Experiment: Elastic Kubernetes Cluster Networking

Objectives:

Examine major components of AWS VPC routed CNI and demonstrate how it handles traffic for:

Pod to Pod communication

Examine the communication channels between Amazon EKS Kubernetes Control Plane and Customer worker nodes.

Overview

Perform intra-node Pod to Pod communication and exam the major components involved in the life of a ping packet

Perform inter-node Pod to Pod communication and exam the major components involved in the life of a ping packet

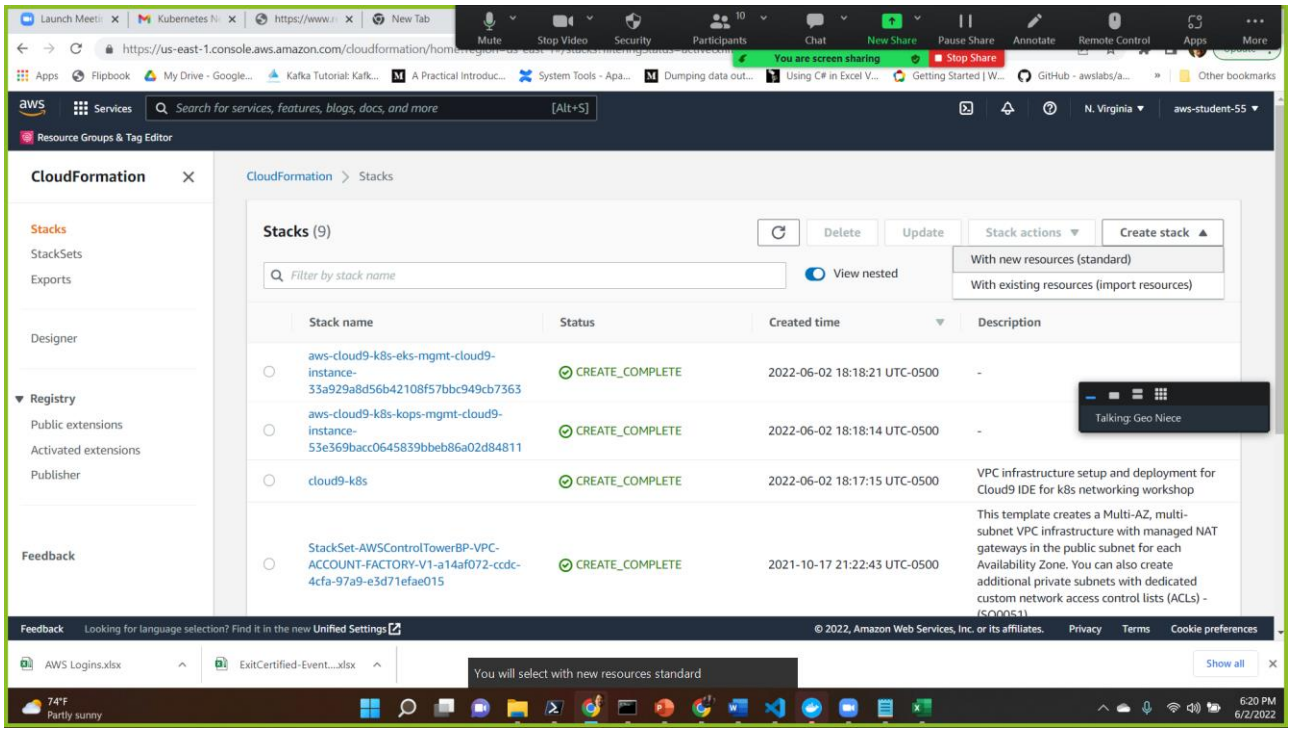
Exam the IPtable for kubernete services

In this experiment, we will deploy a Kubernetes cluster on AWS EKS. We recommend to either use us-east-1 or us-west-2 normally

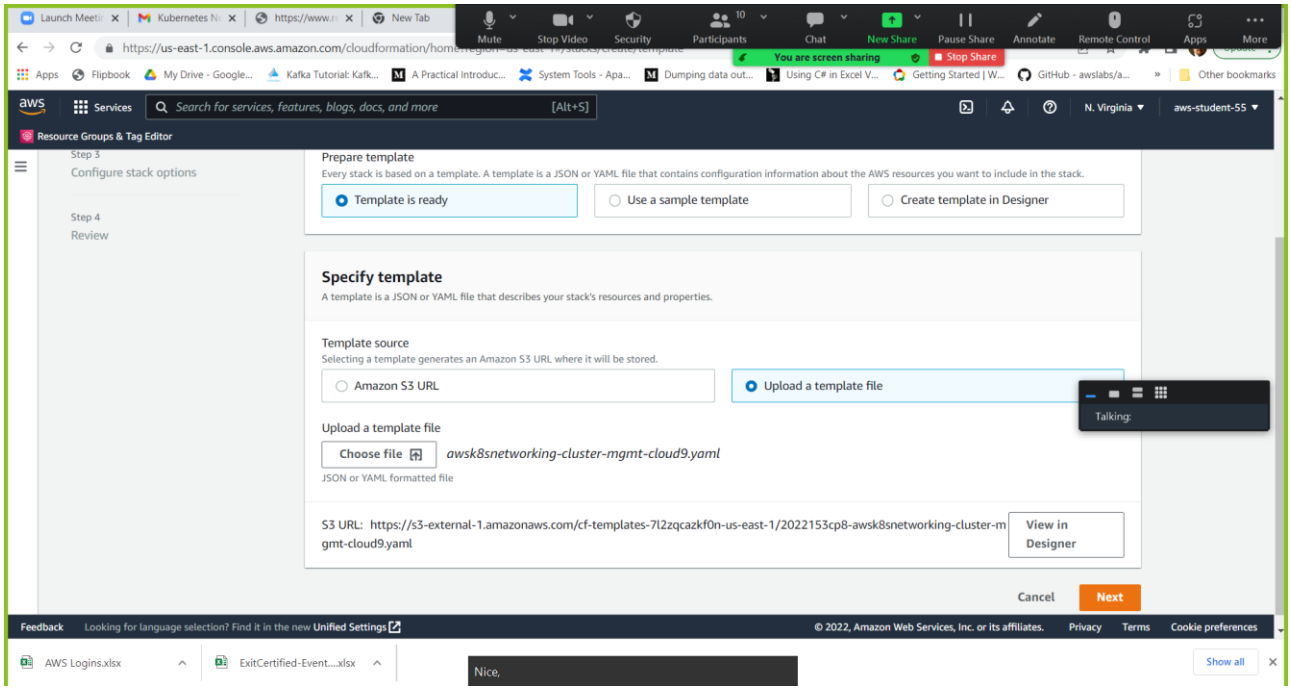
CREATE CLOUD9 WORKSPACE

Before you deploy the CloudFormation template, feel free to view it here, awsk8snetworking-cluster-mgmt-cloud9.yaml

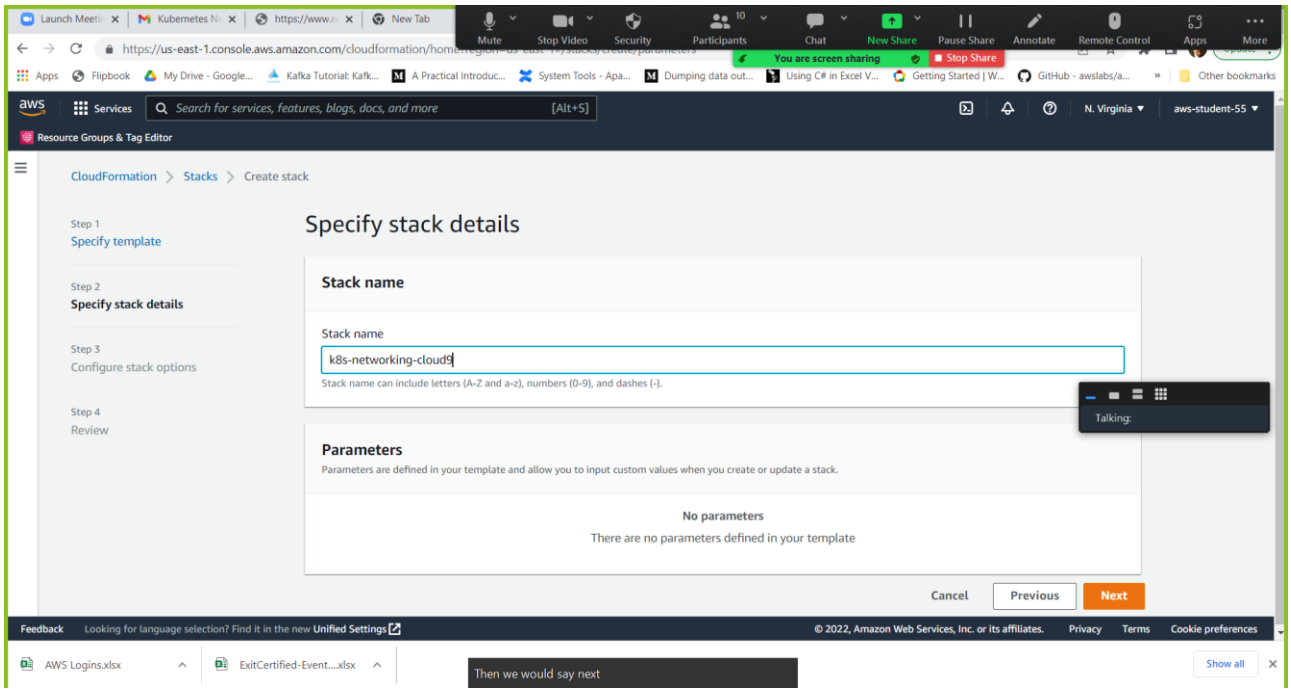
After you've logged into the console you use the CloudFormation management console



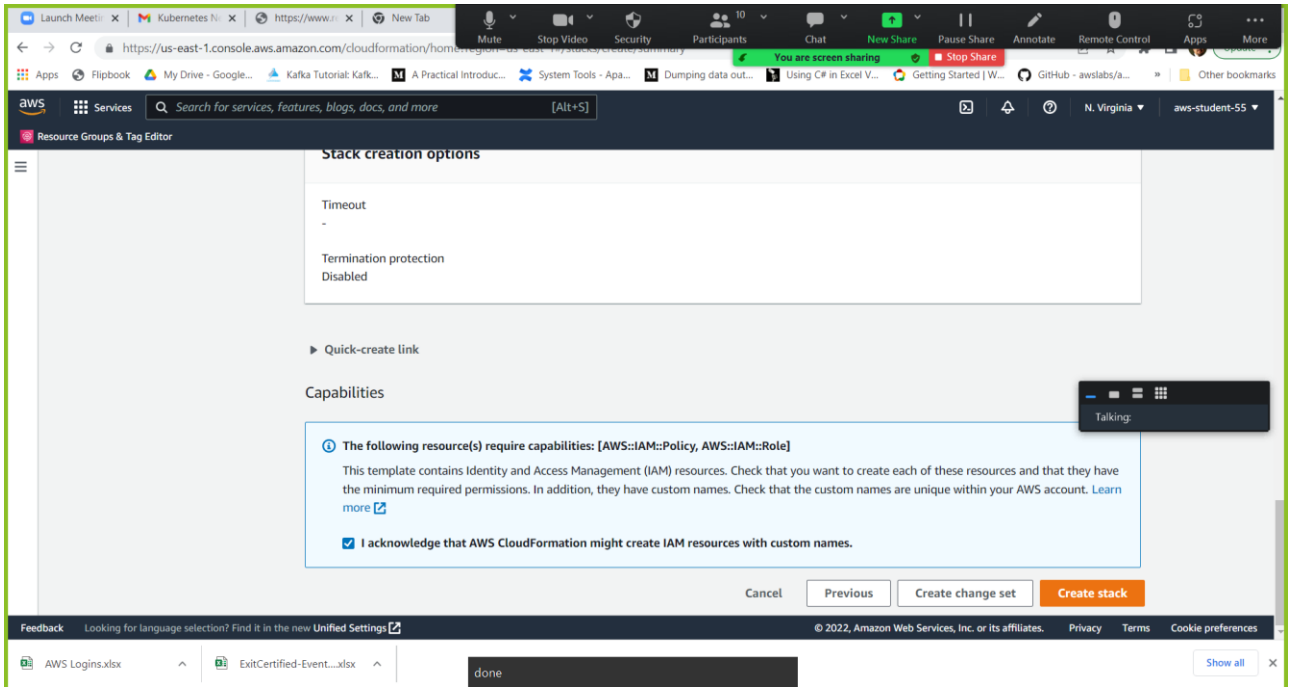
Choose to use the template that is in in the labs/EKS folder in our GitHub repo



Enter a name for our stack like **k8s-networking-cloud9**



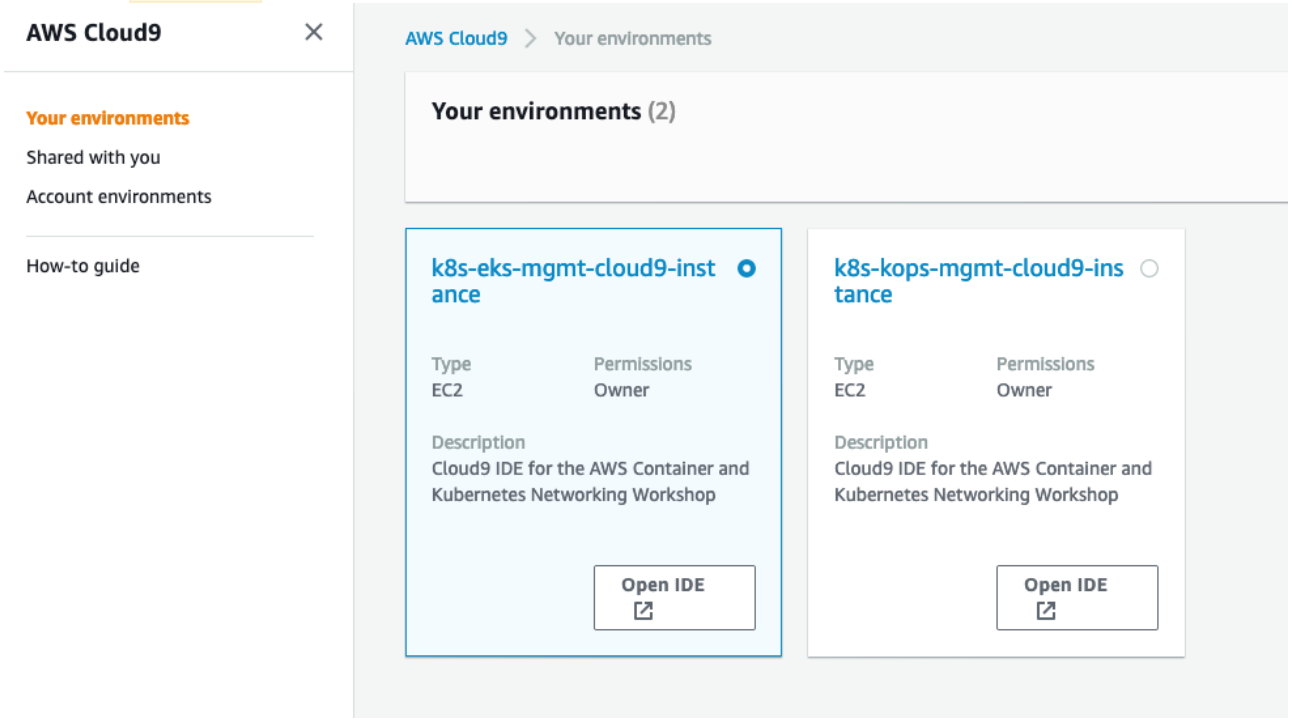
Choose **Next,** and on the customization section select **Next** again



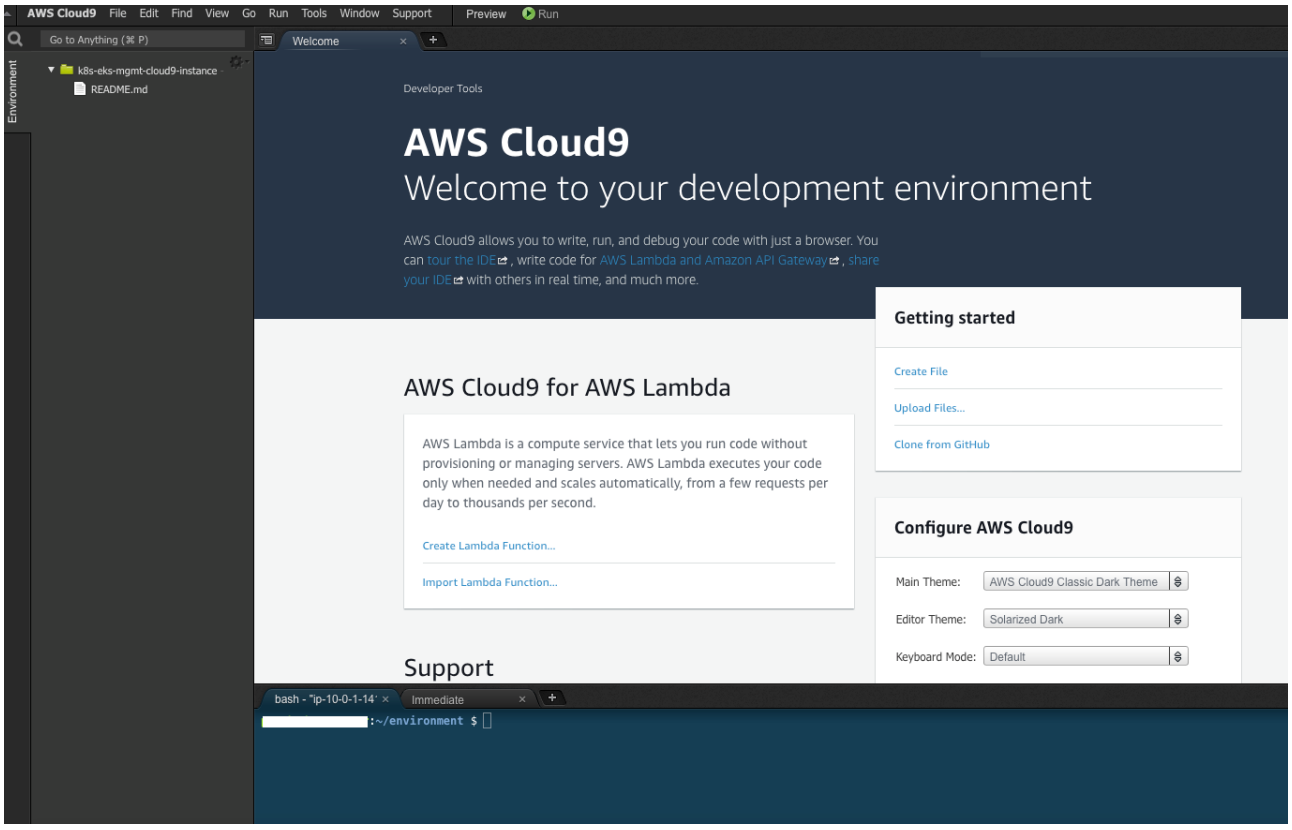
Check the box for acknowledgement and select **Create stack**

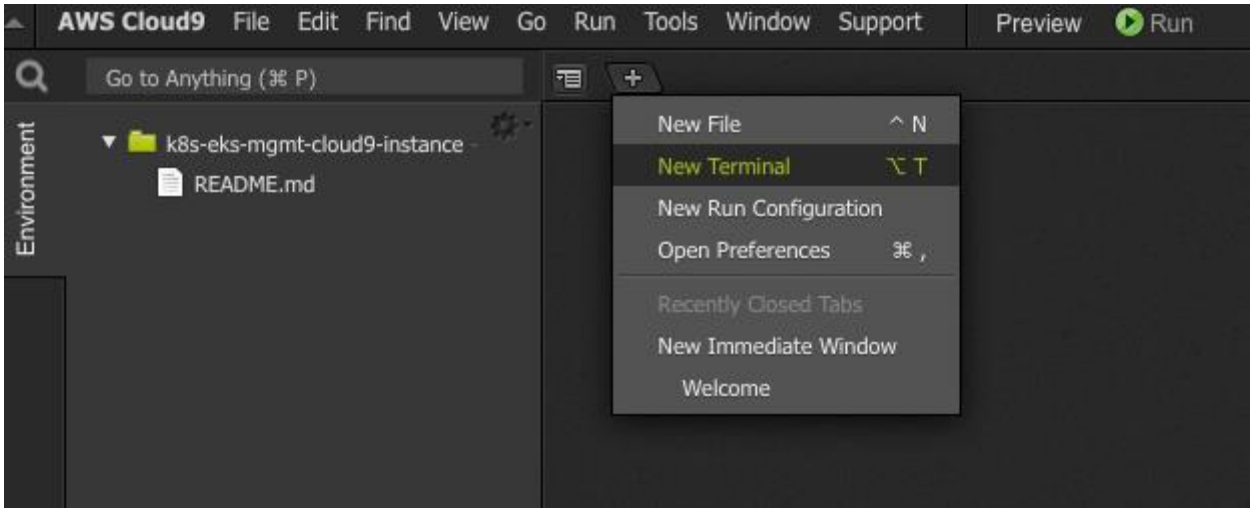
When the Cloud9 environment comes up, open the AWS Cloud9 service in the console in the region where you launched your Cloud9 above:

Click the `Open IDE` button as shown below:

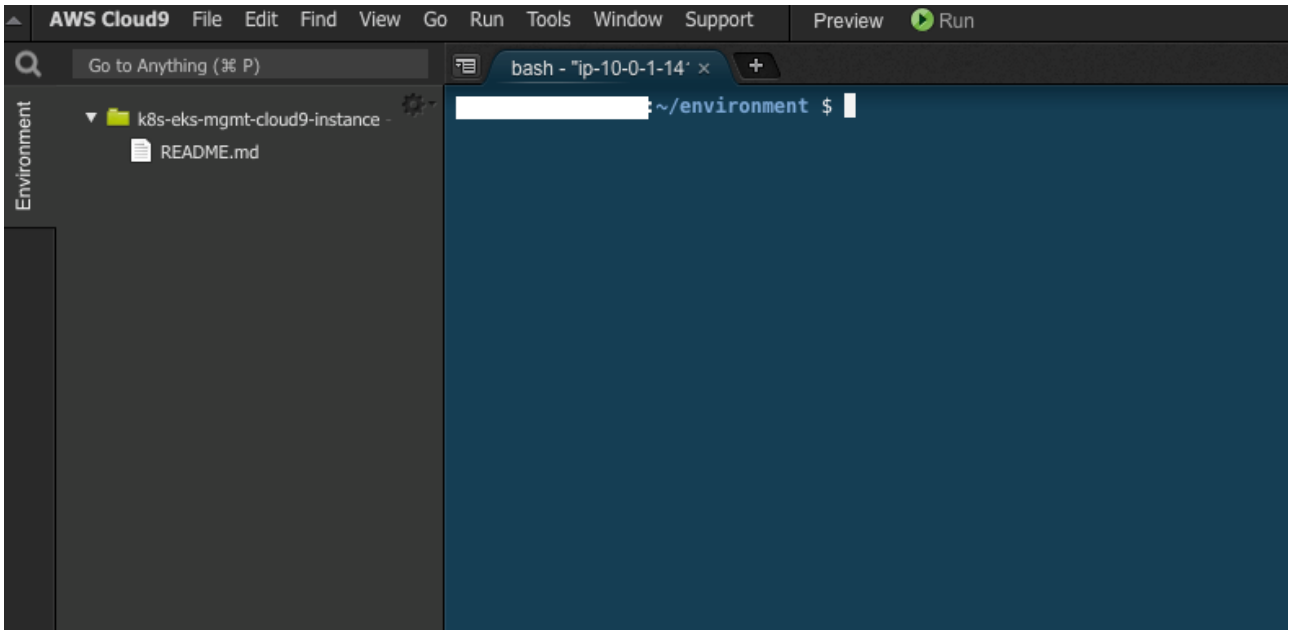


Customize the environment by closing the **welcome tab** and **lower work area**: and opening a new **terminal** tab in the main work area:



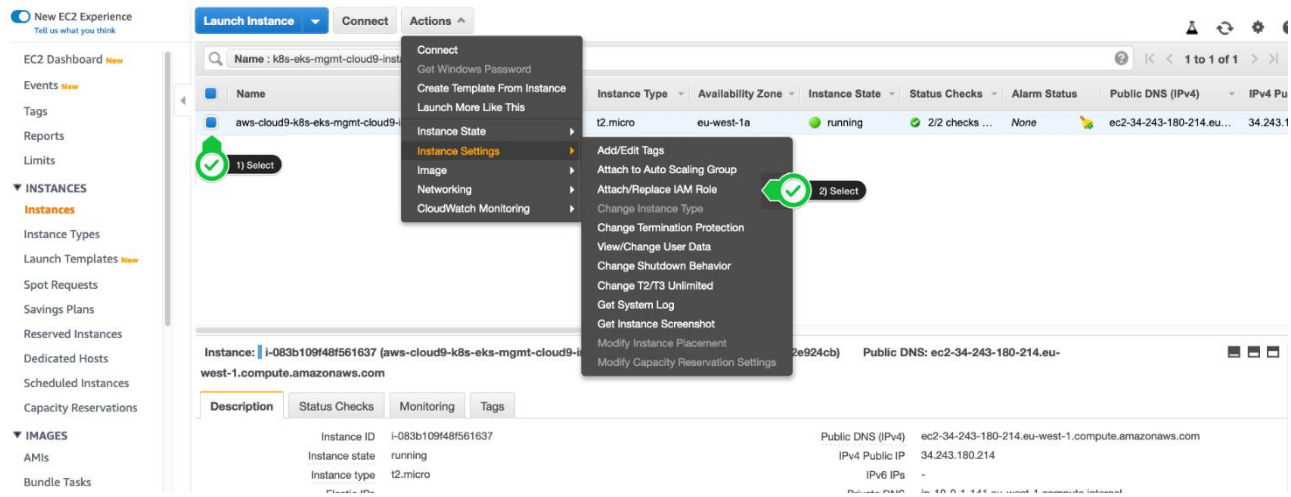


Your workspace should now look like the following screenshot:



Follow this deep link to find your Cloud9 EC2 instance

Select the instance, then choose Actions / Instance Settings / Attach/Replace IAM Role



Choose amazonk8snetworkshop-admin **from the** IAM Role **drop down, and select** Apply

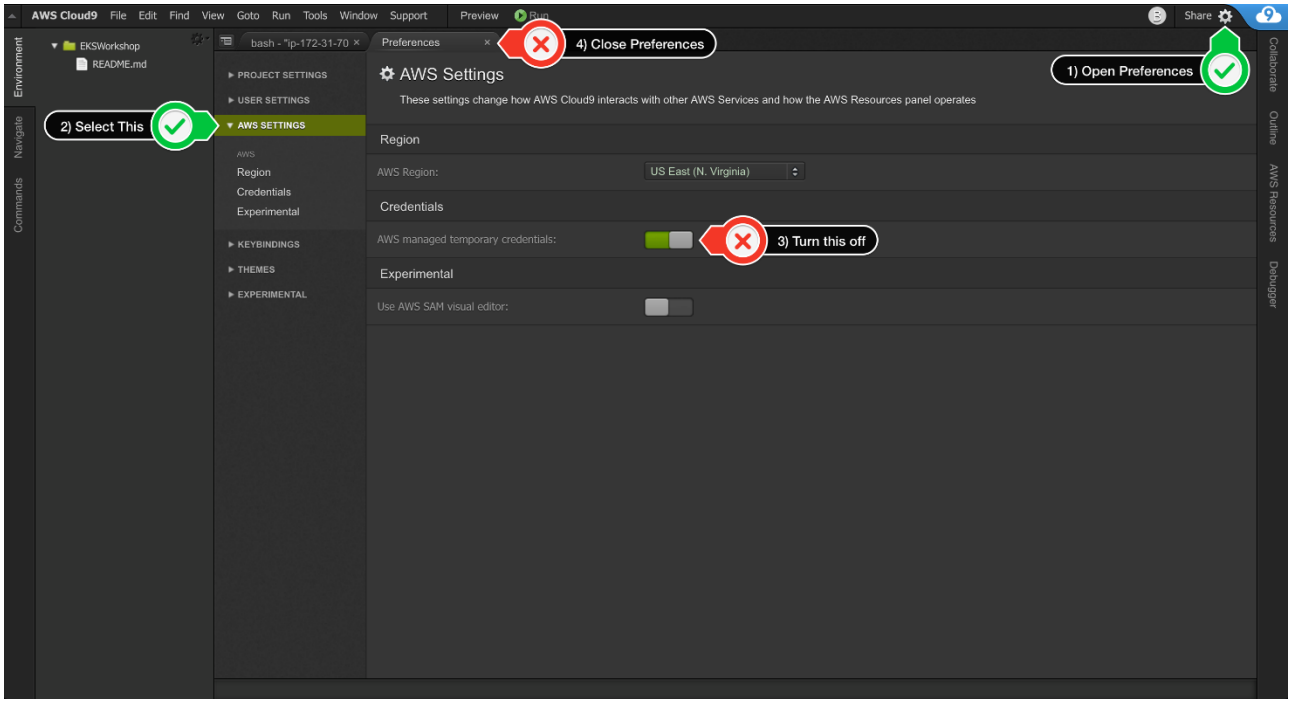
Cloud9 normally manages IAM credentials dynamically. This isn't currently compatible with the EKS IAM authentication, so we will disable it and rely on the IAM role instead.

Return to your workspace and click the sprocket, or launch a new tab to open the Preferences tab

Select AWS SETTINGS

Turn off AWS managed temporary credentials

Close the Preferences tab



To ensure temporary credentials aren't already in place we will also remove any existing credentials file:

$ rm -vf ~/.aws/credentials

Install JQ (JSON Manipulation tool)

$ sudo yum -y install jq

We will configure our aws cli with our current region as default:

Set the AWS_ACCOUNT_ID environment variable.

$ export AWS_ACCOUNT_ID=$(aws sts get-caller-identity --output text --query Account)

Set the AWS_REGION environment variable.

$ export AWS_REGION=$(curl -s 169.254.169.254/latest/dynamic/instance-identity/document | jq -r '.region')

Echo our environment variables to the Bash Profile

$ echo "export AWS_ACCOUNT_ID=${AWS_ACCOUNT_ID}" >> ~/.bash_profile

$ echo "export AWS_REGION=${AWS_REGION}" >> ~/.bash_profile

Configure and validate the default.region value

$ aws configure set default.region ${AWS_REGION}

$ aws configure get default.region

$ cat ~/.bash_profile

Validate the IAM role

Use the GetCallerIdentity CLI command to validate that the Cloud9 IDE is using the correct IAM role.

$ aws sts get-caller-identity

The output assumed-role name has to contain:

`amazonk8snetworkshop-admin`

VALID

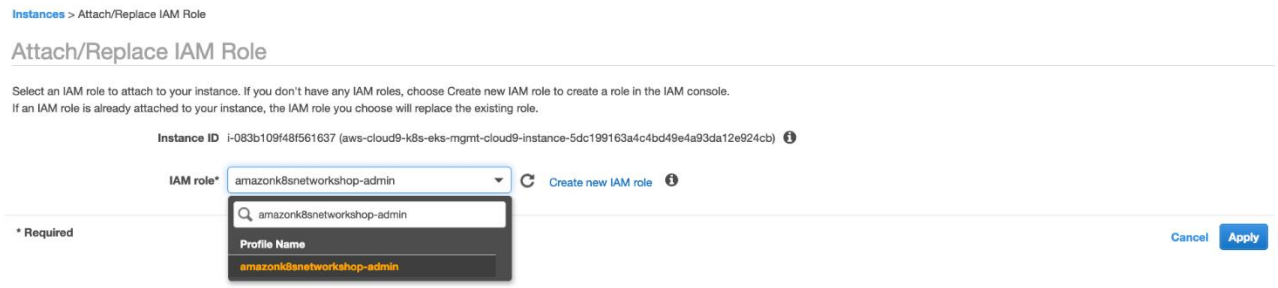
If the *Arn* contains the role name from above and an Instance ID, you may proceed.

```
{
    "Account": "123456789012",
    "UserId": "AROAYD7CFQJWZSIEXYJCC:i-083b109f48f561637",
    "Arn": "arn:aws:sts::123456789012:assumed-
role/amazonk8snetworkshop-admin/i-083b109f48f561637"
}
```

INVALID

If the *Arn* contains `TeamRole`, `MasterRole`, or does not match the role name output above, **DO NOT PROCEED**. Go back and confirm the steps on this page.

```
{
    "Account": "123456789012",
    "UserId": "AROA1SAMPLEAWSIAMROLE:i-01234567890abcdef",
    "Arn": "arn:aws:sts::123456789012:assumed-
role/TeamRole/MasterRole"
}
```



Grab the latest version of eksctl

Console-geoniece:~/environment **$ curl --silent --location "https://github.com/weaveworks/eksctl/releases/download/latest_release/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp**

Console-geoniece:~/environment **$ sudo mv -v /tmp/eksctl /usr/local/bin**

‘/tmp/eksctl’ -> ‘/usr/local/bin/eksctl’

$ eksctl version

`0.59.0-rc.1`

Ensure that kubectl is executable and check our version of kubectl

$ sudo chmod +x /usr/local/bin/kubectl

$ kubectl version

Please run this command to generate SSH Key in Cloud9. This key will be used on the worker node instances to allow ssh access if necessary.

$ ssh-keygen

Press `enter` 3 times to take the default choices

Upload the public key to your EC2 region:

$ aws ec2 import-key-pair --key-name "eksnetworkshop" --public-key-material file://~/.ssh/id_rsa.pub

```
{
    "KeyName": "eksnetworkshop",
    "KeyFingerprint": "ae:41:18:c0:12:3d:62:97:33:43:8b:7e:b7:e0:43:90",
    "KeyPairId": "key-03f01dec284c95e62"
}
```

Set our default terminal editor, use nano if you’re not a vi user

Console-geoniece:~ $ **set -o vi**

Create our cluster

Execute the following command to create our EKS clust for our experiment.

Console-geoniece:~ $ **eksctl create cluster --name=networkshop-eks --nodes=2 --region=${AWS_REGION} --ssh-public-key=eksnetworkshop --zones=us-east-1a,us-east-1b,us-east-1d --version 1.21**

--ssh-public-key : Use the SSH key that we created earlier

--region : Use the region we set earlier.

--nodes : Create the cluster with 2 worker nodes in a nodegroup.

--zones We're going to explicitly set the zones to the ones most likely to not have Insufficient Capacity Exceptions (ICE)

--version : We'll use the most current version available (1.21) at the time of this experiment being created.

```
2021-07-28 03:05:11 [ⓘ]  eksctl version 0.59.0-rc.1
2021-07-28 03:05:11 [ⓘ]  using region us-east-1
2021-07-28 03:05:11 [ⓘ]  subnets for us-east-1a - public:192.168.0.0/19
private:192.168.96.0/19
2021-07-28 03:05:11 [ⓘ]  subnets for us-east-1b - public:192.168.32.0/19
private:192.168.128.0/19
2021-07-28 03:05:11 [ⓘ]  subnets for us-east-1d - public:192.168.64.0/19
private:192.168.160.0/19
2021-07-28 03:05:11 [ⓘ]  nodegroup "ng-0e91fab0" will use ""
[AmazonLinux2/1.21]
2021-07-28 03:05:11 [ⓘ]  using Kubernetes version 1.21
2021-07-28 03:05:11 [ⓘ]  creating EKS cluster "networkshop-eks" in "us-
east-1" region with managed nodes
2021-07-28 03:05:11 [ⓘ]  will create 2 separate CloudFormation stacks
for cluster itself and the initial managed nodegroup
2021-07-28 03:05:11 [ⓘ]  if you encounter any issues, check
CloudFormation console or try 'eksctl utils describe-stacks --region=us-
east-1 --cluster=networkshop-eks'
2021-07-28 03:05:11 [ⓘ]  CloudWatch logging will not be enabled for
cluster "networkshop-eks" in "us-east-1"
2021-07-28 03:05:11 [ⓘ]  you can enable it with 'eksctl utils update-
cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} -
-region=us-east-1 --cluster=networkshop-eks'
2021-07-28 03:05:11 [ⓘ]  Kubernetes API endpoint access will use default
of {publicAccess=true, privateAccess=false} for cluster "networkshop-eks"
in "us-east-1"
2021-07-28 03:05:11 [ⓘ]  2 sequential tasks: { create cluster control
plane "networkshop-eks", 3 sequential sub-tasks: { wait for control plane
to become ready, 1 task: { create addons }, create managed nodegroup "ng-
0e91fab0" } }
2021-07-28 03:05:11 [ⓘ]  building cluster stack "eksctl-networkshop-eks-
cluster"
2021-07-28 03:05:11 [ⓘ]  deploying stack "eksctl-networkshop-eks-
cluster"
```

~/environment $ **POLICY=$(echo -n '{"Version":"2012-10-17","Statement":[{"Effect":"Allow","Principal":{"AWS":"arn:aws:iam::'; echo -n "$ACCOUNT_ID"; echo -n ':root"},"Action":"sts:AssumeRole","Condition":{}}]}')**

:~/environment $ **echo ACCOUNT_ID=$ACCOUNT_ID**

ACCOUNT_ID=124926150123

Console-geoniece:~/environment $ **echo POLICY=$POLICY**

```
POLICY={"Version":"2012-10-
17","Statement":[{"Effect":"Allow","Principal":{"AWS":"arn:aw
s:iam::124926150123:root"},"Action":"sts:AssumeRole","Conditi
on":{}}]}
```

Create a best practice set of roles for our environment

~/environment $ **aws iam create-role --role-name k8sAdmin --description "Kubernetes administrator role (for AWS IAM Authenticator for Kubernetes)." --assume-role-policy-document "$POLICY" --output text --query 'Role.Arn'**

```
arn:aws:iam::124926150123:role/k8sAdmin
```

Console-geoniece:~/environment $ **aws iam create-role --role-name k8sDev --description "Kubernetes developer role (for AWS IAM Authenticator for Kubernetes)." --assume-role-policy-document "$POLICY" --output text --query 'Role.Arn'**

```
arn:aws:iam::124926150123:role/k8sDev
```

Console-geoniece:~/environment $ **aws iam create-role --role-name k8sInteg --description "Kubernetes role for integration namespace in quick cluster." --assume-role-policy-document "$POLICY" --output text --query 'Role.Arn'**

```
arn:aws:iam::124926150123:role/k8sInteg
```

View our ASG created for our EKS cluster

Console-geoniece:~/environment $ **aws autoscaling describe-auto-scaling-groups**

Set the ASG Name for our scripting

$ export ASG_NAME=`aws autoscaling describe-auto-scaling-groups | grep AutoScalingGroupName | cut -d '"' -f 4`

$ echo $ASG_NAME

```
eks-ng-0e91fab0-24bd7592-0660-4254-6772-8a7ef82ffb5d
```

Update the ASG Settings

The defaults for the cluster will be set to a zero worker node sizing. We'll update those configurations.

$ aws autoscaling update-auto-scaling-group --auto-scaling-group-name $ASG_NAME --min-size=1

$ aws autoscaling update-auto-scaling-group --auto-scaling-group-name $ASG_NAME --max-size=2

Verify your Auto Scaling Group changes

Console-geoniece:~/environment $ **aws autoscaling describe-auto-scaling-groups**

Create CMK for encrypting our secrets

$ aws kms create-alias --alias-name alias/eksworkshop --target-key-id $(aws kms create-key --query KeyMetadata.Arn --output text)

Enable Bash completion for eksctl

$ eksctl completion bash >> ~/.bash_completion

$. /etc/profile.d/bash_completion.sh

$. ~/.bash_completion

EKS Cluster Networking Review

Amazon VPC and subnets – All Amazon EKS resources are deployed to one Region in an existing subnet in an existing VPC. For more information, see VPCs and subnets in the Amazon VPC User Guide. Each subnet exists in one Availability Zone. The VPC and subnets must meet requirements such as the following:

VPCs and subnets must be tagged appropriately, so that Kubernetes knows that it can use them for deploying resources, such as load balancers. For more information, see Subnet tagging. If you deploy the VPC using an Amazon EKS provided AWS CloudFormation template or using eksctl, then the VPC and subnets are tagged appropriately for you.

A subnet may or may not have internet access. If a subnet does not have internet access, the pods deployed within it must be able to access other AWS services, such as Amazon ECR, to pull container images. For more information about using subnets that don't have internet access, see Private clusters.

Any public subnets that you use must be configured to auto-assign public IP addresses for Amazon EC2 instances launched within them. For more information, review VPC IP addressing.

The nodes and control plane must be able to communicate over all ports through appropriately tagged security groups. For more information, see Amazon EKS security group considerations.

You can implement a network segmentation and tenant isolation network policy. Network policies are similar to AWS security groups in that you can create network ingress and egress rules. Instead of assigning instances to a security group, you assign network policies to pods using pod selectors and labels. To do this you would install Calico on Amazon EKS.

You can deploy a VPC and subnets that meet the Amazon EKS requirements through manual configuration, or by deploying the VPC and subnets using eksctl, or an Amazon EKS provided AWS CloudFormation template. Both eksctl and the AWS CloudFormation template create the VPC and subnets with the required configuration.

Cluster Verification

$ aws eks list-clusters

Expected output:

ec2-user:~/environment $ **aws eks list-clusters**

```
{
"clusters": [
    "networkshop-eks"
]
}
```

Detailed Cluster Information Verification:

$ aws eks describe-cluster --name <insertclustername>

Expected output:

ec2-user:~/environment $ **aws eks describe-cluster --name networkshop-eks**

```
{
"cluster": {
    "status": "ACTIVE",
    "endpoint": "https://3409E1492A3BD874836B70CE96BB14EF.sk1.us-west-
2.eks.amazonaws.com",
    "logging": {
```

```
        "clusterLogging": [
            {
                "enabled": false,
                "types": [
                    "api",
                    "audit",
                    "authenticator",
                    "controllerManager",
                    "scheduler"
                ]
            }
        ]
    },
    "name": "networkshop-eksctl",
    "tags": {},
    "certificateAuthority": {
        "data":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUN5RENDQWJDZ0F3SUJBZ0lCQURBTkJn
a3Foa2lHOXcwQkFRc0ZBREFWTVJNd0VRWURWUVFERXdwcmRXSmwKY201bGRHVnpNQjRYRFRFN
U1URXlOREEyTURRd05sb1hEVEk1TVRFeU1UQTJNRFF3Tmxvd0ZURVRNQkVHQTFVRQpBeE1LYT
NWaVpYSnVaWFJsY3pDQ0FTSXdEUVlKS29aSWh2Y05BUUVCQlFBRGdnRVBBRENDQVFvQ2dnRUJ
BTGRCCmo0cGhDczZQYWVJazRlay9TRlJzSGtIcnltUk1uWk1YTEtzODh5enlqd25BNW1ET0RO
Njd3aUZEKzVkeDd5UFkKQ2pkNXE5Y3ZoeUpKWmNEanZsSG50WjBoNXhlaTZEMU9FZWpHM3drN
DJWeGtOSURJcmJsNUNVdFp5dUdMUit3ZQpSZ1dyMkhBRHcvMm5Jc3hOcTF2bUEvTjA3YkRvS0
xNWURGN0ZHVHVuN2NMcFlYRUVaUVFyYjAzdXdxYzFJVFZSClZ3em5XRm82Vk9uN3B2K2dyWnp
pcXVnanZnaVdKd0xlTlRCcGVSU3h6SHZRRHhWTnpZeUZDeTdOTE1iUlMrUHMKOStsQUlnbzVS
R2x0cVlhZ3FyWXFteU1NeDhIdHhyOGROcWdnYUtOZHVyQ2JGSkpJYldnS3liVUVVUk5DK3VnZ
QpyY1U0VTZBcTZzQ0J1dEhmUWhzQ0F3RUFBYU1qTUNFd0RnWURWUjBQQVFIL0JBUURBZ0trTU
E4R0ExVWRFd0VCCi93UUZNQU1CQWY4d0RRWUpLb1pJaHZjTkFRUxCUUFEZ2dFQkFJU0hlZ3N
MeTRYMFpSREoyKy91aGExb3hJUUcKblcrbDdpaTlJSUhZRUp3SVpneE9JR3VCQUwvTzRGUGhq
ZEFsbGRFYWhiZGNrNmJiYy9DbUFJUjRTUTk3Y0dTbwpCYk1Pam1TQ2VmS1NHU0xCdzVRQ0pFR
EtHemdLOEpMRDRPSnZNQk1TT0VZMTFpUmZFOGtuYm9IcitXWWh2eE9iCi9wbTMwRWh2NE5Pc1
h4bGh3RU5BS016UjJmdWJ0RVY2eGxKWTl4b2FFSHJUQ21QcjBpd1pvMUozWERma3A2TmEKUVE
0eWQ0NVlUWThndWJVaWdCY2gxQVNUR2NzV0E0elRQbUgvTjlyM25aMmIxM2pOWk9wTzJxYjlM
WEdiVXhkNAo2NVpVUEU1UVAyZUdZVjNZeVZDeUQyZDdJL1VXei92MFNoanBEQlNFU0Z4My9US
1lJZWJkdWFDc05KUT0KLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo="
    },
    "roleArn": "arn:aws:iam::123456789012:role/eksctl-networkshop-eksctl-
cluster-ServiceRole-1K0Z94UFDWU68",
    "resourcesVpcConfig": {
        "subnetIds": [
            "subnet-073d3d88dd38a84fb",
            "subnet-0eda1ddfe765da074",
            "subnet-0d66c7d89c085b399",
            "subnet-0ae64a91c45a9e958",
            "subnet-0b0f809529ad8c1f9",
            "subnet-02be14bbc3ab73b7b"
        ],
        "vpcId": "vpc-05c2e236eb0e83d71",
        "endpointPrivateAccess": false,
        "endpointPublicAccess": true,
        "securityGroupIds": [
            "sg-0103d40263818e3a0"
        ]
    },
    "platformVersion": "eks.3",
    "version": "1.14",
    "arn": "arn:aws:eks:us-west-2:123456789012:cluster/networkshop-
eksctl",
    "identity": {
        "oidc": {
            "issuer": "https://oidc.eks.us-west-
2.amazonaws.com/id/3409E1492A3BD874836B70CE96BB14EF"
        }
    },
    "createdAt": 1574575029.806
}
}
```

Cluster Information for Master and DNS Server (CoreDNS) Endpoints Verification:

$ kubectl cluster-info

Expected output:

```
Kubernetes master is running at
https://3409E1492A3BD874836B70CE96BB14EF.sk1.us-west-
2.eks.amazonaws.com

CoreDNS is running at
https://3409E1492A3BD874836B70CE96BB14EF.sk1.us-west-
2.eks.amazonaws.com/api/v1/namespaces/kube-
system/services/kube-dns:dns/proxy
```

To debug and diagnose cluster problems, we would normally use 'kubectl cluster-info dump'.

ec2-user:~/environment $

Cross Account ENI Information Verification:

Find the ENI for our EKS Cluster

$ aws ec2 describe-network-interfaces --region ${AWS_REGION}

Look for the ENI section for our EKS Cluster

```
"Description": "Amazon EKS networkshop-eks",
"NetworkInterfaceId": "eni-0e5cd759bb9232825",
```

Expected output:

ec2-user:~/environment $ **aws ec2 describe-network-interfaces --network-interface-ids eni-091faa26c79f60aaf --region ${AWS_REGION}**

```
{
"NetworkInterfaces": [
    {
        "Status": "in-use",
        "MacAddress": "0e:59:58:4e:d4:e0",
        "SourceDestCheck": true,
        "AvailabilityZone": "us-west-2d",
        "Description": "Amazon EKS networkshop-eksctl",
        "NetworkInterfaceId": "eni-091faa26c79f60aaf",
        "VpcId": "vpc-05c2e236eb0e83d71",
```

```
        "PrivateIpAddresses": [
            {
                "PrivateDnsName": "ip-192-168-171-7.us-west-
2.compute.internal",
                "Primary": true,
                "PrivateIpAddress": "192.168.171.7"
            }
        ],
        "RequesterManaged": false,
        "PrivateDnsName": "ip-192-168-171-7.us-west-2.compute.internal",
        "RequesterId": "AROAXHTQMN2OZD76U7M4M:AmazonEKS",
        "InterfaceType": "interface",
        "Attachment": {
            "Status": "attached",
            "DeviceIndex": 1,
            "AttachTime": "2019-11-24T06:03:53.000Z",
            "DeleteOnTermination": true,
            "AttachmentId": "eni-attach-03fd6308f320725c7",
            "InstanceOwnerId": "305882430652"
        },
        "Groups": [
            {
                "GroupName": "eksctl-networkshop-eksctl-cluster-
ControlPlaneSecurityGroup-KMLNZFWG6FLM",
                "GroupId": "sg-0103d40263818e3a0"
            },
            {
                "GroupName": "eks-cluster-sg-networkshop-eksctl-
2101059127",
                "GroupId": "sg-084bbe90eaf651d55"
            }
        ],
        "Ipv6Addresses": [],
        "OwnerId": "123456789012",
        "SubnetId": "subnet-02be14bbc3ab73b7b",
        "TagSet": [],
        "PrivateIpAddress": "192.168.171.7"
    }
]
}
```


Verify the Cluster Connectivity with Kubectl

~/environment $ **aws eks --region ${AWS_REGION} update-kubeconfig --name networkshop-eks**

```
Updated context arn:aws:eks:us-east-
1:124926150123:cluster/networkshop-eks in /home/ec2-
user/.kube/config
```

~/environment $ kubectl get svc

```
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   2d15h
```

Console-geoniece:~/environment $ **kubectl get nodes -o wide**

No resources found

Copy K8S manifest files to our working folder

$ wget https://raw.githubusercontent.com/GeorgeNiece/KubernetesNetworking/main/labs/EKS/eksConfigFiles.zip

$ mkdir eksConfigFiles

$ cd eksConfigFiles

$ unzip ../eksConfigFiles.zip

Create Pods in our EKS cluster

ec2-user:~/environment $ **kubectl apply -f ~/eksConfigFiles/worker_hello.yaml**

Pod Creation Verification

ec2-user:~/environment $ **kubectl get pod -o wide**

```
NAME                            READY   STATUS    RESTARTS
AGE   IP                NODE
NOMINATED NODE   READINESS GATES
worker-hello-5bfdf775d7-46f2g   1/1     Running   0
20s   192.168.12.232   ip-192-168-9-39.us-west-
2.compute.internal    <none>           <none>
worker-hello-5bfdf775d7-4rj9r   1/1     Running   0
20s   192.168.17.79    ip-192-168-9-39.us-west-
2.compute.internal    <none>           <none>
worker-hello-5bfdf775d7-5h577   1/1     Running   0
20s   192.168.77.124   ip-192-168-68-66.us-west-
2.compute.internal   <none>           <none>
worker-hello-5bfdf775d7-b74jm   1/1     Running   0
20s   192.168.31.64    ip-192-168-9-39.us-west-
2.compute.internal    <none>           <none>
worker-hello-5bfdf775d7-d2kfb   1/1     Running   0
20s   192.168.71.29    ip-192-168-68-66.us-west-
2.compute.internal   <none>           <none>
worker-hello-5bfdf775d7-fwb8g   1/1     Running   0
20s   192.168.24.254   ip-192-168-9-39.us-west-
2.compute.internal    <none>           <none>
worker-hello-5bfdf775d7-gtj59   1/1     Running   0
20s   192.168.68.68    ip-192-168-68-66.us-west-
2.compute.internal   <none>           <none>
worker-hello-5bfdf775d7-gw5cz   1/1     Running   0
20s   192.168.23.16    ip-192-168-9-39.us-west-
2.compute.internal    <none>           <none>
worker-hello-5bfdf775d7-l86gx   1/1     Running   0
20s   192.168.9.126    ip-192-168-9-39.us-west-
2.compute.internal    <none>           <none>
```

```
worker-hello-5bfdf775d7-ljwzz   1/1     Running   0
20s   192.168.78.199   ip-192-168-68-66.us-west-
2.compute.internal   <none>           <none>
worker-hello-5bfdf775d7-pnckd   1/1     Running   0
20s   192.168.71.96    ip-192-168-68-66.us-west-
2.compute.internal   <none>           <none>
worker-hello-5bfdf775d7-qsbvz   1/1     Running   0
20s   192.168.23.18    ip-192-168-9-39.us-west-
2.compute.internal    <none>           <none>
worker-hello-5bfdf775d7-s4dlk   1/1     Running   0
20s   192.168.93.13    ip-192-168-68-66.us-west-
2.compute.internal   <none>           <none>
worker-hello-5bfdf775d7-vtqrl   1/1     Running   0
20s   192.168.10.254   ip-192-168-9-39.us-west-
2.compute.internal    <none>           <none>
worker-hello-5bfdf775d7-zgc72   1/1     Running   0
20s   192.168.79.52    ip-192-168-68-66.us-west-
2.compute.internal   <none>           <none>
```

Pods in Pending

Console-geoniece:~/environment/eksConfigFiles $ kubectl get pods -o wide

```
NAME                              READY   STATUS    RESTARTS
AGE     IP       NODE     NOMINATED NODE   READINESS GATES
worker-hello-67bf4d7bbc-5cmhp   0/1     Pending   0
8m16s   <none>   <none>   <none>           <none>
worker-hello-67bf4d7bbc-62skg   0/1     Pending   0
8m16s   <none>   <none>   <none>           <none>
worker-hello-67bf4d7bbc-7rtpz   0/1     Pending   0
8m16s   <none>   <none>   <none>           <none>
```

If your pods are all pending, rather than Running, that could indicate that the NodeGroup was not correctly created.

$ kubectl get nodes

```
No resources found
```

If we don't have nodes in our cluster, none of our pods will ever come out of pending state. We need to build them for our cluster

$ eksctl create nodegroup --cluster networkshop-eks --name ng-workshop --node-type t2.small --nodes 3 --nodes-min 2 --nodes-max 3 --ssh-access --ssh-public-key eksnetworkshop

Pod to Pod Communication (Intra Node)

Connect into a Pod using information above and ping the Pod with the same node name e.g. "ip-192-168-17-79.us-east-1.compute.internal":**

$ kubectl exec -ti worker-hello-5bfdf775d7-46f2g sh

~/go$ ping 192.168.17.79

```
PING 192.168.17.79 (192.168.17.79): 56 data bytes
64 bytes from 192.168.17.79: seq=0 ttl=254 time=0.065 ms
64 bytes from 192.168.17.79: seq=1 ttl=254 time=0.062 ms
64 bytes from 192.168.17.79: seq=2 ttl=254 time=0.058 ms
^C
--- 192.168.17.79 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
ec2-user:~/environment $
```

Pod to Pod Communication (Inter Node)

ec2-user:~/environment $ kubectl exec -ti worker-hello-5bfdf775d7-46f2g sh

/go # ping 192.168.77.124

```
PING 192.168.77.124 (192.168.77.124): 56 data bytes
64 bytes from 192.168.77.124: seq=0 ttl=253 time=0.605 ms
64 bytes from 192.168.77.124: seq=1 ttl=253 time=0.584 ms
64 bytes from 192.168.77.124: seq=2 ttl=253 time=0.574 ms
^C
--- 192.168.77.124 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.574/0.587/0.605 ms
```

Pod to External Communication

ec2-user:~/environment $ kubectl exec -ti worker-hello-5bfdf775d7-46f2g sh

/go #ping www.google.com

```
PING www.google.com (172.217.14.196): 56 data bytes
64 bytes from 172.217.14.196: seq=0 ttl=42 time=8.925 ms
64 bytes from 172.217.14.196: seq=1 ttl=42 time=8.948 ms
64 bytes from 172.217.14.196: seq=2 ttl=42 time=9.008 ms
64 bytes from 172.217.14.196: seq=3 ttl=42 time=8.993 ms
^C
--- www.google.com ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 8.925/8.968/9.008 ms
```

Exploring Pod IP

You still need to be in the pod ssh session, or to reconnect to one of the existing pods.

~/go # ifconfig

```
eth0      Link encap:Ethernet  HWaddr 1E:39:1C:DA:DD:B5
          inet addr:192.168.12.232  Bcast:192.168.12.232
Mask:255.255.255.255
          UP BROADCAST RUNNING MULTICAST  MTU:9001  Metric:1
```

```
          RX packets:154 errors:0 dropped:0 overruns:0
frame:0
          TX packets:138 errors:0 dropped:0 overruns:0
carrier:0
          collisions:0 txqueuelen:0
          RX bytes:15880 (15.5 KiB)  TX bytes:13104 (12.7
KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0
carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Quick look at the Node (filtering on instance-id) in EC2 Console will show the Secondary IP addresses allocated to the Node

Exploring Interface used for Pod Default RouteTable

Connect to one of the pods, as we've done in this section of the experiment

ec2-user:~/environment $ kubectl exec -ti worker-hello-5bfdf775d7-46f2g sh

/go # ip route

```
default via 169.254.1.1 dev eth0
169.254.1.1 dev eth0 scope link
```

/go # exit

ec2-user:~/environment $