Engineering
Simplicity

cSRX Deployment Guide Kubernetes

Published
2020-09-30

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

YEAR 2000 NOTICE

END USER LICENSE AGREEMENT

Table of Contents

About the Documentation

IN THIS SECTION

Use this guide to install and configure the cSRX Container Firewall on Kubernetes environment. 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 security feature configuration.

Documentation and Release Notes

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

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

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

Documentation Conventions

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

Table 1: Notice Icons

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

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

Table 2: Text and Syntax Conventions

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

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

Convention	Description	Examples
Italic text like this	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** *domain-name*
Text like this	Represents names of configuration statements, commands, files, and directories; configuration hierarchy levels; or labels on routing platform components.	• 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 *metric*>;**
| (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** **(*string1* | *string2* | *string3*)**
# (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 [*community-ids*]**

Indention and braces ({ })	Identifies a level in the configuration hierarchy.	[edit]
		routing-options {
		  static {
		    route default {
		      nexthop *address*;
		      retain;
		    }
		  }
		}

; (semicolon)	Identifies a leaf statement at a configuration hierarchy level.	
GUI Conventions		

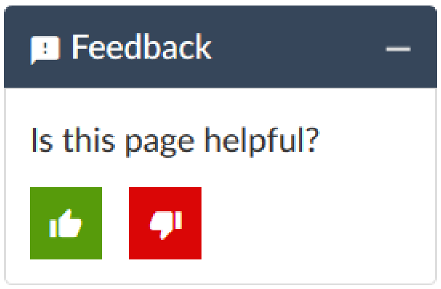
Table 2: Text and Syntax Conventions *(continued)*

Convention	Description	Examples
Bold text like this	Represents graphical user interface (GUI) items you click or select.	• 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

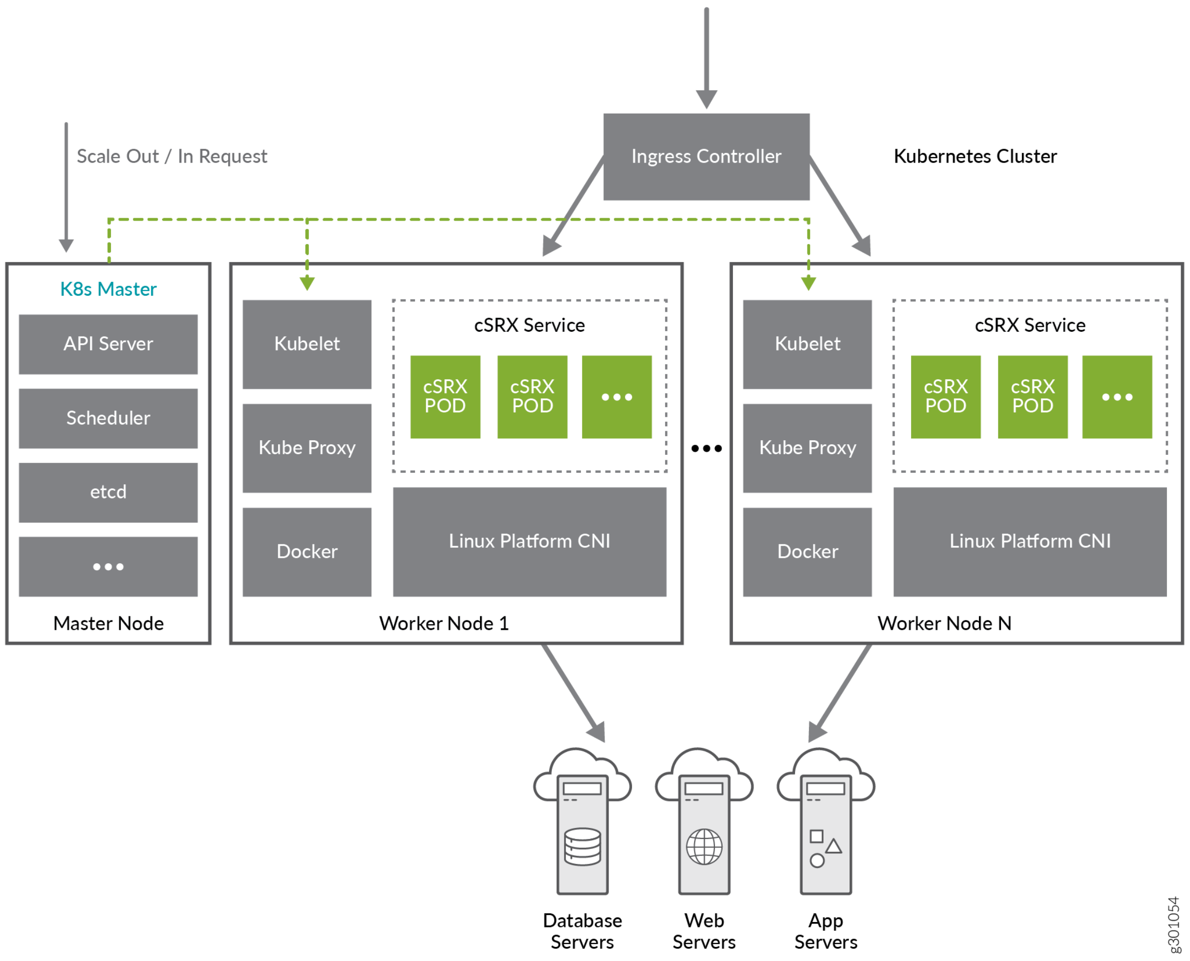
Understanding cSRX with Kubernetes

Containerized SRX (cSRX) is a virtual security solution based on Docker container to deliver agile, elastic and cost-saving security services for comprehensive L7 security protection.

Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications. With K8s support, cSRX can scale out in a cluster running as elastic firewall service with smaller footprint when compared to virtual machines. cSRX running in K8s cluster provides advantages such as:

- Runs services with smaller footprint
- Enables faster Scale out and scale in of cSRX
- Automated management and controlled workflow

Figure 1: cSRX Service in Kubernetes Linux



In K8s deployment, you can use Multus with both Flannel and Weave CNI.

To use Multus with Weave CNI, you can set environment variable *CSRX_MGMT_PORT_REORDER* to **yes** to bind first interface as MGMT port and set to **no** to bind last interface as MGMT port.

To use Multus with Flannel CNI, you can set environment variable *CSRX_MGMT_PORT_REORDER* to **no** to bind first interface as MGMT port and set to **yes** to bind last interface as MGMT port.

In case of multiple networks supported under K8s deployment, the environment variable *CSRX_MGMT_PORT_REORDER* supports reordering of the interface mapping in cSRX.

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 the form of a container. By using a Docker container the cSRX can substantially reduce overhead as 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. Also, because of the containers' lightweight quality, a server can host many more container instances than virtual machines (VMs), yielding tremendous improvements in utilization. With its small footprint and Docker as a container management system, the cSRX Container Firewall enables deployment of agile, high-density security service.

See "Junos OS Features Supported on cSRX" on page 14 for a summary of the features supported on cSRX.

Licensing

The cSRX Container Firewall software features require a license to activate the feature. To understand more about cSRX Container Firewall licenses, see Supported Features on cSRX, Juniper Agile Licensing Guide, and Managing cSRX Licenses.

Kubernetes Overview

K8s is an open-source system for automating deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery.

K8s defines a set of building objects that collectively provide mechanisms that orchestrate containerized applications across a distributed cluster of nodes, based on system resources (CPU, memory, or other custom metrics). K8s masks the complexity of managing a group of containers by providing REST APIs for the required functionalities.

A node refers to a logical unit in a cluster, like a server, which can either be physical or virtual. In context of Kubernetes clusters, a node usually refers specifically to a worker node. Kubernetes nodes in a cluster are the machines that run the end user applications.

There are two type of nodes in a Kubernetes cluster, and each one runs a well-defined set of processes:

- head node: also called master, or master node, it is the head and brain that does all the thinking and makes all the decisions; all of the intelligence is located here.
- worker node: also called node, or minion, it's the hands and feet that conducts the workforce.

The nodes are controlled by the master in most cases.

The interfaces between the cluster and you is the command-line tool kubectl. It is installed as a client application, either in the same master node or in a separate machine.

Kubernetes's objects are:

- Pod
- Service
- Volume
- Namespace
- Replication
- Controller
- ReplicaSet
- Deployment
- StatefulSet
- DaemonSet
- Job

RELATED DOCUMENTATION

What is a Container?

Kubernetes Concepts

Junos OS Features Supported on cSRX

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

Supported SRX Series Features on cSRX

Table 3 on page 14 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 3: Security Features Supported on cSRX

Security Features	Considerations
Application Tracking (AppTrack)	Understanding AppTrack
Application Firewall (AppFW)	Application Firewall Overview
Application Identification (AppID)	Understanding Application Identification Techniques
Basic Firewall Policy	Understanding Security Basics
Brute force attack mitigation	
DoS/DDoS protection	DoS Attack Overview DoS Attack Overview
Intrusion Prevention System (IPS)	For SRX Series IPS configuration details, see: Understanding Intrusion Detection and Prevention for SRX Series
IPv4 and IPv6	Understanding IPv4 Addressing Understanding IPv6 Address Space
Jumbo Frames	Understanding Jumbo Frames Support for Ethernet Interfaces
SYN cookie protection	Understanding SYN Cookie Protection

Table 3: Security Features Supported on cSRX *(continued)*

Security Features	Considerations
Malformed packet protection	
Unified Threat Management (UTM)	Includes support for all UTM functionality on the cSRX platform, such as: • Antispam • Sophos Antivirus • Web filtering • Content filtering For SRX Series UTM configuration details, see: Unified Threat Management Overview For SRX Series UTM antispam configuration details, see: Antispam Filtering Overview
User Firewall	Includes support for all user firewall functionality on the cSRX platform, such as: • Policy enforcement with matching source identity criteria • Logging with source identity information • Integrated user firewall with active directory • Local authentication For SRX Series user firewall configuration details, see: Overview of Integrated User Firewall
Zones and Zone based IP spoofing	Understanding IP Spoofing

2

CHAPTER

cSRX Installation on Kubernetes

Requirements for Deploying cSRX on Kubernetes

IN THIS SECTION

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

Platform and Server Requirements

Table 4 on page 17 lists the requirements for deploying a cSRX container in a Kubernetes (Master and Worker) node.

Table 4: Master and Worker Node Specifications

Component	Specification
Docker Engine	Docker Engine 1.9 or later installed on the same compute node as the cSRX
vCPUs	2
Memory	4 GB
Disk space	50 GB hard drive
Interfaces	16 The environment variable **CSRX_PORT_NUM** is set to=17.

cSRX 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. You can set any number of parameters to take effect when the cSRX image launches. You can pass configuration settings in the YAML file or environment variables to the cSRX when it launches at boot time.

Table 5 on page 18 summarizes the list of available cSRX environment variables.

Table 5: Summary of cSRX Environment Variables

Environment Variable	Mandatory	Description
CSRX_AUTO_ASSIGN_IP	Optional	Automatically configure cSRX **ge-0/0/x** IP address based on IP address of cSRX container when the cSRX works in routing mode. Multus CNI is supports to create more Pod interfaces in Kubernetes. If set to **yes**, the Pod interface IP address is automatically assigned to cSRX revenue port.
CSRX_MGMT_PORT_REORDER	Optional	If set to **yes**, the last Pod interface is changed to management interface. Else, the first Pod interface is management interface.
CSRX_TCP_CKSUM_CALC	Optional	If set to **yes**, cSRX re-compute to correct TCP checksum in packets.
CSRX_LICENSE_FILE	Optional	If set, license file is loaded through ConfigMap.
CSRX_JUNOS_CONFIG	Optional	If set, initial configuration of cSRX is loaded through ConfigMap.
CSRX_SD_HOST	Optional	It is used to define SD server IP address or FQDN address.
CSRX_SD_USER	Optional	It is used to define SD server login account name.
CSRX_SD_PASSWORD	Optional	It is used to define SD server login account password.
CSRX_SD_DEVICE_IP	Optional	It is used to define cSRX management IP address, which is used by SD to connect to cSRX. Else it uses Port IP address.
CSRX_SD_DEVICE_PORT	Optional	It is used to define cSRX management Port, which is used by SD to connect to cSRX. Else it will use default port number **22**.
CSRX_FORWARD_MODE	Optional	It is used in traffic forwarding mode. **"routing"** | **"wire"**
CSRX_PACKET_DRIVER	Optional	It is used in Packet I/O driver. **"poll"** | **"interrupt"**

Table 5: Summary of cSRX Environment Variables *(continued)*

Environment Variable	Mandatory	Description
CSRX_ROOT_PASSWORD	Optional	Initial root account password to log in to the cSRX container using SSH. No default root password ***string***
CSRX_CTRL_CPU	Optional	CPU mask, indicating which CPU is running the cSRX control plane daemons (such as nsd, mgd, nstraced, utmd, and so on). No CPU affinity ***hex value***
CSRX_DATA_CPU	Optional	CPU mask, indicating which CPU is running the cSRX data plane daemon (srxpfe). No CPU affinity ***hex value***
CSRX_ARP_TIMEOUT	Optional	ARP entry timeout value for the control plane ARP learning or response. ***decimal value*** Same as the Linux host
CSRX_NDP_TIMEOUT	Optional	NDP entry timeout value for the control plane NDP learning or response. ***decimal value*** Same as the Linux host
CSRX_PORT_NUM	Optional	Number of interfaces you need to add to container. Default is 3, maximum is 17 (which means 1 management interfaces and 16 data interfaces)

Adding License key File

You can import saved local license key file to cSRX Pod using environment variable **CSRX_LICENSE_FILE** using Kubernetes ConfigMaps.

1. Save the license key file in a text file.

2. Create ConfigMap in Kubernetes.

   root@kubernetes-master:~#kubectl create configmap csrxconfigmap --from-file=<file path>/var/tmp/csrxlicensing

3. Create cSRX using ConfigMaps to import the user defined configuration

   ```
   ---
   deployment.spec.template.spec.containers.
    env:
      - name: CSRX_LICENSE_FILE
        value: "/var/local/config/.csrxlicense"
    volumeMounts:
    - name: lic
        mountPath: "/var/local/config"
   deployment.spec.template.spec.
         volumes:
         - name: lic
    configMap:
      name: csrxconfigmap
      items:
        - key: csrxlicensing
          path: csrxlicensing
   ---
   ```

4. Run the following command to create cSRX deployment using yaml file.

   root@kubernetes-master:~#kubectl apply -f csrx.yaml

5. Login to cSRX pods to verify the license installed

   root@kubernetes-master:~#kubectl exec -it csrx bash

   root@csrx:~#cli

   root@csrx>show system license

Setting Root Password

You can set root password using Kubernetes secrets.

1. Create a generic secret in Kubernetes csrx home namespce.

   root@kubernetes-master:~#kubectl create secret generic csrxrootpasswd --fromliteral= CSRX_ROOT_PASSWORD=XXXXX

2. Run the following command to verify the password is created.

   root@kubernetes-master:~#kubectl describe secret csrxrootpasswd

3. Run the following command to use Kubernetes Secrets to save root password in csrx deployment yaml file.

   ```
   ---
   deployment.spec.template.spec.containers.
   env:
   - name: CSRX_ROOT_PASSWORD
   valueFrom:
   secretKeyRef:
   name: csrxrootpasswd
   key: CSRX_ROOT_PASSWORD
   ---
   ```

4. Run the following command to create cSRX deployment using yaml file.

   root@kubernetes-master:~#kubectl apply -f csrx.yaml

RELATED DOCUMENTATION

Downloading cSRX Software

To download the cSRX software:

1. Download the cSRX software image from the Juniper Networks website. The filename of the downloaded cSRX software image must not be changed to continue with the installation.

2. You can either download the cSRX image file normally using the browser or use the URL to download the image directly on your device as in the following example:

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

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

   root@csrx-ubuntu3:/var/tmp# curl -o /var/tmp/images/junos-csrx-docker-20.3R1.10.img “https://cdn.juniper.net/software/csrx/20.2R1.10/junos-csrx-docker-20.3R1.10.img?SM_USER=user&__gda__=1595350694_5dbf6e62442de6bf14079d05a72464d4”

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

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

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

4. Load the downloaded cSRX image from the download site to the local registry using the following command.

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

```
e758932b9168: Loading layer [==================================================>]
    263MB/263MB
23f7a9961879: Loading layer [==================================================>]
  14.51MB/14.51MB
1e4139e6fa81: Loading layer [==================================================>]
  270.3MB/270.3MB
10334b424f86: Loading layer [==================================================>]
   16.9kB/16.9kB
202ebb2f1137: Loading layer [==================================================>]
   2.56kB/2.56kB
bc4a16173327: Loading layer [==================================================>]
```

```
  1.536kB/1.536kB
8f9a9945544a: Loading layer [==================================================>]
  2.048kB/2.048kB
Loaded image: csrx:20.2R1.10
```

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

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

```
REPOSITORY                        TAG                 IMAGE ID            CREATED
             SIZE
csrx                              20.2R1.10           88597d2d4940        2 weeks
 ago         534MB
```

Automate Initial Configuration Load with Kubernetes ConfigMap

IN THIS SECTION

Loading Initial Configuration with Kubernetes ConfigMap

ConfigMap is Kubernetes standard specification.

ConfigMaps allow you to decouple configuration artifacts from image content to keep containerized applications portable. cSRX use ConfigMaps to load initial configuration file at cSRX container startup.

You can also add license from license key file using the steps similar to loading the initial configuration file in kubernetes.

To create cSRX ConfigMap according to cSRX initial configurations:

1. Create the csrx.yaml file on Kubernetes-master and add the text content to deploy cSRX Pod with ConfigMap:

```
--------
apiVersion: v1
kind: ConfigMap
metadata:
  name: csrx-config-map
  data:
     csrx_config: | interfaces { ge-0/0/0 { unit 0; } ge-0/0/1 { unit 0; } }
security { policies { default-policy { permit-all; } } zones { security-zone
trust { host-inbound-traffic { system-services { all; } protocols { all; } }
interfaces { ge-0/0/0.0; } } security-zone untrust { host-inbound-traffic {
system-services { all; } protocols { all; } } interfaces { ge-0/0/1.0; } } } }
```

root@kubernetes-master:~#kubectl create -f pod_with_configmap.txt

```
------------------
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: csrx
    securityContext:
       privileged: true
    image: csrx-image:20.3
    env:
    - name: CSRX_ROOT_PASSWORD
      value: "xxxxx"
    - name: CSRX_HUGEPAGES
      value: "no"
    - name: CSRX_PACKET_DRIVER
      value: "interrupt"
    - name: CSRX_FORWARD_MODE
      value: "routing"
    volumeMounts:
    - name: disk
      mountPath: "/dev"
    - name: config
      mountPath: "/var/jail"
  volumes:
  - name: disk
    hostPath:
     path: /dev
     type: Directory
  - name: config
    configMap:
      name: csrx-config-map
      items:
      - key: csrx_config
        path: csrx_config
------------------
```

2. Run the following command to create csrx using yaml file.

   root@kubernetes-master:~#kubectl apply -f csrx.yaml

3. Run the following command to start cSRX in CLI mode

   root@kubernetes-master:~#kubectl exec -it csrx bash

   root@csrx:~#cli

   root@csrx#configure

```
Entering configuration mode
```

4. After cSRX Pod startup, you can check cSRX initial configuration from cSRX CLI.

   root@csrx> show

```
## Last changed: 2019-10-18 01:53:36 UTC
version "20190926.093332_rbu-builder.r1057567 [rbu-builder]";
interfaces {
    ge-0/0/0 {
        unit 0 {
            family inet {
                address 20.0.0.11/24;
            }
        }
    }
    ge-0/0/1 {
        unit 0 {
            family inet {
                address 30.0.0.11/24;
            }
        }
    }
}
security {
    policies {
        default-policy {
            permit-all;
        }
    }
    zones {
        security-zone trust {
            host-inbound-traffic {
                system-services {
                    all;
                }
                protocols {
                    all;
                }
            }
            interfaces {
                ge-0/0/0.0;
            }
        }
        security-zone untrust {
            host-inbound-traffic {
                system-services {
                    all;
                }
                protocols {
                    all;
                }
            }
            interfaces {
                ge-0/0/1.0;
            }
        }
    }
}
```

cSRX Pod With External Network

IN THIS SECTION

Understanding cSRX Pod with External Network

You can connect cSRX with external network with two additional interfaces. Both of those interfaces are attached into srxpfe and handled by FLOW.

cSRX can leverage Linux native CNI to connect to external network.

cSRX use Multus plugin to support multiple interfaces connect to the external network. Applications which monitor network traffic are directly connected to the physical network. You can use the **macvlan** network driver to assign a MAC address to each container’s virtual network interface, making it appear to be a physical network interface directly connected to the physical network. In this case, you need to designate a physical interface on your Docker host to use for the **macvlan**, as well as the subnet and gateway of the **macvlan**. You can even isolate your macvlan networks using different physical network interfaces.

Connecting cSRX to External Network

macvlan functions like a switch that is already connected to the host interface. A host interface gets enslaved with the virtual interfaces sharing the physical device but having distinct MAC addresses. Since each macvlan interface has its own MAC address, it makes it easy to use with existing DHCP servers already present on the network.

To connect cSRX with external network using **macvlan**:

Figure 2: Connecting cSRX to External Network with Macvlan Plugin

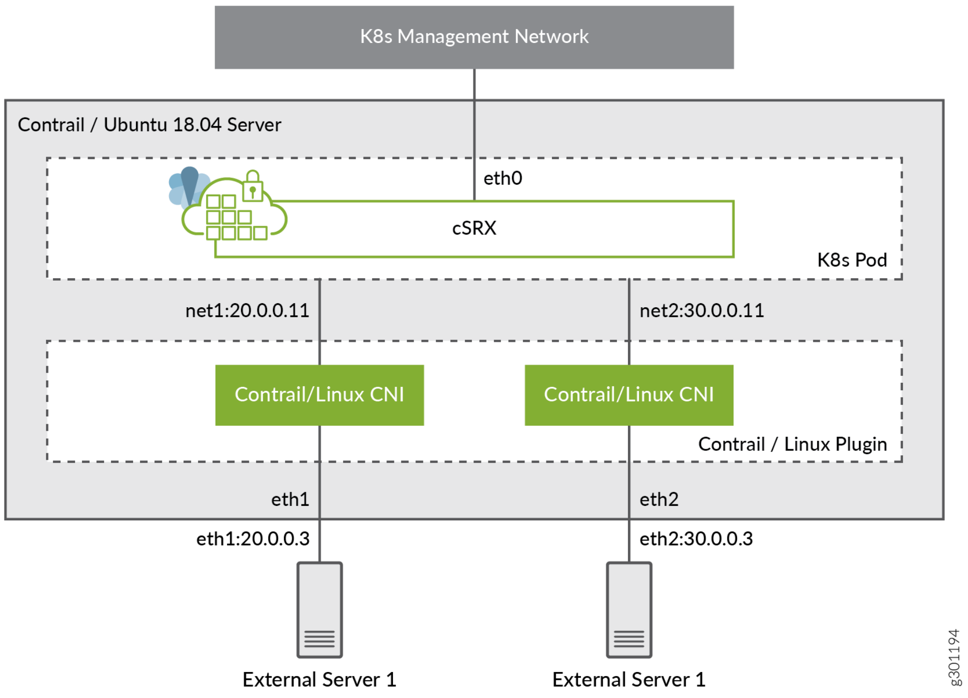
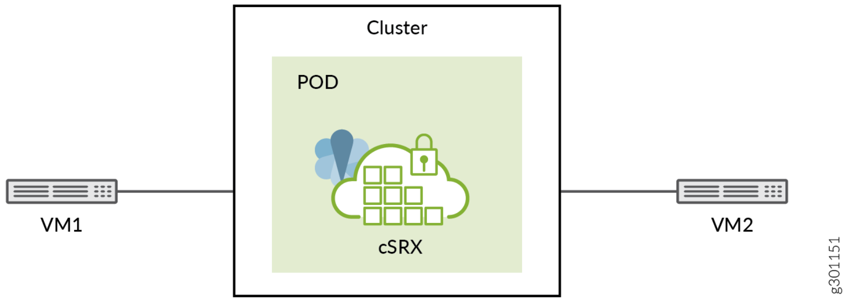


Figure 3: cSRX in External Network



1. Create the network-conf-1.yaml file and add the text content.

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: network-conf-1
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "macvlan",
      "master": "eth1",
      "mode": "bridge",
      "ipam": {
        "type": "static",
  "addresses": [
   {
    "address": "20.0.0.10/24",
    "gateway": "20.0.0.2"
   }
  ],
  "routes": [
   { "dst": "0.0.0.0/0" },
   { "dst": "30.0.0.0/24", "gw": "20.0.0.11" }
  ]
      }
    }'
```

2. Create the network-conf-1-1.yaml file and add the text content. .

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: network-conf-1-1
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "macvlan",
      "master": "eth1",
      "mode": "bridge",
      "ipam": {
        "type": "static",
  "addresses": [
   {
    "address": "20.0.0.11/24",
    "gateway": "20.0.0.2"
   }
  ],
  "routes": [
   { "dst": "0.0.0.0/0" }
  ]
      }
    }'
```

3. Create the network-conf-2-1.yaml and add the text content. .

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: network-conf-2-1
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "macvlan",
      "master": "eth2",
      "mode": "bridge",
      "ipam": {
        "type": "static",
  "addresses": [
   {
    "address": "30.0.0.11/24",
    "gateway": "30.0.0.2"
   }
  ],
  "routes": [
   { "dst": "0.0.0.0/0" }
  ]
      }
    }'
```

4. Create the network-conf-2.yaml file and add the text content.

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: network-conf-2
```

```
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "macvlan",
      "master": "eth2",
      "mode": "bridge",
      "ipam": {
        "type": "static",
  "addresses": [
   {
    "address": "30.0.0.10/24",
    "gateway": "30.0.0.2"
   }
  ],
  "routes": [
   { "dst": "0.0.0.0/0" },
   { "dst": "20.0.0.0/24", "gw": "30.0.0.11" }
  ]
      }
    }'
```

5. Create the csrx.yaml file and add the text content.

```
apiVersion: v1
kind: Pod
metadata:
  name: csrx
  annotations:
    k8s.v1.cni.cncf.io/networks: network-conf-1-1,network-conf-2-1
spec:
  containers:
  - name: csrx
    securityContext:
       privileged: true
    image: csrx-images:20.2
    env:
    - name: CSRX_ROOT_PASSWORD
      value: "xxxxx"
    - name: CSRX_HUGEPAGES
      value: "no"
    - name: CSRX_PACKET_DRIVER
      value: "interrupt"
    - name: CSRX_FORWARD_MODE
```

```
        value: "routing"
      volumeMounts:
      - name: disk
        mountPath: "/dev"
    volumes:
    - name: disk
      hostPath:
       path: /dev
       type: Directory
```

Configuring Nodeport service for cSRX Pods

You can deploy cSRX with Nodeport service type. All the traffic will be forward to worker node by Kubernetes in the external network.

To create a NodePort service:

1. Create the cSRX Pod yaml file and expose it as service on NodePort.

```
------------------
apiVersion: v1
kind: Service
metadata:
  name: csrx1
spec:
  selector:
    app: csrx1
  ports:
    - name: ssh
      port: 22
      nodePort: 30122
  type: NodePort
---
```

2. To access cSRX:

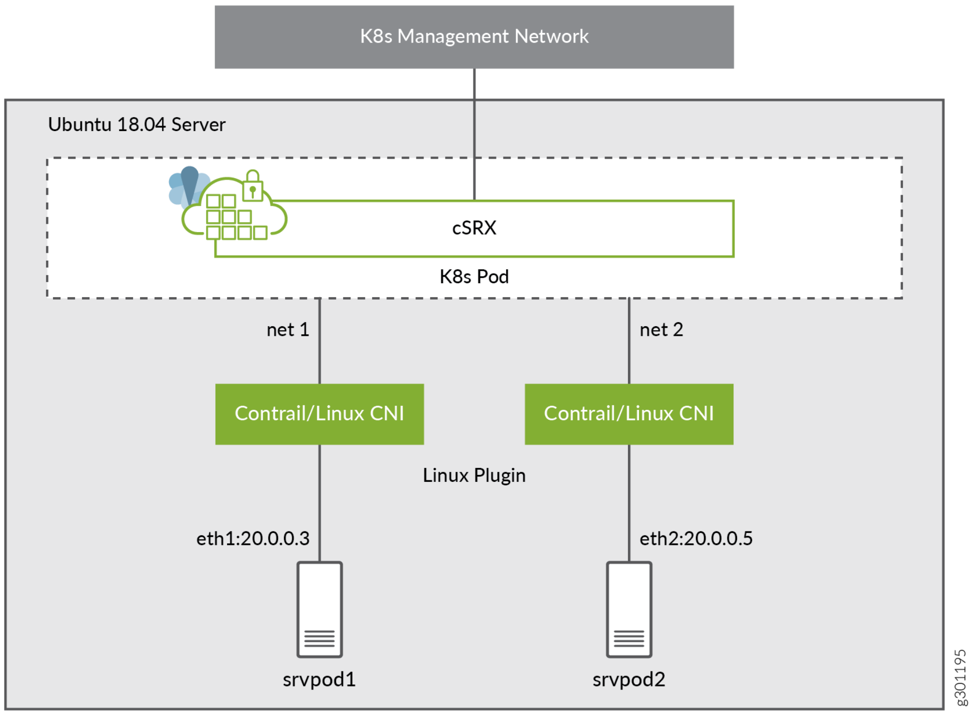
   root@kubernetes-master:~#ssh -p 30122 root@192.168.42.81

cSRX Pod With Internal Network

With **bridge** plugin, all containers on the same host are plugged into a bridge (virtual switch) that resides in the host network name space. The containers receive one end of the veth pair with the other end connected to the bridge. An IP address is only assigned to one end of the veth pair in the container. The bridge itself can also be assigned an IP address, turning it into a gateway for the containers. Alternatively, the bridge can function in L2 mode and would need to be bridged to the host network interface (if other than container-to-container communication on the same host is desired). The network configuration specifies the name of the bridge to be used.

To connect cSRX with external network using **bridge**:

Figure 4: Connecting cSRX to Internal Network with Bridge Plugin



1. Create the network-conf-1-1.yaml file and add the text content.

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: network-conf-1-1
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "bridge",
      "bridge": "south-bridge",
      "promiscMode": true,
      "ipam": {
        "type": "static",
  "addresses": [
   {
    "address": "20.0.0.20/24",
    "gateway": "20.0.0.1"
   }
  ],
  "routes": [
   { "dst": "0.0.0.0/0" }
  ]
      }
    }'
```

2. Create the network-conf-2-1.yaml file and add the text content.

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: network-conf-2-1
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "bridge",
      "bridge": "north-bridge",
      "promiscMode": true,
      "ipam": {
        "type": "static",
  "addresses": [
   {
    "address": "20.0.0.30/24",
    "gateway": "20.0.0.1"
   }
  ],
  "routes": [
   { "dst": "0.0.0.0/0" }
  ]
      }
    }'
```

3. Create the srv-pod-1.yaml file and add the text content.

```
apiVersion: v1
kind: Pod
metadata:
  name: srv-pod-1
  annotations:
    k8s.v1.cni.cncf.io/networks: network-conf-1@north0
spec:
  containers:
  - name: srv-pod-1
    securityContext:
      privileged: true
    image: docker.io/centos/tools:latest
    command:
    - /sbin/init
```

4. Create the csrx.yaml file and add the text content.

```
apiVersion: v1
kind: Pod
metadata:
  name: csrx
  annotations:
    k8s.v1.cni.cncf.io/networks: network-conf-1-1,network-conf-2-1
spec:
  containers:
  - name: csrx
    securityContext:
       privileged: true
    image: csrx-images:20.2
    env:
    - name: CSRX_ROOT_PASSWORD
```

```
        value: "xxxxx"
      - name: CSRX_HUGEPAGES
        value: "no"
      - name: CSRX_PACKET_DRIVER
        value: "interrupt"
      - name: CSRX_FORWARD_MODE
        value: "wire"
      volumeMounts:
      - name: disk
        mountPath: "/dev"
    volumes:
    - name: disk
      hostPath:
       path: /dev
       type: Directory
```

5. Create the srv-pod-3.yaml file and add the text content.

```
apiVersion: v1
kind: Pod
metadata:
  name: srv-pod-3
  annotations:
    k8s.v1.cni.cncf.io/networks: network-conf-2@north0
spec:
  containers:
  - name: srv-pod-3
    image: docker.io/centos/tools:latest
    command:
    - /sbin/init
```

cSRX Deployment in Kubernetes

IN THIS SECTION

cSRX Installation on Kubernetes Linux Server

Prerequisites

Following are the prerequisites required for installing cSRX on one master node and 'n' number of worker nodes. Before you begin the installation:

- Install kubeadm tool on both master and worker nodes to create a cluster. See Install Kubeadm
- Install and configure Docker on Linux host platform to implement the Linux container environment, see Install Docker for installation instructions on the supported Linux host operating systems.
- Verify the system requirement specifications for the Linux server to deploy the cSRX, see "Requirements for Deploying cSRX on Kubernetes" on page 17.
- Download cSRX software, see "Downloading cSRX Software" on page 22.

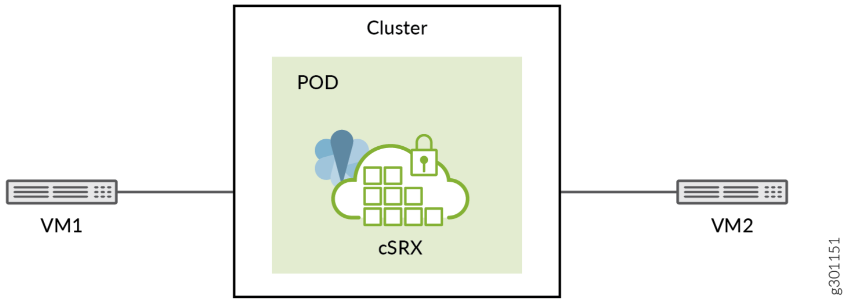
Deploying cSRX Pod

You can create cSRX as a Pod in routing mode and secure-wire mode to send traffic from one virtual machine to another virtual machine. You can define multiple virtual networks and connect cSRX interfaces to those virtual networks.

The network attachment definition is created with plugin **ipam** type as **host-local** which allocates IPv4 and IPv6 addresses out of a specified address range to ensure the uniqueness of IP addresses on a single host. The **ipam** type as **static** assigns IPv4 and IPv6 addresses statically to container.

To deploy cSRX with Kubernetes:

Figure 5: Deploying cSRX



1. Create network attachment definition for cSRX-eth1, cSRX-eth2 with **type: bridge** . For details on **type: bridge** and **type: macvlan** networks, see "cSRX Pod With External Network" on page 28.

```
------------------
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: network-conf-1
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "bridge",
      "bridge": "br-1",
      "isDefaultGateway": true,
      "promiscMode": true,
      "ipam": {
        "type": "host-local",
        "ranges": [
       [
       {
        "subnet": "10.10.0.0/16",
        "rangeStart": "10.10.1.20",
        "rangeEnd": "10.10.3.50"
       }
       ]
     ],
        "routes": [
            { "dst": "0.0.0.0/0" }
        ]
      }
```

```
    }'
---------------------
```

```
------------------
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: network-conf-1-1
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "bridge",
      "bridge": "br-2",
      "isDefaultGateway": true,
      "promiscMode": true,
      "ipam": {
        "type": "host-local",
        "ranges": [
       [
       {
        "subnet": "55.0.0.0/16",
        "rangeStart": "55.0.0.11",
        "rangeEnd": "55.0.0.21"
       }
       ]
     ],
        "routes": [
            { "dst": "0.0.0.0/0" }
        ]
      }
    }'
------------------
```

To create network interfaces with **type: macvlan**.

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: network-conf-1-1
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "macvlan",
      "master": "eth1",
      "mode": "bridge",
      "ipam": {
        "type": "static",
  "addresses": [
   {
    "address": "20.0.0.11/24",
    "gateway": "20.0.0.2"
   }
  ],
  "routes": [
   { "dst": "0.0.0.0/0" }
  ]
      }
    }'
```

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: network-conf-2-1
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "macvlan",
      "master": "eth2",
      "mode": "bridge",
      "ipam": {
        "type": "static",
  "addresses": [
   {
    "address": "30.0.0.11/24",
    "gateway": "30.0.0.2"
   }
  ],
  "routes": [
   { "dst": "0.0.0.0/0" }
  ]
      }
    }'
```

2. Create the csrx-deployment.yaml file on Kubernetes-master using **kind: Deployment**. cSRX as **kind: Deployment** is used to create ReplicaSet, Scaling, Rollout, Rollback in Kubernetes in this topic.

```
------------------
apiVersion: apps/v1
kind: Deployment
metadata:
  name: csrx-deployment
  labels:
    app: firewall
spec:
  replicas: 5
  selector:
    matchLabels:
      app: firewall
  template:
metadata:
  labels:
        app: firewall
      annotations:
        k8s.v1.cni.cncf.io/networks:
        network-conf-1, network-conf-1-1
spec:
  containers:
  - name: csrx
        securityContext:
 privileged: true
    image: csrx-images:20.2
     env:
    - name: CSRX_SIZE
      value: "large"
    - name: CSRX_HUGEPAGES
      value: "no"
    - name: CSRX_PACKET_DRIVER
      value: "interrupt"
    - name: CSRX_FORWARD_MODE
      value: "routing"
    volumeMounts:
    - name: disk
      mountPath: "/dev"
  volumes:
  - name: disk
    hostPath:
      path: /dev
      type: Directory
------------------
```

3. View the cSRX deployment:

root@kubernetes-master:~#kubectl get deployment csrx-deployment

```
NAME                       READY    UP-TO-DATE    AVAILABLE    AGE
csrx-deployment            5/5      5             5            119m
```

cSRX Image Upgrade Using Deployment Rollout

You can upgrade the cSRX software image using Kubernetes Deployment rollout.

1. Run the following command to upgrade cSRX image using Kubernetes Deployment name in the cSRX Pod:

   root@kubernetes-master:~#kubectl set image deployment csrx-deployment csrx=<new-csrx-image>

   ```
   NAME                       READY    UP-TO-DATE    AVAILABLE    AGE
   csrx-deployment            5/5      5             5            119m
   ```

2. Run the following command to monitor rollout status:

   root@kubernetes-master:~#kubectl rollout history deployment csrx-deployment

   root@kubernetes-master:~#kubectl rollout status -w deployment csrx-deployment

   ```
   Waiting for deployment "csrx-deployment" rollout to finish: 1 old replicas are
    pending termination...

   Waiting for deployment "csrx-deployment" rollout to finish: 1 old replicas are
    pending termination...
   deployment "csrx-deployment" successfully rolled out
   ```

   You can verify the upgraded image version by logging into the newly created cSRX Pods.

cSRX Image Rollback

The cSRX image can be rolled back to previous version using Kubernetes Deployment rollout components.

1. Rollack cSRX image using Kubernetes Deployment **rollout undo**:

   root@kubernetes-master:~#kubectl rollout history deployment csrx-deploy

2. Rollback to previous Deployment.

   root@kubernetes-master:~#kubectl rollout undo deployment csrx-deploy

3. Rollback to a specified version.

   root@kubernetes-master:~#kubectl rollout undo deployment csrx-deploy --to-version=2

4. Monitor the old cSRX Pods are terminated and new cSRX Pods are created.

   root@kubernetes-master:~#kubectl rollout history deployment csrx-deploy

   root@kubernetes-master:~#kubectl rollout status -w deployment csrx-deploy

   You can verify the image version that has been rolled back by logging into the newly created cSRX Pod.

Scaling cSRX Deployment

To scale the cSRX deployment:

1. Ensure to have cSRX Pods created in **kind: deployment** running in Kubernetes cluster.

   root@kubernetes-master:~#kubectl describe deployment csrx-deployment

2. Scale up or down by changing the **replicas** number:

   root@kubernetes-master:~#kubectl scale deployment csrx-deployment --replicas=2

3. View the pods:

   root@kubernetes-master:~#kubectl get pod

```
NAME                               READY   STATUS    RESTARTS   AGE
csrx-deployment-547fcf68dd-7hl7r   1/1     Running   0          8m8s
csrx-deployment-547fcf68dd-xbg4b   1/1     Running   0          35s
```

3

CHAPTER

Managing cSRX

cSRX Service With Load Balancing Support

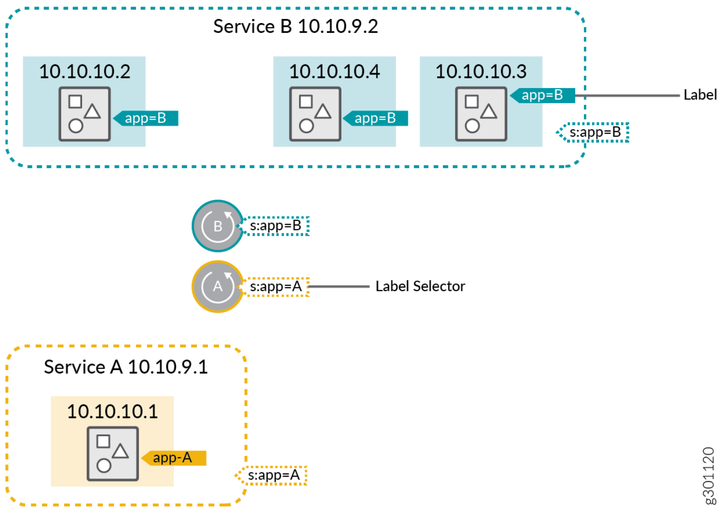
IN THIS SECTION

Understanding cSRX as Kubernetes Service with Load Balancing Support

cSRX Pod is identified with predefined selectors and exposed with supported load balancer to distribute traffic among different cSRX Pods. The standard load balancer is ingress controller, external load balancer or cluster IP.

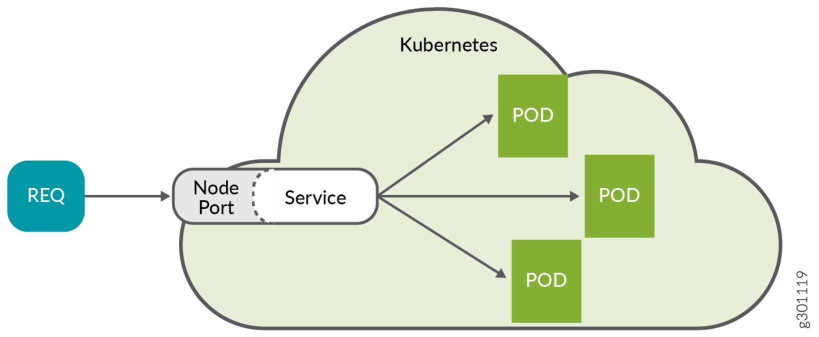
A Service enables network access to a set of Pods in Kubernetes. Services select Pods based on their labels. When a network request is made to the service, it selects all Pods in the cluster matching the service's selector, chooses one of them, and forwards the network request to it. A deployment is responsible for keeping a set of pods running.

Figure 6: Services and Labels



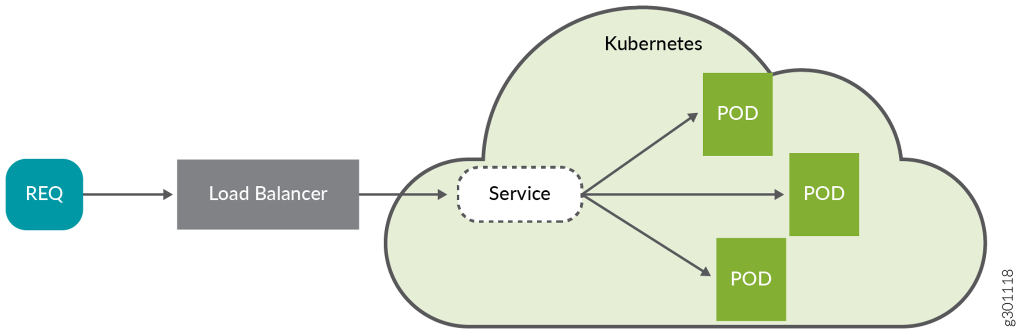
Service is to group a set of Pod endpoints into a single resource. By default, clients inside the cluster can access Pods in the Service using cluster IP address. A client sends a request to the IP address, and the request is routed to one of the Pods in the Service. The types of Services are ClusterIP (default), NodePort, LoadBalancer, and ExternalName.

Figure 7: NodePort



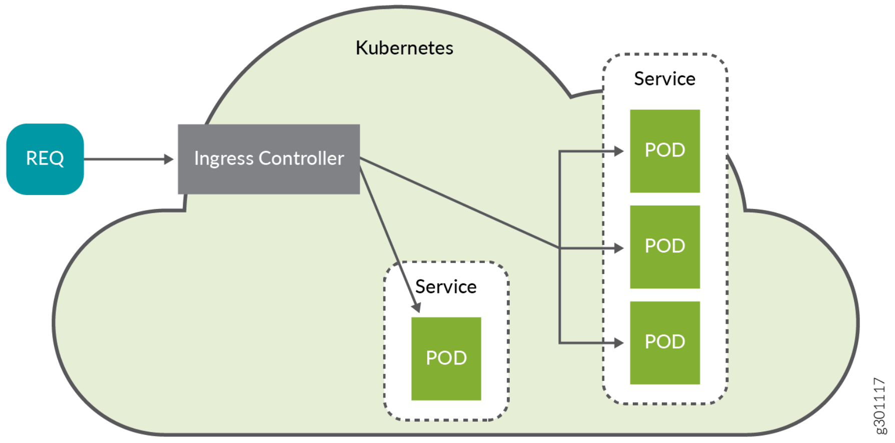
When you set a service's type to NodePort, that service starts to listen on a static port on every node in the cluster. So, you can reach the service through any node's IP address and the assigned port.

Figure 8: LoadBalancer



When you set a service's type to Load Balancer, it exposes the service externally. However, to use it, you need to have an external load balancer. The external load balancer needs to be connected to the internal Kubernetes network on one end and opened to public-facing traffic on the other in order to route incoming requests.

Figure 9: Ingress Controller



An Ingress Controller watches for new services within the cluster and is able to dynamically create routing rules for them. An Ingress object is an independent resource, apart from Service objects, that configures external access to service’s pods. You can define the Ingress, after the Service has been deployed, to connect it to external traffic. This way You can isolate service definitions from the logic of how clients connect to them. L7 routing is one of the core features of Ingress, allowing incoming requests to be routed to the exact pods that can serve them based on HTTP characteristics such as the requested URL path. Other features include terminating TLS, using multiple domains, and load balancing traffic.

Nginx ingress controller is supported to view the traffic distribution among different cSRX Pods. For more details, see Set Up Ingress on Kubernetes Using Nginx Controller.

Configuring Ingress Service for cSRX Pods

Service is used by cSRX to connect application with cSRX Pods. cSRX Service is standard Kubernetes service, in which, the load is balanced to different cSRX Pods, and the Pods are located at different work nodes. It also monitors the backend cSRX Pod and selects working cSRX Pod according to Kubernetes Pod labels. You can use YAML file to create a cSRX service.

To create a cSRX service:

1. Create the yaml file and add the following text content:

   ```
   ------------------
   apiVersion: v1
   kind: Service
   metadata:
     labels:
       app: firewall
     name: firewall
   spec:
     selector:
       app:firewall
     ports:
     - name: port-1
       port: 80
       protocol: TCP
       targetPort: 80
   ```

2. Define routing for cSRX Pods. Ingress will co-operate with Ingress controller to route outside traffic into cSRX service, then into cSRX Pods. Create a file named ingress.yaml.

   ```
   apiVersion: networking.k8s.io/v1beta1
   kind: Ingress
   metadata:
     name: web-ingress
     namespace: default
   spec:
    rules:
    - host: foo.bar
      http:
       paths:
       - path: /
          backend:
            serviceName: firewall
            servicePort: 80
   ```

   Traffic routes to cSRX interface on **ge-0/0/0**.

3. View the cSRX service.

   root@kubernetes-master:~#kubectl get svc -A

```
NAMESPACE       NAME            TYPE           CLUSTER-IP       EXTERNAL-IP
PORT(S)                      AGE
default         csrx-service    ClusterIP      10.102.115.211   <none>
80/TCP                       13d
default         kubernetes      ClusterIP      10.96.0.1        <none>
443/TCP                      75d
default         nginx           NodePort       10.110.8.221     <none>
80:31454/TCP                 18d
default         test-service    ClusterIP      10.108.236.26    <none>
80/TCP                       11d
kube-system     kube-dns        ClusterIP      10.96.0.10       <none>
53/UDP,53/TCP,9153/TCP       75d
```

4. View the Pod.

   root@kubernetes-master:~#kubectl get pod -A

```
NAMESPACE       NAME                                                   READY
  STATUS    RESTARTS   AGE
default         csrx-deployment-86f49b8dcf-7zzq9                       1/1
  Running   0          11d
default         csrx-deployment-86f49b8dcf-dm6nv                       1/1
  Running   0          11d
```

4

CHAPTER

Configuring cSRX

cSRX Image with Packaged Pre-Installed Signatures

IN THIS SECTION

Understanding Pre-Installed Signatures

To support pre-installed signatures package in cSRX image, a Docker file is placed in localhost repository to help user compile cSRX with installed signatures. With the new image, you can launch cSRX Pod which protects workload immediately after container is launched.

The supported functions for signature packaging are:

- Intrusion Detection and Prevention (IDP)
- Application Identification (AppID)
- Unified Threat Management (UTM)

Repackaging cSRX Image with Signatures

- Ensure to have the cSRX image placed in the local repository or any other Docker registry.
- Ensure to include license file together with Docker file.

To repackage cSRX image with signatures:

1. Create DockerFile.

   root@host>cat Dockerfile

   ```
   FROM localhost:5000/csrx
   ARG CSRX_BUILD_WITH_SIG=yes
   ```

```
ENV CSRX_LICENSE_FILE=/var/local/.csrx_license
COPY csrx.lic $CSRX_LICENSE_FILE
RUN ["/etc/rc_build.local"]
CMD ["/etc/rc.local","init"]
```

2. Repackage image to include APPID and IDP signature.

   root@host>docker build -t localhost:5000/csrx-sig

3. Push the image to the registry.

   root@host>docker push localhost:5000/csrx-sig

   The new cSRX image **localhost:5000/csrx-sig:latest** is ready to use.

Downloading of Juniper Signature Pack

You can download the signature pack from the Juniper Signature Repository directly when cSRX doesn't have a preinstalled signature pack.

1. To download the signature pack from Juniper Signature Repository:

   root@host> request services application-identification download

   root@host> request security idp security-package download

Downloading Signature Pack through Proxy Server

You can download the signature pack through a proxy server. AppIDD and IDPD processes first connects to the configured proxy server. The proxy server then communicates with the signature pack download server and provides the response to the process running on the device.

To download the signature pack through the proxy server:

1. Configure the proxy server so that the IP address of the proxy server is reachable from cSRX.

2. Run the following command to enter the configuration mode from the CLI.

   root@host> configure

```
Entering configuration mode
```

[edit]

root@host#

3. Configure the proxy server profile on cSRX using the IP address and port of the proxy server.

   root@host#set services proxy profile appid_sigpack_proxy protocol http host 4.0.0.1

   root@host#set services proxy profile appid_sigpack_proxy protocol http port 3128

4. Attach the profile to AppID and IDP.

   root@host#set services application-identification download proxy-profile appid_sigpack_proxy

   root@host#set security idp security-package proxy-profile appid_sigpack_proxy

5. Commit the configuration.

   root@host#commit and-quit

   ```
   commit complete
   Exiting configuration mode
   ```

6. Download the IDP and APPID signature pack through proxy server.

   root@host>request services application-identification download

   root@host>request security idp security-package download

To verify that the download is happening through the proxy server:

1. Verify the logs in the proxy server.

   [root@srxdpi-lnx39 squid]# cat /var/log/squid/access.log

   ```
   1593697174.470   1168 4.0.0.254 TCP_TUNNEL/200 5994 CONNECT
   signatures.juniper.net:443 - HIER_DIRECT/66.129.242.156 -
   1593697175.704   1225 4.0.0.254 TCP_TUNNEL/200 11125 CONNECT
   signatures.juniper.net:443 - HIER_DIRECT/66.129.242.156 -
   1593697176.950   1232 4.0.0.254 TCP_TUNNEL/200 5978 CONNECT
   signatures.juniper.net:443 - HIER_DIRECT/66.129.242.156 -
   1593697178.195   1236 4.0.0.254 TCP_TUNNEL/200 11188 CONNECT
   signatures.juniper.net:443 - HIER_DIRECT/66.129.242.156 -
   1593697198.337   1243 4.0.0.254 TCP_TUNNEL/200 6125 CONNECT
   signatures.juniper.net:443 - HIER_DIRECT/66.129.242.156 -
   ```

   In cSRX, the TLS protocol is used and traffic the through proxy server is encrypted.

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 which is accessed by cSRX exposed service port.

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

2. Start the CLI as root user.

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

   ```
   root#cli
   ```

```
root@>
```

3. Verify the interfaces.

   root@> show interfaces

```
Physical interface: ge-0/0/1, Enabled, Physical link is Up
  Interface index: 100
  Link-level type: Ethernet, MTU: 1514
  Current address: 02:42:ac:13:00:02, Hardware address: 02:42:ac:13:00:02
Physical interface: ge-0/0/0, Enabled, Physical link is Up
  Interface index: 200
  Link-level type: Ethernet, MTU: 1514
  Current address: 02:42:ac:14:00:02, Hardware address: 02:42:ac:14:00:02
```

4. Enter configuration mode.

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

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

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

6. Configure the hostname.

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

7. Configure the two traffic interfaces.

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

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

```
[edit]
root@# set security zones security-zone untrust interfaces ge-0/0/0.0
root@# set security zones security-zone trust interfaces ge-0/0/1.0
root@# set security policies default-policy permit-all
```

9. Verify the configuration.

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

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

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

11. (Optional) Use the **show** command to display the configuration for verification.

RELATED DOCUMENTATION

Junos OS for SRX Series

Introducing the Junos OS Command-Line Interface