**EKS Cluster with Scalable Custom Node Group**

**ReadMe**

**Introduction:**

This document will show a detailed process on how to create a Scalable Kubernetes Cluster using EKS, Custom Node Group, Autoscaling and Service Deployment.

Step 1: Pre-requisites to build an environment on the local system to Install and Manage the Cluster remotely.

Step 2: Installing EKS Cluster to create a Control Plane without Nodes.

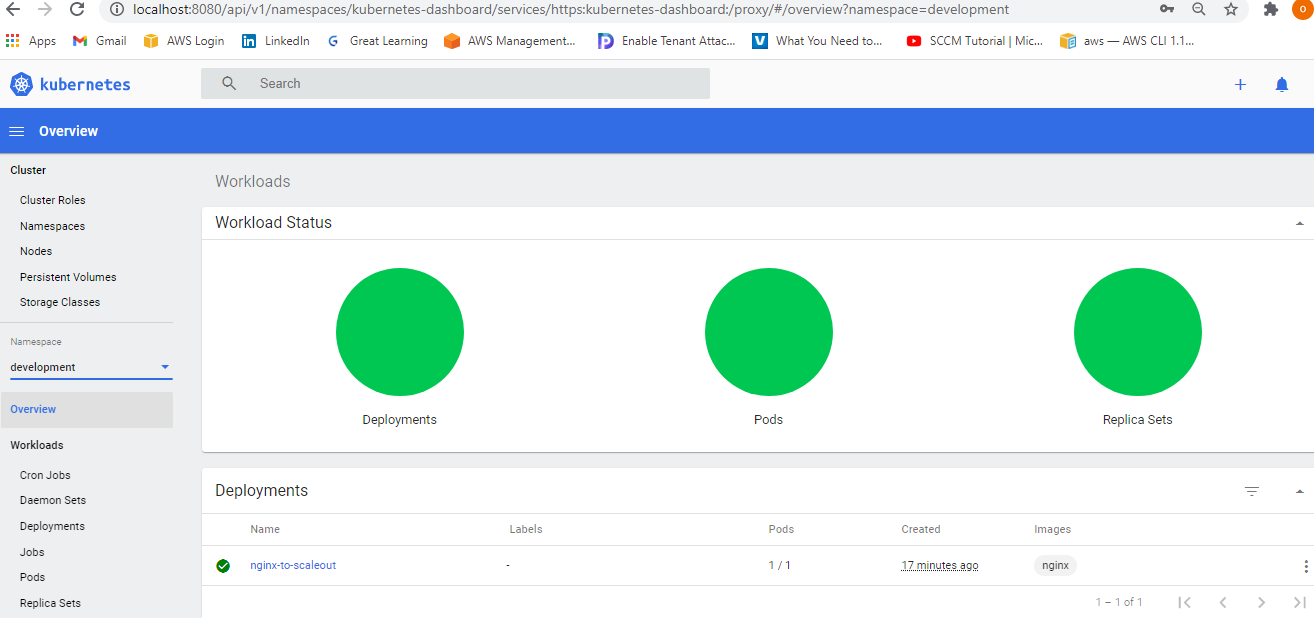
Step 3: Install a custom Node Group.

Step 4: Configure Cluster and set-up monitoring tools like Dashboard.

Step 5: Set-up Auto Scalar for Cluster.

Step 6: Deploy an application, Scaling at the Pods level and Scaling at the Cluster level.

Step 7: Clean-up by deleting the resources



EKS Dashboard

**Overview – EKS Cluster**



AWS Cloud

Availability Zone A

Availability Zone B



VPC

192.168.0.0/16



NAT Gateway



Public Subnet

192.168.32.0/19



Private Subnet 1

192.168.64.0/19



Private Subnet 2

192.168.96.0/19

Node 1

Node 2

Amazon EKS

Auto Scaling Group

Internet Gateway

**Pre-requisites:**

1. **Kubectl** - A command line tool for working with Kubernetes clusters

$curl -o kubectl <https://amazon-eks.s3.us-west-2.amazonaws.com/1.19.6/2021-01-05/bin/darwin/amd64/kubectl>

Provide executable permission:

$chmod +x ./kubectl

$mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH

$echo 'export PATH=$PATH:$HOME/bin' >> ~/.bash\_profile

$kubectl version --short --client

Client Version: v1.19.6-eks-49a6c0

Install on WSL Ubuntu:

$ curl -LO [https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl](https://dl.k8s.io/release/$(curl%20-L%20-s%20https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl)

Verify with checksum:

$ curl -LO [https://dl.k8s.io/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256](https://dl.k8s.io/$(curl%20-L%20-s%20https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256)

$ echo "$(<kubectl.sha256) kubectl" | sha256sum –check

kubectl: OK

$ kubectl version --short --client

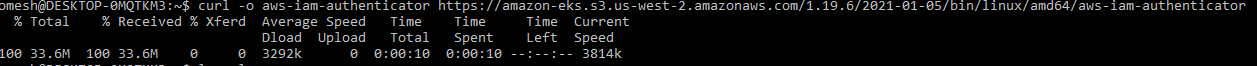
Client Version: v1.21.0

1. **Required IAM permissions** – The IAM security principal that you're using must have permissions to work with Amazon EKS IAM roles and service linked roles, AWS CloudFormation, and a VPC and related resources



Download the Amazon EKS vended aws-iam-authenticator binary from Amazon S3

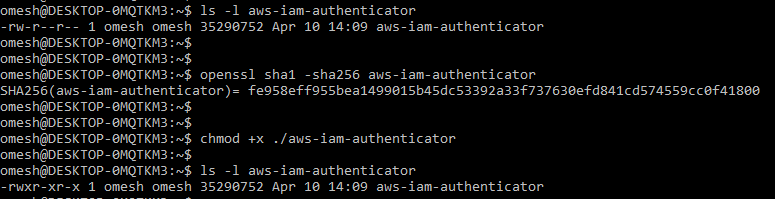
$curl -o aws-iam-authenticator <https://amazon-eks.s3.us-west-2.amazonaws.com/1.19.6/2021-01-05/bin/linux/amd64/aws-iam-authenticator>



$openssl sha1 -sha256 aws-iam-authenticator

Apply execute permissions to the binary

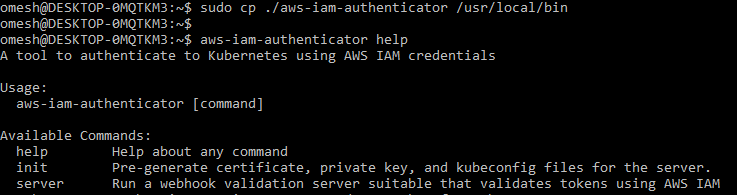
$chmod +x ./aws-iam-authenticator



$sudo cp ./aws-iam-authenticator /usr/local/bin

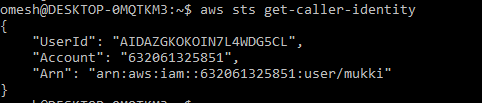
Test that the aws-iam-authenticator binary works

$ aws-iam-authenticator help



Verify IAM Role:

$ aws sts get-caller-identity (after Profile set)



1. **Ekctl-** A command line tool for working with EKS clusters that automates many individual tasks. The eksctl command line utility provides the fastest and easiest way to create a new cluster with nodes for Amazon EKS.

$curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz" | tar xz -C /tmp

$ sudo mv /tmp/eksctl /usr/local/bin

$ eksctl version

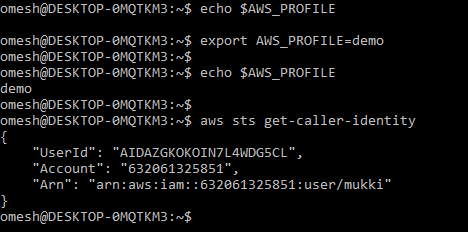
0.44.0

1. **AWSCLI:**

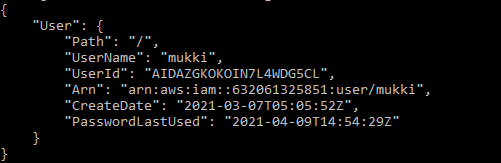
$ aws configure --profile demo

$ export AWS\_PROFILE=demo

$ echo $AWS\_PROFILE



$ aws iam get-user



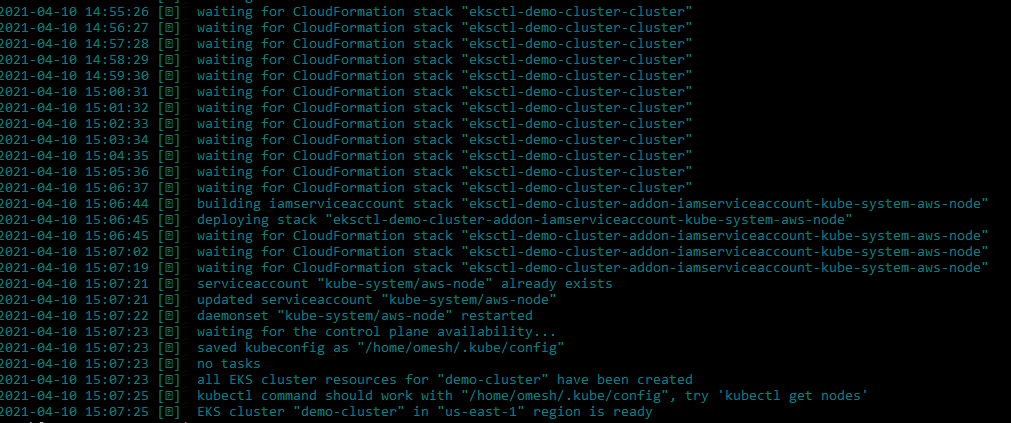
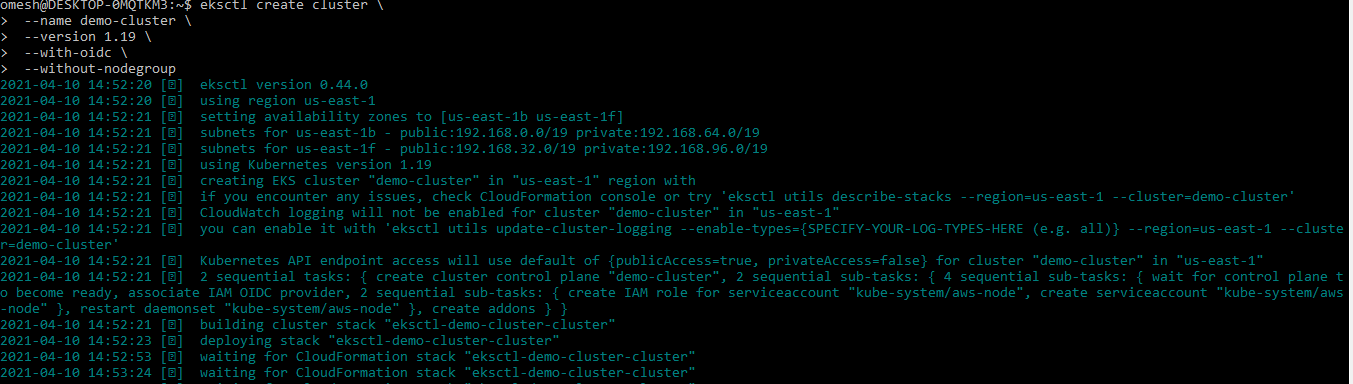
$ eksctl get clusters

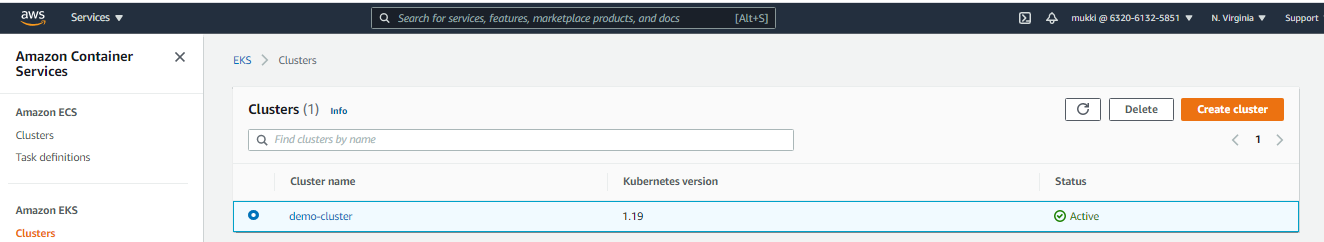


**EKS Cluster Creation without Node Group**

1). Create the EKS Control Plane in the Region **us-east-1** by setting the **AWS\_Profile=demo**

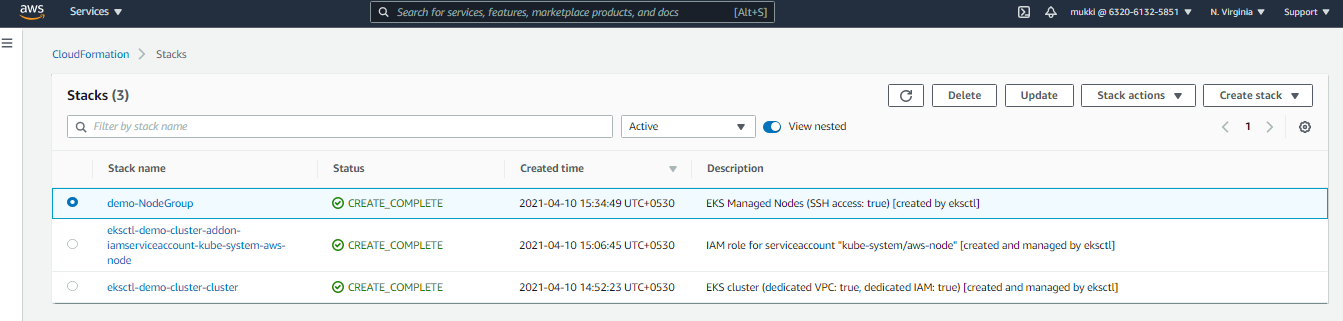
$ eksctl create cluster \ --name demo-cluster \ --version 1.19 \ --with-oidc \ --without-nodegroup

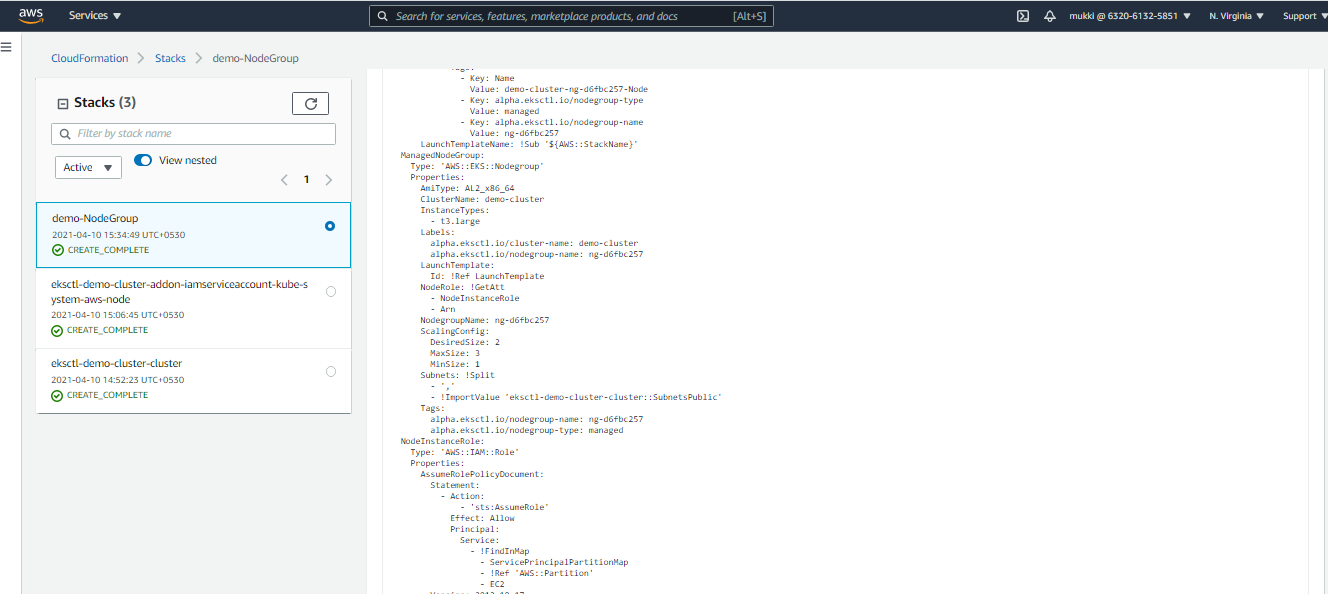




#### Create Node Group using Cloud Formation template demo-ng.yaml:

Since the Cluster is set-up without any Node Group so we will add a custom Node Group demo-NodeGroup using t3.large rather than custom m5.large as it’s a development environment only.



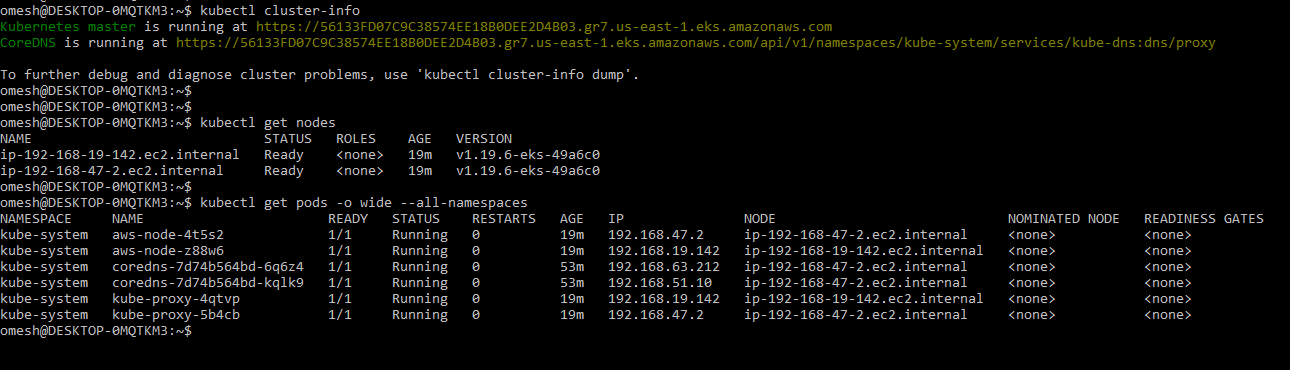


#### Test the cluster status:

$ kubectl cluster-info

$ kubectl get nodes

$ kubectl get pods --all-namespaces -o wide



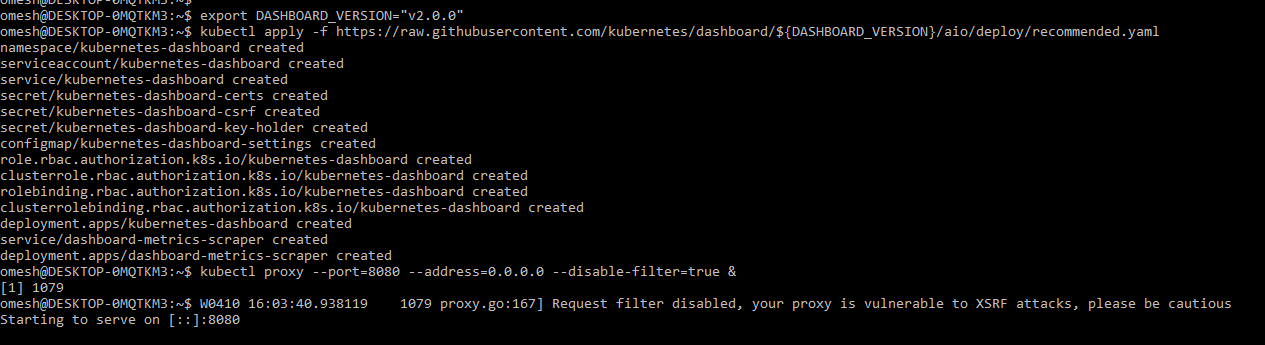
# DEPLOY KUBERNETES DASHBOARD:

$export DASHBOARD\_VERSION="v2.0.0"

$kubectl apply -f [https://raw.githubusercontent.com/kubernetes/dashboard/${DASHBOARD\_VERSION}/aio/deploy/recommended.yaml](https://raw.githubusercontent.com/kubernetes/dashboard/$%7bDASHBOARD_VERSION%7d/aio/deploy/recommended.yaml)

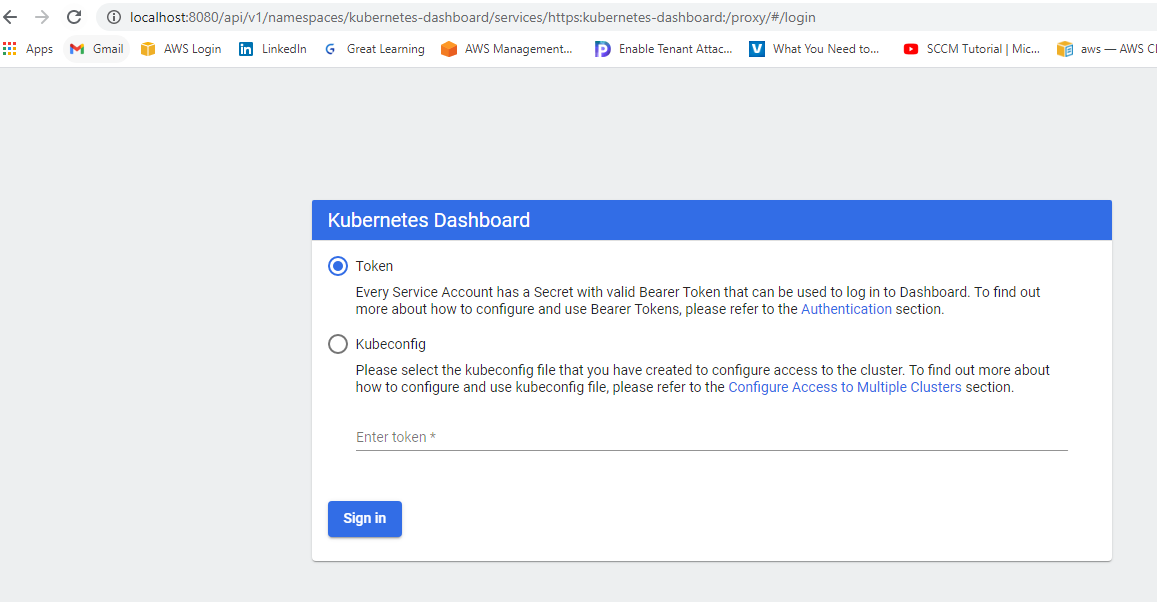
Since this is deployed to our private cluster, we need to access it via a proxy. kube-proxy is available to proxy our requests to the dashboard service. In your workspace, run the following command:

$kubectl proxy --port=8080 --address=0.0.0.0 --disable-filter=true &



Browse the URL from the browser of your Local System:

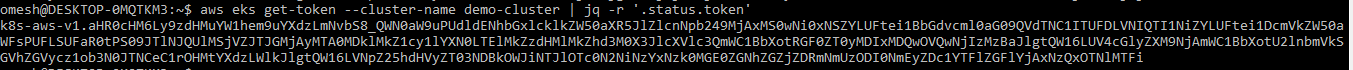
http://localhost:8080/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/



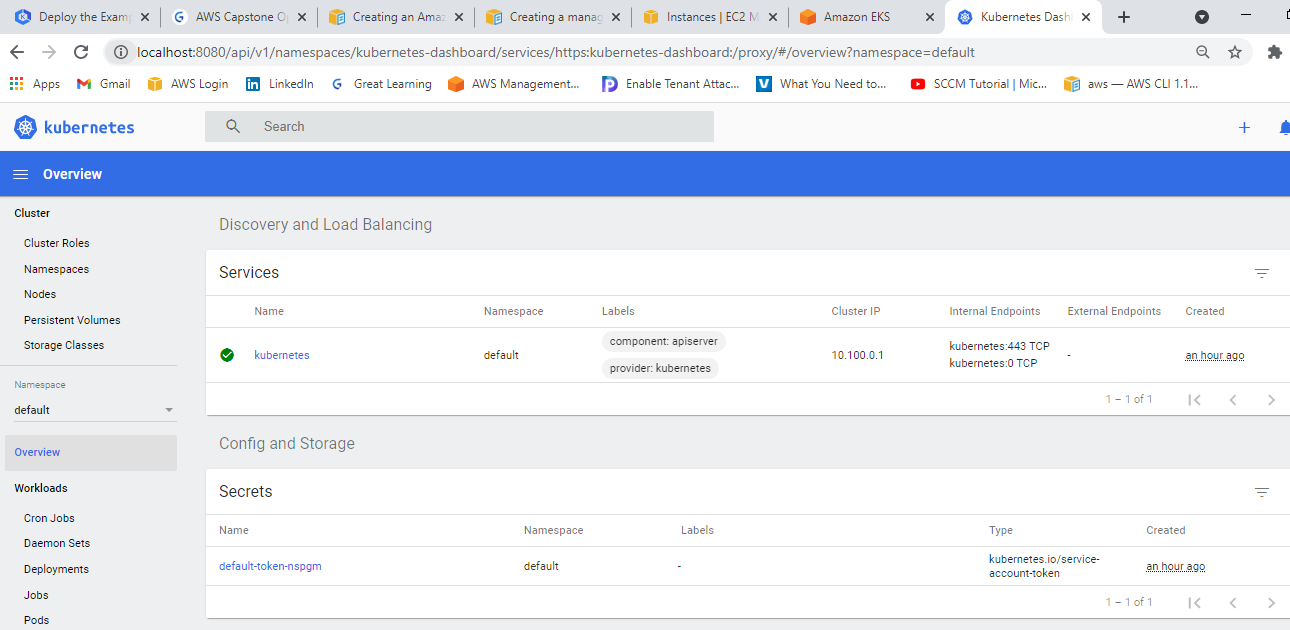
Open a New Terminal Tab and enter and generate the Token:

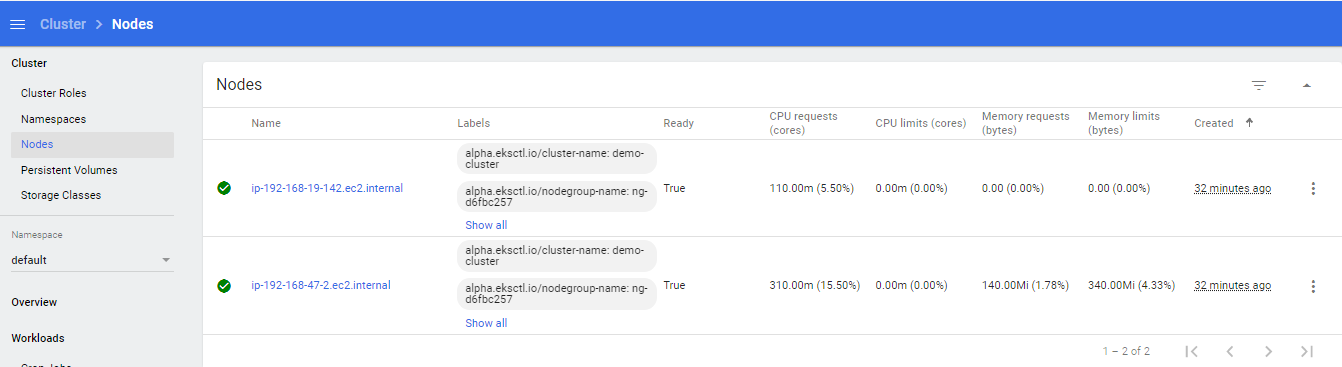
(aws eks get-token --cluster-name eksworkshop-eksctl | jq -r '.status.token')

$ aws eks get-token --cluster-name demo-cluster | jq -r '.status.token'



Copy and Paste this URL token to generate the Kubernetes Dashborad.



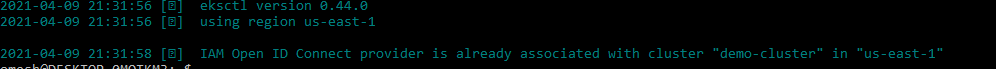


**Autoscaling**

With IAM roles for service accounts on Amazon EKS clusters, you can associate an IAM role with a [Kubernetes service account](https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/). This service account can then provide AWS permissions to the containers in any pod that uses that service account. With this feature, you no longer need to provide extended permissions to the node IAM role so that pods on that node can call AWS APIs.

Enabling IAM roles for service accounts on your cluster

$eksctl utils associate-iam-oidc-provider \ --cluster demo-cluster \ --approve



( Note: Cluster built with eksctl auto creates oidc provider)

$ nano cluster-autoscaler-policy.json

$aws iam create-policy \

> --policy-name AmazonEKSClusterAutoscalerPolicy \

> --policy-document <file://cluster-autoscaler-policy.json>

1. Create Role and attach IAM Policy using eksctl

Run the following command if you created your cluster with eksctl

$eksctl create iamserviceaccount \

--cluster=demo-cluster \

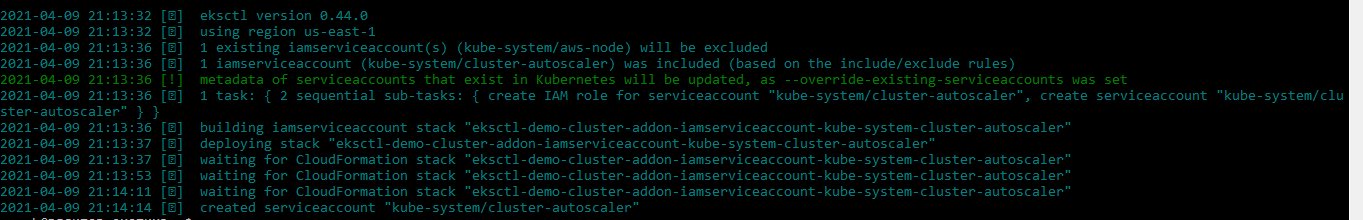
--namespace=kube-system \

--name=cluster-autoscaler \

--attach-policy-arn=arn:aws:iam::632061325851:policy/AmazonEKSClusterAutoscalerPolicy \

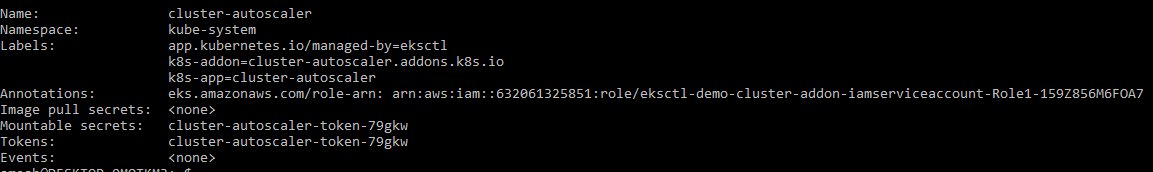
--override-existing-serviceaccounts \

--approve



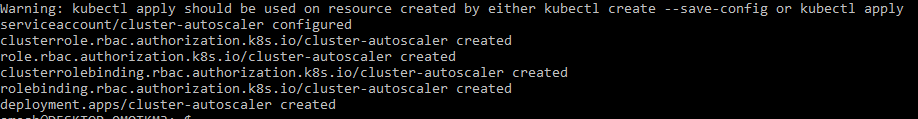
Make sure your service account with the ARN of the IAM role is annotated

$ kubectl -n kube-system describe sa cluster-autoscaler



**Deploy the Cluster Autoscaler:**

$kubectl apply -f <https://raw.githubusercontent.com/kubernetes/autoscaler/master/cluster-autoscaler/cloudprovider/aws/examples/cluster-autoscaler-autodiscover.yaml>



To prevent CA from removing nodes where its own pod is running, we will add the cluster-autoscaler.kubernetes.io/safe-to-evict annotation to its deployment with the following command

$kubectl -n kube-system \ annotate deployment.apps/cluster-autoscaler \

cluster-autoscaler.kubernetes.io/safe-to-evict="false"



Patch the deployment to add the cluster-autoscaler.kubernetes.io/safe-to-evict annotation to the Cluster Autoscaler pods

$ kubectl patch deployment cluster-autoscaler \ -n kube-system \

-p '{"spec":{"template":{"metadata":{"annotations":{"cluster-autoscaler.kubernetes.io/safe-to-evict": "false"}}}}}'



Edit the Cluster Autoscaler deployment to change cluster name and add below lines.

$ kubectl -n kube-system edit deployment.apps/cluster-autoscaler

--balance-similar-node-groups

--skip-nodes-with-system-pods=false



Finally let’s update the autoscaler image:

$ kubectl set image deployment cluster-autoscaler \ -n kube-system \

cluster-autoscaler=k8s.gcr.io/autoscaling/cluster-autoscaler:v1.19.n



Watch the logs

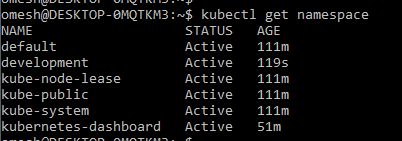
$kubectl -n kube-system logs -f deployment.apps/cluster-autoscaler

**Deploy a sample app**

Create a namespace “development”

$ kubectl create namespace developmet

$ kubectl get namespace



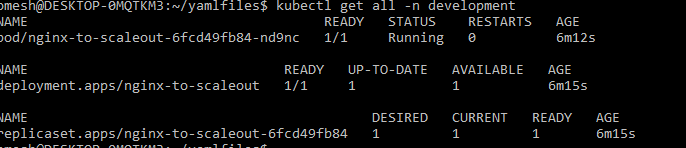
Make the development as default space:

$ kubectl config set-context --current --namespace=development

Deploy the pods:

$ kubectl apply -f nginx.yaml

$ kubectl get all -n development



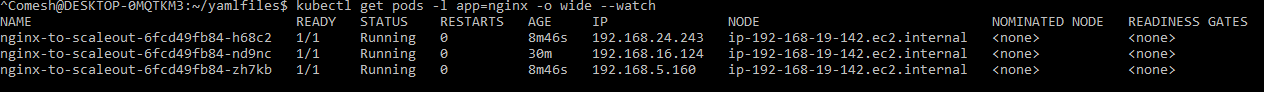
**Autoscaling Test: Pod Level and Cluster Level**

**Initial Set-up: Service deployed with Pod Replicas = 3 and Cluster Node desired state=2**

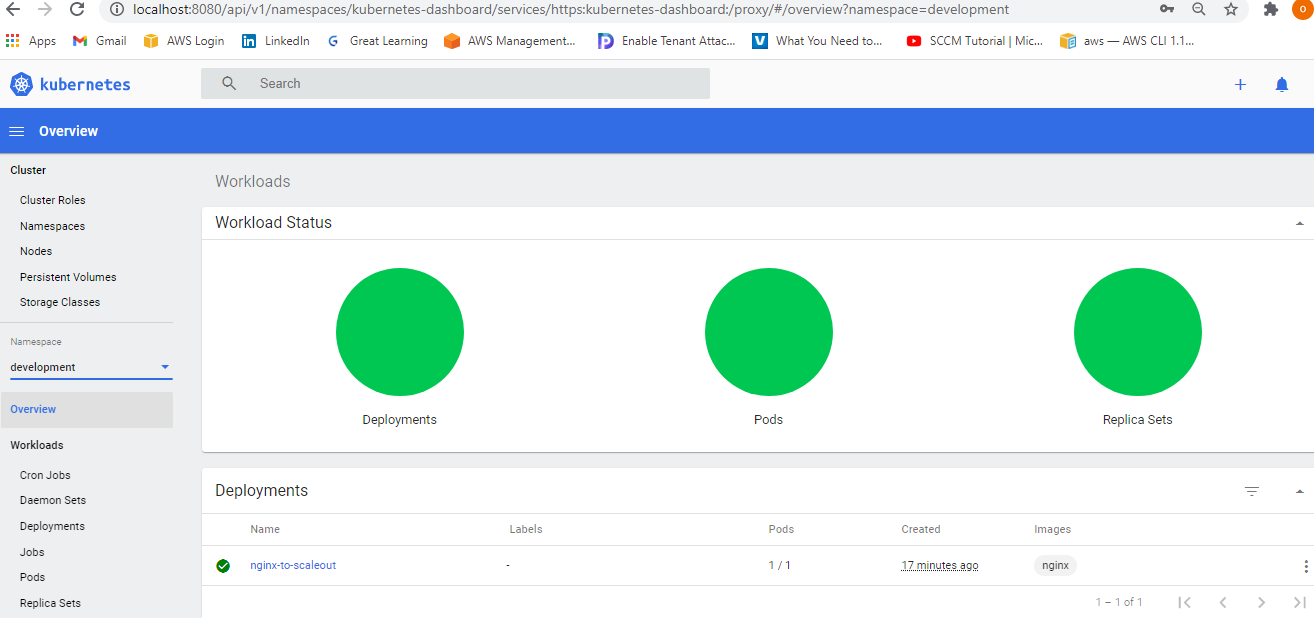
Deployment Status: Successful all green

$ kubectl get pods -l app=nginx -o wide –watch

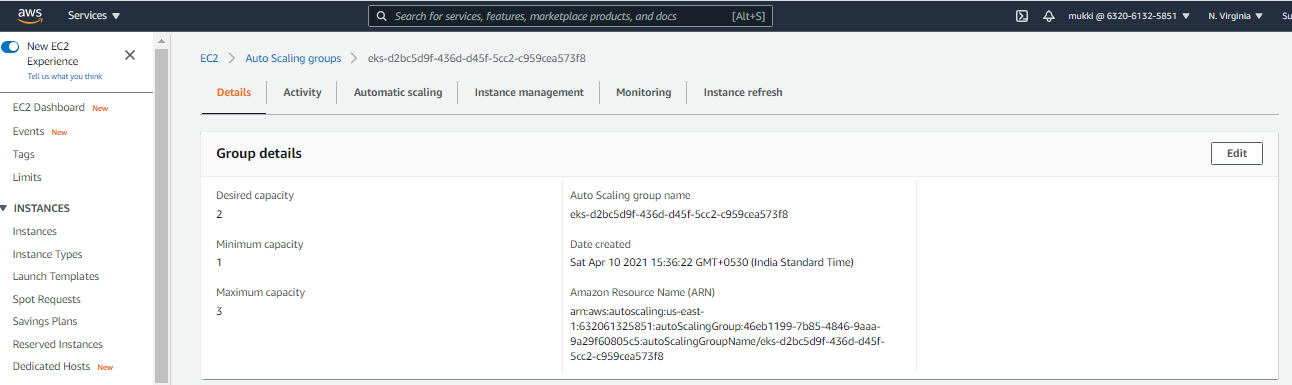
(All the three replica pods are running)



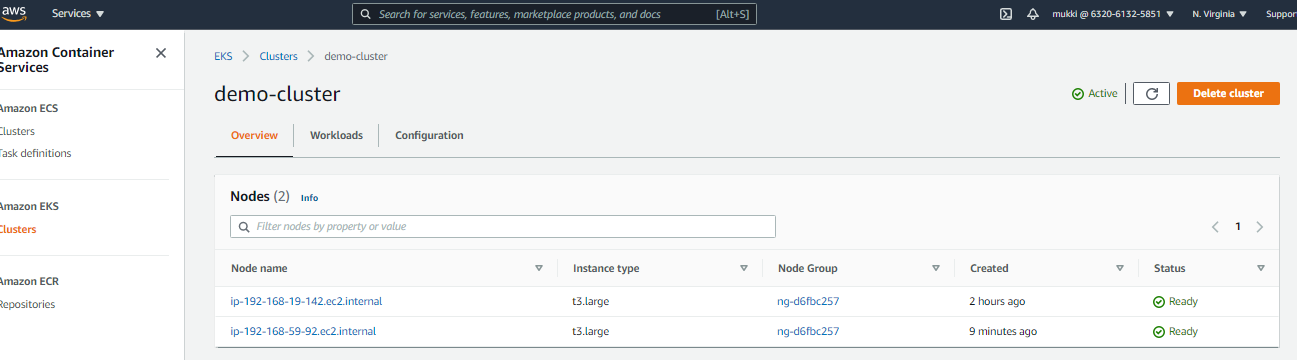
Dashborad Healthy Status: Healthy



Auto Scaling Dashboard status: desired state=2



Cluster Status: 02 Nodes up and ready in line of ASG desired status



**2nd Set-up: Pod Scaling-out to improve Service Performance**

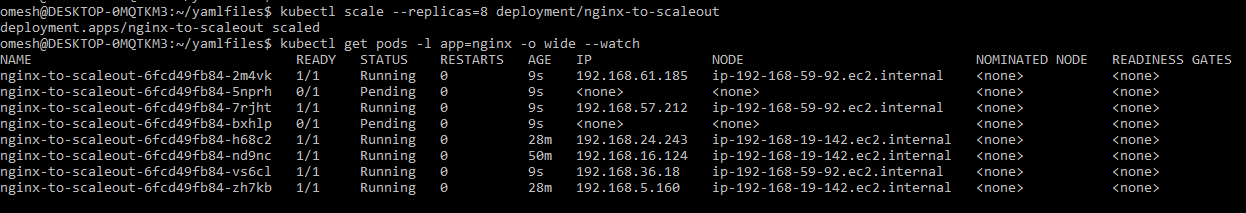
Pods were scaled out to 8 and Cluster Node desired state Unchanged

$ kubectl scale --replicas=8 deployment/nginx-to-scaleout

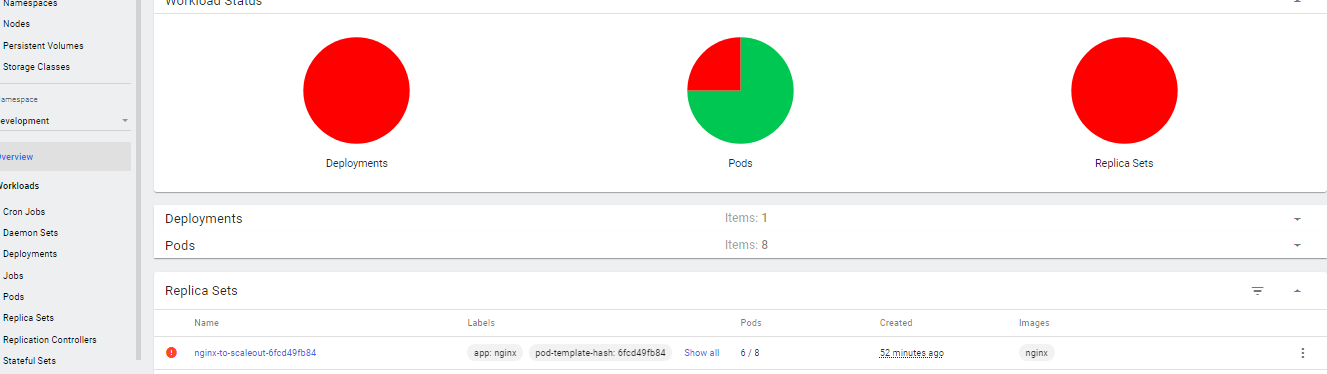
Deployment Status: – Mix (Green and Red)

$ kubectl get pods -l app=nginx -o wide –watch

(06 pods running while 02 stuck in the pending state because cluster does not have enough resources to scale out)



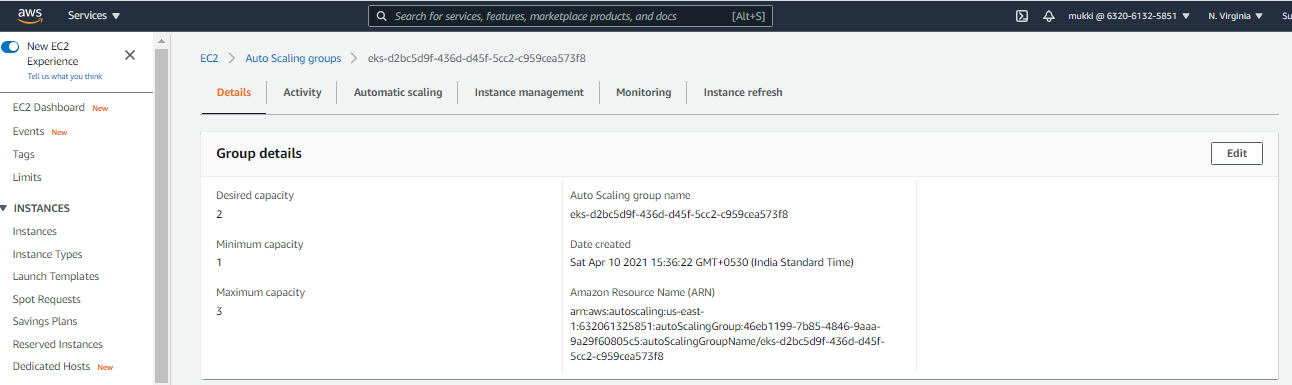
Dashborad is Status: Unheathy



Detailed view of the Dashboard:



Auto Scaling Dashboard status: desired state=2



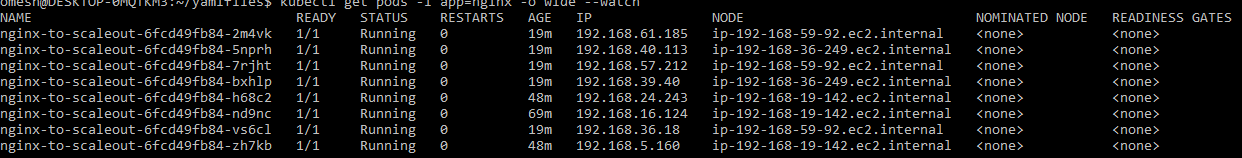
**3rd Set-up: Pod Scaling-out unchanged at 8 while Cluster Scaling changed for the desired state.**

Desired Pods: 8 and Cluster Node desired state change to 3

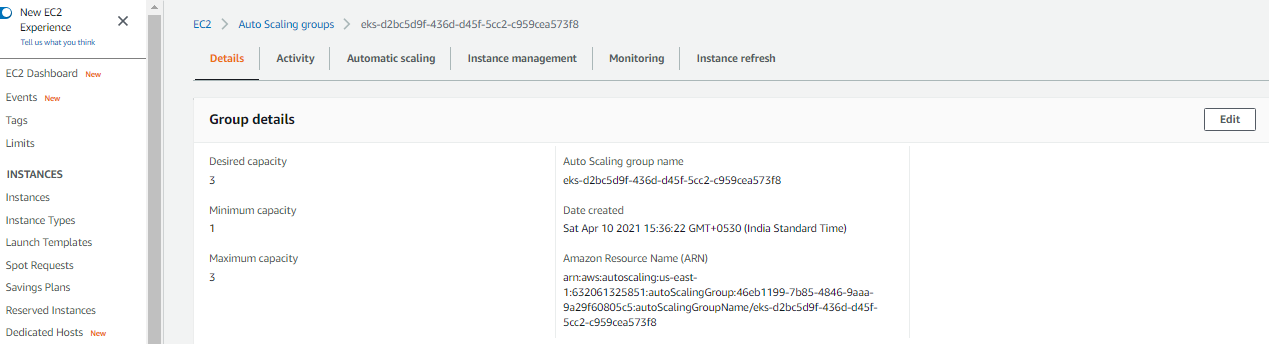
Deployment Status: – Successful all **Green**

$ kubectl get pods -l app=nginx -o wide –watch

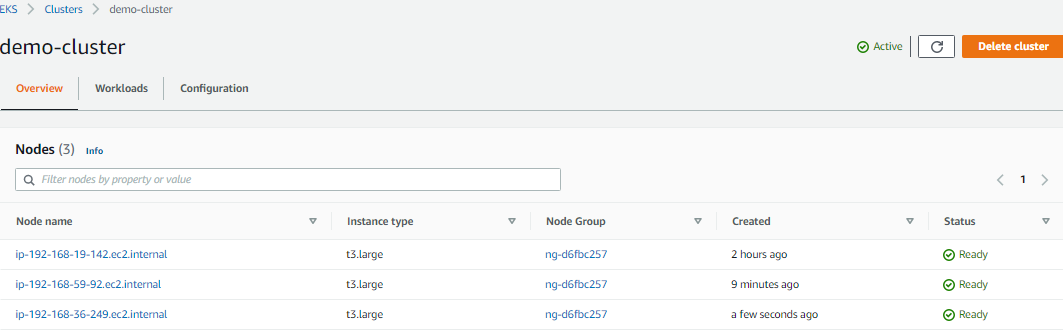
(All the 08 pods are running now)



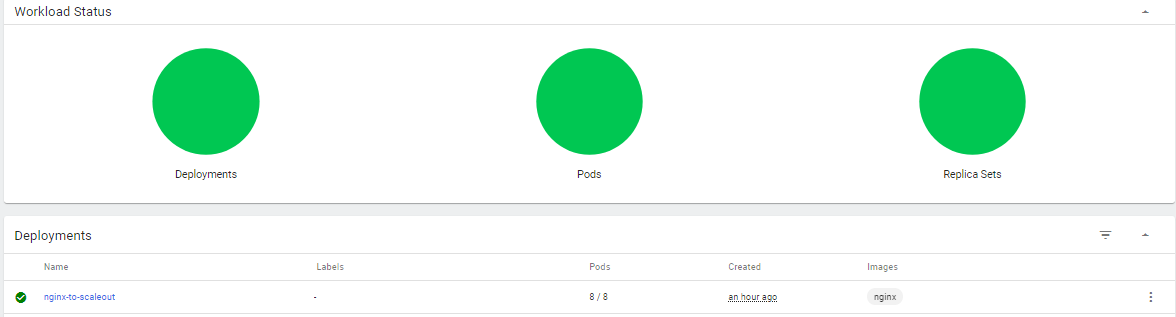
Auto Scaling Dashboard status: desired state updated= 3



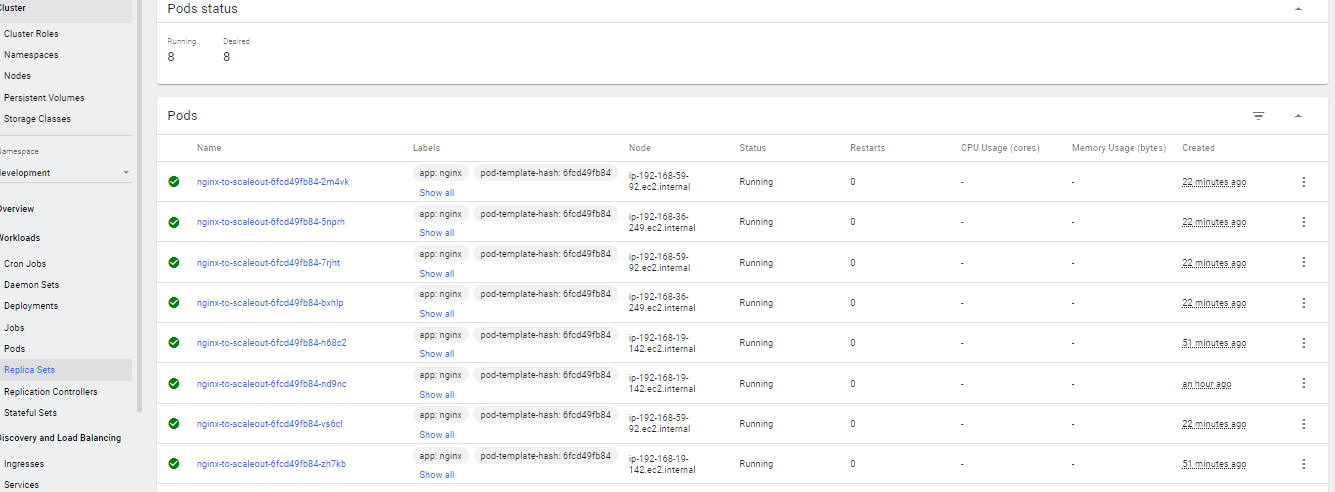
Cluster Status: 03 Nodes up and ready in line of ASG desired status



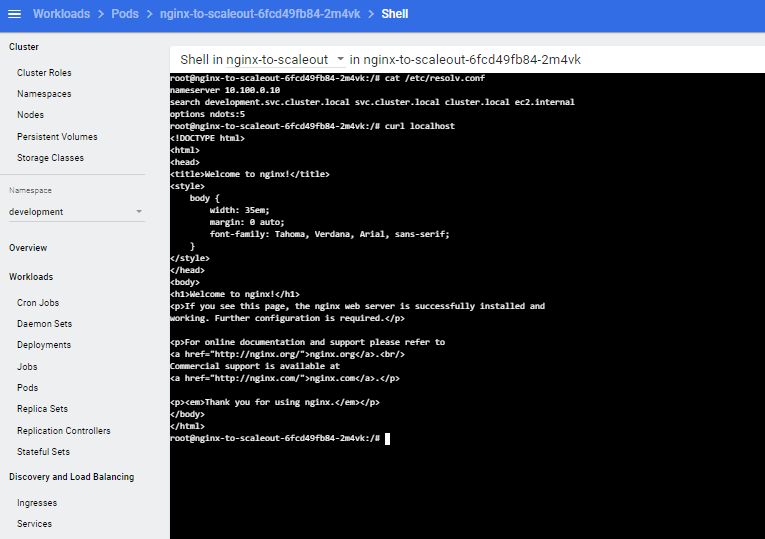
Dashborad Healthy Status: Healthy



Dashboard: Detailed status of the pods



SSH into a running pod to verify the ngnix service is deployed as intended.



**Clean-up**

Delete the Pods

Delete the Service

Delete the Node Group Stack

Delete the Cluster Stack