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 EKS Cluster Configuration section.

This example is inline with the XRd vRouter, and uses `m6in.16xlarge` instance with three interfaces:

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

Prerequisites

- **Find the number of cores on the instance**

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

  ```
  aws ec2 describe-instance-types \
    --instance-type m6in.16xlarge \
    --query "InstanceTypes[0].VCpuInfo.DefaultCores" \
    --output text
  ```

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

- **Find cluster information**

  To find the required cluster information, use the following commands:

  - To find the API endpoint:

```
aws eks describe-cluster \
  --name xrd-cluster \
  --query "cluster.endpoint" \
  --output text
```

- To find the certificate:

```
aws eks describe-cluster \
  --name xrd-cluster \
  --query "cluster.certificateAuthority.data" \
  --output text
```

- To find the Classless Inter-Domain Routing (CIDR) IP address:

```
aws eks describe-cluster \
  --name xrd-cluster \
  --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr" \
  --output text
```

- **Create a user data file**

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

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: xrd-cluster
    apiServerEndpoint: <api-endpoint>
    certificateAuthority: <certificate>
    cidr: <cidr>
```

  Use the values obtained for `<api-endpoint>`, `<certificate>`, and `<cidr>`.

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 m6in.16xlarge \
  --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.yaml
```

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 contains the required NodeConfig for EKS.

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.

When using the base Amazon Linux 2023 for EKS AMI, ensure that the user data file is in MIME format with a bash section that runs:

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

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     v1.31.3-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: "25.2.1"
resources:
  limits:
    memory: 16Gi
    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: <cpuset>
pciDriver: "igb_uio"
```

Replace all the values in angle brackets with:

- The ECR repository URI from the top.
- An XR root username and password.
- The CPUset: Use "7-23" for m5[n].12xlarge or m5[n].24xlarge instance or "15-31" for m6in.16xlarge instance.

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: "25.2.1"
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: ens6
- type: multus
  config:
    type: host-device
    device: ens7
```

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 interface brief** command.

```
Router# show ip interface brief
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.