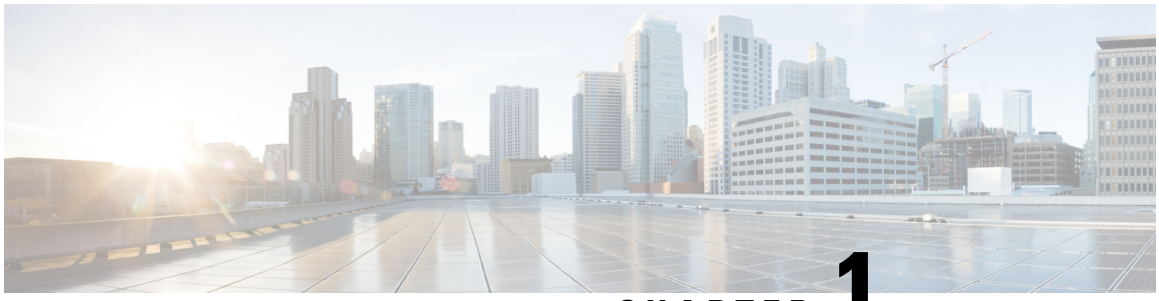
Cisco IOS XRd Setup Guide for AWS

First Published: 2023-08-01

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883



CHAPTER 1

Overview

This module provides an overview of deploying Cisco IOS XRd Router on AWS.

Overview

Cisco IOS® XRd complements existing physical Cisco® router platforms that rely on Cisco IOS XR Software, such as Cisco Network Convergence System routers, Cisco ASR 9000 Series Routers, and Cisco 8000 Series Routers. Service providers can now improve their operations and offerings by leveraging XRd platform to offer new services that require routing functionality in a compact manner. XRd offers the option to deploy routing in a containerized form-factor enabling routing between network functions in the public cloud and those running on-premises in data centers. This deployment offers flexibility as an increasing number of network functions are moved to the public cloud. The Cisco IOS XRd offers greater agility, improved network efficiency at a lower capital cost and operational expenditures. It also provides the ability to efficiently scale up or scale down the network capacity based on demand.

Features of Cisco IOS XRd Router

XRd runs on generic Kubernetes, and can be integrated into your Kubernetes-based platforms such as Amazon AWS EKS (Elastic Kubernetes Service), as described in this document. The XRd software provides the capability to administer routing functionality, similar to other applications operating within the data center or cloud environment.

XRd software, based on IOS XR7 release, is of lightweight design and expedited boot process. XRd supports contemporary operating system programmability features, including YANG models (both native and open configuration) and Model-Driven Telemetry.

XRd software is a derivative of the highly resilient, stable, and feature-rich Cisco IOS XR Software, possessing the same northbound interfaces and management features as Cisco IOS-XR. As a result, XRd seamlessly integrates with the existing monitoring, automation, and orchestration systems.

Cisco IOS XRd is available as:

- Cisco IOS XRd Control Plane
- Cisco IOS XRd vRouter

For more details, see Cisco IOS XRd Data Sheet.

Known Issues

The following table addresses known issues.

Issue	Description	Condition	Workaround
Traffic Reconvergence Delay during XRd vRouter Cloud HA Failover	This issue impacts the XRd vRouter Cloud HA solution, causing a delay in traffic reconvergence when transitioning from the primary to the secondary XRd vRouter instance. The duration can extend up to the ARP cache timeout on the client workload, which is 4 hours for Cisco IOS XR-based workloads. In contrast, Linux-based workloads have a default ARP cache timeout of 30 seconds.	The likelihood of encountering the issue increases when initiating failover immediately after launching or restarting the EC2 instance hosting the Secondary XRd vRouter instance. Standard prerequisites for the XRd vRouter Cloud Hgh Availability solution include the presence of the XRd HA App container application in the Amazon AWS EKS (Elastic Kubernetes Service) cluster and the configuration of Linux networking, VRRP, and Telemetry settings in Cisco IOS XR.	To address the issue, implement the following workaround: clear the ARP cache of the workload after an unsuccessful failover.

Licensing

The licensing scheme for Cisco IOS XRd is based on a simplified subscription model, which consists of:

- Per-instance licensing for the control plane
- Throughput utilization licensing for the virtual router (vRouter)

This licensing structure enables service providers to pay for the resources they consume, and the licenses are activated through Cisco Smart Software Licensing. You must register Cisco IOS XRd within 90 days to retain the ability to modify XRd configurations. If you try to modify the configuration, you will be notified that your configuration mode is locked due to the unlicensed product. However, if a purchased license expires, there is no impact.

Additional information about Cisco Smart Software Licensing can be found at Licensing.

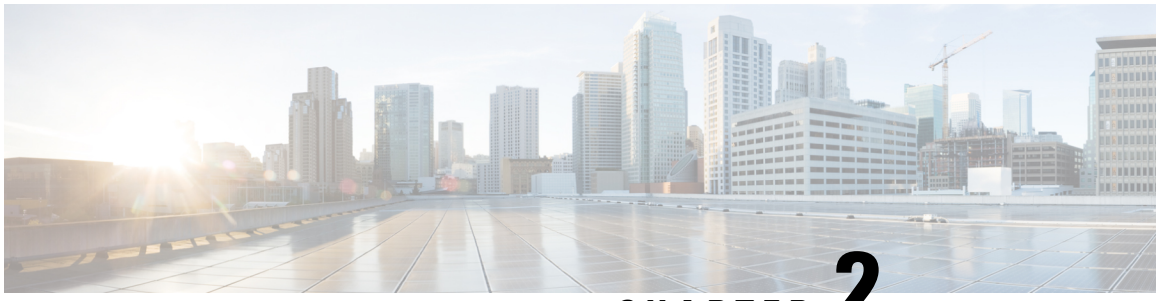
XRd Deployment Procedure

The procedure to deploy and bring up the XRd vRouter or XRd Control Plane on AWS EKS includes the following steps:

1. Create a VPC.
2. Bring Up and Configure an EKS Cluster.
3. Set Up Container Registry.
4. Create an AMI.
5. Create a Worker Node.
6. Helms Chart Repository.

You can use these steps as a reference to deploy XRd as per your use case.

The examples provided in this document use Cisco IOS XR Release 7.8.1 but are applicable to other Cisco IOS XR releases that support Cisco IOS XRd.



CHAPTER 2

Prerequisites

This module provides the prerequisites for deploying Cisco IOS XRd on Amazon EKS.

Prerequisites

Before you begin to deploy the XRd on AWS EKS, ensure that you have the following:

- Basic knowledge of Kubernetes and Helm
- Familiarity with AWS Services, such as:
  - IAM
  - EC2
  - VPC
  - ECR
  - EKS
- Familiarity and availability of tools, such as:
  - AWS CLI tool

    You must configure the AWS CLI tool with an account with administrator permissions. Admin permissions are required to create IAM roles.
  - Standard CLI tools, for example, **tar** and **SSH**
  - Helm
  - kubectl
  - skopeo, or, docker, or podman
- Availability of an EC2 key-pair to access the worker nodes. If you have not created a key-pair already in the region that you are deploying the cluster, create a key-pair using these instructions.

Note Most commands that run on the AWS CLI tool do not require the AWS region to be specified. These commands use the default region from the CLI tool configuration. Commands that require the region to be specified explicitly are indicated as `<region>` in the command example.

Cisco IOS XRd Deployment Requirements

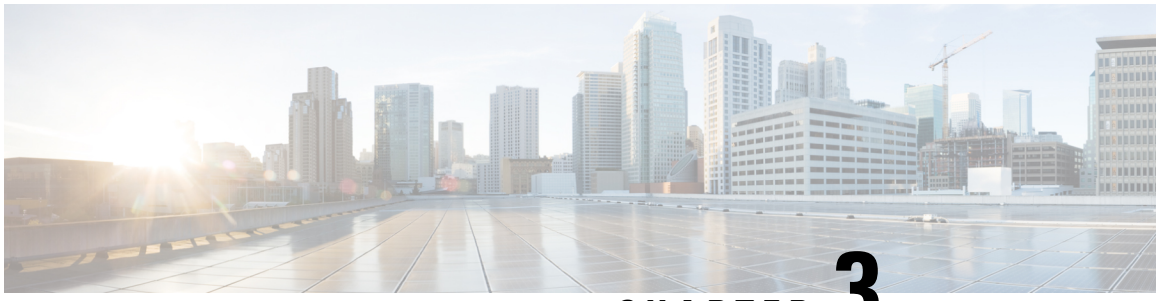
The following table provides the Cisco IOS XRd deployment requirements:

Use Case		**SR-PCE**	**Cloud Router**
Platform		**XRd Control Plane**	**XRd vRouter**
Worker Node	Validated Instance Type	m5.2xlarge	m5[n].24xlarge
	Hyper Threading	No	No
	CPU Isolation	None	Cores 16-23 isolated
	OS	Amazon Linux 2 with EKS Optimizations	
	AMI	Base Amazon Linux 2 with EKS AMI	Modified Amazon Linux 2 with EKS AMI*
	Sysctl Settings	inotify settings, kernel core pattern**	
	Robust core handling to avoid disk exhaustion	A pipe script, rather than a path pattern, can be used to ensure that cores are handled on the host. `systemd-coredump` service can be used to collect core files safely.	
	Hugepages	None	6 x 1 GiB
	Storage	56 GiB	56 GiB
	Extra Kernel Modules	None	UIO (in AL2 but not loaded) igb_uio from the dpdk-mods package.
Cluster	Kubernetes Version	Version 1.25, and versions 1.27 and above.	
	AWS-Node Settings	MAX_ENI=1	
Multus	Version	v4.0.2-eksbuild.1	NA
	Attachment Types	AWS-CNI (default CNI interface only) Host-device	

Use Case		SR-PCE	Cloud Router
Platform		**XRd Control Plane**	**XRd vRouter**
XRd	RAM (excluding hugetlb) - Deployment requirements	4 GiB	10 GiB
	Hugepages - Deployment Requirements	None	3 GiB
	CPU - Deployment Requirements	4 physical CPUs (shared with OS)	12 physical CPUs
	CPU Split	NA	4xCP, 1xDPA main, 7xDP
	CPU Settings		Override cpuset to 12-23
	CPU Mapping		Standard 12 core layout: 12-15 CP, 16 DPA, 17-23 DP
	CPU Requirements		Number of CP cores > 1 must be non-isolated, DP must be isolated always, and full physical CPUs always
	DPA Tuning		Off
	Interface Driver	NA	igb_uio with Write Combine support
	Security Context	Privileged	
	Persistent Volume	6 GiB, "gp2" class, dynamically allocated	
	Scheduling	Pinned to specific worker node	
	Node Exclusivity	No other workloads running on the node	
	Initial Configuration	In workload definition, dependent on worker node (and interface) settings	
	Redundancy Model	Pair of separate, redundant, instances each in a separate StatefulSet	

* For Amazon Linux 2 modification details, see Create an AMI section.

**For inotify settings, and kernel core pattern details, see Tuning section.



CHAPTER 3

Create a VPC

This module describes how to create a Virtual Private Cloud (VPC).

Create a VPC

As an XRd-specific requirement, the data traffic flowing through the XRd instance must be separated from the cluster control-plane traffic. The separation is achieved by using different Virtual Private Cloud (VPC) subnets for the control-plane and data-plane traffic.

Amazon provides a simple CloudFormation template to create a VPC that has two public subnets and two private subnets. An internet gateway is used to give the public subnets full access to the internet, and the private subnets are shielded through a NAT gateway.

Note This template brings up a VPC with access to the internet. If you want to shield the VPC from the internet, you can create a VPC without an internet gateway. In that case, ensure that all the additional requirements described in the AWS documentation on Private Clusters is met.

If you want to diverge from this example configuration, refer to the AWS documentation and ensure that your VPC is correctly set up for an EKS cluster.

You can create a VPC by running the following CLI commands:

```
aws cloudformation create-stack \
  --stack-name xrd-eks-vpc-stack \
  --template-url
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml
 \
  --parameters \
    ParameterKey=VpcBlock,ParameterValue=10.0.0.0/16 \
    ParameterKey=PublicSubnet01Block,ParameterValue=10.0.50.0/24 \
    ParameterKey=PublicSubnet02Block,ParameterValue=10.0.51.0/24 \
    ParameterKey=PrivateSubnet01Block,ParameterValue=10.0.0.0/24 \
    ParameterKey=PrivateSubnet02Block,ParameterValue=10.0.1.0/24
```

This command returns quickly, but the process continues in the background. If you want to wait until the completion of VPC creation, use the following command:

```
aws cloudformation wait stack-create-complete \
  --stack-name xrd-eks-vpc-stack
```

VPC is created with a VPC block (`10.0.0.0/16`), two public subnets (`10.0.50.0/24, 10.0.51.0/24`), and two private subnets (`10.0.0.0/24, 10.0.1.0/24`).

Once the VPC creation is completed, you can run the following command to get the output details.

```
aws cloudformation describe-stacks \
  --stack-name xrd-eks-vpc-stack \
  --query "Stacks[0].Outputs"
```

Make a note of the VPC ID `<vpc-id>` and private subnet IDs `<private-subnet-1>, <private-subnet-2>`, as you may require them when you create cluster and nodes.

Create a Security Group

You must create a security group in the VPC for EKS Control Plane traffic.

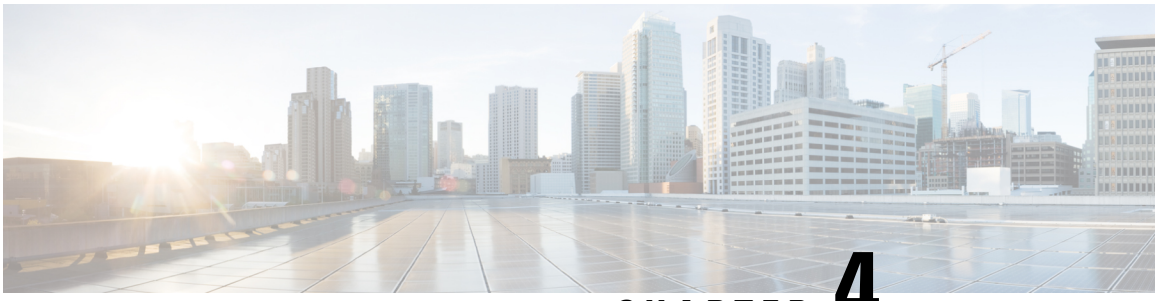
To create a security group that allows communication between the worker nodes and the cluster, use the following command:

```
aws ec2 create-security-group --group-name "xrd-eks-cp-comms" \
  --description "Control plane communication for the XRd EKS Cluster" \
  --vpc-id <vpc-id>
```

Make a note of the output group ID `<sg_id>`.

When a security group is created, Amazon EC2 creates a default egress rule that allows egress traffic. Since ingress rules aren't added by default, all incoming traffic is dropped. So, you must add an ingress rule to allow all traffic within the security group. For example,

```
aws ec2 authorize-security-group-ingress \
  --group-id <sg-id> \
  --source-group <sg-id> \
  --protocol all
```



CHAPTER 4

Bring Up and Configure an EKS Cluster

This module describes the steps to bring up and configure an EKS Cluster.

Bring up an EKS Cluster

Create a Cluster Role

You must create a cluster role that has the permission to access EKS resources.

Copy the following contents to a file named `eks-cluster-role-trust-policy.json`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Create an AWS Identity and Access Management (IAM) role with this policy document.

```
aws iam create-role \
  --role-name xrd-eks-cluster-role \
  --assume-role-policy-document "file://eks-cluster-role-trust-policy.json"
```

Make a note of the role Amazon Resource Names (ARN), `<cluster-role-arn>`.

Attach the EKS managed IAM policy to the role.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy \
  --role-name xrd-eks-cluster-role
```

Create a Worker Node Role

You must create a role for the EKS worker nodes to connect to the EKS cluster.

Copy the following contents to the file named `eks-node-role-trust-policy.json`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Note

- 2012-10-17 is the latest trust policy version.
- The **Statement** allows any EC2 resource to assume the worker node role and therefore obtain the associated permissions, which enable a node to join to an EKS cluster.

Create the IAM role using the following command:

```
aws iam create-role \
  --role-name xrd-eks-node-role \
  --assume-role-policy-document "file://eks-node-role-trust-policy.json"
```

Make a note of the node role ARN, `<node-role-arn>`.

Attach the required IAM policies to the role. For example,

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy \
  --role-name xrd-eks-node-role
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly \
  --role-name xrd-eks-node-role
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
  --role-name xrd-eks-node-role
```

Create Worker Node Profile

An instance profile is required to apply the Worker node role to EC2 instances.

To create an instance profile, use the following command:

```
aws iam create-instance-profile --instance-profile-name xrd-eks-node-profile
```

Make a note of the profile ARN, `<node-profile-arn>`.

Add the role to the new profile using the following command:

```
aws iam add-role-to-instance-profile \
  --instance-profile-name xrd-eks-node-profile \
  --role-name xrd-eks-node-role
```

Create EKS Cluster

Use the following command to create an EKS cluster:

```
aws eks create-cluster \
  --name xrd-cluster \
  --role-arn <cluster-role-arn> \
  --resources-vpc-config
"subnetIds=<private-subnet-1>,<private-subnet-2>,securityGroupIds=<sg-id>,endpointPublicAccess=true,endpointPrivateAccess=true"
 \
  --kubernetes-version <k8s-version>
```

This command execution completes quickly, but the Control plane takes around 20-30 minutes to come up completely in AWS. If you want the AWS CLI tool to monitor and wait for the cluster to become active, use the following command:

```
aws eks wait cluster-active --name xrd-cluster
```

To check the status of the cluster manually, run the following command:

```
aws eks describe-cluster --name xrd-cluster
```

In the output, the "status" will be `CREATING` until the cluster comes up completely, and then the status changes to `ACTIVE`.

Note This sample configuration sets up an EKS cluster with both public and private endpoints. The EKS cluster control plane can now receive traffic from the internet.

You can restrict the IP range allowed to access the public endpoint using the `publicAccessCirds` configuration item, or you can remove the public endpoint entirely by setting the `endpointPublicAccess` item to `false`.

For more details, see Amazon EKS cluster Endpoint Access Control.

Configure EKS Cluster

You must configure the EKS cluster after it becomes `ACTIVE`. The configuration steps are as follows:

1. First, generate `kubectl` configuration for the cluster by running the following command:

   ```
   aws eks update-kubeconfig --name xrd-cluster
   ```

2. To verify the configuration, run the following command:

   ```
   kubectl get svc
   ```

   The output must be similar to the following:

   ```
   NAME             TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
   svc/kubernetes   ClusterIP   172.20.0.1   <none>        443/TCP   1m
   ```

EKS Cluster configuration includes:

- Role authentication configuration—Allows worker nodes to join the cluster.
- AWS cluster networking configuration—Prevents AWS from using more than one Elastic Network Adapter (ENA) for Pod IPs. This configuration ensures that maximum number of Elastic Network Interfaces (ENIs) are available for use in XRd.

Role Authentication Configuration

1. Copy the following contents to a file named `aws-auth-cm.yaml`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - rolearn: <node-role-arn>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
```

2. Replace `<node-role-arn>` with the node role ARN created earlier.
3. Apply this configuration to the cluster using the following command:

```
kubectl apply -f aws-auth-cm.yaml
```

4. Then, apply the networking configuration to the cluster using the following command:

```
kubectl set env ds aws-node -n kube-system MAX_ENI=1
```

EBS CSI Driver

EKS clusters at version 1.23 or later require installation of the Amazon Elastic Block Store (Amazon EBS) Container Storage Interface (CSI) driver to manage Amazon EBS volumes for persistent volumes in Cisco IOS XRd. Configuration persistence over container restart necessitates this installation.

Install the EBS CSI driver as an EKS addon, following the recommended approach outlined in the AWS documentation.

Note The addon state will be 'Degraded' until worker nodes are brought up in the cluster, as a minimum number of EBS CSI containers are expected to be running in the cluster.

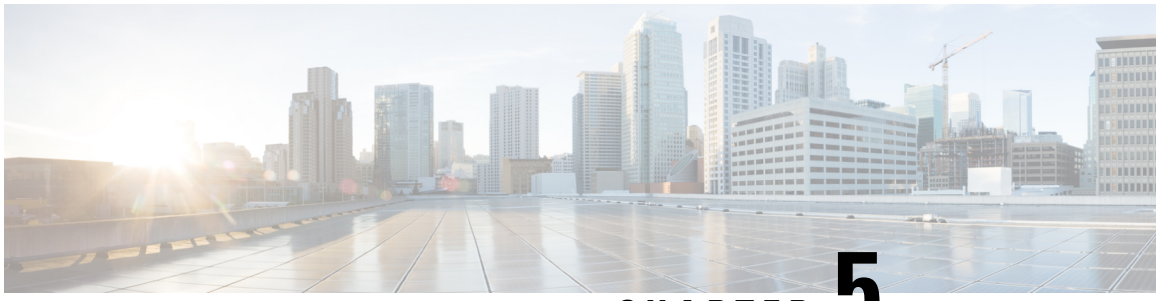
AWS Cluster Networking Configuration

Note This configuration is required only during XRd Control Plane deployment.

When deploying XRd Control Plane in the cluster, you need the Multus Container Network Interface (CNI) plugin to configure container networking.

To install Multus, run the following command:

```
kubectl apply -f
https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/master/config/multus/v4.0.2-eksbuild.1/multus-daemonset-thick.yml
```



CHAPTER 5

Set Up Container Registry

This module describes how to set up a container registry.

Set Up a Container Registry

The steps to set up a container registry for both XRd vRouter and XRd Control Plane are the same, except the image download path, and the naming of respective repositories. You can name the control-plane repository as "xrd-control-plane" to avoid confusion.

- To download the XRd vRouter tarball, go to Software Download.
- To download XRd Control Plane tarball, go to Software Download.

The container image is saved inside this tarball, and you must extract and verify it before use.

To set up a container registry, perform the following steps:

1. Go to the directory where the tarball has been downloaded, extract the tarball using the following command:

   ```
   tar zxvf xrd-vrouter-container-x86.7.8.1.tgz
   ```

2. To ensure that the image isn't corrupted, run the following python script and verify the signature.

   ```
   cd xrd-vrouter-container-x86.7.8.1
   python3 cisco_x509_verify_release.py3 \
     -e IOS-XR-SW-XRd.crt \
     -i xrd-vrouter-container-x64.dockerv1.tgz \
     -s xrd-vrouter-container-x64.dockerv1.tgz.signature \
     -v smime --container xr --sig_type DER
   ```

   The following is a sample output:

   ```
   Retrieving CA certificate from http://www.cisco.com/security/pki/certs/crrca.cer ...
   Successfully retrieved and verified crrca.cer.
   Retrieving SubCA certificate from http://www.cisco.com/security/pki/certs/xrcrrsca.cer
   ...
   Successfully retrieved and verified xrcrrsca.cer.
   Successfully verified root, subca and end-entity certificate chain.
   Successfully verified the signature of xrd-vrouter-container-x64.dockerv1.tgz using
   IOS-XR-SW-XRd.crt
   ```

3. Create an ECR repository to host the image.

   ```
   aws ecr create-repository --repository-name xrd-vrouter
   ```

4. Log into the repository using a container image tool. This example uses `skopeo`, but you can also use `docker` or `podman`.

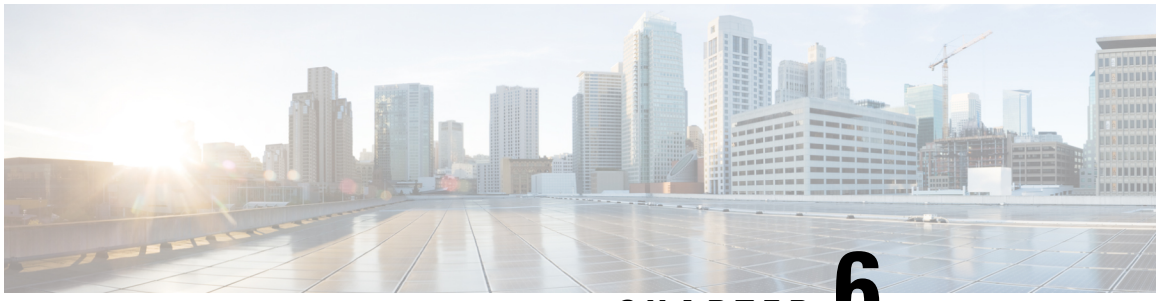
```
aws ecr get-login-password --region <region> | skopeo login --username AWS
--password-stdin <repository-uri>
```

5. Copy the image using `skopeo`, or load, tag, and push the image using `docker` or `podman`. For example,

```
 skopeo copy \
  "docker-archive:xrd-vrouter-container-x64.dockerv1.tgz" \
  "docker://<repository-uri>:7.8.1"
```

Or,

```
docker load -i xrd-vrouter-container-x64.dockerv1.tgz

docker tag <image-tag> <repository-uri>:7.8.1

docker push <repository-uri>:7.8.1
```



CHAPTER 6

Create an AMI

This module provides instructions to create an Amazon Machine Image (AMI).

Create an AMI

The EC2 hosts on which the XRd runs must be configured to meet XRd's requirements.

To ensure high XRd Router performance and maintain consistency across the cluster, you must generate an Amazon Machine Image (AMI) that meets all XRd's requirements. This AMI can then be employed as a template for launching all worker nodes within the cluster.

An AMI can be shared among all the worker nodes in a single region, so you can save time by following these instructions only once. You can then copy the AMIs to different regions.

Find the Base AMI

The base AMI that is used to create an AMI for XRd worker nodes must be Amazon EKS optimized Amazon Linux (based on Amazon Linux 2). Amazon provides separate images for every supported version of Kubernetes.

You can find the latest AMI for each Kubernetes version in each region by running the following command.

```
aws ssm get-parameter \
 --name /aws/service/eks/optimized-ami/<k8s-version>/amazon-linux-2/recommended/image_id \
 --region <region> --query "Parameter.Value" --output text
```

Note The Kubernetes version specified here must be the same as the version that was used to create the cluster.

Make a note of the AMI ID, `<al2-ami-id>`.

For XRd Control Plane, you can use the base AMI with sysctl settings applied to the user data file when creating the worker node. For details, see the Create a Worker Node section.

Create an EC2 Instance

This EC2 instance must have access to the internet for installing packages and pulling in resources. If not, you must provide all these resources locally. Run the following command to create an EC2 instance.

```
aws ec2 run-instances --image-id <al2-ami-id> --count 1 \
  --instance-type m5.24xlarge \
  --key-name <key-pair-name> \
  --security-group-ids <security-group-ids> \
  --subnet-id <subnet-id>
```

You must specify the security group and subnet IDs so that the EC2 node has access to the internet. These are not necessarily the security group and subnet IDs created earlier. `<key-pair-name>` is the name of the key-pair which you must have created earlier.

Make a note of the instance ID returned from the command, `<ami-instance-id>`.

When the EC2 instance is created, connect to the EC2 instance over SSH using the specified SSH key and the username, `ec2-user`.

Copy XRd Resources

A set of resources on the XRd is available on the EKS public GitHub at xrd-packer.

Note HashiCorp Packer is a tool used for building virtual machine images, and the **xrd-packer** GitHub repository provides resources and Packer templates to create an AMI suitable for XRd vRouter. The manual steps described in this document use the resources from the repository as the source for some of the host OS settings, but the Packer tool is not used. The README file in the **xrd-packer** GitHub repository provides a description on how to build an AMI using Packer and the provided templates, as an alternative to the manual steps provided in this document.

Copy all of the resources under the **files** folder to the EC2 instance, mirroring the structure in the repository. For example,

- Copy `files/etc/modprobe.d/igb_uio.conf` to `/etc/modprobe.d/igb_uio.conf` on the EC2 instance
- Copy `files/etc/tuned/xrd-eks-node-variables.conf` to `/etc/tuned/xrd-eks-node-variables.conf` on the EC2 instance

If you have access to the internet, you can accomplish this task by running the following commands:

```
sudo yum install -y git
git clone https://github.com/ios-xr/xrd-packer
sudo cp -r xrd-packer/files/* /
```

Clean up the unnecessary data by running the following commands:

```
sudo yum remove -y git
rm -rf xrd-packer
```

Tuning

TuneD is a daemon for monitoring and tuning of system devices. The files copied in the previous step, contain a TuneD profile tailored for XRd hosts. This profile sets up some kernel parameters and tuning options to help XRd achieve high performance.

Note Previously, TuneD was installed from the Amazon Linux 2 repositories. However, the version provided in these repositories is now outdated, as it has reached end-of-life status. Hence, it is necessary to install TuneD from the source.

Perform the following steps to install TuneD from the source.

Install the prerequisite packages using the following commands:

```
sudo yum install -y \
    dbus \
    dbus-python \
    ethtool \
    gawk \
    polkit \
    python-configobj \
    python-decorator \
    python-gobject \
    python-linux-procfs \
    python-perf \
    python-pyudev \
    python-schedutils \
    tuna \
    util-linux \
    virt-what
```

Download the TuneD v2.20.0 source tarball using the following commands:

```
mkdir tuned
curl -L https://github.com/redhat-performance/tuned/archive/refs/tags/v2.20.0.tar.gz | tar
 -xz -C tuned --strip-components 1
```

Install TuneD using the following command:

```
sudo make -C tuned PYTHON=/usr/bin/python2 install
```

Take note that TuneD attempts to install a desktop file; skipping this causes the above **make** command to fail with the following output:

```
# desktop file
install -dD /usr/share/applications
desktop-file-install --dir=/usr/share/applications tuned-gui.desktop
make: desktop-file-install: Command not found
make: *** [install] Error 127
```

This is expected and can be ignored.

You can safely remove the TuneD sources afterward using the following command:

```
rm -r tuned
```

After installing TuneD, specific files copied from the XRd resources necessitate adjustments for individual deployments. Update the file: /etc/tuned/xrd-eks-node-variables.conf with the appropriate isolated cores and required number of hugepages.

This documentation assumes deployment on an m5.24xlarge or m5n.24xlarge instance, and in this scenario, the file should appear as follows:

```
isolated_cores=16-23
hugepages_gb=12
```

Note When running on an m5.24xlarge instance, the number of hugepages is doubled from the Cisco IOS XRd requirement. This is because the configured hugepages are distributed across all non-uniform memory access (NUMA) nodes in the system, whereas Cisco IOS XRd necessitates all its hugepages to be on the single NUMA node it operates on.

Lastly, initiate the TuneD service, and select the Cisco IOS XRd profile by executing the following command:

```
sudo systemctl start tuned
sleep 10
sudo tuned-adm profile xrd-eks-node
```

Note Running the `profile` command immediately after starting the service will prevent it from taking effect. Allow for a reasonable delay before running the `profile` command.

Kernel Parameters

Setting kernel parameters (sysctl settings) on the host is either required or recommended when running Cisco IOS XRd containers. These parameters address the needs of both the XRd Control Plane and XRd vRouter platforms.

Set the kernel parameters by running the following commands:

```
sudo -i
echo "fs.inotify.max_user_instances=64000" >> /etc/sysctl.conf
echo "fs.inotify.max_user_watches=64000" >> /etc/sysctl.conf
echo "kernel.randomize_va_space=2" >> /etc/sysctl.conf
echo "net.core.rmem_max=67108864" >> /etc/sysctl.conf
echo "net.core.wmem_max=67108864" >> /etc/sysctl.conf
echo "net.core.rmem_default=67108864" >> /etc/sysctl.conf
echo "net.core.wmem_default=67108864" >> /etc/sysctl.conf
echo "net.core.netdev_max_backlog=300000" >> /etc/sysctl.conf
echo "net.core.optmem_max=67108864" >> /etc/sysctl.conf
echo "net.ipv4.udp_mem=1124736 10000000 67108864" >> /etc/sysctl.conf
sysctl -p --system
exit
```

Note `fs.inotify.max_user_watches=64000` - Indicates the maximum number of files you are allowed to watch for changes. You will get notified when there is a file change.

Configuring Core Dumping

Core files must be collected for detailed debugging. To prevent the node's disk from being exhausted, you must configure the system carefully. The recommended solution is to use the `systemd-coredump` tool.

Configure the kernel to use the `systemd-coredump` tool when a process crashes.

```
sudo -i
echo "kernel.core_pattern=|/lib/systemd/systemd-coredump %P %u %g %s %t 9223372036854775808
 %h" >> /etc/sysctl.conf
sysctl -p --system
exit
```

Copy the following contents to `/etc/systemd/coredump.conf`.

```
[Coredump]
Storage=external # (this is the default)
Compress=yes     # (this is the default)
MaxUse=32G       # (twice the sum of RAM given to XRd)
KeepFree=16G     # (the sum of RAM given to XRd)
ProcessSizeMax=32G  # (required if < v251 of systemd)
ExternalSizeMax=32G # (required if < v251 of systemd)
```

These values are applicable for a single XRd vRouter installation, described in Install XRd vRouter.

For the XRd Control Plane installation, described in Install XRd Control Plane, the copied values must be `MaxUse=12G` and `KeepFree=6G`.

In this configuration, `systemd-coredump` is configured to write the core files to `/var/lib/systemd/coredump` on the host.

Building and Installing the Interface Driver

You must not use the interface driver bundled with Amazon Linux 2, as it does not support an ENAv2 interface feature.

Perform the following steps to build and install the interface driver that has Write Combining support.

1. Install the required packages

   ```
   sudo yum install -y kernel-devel-$(uname -r)
   ```

2. Pull the DPDK source code, extract it, and build `igb_uio`.

   ```
   curl -O
   https://git.dpdk.org/dpdk-kmods/snapshot/dpdk-kmods-e721c733cd24206399bebb8f0751b0387c4c1595.tar.gz
   tar zxvf dpdk-kmods-e721c733cd24206399bebb8f0751b0387c4c1595.tar.gz
   cd dpdk-kmods-e721c733cd24206399bebb8f0751b0387c4c1595
   make -C linux/igb_uio
   ```

3. When the the driver is built, copy the interface driver kernel module into the appropriate location and register it.

   ```
   sudo cp linux/igb_uio/igb_uio.ko /lib/modules/"$(uname -r)"/kernel/drivers/uio
   sudo depmod -a
   cd ..
   ```

4. Clean up the packages.

   ```
   rm -f dpdk-kmods-e721c733cd24206399bebb8f0751b0387c4c1595.tar.gz
   rm -rf dpdk-kmods-e721c733cd24206399bebb8f0751b0387c4c1595
   sudo yum remove -y kernel-devel-$(uname -r)
   sudo yum autoremove -y
   ```

5. Load the kernel module with the following:

   ```
   sudo modprobe uio
   sudo modprobe igb_uio wc_activate=1
   ```

6. Set the kernel module to load on boot, which is handled by copying the files earlier.

Reboot and Check

Reboot the EC2 instance to allow all the settings to take effect.

When the EC2 instance is rebooted, copy the `host-check` script from the **xrd-tools** repository at xrd-tools.

The `host-check` tool verifies if the host environment is configured correctly for running XRd containers. For more information on the usage of `host-check` tool, see the README file.

Run the `host-check` script.

The following is a sample output from the XRd vRouter. You can run the script similarly on the XRd Control Plane.

```
[ec2-user@ip-172-31-3-90 ~]$ ./host-check -p xrd-vrouter
==============================
Platform checks - xrd-vrouter
==============================
PASS -- CPU architecture (x86_64)
PASS -- CPU cores (96)
PASS -- Kernel version (5.4)
PASS -- Base kernel modules
        Installed module(s): dummy, nf_tables
PASS -- Cgroups (v1)
PASS -- Inotify max user instances
        64000 - this is expected to be sufficient for 16 XRd instance(s).
PASS -- Inotify max user watches
        64000 - this is expected to be sufficient for 16 XRd instance(s).
PASS -- Socket kernel parameters (valid settings)
PASS -- UDP kernel parameters (valid settings)
INFO -- Core pattern (core files managed by the host)
PASS -- ASLR (full randomization)
INFO -- Linux Security Modules (No LSMs are enabled)
PASS -- CPU extensions (sse4_1, sse4_2, ssse3)
PASS -- RAM
        Available RAM is 365.2 GiB.
        This is estimated to be sufficient for 73 XRd instance(s), although memory
        usage depends on the running configuration.
        Note that any swap that may be available is not included.
PASS -- Hugepages (12 x 1GiB)
PASS -- Interface kernel driver
        Loaded PCI drivers: vfio-pci, igb_uio
INFO -- IOMMU
        vfio-pci is set up in no-IOMMU mode, but IOMMU is recommended for security.
PASS -- Shared memory pages max size (17179869184.0 GiB)

================================================================
Host environment set up correctly for xrd-vrouter
================================================================
```

Delete the Cloud Folder

For the AMI to run the User Data passed into an EC2 instance, you must delete the `/var/lib/cloud` folder.

```
sudo rm -rf /var/lib/cloud
```

Take a Snapshot

Stop the EC2 instance, by running the following command:

```
aws ec2 stop-instances --instance-ids <ami-instance-id>
```

Note This command returns immediately, but it may take a minute for the container to be fully stopped.

To check if the EC2 instance is fully stopped, run the following command and ensure that the instance state is "stopped":

```
ec2 describe-instances --instance-ids <ami-instance-id> --query
'Reservations[0].Instances[0].State.Name
```

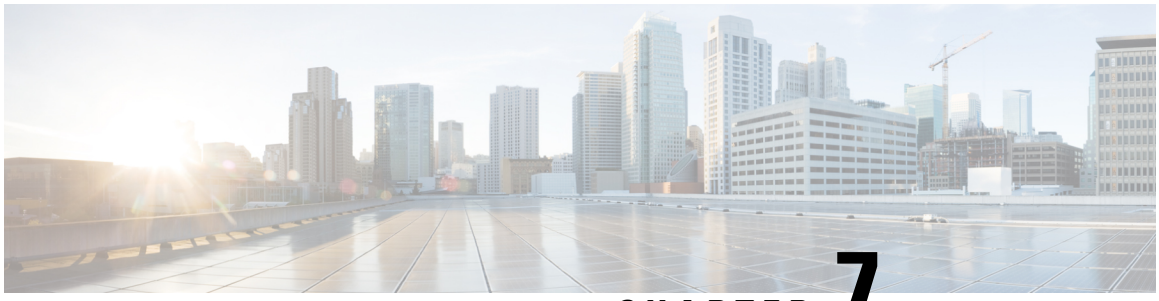
Once the EC2 instance has stopped, take a snapshot using the following command:

```
aws ec2 create-image --instance-id <ami-instance-id> --name xrd-vrouter-ami
```

Make a note of the AMI ID, `<xrd-ami-id>`.

Terminate the EC2 instance.

```
aws ec2 terminate-instances --instance-id <ami-instance-id>
```



CHAPTER 7

Create a Worker Node

This module describes the tasks to create a worker node.

Create a Worker Node

This section describes how to create a self-managed worker node in the EKS cluster that satisfies all of XRd's requirements on the host operating system.

Before creating a worker node, ensure that the EKS cluster is in `ACTIVE` state, and the authentication and networking configuration has been applied as described in the Create EKS Cluster section.

This example is inline with the XRd vRouter, and uses `m5n.24xlarge` instance with three interfaces:

- One interface reserved for cluster communication.
- Two XRd data interfaces.

Prerequisites

- Find the number of cores on the instance type

  To find the number of cores, run the following command:

  ```
  aws ec2 describe-instance-types \
    --instance-type m5.24xlarge \
    --query "InstanceTypes[0].VCpuInfo.DefaultCores" \
    --output text
  ```

  This value must be substituted for `<cpu-cores>` in the **EC2 run-instances** command.

- Create a user data file

  Create the user data file by copying the following contents into a file named `worker-user-data.bash`:

  ```
  #!/bin/bash
  /etc/eks/bootstrap.sh xrd-cluster
  ```

For XRd Control Plane, add the following two sysctl settings to the user data file:

```
echo "fs.inotify.max_user_instances=64000" >> /etc/sysctl.conf
echo "fs.inotify.max_user_watches=64000" >> /etc/sysctl.conf
```

Bring Up the Worker Node

Bring up the worker node by running the following command:

```
aws ec2 run-instances \
  --image-id <xrd-ami-id> \
  --count 1 \
  --instance-type m5.24xlarge \
  --key-name <key-pair-name> \
  --block-device-mappings "DeviceName=/dev/xvda,Ebs={VolumeSize=56}" \
  --iam-instance-profile "Arn=<node-profile-arn>" \
  --network-interfaces "DeleteOnTermination=true,DeviceIndex=0,Groups=<sg-id>,
SubnetId=<private-subnet-1>,PrivateIpAddress=10.0.0.10" \
  --cpu-options CoreCount=<cpu-cores>,ThreadsPerCore=1 \
  --tag-specifications
"ResourceType=instance,Tags=[{Key=kubernetes.io/cluster/xrd-cluster,Value=owned}]" \
  --user-data file://worker-user-data.bash
```

Make a note of the instance id, `<worker-instance-id>`.

This command brings up an EC2 instance with the following settings:

- A 56-GB primary partition - required to store any process cores that the XRd generates.
- A single interface in the first private subnet with permissions to communicate with the EKS control plane. This interface is used for cluster control plane communications. The assigned IP address is 10.0.0.10.
- One thread per core (SMT or Hyper-Threading turned off). This is to prevent the "noisy neighbor effect" (where processes scheduled on a different logical but same physical core hampers the performance of high priority processes) for the high-performance packet processing threads.
- A tag that is required by EKS to display the node should be allowed to join the cluster.
- A user data file that runs the EKS bootstrap script with the cluster name.

The requirements for XRd Control Plane are as follows:

- You can use a smaller (and cheaper) instance type, for example, `m5.2xlarge`.
- The `--cpu-options` line is not required.

Turn off the source or destination check for the instance, by running the following command:

```
aws ec2 modify-instance-attribute \
  --instance-id <worker-instance-id> \
  --no-source-dest-check
```

When the worker node is up, check if the worker node is added to the cluster.

```
# kubectl get nodes
NAME                         STATUS   ROLES    AGE    VERSION
ip-10-0-0-10.ec2.internal    Ready    <none>   1m     v1.22.17-eks-48e63af
```

Note If you do not see the worker node, check the EKS configuration steps.

Add Interfaces for XRd

The worker node created in the previous step has a single interface used for cluster control plane traffic. So, you must create more XRd interfaces and attach them to the worker node.

Also, you must create additional subnets and security groups. You must not use the subnets and security groups created while creating a VPC here, because the cluster control plane traffic must be kept separate from the data traffic flowing through the XRd instances.

Add Subnets for Data Traffic

The setup for data subnets and security groups vary with each deployment. In this example two subnets are created, and isolated from the internet. One security group is created, which allows all communication inside the security group, but rejects any traffic from outside the security group.

These subnets and the security group can be used for interfaces on multiple worker nodes.

Note The data traffic subnets must be created in the same Availability Zone (AZ) as the worker node.

Create two subnets for data traffic using the following command:

```
aws ec2 create-subnet \
  --vpc-id "<vpc-id>" \
  --cidr-block "10.0.100.0/24" \
  --availability-zone "<region>a"
aws ec2 create-subnet \
  --vpc-id <vpc-id> \
  --cidr-block "10.0.101.0/24" \
  --availability-zone "<region>a"
```

Note The `a` after the `<region>` in the availability zone specification indicates the first subnet in the region.

Make a note of the subnet IDs, `<data-subnet-1>` and `<data-subnet-2>`.

Create the Security Group

Create a security group for the data network traffic using the following command:

```
aws ec2 create-security-group --group-name "xrd-data" \
  --description "Data traffic for XRd EKS Cluster" \
  --vpc-id <vpc-id>
```

Make a note of the output group ID, `<data-sg-id>`.

A single egress rule that allows all traffic egress is added when a security group is created, but no ingress rules are added. So, all incoming traffic is dropped. You must add an ingress rule for all traffic originating from the same security group, so that all traffic is allowed within the security group.

```
aws ec2 authorize-security-group-ingress \
  --group-id <data-sg-id> \
  --source-group <data-sg-id> \
  --protocol all
```

Create and Attach Interfaces

The following example creates two interfaces on the worker node to be used by XRd, one in each of the two subnets created. Run the following commands to create these interfaces:

```
aws ec2 create-network-interface \
  --description "XRd worker 1 data 1" \
  --groups <data-sg-id> \
  --private-ip-address "10.0.100.10" \
  --subnet-id <data-subnet-1> \
  --tag-specifications
"ResourceType=network-interface,Tags=[{Key=node.k8s.amazonaws.com/no_manage,Value=true}]"
aws ec2 create-network-interface \
  --description "XRd worker 1 data 2" \
  --groups <data-sg-id> \
  --private-ip-address "10.0.101.10" \
  --subnet-id <data-subnet-2> \
  --tag-specifications
"ResourceType=network-interface,Tags=[{Key=node.k8s.amazonaws.com/no_manage,Value=true}]"
```

Make a note of both the network interface IDs, `<data-interface-1>` and `<data-interface-2>`.

To attach the interfaces to the worker node, run the following commands:

```
aws ec2 attach-network-interface \
  --device-index 1 \
  --instance-id <worker-instance-id> \
  --network-interface-id <data-interface-1>
aws ec2 attach-network-interface \
  --device-index 2 \
  --instance-id <worker-instance-id> \
  --network-interface-id <data-interface-2>
```

Running Cisco IOS XRd on the Worker Node

This section provides instruction on how to run a Cisco IOS XRd workload on the previously configured worker node.

Perform the following steps to run the Cisco IOS XRd on the Worker Node:

- Provide a label to the worker node so that it can be identified in Kubernetes metadata using the following command:

  `kubectl label node/ip-10-0-0-10.ec2.internal xrd-worker=one`

- Add the XRd Helm repository using the following command:

  `helm repo add xrd https://ios-xr.github.io/xrd-helm`

- Proceed with either of the XRd installation sections provided below:
  - Install XRd vRouter
  - Install XRd Control Plane

Install XRd vRouter

Copy the following contents to a file named `xrd-one.yaml`.

```
image:
  repository: "<repository-uri>"
  tag: "7.8.1"
resources:
  limits:
    memory: 10Gi
    hugepages-1Gi: 6Gi
nodeSelector:
  xrd-worker: "one"
persistence:
  enabled: true
  storageClass: "gp2"
config:
  username: <xr-root-username>
  password: <xr-root-password>
  ascii: |
    interface HundredGigE0/0/0/0
     ipv4 address 10.0.100.10 255.255.255.0
    !
    interface HundredGigE0/0/0/1
     ipv4 address 10.0.101.10 255.255.255.0
    !
interfaces:
- type: pci
  config:
    last: 2
cpu:
  cpuset: "12-23"
pciDriver: "igb_uio"
```

Replace all the values in angle brackets with:

- The ECR repository URI from the top
- An XR root username and password

Run the following command to install XRd into the cluster:

```
helm install xrd-one xrd/xrd-vrouter -f xrd-one.yaml
```

Install XRd Control Plane

Copy the following contents to a file named `xrd-one.yaml`.

```
image:
  repository: "<repository-uri>"
  tag: "7.8.1"
resources:
  limits:
    memory: 6Gi
nodeSelector:
  xrd-worker: "one"
persistence:
  enabled: true
  storageClass: "gp2"
config:
  username: <xr-root-username>
  password: <xr-root-password>
  ascii: |
    interface GigabitEthernet0/0/0/0
     ipv4 address 10.0.100.10 255.255.255.0
    !
```

```
    interface GigabitEthernet0/0/0/1
     ipv4 address 10.0.101.10 255.255.255.0
    !
interfaces:
- type: multus
  config:
    type: host-device
    device: eth1
- type: multus
  config:
    type: host-device
    device: eth2
```

Replace all the values in angle brackets with:

- The ECR repository URI from the top
- An XR root username and password

Run the following command to install XRd Control plane into the cluster.

```
helm install xrd-one xrd/xrd-control-plane -f xrd-one.yaml
```

Access XRd

After the XRd is installed, it takes around a minute to come up as the image is pulled from the ECR repository. You can monitor the status of the XRd pod by running the following command:

```
kubectl get pods
```

When the pod is in `Running` state, attach to the XRd pod using the following command:

```
kubectl attach -it pod/<pod-name>
```

Log in using the root username and password that were added into `xrd-one.yaml`.

The following is a sample output:

```
# kubectl attach -it pod/<pod-name>
If you don't see a command prompt, try pressing enter.

User Access Verification

Username: myuser
Password:


RP/0/RP0/CPU0:ios#
```

To check the status of the XR data interfaces, run the **show ip int br** command.

```
RP/0/RP0/CPU0:ios#show ip int br
Wed Mar  8 12:27:35.949 UTC

Interface                      IP-Address      Status          Protocol Vrf-Name
HundredGigE0/0/0/0             10.0.100.10     Up              Up       default
HundredGigE0/0/0/1             10.0.101.10     Up              Up       default
```

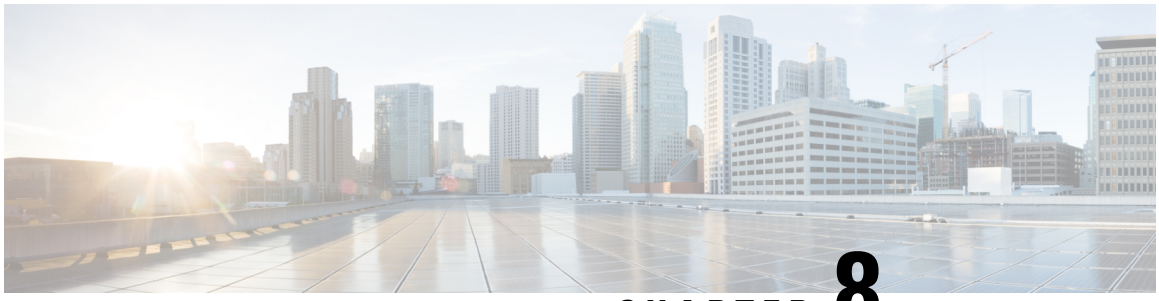
Note On XRd Control Plane, the interface name is `GigabitEthernet*`.

You can detach the console using `^P^Q`.

Note In this example, the only method to attach to the XRd console is using `kubectl attach` (or `kubectl exec`).

In this example setup, the XRd instance has interfaces only in the data subnets. Interfaces in the data subnets can communicate only with other interfaces in the data subnets (due to the security group). SSH access is not possible because there are no other hosts with interfaces in the data subnet.

If more hosts are added with interfaces in the data subnet (appropriate security group settings applied), and when standard XR SSH server configuration is applied, SSH access from that host is possible.



CHAPTER 8

Helms Chart Repository

Helm Charts Repository

Helm is a tool that you can use to manage packages in the Kubernetes environment. You can use Helm to modify the current deployment. The helm repository is a repository in which you can find the sample value files for XRd and review all available settings that can be configured during the deployment of XRd on K8s.

To deploy XRd on your Amazon Elastic Container Service for Kubernetes (EKS) cluster, use the appropriate XRd Helm Chart. The below-mentioned application charts are available in the XRd Helm Repository:

- xrd-control-plane, for running XRd Control Plane containers.
- xrd-vrouter, for running XRd vRouter containers.

Follow the instructions provided in the Helm documentation to install and setup Helm.

The following commands serve as options for interacting with XRd's Helm repository:

- To add the Helm repository, enter the `helm repo add xrd https://ios-xr.github.io/xrd-helm` command.
- To retrieve the latest versions of packages from a previously added Helm repository, enter the `helm repo update xrd` command.
- To view the charts available in a specific Helm repository, enter the `helm search repo xrd` command.
- To install the chart, enter the `helm install my-<chart-name> xrd/<chart-name> command.`
- To uninstall the chart, enter the `helm delete my-<chart-name> xrd/<chart-name> command.`
- To download and view files for a published chart without installing them, enter the `helm pull chartrepo/chartname` command.

The process of creating a Kubernetes StatefulSet object through XRd charts follows the generation of a singular replica.

Sample Helm Values for XRd vRouter

The following is a sample Helm values file for XRd vRouter deployment. This example uses default resource requirements from the data sheet.

Perform the following updates before use:

- Set `image.repository` to a container repository URL containing XRd vRouter images.
- Update `image.tag` depending on the tagging scheme for your container repository.
- Update `config.username` and `config.password` with a username and password of your choice to use for the XR root user.
- Update `interfaces` and `mgmtInterfaces` depending on your network topology.

EXAMPLE

```
# Sample values file for running an XRd workload on an m5n.24xlarge
# or m5.24xlarge EC2 instance.
#
# This values file is for the xrd-vrouter chart at version 1.0.0.

# Image configuration
image:
  # Repository for the container image (required).
  repository: "my-repo/xrd-vrouter"
  # Image tag (required).
  tag: "7.8.1"

# Pod resource configuration.
resources:
  limits:
    # All this memory must be available on NUMA node 0 (i.e. CPU socket 0).
    memory: 10Gi
    hugepages-1Gi: 6Gi

# Security context for the XRd container.
# Privileged mode is currently required for XRd in K8s due to device
# access requirements.
securityContext:
  privileged: true

# Persistent storage controls.
persistence:
  enabled: true
  size: "6Gi"
  accessModes:
  - ReadWriteOnce
  storageClass: "gp2"

# XR configuration and boots scripts.
config:
  # XR root username and password.
  username: "myusername"
  password: "mypassword"

# XRd line interfaces.
# N.B. This requires there to be 15 interfaces attached to the worker node -
# one for the K8s cluster networking and 14 for XRd.
interfaces:
- type: pci
  config:
    last: 14

# XRd management interfaces.
mgmtInterfaces: []

# CPU settings.
```

```
cpu:
  # The cpuset to use for XRd vRouter.
  cpuset: "12-23"

# PCI interface driver.
pciDriver: "igb_uio"
```

Sample Helm Values for XRd Control Plane

The following is a sample Helm values file for an XRd Control Plane deployment. This example uses default resource requirements from the data sheet.

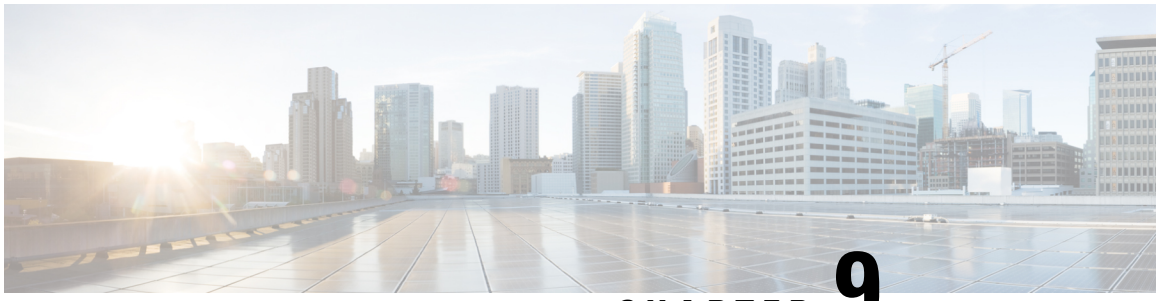
Perform the following updates before use:

- Set `image.repository` to a container repository URL containing XRd Control Plane images.
- Update `image.tag` depending on the tagging scheme for your container repository.
- Update `config.username` and `config.password` with a username and password of your choice to use for the XR root user.
- Update `interfaces` and `mgmtInterfaces` depending on your network topology.

EXAMPLE

```
# Sample values file for running an XRd SR-PCE workload on AWS EKS.
#
# This values file is for the xrd-control-plane chart at version 1.0.0.

# Image configuration
image:
  # Repository for the container image (required).
  repository: "my-repo/xrd-control-plane"
  # Image tag (required).
  tag: "7.8.1"

# Pod resource configuration.
# See https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
resources:
  limits:
    memory: 16Gi

# Security context for the XRd container.
# Privileged mode is currently required for XRd in K8s due to device
# access requirements.
securityContext:
  privileged: true

# Persistent storage controls.
persistence:
  enabled: true
  size: "6Gi"
  accessModes:
  - ReadWriteOnce
  storageClass: "gp2"

# XR configuration and boots scripts.
config:
  # XR root username and password.
  username: "myusername"
  password: "mypassword"

# XRd line interfaces.
```

```
interfaces:
- type: multus
  config:
    type: host-device
    device: eth1

# XRd management interfaces.
mgmtInterfaces:
- type: defaultCni
  chksum: true
```



CHAPTER 9

Lifecycle Management and Redundancy Strategy

This module describes the operations required for life cycle management and redundancy.

Lifecycle Management

Upon uninstallation of an XRd instance, if you had configured persistence, a persistent volume claim (PVC) remains. To fully eliminate the XRd instance, you must delete the PVC manually.

To enable support upgrade flows, bring down the worker node, uninstall the XRd deployment while retaining the PVC to allow persistence over upgrades, reinstall and reinstate with new settings.

You can use the standard Helm upgrade path if no worker node modifications are needed. The worker node can remain operational during the upgrade. Kubernetes must accept the resource changes that ensue.

Testing in the lab may be necessary to determine if an upgrade is compatible because not all circumstances are considered. Most configuration upgrades, including images, base configurations, and interfaces, are expected to work with this method.

Redundancy Strategy

The redundancy approach for XRd involves the utilization of two distinct XRd installations to support a redundant pair of instances. This ensures the establishment of two separate Kubernetes resources.

When modifications are necessary, you can make the changes to either of the instances initially. When the changes have stabilized, you can implement the same changes to the other instance.

High-Availability for XRd vRouter

Table 1: Feature History Table

Feature Name	Release Name	Description
High-Availability for XRd vRouter	Release 7.11.1	Now you can run XRd vRouter as a highly available redundant pair, actively managing session state. It incorporates VRRP monitoring and high-availability automation logic in peer containers, configured to update the AWS APIs.

Overview

To ensure continuous availability, the redundant instance pair may be configured to operate as a highly available cluster. This configuration utilizes VRRP monitoring to proactively manage session state. Additionally, high-availability automation logic is implemented within peer containers to facilitate dynamic updates to AWS APIs. This combination of proactive monitoring and dynamic automation ensures seamless service delivery even in the event of individual instance failures.

References

For Information On	Refer
XRd HA app container application and its deployment and configuration	XRd HA App repository
VRRP configured on unicast mode	The *Unicast VRRP* section in the *IP Addresses and Services Configuration Guide for Cisco ASR 9000 Series Routers, IOS XR Release 7.11.x*