3. Select **Soft Reboot Instance** from the More list to restart the container.

Deleting the cSRX Container from OpenStack CLI

To delete the cSRX container from the OpenStack CLI:

1. From the OpenStack CLI, enter **nova list**. The list of existing instances appears, including the cSRX container.

2. Enter `nova delete<csrx_name>`.

Deleting the cSRX Container from Contrail

To delete the container from Contrail:

1. From the Contrail GUI for your project, select **Configure > Services > Service Instances**. The list of existing service instances appears.
2. Select the container that you want to delete.
3. Click the trash icon on the upper right menu to delete the selected containers.

Monitoring Basic cSRX Statistics with the Contrail Monitor

To monitor basic statistics on the cSRX container with the Contrail Monitor:

1. From the Contrail GUI, select **Monitor > Networking>Instances**. The list of existing VMs appears.
2. Expand the row for the cSRX that you want to monitor. The CPU and memory statistics appear.
3. Select **Monitor > Networking > Networks**. The list of existing virtual networks appears.
4. Expand the row for the virtual network that you want to monitor and select **Traffic Statistics**. The traffic and throughput statistics appear.

RELATED DOCUMENTATION

[Monitoring the System](#)

[OpenStack End User Guide](#)

4

CHAPTER

Configuring cSRX

Configuring cSRX Using the Junos OS CLI | 78

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@#
```

- 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
```

- Configure the hostname.

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

- Configure the two traffic interfaces.

NOTE: Docker automatically connects the fxp0 management interface (eth0) to the Linux bridge and automatically assigns an IP address. It 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
```

- 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
```

- Verify the configuration.

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

- 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](#)