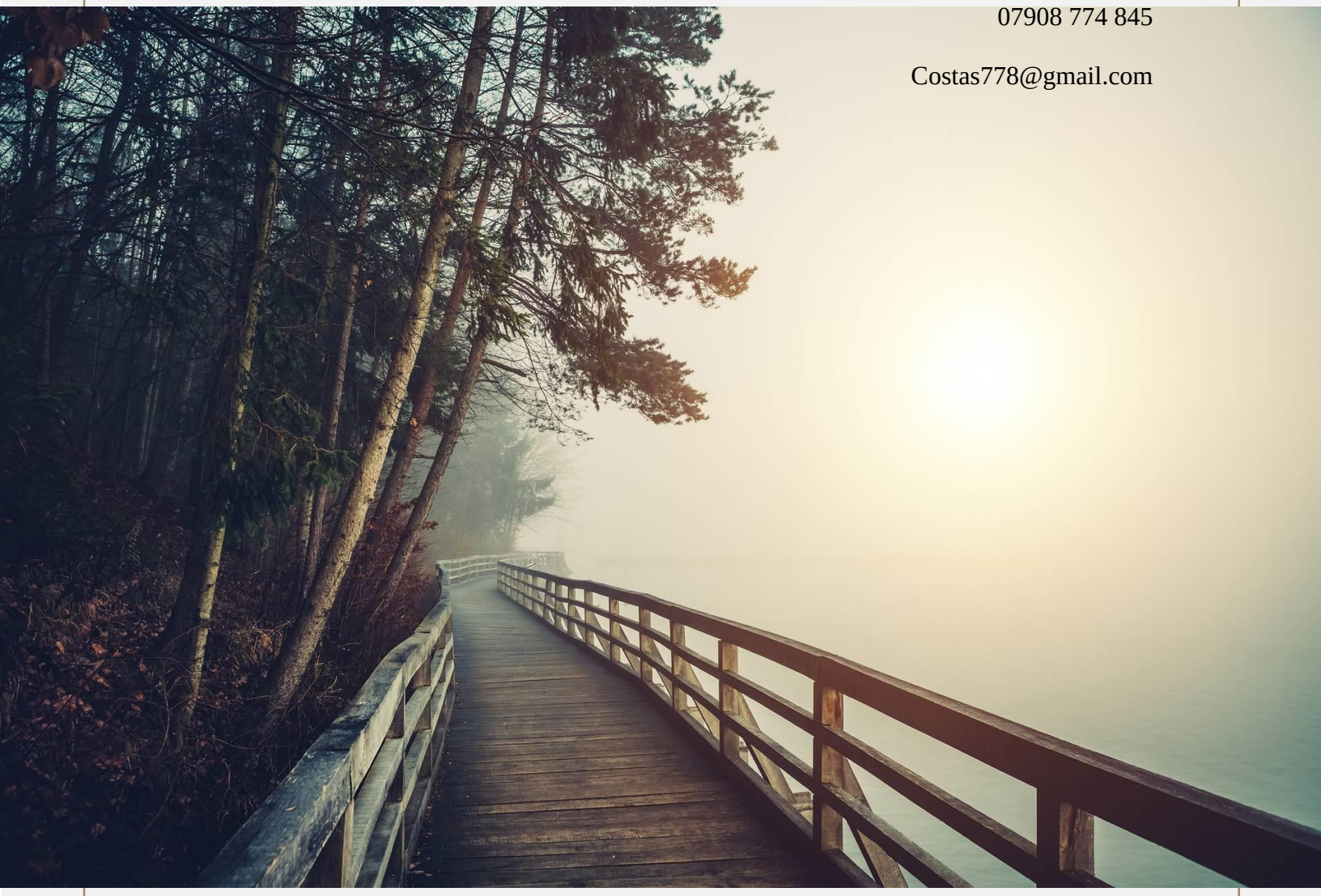
DEVOPS Projects

Costas Constantinou MSc

Version 1.5



07908 774 845

Costas778@gmail.com

DEVOPS PROJECTS

What is this document for?

Having spent over 10 years in the IT Industry, I have noticed that there are, from, time to time, new "flavors of the month". This leads to shortages of skilled professionals. DEVOPS is no exception.

What never changes are the dilemma of sifting through hundreds (and even thousands) of applications claiming they can add value to either your organisation or corporate entity.

HR professionals and recruiters must sift through many applications that have all the 'right words' on their CV or covering letter. However, there are more than a few applicants that slip through that turn out to be "economical with the truth". In other words, they cannot support, either by knowledge or experience, what is written by them in their documents.

Now it is true that many of these people are screened and rejected by a technical test. However, even this is no guarantee that they, genuinely, know their subject matter considering all the resources on the internet to help you prep in various technical disciplines!

As a result, I decided to come up with this DEVOPS projects document that demonstrates, practically, what I can do with the various DEVOPS tools. Within this document you will find various projects I have done. It will be a growing list and one that will be updated as technology never sleeps.

If you are a HR or recruiter simply attach this to my application and pass it on to others that are best placed to make an assessment.

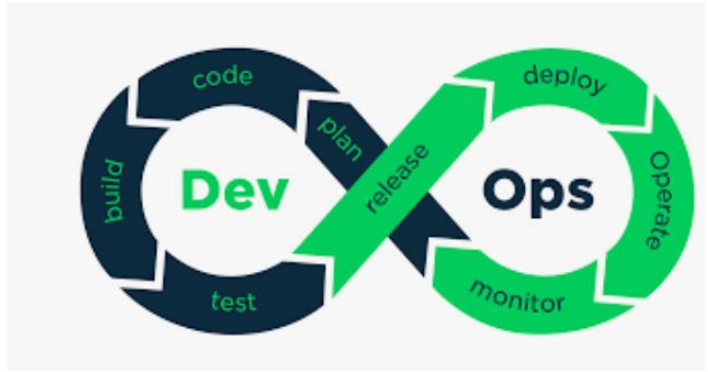
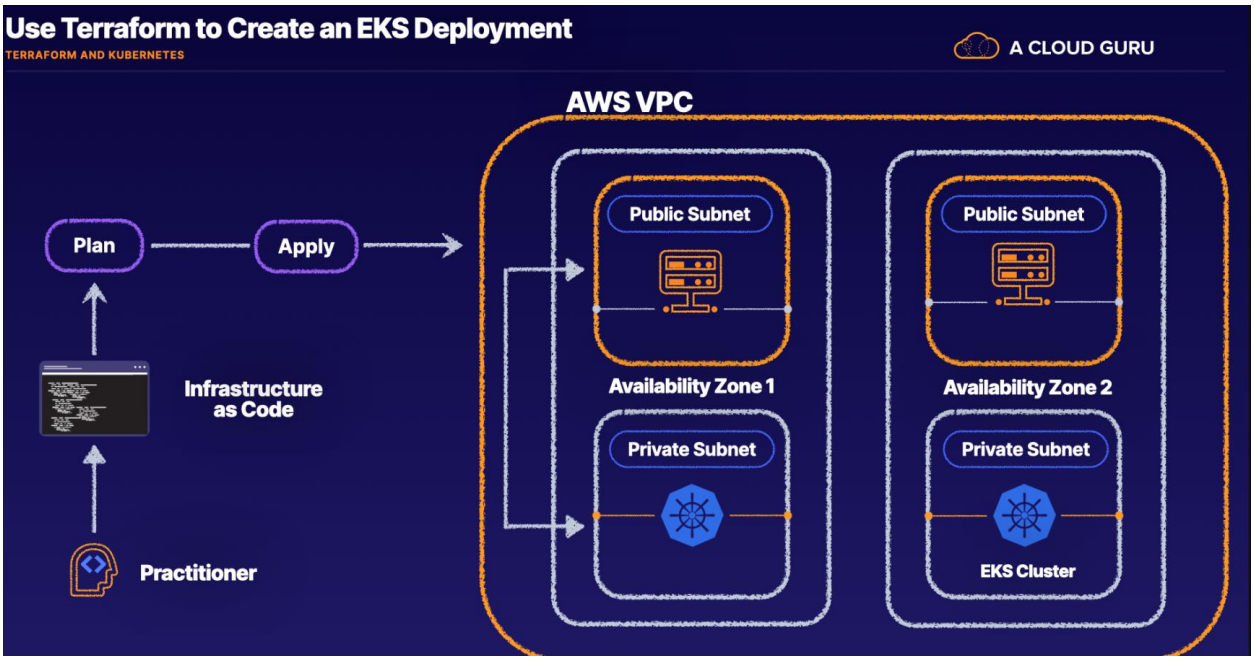


Table of Contents

[EKS Projects] — Use Terraform to Create an EKS Deployment



Solution

Log in to the server using the credentials provided:

```
Host '3.80.152.91' is not in the trusted hosts file.
(ssh-ed25519 fingerprint sha1!! 58:fd:99:ca:9a:f3:7a:6e:2a:32:9b:2c:37:0e:35:8c:63:0c:e8:f2)
Do you want to continue connecting? (y/n) y
cloud_user@3.80.152.91's password:
Last login: Fri Sep 22 09:45:08 2023

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
```

ssh cloud_user@<PUBLIC_IP_ADDRESS>

Configure your AWS CLI:

aws configure

When prompted for your AWS Access Key ID, copy, and paste in the **Access Key**.

When prompted for your AWS Secret Access Key, copy, and paste in the **Secret Access Key**.

Press **Enter** to accept the **default** region.

Press **Enter** to accept the **default** output.

NOTE: if you don't have the above details, go to **IAM,** and generate the above.

```
[cloud_user@ip-10-0-1-201 ~]$ aws configure
AWS Access Key ID [None]: AKIAZXMQHZEGXNLZ67HY
AWS Secret Access Key [None]: 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
Default region name [us-east-1]:
Default output format [None]:
[cloud_user@ip-10-0-1-201 ~]$
```

Review the Contents of the Configuration Files in the lab-terraform-eks Directory:

View the contents of the directory:
Ls

Go into the lab-terraform-eks-2 directory:
cd lab-terraform-eks-2/

View the contents of the lab-terraform-eks-2 directory:
ls

```
[cloud_user@ip-10-0-1-201 ~]$ ls
lab-terraform-eks-2  lab-terraform-eks-2.zip  resource_ids.txt
[cloud_user@ip-10-0-1-201 ~]$ cd lab-terraform-eks-2/
[cloud_user@ip-10-0-1-201 lab-terraform-eks-2]$ ls
main.tf  outputs.tf  terraform.tf  variables.tf
[cloud_user@ip-10-0-1-201 lab-terraform-eks-2]$ vi terraform.tf
```

Open the terraform.tf file:

vim terraform.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.47.0"
    }
    kubernetes = {
      source  = "hashicorp/kubernetes"
      version = ">= 2.16.1"
    }
    random = {
      source  = "hashicorp/random"
      version = "~> 3.4.3"
    }
    tls = {
```

```
    source  = "hashicorp/tls"
    version = "~> 4.0.4"
   }
  }
  required_version = "~> 1.3"
}
```

Review the contents of the file, and then enter **:q!** to exit the file.

Open the main.tf file:

vim main.tf

```
provider "aws" {
  region = var.region
}
data "aws_availability_zones" "available" {}
locals {
  cluster_name = "guru-eks-${random_string.suffix.result}"
}
resource "random_string" "suffix" {
  length  = 8
  special = false
}
module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "3.19.0"
  name = "guru-vpc"
  cidr = "10.0.0.0/16"
  azs  = slice(data.aws_availability_zones.available.names, 0, 3)
  private_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  public_subnets  = ["10.0.4.0/24", "10.0.5.0/24", "10.0.6.0/24"]
  enable_nat_gateway   = true
  single_nat_gateway   = true
  enable_dns_hostnames = true
  public_subnet_tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
    "kubernetes.io/role/elb"                      = 1
  }
  private_subnet_tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
    "kubernetes.io/role/internal-elb"             = 1
```

Review the contents of the file, and then enter **:q!** to exit the file.

Open the variables.tf file:

vim variables.tf

```
variable "region" {
  description = "AWS region"
  type        = string
  default     = "us-east-1"
}
```

Review the contents of the file, and then enter **:q!** to exit the file.

Open the outputs.tf file:

vim outputs.tf

```
output "cluster_endpoint" {
  description = "Endpoint for EKS control plane"
  value       = module.eks.cluster_endpoint
}
output "cluster_security_group_id" {
  description = "Security group ids attached to the cluster control plane"
  value       = module.eks.cluster_security_group_id
}
output "region" {
  description = "AWS region"
  value       = var.region
}
output "cluster_name" {
  description = "Kubernetes Cluster Name"
  value       = module.eks.cluster_name
}
```

Review the contents of the file, and then enter :q! to exit the file.

NOTE: to recreate this Terraform project use the above code to create the files required.

```
[cloud_user@ip-10-0-1-201 ~]$ ls
lab-terraform-eks-2  lab-terraform-eks-2.zip  resource_ids.txt
[cloud_user@ip-10-0-1-201 ~]$ cd lab-terraform-eks-2/
[cloud_user@ip-10-0-1-201 lab-terraform-eks-2]$ ls
main.tf  outputs.tf  terraform.tf  variables.tf
[cloud_user@ip-10-0-1-201 lab-terraform-eks-2]$ vi terraform.tf
[cloud_user@ip-10-0-1-201 lab-terraform-eks-2]$ vi main.tf
[cloud_user@ip-10-0-1-201 lab-terraform-eks-2]$ vi variables.tf
[cloud_user@ip-10-0-1-201 lab-terraform-eks-2]$ vi outputs.tf
[cloud_user@ip-10-0-1-201 lab-terraform-eks-2]$
```

Deploy the EKS Cluster

Initialize the working directory (and download the appropriate plugins):

terraform init

Review the actions that will be performed and check for any potential errors:

```
erraform has been successfully initialized!

ou may now begin working with Terraform. Try running "terraform plan" to see
ny changes that are required for your infrastructure. All Terraform commands
hould now work.

f you ever set or change modules or backend configuration for Terraform,
erun this command to reinitialize your working directory. If you forget, other
ommands will detect it and remind you to do so if necessary.
cloud_user@ip-10-0-1-201 lab-terraform-eks-2]$
```

Check to see if the provisioning plan of what you wish to deploy is acceptable to Terraform with

the following command.

terraform plan

Now apply the configuration and deploy your cluster:

terraform apply

Type **"yes"** and hit **Enter** to confirm. Allow the configuration to complete successfully

and create your resources, including the EKS cluster.

```
aws_eks_addon.ebs-csi: Still creating... [14m40s elapsed]
aws_eks_addon.ebs-csi: Still creating... [14m50s elapsed]
aws_eks_addon.ebs-csi: Still creating... [15m0s elapsed]
aws_eks_addon.ebs-csi: Creation complete after 15m8s [id=guru-eks-jxyiEf3V:aws-ebs-csi-driver]

Apply complete! Resources: 58 added, 0 changed, 0 destroyed.

Outputs:

cluster_endpoint = "https://3830E9EF67E279F402DEEAAD9AC100DA.gr7.us-east-1.eks.amazonaws.com"
cluster_name = "guru-eks-jxyiEf3V"
cluster_security_group_id = "sg-0657edeb6b12b8238"
region = "us-east-1"
[cloud_user@ip-10-0-1-201 lab-terraform-eks-2]$
```

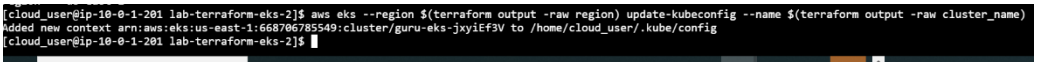
Note: This can take around 10 minutes or more to complete so please be patient.

The **Output.tf** gives us the following:

cluster_endpoint = "https://3830E9EF67E279F402DEEAAD9AC100DA.gr7.us-east-1.eks.amazonaws.com"
cluster_name = "guru-eks-jxyiEf3V"
cluster_security_group_id = "sg-0657edeb6b12b8238"
region = "us-east-1"
[cloud_user@ip-10-0-1-201 lab-terraform-eks-2]$

Configure kubectl to interact with the cluster:

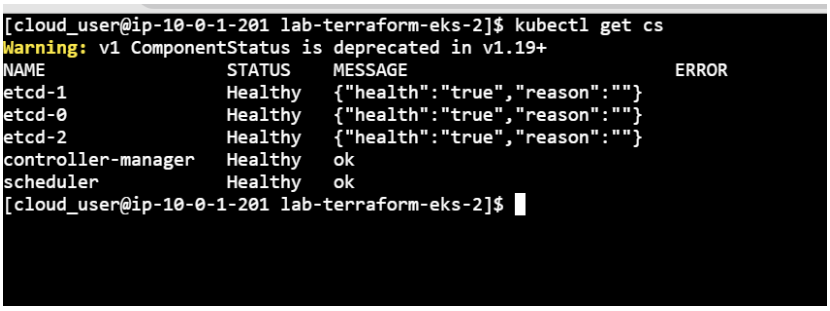
aws eks --region $(terraform output -raw region) update-kubeconfig --name $(terraform output -raw cluster_name)



Confirm that kubectl was configured properly and that the cluster was successfully deployed:

kubectl get cs

All the components should be up and running with a status of Healthy as below.



Deploy the NGINX Pods

Now download the **nginx.tf** file that contains the configuration for the

NGINX deployments from the GitHub repo:

wget https://raw.githubusercontent.com/ACloudGuru-Resources/content-terraform-2021/main/nginx.tf

Confirm that the file was downloaded to the directory successfully:
ls

Open the nginx.tf file:
vim nginx.tf

```
data "terraform_remote_state" "eks" {
  backend = "local"

  config = {
    path = "../lab-terraform-eks-2/terraform.tfstate"
  }
}
```

```
data "aws_eks_cluster" "cluster" {
  name = data.terraform_remote_state.eks.outputs.cluster_name
}

provider "kubernetes" {
  host                   = data.aws_eks_cluster.cluster.endpoint
  cluster_ca_certificate = base64decode(data.aws_eks_cluster.cluster.certificate_authority.0.data)
  exec {
    api_version = "client.authentication.k8s.io/v1beta1"
    command     = "aws"
    args = [
      "eks",
      "get-token",
      "--cluster-name",
      data.aws_eks_cluster.cluster.name
    ]
  }
}

resource "kubernetes_deployment" "nginx" {
  metadata {
    name = "long-live-the-bat"
    labels = {
      App = "LongLiveTheBat"
    }
  }

  spec {
    replicas = 2
    selector {
      match_labels = {
        App = "LongLiveTheBat"
      }
    }
    template {
      metadata {
        labels = {
          App = "LongLiveTheBat"
        }
      }
      spec {
        container {
          image = "nginx:1.7.8"
          name  = "batman"

          port {
            container_port = 80
          }
```

```
        resources {
          limits = {
            cpu    = "0.5"
            memory = "512Mi"
          }
          requests = {
            cpu    = "250m"
            memory = "50Mi"
          }
        }
      }
    }
  }
}
```

Review the contents of the file, and then enter :q! to exit the file.

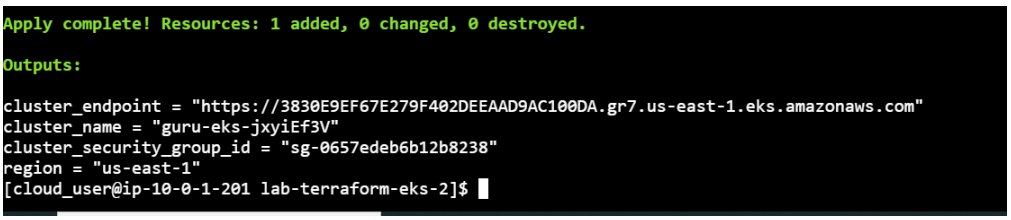
Review the actions that will be performed and check for any potential errors:

Since we are updating the infrastructure, we need to run the same came commands again.
terraform plan

Apply the configuration and deploy your NGINX pods to your EKS cluster:

terraform apply

Type **"yes"** and hit Enter to confirm. Allow the configuration to complete successfully and create your resources, including the NGINX pods.



Confirm that the NGINX pods were successfully deployed:

kubectl get deployments

The two long-live-the-bat pods should both be up and running.

```
[cloud_user@ip-10-0-1-201 lab-terraform-eks-2]$ kubectl get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
long-live-the-bat   2/2     2            2           111s
[cloud_user@ip-10-0-1-201 lab-terraform-eks-2]$
```

Destroy Your Cluster

Before you destroy the infrastructure, view the resources Terraform is managing:

terraform state list

A whole list of resources will appear.

Destroy all of the resources you just created:

terraform destroy

Type **"yes"** and hit Enter to confirm.

```
random_string.suffix: Destroying... [id=jXyIEf3V]
random_string.suffix: Destruction complete after 0s
module.vpc.aws_vpc.this[0]: Destroying... [id=vpc-0
module.vpc.aws_vpc.this[0]: Destruction complete af

Destroy complete! Resources: 59 destroyed.
[cloud_user@ip-10-0-1-201 lab-terraform-eks-2]$
```

Confirm the resources were all destroyed:

terraform state list

Nothing should be returned, confirming that all resources were properly destroyed.

Addendum

To recreate the project we went through we will need to take the following pre requisite steps beforehand.

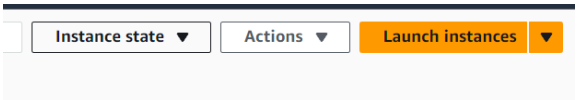
Create and connect to a new instance on AWS (hence we will need an AWS account).
Create an Admin User Role within IAM
Upgrade AWS CLI to version 2 (for AMI distros)
Install the AWS CLI into that directory
Install and configure Terraform on an EC2 instance
Install Kubectl

Create and Connect to New Instance

In the search bar on top, enter **EC2** and select EC2 from the search results to access EC2.

Select **Instances (running).**

Click the **Launch instances** button.



On the Launch an instance button, set the following parameters:

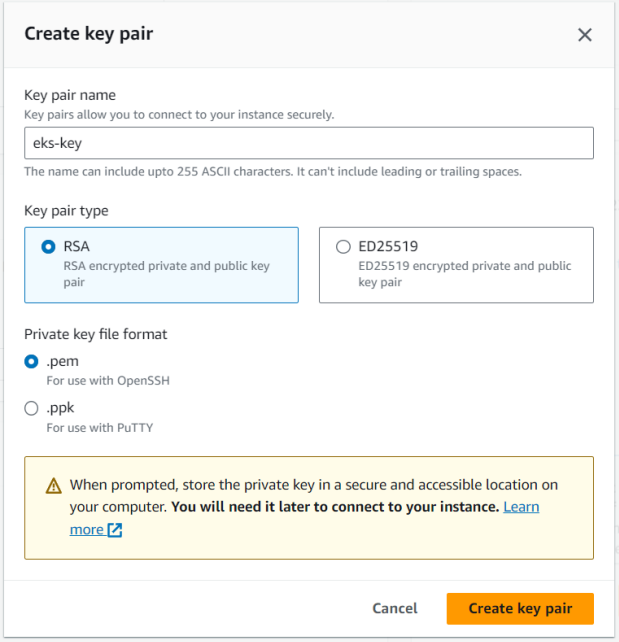
Name and tags: Enter **eks-admin**.

Application and OS Images (Amazon Machine Image): Select **Amazon Linux**.

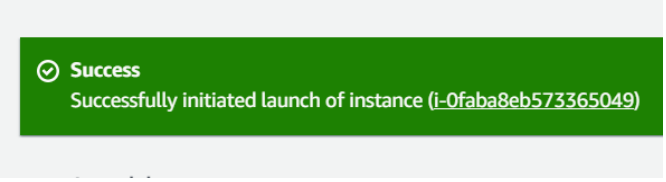
Key pair name: Select **Create new key pair**.

On the Create key pair page, under Key pair name, enter **eks-key**.

Click the **Create key pair** button to create the key pair.

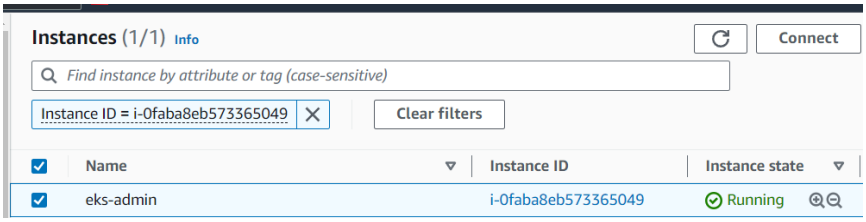


Click the **Launch instance** button.
Under Success, select the link to the instance to see its status. It can take a little while for the instance to launch successfully and for the Instance state to change from Pending to Running.

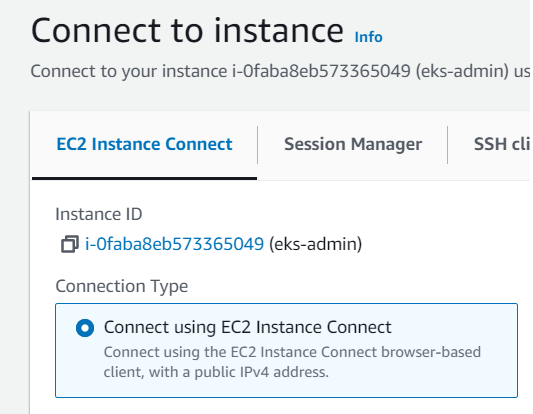


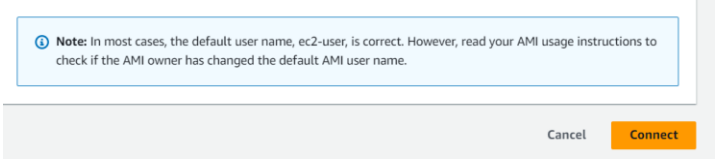
Click the checkbox next to the eks-admin instance to select it.

Click the **Connect** button.



On the Connect to instance page, click the **Connect** button.





Create an Admin User Role within IAM

Under Favourites, select **IAM**. If you don't see a Favourites section, you can enter **IAM** in the search bar on top of the console and select *IAM* from the search results to access the **IAM** dashboard.

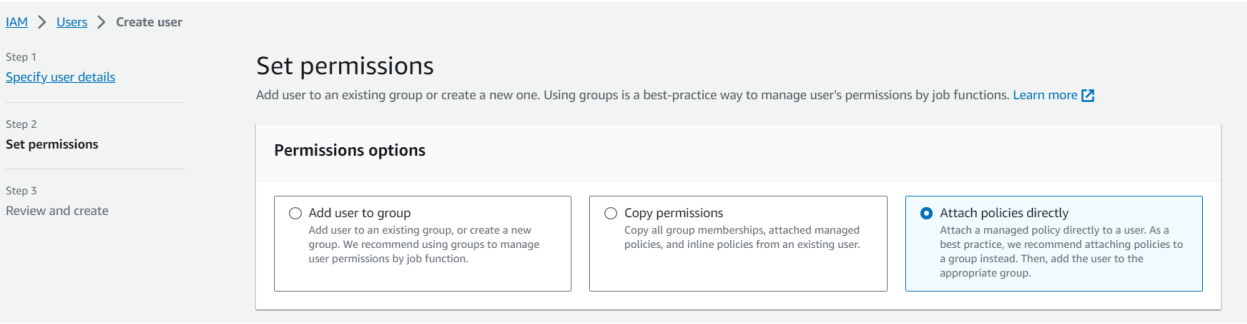
In the lefthand navigation menu, select **Users**.

Click the **Add Users** button.

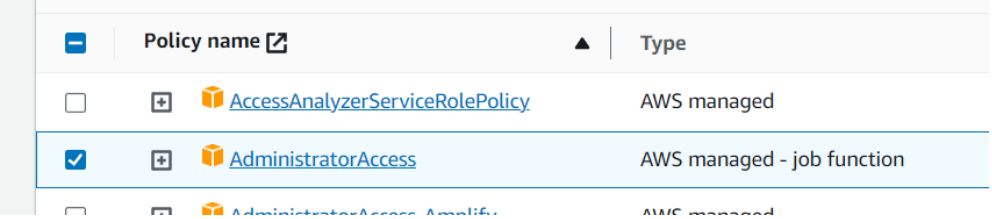
On the Specify user details page, under Username, enter **eks-user**.

Click the **Next** button.

On the Set permissions page, under Permissions options, select **Attach policies directly**.



Under Permissions policies, click the checkbox next to **AdminstratorAccess** to grant admin access to the role being created.



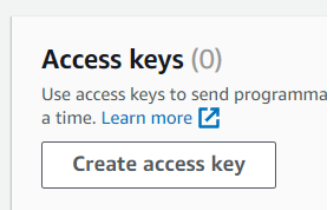
Click the **Next** button.

On the Review and create page, click the **Create user** button.

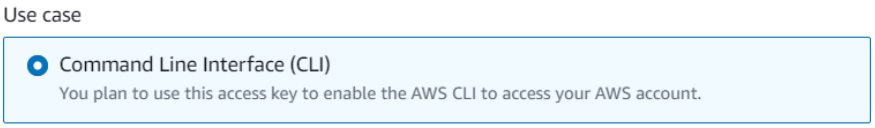
Once the user is created, select **eks-user**.

Select the **Security credentials** tab.

Under **Access keys**, click the **Create access key** button.



On the Access key best practices & alternatives page, select **Command Line Interface (CLI).**



Click the checkbox next to **I understand the recommendation and want to proceed to create an access key**.

Click the **Next** button.

On the Set description tag - optional page, click the **Create access key** button.

Click the **Download .csv** file.

Upgrade AWS CLI to version 2

Download AWS CLI 2:

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

Unzip the downloaded CLI 2 file:

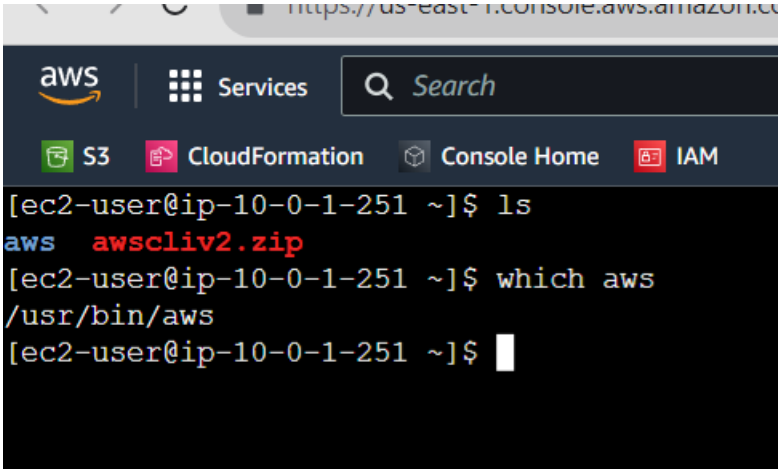
unzip awscliv2.zip

This can take a few minutes to unzip and create everything needed.

Check the location of our AWS files:

which aws

You should see they are located in **usr/bin/aws**.



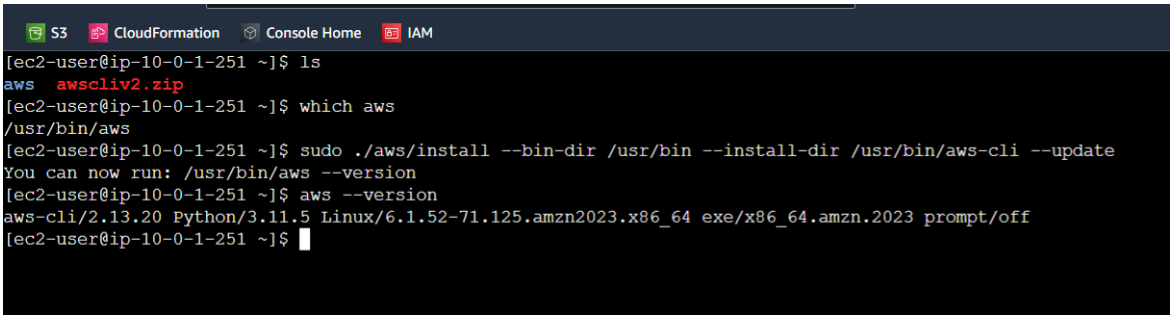
Install the AWS CLI into that directory:

sudo ./aws/install --bin-dir /usr/bin --install-dir /usr/bin/aws-cli --update

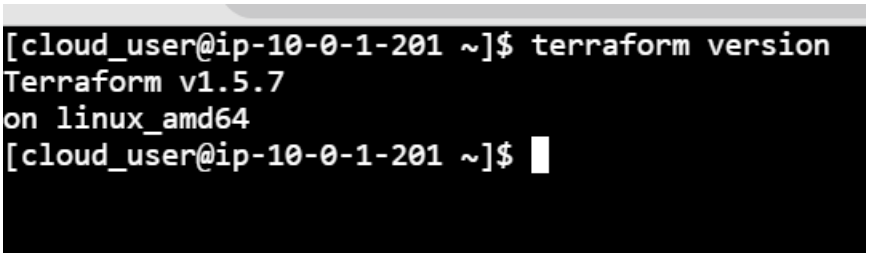
Re-check which version of AWS is being used:

aws --version

You should see we are now using AWS CLI 2.11 or higher



Install and configure Terraform on an EC2 instance



sudo yum update -y

wget https://releases.hashicorp.com/terraform/1.5.7/terraform_1.5.7_linux_amd64.zip

unzip terraform_1.5.7_linux_amd64.zip

sudo mv terraform /usr/local/bin/

terraform --version

rm terraform_1.5.7_linux_amd64.zip

Install Kubectl

curl -LO "https://dl.k8s.io/release/v1.21.2/bin/linux/amd64/kubectl"

chmod +x kubectl

sudo mv kubectl /usr/local/bin/

kubectl version --client

rm kubectl

[cloud_user@ip-10-0-1-201 ~]$ **kubectl version**

Client Version: version.Info{Major:"1", Minor:"21+", GitVersion:"v1.21.2-13+d2965f0db10712",

GitCommit:"d2965f0db1071203c6f5bc662c2827c71fc8b20d", GitTreeState:"clean", BuildDate:"2021-06-

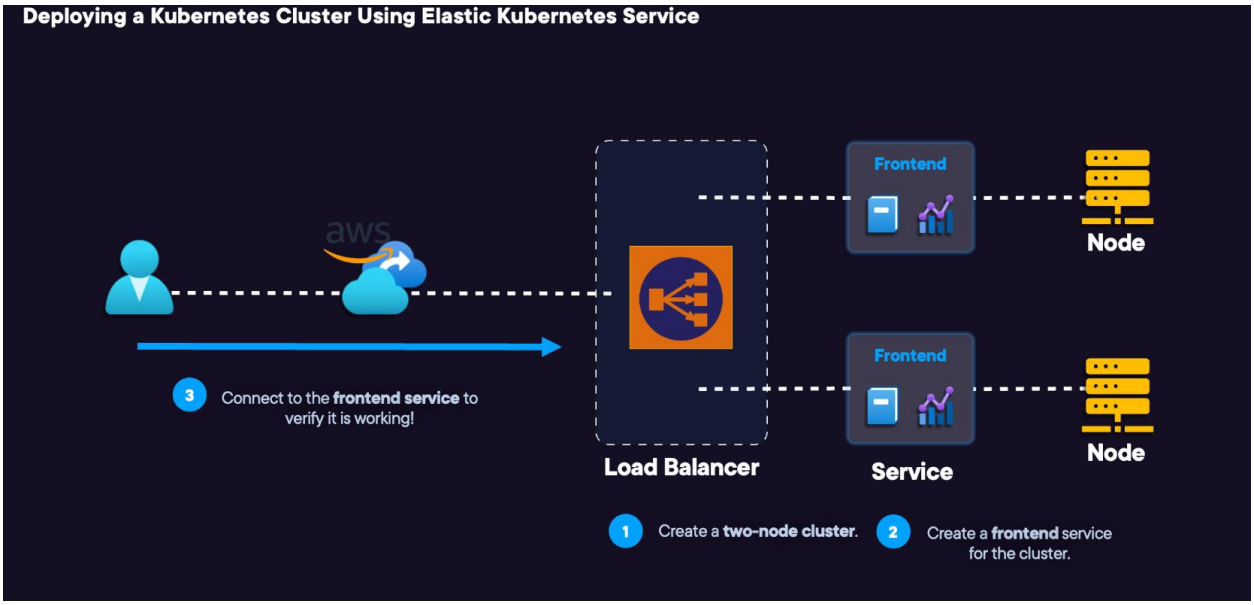
26T01:02:11Z", GoVersion:"go1.16.5", Compiler:"gc", Platform:"linux/amd64"}

The connection to the server localhost:8080 was refused -

did you specify the right host or port?

```
[cloud_user@ip-10-0-1-201 ~]$
```

Deploying a Kubernetes Cluster Using Elastic Kubernetes Service



Solution

Log in to the AWS Management Console.

Note: you will need to have a AWS account prior to starting.

Note: Make sure you're using the us-east-1 Region.

Create an EKS Cluster in the us-east-1 Region

Create an Admin User Role:

Under Favourites, select **IAM**. If you don't see a Favourites section, you can enter **IAM** in the search bar on top of the console and select **IAM** from the search results to access the **IAM** dashboard.

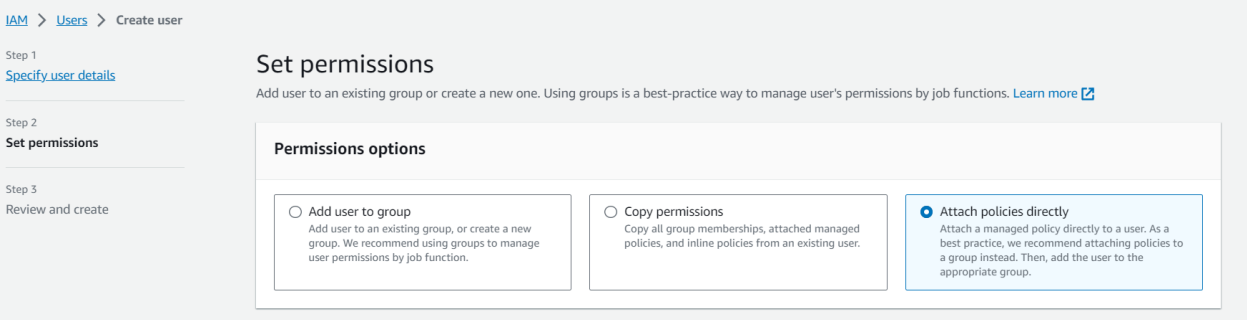
In the lefthand navigation menu, select **Users**.

Click the **Add Users** button.

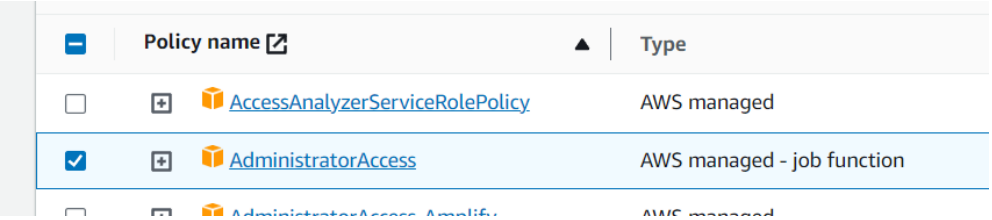
On the Specify user details page, under Username, enter **eks-user**.

Click the **Next** button.

On the **Set permissions** page, under Permissions options, select **Attach policies directly**.



Under Permissions policies, click the checkbox next to **AdminstratorAccess** to grant admin access to the role being created.



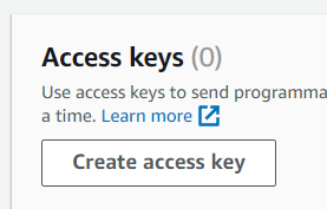
Click the **Next** button.

On the Review and create page, click the **Create user** button.

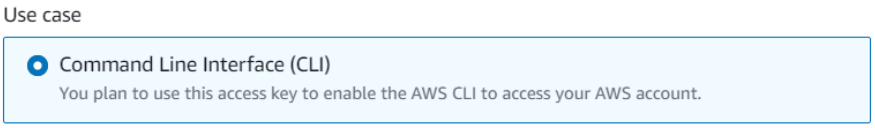
Once the user is created, select **eks-user**.

Select the **Security credentials** tab.

Under **Access keys**, click the **Create access key** button.



On the Access key best practices & alternatives page, select **Command Line Interface (CLI).**



Click the checkbox next to **I understand the recommendation and want to proceed to create an access key**.

Click the **Next** button.

On the Set description tag - optional page, click the **Create access key** button.

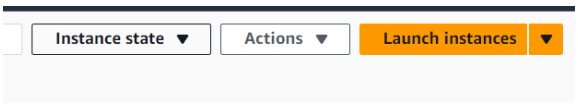
Click the **Download .csv** file.

Create and Connect to New Instance

In the search bar on top, enter **EC2** and select **EC2** from the search results to access EC2.

Select **Instances (running).**

Click the **Launch instances** button.



On the Launch an instance button, set the following parameters:

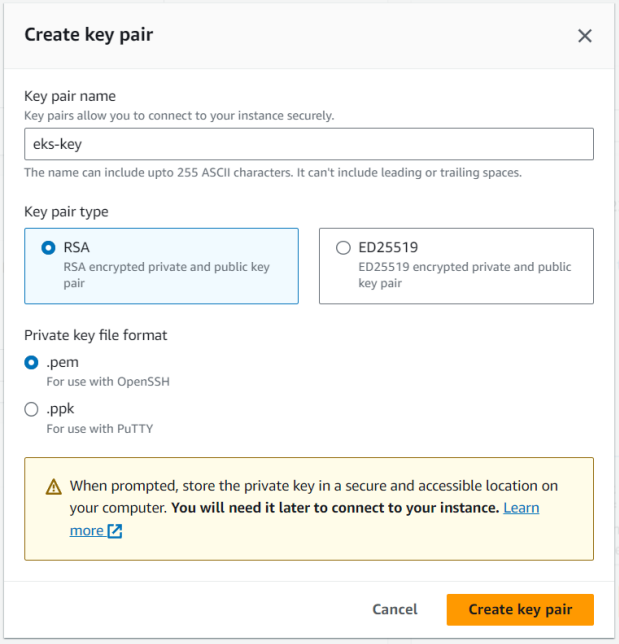
Name and tags: Enter **eks-admin**.

Application and OS Images (Amazon Machine Image): Select **Amazon Linux**.

Key pair name: Select **Create new key pair**.

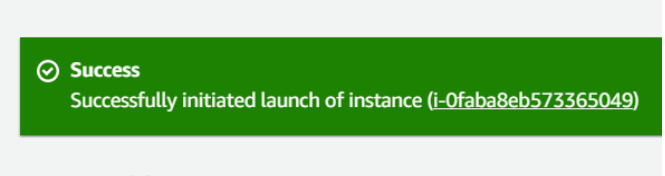
On the Create key pair page, under Key pair name, enter **eks-key**.

Click the **Create key pair** button to create the key pair.



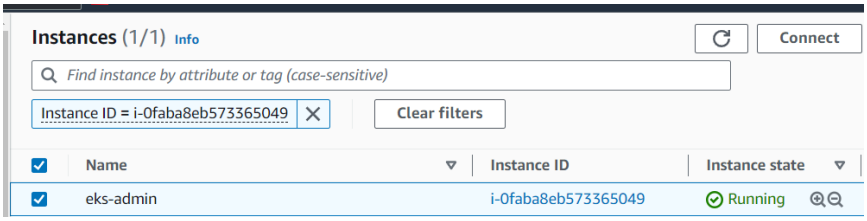
Click the **Launch instance** button.

Under **Success**, select the link to the instance to see its status. It can take a little while for the instance to launch successfully and for the Instance state to change from Pending to Running.

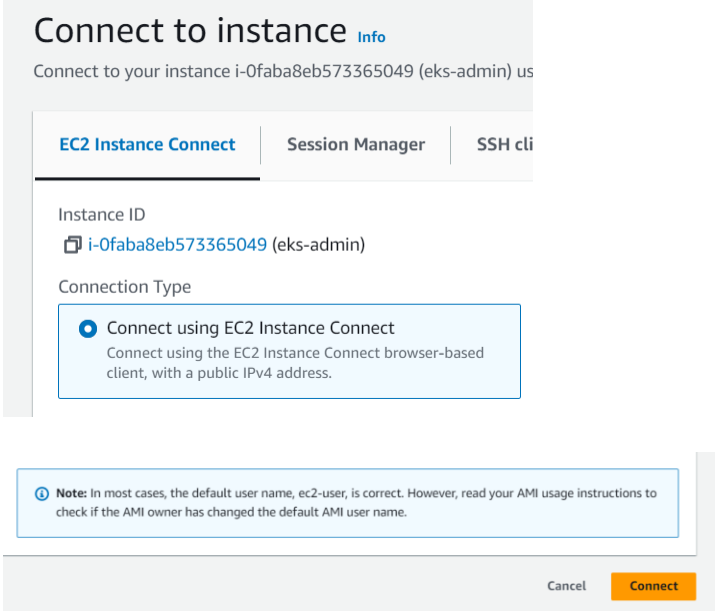


Click the checkbox next to the eks-admin instance to select it.

Click the **Connect** button.



On the Connect to instance page, click the **Connect** button.

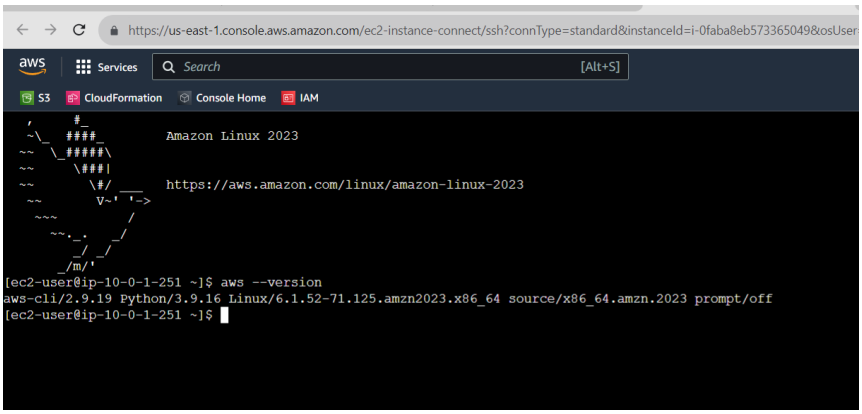


Create Cluster

Check the version of the CLI being used to verify whether it is NOT AWS CLI 2:

aws --version

You should see that the version being used is AWS CLI 1.18 or above.



For this project we will need to upgrade our AWS CLI to version 2.

Note: if AWS CLI does NOT exist at all then follow these steps as well.

Download AWS CLI 2:

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

Unzip the downloaded CLI 2 file:

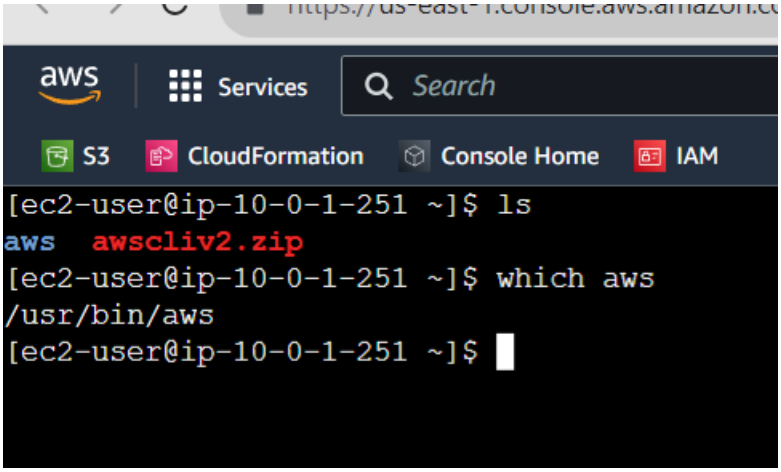
unzip awscliv2.zip

This can take a few minutes to unzip and create everything needed.

Check the location of our AWS files:

which aws

You should see they are located in **usr/bin/aws**.



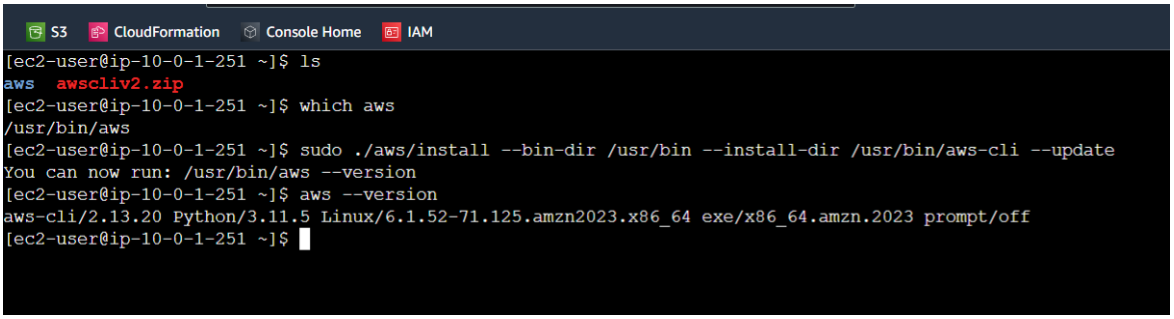
Install the AWS CLI into that directory:

sudo ./aws/install --bin-dir /usr/bin --install-dir /usr/bin/aws-cli --update

Re-check which version of AWS is being used:

aws --version

You should see we are now using AWS CLI 2.11 or above



Log into our AWS environment:

aws configure

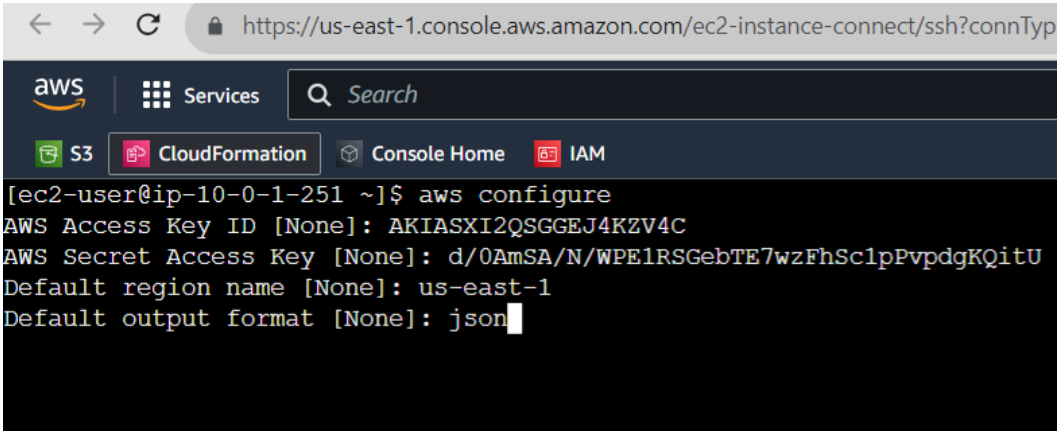
From the CSV file created earlier, copy and paste the **Access Key ID** when prompted.

From the same CSV file, copy and paste the **Secret Access Key** when prompted

	A	B	C	D	E	F	G
1	Access key ID	Secret access key					
2	AKIASXI2QSGGEJ4KZV4C	d/0AmSA/N/WPE1RSGebTE7wzFhSc1pPvpdgKQitU					

When prompted for the default region name, enter **us-east-1**.

When prompted for the default output format, enter **json**.



Note: In a few hours this infrastructure will be gone with the key pair credentials!

Download kubectl:

curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.25.6/2023-01-30/bin/linux/amd64/kubectl

Add execution permissions to the file:

chmod +x ./kubectl

Create a new bin directory and copy kubectl to the bin directory and home directory:

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

Download eksctl:

curl --silent --location

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

This will install the CLI tool you will use to create a cluster on Amazon EKS.

Move the files to usr/local/bin:

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

Check the eksctl version:

eksctl version

This should show the version as 0.133.0 or higher!

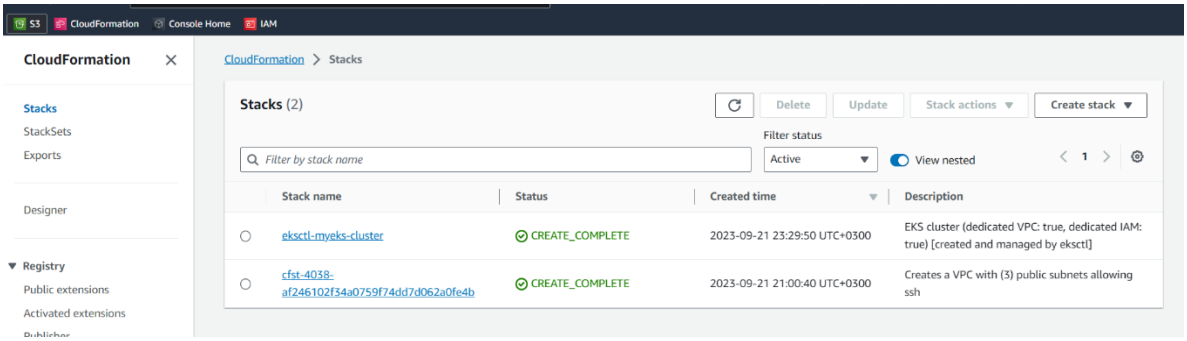
```
[ec2-user@ip-10-0-1-251 ~]$ curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.25.6/2023-01-30/bin/linux/amd64/kubectl
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 42.9M  100 42.9M    0     0  17.2M      0  0:00:02  0:00:02 --:--:-- 17.2M
[ec2-user@ip-10-0-1-251 ~]$ chmod +x ./kubectl
[ec2-user@ip-10-0-1-251 ~]$ mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$PATH:$HOME/bin
[ec2-user@ip-10-0-1-251 ~]$ curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
[ec2-user@ip-10-0-1-251 ~]$ sudo mv /tmp/eksctl /usr/local/bin
[ec2-user@ip-10-0-1-251 ~]$ eksctl version
0.157.0
```

Create the cluster:

eksctl create cluster --name myeks --region us-east-1 --zones=us-east-1a,us-east-1b --nodegroup-name eks-workers --node-type t3.medium --nodes 2 --nodes-min 2 --nodes-max 4 --managed

This may take several minutes to finish creating the cluster.

```
[ec2-user@ip-10-0-1-251 ~]$ eksctl create cluster --name myeks --region us-east-1 --zones=us-east-1a,us-east-1b --nodegroup-name eks-workers --node-type t3.medium --nodes
 --nodes-min 2 --nodes-max 4 --managed
2023-09-21 20:29:49 [ℹ]  eksctl version 0.157.0
2023-09-21 20:29:49 [ℹ]  using region us-east-1
2023-09-21 20:29:50 [ℹ]  subnets for us-east-1a - public:192.168.0.0/19 private:192.168.64.0/19
2023-09-21 20:29:50 [ℹ]  subnets for us-east-1b - public:192.168.32.0/19 private:192.168.96.0/19
2023-09-21 20:29:50 [ℹ]  nodegroup "eks-workers" will use "" [AmazonLinux2/1.25]
2023-09-21 20:29:50 [ℹ]  using Kubernetes version 1.25
2023-09-21 20:29:50 [ℹ]  creating EKS cluster "myeks" in "us-east-1" region with managed nodes
2023-09-21 20:29:50 [ℹ]  will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup
2023-09-21 20:29:50 [ℹ]  if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=us-east-1 --cluster=myeks'
2023-09-21 20:29:50 [ℹ]  Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "myeks" in "us-east-1"
2023-09-21 20:29:50 [ℹ]  CloudWatch logging will not be enabled for cluster "myeks" in "us-east-1"
2023-09-21 20:29:50 [ℹ]  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=
eks'
2023-09-21 20:29:50 [ℹ]
2 sequential tasks: { create cluster control plane "myeks",
    2 sequential sub-tasks: {
        wait for control plane to become ready,
        create managed nodegroup "eks-workers",
    }
}
2023-09-21 20:29:50 [ℹ]  building cluster stack "eksctl-myeks-cluster"
2023-09-21 20:29:50 [ℹ]  deploying stack "eksctl-myeks-cluster"
```



Once the cluster is created, connect to the cluster:

aws eks update-kubeconfig --name myeks --region us-east-1

Verify that it is running eksctl:

eksctl get cluster

You should see that it is running.

Deploy a Test Application to Mimic the Application

Install the Git package:

sudo yum install git -y

Clone a repository from the EKS Basics course:

git clone https://github.com/costas778/Course_EKS-Basics

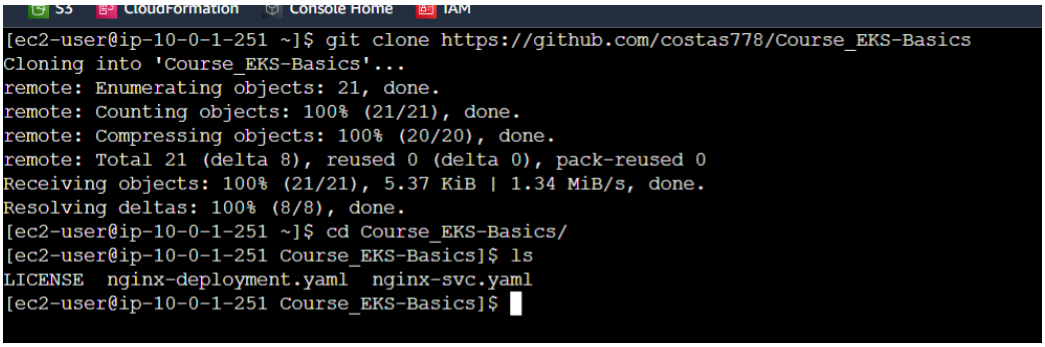
Change into the Course_EKS-Basics directory:

cd Course_EKS-Basics/

View the contents of the directory:

ls

You should see a license file, an nginx-deployment.yaml file, and an nginx-svc.yaml file.



View the nginx-deployment.yaml file:

cat nginx-deployment.yaml

View the nginx-svc.yaml file:

cat nginx-svc.yaml

Apply the nginx-svc.yaml file:

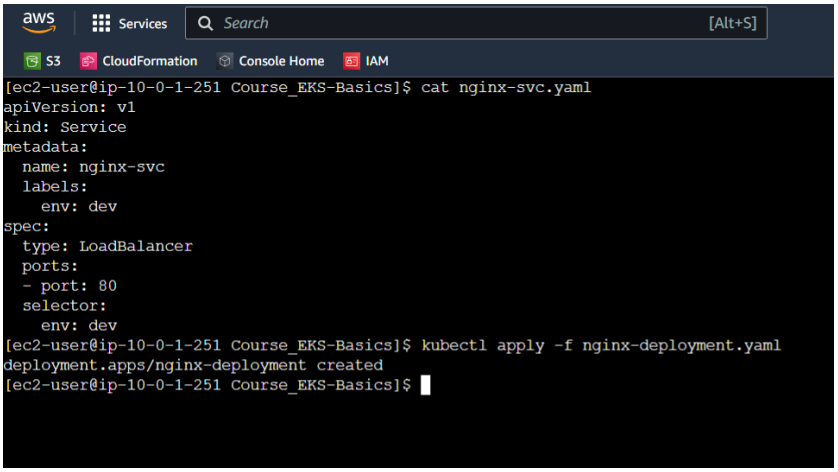
kubectl apply -f nginx-svc.yaml

This should create the Nginx service.

Apply the nginx-deployment.yaml file:

kubectl apply -f nginx-deployment.yaml

This should create the Nginx deployments.



Verify that the deployments are available:

kubectl get deployment

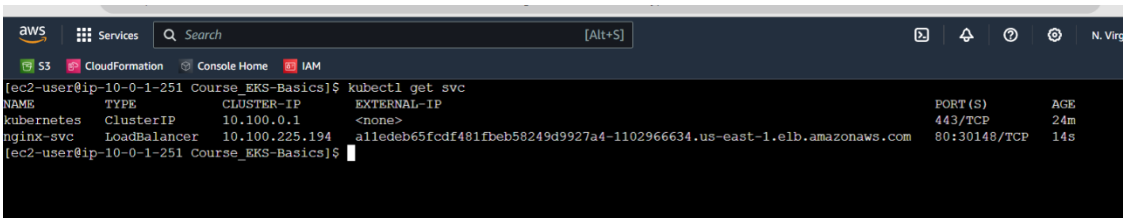
You should see that all three deployments are available.

Verify that all Pods are running:

kubectl get pods

Verify that the service is running:

kubectl get svc



You should see that the load balancer has a DNS name.

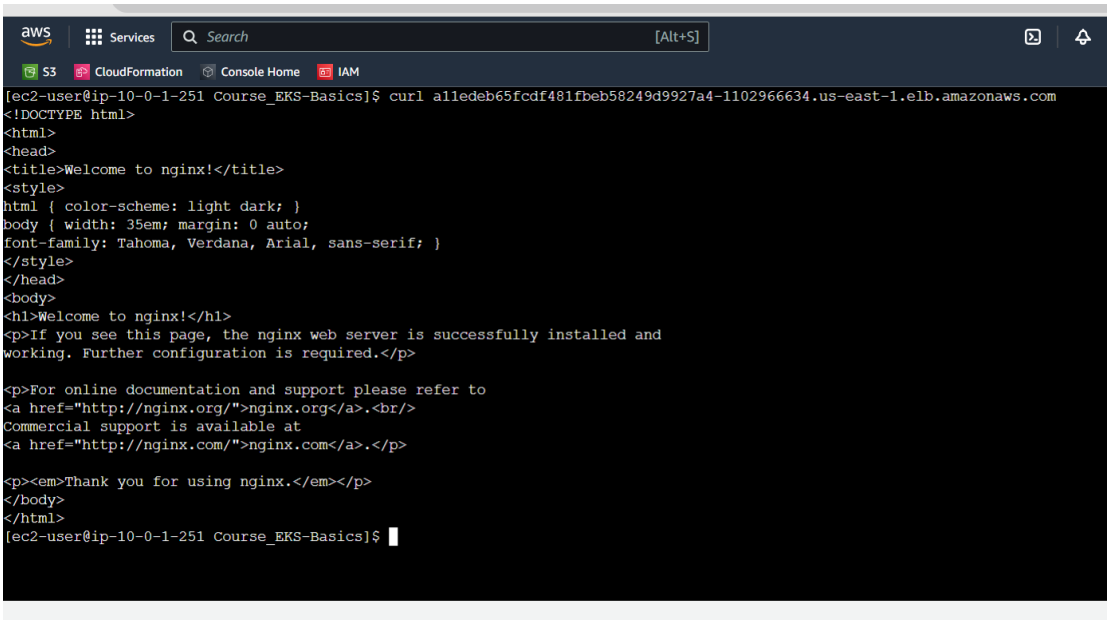
Use the DNS Name of the Load Balancer to Test the Cluster

Copy the DNS name of the load balancer.

Use the DNS name to test that the cluster is working:

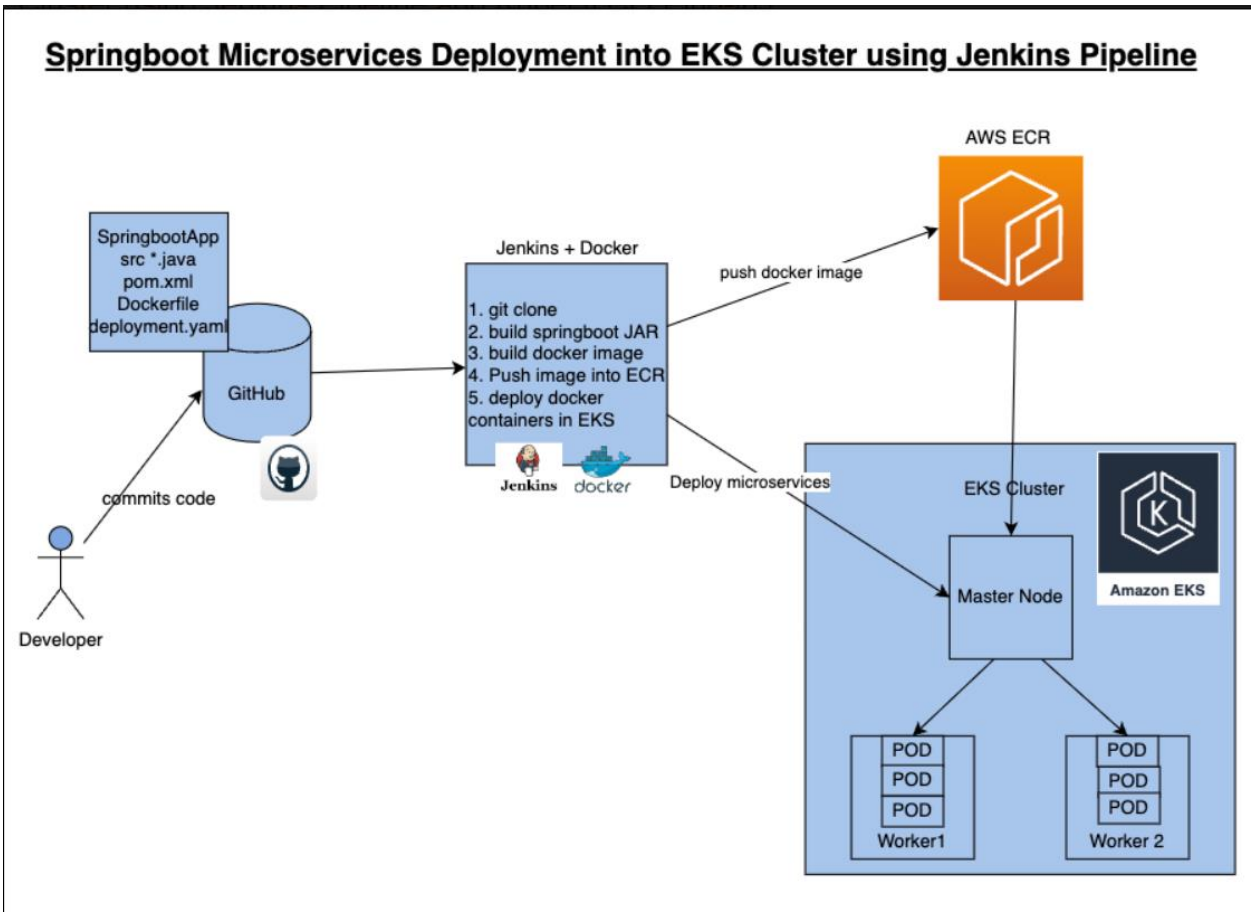
curl <PASTE DNS NAME HERE>

curl a11edeb65fcdf481fbeb58249d9927a4-1102966634.us-east-1.elb.amazonaws.com



You should see the test page for the Nginx service.

Deploy Springboot Microservices App into Amazon EKS Cluster using Jenkins Pipeline and Kubectl CLI Plug-in | Containerize Springboot App and Deploy into EKS Cluster using Jenkins Pipeline



Pre-requisites:

1. Amazon EKS Cluster is setup and running.

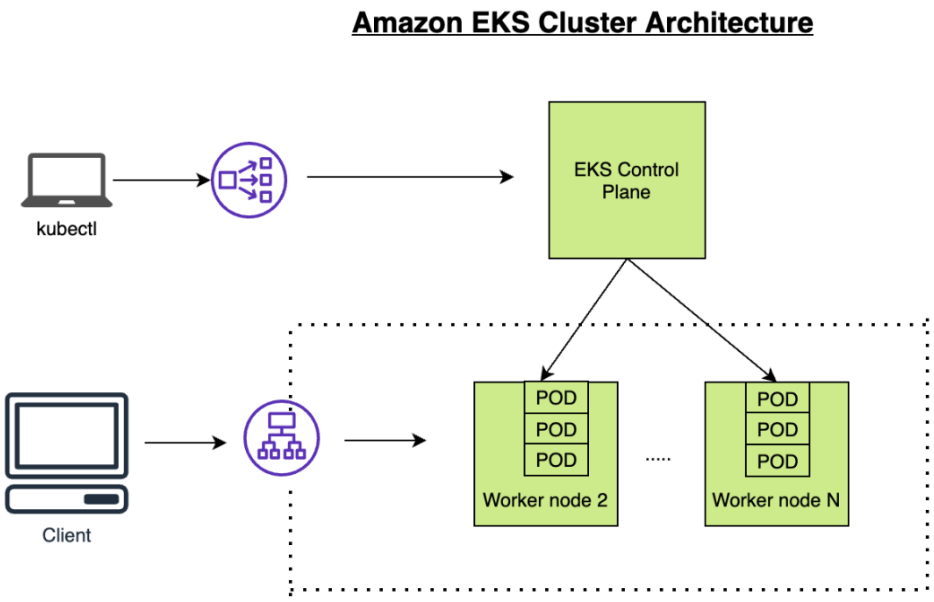
2. Create ECR repo in AWS

3. Jenkins Master is up and running

4. Docker installed on Jenkins instance

5. Docker, Docker pipeline and Kubernetes CLI plug-ins are installed in Jenkins

Setup Amazon EKS Cluster



To help achieve this we need to first setup a Jenkins **EC2** instance. This **EC2 instance**, that will be an **Ubuntu 22.4 LTS** server, needs to have following both installed and configured:

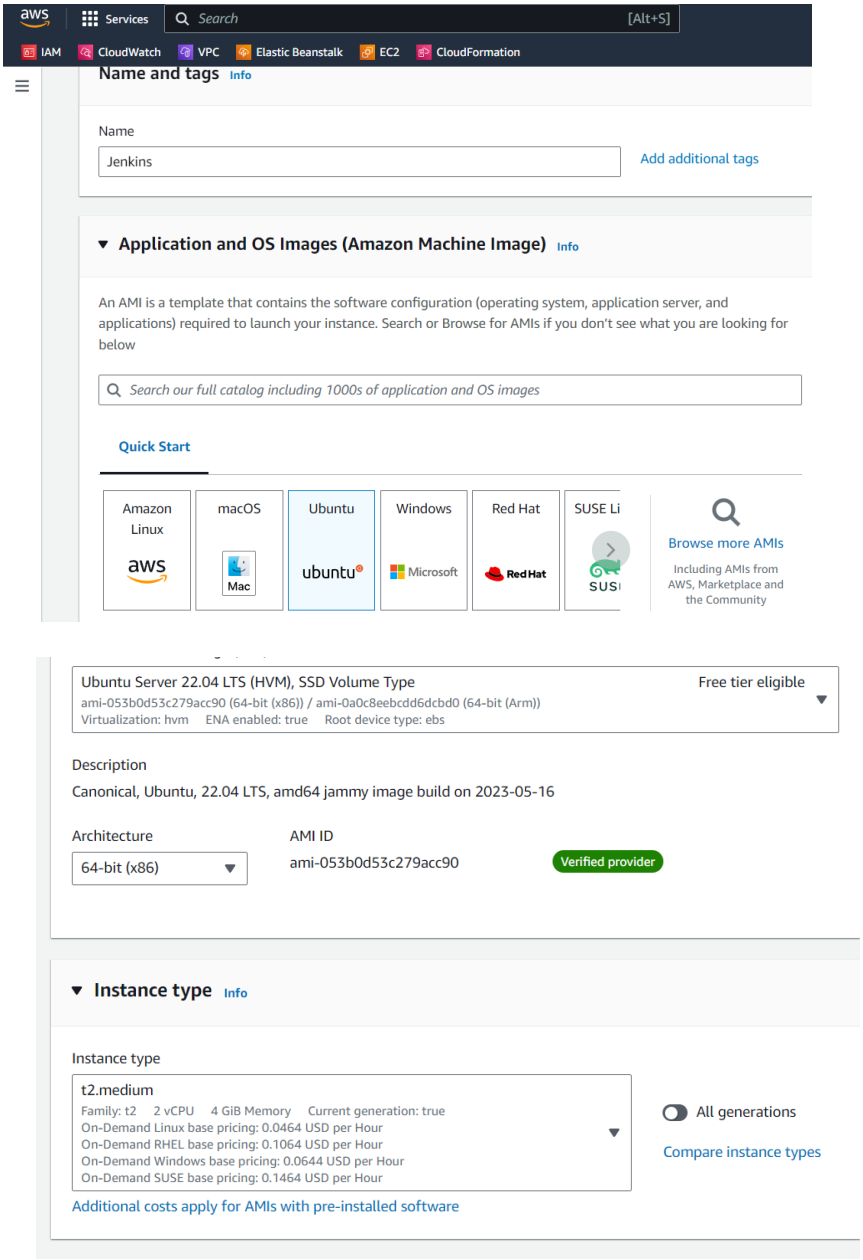
Install AWS CLI – Command line tools for working with AWS services, including Amazon EKS.

Install eksctl – A command line tool for working with EKS clusters that automates many individual tasks.

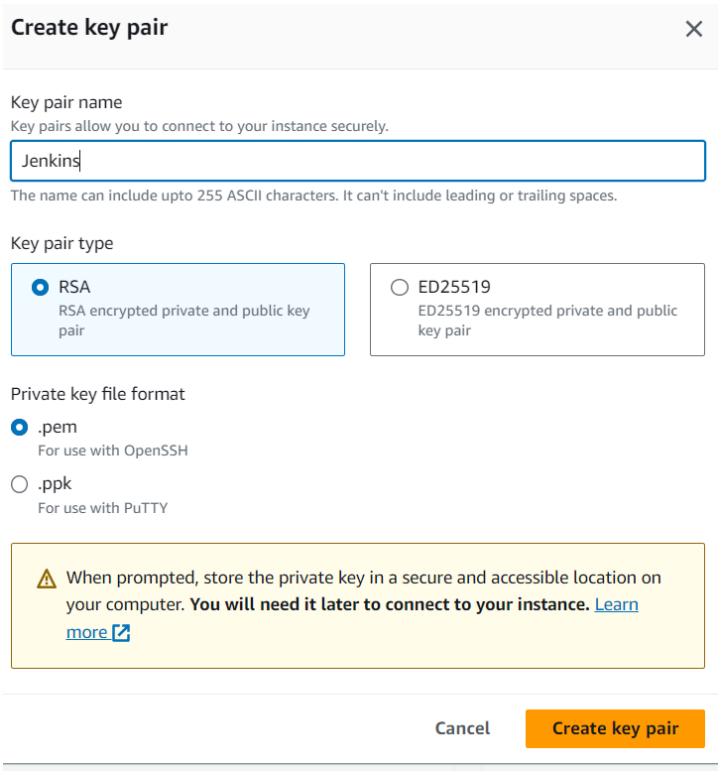
Install kubectl – A command line tool for working with Kubernetes clusters.

Create the Ubuntu instance.

Go to EC2 and launch an instance.

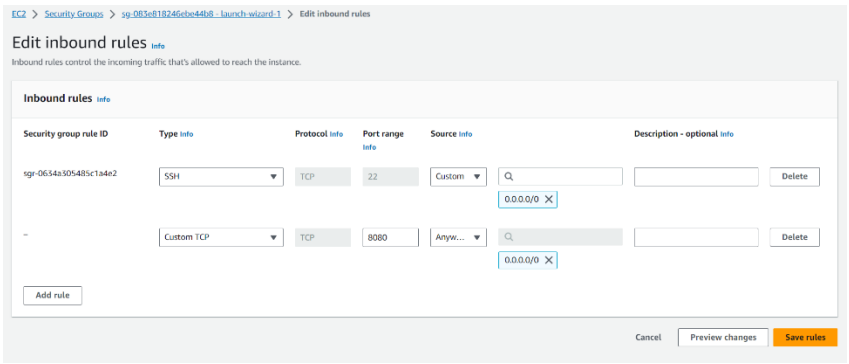


Give it the name **Jenkins,** choose Ubuntu under **Quick Start** and select a **t2.medium** instance type.



On the same page choose to create a key pair and click **Create key pair.**

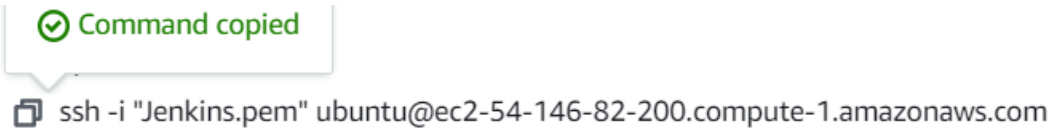
Create the instance, then once created navigate to EC2 and locate the Jenkins instance you just created security group.



Create a new inbound rule of port **8080** with **0.0.0.0/0** to give access to the Jenkins server from anywhere. Click **Save Rules**.

Click the **Connect** button within **EC2**.

Use the key pair to login into the Ubuntu instance.



Copy the above link into a command prompt on your local workstation that has the **Jenkins.pem** key pair file.

Note: Ensure that you run the above command in the same directory as the **Jenkins.pem** file.

Otherwise, you will have to alter the above command to reflect the path of the **Jenkins.pem** file.

Please follow the steps to install **Java**, **Jenkins**, and **Maven** on Ubuntu 22.0.4 instance. **Jenkins** and, **Maven** are Java based applications, so we need to install Java first.

Change Host Name to Jenkins

```
sudo hostnamectl set-hostname Jenkins
```

Note: if you have already name the server Jenkins you can skip this step.

Perform update first

```
sudo apt update
```

Install Java 11

```
sudo apt install default-jdk -y
```

```
ubuntu@ip-172-31-44-89:~$ sudo apt install default-jdk -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  at-spi2-core default-jdk-headless default-jre fonts-dejavu-extra libatk-bridge2.0-0
  libatk-wrapper-java libatk-wrapper-java-jni libatk1.0-0 libatk1.0-data libatspi2.0-0
  libdrm-amdgpu1 libdrm-intel1 libdrm-nouveau2 libdrm-radeon1 libfontenc1 libgif7 libgl1
  libgl1-mesa-dri libglapi-mesa libglvnd0 libglx-mesa0 libglx0 libice-dev libice6
  libllvm10 libpciaccess0 libpthread-stubs0-dev libsensors4 libsm-dev libsm6 libx11-dev
  libx11-doc libx11-xcb1 libxau-dev libxaw7 libxcb-dri2-0 libxcb-dri3-0 libxcb-glx0
  libxcb-present0 libxcb-shape0 libxcb-sync1 libxcb1-dev libxcomposite1 libxdamage1
```

Once install java, enter the below command

Verify Java Version

java -version

```
ubuntu@ip-172-31-19-22:/opt$ java -version
openjdk version "11.0.6" 2020-01-14
OpenJDK Runtime Environment (build 11.0.6+10-post-Ubuntu-1ubuntu118.04.1)
OpenJDK 64-Bit Server VM (build 11.0.6+10-post-Ubuntu-1ubuntu118.04.1, mixed mode, sharing)
ubuntu@ip-172-31-19-22:/opt$
```

Maven Installation

You can install Maven by executing below command:

sudo apt install maven -y

you can type mvn --version

you should see the below output.

```
ubuntu@ip-172-31-34-93:~$ mvn -version
Apache Maven 3.6.0
Maven home: /usr/share/maven
Java version: 11.0.7, vendor: Ubuntu, runtime: /usr/lib/jvm/java-11-openjdk-amd64
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "4.15.0-1065-aws", arch: "amd64", family: "unix"
ubuntu@ip-172-31-34-93:~$
```

Note: you can skip the Java step and just install Maven. This is because Maven will do the job of installing Java for you.

Now let's start Jenkins installation.

Jenkins Setup

Add Repository key to the system

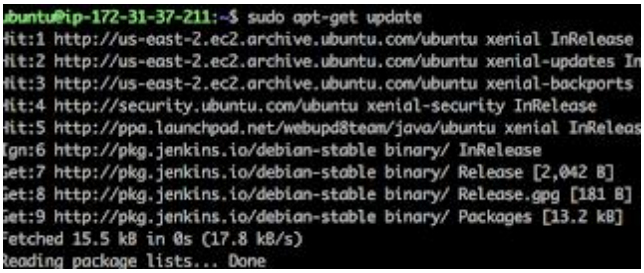
```
curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null
```

Append debian package repo address to the system

```
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
```

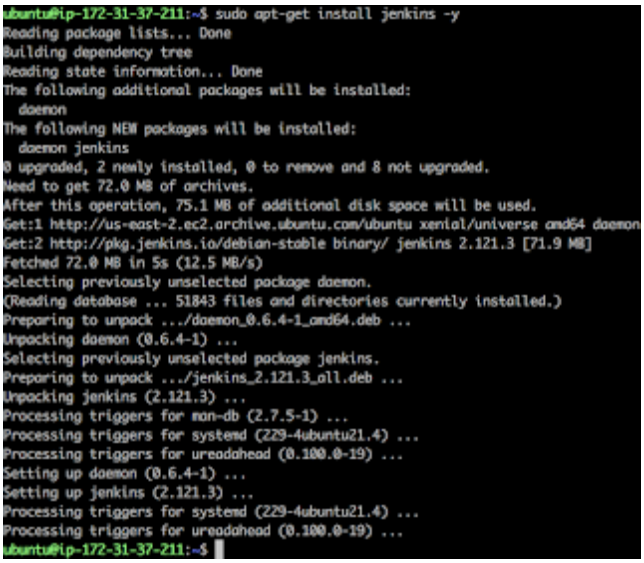
Update Ubuntu package

```
sudo apt update
```



Install Jenkins

```
sudo apt install jenkins -y
```

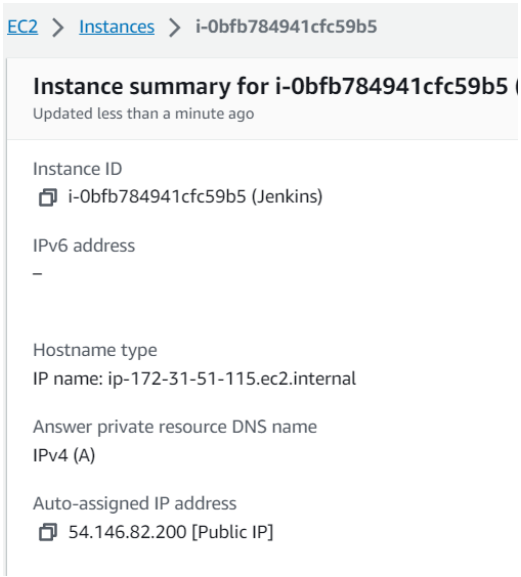


The above screenshot should confirm that Jenkins is successfully installed.

Access Jenkins in web browser

Now Go to AWS console. Click on **EC2** then click on the running instances link.

Connect to your instance using its Public IP Address:



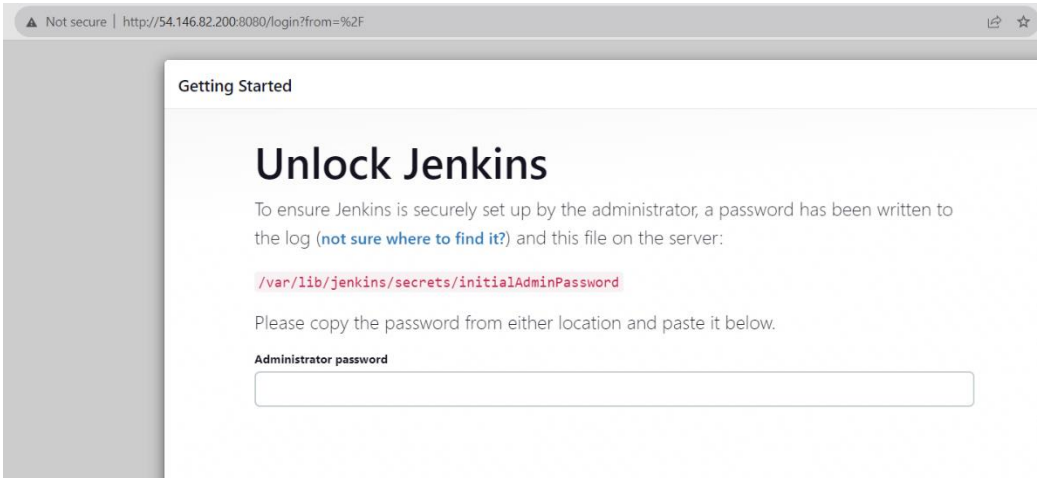
Now go to a browser and enter the **Auto-assigned (public) IP address** with port no **8080**.

http://Public IP Address:8080

It should load without issues after adding the inbound rule with the port 8080.

Unlock Jenkins

Since this is a fresh build, you will get the following screen. Enter the **cat** command below into the bash shell. (i.e., your Ubuntu console).



Get the initial password from the below file

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

Copy the password and paste it into the browser's text box under **Administrator password**.

Move to the next page.

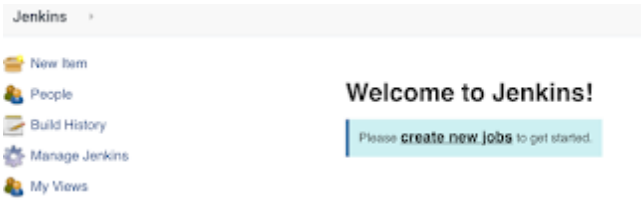
Click on **Install suggested plugins**.

Also create an admin user.

We choose for our non-production server to use both the username admin and password admin.

Click **Create User**.

Click on **Save** and **Finish**. Click on **start using Jenkins**. Now you should see a screen like below:



Install the AWS CLI version 2 on Linux.

Follow these steps from the command line to install AWS CLI on Linux.

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

sudo apt install unzip

sudo unzip awscliv2.zip

sudo ./aws/install

aws --version

it should display the below output.

Install eksctl on a Ubuntu Instance

eksctl is a command line tool for working with EKS clusters that automates many individual tasks. The ***eksctl*** tool uses **CloudFormation** under the hood, creating one stack for the EKS master control plane and another stack for the worker nodes.

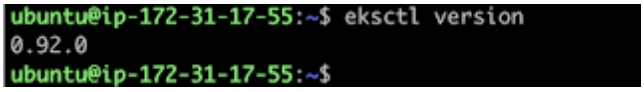
In Amazon **Elastic Kubernetes Service** (**EKS**), a **stack** typically refers to a collection of AWS **CloudFormation** resources that are used to provision and manage an EKS cluster and its associated infrastructure

Download and extract the latest release of **eksctl** with the following command.

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

Move the extracted binary to /usr/local/bin.

```
sudo mv /tmp/eksctl /usr/local/bin
eksctl version
```



Install kubectl on Ubuntu Instance

Kubernetes uses a command line utility called ***kubectl*** for communicating with the cluster API server. It is tool for controlling **Kubernetes** clusters. **kubectl** looks for a file named config in the $HOME directory.

Install kubectl

sudo curl --silent --location -o /usr/local/bin/kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.6/2022-03-09/bin/linux/amd64/kubectl

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

Verify if kubectl got installed

kubectl version --short –client

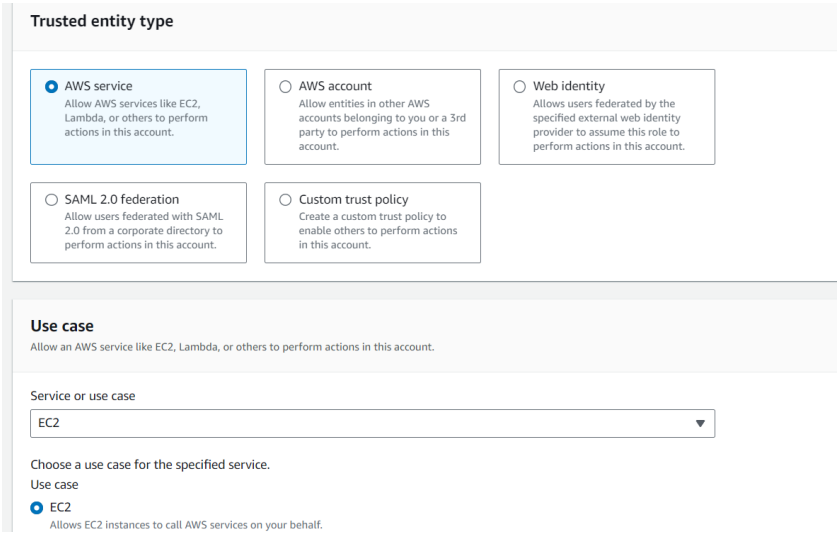
NOTE: You may get an error with Ubuntu with the above command. It may complain of deprecated command. If you find this, drop the -**short** switch and just use **-client**.

Its fine on AMI as of writing this!

Create an IAM Role with Administrator Access

You need to create an **IAM** role with **AdministratorAccess** policy.

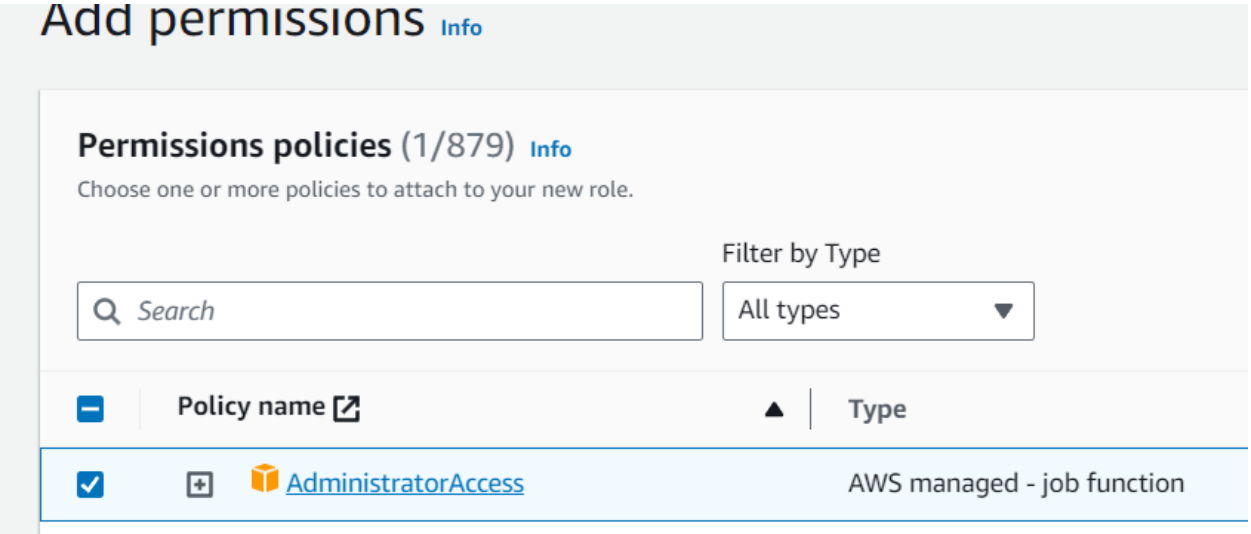
Go to the AWS console, **IAM**, click on **Roles**. create a **role**.



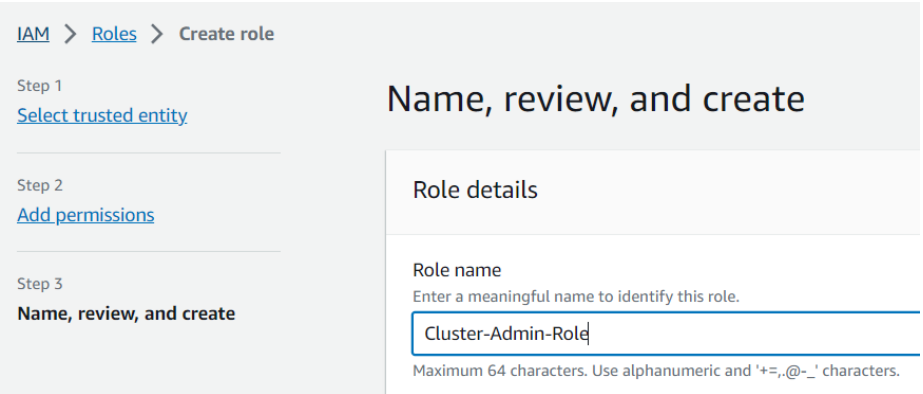
Select **AWS service**.

Under **Use case**, select **EC2**.

Click **Next**



Search for **AdministratorAccess**, select it and choose **Next**.

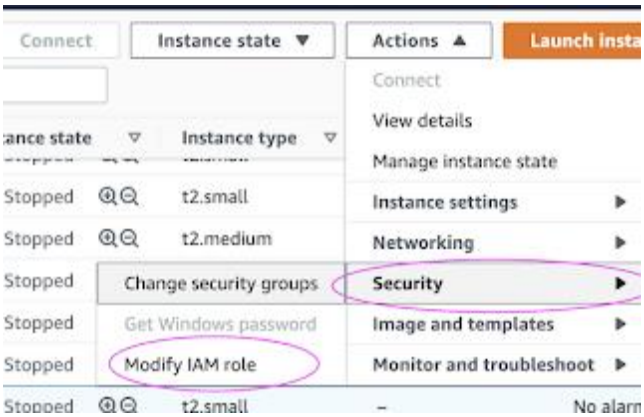


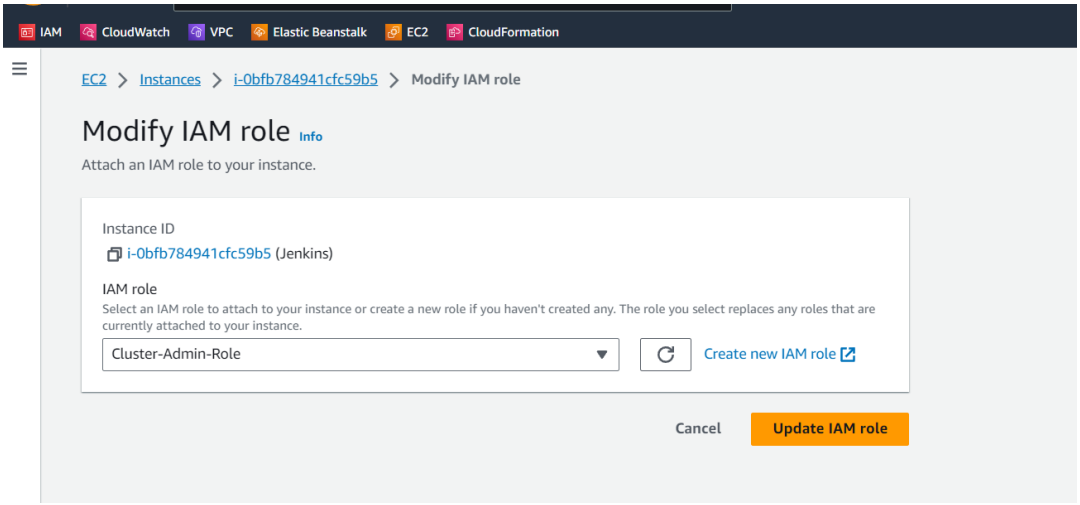
Give it a name such as **Cluster-Admin-Role** and click **Create Role**.

Assign the role to EC2 instance

Go to AWS console, click on **EC2**, select **EC2** instance, Choose **Security**.

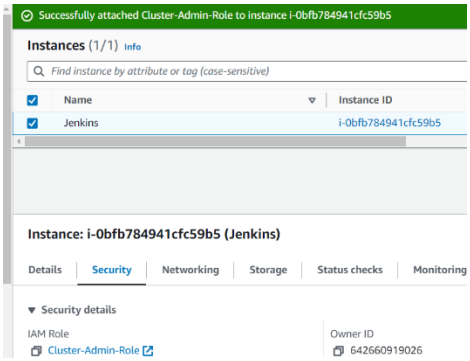
Click on **Modify IAM Role**





Choose the role you have created from the dropdown.

Click on **Update IAM role**.



You can see from above that the IAM role is attached to the Jenkins instance.

Let's got back to the Ubuntu Instance.

Switch to Jenkins user

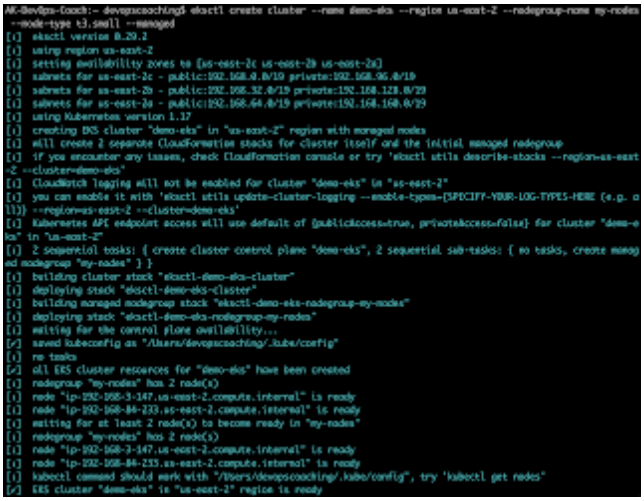
sudo su - jenkins

Create EKS Cluster with two worker nodes using eksctl

eksctl create cluster --name demo-eks --region **us-east-1** --nodegroup-name my-nodes --node-type t3.small --managed --nodes 2

Note: if we didn't explicitly declare the -nodes 2 it will create two nodes by default.

The above command should create an EKS cluster in AWS, it might take 15 to 20 mins.



After the cluster build run the following command:

eksctl get cluster --name demo-eks --region us-east-1

This should confirm that EKS cluster is up and running.

Update Kube config by entering below command:

aws eks update-kubeconfig --name demo-eks --region us-east-1

```
jenkins@ip-172-31-10-53:~$ aws eks update-kubeconfig --name demo-eks --region us-east-2
Added new context arn:aws:eks:us-east-2:211223709150:cluster/demo-eks to /var/lib/jenkins/.kube/config
```

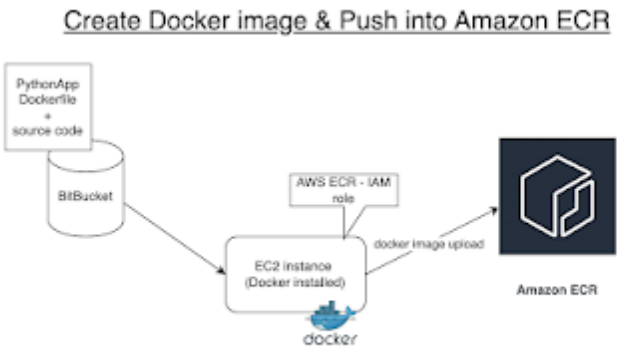
kubeconfig file be updated under the **/var/lib/jenkins/.kube** folder.

You can view the kubeconfig file by entering the below command:

cat /var/lib/jenkins/.kube/config

Create ECR repo in AWS

Amazon Elastic Container Registry (ECR) uses Amazon **S3** for storage to make your container images highly available and accessible, allowing you to reliably deploy new containers for your applications. Amazon ECR transfers your container images over HTTPS and automatically encrypts your images at rest. Amazon **ECR** is integrated with Amazon **Elastic Container Service (ECS)**, simplifying your development to production workflow.

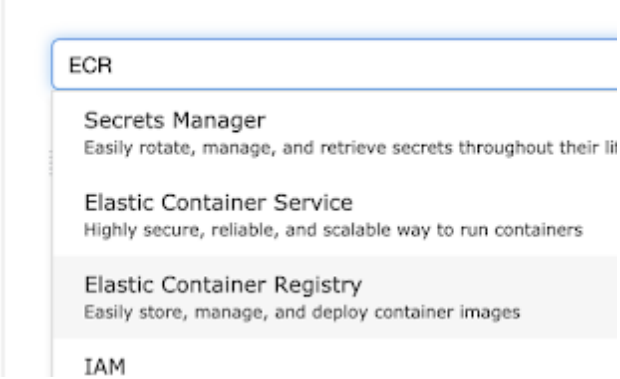


Pre-requisites:

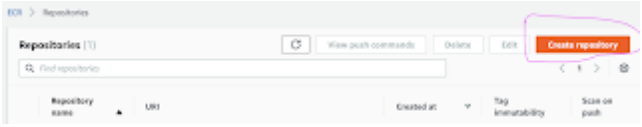
- **EC2** instance up and running with **Docker** installed.
- Make sure you open port **8081**.
- Install **aws cli**

Create a repo in ECR

Go to AWS console and search for ECR.

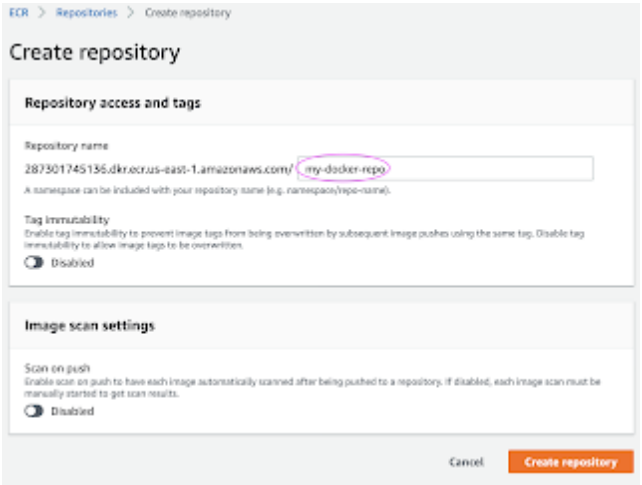


Click on **Create Repository**



Enter a name for your repo - all lower case and click create repository.

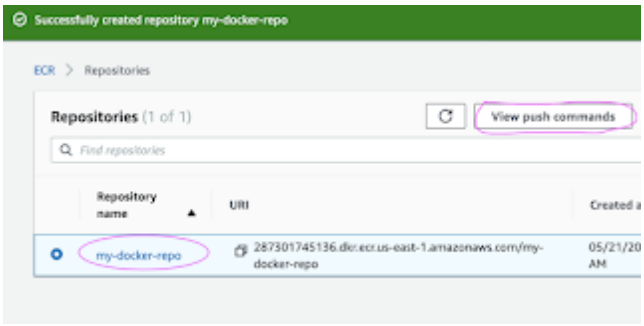
We chose **my-docker-repo**



Once the repo is created, choose the repo and click on **View push commands**.

Note down the **account ID**

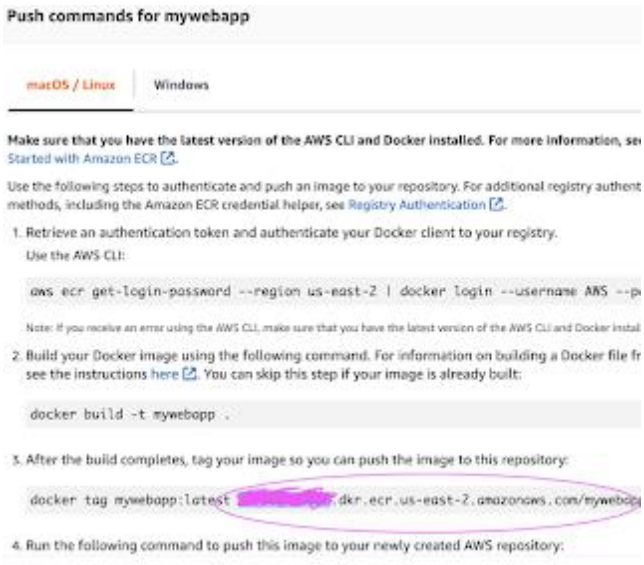
my-docker-repo



Note the URL below, as this will be used for tagging and pushing docker images into ECR.

642660919026.dkr.ecr.us-east-1.amazonaws.com/my-docker-repo

642660919026.dkr.ecr.us-east-1.amazonaws.com/my-python-repo



Copy the above into a location for later reference.

Make sure that you have the latest version of AWS CLI and Docker installed. For more information, see Getting Started with Amazon ECR .

Use the following steps to authenticate and push an image to your repository. For additional registry authentication methods, including the Amazon ECR credential helper, see Registry Authentication .

1. Retrieve an authentication token and authenticate your Docker client to your registry. Use the AWS CLI:

   aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 642660919026.dkr.ecr.us-east-1.amazonaws.com

   Note: If you receive an error using the AWS CLI, make sure that you have the latest version of the AWS CLI and Docker installed.

2. Build your Docker image using the following command. For information on building a Docker file from scratch see the instructions here . You can skip this step if your image is already built:

   docker build -t my-docker-repo .

3. After the build completes, tag your image so you can push the image to this repository:

   docker tag my-docker-repo:latest 642660919026.dkr.ecr.us-east-1.amazonaws.com/my-docker-repo:latest

4. Run the following command to push this image to your newly created AWS repository:

   docker push 642660919026.dkr.ecr.us-east-1.amazonaws.com/my-docker-repo:latest

That's it, you have created repo successfully. Let us now create docker images and push it to the above repo in ECR.

Docker installation steps using default repository from Ubuntu

Update local packages by executing below command:

sudo apt update

Install the below packages

sudo apt install gnupg2 pass -y

gnupg2 is tool for secure communication and data storage. It can be used to encrypt data and to create digital signatures.

Install docker

sudo apt install docker.io -y

Add Ubuntu user to Docker group

sudo usermod -aG docker $USER

We need to reload shell to have new group settings applied. Now you need to logout and log back in command line or execute the below command:

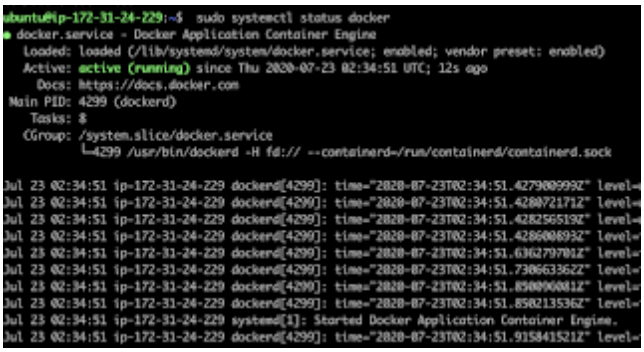
newgrp docker

The Docker service needs to be set up to run at startup. To do so, type in each command followed by enter:

sudo systemctl start docker

sudo systemctl enable docker

sudo systemctl status docker



The above screenshot should confirm that Docker daemon is up and running.

Docker, Docker pipeline and Kubernetes CLI plug-ins are installed in Jenkins.

Add jenkins user to Docker group

```
sudo usermod -a -G docker jenkins
```

Restart Jenkins service

```
sudo service jenkins restart
```

Reload system daemon files

```
sudo systemctl daemon-reload
```

Restart Docker service as well

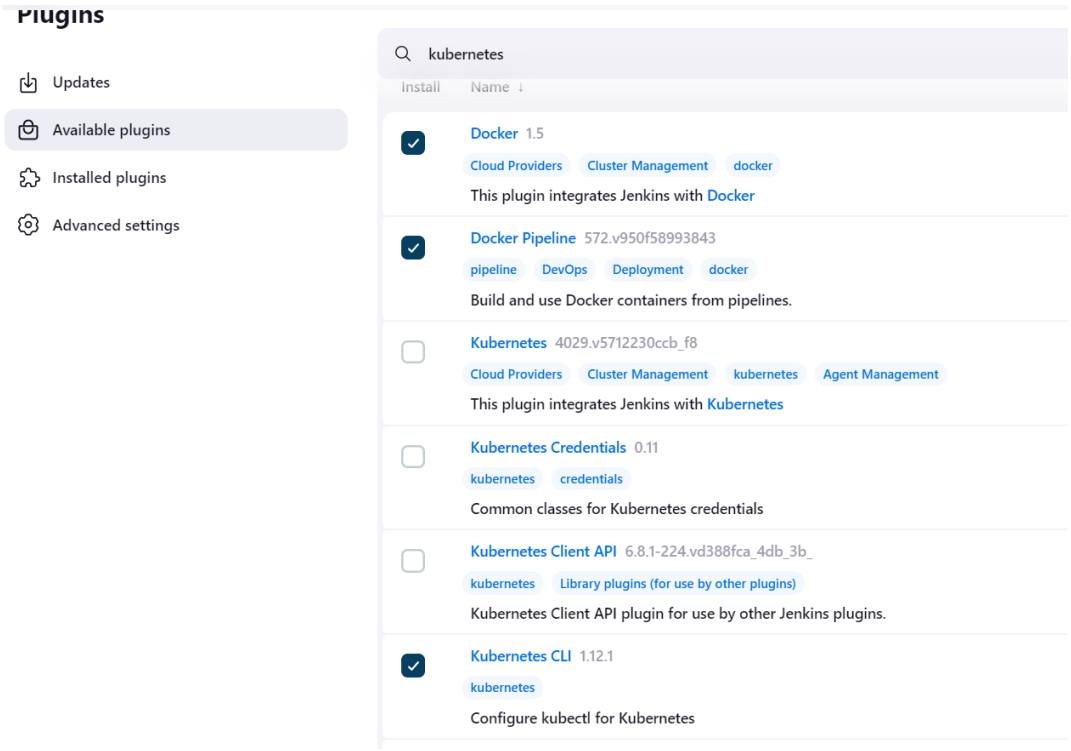
```
sudo service docker stop
sudo service docker start
```

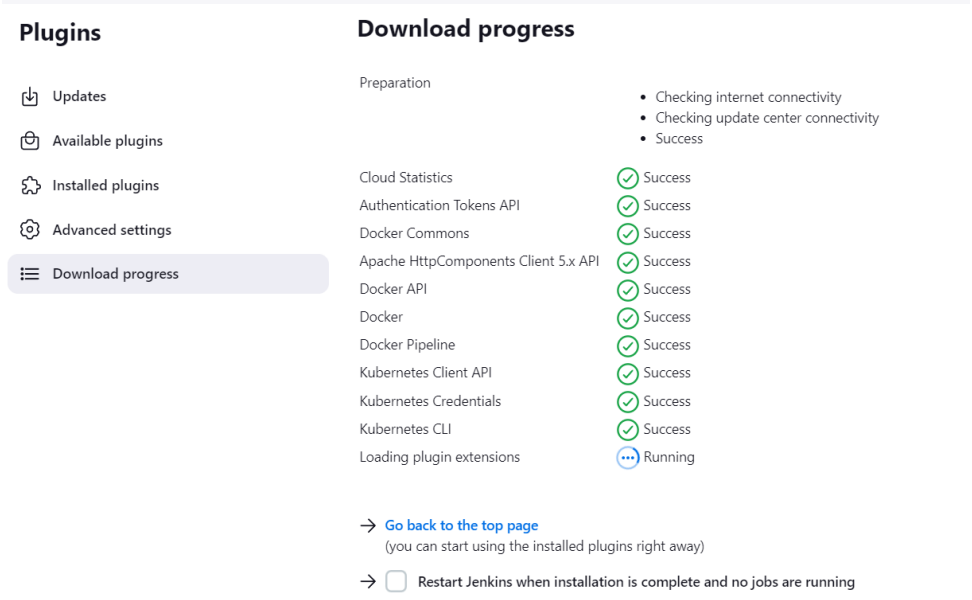
Go to the Jenkins portal

Manage Jenkins > Plugins

Available plugins

Select the following:



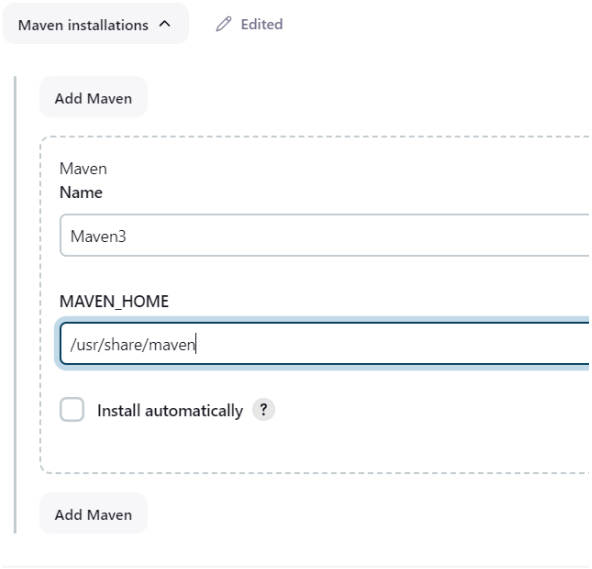


Note: do not opt to restart Jenkins

Step # 1 - Create Maven3 variable under Global tool configuration in Jenkins.

Manage Jenkins > Global tool configuration

Make sure you create a **Maven3** variable under **Global tool configuration**.

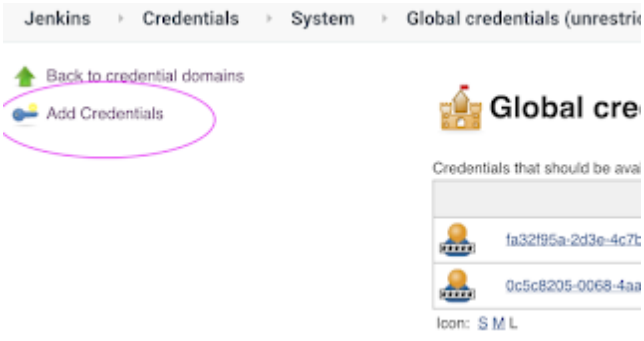


Click **Save**

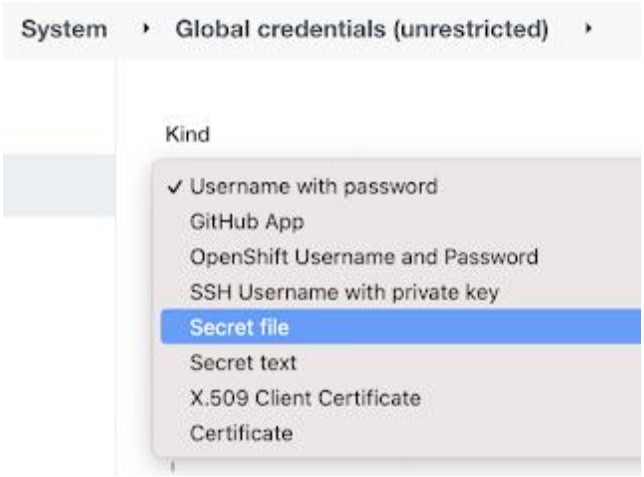
Step #2 - Create Credentials for connecting to Kubernetes Cluster using kubeconfig

Manage Jenkins > Credentials

Click on Add Credentials, use Kubernetes configuration from drop down.



Use secret file from drop down.



Let's now go back to our Jenkins server to generate the Secret file.

Execute the below command to login as jenkins user.

sudo su - jenkins

You should see the nodes running in EKS cluster.

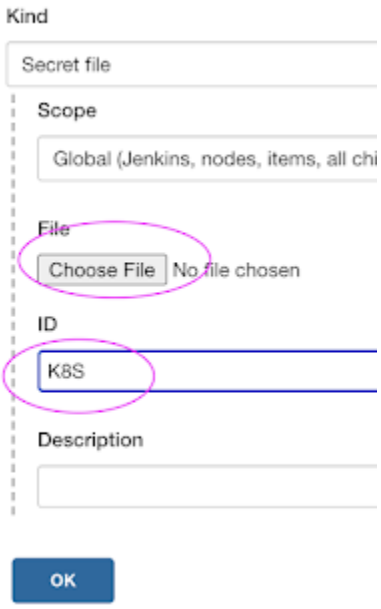
kubectl get nodes

Execute the below command to get kubeconfig info, copy the entire content of the file:

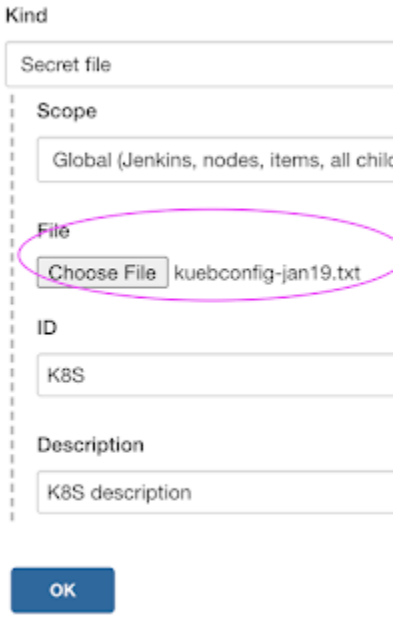
cat /var/lib/jenkins/.kube/config

Open your text editor or notepad, copy and paste the entire content and save in a file.

We will upload this file.



Enter ID as **K8S** and Choose File.

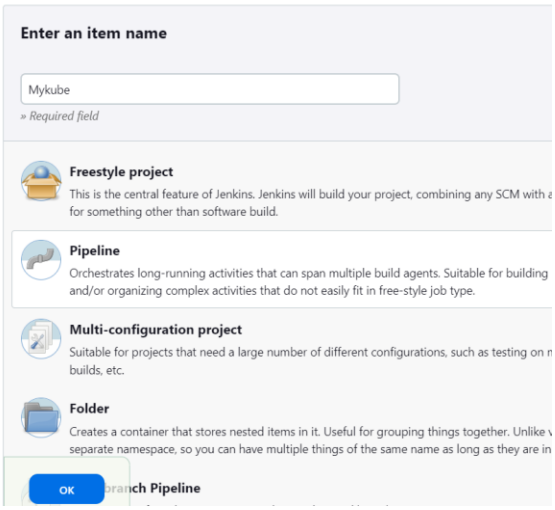


Upload the file and click **OK** to save.

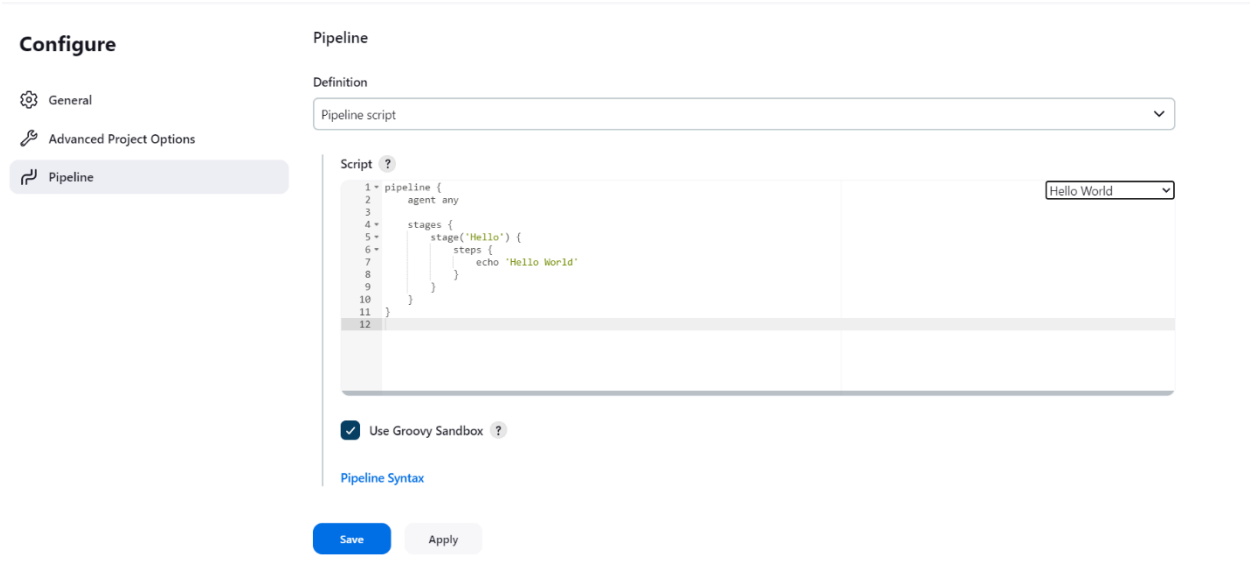
Make note of the ID **K8S** for the pipeline script later.

Step # 3 - Create a pipeline in Jenkins

Choose **New Item > Pipeline**



Call the item **Mykube** and click **OK**.

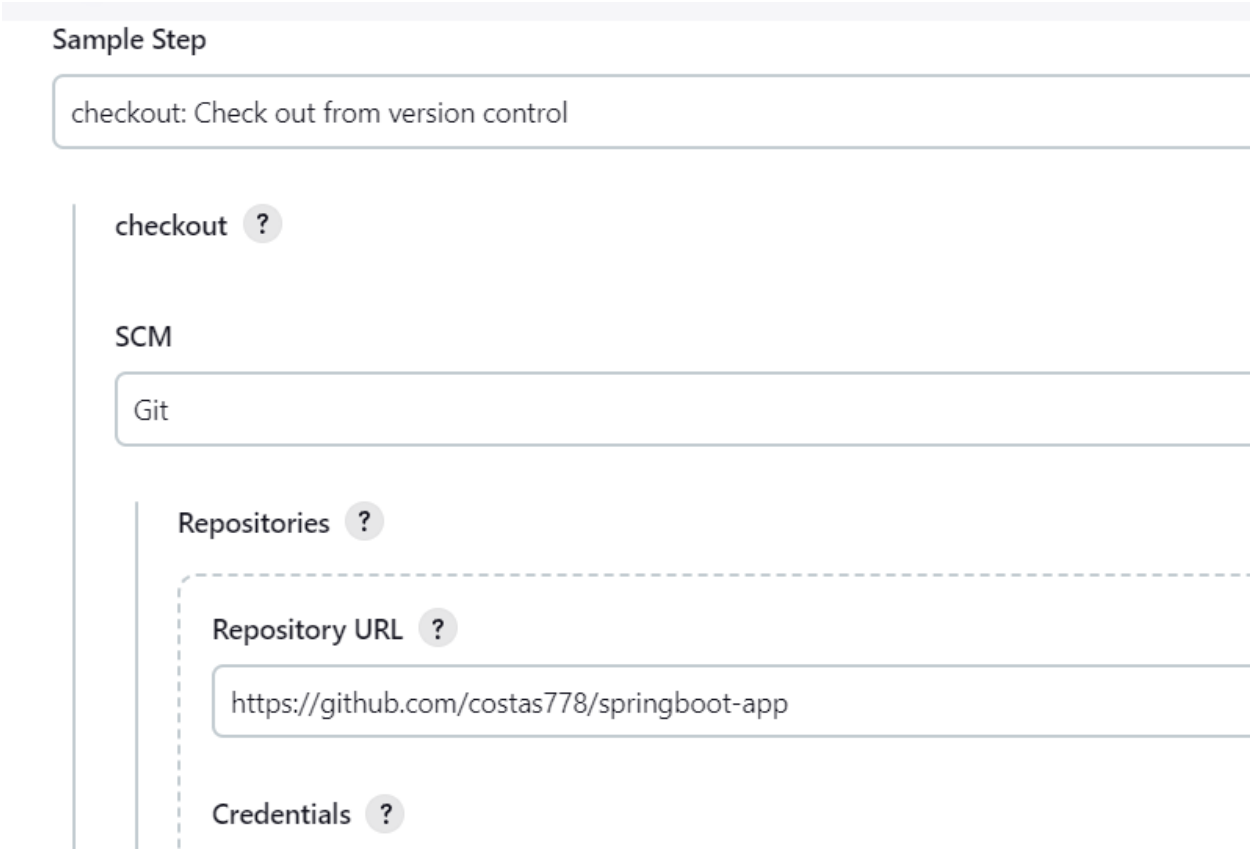


Select **Pipeline > Hello World**

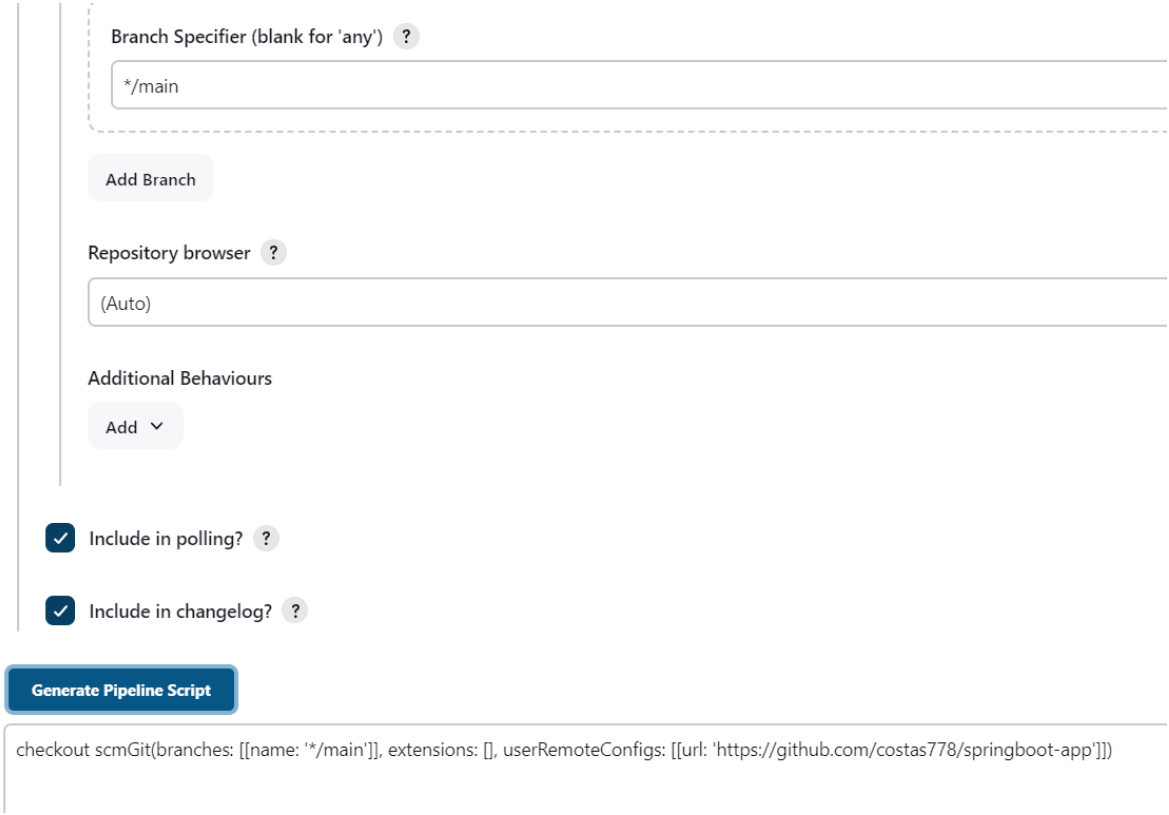
Script ?

```
pipeline {
    agent any

    stages {
        stage('Checkout') {
            steps {
                echo 'Hello World'
            }
        }
    }
}

```

Click **Pipeline Syntax**



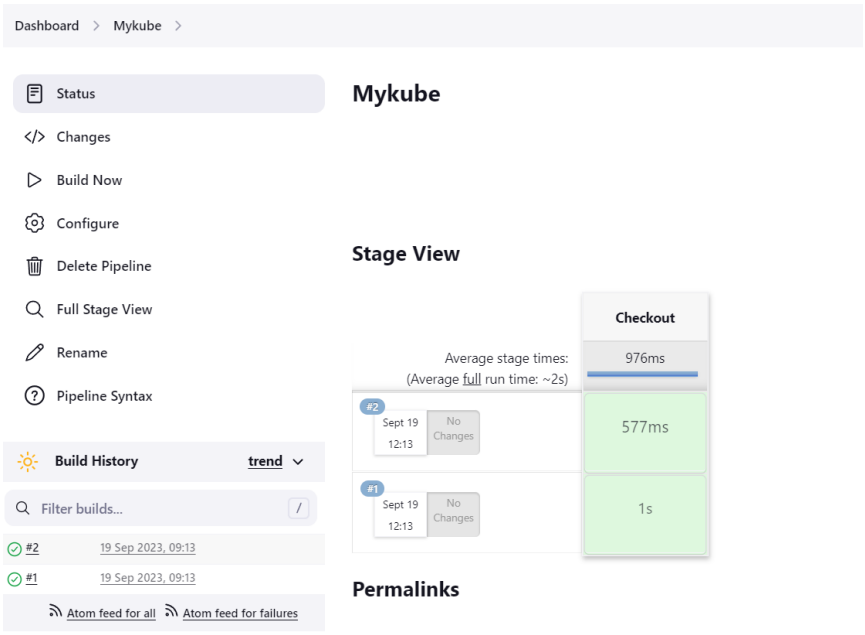
Select the above options. Ensure that the Repository URL has the URL pointing to the springboot-app as above.



Click the Generate Pipeline Script.

Copy the string back into the pipeline script.

Save the script.



Step # 4 - Copy the pipeline code from below.

Make sure you change yellow highlighted values below as per your settings:

Your docker user id should be updated with the following:

642660919026.dkr.ecr.us-east-1.amazonaws.com/my-docker-repo

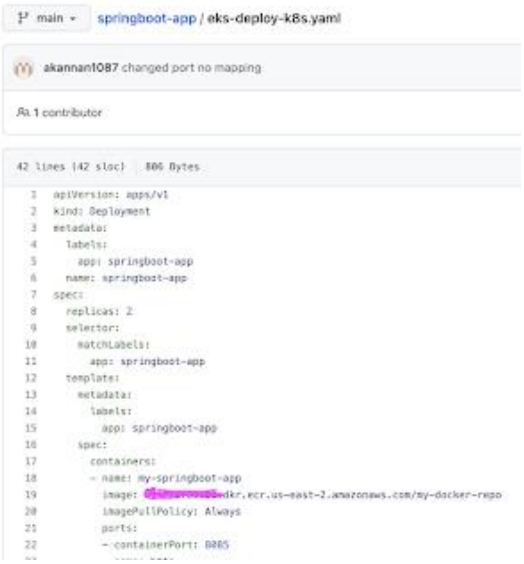
Your registry credentials ID from Jenkins we created earlier when we created the Secret file is as follows:

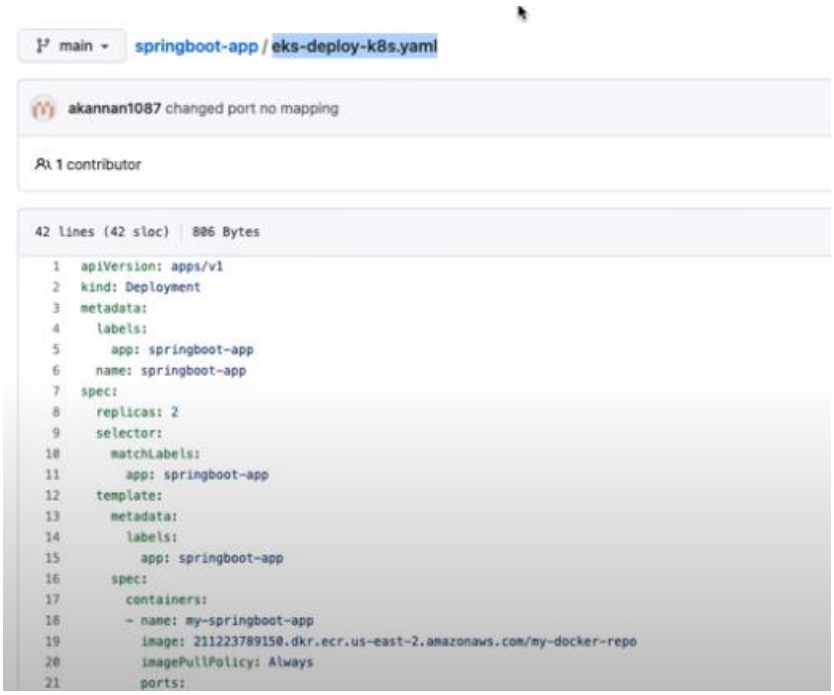
K8S

Finally, change the following **springboot-app** file. For myself this is located as follows:

https://github.com/costas778/springboot-app

Make changes in eks-deploy-k8s.yaml to pull Docker image from your AWS ECR repo.





Go to Spec. In image, update it with 642660919026.dkr.ecr.us-east-1.amazonaws.com/my-docker-repo

```
pipeline {
  tools {
      maven 'Maven3'
  }
  agent any
  environment {
      registry = "account_id.dkr.ecr.us-east-2.amazonaws.com/my-docker-repo"
  }

  stages {
      stage('Cloning Git') {
          steps {
              checkout([$class: 'GitSCM', branches: [[name: '*/main']],
doGenerateSubmoduleConfigurations: false, extensions: [], submoduleCfg: [], userRemoteConfigs:
[[credentialsId: '', url: ' https://github.com/<your GITHUB username>/springboot-app']]])
          }
      }
     stage ('Build') {
          steps {
              sh 'mvn clean install'
          }
     }
   // Building Docker images
   stage('Building image') {
     steps{
      script {
          dockerImage = docker.build registry
      }
```

```
    }
  }

  // Uploading Docker images into AWS ECR
  stage('Pushing to ECR') {
   steps{
     script {
            sh 'aws ecr get-login-password --region us-east-2 | docker login --username AWS --password-
stdin account_id.dkr.ecr.us-east-2.amazonaws.com'
            sh 'docker push account_id.dkr.ecr.us-east-2.amazonaws.com/my-docker-repo:latest'
     }
    }
  }


    stage('K8S Deploy') {
     steps{
       script {
            withKubeConfig([credentialsId: 'K8S', serverUrl: '']) {
            sh ('kubectl apply -f eks-deploy-k8s.yaml')
         }
       }
    }
  }
}
```

Once the above script is updated you can overwrite the whole pipeline script.

Step # 5 - Build the pipeline

Once you create the pipeline and changes values per your configuration, click on **Build now**:

Step # 6 - Verify deployments to K8S

kubectl get pods

```
AK-DevOps-Coach:Downloads devopscoaching$ kubectl get pods
NAME                                 READY   STATUS    RESTARTS   AGE
my-ak-deployment-67984f8cd9-2vl47    1/1     Running   0          28m
```

kubectl get deployments

```
AK-DevOps-Coach:Downloads devopscoaching$ kubectl get deployments
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
my-ak-deployment   1/1     1            1           27m
```

kubectl get services

If you see any errors after deploying the pods, you can check the pod logs.

kubectl logs <pod_name>

```
curl: (6) Could not resolve host: a7898d8a8449f48afb356e0d205069c2-41280
jenkins@Jenkins:~$ kubectl logs springboot-app-5cff7b788f-6sk7f
```

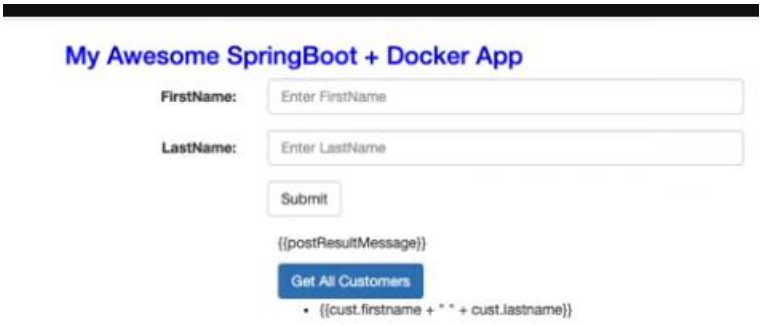
Steps # 7 - Access SpringBoot App in K8S cluster

```
jenkins@Jenkins:~$
jenkins@Jenkins:~$ kubectl get deployment
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
springboot-app   2/2     2            2           22s
jenkins@Jenkins:~$ kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
springboot-app-5cff7b788f-6sk7f   1/1     Running   0          30s
springboot-app-5cff7b788f-9cp79   1/1     Running   0          30s
jenkins@Jenkins:~$ kubectl get svc
NAME             TYPE           CLUSTER-IP       EXTERNAL-IP
      PORT(S)        AGE
kubernetes       ClusterIP      10.100.0.1       <none>
      443/TCP        21m
springboot-app   LoadBalancer   10.100.197.254   a7898d8a8449f48afb356e0d205069c2-41286874.us-east-2.elb.amazonaws
.com   80:32179/TCP   39s
jenkins@Jenkins:~$
jenkins@Jenkins:~$
jenkins@Jenkins:~$ curl a7898d8a8449f48afb356e0d205069c2-41286874.us-east-2.elb.amazonaws.com
curl: (6) Could not resolve host: a7898d8a8449f48afb356e0d205069c2-41286874.us-east-2.elb.amazonaws.com
jenkins@Jenkins:~$ kubectl
```

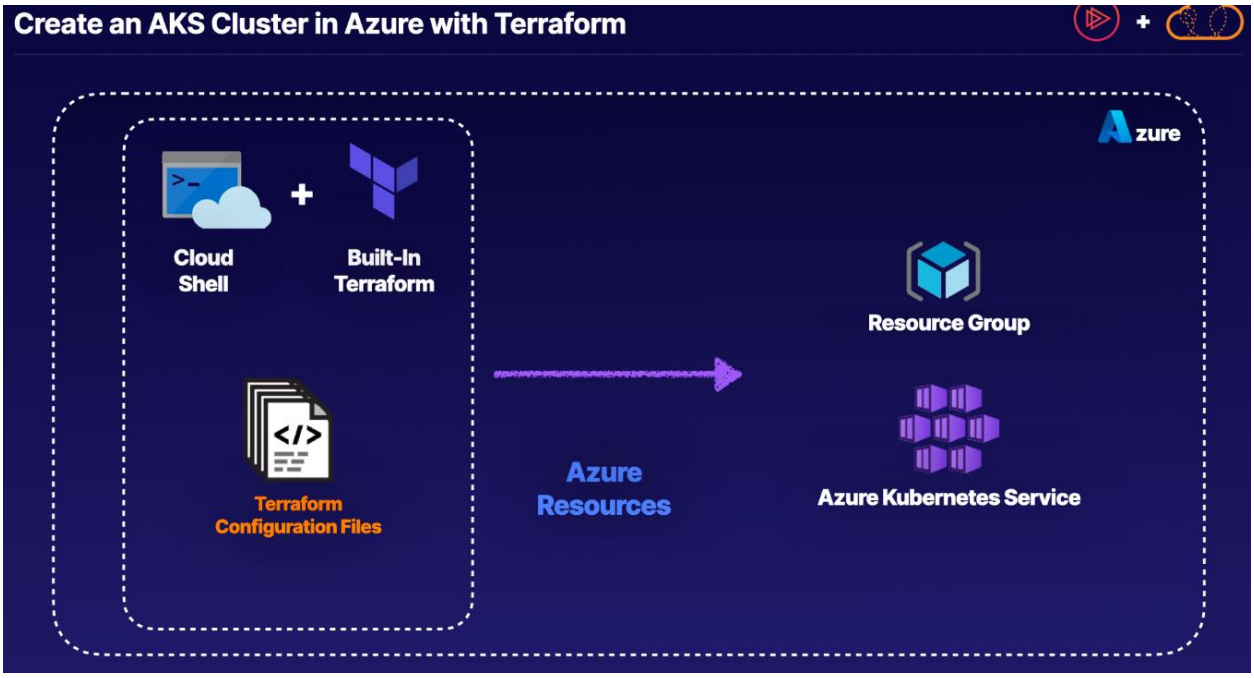
Once build is successful, go to browser and enter master or worker node public IP address along with port number mentioned above

http://loadbalancer_ip_address

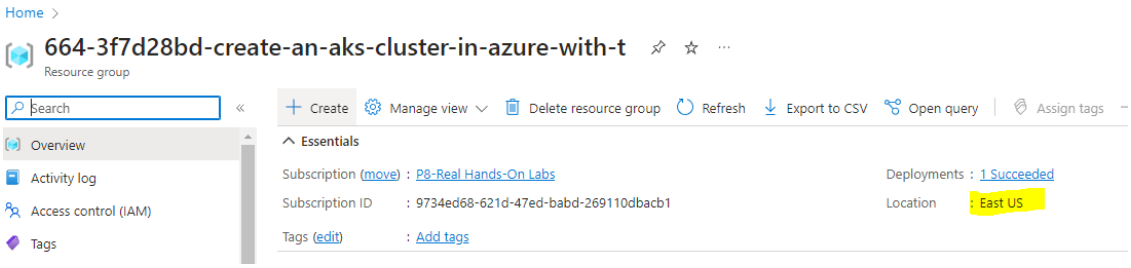
You should see page like below:



Create an AKS Cluster in Azure with Terraform



Solution



Set Up the Cloud Shell and Lab Environment

In the Azure portal, click on the Cloud Shell icon **(>_)** at the top of the page,

to the right of the search bar.

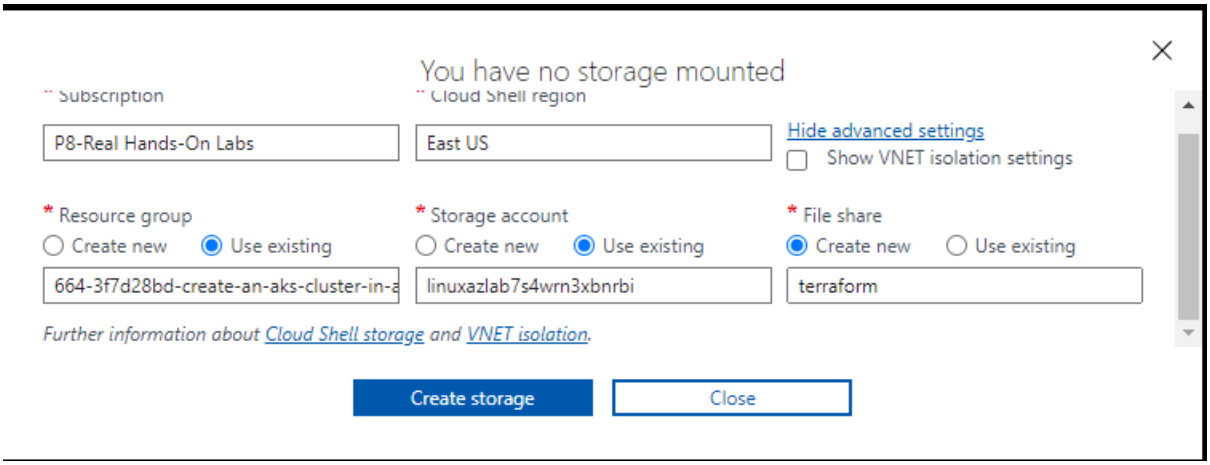
Select **Bash**.

Click Show **advanced settings**.

For the Cloud Shell region, select the **same region** as your **resource group location.**

For **Storage account**, choose Use **existing**.

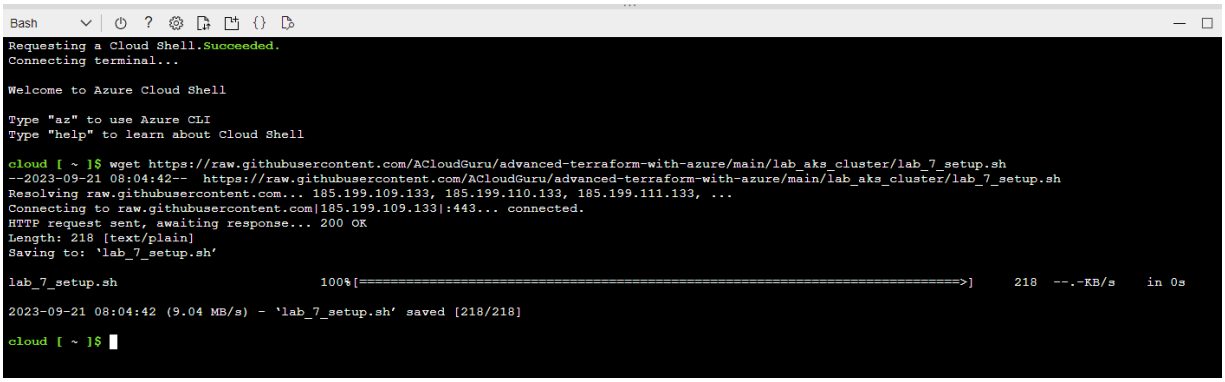
Under **File share**, select **Create new** and type in the name of **terraform**.



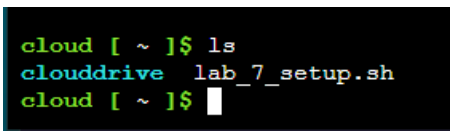
Click **Create storage**. Your Cloud Shell should begin to configure.

Run the following command to pull down the lab setup script from the GitHub repo:

wget https://raw.githubusercontent.com/ACloudGuru/advanced-terraform-with-azure/main/lab_aks_cluster/lab_7_setup.sh



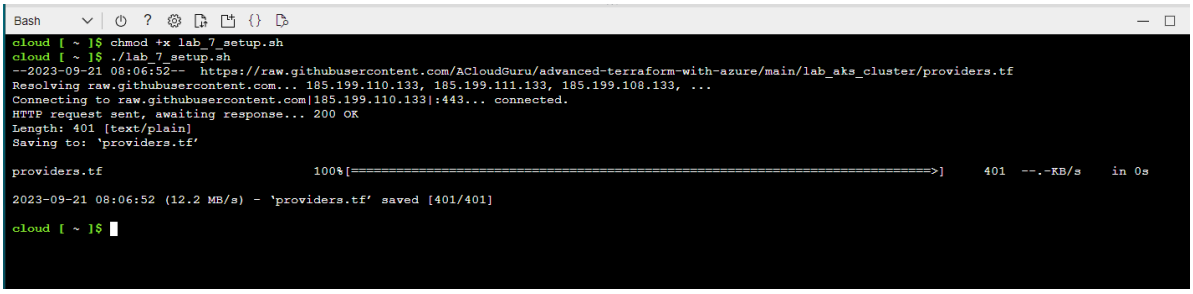
Run **ls** to list the contents.



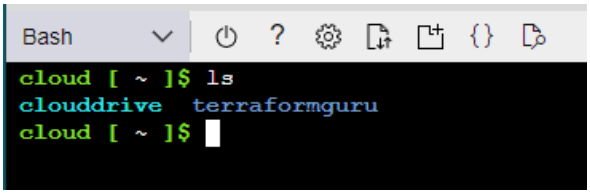
You should see the lab setup script listed, **lab_7_setup.sh**.

Run **chmod +x lab_7_setup.sh** to make it executable. .

Run the script, ./lab_7_setup.sh.



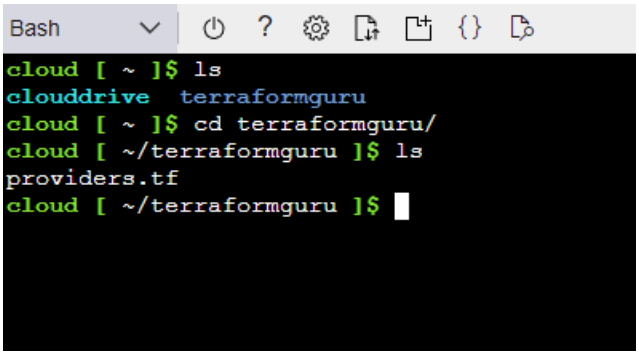
Run **ls** to list the contents.



You should see a terraformguru directory listed.

Run **cd terraformguru/** to change into that directory.

List the contents by running **ls**.



You should see one configuration file listed: **providers.tf**.

Run **vim providers.tf** to look at the file.

```
terraform {

  required_version = ">=0.12"

  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "~>2.0"
    }
  }
}

provider "azurerm" {
  features {}
  skip_provider_registration = true
}

# Import the Azure Resource Group
resource "azurerm_resource_group" "k8s" {
  #name     = "<RESOURCE_GROUP_NAME>"
  #location = "<RESOURCE_GROUP_LOCATION>"
}
```

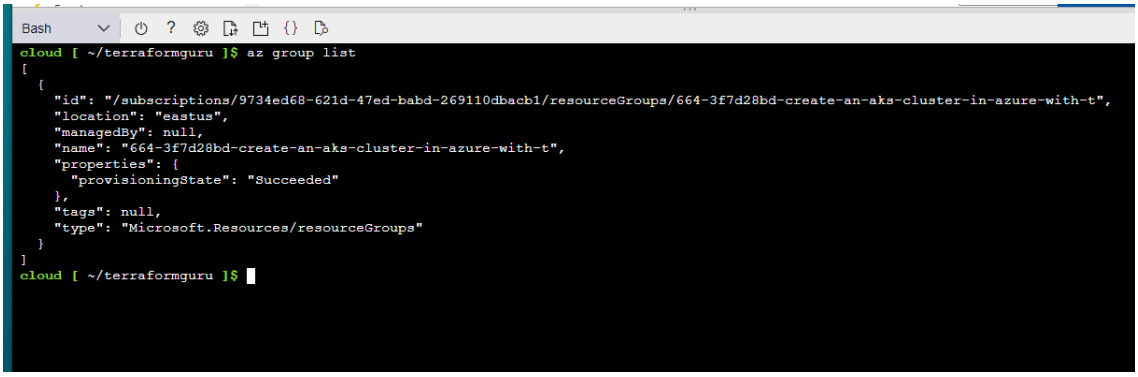
Type **Esc** then **:q** to quit out of the file.

Import the Resource Group

Run **terraform init** to initialize the working directory.

```
Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/azurerm versions matching "~> 2.0"...
- Installing hashicorp/azurerm v2.99.0...
- Installed hashicorp/azurerm v2.99.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
cloud [ ~/terraformguru ]$
```

Run **az group list** to look up the **subscription ID**.



Copy the **subscription ID** to your clipboard. It should be located on the top line after "id":

Make sure to copy all of the characters in between the quotation marks.

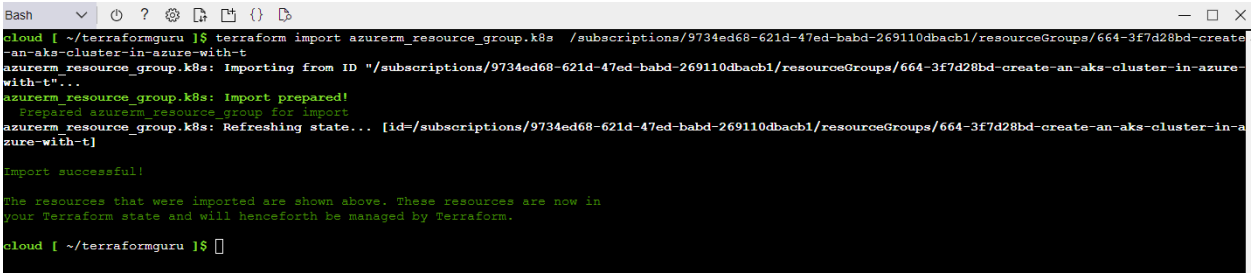
/subscriptions/9734ed68-621d-47ed-babd-269110dbacb1/resourceGroups/664-3f7d28bd-create-an-aks-cluster-in-azure-with-t

Run the following command, making sure to paste in your copied subscription ID to replace <SUBSCRIPTION_ID>:

terraform import azurerm_resource_group.k8s <SUBSCRIPTION_ID>

terraform import azurerm_resource_group.k8s /subscriptions/9734ed68-621d-47ed-babd-269110dbacb1/resourceGroups/664-3f7d28bd-create-an-aks-cluster-in-azure-with-t

Note: It may take a minute to import your resource.



Run **vim providers.tf** to edit the file.

Delete the comment hashes (#) in front of name and location.

Replace the placeholder <RESOURCE_GROUP_NAME> next to name.

Copy the resource group name located at the top left of the Azure portal, under Home.

Paste it into the file, to replace <RESOURCE_GROUP_NAME> making sure not to replace the quotation marks.

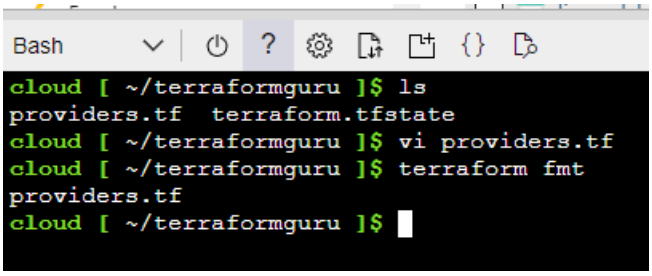
Replace the placeholder <RESOURCE_GROUP_LOCATION> next to location.

Copy the resource group location listed to the right of Location in the Azure portal.

(If you hover over it, a copy icon should appear that you can click to copy it to your clipboard.)

Paste it into the file, to replace <RESOURCE_GROUP_LOCATION> making sure not to replace the quotation marks.

Type Esc followed by :wq to save and quit the file.



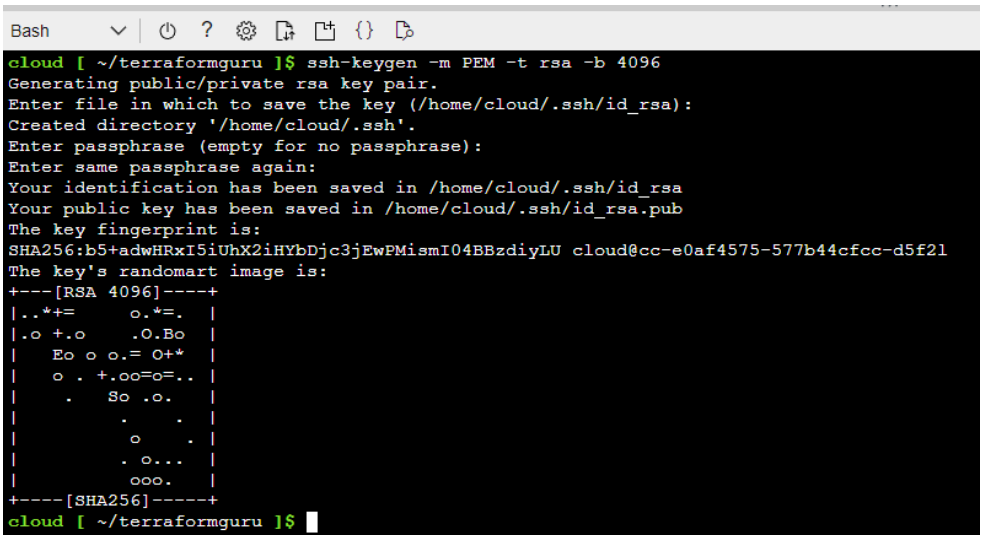
Run the following command to create an SSH key:

ssh-keygen -m PEM -t rsa -b 4096

Hit **Enter** to keep the defaults.

Hit **Enter** to leave the passphrase empty.

Hit **Enter** again to create your key pair.



Add the AKS Config, Variables, and Outputs to the Configuration:

Run **vim aks.tf** to create your first configuration file.

Enter the following configuration:

```
resource "azurerm_kubernetes_cluster" "k8s" {

  name                = var.cluster_name

  location            = azurerm_resource_group.k8s.location

  resource_group_name = azurerm_resource_group.k8s.name

  dns_prefix          = var.dns_prefix


  linux_profile {

    admin_username = "ubuntu"

    ssh_key {

      key_data = file(var.ssh_public_key)

    }

  }
```

```
  default_node_pool {
    name          = "agentpool"
    node_count     = var.agent_count
    vm_size        = "Standard_D2s_v3"
    os_disk_size_gb = 30
  }

  service_principal {
    client_id     = var.aks_service_principal_app_id
    client_secret = var.aks_service_principal_client_secret
  }

  network_profile {
    load_balancer_sku = "Standard"
    network_plugin = "kubenet"
  }

  tags = {
    Environment = "Development"
  }
}
```

Type **Esc** followed by **:wq** to save and quit the file.

Run **vim variables.tf** to create your next configuration file.

Enter the following configuration. Be sure to replace **<YOUR_RESOURCE_GROUP_LOCATION>** with the location of your resource group, and replace **<SERVICE_PRINCIPAL_APP_ID>** and **<SERVICE_PRINCIPAL_CLIENT_SECRET>** with the service principal IDs generated for this lab, which can be found in the lab credentials section.

```
variable "resource_group_location" {
   default = "<YOUR_RESOURCE_GROUP_LOCATION>"
}

variable "agent_count" {
   default = 3
}

variable "ssh_public_key" {
   default = "~/.ssh/id_rsa.pub"
}

variable "dns_prefix" {
   default = "k8sguru"
}
```

```
variable cluster_name {
    default = "k8sguru"
}

variable aks_service_principal_app_id {
   default = "<SERVICE_PRINCIPAL_APP_ID>"
}

variable aks_service_principal_client_secret {
   default = "<SERVICE_PRINCIPAL_CLIENT_SECRET>"
}
```

Type **Esc** followed by **:wq** to save and quit the file.

Run **vim output.tf** to create your final configuration file.

Enter the following configuration:

```
output "resource_group_name" {
   value = azurerm_resource_group.k8s.name
}

output "client_key" {
   value = azurerm_kubernetes_cluster.k8s.kube_config.0.client_key
}
```

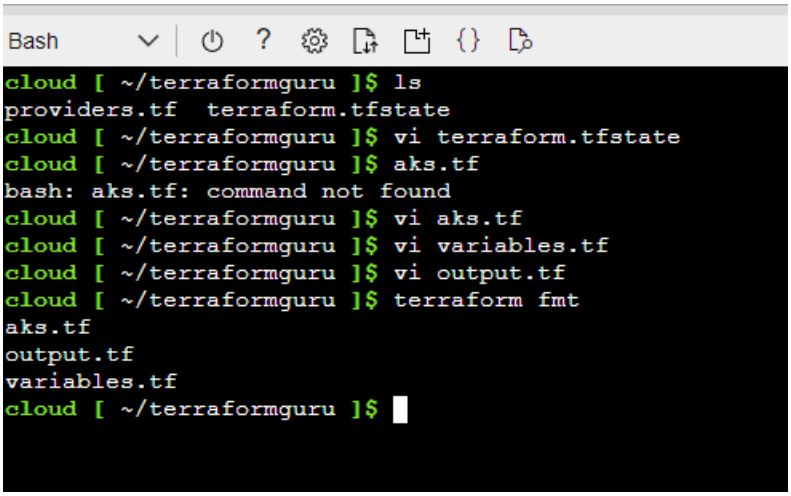
```
output "client_certificate" {
  value = azurerm_kubernetes_cluster.k8s.kube_config.0.client_certificate
}

output "cluster_ca_certificate" {
  value = azurerm_kubernetes_cluster.k8s.kube_config.0.cluster_ca_certificate
}

output "cluster_username" {
  value = azurerm_kubernetes_cluster.k8s.kube_config.0.username
}

output "cluster_password" {
  value = azurerm_kubernetes_cluster.k8s.kube_config.0.password
}
output "kube_config" {
  value = azurerm_kubernetes_cluster.k8s.kube_config_raw
  sensitive = true
}
output "host" {
  value = azurerm_kubernetes_cluster.k8s.kube_config.0.host
}
```

Type **Esc** followed by **:wq** to save and quit the file.

Deploy and Verify the Kubernetes Cluster is Running

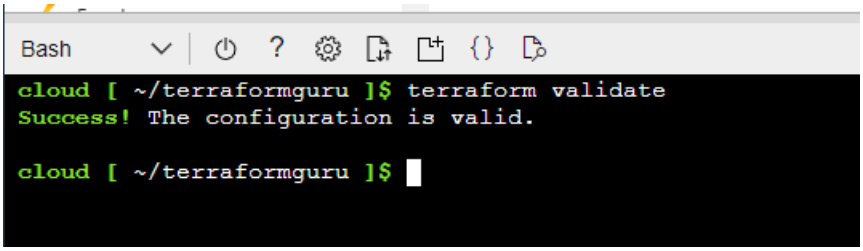
Run **terraform fmt** to check the formatting of your configuration files.

Your aks.tf, output.tf, providers.tf, and variables.tf files should be listed.

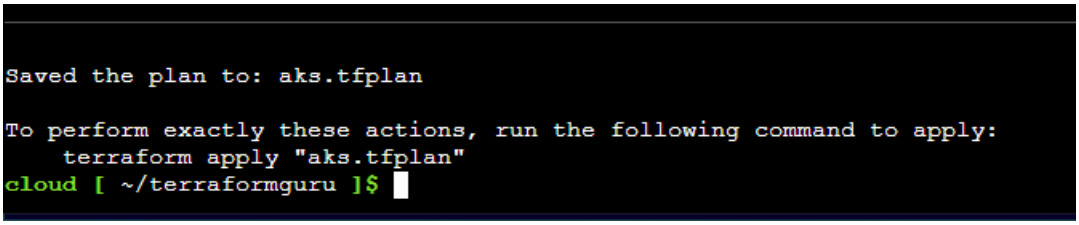


Run **terraform validate** to validate the code in your configuration files.

You should see a message confirming that your configuration is valid.

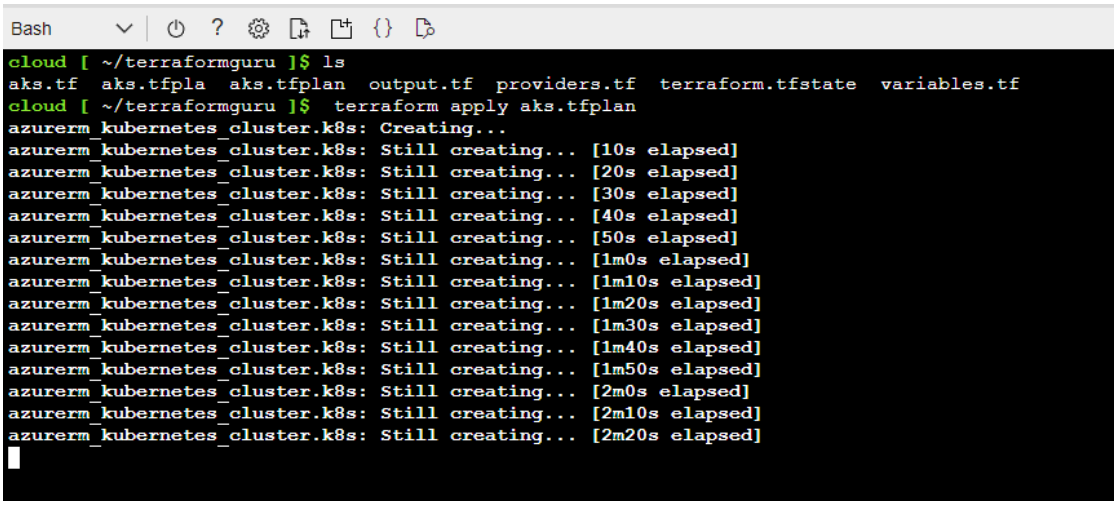


Run **terraform plan -out aks.tfplan** to create your execution plan.



Run **terraform apply aks.tfplan** to execute your execution plan.

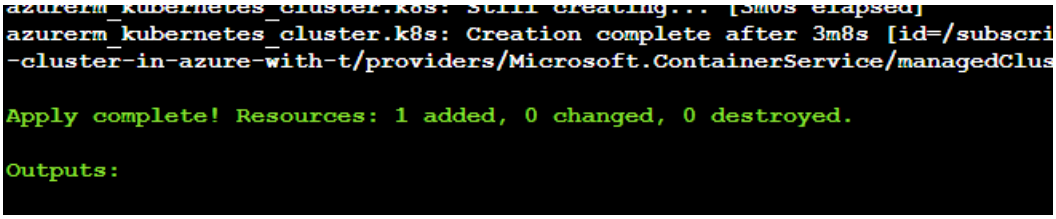
Note: It may take a couple of minutes to deploy your resources.



You will see a big block of text appear, which should mean that your cluster deployed successfully.

You can scroll up to view the Apply complete message in green to confirm.

Scrolling down from the Apply complete message, you can view the client_certificate, client_key, cluster_ca_certificate, cluster_password, and cluster_username. Lastly, you should see the host address, kube_config, and resource_group_name.



Run the following command to move your kube_config to a different file:

echo "$(terraform output kube_config)" > ./azurek8s

Run **cat ./azurek8s** to check the file.

You should see EOT at the beginning and end of the file, which will need to be removed.

Run **vim ./azurek8s** to edit the file.

Delete the **<<EOT** at the beginning and the **EOT** at the end of the file.

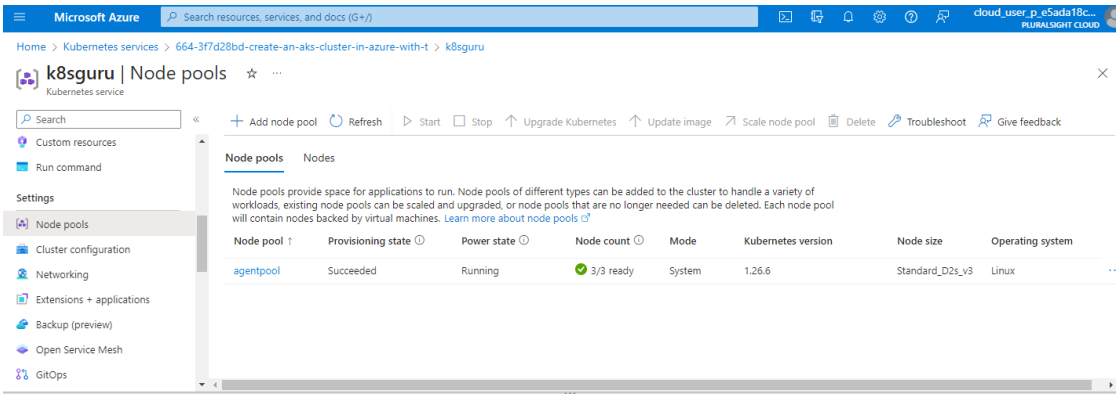
Type **Esc** followed by **:wq** to save and quit the file.

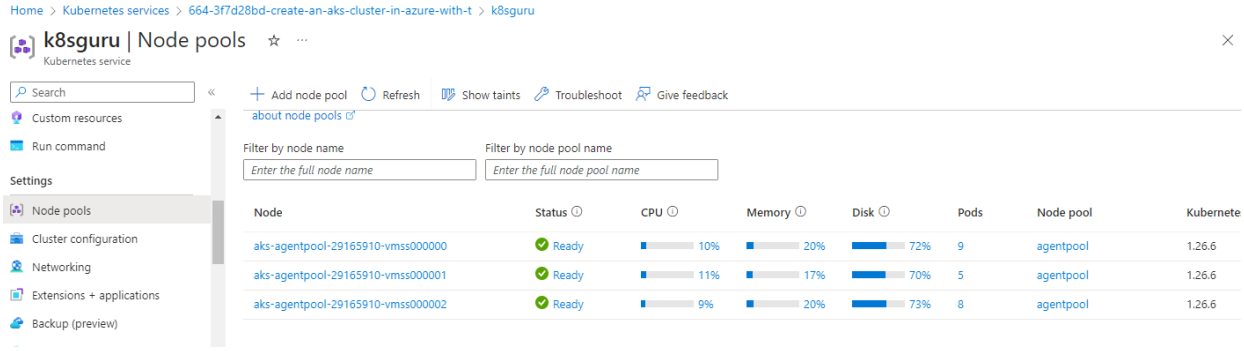
Run **export KUBECONFIG=./azurek8s** to create your environment variable.

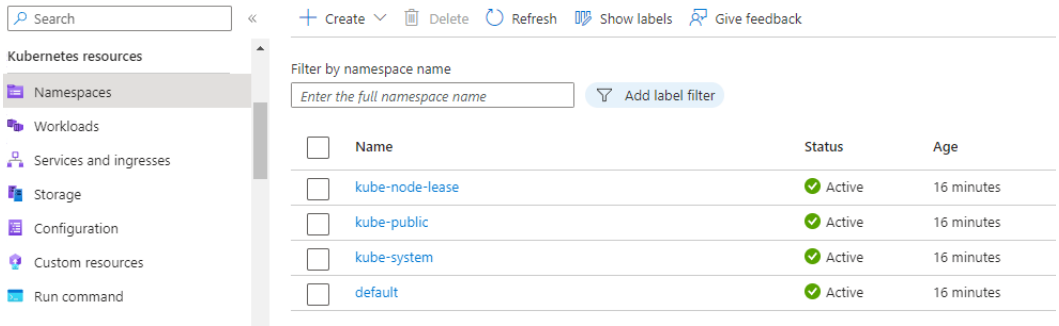
Run **kubectl get nodes** to check if your nodes are running and healthy.

You should see your 3 nodes returned with a **STATUS** of Ready.

Go to the Kubernetes Services within the Azure portal to see the same 3 nodes with a **STATUS** of Ready as well.







Addendum

```
cloud [ ~/terraformguru ]$ kubectl get namespaces
NAME              STATUS   AGE
default           Active   13m
kube-node-lease   Active   13m
kube-public       Active   13m
kube-system       Active   13m
cloud [ ~/terraformguru ]$ kubectl get pods
No resources found in default namespace.
cloud [ ~/terraformguru ]$
```

To check the current default namespace, run the following command

kubectl config view | grep namespace:

To set a new default namespace, run the following command:

kubectl config set-context --current --namespace=NAMESPACE_NAME

kubectl config set-context --current --namespace=kube-node-lease

To verify that the new default namespace has been set, run the following command:

kubectl config view | grep namespace:

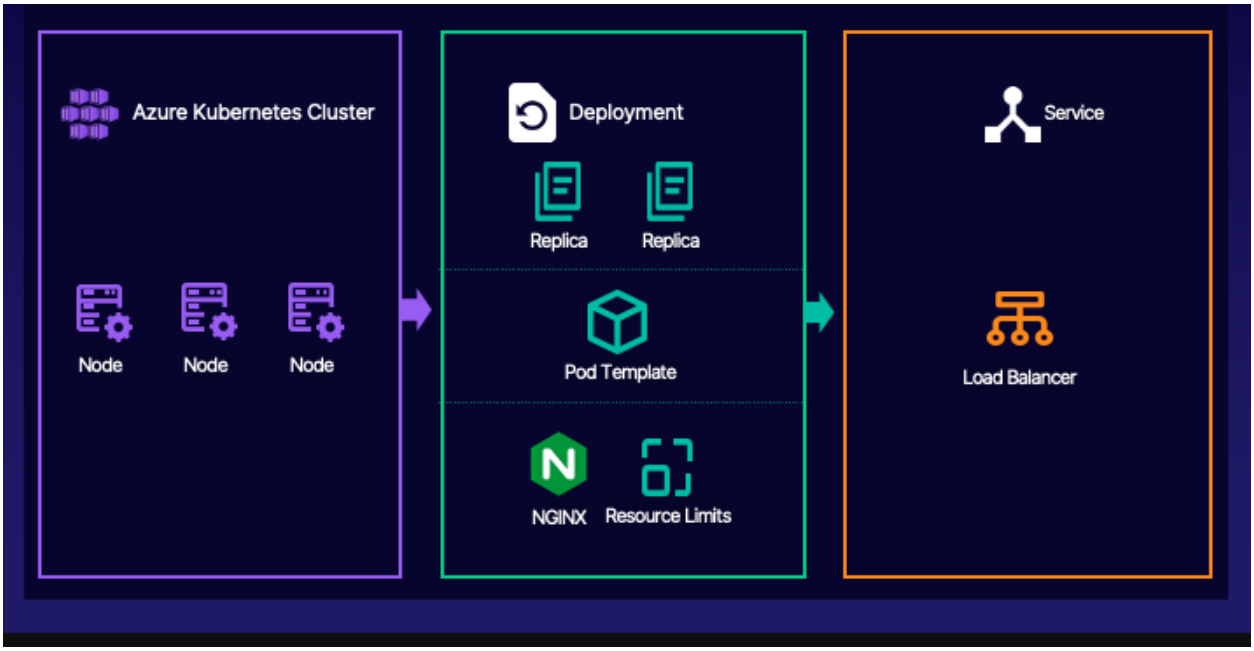
Change Namespace for a Single Command

If you want to use a different namespace for a single command, you can specify it using the "--namespace" flag. For example, to list all the pods in a different namespace, run the following command:

kubectl get pods --namespace=NAMESPACE_NAME

kubectl get pods --namespace=KUBE-NODE-LEASE

Deploying and Accessing an Application to an AKS Cluster

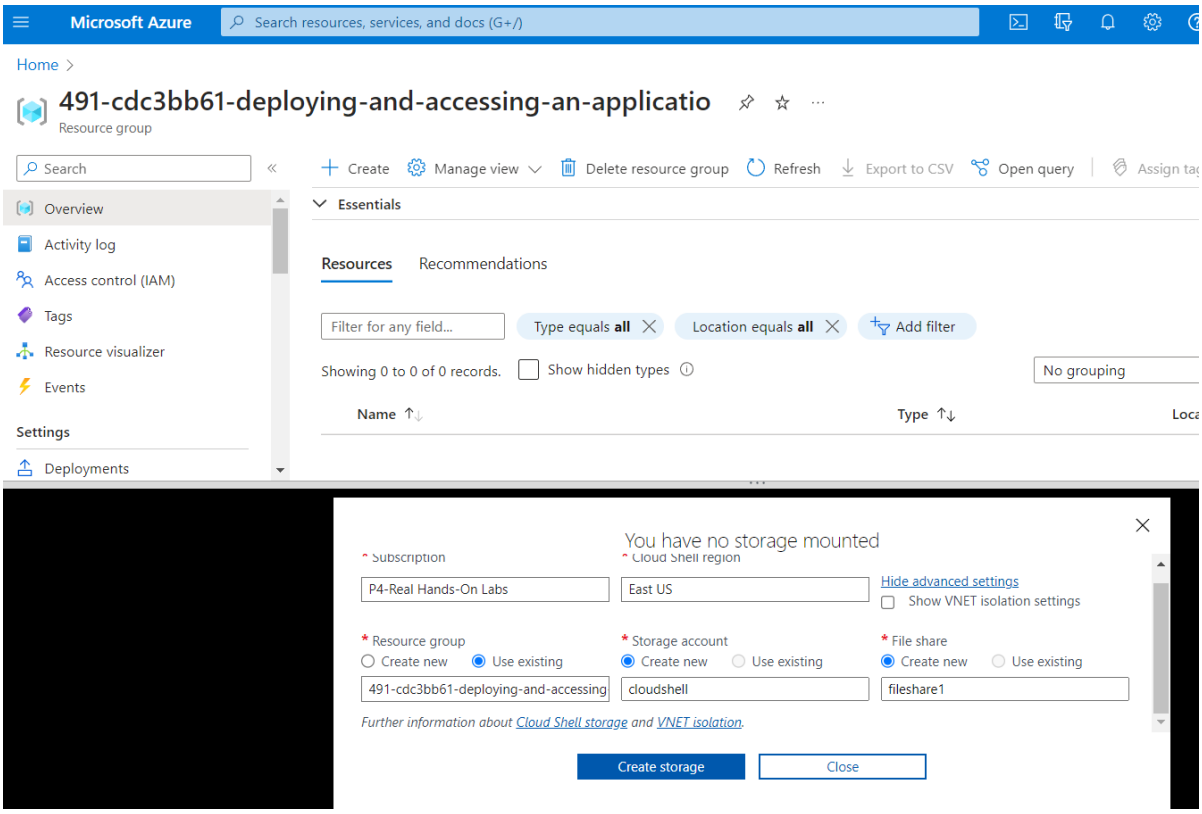


Introduction

We will work with the Azure Command Line Interface (CLI) to create an AKS cluster. We will then use the kubectl command line utility to deploy an application to that cluster, and then add a kubernetes service, so that the application is accessible over the internet.

Solution
Log in to the Azure portal using the credentials provided on the lab instructions page.

Create AKS Cluster
In the Azure Portal, click the Cloud Shell icon **(>_)** in the upper right.
Select **Bash**.
Click **Show advanced settings**.
Set the **Cloud Shell region** to the same location as the existing resource group.
For **Storage account**, select **Create new** and give it a globally unique name (e.g., "**cloudshell**" with a series of numbers at the end).
For **File share**, select **Create new** and give it a name of "**fileshare1**".
Click **Create storage**.



Locate the resource group name at the top of the webpage and copy it for later use:

491-cdc3bb61-deploying-and-accessing-an-applicatio

az group list

```
cloud [ ~ ]$ az group list
[
 {
  "id": "/subscriptions/0cfe2870-d256-4119-b0a3-16293ac11bdc/resourceGroups/491-cdc3bb61-deploying-and-accessing-an-applicatio",
  "location": "eastus",
  "managedBy": null,
  "name": "491-cdc3bb61-deploying-and-accessing-an-applicatio",
  "properties": {
   "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": "Microsoft.Resources/resourceGroups"
 }
]
cloud [ ~ ]$
```

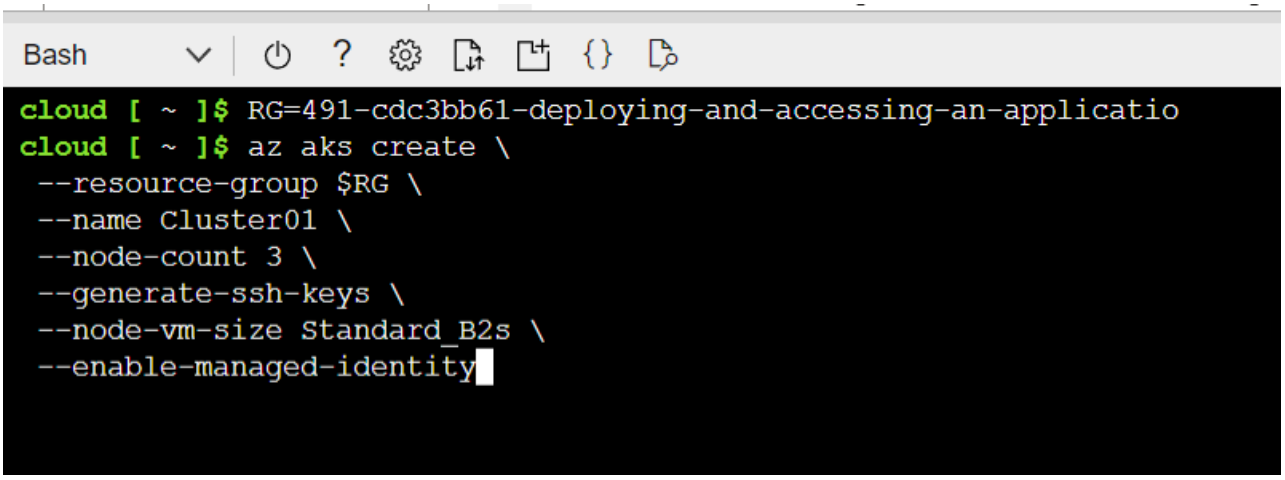
Create a variable for the resource group name:

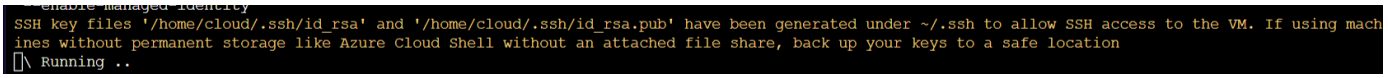
RG=<RESOURCE_GROUP_NAME>

RG=491-cdc3bb61-deploying-and-accessing-an-applicatio

Create an AKS cluster:

```
az aks create \
 --resource-group $RG \
 --name Cluster01 \
 --node-count 3 \
 --generate-ssh-keys \
 --node-vm-size Standard_B2s \
 --enable-managed-identity
```





In the Azure Portal, under **All services**, navigate to **Kubernetes services**, and verify that the cluster was deployed successfully.

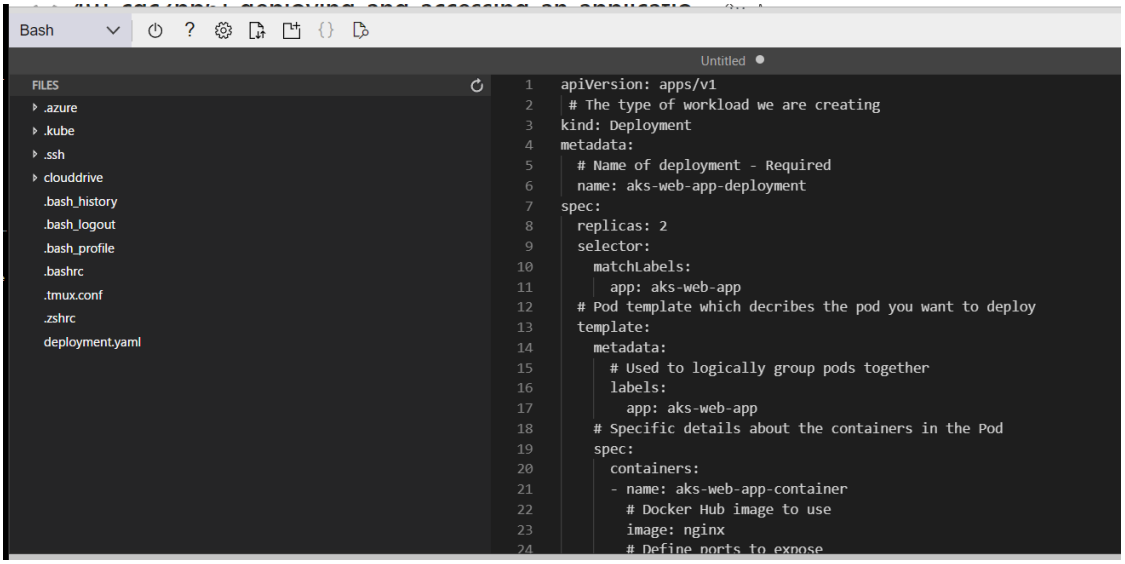
Back in the Cloud Shell, configure kubectl so that commands can be run against the cluster:

az aks get-credentials --name Cluster01 --resource-group $RG

```
Read more about the command in reference docs
cloud [ ~ ]$ RG=491-cdc3bb61-deploying-and-accessing-an-applicatio
cloud [ ~ ]$ az aks get-credentials --name Cluster01 --resource-group $RG
Merged "Cluster01" as current context in /home/cloud/.kube/config
cloud [ ~ ]$
```

Deploy the Application

Create a new deployment manifest file:
touch deployment.yaml
Open the editor by clicking the curly braces **{ }**, and open the **deployment.yaml** file.
Copy and paste the contents of the **deployment.yaml** file from the **GitHub** repo, into the new file.

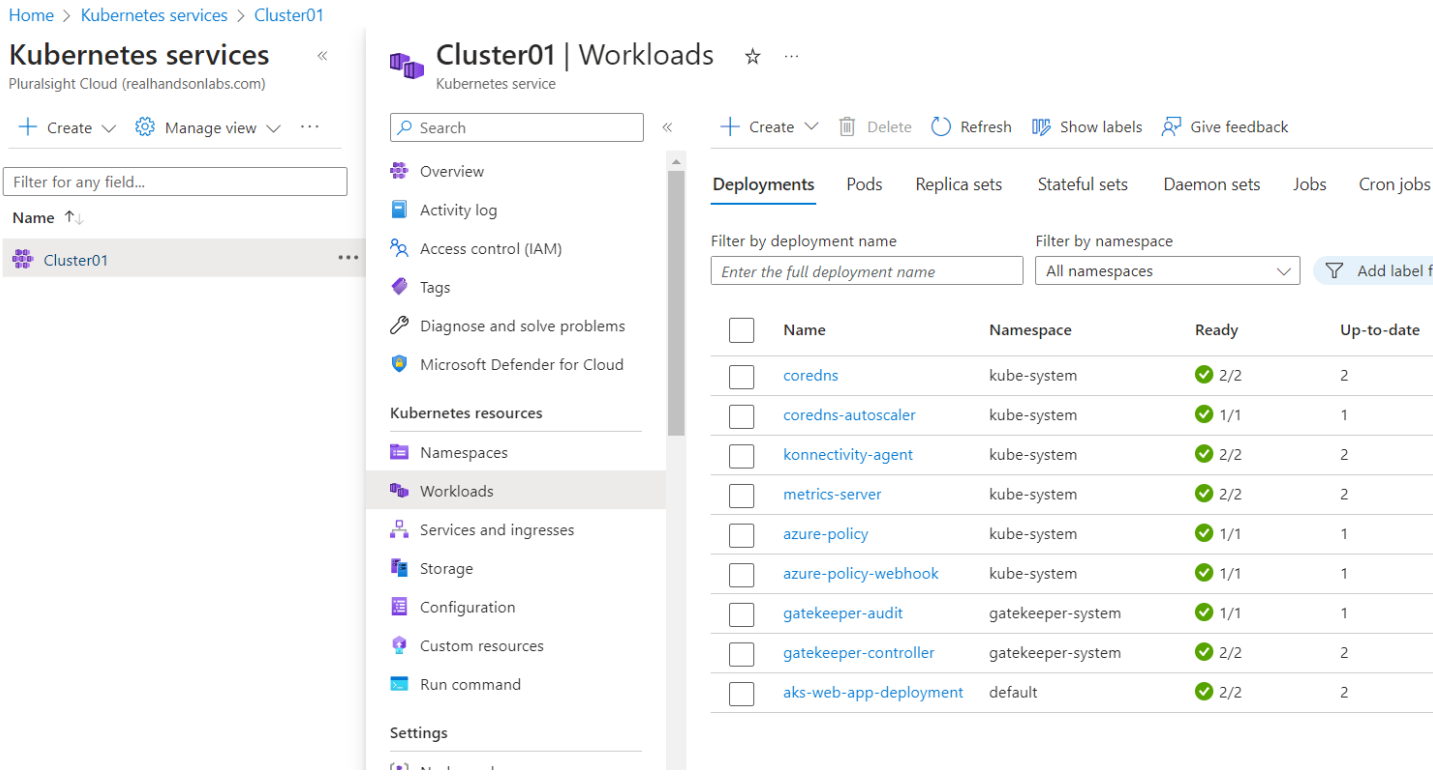


Click the menu icon in the top right (...), and click Close Editor.

Make sure to save the file. Use **ctrl + S**
Name the file **deployment.yaml.**

Deploy the application:
kubectl apply -f ./deployment.yaml

In the Azure Portal, under Kubernetes services, open **Cluster01**.
In the left-hand menu, under **Kubernetes Services**, click **Workloads**.



In the **Deployments** tab, open the new deployment, and verify that the pods are running.

Access the Application Externally

Create a new file named **service.yaml**:

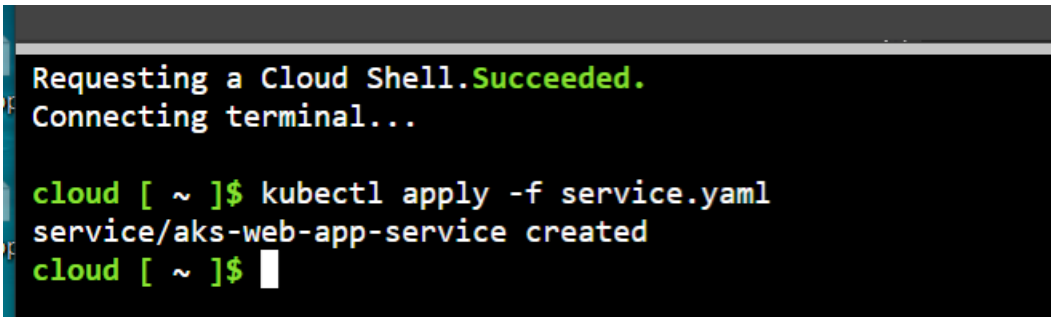
touch service.yaml

Open the editor by clicking the curly braces **{ }**, and open the **service.yaml** file.
Copy and paste the contents of the **service.yaml** file from the **GitHub** repo, into the new file.
Click the menu icon in the top right (...), and click Close Editor.

Make sure to save the file. Use **ctrl + S**
Name the file **service.yaml**

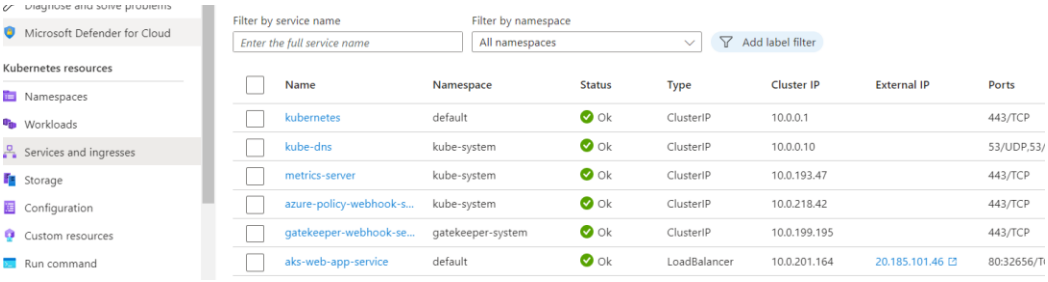
Deploy the LoadBalancer service:

kubectl apply -f service.yaml

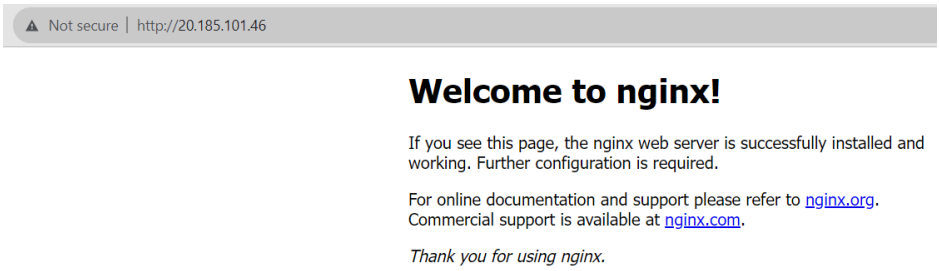


Back in the **Azure** Portal, go to back to **Kubernetes serv**ices. In **Cluster01**, click **Services and ingresses**.

Open the newly deployed **LoadBalancer**.



Click the External IP and verify access to the application.

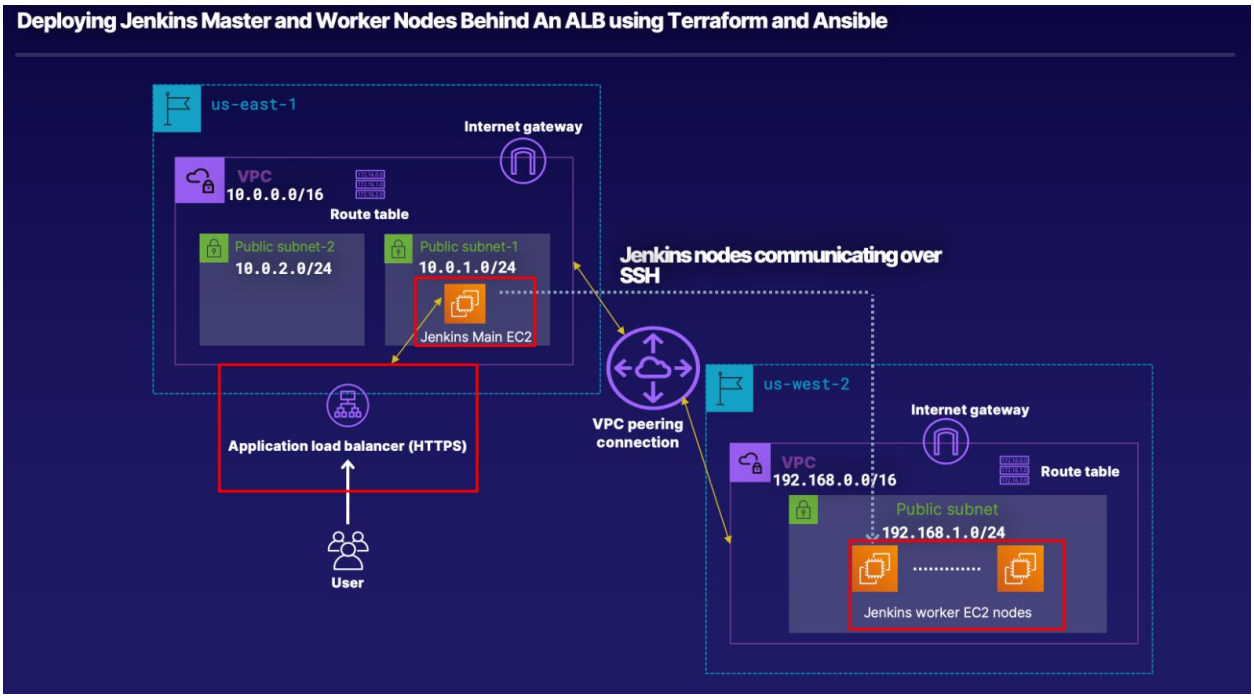


Addendum
GitHub Repo files are located as located as follows:

https://github.com/linuxacademy/content-aks-basics/tree/master/Lab-Deploying-and-accessing-an-application-to-an-AKS-cluster

Deploying Jenkins Master and Worker Nodes in AWS

Behind an ALB Using Terraform and Ansible



Solution

Pre-Steps:

1. Go and setup within an AWS environment (SANDBOXES are fine) a new EC2 instance within EC2.

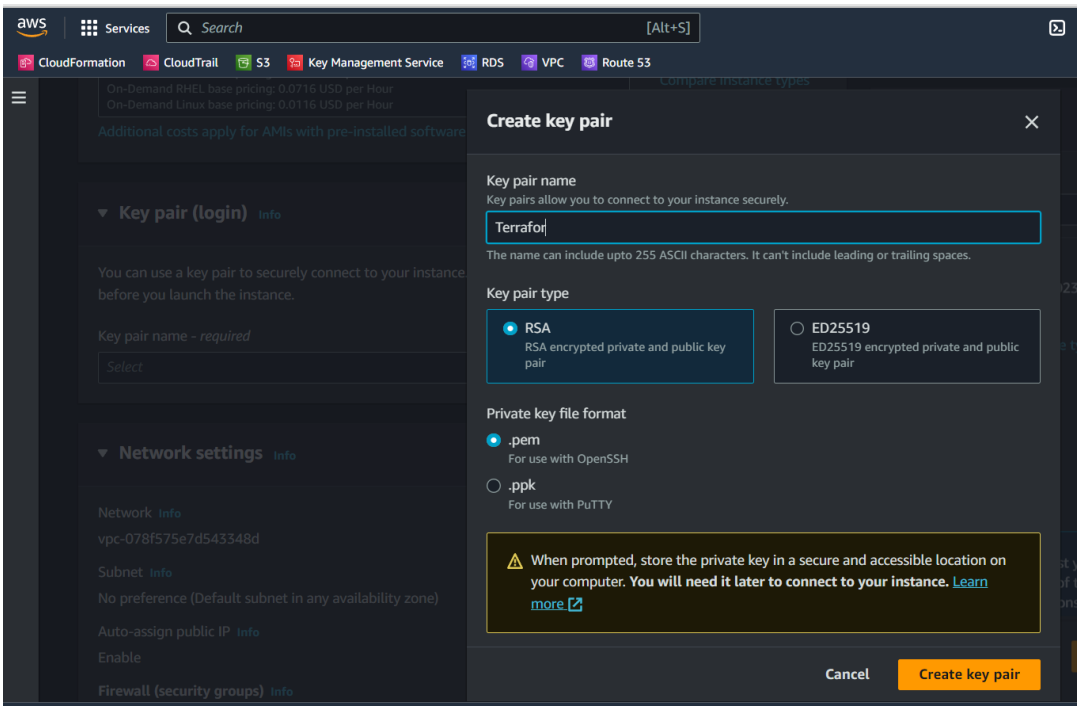
Click on **Launch Instance**

Choose one of the free tier Amazon Linux machines.

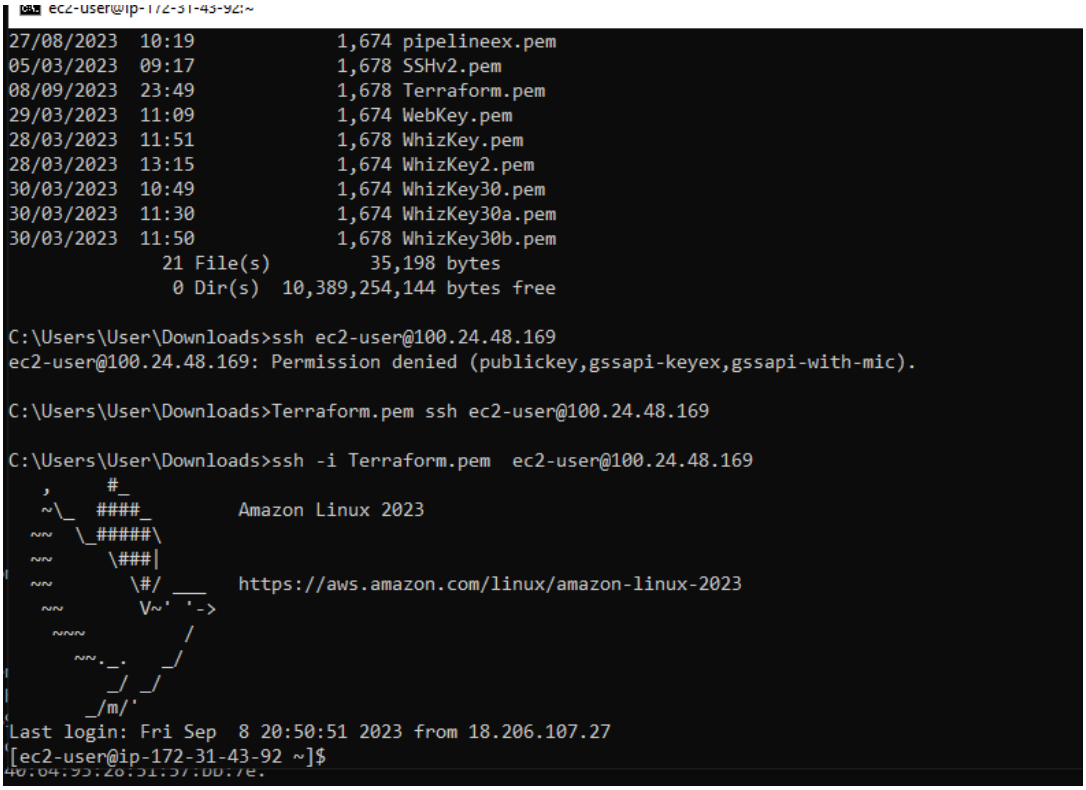
Ensure you also create a key pair. Call it "**Terraform**".

Click on **Create Key Pair**

Scroll to the bottom of the webpage and create the instance.



2. Launch a **command prompt** from your local system and login to the EC2 instance using the command below.



ssh -i /path/to/your/Terraform.pem ec2-user@<Public IP of the EC2 Instance you just created>

Run the following **cat** command:

cat /etc/os-release

[cloud_user@ip-10-0-1-79 ~]$ **cat /etc/os-release**

NAME="Amazon Linux"

VERSION="2"

ID="amzn"

ID_LIKE="centos rhel fedora"

VERSION_ID="2"

PRETTY_NAME="Amazon Linux 2"

ANSI_COLOR="0;33"

CPE_NAME="cpe:2.3:o:amazon:amazon_linux:2"

HOME_URL="https://amazonlinux.com/"

[cloud_user@ip-10-0-1-79 ~]$

3. Check if Python is installed

Use the following command: **python3 --version**

NOTE: It is important to ensure that both **PIP** and **botocore boto3** libraries are installed on this system.

Please follow these steps:

sudo yum update

sudo yum install python3

sudo yum install python3-pip

pip install botocore boto3

python3 --version

NOTE: if **botocore boto3** libraries are missing than Jenkins will result in a Gateway 502 error!

4. Install Ansible using PIP

sudo pip3 install ansible

You may get the following warning:

WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead:

https://pip.pypa.io/warnings/venv

Ignore the above warning.

Issue the following to confirm the installation of ansible:

ansible --version

5. Install Git

sudo yum install git

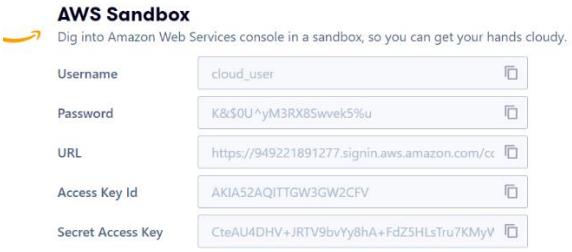
Y

6. Authenticate to AWS using **aws configure**

Type **aws configure** and press <enter>.

Provide both the **Access Key ID** and **Secret Access key ID** went prompted.

Accept the **defaults** for the other prompts.



NOTE: if you don't have both the Access key ID and Secret Access Key you can go to IAM within the AWS console and generate them. Please review earlier projects for The process.

7. Install Terraform

Check if Terraform is installed on the system with the follow command:

terraform version

Note: The Terraform version should be **12.13** . If it's a different version there will be issues when processing the Terraform files. There will be different errors that will require editing of the code.

To avoid such headaches simply ensure the version above is installed (even if you have to remove another version).

Uninstall any existing Terraform version.

sudo rm /usr/local/bin/terraform

Download Terraform v0.12.13 for Linux (adjust for your OS)

wget https://releases.hashicorp.com/terraform/0.12.13/terraform_0.12.13_linux_amd64.zip

Unzip the downloaded file

unzip terraform_0.12.13_linux_amd64.zip

Move the Terraform binary to /usr/local/bin to make it globally accessible

sudo mv terraform /usr/local/bin/

Verify the installation

terraform version

8. Download git files and deploy the Terraform code

Clone the GitHub Repo for Terraform Code

a. Use the git command to clone the GitHub repo which has the Terraform code for deploying the solution.

Execute the following command:

git clone https://github.com/ACloudGuru-Resources/content-deploying-to-aws-ansible-terraform.git

b. Generate a key pair using Ansible

cd content-deploying-to-aws-ansible-terraform/lab_jenkins_master_worker

Examine the contents of the directory you're in:

Ls

Run the `gen_ssh_key.yaml` Ansible Playbook to generate a SSH Key Pair

ansible-playbook ansible_templates/gen_ssh_key.yaml

This Ansible Playbook will generate an SSH key pair for your user cloud_user which is required for deploying EC2 key pairs in our code.

Note: Alternatively, you may also run the following Linux command to do the same:

ssh-keygen -t rsa

Note: When this above command prompts for input, keep pressing enter until you're returned to the prompt.

Do not enter a passphrase.

c. **Deploy the Terraform Code**

terraform init

terraform fmt

terraform validate

terraform plan

terraform apply

Enter **yes** when prompted.

After terraform apply has run successfully, you can either use the AWS CLI on the Controller node to list and describe created resources or you can log in to the AWS Console to verify and investigate created resources.

After a successful terraform apply, you will get the DNS URL of the ALB.

Test it out to see if you can reach your Jenkins deployment.

Apply complete! Resources: 3 added, 0 changed, 1 destroyed.

Outputs:

Jenkins-Master-AMI-ID = ami-0e1c5d8c23330dee3

Jenkins-Master-Private-IP = 10.0.1.217

Jenkins-Worker-AMI-ID = ami-04288abc8d2000768

Jenkins-Worker-Private-IPs = {

"i-0cd8f79141ddd30c4" = "192.168.1.137"

}

Jenkins-Worker-Public-IPs = {

"i-0cd8f79141ddd30c4" = "18.237.211.240"

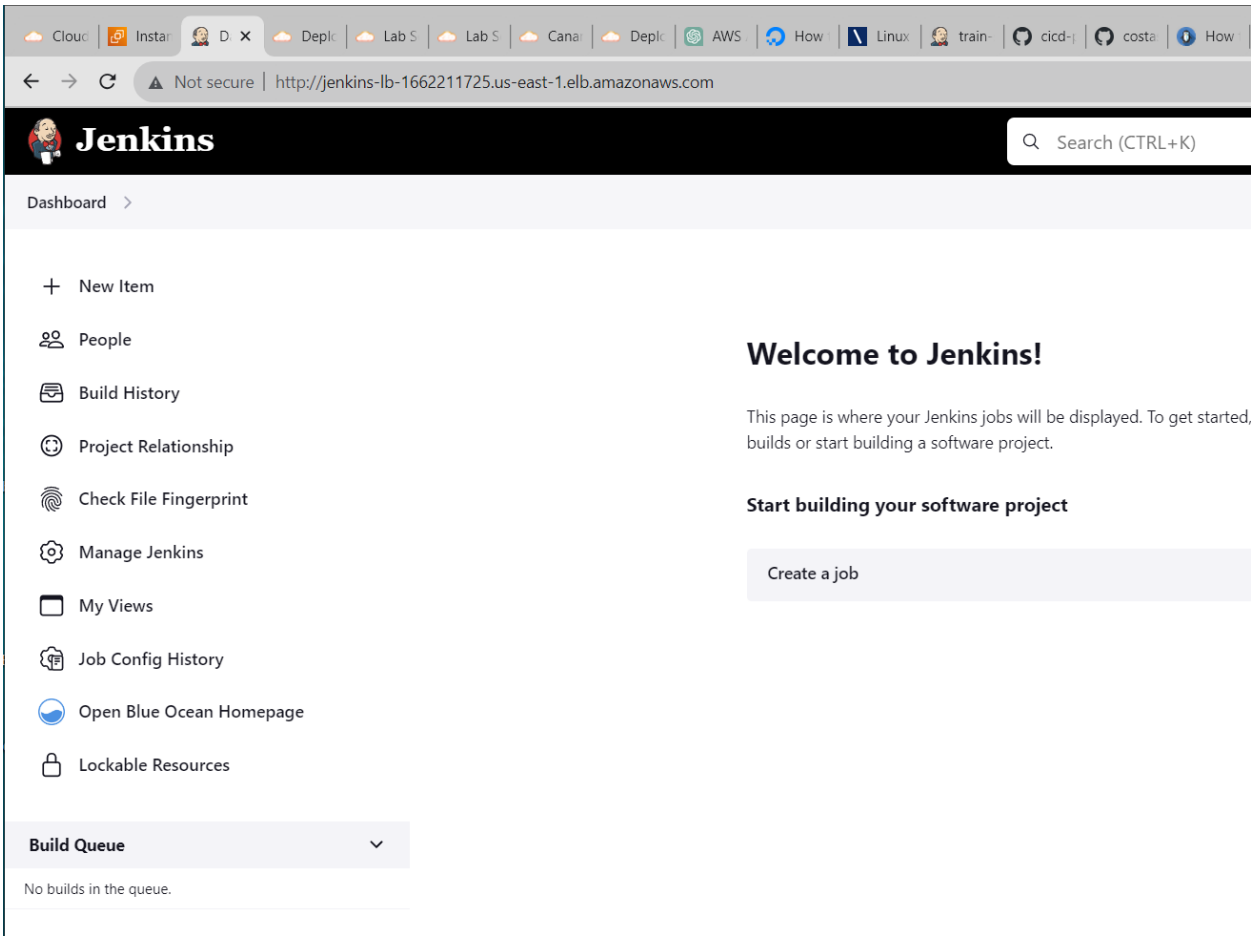
}

Loadbalancer-DNS-URL = **jenkins-lb-22632709.us-east-1.elb.amazonaws.com**

d. Use the following Jenkins credentials to login:

username: admin

password: password



e. Finally, on the Terraform Controller node CLI, delete all resources which were created and ensure that it runs through successfully.

terraform destroy

Addendum

1. If you fail to install the Python libraries above using PIP you will get the following errors:

aws_instance.jenkins-worker-oregon[0] (local-exec): [WARNING]: * Failed to parse /home/ec2-user/content-deploying-to-aws-ansible-t
aws_instance.jenkins-worker-oregon[0] (local-exec): erraform/lab_jenkins_master_worker/ansible_templates/inventory_aws/tf_aws_ec2.y
aws_instance.jenkins-worker-oregon[0] (local-exec): ml with auto plugin: Failed to import the required Python library (botocore and
aws_instance.jenkins-worker-oregon[0] (local-exec): boto3) on ip-172-31-43-130.ec2.internal's Python /usr/bin/python3. Please read
aws_instance.jenkins-worker-oregon[0] (local-exec): the module documentation and install it in the appropriate location. If the
aws_instance.jenkins-worker-oregon[0] (local-exec): required library is installed, but Ansible is using the wrong Python
aws_instance.jenkins-worker-oregon[0] (local-exec): interpreter, please consult the documentation on ansible_python_interpreter
aws_instance.jenkins-worker-oregon[0] (local-exec): [WARNING]: * Failed to parse /home/ec2-user/content-deploying-to-aws-ansible-t
aws_instance.jenkins-worker-oregon[0] (local-exec): erraform/lab_jenkins_master_worker/ansible_templates/inventory_aws/tf_aws_ec2.y
aws_instance.jenkins-worker-oregon[0] (local-exec): ml with yaml plugin: Plugin configuration YAML file, not YAML inventory
aws_instance.jenkins-worker-oregon[0] (local-exec): [WARNING]: * Failed to parse /home/ec2-user/content-deploying-to-aws-ansible-t
aws_instance.jenkins-worker-oregon[0] (local-exec): erraform/lab_jenkins_master_worker/ansible_templates/inventory_aws/tf_aws_ec2.y
aws_instance.jenkins-worker-oregon[0] (local-exec): ml with ini plugin: Invalid host pattern '---' supplied, '---' is normally a
aws_instance.jenkins-worker-oregon[0] (local-exec): sign this is a YAML file.
aws_instance.jenkins-worker-oregon[0] (local-exec): [WARNING]: Unable to parse /home/ec2-user/content-deploying-to-aws-ansible-terr
aws_instance.jenkins-worker-oregon[0] (local-exec): aform/lab_jenkins_master_worker/ansible_templates/inventory_aws/tf_aws_ec2.yml
aws_instance.jenkins-worker-oregon[0] (local-exec): as an inventory source
aws_instance.jenkins-worker-oregon[0] (local-exec): [WARNING]: No inventory was parsed, only implicit localhost is available

aws_instance.jenkins-worker-oregon[0] (local-exec): [WARNING]: provided hosts list is empty, only localhost is available. Note that
aws_instance.jenkins-worker-oregon[0] (local-exec): the implicit localhost does not match 'all'
aws_instance.jenkins-worker-oregon[0] (local-exec): [WARNING]: Could not match supplied host pattern, ignoring:
aws_instance.jenkins-worker-oregon[0] (local-exec): tag_Name_jenkins_worker_tf_1

The above was the Output at the end of **Terraform Apply**

It will generate a load balance string similar to the following.

DNS-URL = **jenkins-lb-22632709.us-east-1.elb.amazonaws.com**

However, when you load it into a web browser it will generate a Gateway 502 error!

2. If you fail to login with aws configure you will get the following during the Terraform Plan command.

ec2-user@ip-172-31-43-92 lab_jenkins_master_worker]$ **terraform plan**

Refreshing Terraform state in-memory prior to plan...

The refreshed state will be used to calculate this plan, but will not be

persisted to local or remote state storage.

Error: error configuring Terraform AWS Provider: **no valid credential sources** for Terraform AWS

Provider found.

Please see https://registry.terraform.io/providers/hashicorp/aws

for more information about providing credentials.

Error: NoCredentialProviders: no valid providers in chain. Deprecated.

For verbose messaging see aws.Config.CredentialsChainVerboseErrors

on network_setup.tf line 1, in provider "aws":

1: provider "aws" {

Error: error configuring Terraform AWS Provider: no valid credential sources for Terraform AWS Provider found.

Please see https://registry.terraform.io/providers/hashicorp/aws

for more information about providing credentials.

Error: NoCredentialProviders: no valid providers in chain. Deprecated.

For verbose messaging see aws.Config.CredentialsChainVerboseErrors

on network_setup.tf line 7, in provider "aws":

7: provider "aws" {

3. If you have another version of Terraform than the one mentioned above you will get the following errors when you run Terraform Init:

Error: Invalid reference from destroy provisioner

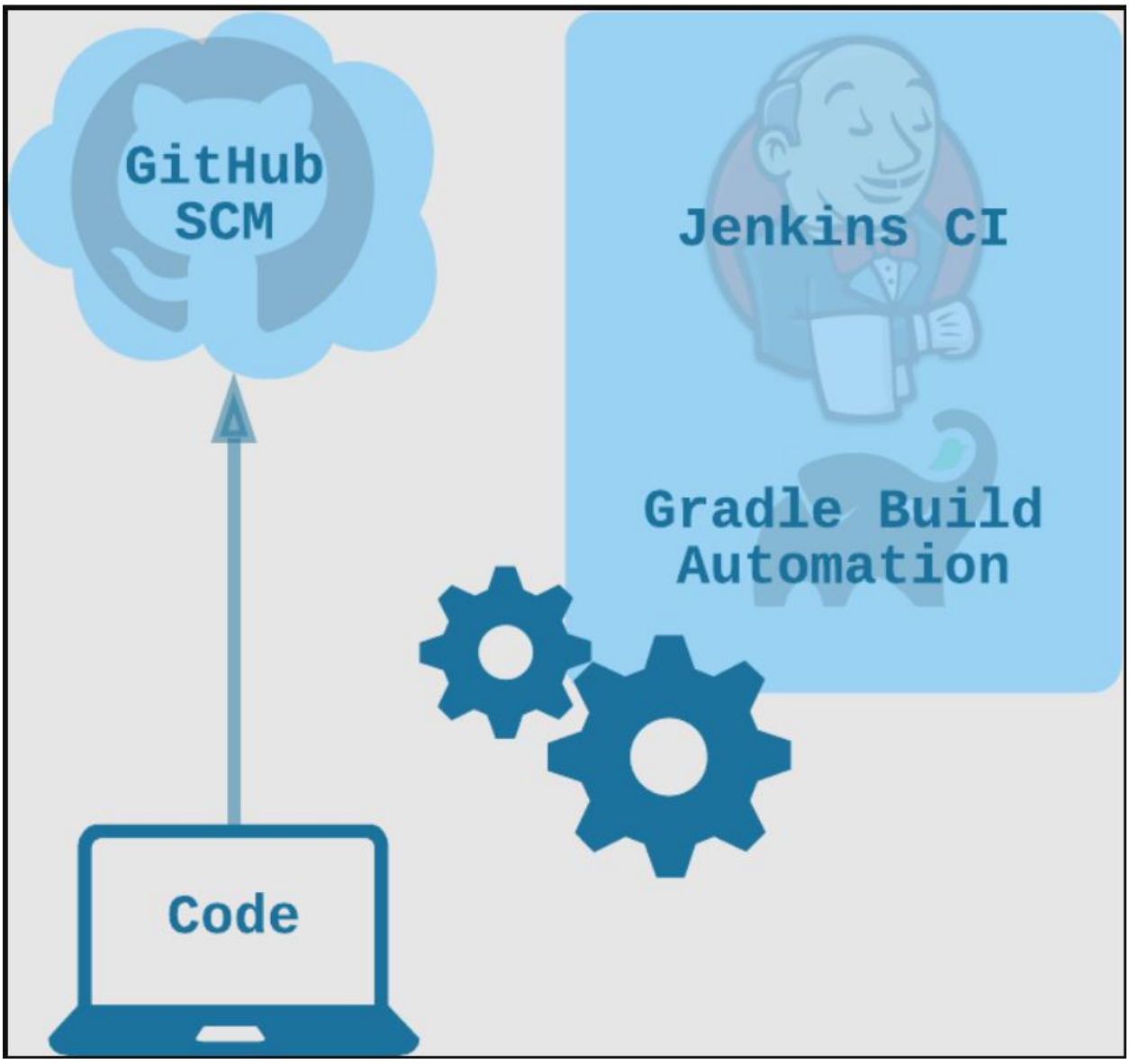
|

| on instances.tf line 60, in resource "aws_instance" "jenkins-worker-oregon":

| 60: inline = [

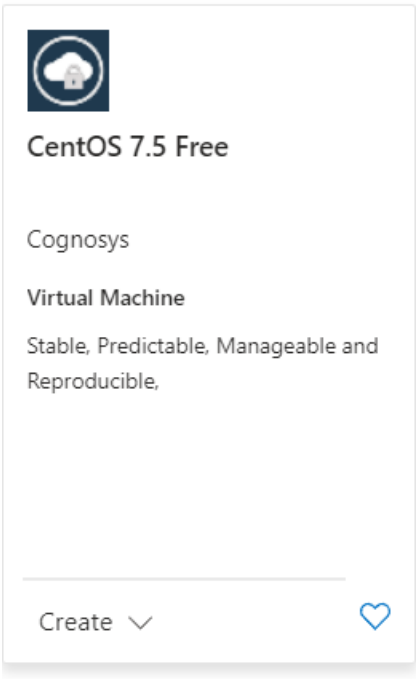
```
|   61:       "java -jar /home/ec2-user/jenkins-cli.jar -auth @/home/ec2-user/jenkins_auth -s
http://${aws_instance.jenkins-master.private_ip}:8080 -auth @/home/ec2-user/jenkins_auth delete-
node ${self.private_ip}"
|   62:     ]
|
| Destroy-time provisioners and their connection configurations may only reference attributes of the
related resource, via 'self',
| 'count.index', or 'each.key'.
|
| References to other resources during the destroy phase can cause dependency cycles and interact
poorly with create_before_destroy.
'
[cloud_user@9c438d91c5a44174a4a8c7a5cd9beb884c lab_jenkins_master_worker]$
```

Deploying a Docker Container with Jenkins Pipelines



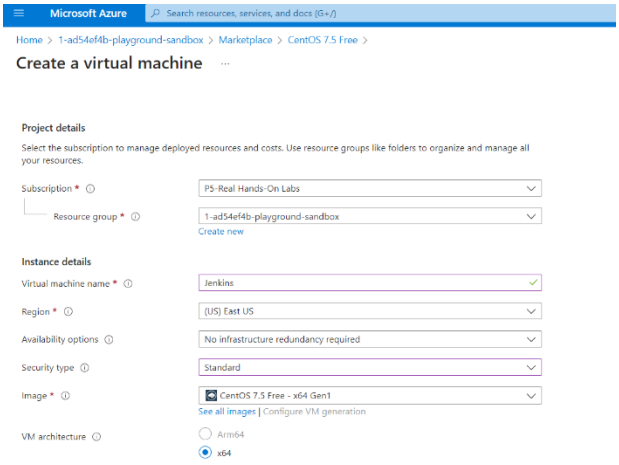
Pre-Steps:

1. Setup two (FREE) Centos 7.5 servers on the Azure portal.

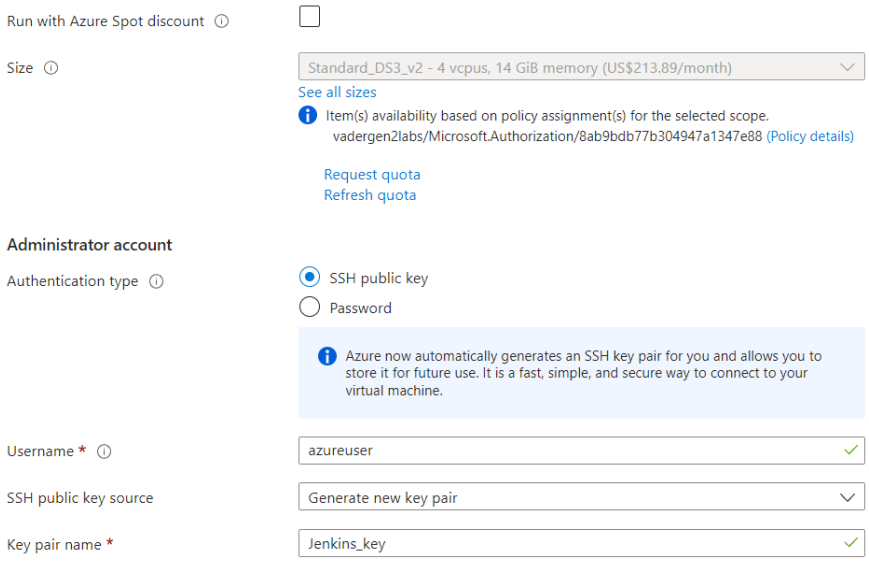


On one we will Install Jenkins.

On the other we will setup our production server that we will deploy the Docker Container.

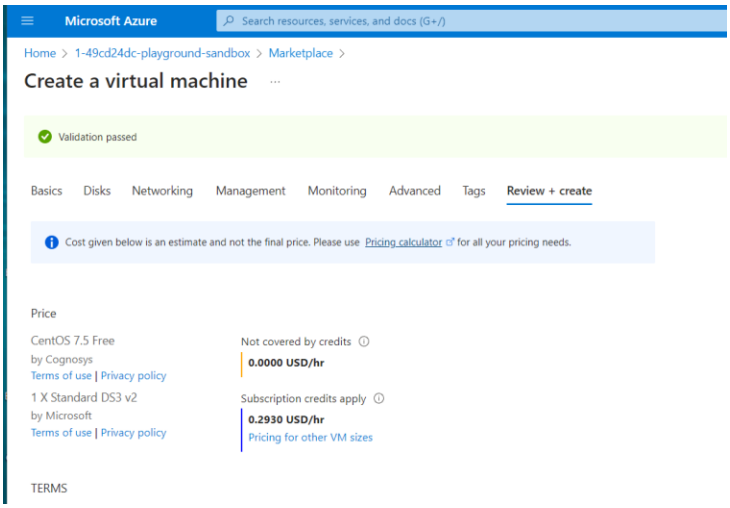


Give the Jenkins server a name (e.g., Jenkins)

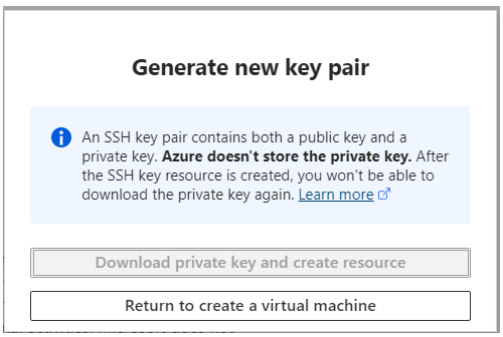


The options for both servers will likely be as above. It depends, in part, on your subscription.

The key option to ensure you select **SSH public key source** radio button as the **authentication type**. This is because we intent to remote into the virtual machines from our local workstation.



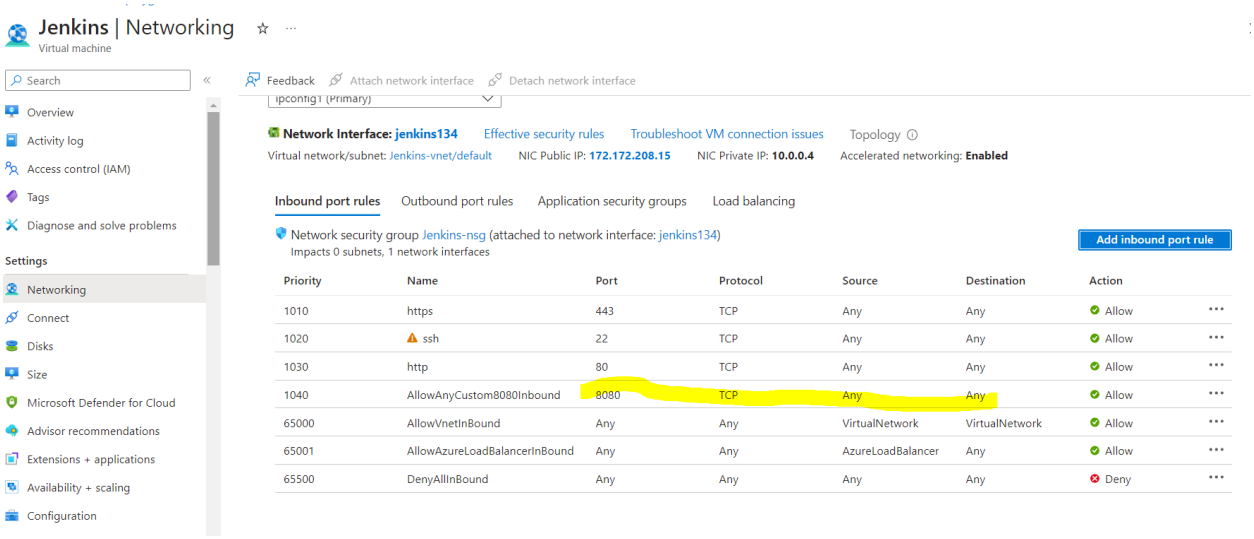
When the Validation passes as above click **create**.



When you select the **Download private key and create resource** button this will generate a new key pair to help you securely login to the server remotely!

When the virtual machines are generated select each, one at a time, and create a Network interface inbound rule. This function is located by going to **Networking** under **Settings**.

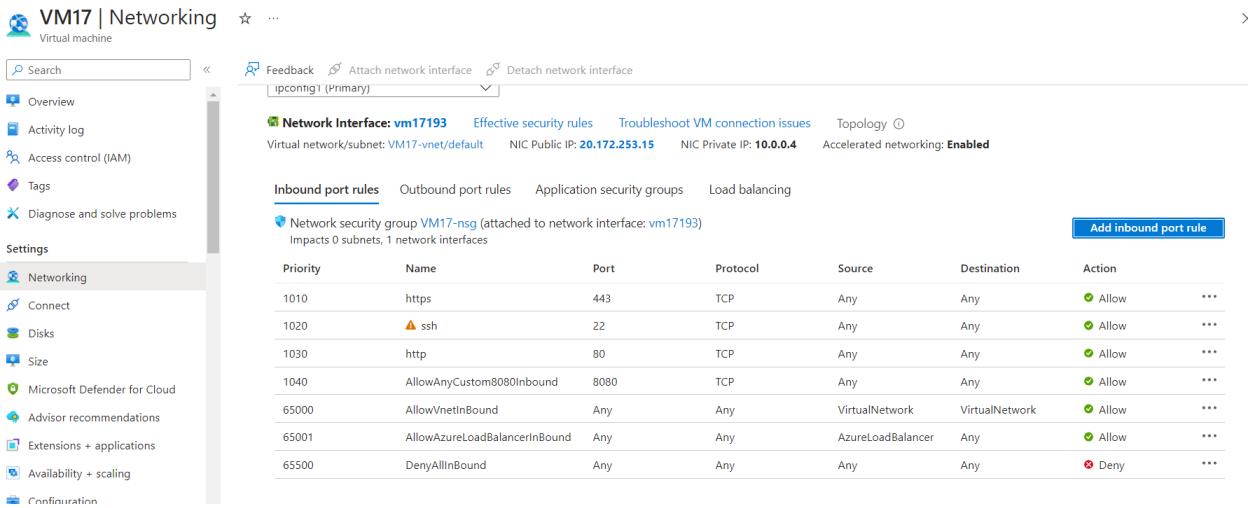
The port rule you will create for each server is to enable port 8080 from any source are as shown below.



The port 8080 rule in the case of the Jenkins server is to allow it to communicate externally. Failure to do so will prevent

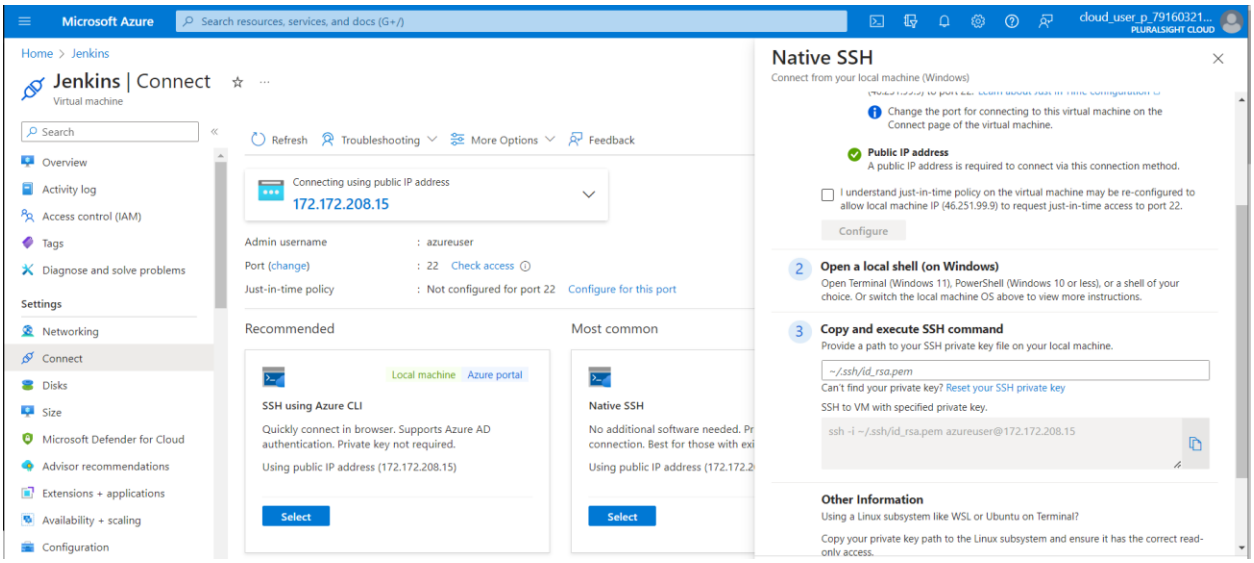
you from loading the Jenkins web portal.

<Jenkins server public IP address>:8080



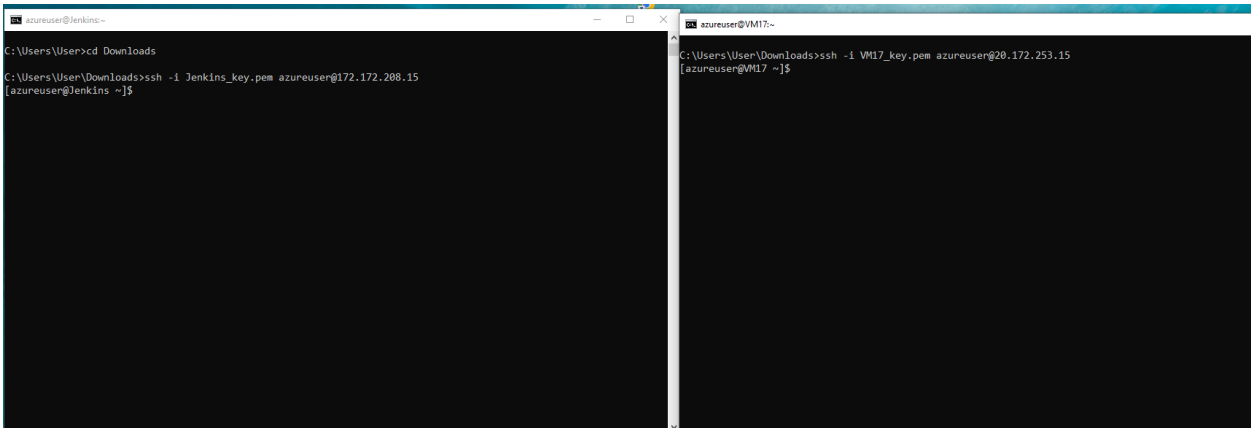
Follow the same process to create similar rule for the Production server. Failure, to do so will prevent the Train Schedule application loading.

<Production server public IP address>:8080



Once the networking rule is completed, go to Connect under Settings. Locate the **Native SSH** option and click **Select**.

Follow the instructions to launch a SSH connection with the server.



Note: please ensure that the command above points to where the .PEM key pair file resides.

```
Host '44.212.64.255' is not in the trusted hosts file.
(ssh-ed25519 fingerprint sha1!! 18:50:bb:a7:54:fb:af:fe:44:10:bb:22:57:c8:26:4a:16:e1:8a:e9)
Do you want to continue connecting? (y/n) y
cloud_user@44.212.64.255's password:
[cloud_user@ip-10-0-1-162 ~]$ cat /etc/os-release
NAME="CentOS Linux"
VERSION="7 (Core)"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="7"
PRETTY_NAME="CentOS Linux 7 (Core)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:7"
HOME_URL="https://www.centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"

CENTOS_MANTISBT_PROJECT="CentOS-7"
CENTOS_MANTISBT_PROJECT_VERSION="7"
REDHAT_SUPPORT_PRODUCT="centos"
REDHAT_SUPPORT_PRODUCT_VERSION="7"

[cloud_user@ip-10-0-1-162 ~]$
```

The **cat /etc/os-release** commands should confirm the Linux OS version.

2. Commands to run on the Jenkins server using SSH

Install Jenkins

Update the server:

sudo yum update

Install Java and the epel-release package:

sudo yum install -y epel-release java-11-openjdk

Configure the Jenkins YUM repository:

sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat-stable/jenkins.repo

Install the Jenkins key:

sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key

Install Jenkins:

sudo yum -y install jenkins

Enable Jenkins:

sudo systemctl enable jenkins

Start Jenkins:

sudo systemctl start Jenkins

Check the status of Jenkins:

sudo systemctl status jenkins

Post Jenkins Install

In a browser, navigate to the Jenkins IP address:

JENKINS_PUBLIC_IP:**8080**

On the server, find the temporary password:

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

Copy the temporary password.

In the browser, paste in the temporary admin password into the box provided.

Click **Install** suggested plugins.

Enter details for an "**admin**" user.

Click **Save** and **Continue**.

Click **Save** and **Finish**.

Click Start using Jenkins.

Install Docker Plugins

Manage Jenkins > **Plugins**

Go to **available** plugs. Search for Docker plugs.

Select all the Docker plugins.

Click on **Install**.

Select the option to reboot Jenkins after the install.

When the installation finishes go back to the SSH command prompt.

Install Git

sudo yum update

sudo yum install git

git --version

Install Docker

sudo yum update

sudo yum install -y yum-utils device-mapper-persistent-data lvm2

sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

sudo yum install -y docker-ce

sudo systemctl start docker

sudo systemctl enable docker

sudo docker –version

Set permissions for Docker

docker run hello-world

docker: Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post http://%2Fvar%2Frun%2Fdocker.sock/v1.38/containers/create: dial unix /var/run/docker.sock: connect: permission denied.

See 'docker run --help'.

sudo chmod 666 /var/run/docker.sock

docker run hello-world

Unable to find image 'hello-world:latest' locally

latest: Pulling from library/hello-world

1b930d010525: Pull complete

Digest: sha256:c3b4ada4687bbaa170745b3e4dd8ac3f194ca95b2d0518b417fb47e5879d9b5f

Status: Downloaded newer image for hello-world:latest

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it

to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:

https://hub.docker.com/

For more examples and ideas, visit:

https://docs.docker.com/get-started/

Install sshpass

Install sshpass

sudo yum install sshpass

3.Commands to run on the Production server using SSH

Update the server

sudo yum update

Install Java and the epel-release package

sudo yum install -y epel-release java-11-openjdk

Install Git

sudo yum update

sudo yum install git

git --version

Install Docker

sudo yum update

sudo yum install -y yum-utils device-mapper-persistent-data lvm2

sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

sudo yum install -y docker-ce

sudo systemctl start docker

sudo systemctl enable docker

sudo docker --version

Set permissions for Docker

When you run the following command on docker, by default, you will get the a permission denied error!

docker run hello-world

docker: Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post http://%2Fvar%2Frun%2Fdocker.sock/v1.38/containers/create: dial unix /var/run/docker.sock: connect: permission denied.

See 'docker run --help'.

sudo chmod 666 /var/run/docker.sock

docker run hello-world

Unable to find image 'hello-world:latest' locally

latest: Pulling from library/hello-world

1b930d010525: Pull complete

Digest: sha256:c3b4ada4687bbaa170745b3e4dd8ac3f194ca95b2d0518b417fb47e5879d9b5f

Status: Downloaded newer image for hello-world:latest

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:

https://hub.docker.com/

For more examples and ideas, visit:

https://docs.docker.com/get-started/

Setup the Deploy account for Jenkins on the Production server.

[azureuser@vm16 ~]$ **whoami**

azureuser

[azureuser@vm16 ~]$

Create a User:

sudo useradd -m deploy

Set a password:

sudo passwd deploy

Enter password

Confirm password

Note: we will use this later on the Jenkins portal.

List all user accounts

cat /etc/passwd

Host '54.210.24.6' is not in the trusted hosts file.

(ssh-ed25519 fingerprint sha1!! 5c:11:16:0b:b6:77:19:f9:aa:7c:64:01:02:c3:4d:97:e2:03:50:78)

```
Do you want to continue connecting? (y/n) y
cloud_user@54.210.24.6's password:
[cloud_user@ip-10-0-1-89 ~]$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
systemd-bus-proxy:x:999:998:systemd Bus Proxy:/:/sbin/nologin
systemd-network:x:192:192:systemd Network Management:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:998:997:User for polkitd:/:/sbin/nologin
tss:x:59:59:Account used by the trousers package to sandbox the tcsd daemon:/dev/null:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
chrony:x:997:995::/var/lib/chrony:/sbin/nologin
```

cloud_user:x:1000:1000:Cloud Assessment User:/home/cloud_user:/bin/bash

centos:x:1001:1001:Cloud User:/home/centos:/bin/bash

jenkins:x:996:993:Jenkins Automation Server:/var/lib/jenkins:/bin/false

deploy:x:1002:1002::/home/deploy:/bin/bash

[cloud_user@ip-10-0-1-89 ~]$

Setup permissions for the Jenkins server so that it can SSH to the Production server.

When you run the following command you will get, by default, a permissions denied error as below

sshpass -p '<password for the deploy account on the production server>' -v ssh -o

StrictHostKeyChecking=no deploy@<production server Public IP Address>

[azureuser@Jenkins ~]$ sudo sshpass -p 'Harvee777$' -v ssh -o StrictHostKeyChecking=no

deploy@20.172.253.15

Warning: Permanently added '20.172.253.15' (ECDSA) to the list of known hosts.

Permission denied (publickey,gssapi-keyex,gssapi-with-mic).

[azureuser@Jenkins ~]$

The solution to this is to enable password authentication.

Go to the Production server and use the following commands:

sudo vi /etc/ssh/sshd_config

In the file, find the **PasswordAuthentication** line and make sure it ends with yes.

Find the **ChallengeResponseAuthentication** option and disable it by adding **no**.

If lines are commented out, remove the hash sign **#** to uncomment them.

Switch to command mode by pressing the **Esc** key.

Type **:** (colon). This will open the prompt bar in the bottom left corner of the window.

Type **wq** after the colon and hit Enter. This will save in **Vim** the changes made to the file and exit you out of the editor.

Now restart SSH with the following command:

sudo systemctl restart sshd

Now return to the Jenkins server and issue the same command as before.

sshpass -p '<password for the deploy account on the production server>' -v ssh -o StrictHostKeyChecking=no deploy@<production server Public IP Address>

```
[azureuser@Jenkins ~]$ sudo sshpass -p 'Harvee777$' -v ssh -o StrictHostKeyChecking=no deploy@172.172.155.55
SSHPASS searching for password prompt using match "assword"
SSHPASS read: deploy@172.172.155.55's password:
SSHPASS detected prompt. Sending password.
SSHPASS read:

[deploy@vm777 ~]$ logout
Connection to 172.172.155.55 closed.
[azureuser@Jenkins ~]$
```

It has the effect of creating an SSH session from the Jenkins server to the production server. Then logout to close the session.

The reason for the Jenkins to have these permissions is because it uses the above command to remote to the Production server then issue a docker command to pull the built docker container

locally to the production server.

sshpass -p 'your_password_here' -v ssh -o StrictHostKeyChecking=no deploy@<production server public IP address> 'docker pull costas778/train-schedule'

NOTE: the password above is NOT one I will use for production purposes.

Deploying a Docker Container with a Jenkins Pipeline

We are, finally, in a position to deploy the Docker container using a Jenkins pipeline!

Go to **Manage Jenkins > Credentials**, then click **(global)**.

Click + **Add Credentials**

Create a global Jenkins credential called **deploy**

Kind: Username with password

Username: **deploy**

Password:

ID: **webserver_ID**

Description: **Webserver ID**

Use the same deploy password that you used for the production server.

Click **Create**

Create a global Jenkins credential for the Docker image registry (Docker Hub).

Click + **Add Credentials**:

Kind: Username with password

Username: **<DOCKER_HUB_USERNAME>**

Password: **<DOCKER_HUB_PASSWORD>**

ID: **docker_hub_login**

Description: **Docker Hub Login**

Click **Create**

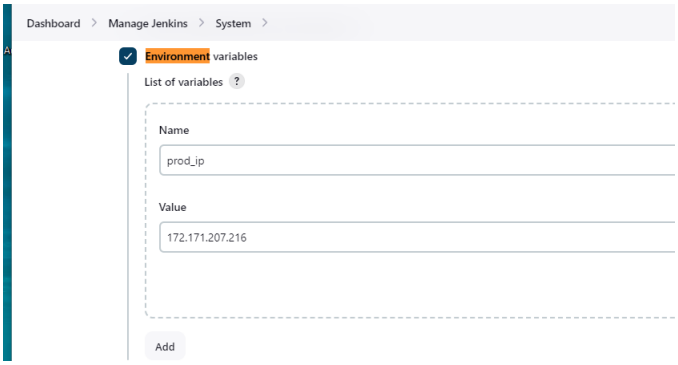
Note: You will need a Docker hub account to use Docker Hub as an image registry.

Configure a global property in Jenkins to store the production server IP by navigating to **Manage Jenkins > System** and adding an **environment variable** under **Global properties**.

Name: **prod_ip**

Value: **<PRODUCTION_SERVER_PUBLIC_IP_ADDRESS>**

Click **Save**



Go to **GitHub**

Make a personal fork from the following GitHub repository:

https://github.com/ACloudGuru-Resources/cicd-pipeline-train-schedule-dockerdeploy

https://github.com/<YOUR_GITHUB_USER>/cicd-pipeline-train-schedule-dockerdeploy

Generate a new **GitHub API key** to allow Jenkins to access the forked repo by navigating to **Profile > Settings > Developer Settings > Personal Access Tokens > Tokens (classic) > Generate new token (classic)**.

Note: **Jenkinsapp**

Permissions: **admin:repo_hook**

Copy the GitHub token.

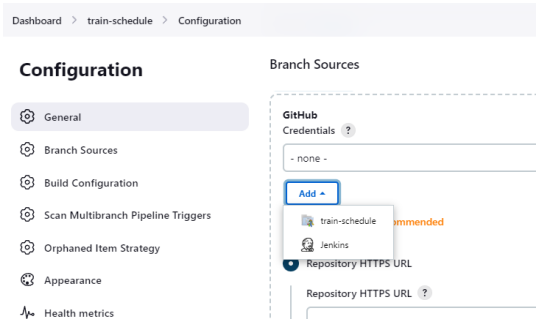
Go back to the Jenkins portal.

Click **New Items**

In Jenkins, create a **Multibranch Pipeline** project named **train-schedule**.

Click **OK**

Under **Branch Sources**, click **Add** below **GitHub**, then choose **Jenkins** to configure your forked repository



Kind: Username with password

Scope: Global

Username: <GITHUB_USERNAME>

Password: <YOUR_GITHUB_PERSONAL_ACCESS_TOKEN>

ID: **github_key**

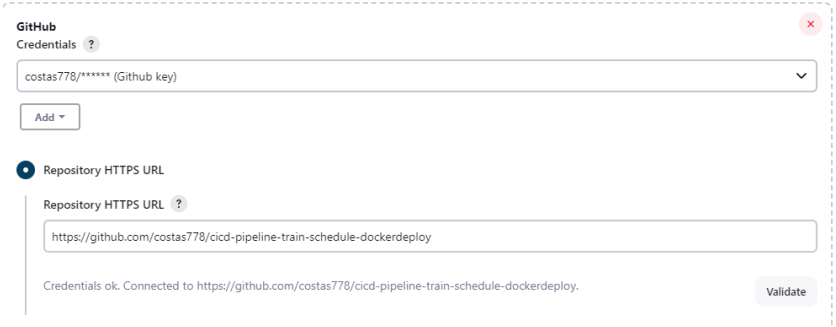
Description: **GitHub Key**

Click **Add**

Select the **GitHub Key** and paste your forked repository under **Repository HTTPS URL**:

https://github.com/<YOUR_GITHUB_USER>/cicd-pipeline-train-schedule-dockerdeploy

Click **Validate** to test your credentials



Click **Save**.

Successfully deploy the train-schedule app to production as a Docker container using the Jenkins Pipeline.

Modify the **Jenkinsfile** in GitHub to build and push the Docker image to Docker Hub and commit the changes. Replace willbla from the example file with your own Docker Hub username.

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                echo 'Running build automation'
                sh './gradlew build --no-daemon'
                archiveArtifacts artifacts: 'dist/trainSchedule.zip'
            }
        }
        stage('Build Docker Image') {
            when {
                branch 'master'
            }
            steps {
                script {
                    app = docker.build("<DOCKER_HUB_USERNAME>/train-schedule")
                    app.inside {
                        sh 'echo $(curl localhost:8080)'
                    }
                }
            }
        }
        stage('Push Docker Image') {
```

```
        when {
          branch 'master'
        }
        steps {
          script {
            docker.withRegistry('https://registry.hub.docker.com', 'docker_hub_login') {
              app.push("${env.BUILD_NUMBER}")
              app.push("latest")
            }
          }
        }
      }
    }
}
```

In Jenkins, click **Build Now**.

Note: The initial build may take several minutes to complete.

In **Docker Hub**, under **Repositories**, select the **train-schedule app**.

Click the **Tags tab** to verify that the build was pushed successfully.

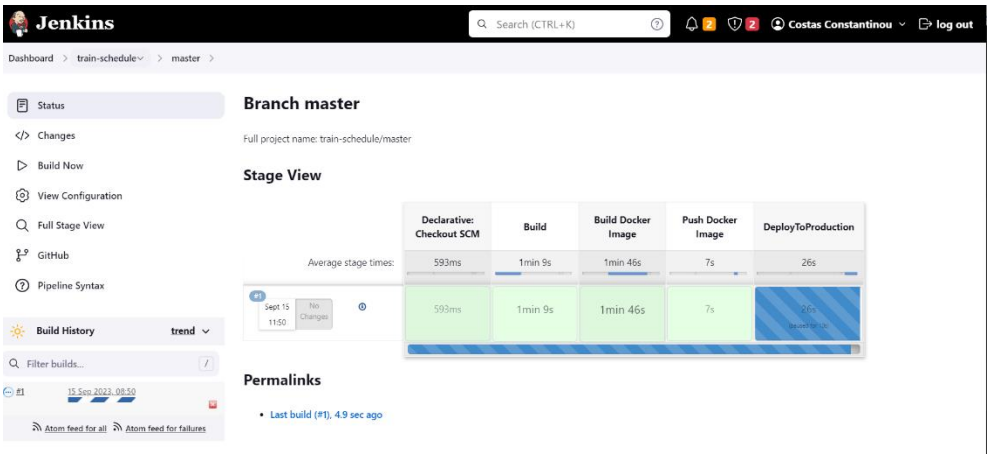
In GitHub, modify the **Jenkinsfile** to include a stage that pushes the build to the production server, and commit the changes.

```
stage ('DeployToProduction') {
   when {
      branch 'master'
   }
   steps {
      input 'Deploy to Production'
      milestone(1)
      withCredentials ([usernamePassword(credentialsId: 'webserver_login', usernameVariable:
'USERNAME', passwordVariable: 'USERPASS')]) {
         script {
            sh "sshpass -p '$USERPASS' -v ssh -o StrictHostKeyChecking=no $USERNAME@${env.prod_ip}
\"docker pull <DOCKER_HUB_USERNAME>/train-schedule:${env.BUILD_NUMBER}\""
            try {
               sh "sshpass -p '$USERPASS' -v ssh -o StrictHostKeyChecking=no $USERNAME@${env.prod_ip}
\"docker stop train-schedule\""
               sh "sshpass -p '$USERPASS' -v ssh -o StrictHostKeyChecking=no $USERNAME@${env.prod_ip}
\"docker rm train-schedule\""
            } catch (err) {
               echo: 'caught error: $err'
            }
            sh "sshpass -p '$USERPASS' -v ssh -o StrictHostKeyChecking=no $USERNAME@${env.prod_ip}
\"docker run --restart always --name train-schedule -p 8080:8080 -d <DOCKER_HUB_USERNAME>/train-
schedule:${env.BUILD_NUMBER}\""
         }
```

```
    }
  }
}
```

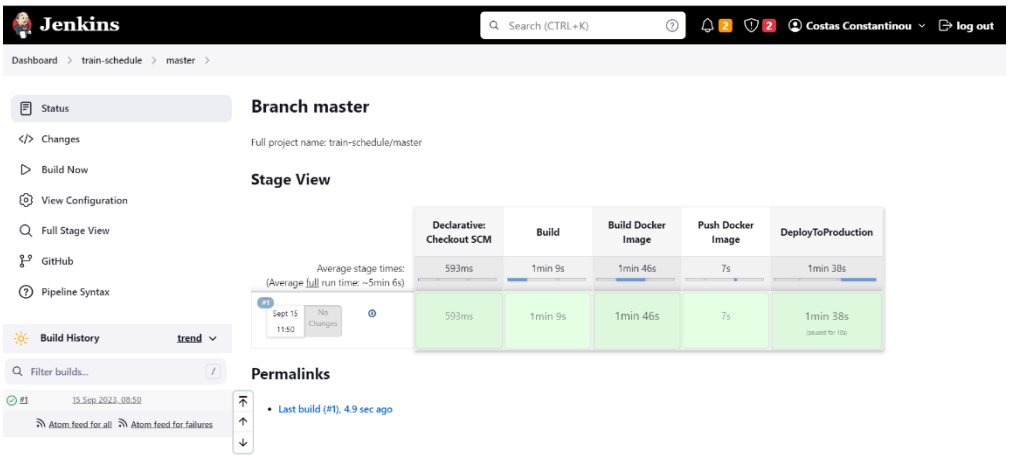
Place your Docker hub username above and **commit** the changes.

In Jenkins, click **Build Now**.

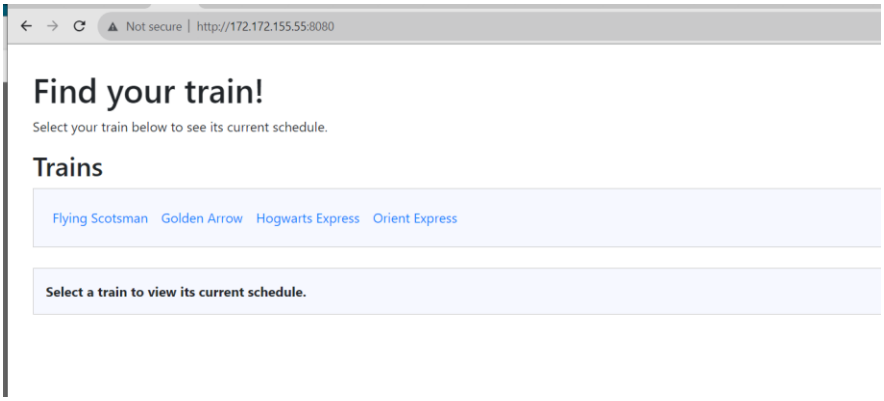


Note: to proceed from Push Docker Image to **DeployToProduction** you will need to hover over Push Docker Image within the green area and allow it to proceed further.

Once the build is complete, using a web browser, verify that the application has been deployed successfully.



Place the following in a web browser **<PRODUCTION_SERVER_PUBLIC_IP_ADDRESS>:8080**



If successful, you will see the above!

Addendum

1) Build Failure

Started by user Costas constantinou

08:44:29 Connecting to https://api.github.com using costas778/****** (Github key)

Obtained Jenkinsfile from 248d9bbd2a5c031c08db8c6b21a28f6336cc4173

```
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/train-schedule_master
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
The recommended git tool is: NONE
using credential github_key
 > git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/train-schedule_master/.git # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/costas778/cicd-pipeline-train-schedule-
dockerdeploy.git # timeout=10
Fetching without tags
Fetching upstream changes from https://github.com/costas778/cicd-pipeline-train-schedule-
dockerdeploy.git
 > git --version # timeout=10
 > git --version # 'git version 1.8.3.1'
using GIT_ASKPASS to set credentials Github key
 > git fetch --no-tags --progress https://github.com/costas778/cicd-pipeline-train-schedule-
dockerdeploy.git +refs/heads/master:refs/remotes/origin/master # timeout=10
Checking out Revision 248d9bbd2a5c031c08db8c6b21a28f6336cc4173 (master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 248d9bbd2a5c031c08db8c6b21a28f6336cc4173 # timeout=10
```

Commit message: "Update Jenkinsfile"

> git rev-list --no-walk 33c78dcc8603870d855e4caa9653c89b7848b739 # timeout=10

[Pipeline] }

[Pipeline] // stage

[Pipeline] withEnv

[Pipeline] {

[Pipeline] stage

[Pipeline] { (Build)

[Pipeline] echo

Running build automation

[Pipeline] sh

+ ./gradlew build --no-daemon

WARNING: An illegal reflective access operation has occurred

WARNING: Illegal reflective access by org.codehaus.groovy.vmplugin.v7.Java7$1

(file:/var/lib/jenkins/.gradle/wrapper/dists/gradle-6.0.1-bin/1lxlpkiy24sb18odw96cp4ojv/gradle-6.0.1/lib/groovy-all-1.3-2.5.8.jar) to constructor

java.lang.invoke.MethodHandles$Lookup(java.lang.Class,int)

WARNING: Please consider reporting this to the maintainers of

org.codehaus.groovy.vmplugin.v7.Java7$1

WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations

WARNING: All illegal access operations will be denied in a future release

> Task :nodeSetup UP-TO-DATE

> Task :npmSetup UP-TO-DATE

> Task :npmInstall UP-TO-DATE

```
> Task :npm_test

> cicd-pipeline-train-schedule-git@0.0.0 test /var/lib/jenkins/workspace/train-schedule_master

> mocha

  Index Page

[0mGET / [32m200 [0m186.268 ms - 829[0m

    ✓ renders successfully (205ms)


  Trains API

[0mGET /trains [32m200 [0m2.163 ms - 1093[0m

    ✓ returns data successfully

  2 passing (227ms)

> Task :npm_build

> Task :zip

> Task :build


Deprecated Gradle features were used in this build, making it incompatible with Gradle 7.0.

Use '--warning-mode all' to show the individual deprecation warnings.

See

https://docs.gradle.org/6.0.1/userguide/command_line_interface.html#sec:command_line_warnings


BUILD SUCCESSFUL in 7s

6 actionable tasks: 3 executed, 3 up-to-date

[Pipeline] archiveArtifacts

Archiving artifacts
```

```
[Pipeline] }

[Pipeline] // stage

[Pipeline] stage

[Pipeline] { (Build Docker Image)

[Pipeline] script

[Pipeline] {

[Pipeline] }

[Pipeline] // script

[Pipeline] }

[Pipeline] // stage

[Pipeline] stage

[Pipeline] { (Push Docker Image)

Stage "Push Docker Image" skipped due to earlier failure(s)

[Pipeline] }

[Pipeline] // stage

[Pipeline] stage

[Pipeline] { (DeployToProduction)

Stage "DeployToProduction" skipped due to earlier failure(s)

[Pipeline] }

[Pipeline] // stage

[Pipeline] }

[Pipeline] // withEnv

[Pipeline] }

[Pipeline] // node
```

[Pipeline] End of Pipeline

Also: org.jenkinsci.plugins.workflow.actions.ErrorAction$ErrorId: 78a67a41-c4f8-405c-a52c-f0f33958a0bf

groovy.lang.MissingPropertyException: No such property: docker for class: groovy.lang.Binding

at groovy.lang.Binding.getVariable(Binding.java:63)

at org.jenkinsci.plugins.scriptsecurity.sandbox.groovy.SandboxInterceptor.onGetProperty(SandboxInterceptor.java:285)

at org.kohsuke.groovy.sandbox.impl.Checker$7.call(Checker.java:375)

at org.kohsuke.groovy.sandbox.impl.Checker.checkedGetProperty(Checker.java:379)

at org.kohsuke.groovy.sandbox.impl.Checker.checkedGetProperty(Checker.java:355)

at org.kohsuke.groovy.sandbox.impl.Checker.checkedGetProperty(Checker.java:355)

at org.kohsuke.groovy.sandbox.impl.Checker.checkedGetProperty(Checker.java:355)

at com.cloudbees.groovy.cps.sandbox.SandboxInvoker.getProperty(SandboxInvoker.java:29)

at org.jenkinsci.plugins.workflow.cps.LoggingInvoker.getProperty(LoggingInvoker.java:121)

at com.cloudbees.groovy.cps.impl.PropertyAccessBlock.rawGet(PropertyAccessBlock.java:20)

at WorkflowScript.run(WorkflowScript:17)

at ___cps.transform___(Native Method)

at com.cloudbees.groovy.cps.impl.PropertyishBlock$ContinuationImpl.get(PropertyishBlock.java:73)

at com.cloudbees.groovy.cps.LValueBlock$GetAdapter.receive(LValueBlock.java:30)

at com.cloudbees.groovy.cps.impl.PropertyishBlock$ContinuationImpl.fixName(PropertyishBlock.java:65)

at jdk.internal.reflect.GeneratedMethodAccessor386.invoke(Unknown Source)

at

java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorI

mpl.java:43)

at java.base/java.lang.reflect.Method.invoke(Method.java:566)

at

com.cloudbees.groovy.cps.impl.ContinuationPtr$ContinuationImpl.receive(ContinuationPtr.java:7

2)

at com.cloudbees.groovy.cps.impl.ConstantBlock.eval(ConstantBlock.java:21)

at com.cloudbees.groovy.cps.Next.step(Next.java:83)

at com.cloudbees.groovy.cps.Continuable$1.call(Continuable.java:152)

at com.cloudbees.groovy.cps.Continuable$1.call(Continuable.java:146)

at

org.codehaus.groovy.runtime.GroovyCategorySupport$ThreadCategoryInfo.use(GroovyCategoryS

upport.java:136)

at org.codehaus.groovy.runtime.GroovyCategorySupport.use(GroovyCategorySupport.java:275)

at com.cloudbees.groovy.cps.Continuable.run0(Continuable.java:146)

at

org.jenkinsci.plugins.workflow.cps.SandboxContinuable.access$001(SandboxContinuable.java:18)

at org.jenkinsci.plugins.workflow.cps.SandboxContinuable.run0(SandboxContinuable.java:51)

at org.jenkinsci.plugins.workflow.cps.CpsThread.runNextChunk(CpsThread.java:187)

at org.jenkinsci.plugins.workflow.cps.CpsThreadGroup.run(CpsThreadGroup.java:422)

at org.jenkinsci.plugins.workflow.cps.CpsThreadGroup$2.call(CpsThreadGroup.java:330)

at org.jenkinsci.plugins.workflow.cps.CpsThreadGroup$2.call(CpsThreadGroup.java:294)

at

org.jenkinsci.plugins.workflow.cps.CpsVmExecutorService$2.call(CpsVmExecutorService.java:97)

at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)

at hudson.remoting.SingleLaneExecutorService$1.run(SingleLaneExecutorService.java:139)

at jenkins.util.ContextResettingExecutorService$1.run(ContextResettingExecutorService.java:28)

at jenkins.security.ImpersonatingExecutorService$1.run(ImpersonatingExecutorService.java:68)

at jenkins.util.ErrorLoggingExecutorService.lambda$wrap$0(ErrorLoggingExecutorService.java:51)

at java.base/java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:515)

at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)

at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1128)

at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:628)

at java.base/java.lang.Thread.run(Thread.java:829)

Could not update commit status, please check if your scan credentials belong to a member of the organization or a collaborator of the repository and repo:status scope is selected

GitHub has been notified of this commit's build result.

Finished: FAILURE

Solution:

Make sure that the **Docker plugins** are installed and configured correctly on your Jenkins server.

2) Failure to Deploy to Production (Stage)

Shell Script -- sshpass -p '${USERPASS}' -v ssh -o StrictHostKeyChecking=no jenkinsad@172.171.207.216 "docker pull costas778/train-schedule:3" (self time 288ms)

Warning: A secret was passed to "sh" using Groovy String interpolation, which is insecure.

Affected argument(s) used the following variable(s): [USERPASS]

See https://jenkins.io/redirect/groovy-string-interpolation for details.

+ sshpass -p **** -v ssh -o StrictHostKeyChecking=no jenkinsad@172.171.207.216 'docker pull costas778/train-schedule:3'

/var/lib/jenkins/workspace/train-schedule_master@tmp/durable-b344b4a1/script.sh: line 1: sshpass: command not found

Solution:

Install SSH with the following command.

sudo yum install sshpass

3) Failure to Deploy To Production (Stage) after installing sshpass

Warning: A secret was passed to "sh" using Groovy String interpolation, which is insecure.

Affected argument(s) used the following variable(s): [USERPASS]

See https://jenkins.io/redirect/groovy-string-interpolation for details.

+ sshpass -p **** -v ssh -o StrictHostKeyChecking=no jenkinsad@172.171.207.216 'docker pull costas778/train-schedule:4'

Warning: Permanently added '172.171.207.216' (ECDSA) to the list of known hosts.

Permission denied (publickey,gssapi-keyex,gssapi-with-mic).

Solution:

Go to sshd_config file on the production server and make the following changes:

sudo vi /etc/ssh/sshd_config

In the file, find the **PasswordAuthentication** line and make sure it ends with yes.

Find the **ChallengeResponseAuthentication** option and disable it by adding no.

If lines are commented out, remove the hash sign # to uncomment them.

Save the changes in the file then restart the **SSH** service for the changes to take effect.

sudo systemctl restart sshd

4) Build error - Failure to download Node with Gradle.

+ ./gradlew build --no-daemon

Downloading https://services.gradle.org/distributions/gradle-6.0.1-bin.zip

.....

Unzipping /var/lib/jenkins/.gradle/wrapper/dists/gradle-6.0.1-bin/1lxlpkiy24sb18odw96cp4ojv/gradle-6.0.1-bin.zip to /var/lib/jenkins/.gradle/wrapper/dists/gradle-6.0.1-bin/1lxlpkiy24sb18odw96cp4ojv

Set executable permissions for: /var/lib/jenkins/.gradle/wrapper/dists/gradle-6.0.1-bin/1lxlpkiy24sb18odw96cp4ojv/gradle-6.0.1/bin/gradle

Welcome to Gradle 6.0.1!

Here are the highlights of this release:

- Substantial improvements in dependency management, including
  - Publishing Gradle Module Metadata in addition to pom.xml
  - Advanced control of transitive versions

- Support for optional features and dependencies

- Rules to tweak published metadata

- Support for Java 13

- Faster incremental Java and Groovy compilation

- New Zinc compiler for Scala

- VS2019 support

- Support for Gradle Enterprise plugin 3.0

For more details see https://docs.gradle.org/6.0.1/release-notes.html

WARNING: An illegal reflective access operation has occurred

WARNING: Illegal reflective access by org.codehaus.groovy.vmplugin.v7.Java7$1

(file:/var/lib/jenkins/.gradle/wrapper/dists/gradle-6.0.1-bin/1lxlpkiy24sb18odw96cp4ojv/gradle-6.0.1/lib/groovy-all-1.3-2.5.8.jar) to constructor

java.lang.invoke.MethodHandles$Lookup(java.lang.Class,int)

WARNING: Please consider reporting this to the maintainers of

org.codehaus.groovy.vmplugin.v7.Java7$1

WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations

WARNING: All illegal access operations will be denied in a future release

> Task :nodeSetup

> Task :nodeSetup FAILED

FAILURE: Build failed with an exception.

* What went wrong:

Execution failed for task ':nodeSetup'.

> Could not resolve all files for configuration ':detachedConfiguration1'.

  > Could not download node-9.11.1-linux-x64.tar.gz (org.nodejs:node:9.11.1)

> Could not get resource 'https://nodejs.org/dist/v9.11.1/node-v9.11.1-linux-x64.tar.gz'.

> Read timed out

* Try:

Run with --stacktrace option to get the stack trace. Run with --info or --debug option to get more log output. Run with --scan to get full insights.

* Get more help at https://help.gradle.org

Deprecated Gradle features were used in this build, making it incompatible with Gradle 7.0.

Use '--warning-mode all' to show the individual deprecation warnings.

See

https://docs.gradle.org/6.0.1/userguide/command_line_interface.html#sec:command_line_warnings

BUILD FAILED in 10m 17s

1 actionable task: 1 executed

Solution:

This is, likely, down to internet issues.

Select **Run Build** again.

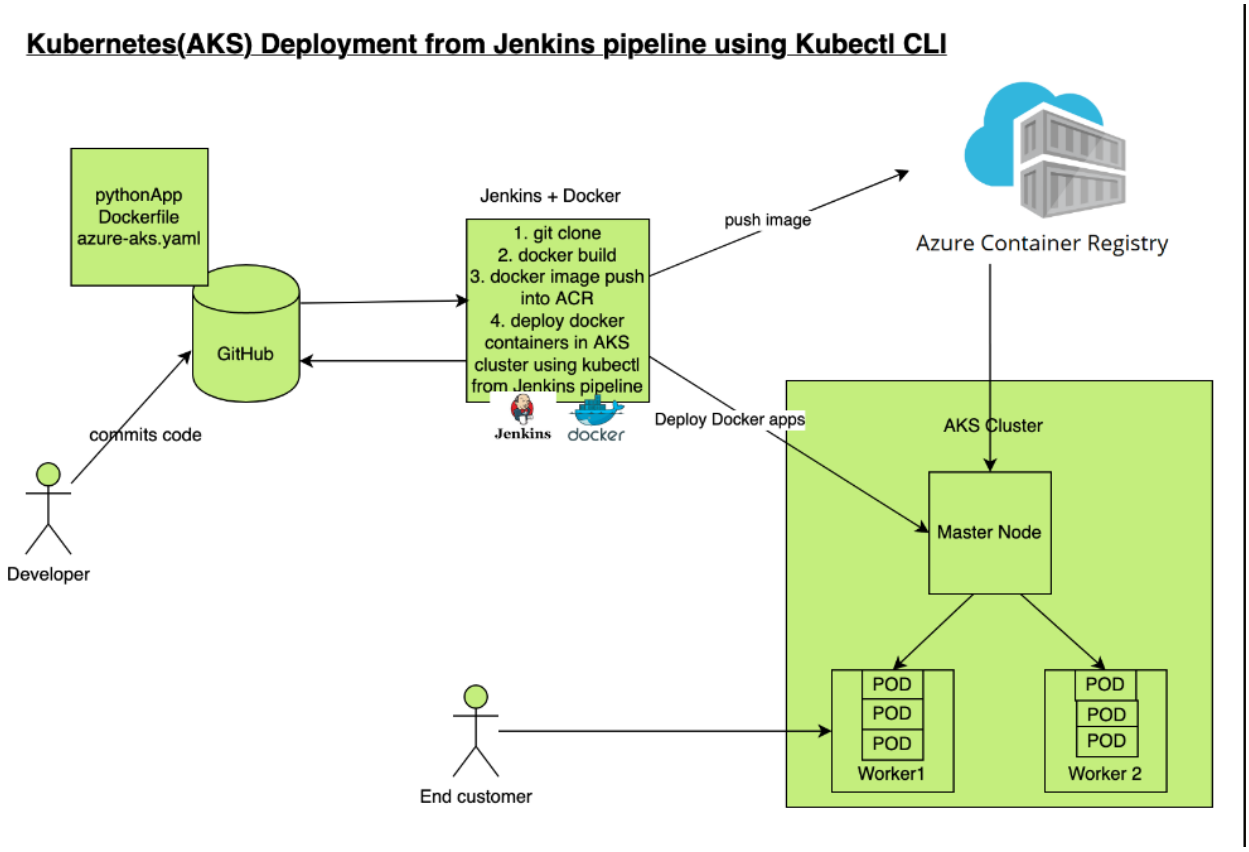
If this still fails then perform troubleshooting on your internet connection!

Alternatively, download the following manually and install it.

sudo wget https://nodejs.org/dist/v9.11.1/node-v9.11.1-linux-x64.tar.gz

https://nodejs.org/en/download

Deploy Python App into Kubernetes Cluster using Jenkins Pipeline | Containerize Python App and Deploy into AKS Cluster



Pre-requisites:

Create an Ubuntu 22.0.4 LTS machine on the Azure Portal.
Remote into the Ubuntu machine using a keypair from your command prompt.

1. AKS Cluster is setup and running.
First check to see if Azure CLI is installed on your local machine.

Use the command, **az version** to confirm.

If you do not see a version of **azure-cli** then proceed to the following instructions:

Run the update first

sudo apt update

Install with one command

curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash

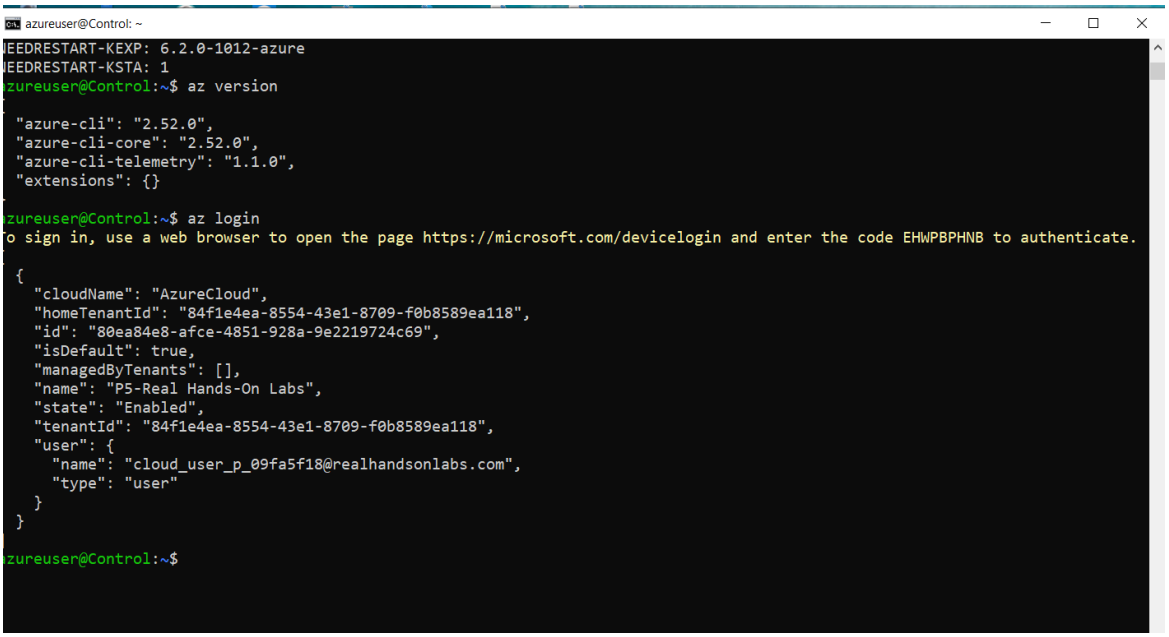
Check again to see if the system has a version of Azure CLI

az version

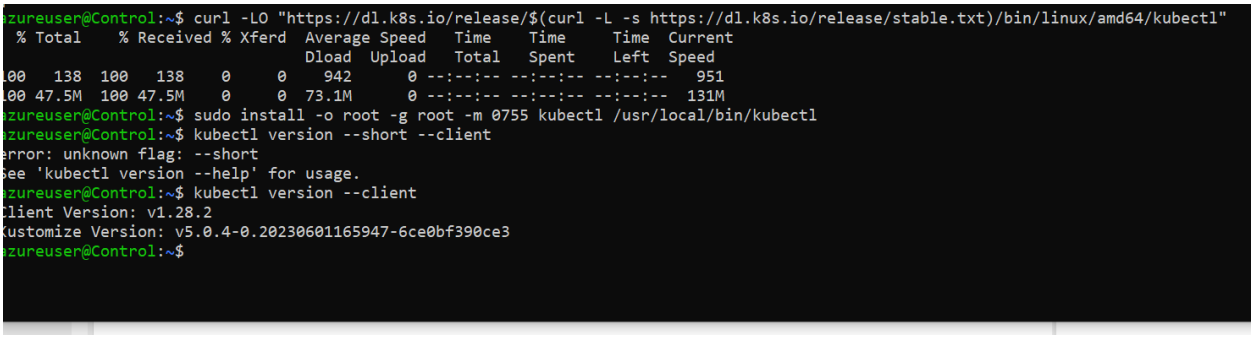
Second, insure you have an account setup in Azure cloud.

To sign in, use the **az** login command and follow the instructions in the command prompt.

az login



Third, Install kubectl.

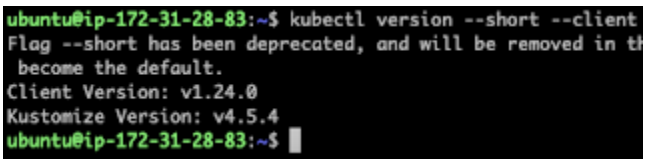


Install kubectl binary with curl on Linux

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

Install kubectl
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

Verify if kubectl got installed
kubectl version --short --client

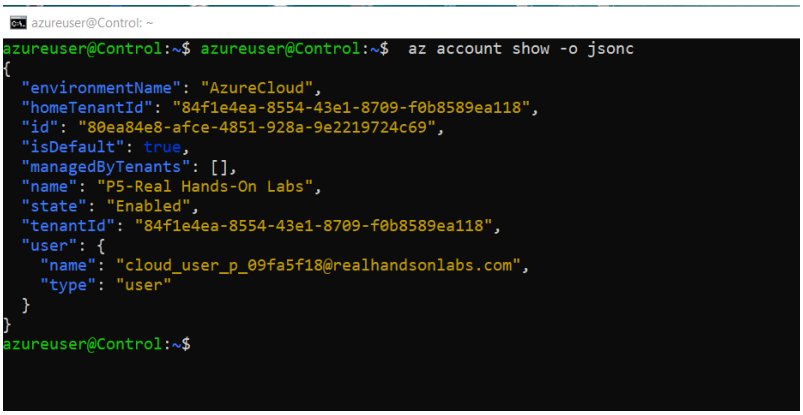


Now let's setup the AKS cluster

Step 1

Make sure, once again, you are logged into the Azure portal first with
The following command:

az account show -o jsonc

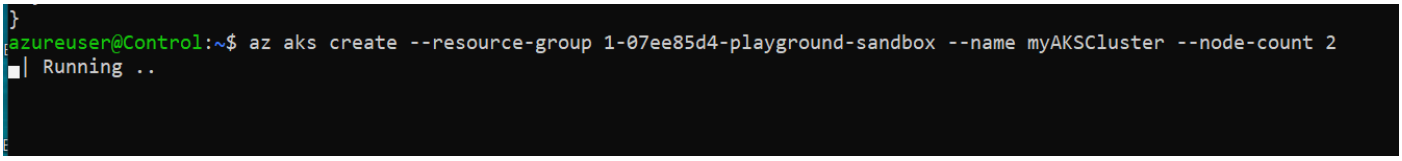


Step 2 - create a resource group first
az group create --name myResourceGroup --location southcentralus

Step 3 Create AKS cluster with 2 worker nodes
az aks create --resource-group myResourceGroup --name myAKSCluster **--node-count 2**

az aks create --resource-group 1-07ee85d4-playground-sandbox --name myAKSCluster --node-count 2

NOTE: if you do not include the switch or flag, **--node-count 2** it will, by default, generate 2 nodes.



Output:

azureuser@Control:~$ az aks create --resource-group 1-07ee85d4-playground-sandbox --name myAKSCluster --node-count 2
{
  "aadProfile": null,
  "addonProfiles": null,
  "agentPoolProfiles": [
    {
      "availabilityZones": null,
      "count": 2,
      "creationData": null,
      "currentOrchestratorVersion": "1.26.6",
      "enableAutoScaling": false,
      "enableEncryptionAtHost": false,
      "enableFips": false,
      "enableNodePublicIp": false,
      "enableUltraSsd": false,
      "gpuInstanceProfile": null,
      "hostGroupId": null,
      "kubeletConfig": null,
      "kubeletDiskType": "OS",
      "linuxOsConfig": null,
      "maxCount": null,
      "maxPods": 110,
      "minCount": null,
      "mode": "System",
      "name": "nodepool1",
      "nodeImageVersion": "AKSUbuntu-2204gen2containerd-202308.22.0",
      "nodeLabels": null,
      "nodePublicIpPrefixId": null,
      "nodeTaints": null,

```
    "orchestratorVersion": "1.26.6",
    "osDiskSizeGb": 128,
    "osDiskType": "Managed",
    "osSku": "Ubuntu",
    "osType": "Linux",
    "podSubnetId": null,
    "powerState": {
      "code": "Running"
    },
    "provisioningState": "Succeeded",
    "proximityPlacementGroupId": null,
    "scaleDownMode": null,
    "scaleSetEvictionPolicy": null,
    "scaleSetPriority": null,
    "spotMaxPrice": null,
    "tags": null,
    "type": "VirtualMachineScaleSets",
    "upgradeSettings": {
      "drainTimeoutInMinutes": null,
      "maxSurge": null
    },
    "vmSize": "Standard_DS2_v2",
    "vnetSubnetId": null,
    "workloadRuntime": null
  }
 ],
 "apiServerAccessProfile": null,
 "autoScalerProfile": null,
 "autoUpgradeProfile": {
  "nodeOsUpgradeChannel": "NodeImage",
  "upgradeChannel": null
 },
 "azureMonitorProfile": null,
 "azurePortalFqdn": "myaksclust-1-07ee85d4-playg-80ea84-qkr1nwg3.portal.hcp.westus.azmk8s.io",
 "currentKubernetesVersion": "1.26.6",
 "disableLocalAccounts": false,
 "diskEncryptionSetId": null,
 "dnsPrefix": "myAKSClust-1-07ee85d4-playg-80ea84",
 "enablePodSecurityPolicy": null,
 "enableRbac": true,
 "extendedLocation": null,
 "fqdn": "myaksclust-1-07ee85d4-playg-80ea84-qkr1nwg3.hcp.westus.azmk8s.io",
 "fqdnSubdomain": null,
 "httpProxyConfig": null,
 "id": "/subscriptions/80ea84e8-afce-4851-928a-9e2219724c69/resourcegroups/1-07ee85d4-
playground-sandbox/providers/Microsoft.ContainerService/managedClusters/myAKSCluster",
 "identity": {
  "delegatedResources": null,
```

```
    "principalId": "26942b51-f827-4b6e-b0b0-6da6368b5e56",
    "tenantId": "84f1e4ea-8554-43e1-8709-f0b8589ea118",
    "type": "SystemAssigned",
    "userAssignedIdentities": null
  },
  "identityProfile": {
    "kubeletidentity": {
      "clientId": "71c5b1ff-8c95-4cd8-b461-372eee4e2ffe",
      "objectId": "3cc16e6d-7e41-4097-a312-aadb5ae2adc0",
      "resourceId": "/subscriptions/80ea84e8-afce-4851-928a-9e2219724c69/resourcegroups/MC_1-
07ee85d4-playground-
sandbox_myAKSCluster_westus/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myAKSCl
uster-agentpool"
    }
  },
  "kubernetesVersion": "1.26.6",
  "linuxProfile": {
    "adminUsername": "azureuser",
    "ssh": {
      "publicKeys": [
        {
          "keyData": "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCgrNWX7252YzD3Rr95Yq1NRommZLti9mmjcMEQYeHujWmy4
VyPVOeUdXd25rzzXk5CxG7ZJyTqyvmz+IubS/6L40blYlI/Z4ckBrDRt9CEbJGDEYwb7qM9pNyoXtW0W+XfV4
Xpf/XlOG1OS/ei7DUs8mo8kNLwEyZXKr2ZvTpX51o9N7Ahvr81aiDLX6fcxyIL/x6zu57Xe1Zy2zqxhI/GfjdDQ4
HooejXndqqy21X1B8hoMT3jFWTfwTvQr5ItJmJswQpNmFa+WDV7166hQHroe7XBp6/GT9yECoP7Q331e
MKzUJ7X64JwEAUqmKhlzvmN6lzk+7OBau8X0cAnhMx"
        }
      ]
    }
  },
  "location": "westus",
  "maxAgentPools": 100,
  "name": "myAKSCluster",
  "networkProfile": {
    "dnsServiceIp": "10.0.0.10",
    "ipFamilies": [
      "IPv4"
    ],
    "loadBalancerProfile": {
      "allocatedOutboundPorts": null,
      "effectiveOutboundIPs": [
        {
          "id": "/subscriptions/80ea84e8-afce-4851-928a-9e2219724c69/resourceGroups/MC_1-07ee85d4-
playground-
sandbox_myAKSCluster_westus/providers/Microsoft.Network/publicIPAddresses/e8443d68-4b32-48fa-
bd74-71c057458ab5",
          "resourceGroup": "MC_1-07ee85d4-playground-sandbox_myAKSCluster_westus"
```

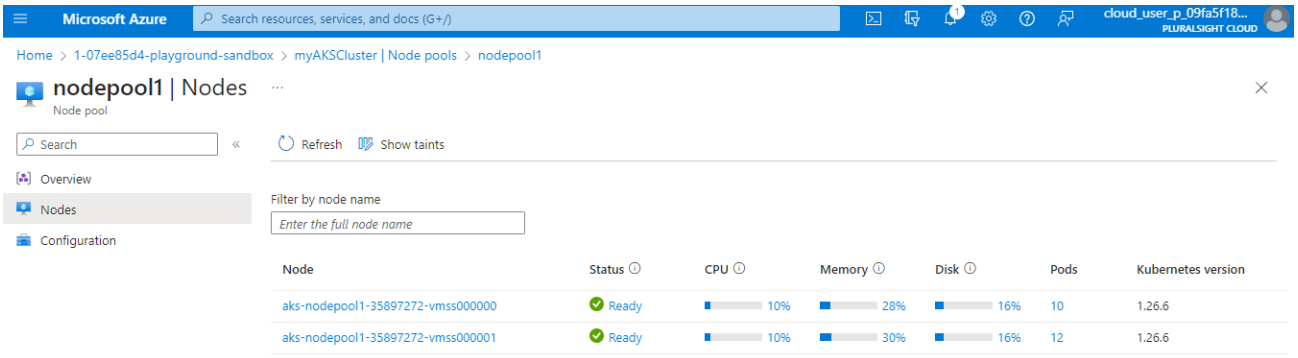
```
      }
     ],
     "enableMultipleStandardLoadBalancers": null,
     "idleTimeoutInMinutes": null,
     "managedOutboundIPs": {
      "count": 1,
      "countIpv6": null
     },
     "outboundIPs": null,
     "outboundIpPrefixes": null
    },
    "loadBalancerSku": "Standard",
    "natGatewayProfile": null,
    "networkDataplane": null,
    "networkMode": null,
    "networkPlugin": "kubenet",
    "networkPluginMode": null,
    "networkPolicy": null,
    "outboundType": "loadBalancer",
    "podCidr": "10.244.0.0/16",
    "podCidrs": [
     "10.244.0.0/16"
    ],
    "serviceCidr": "10.0.0.0/16",
    "serviceCidrs": [
     "10.0.0.0/16"
    ]
  },
  "nodeResourceGroup": "MC_1-07ee85d4-playground-sandbox_myAKSCluster_westus",
  "oidcIssuerProfile": {
    "enabled": false,
    "issuerUrl": null
  },
  "podIdentityProfile": null,
  "powerState": {
    "code": "Running"
  },
  "privateFqdn": null,
  "privateLinkResources": null,
  "provisioningState": "Succeeded",
  "publicNetworkAccess": null,
  "resourceGroup": "1-07ee85d4-playground-sandbox",
  "securityProfile": {
    "azureKeyVaultKms": null,
    "defender": null,
    "imageCleaner": null,
    "workloadIdentity": null
  },
```

```
 "servicePrincipalProfile": {
  "clientId": "msi",
  "secret": null
 },
 "sku": {
  "name": "Base",
  "tier": "Free"
 },
 "storageProfile": {
  "blobCsiDriver": null,
  "diskCsiDriver": {
   "enabled": true
  },
  "fileCsiDriver": {
   "enabled": true
  },
  "snapshotController": {
   "enabled": true
  }
 },
 "supportPlan": "KubernetesOfficial",
 "systemData": null,
 "tags": null,
 "type": "Microsoft.ContainerService/ManagedClusters",
 "upgradeSettings": null,
 "windowsProfile": null,
 "workloadAutoScalerProfile": {
  "keda": null,
  "verticalPodAutoscaler": null
 }
}
```


Display Details of Cluster

az aks show --name myAKSCluster --resource-group myResourceGroup
az aks show --name myAKSCluster --resource-group 1-07ee85d4-playground-sandbox

The above command will display Cluster details.



The two nodes that we created can be seen above!

2. Jenkins Master and Slave is up and running.

Jenkins Master

Change Host Name to Jenkins

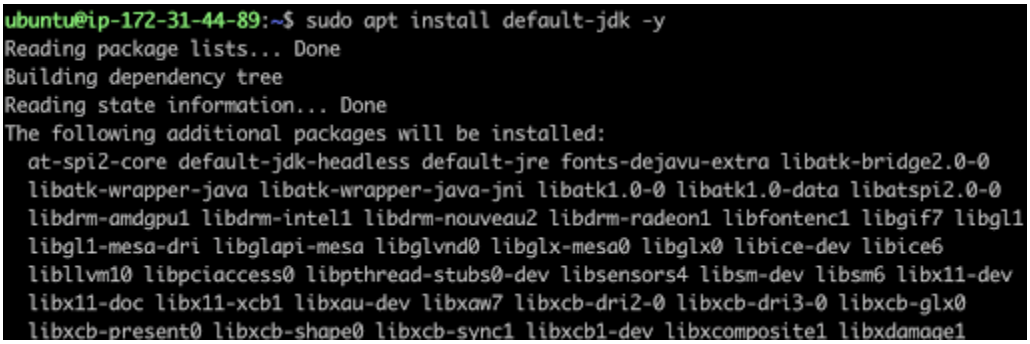
sudo hostnamectl set-hostname Jenkins

Perform update first

sudo apt update

Install Java 11

sudo apt install default-jdk -y



Once install java, enter the below command.

Verify Java Version

java -version

Maven Installation

You can install Maven by executing below command:

sudo apt install maven -y

Now type the following:

mvn --version

Now let's start the Jenkins installation.

Jenkins Setup

Add a Repository key to the system

curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key | sudo tee \
/usr/share/keyrings/jenkins-keyring.asc > /dev/null

Append debian package repo address to the system

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null

Update Ubuntu package

sudo apt update

```
ubuntu@ip-172-31-37-211:~$ sudo apt-get update
Hit:1 http://us-east-2.ec2.archive.ubuntu.com/ubuntu xenial InRelease
Hit:2 http://us-east-2.ec2.archive.ubuntu.com/ubuntu xenial-updates In
Hit:3 http://us-east-2.ec2.archive.ubuntu.com/ubuntu xenial-backports
Hit:4 http://security.ubuntu.com/ubuntu xenial-security InRelease
Hit:5 http://ppa.launchpad.net/webupd8team/java/ubuntu xenial InReleas
Ign:6 http://pkg.jenkins.io/debian-stable binary/ InRelease
Get:7 http://pkg.jenkins.io/debian-stable binary/ Release [2,042 B]
Get:8 http://pkg.jenkins.io/debian-stable binary/ Release.gpg [181 B]
Get:9 http://pkg.jenkins.io/debian-stable binary/ Packages [13.2 kB]
Fetched 15.5 kB in 0s (17.8 kB/s)
Reading package lists... Done
```

Install Jenkins

sudo apt install jenkins -y

```
ubuntu@ip-172-31-37-211:~$ sudo apt-get install jenkins -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  daemon
The following NEW packages will be installed:
  daemon jenkins
0 upgraded, 2 newly installed, 0 to remove and 8 not upgraded.
Need to get 72.0 MB of archives.
After this operation, 75.1 MB of additional disk space will be used.
Get:1 http://us-east-2.ec2.archive.ubuntu.com/ubuntu xenial/universe amd64 daemon
Get:2 http://pkg.jenkins.io/debian-stable binary/ jenkins 2.121.3 [71.9 MB]
Fetched 72.0 MB in 5s (12.5 MB/s)
Selecting previously unselected package daemon.
(Reading database ... 51843 files and directories currently installed.)
Preparing to unpack .../daemon_0.6.4-1_amd64.deb ...
Unpacking daemon (0.6.4-1) ...
Selecting previously unselected package jenkins.
Preparing to unpack .../jenkins_2.121.3_all.deb ...
Unpacking jenkins (2.121.3) ...
Processing triggers for man-db (2.7.5-1) ...
Processing triggers for systemd (229-4ubuntu21.4) ...
Processing triggers for ureadahead (0.100.0-19) ...
Setting up daemon (0.6.4-1) ...
Setting up jenkins (2.121.3) ...
Processing triggers for systemd (229-4ubuntu21.4) ...
Processing triggers for ureadahead (0.100.0-19) ...
ubuntu@ip-172-31-37-211:~$
```

The above screenshot should confirm that Jenkins is successfully installed.

Allow Access in Jenkins in a web browser

Create an inbound rule within **Settings > Networking** for port 8080.

Now go to browser and enter the virtual machines public IP address with port no 8080.

http://Public IP address:8080

Unlock Jenkins

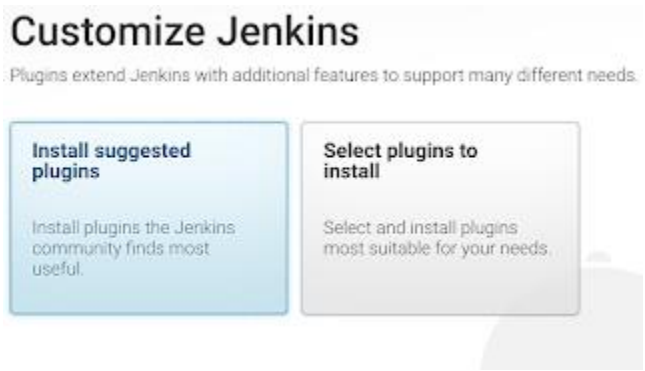
When this page loads, enter the below command in Git bash (Ubuntu console)

Get the initial password from the below file

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

Now copy the password and paste it into the browser text box.

Then click on install suggested plug-ins.



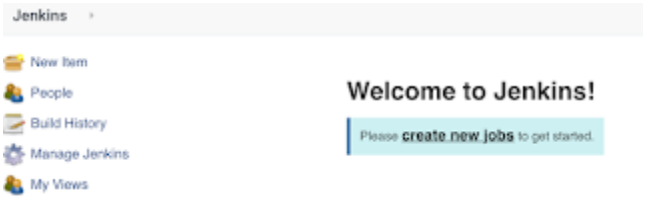
Also create a username and password.

Enter everything as admin. At least username as admin password as admin

Click on **Create User**.

Click on **Save** and **Finish**.

Click on **start using Jenkins**. Now you should see a screen like below:



Setup Jenkins slave. Then ensure it can run Docker builds.

Create a similar Ubuntu 22.4 LTS to be a slave.

Note: You need to create at least a t2.small Ubuntu 22.0.4 instance for this slave.
Only port 22 needs to be open

Configure it with the following:

Perform update first
sudo apt update

Install Java 11
sudo apt install default-jdk -y

```
ubuntu@ip-172-31-44-89:~$ sudo apt install default-jdk -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  at-spi2-core default-jdk-headless default-jre fonts-dejavu-extra libatk-bridge2.0-0
  libatk-wrapper-java libatk-wrapper-java-jni libatk1.0-0 libatk1.0-data libatspi2.0-0
  libdrm-amdgpu1 libdrm-intel1 libdrm-nouveau2 libdrm-radeon1 libfontenc1 libgif7 libgl1
  libgl1-mesa-dri libglapi-mesa libglvnd0 libglx-mesa0 libglx0 libice-dev libice6
  libllvm10 libpciaccess0 libpthread-stubs0-dev libsensors4 libsm-dev libsm6 libx11-dev
  libx11-doc libx11-xcb1 libxau-dev libxaw7 libxcb-dri2-0 libxcb-dri3-0 libxcb-glx0
  libxcb-present0 libxcb-shape0 libxcb-sync1 libxcb1-dev libxcomposite1 libxdamage1
```

Once install java, enter the below command

Verify Java Version

java -version

```
ubuntu@ip-172-31-19-22:/opt$ java -version
openjdk version "11.0.6" 2020-01-14
OpenJDK Runtime Environment (build 11.0.6+10-post-Ubuntu-1ubuntu118.04.1)
OpenJDK 64-Bit Server VM (build 11.0.6+10-post-Ubuntu-1ubuntu118.04.1, mixed mode, sharing)
ubuntu@ip-172-31-19-22:/opt$
```

Maven Installation

You can install Maven by executing below command:

```
sudo apt install maven -y
```

you can type `mvn --version`

you should see the below output.

Steps for installing Docker

```
sudo apt-get update && sudo apt install docker.io -y
```

Add Jenkins to Docker Group

```
sudo usermod -aG docker jenkins

sudo newgrp docker
sudo systemctl daemon-reload
```

Restart Docker service

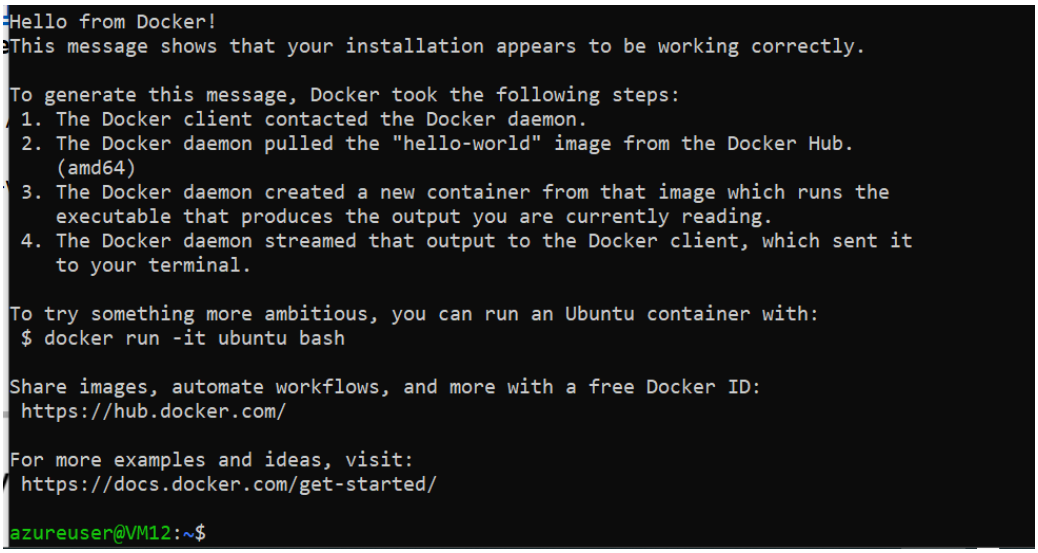
```
sudo systemctl start docker
sudo systemctl enable docker
sudo systemctl restart docker
sudo systemctl status docker
```

docker run hello-world

permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fvar%2Frun%2Fdocker.sock/v1.24/images/create?fromImage=tiangolo%2Fuwsgi-nginx-flask&tag=python3.6": dial unix /var/run/docker.sock: connect: permission denied

Run the following:
sudo chmod 666 /var/run/docker.sock

Now run the following hello-world command again.
docker run hello-world



Slave node configuration

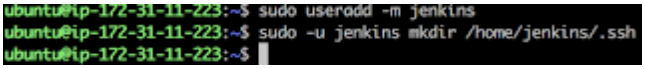
Change Host Name to Slave

sudo hostnamectl set-hostname Slave

Create User as Jenkins

sudo useradd -m jenkins

sudo -u jenkins mkdir /home/jenkins/.ssh

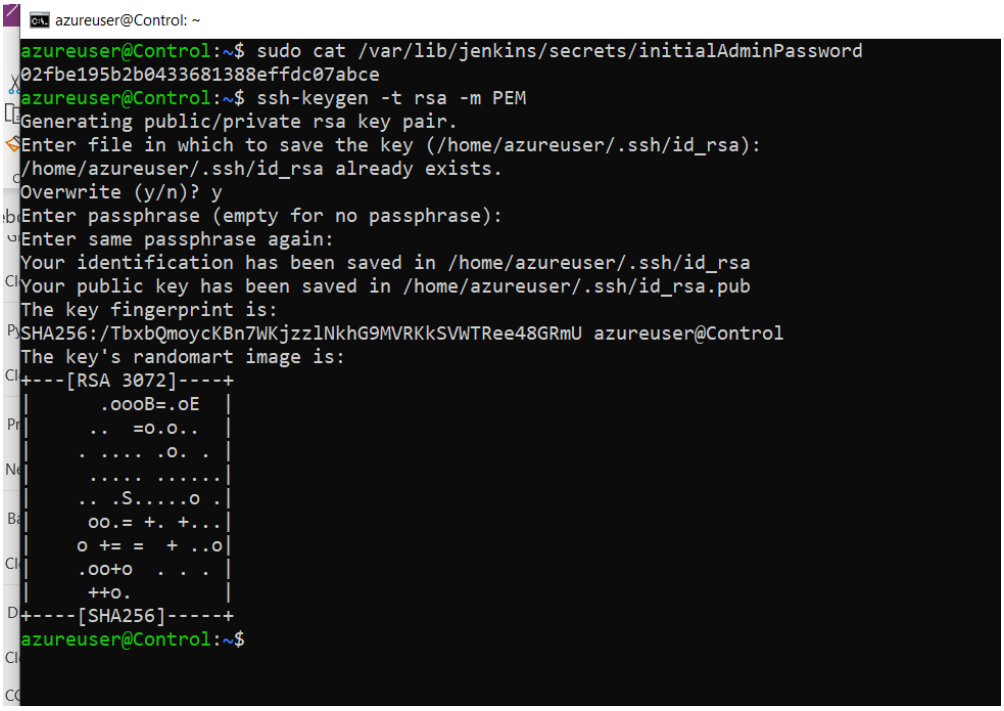


Now Login to Jenkins Master

Create SSH keys by executing below command:

ssh-keygen -t rsa -m PEM

Please overwrite your existing keys.



Copy SSH Keys from Master to Slave

Execute the below command in Jenkins master EC2.
sudo cat ~/.ssh/id_rsa.pub

Copy the output of the above command:

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQDcKK7OR+/aMiJfkUkqGXC0EqZMBY+069Dna38HlDtzAiK0YYqDuEIcphZOHSxjyrD6CPhaL6SC+Bx3zW2JIIWUVJZ3qsELjCzgLftKQLyvd9aWARzQCbcJGlOTmEX/kL2IUsb+sIFyDIFl592nk0QpG5fYBC6Y8LrG9+FgQQrue8o7BC8aHvQUz6o1c4k+aefc2CqQexonF9cDF4KMLnzYB+h3RRHUGMJ/ECd6OxmbprzQ4p1AeMcIV55OaJ7y34c/Pqq6llv2CF2ZpEd9dcROYW+hsNAZRHGEgdxmXlA5pmFuLh3DBVzUkVV0FL+/h5NUxfecY13BVTEfsQehCEdc1lE69oVUD3tsfxlt61IZrEdtkAcoJ9iLPY3WF8zLS1vMl7eeHDiYydSHSgFuZe2z/2D4peuY9/dKwc2nLqHSraD5vir6Xcz79YTx8+tA5r9aPpW9+hKV0FT9GiuRZmyMJk8j/383wWcLqVTM2ZAkg44o+Y9bsK6Z4YJ1FY6qlY0= azureuser@Control

Now Login to Slave node and execute the below command

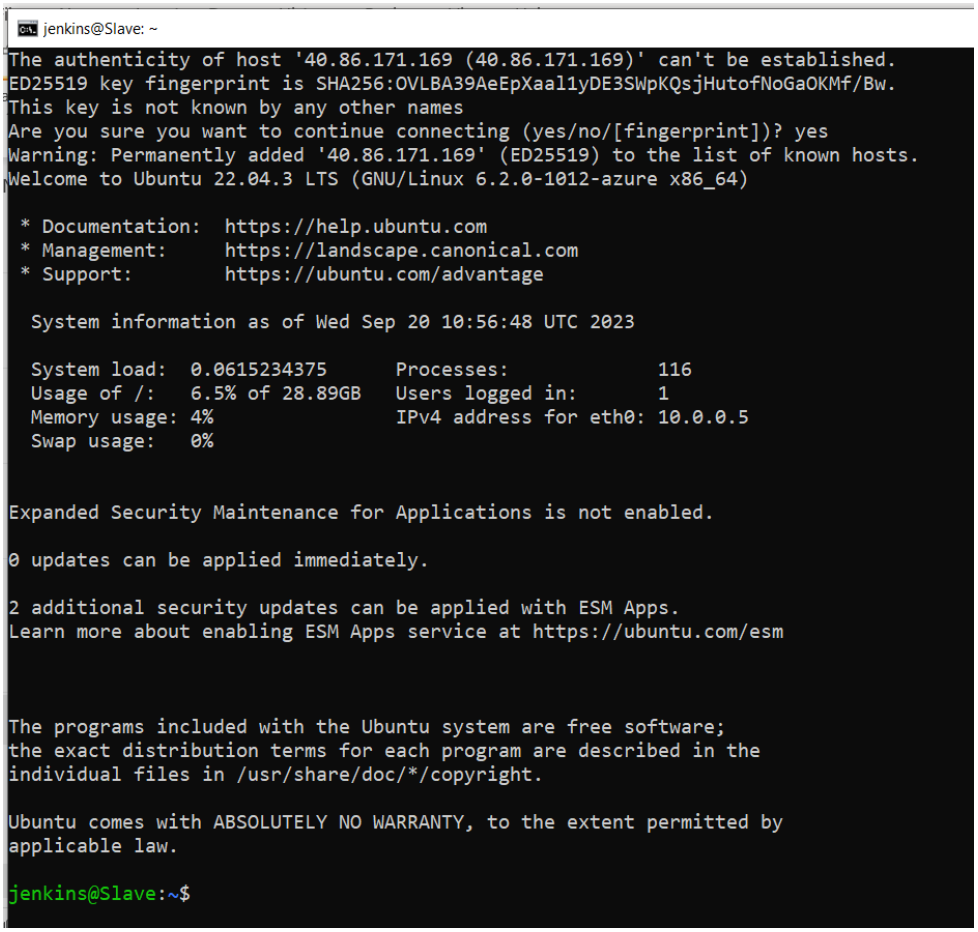
sudo -u jenkins vi /home/jenkins/.ssh/authorized_keys

This will be an empty file, now copy the public keys from master into above file.
Once you pasted the public keys in the above file in Slave, come out of the file by entering :wq!

Now go into master node

ssh jenkins@*slave_node_ip*

ssh jenkins@40.86.171.169

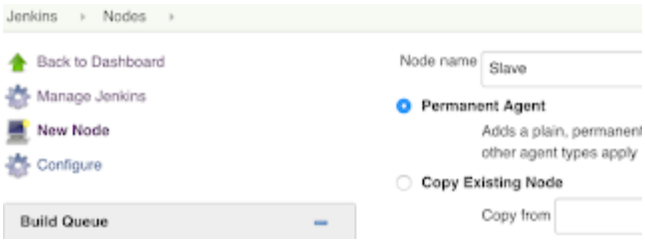


this is to make sure master is able to connect slave node. once you are successfully logged into slave, type exit to come out of slave.

Register slave node in Jenkins:

Now to go to Jenkins Master.

Manage Jenkins > Manage nodes.



Click on new node.
Give name and check permanent agent.
Set no of executors as 1.
Enter /home/jenkins as remote directory.
Select launch method as **Launch slaves nodes via SSH**.
Enter Slave node IP address as Host.

Click on credentials. Enter user name as jenkins.

Make jenkins as lowercase as it is shown.

Kind as SSH username with private key.

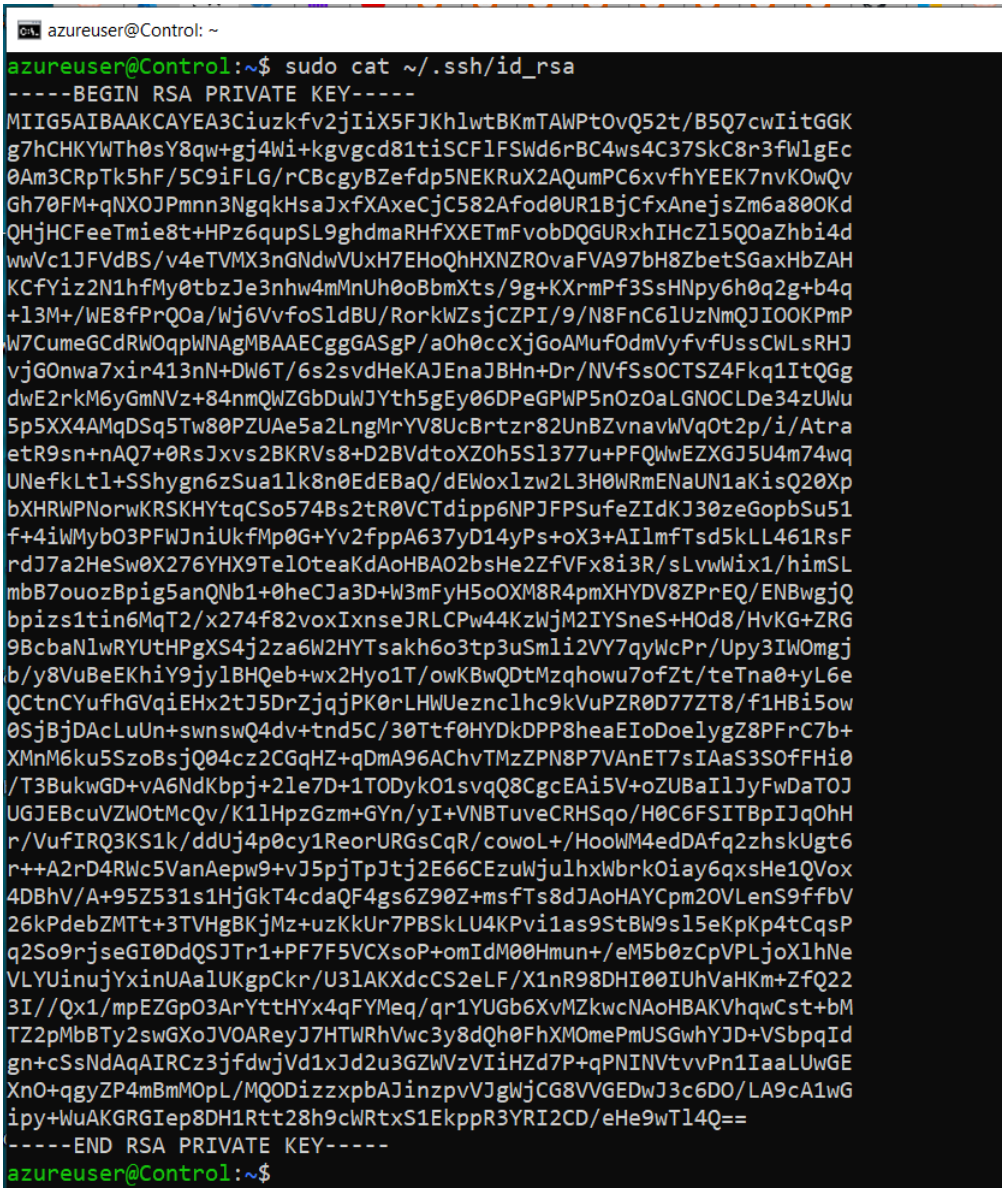
Enter private key of master node directly by executing below command:

sudo cat ~/.ssh/id_rsa

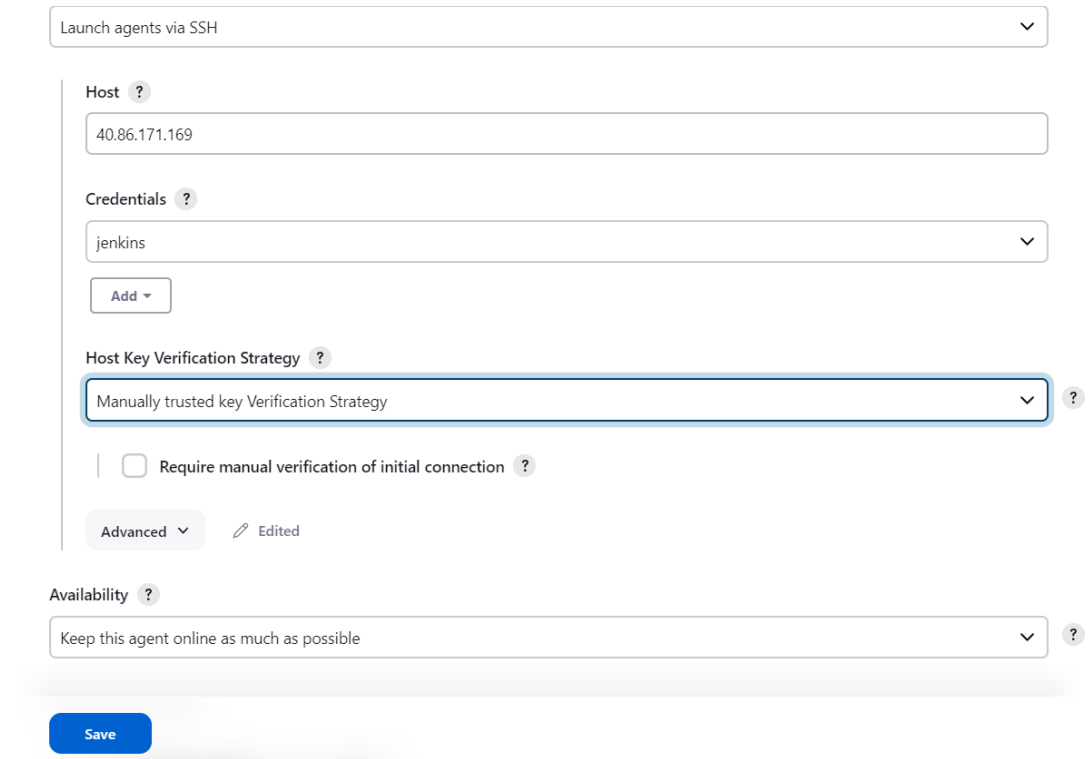
(Make sure you copy the whole key including the below without missing anything)

-----BEGIN RSA PRIVATE KEY-----

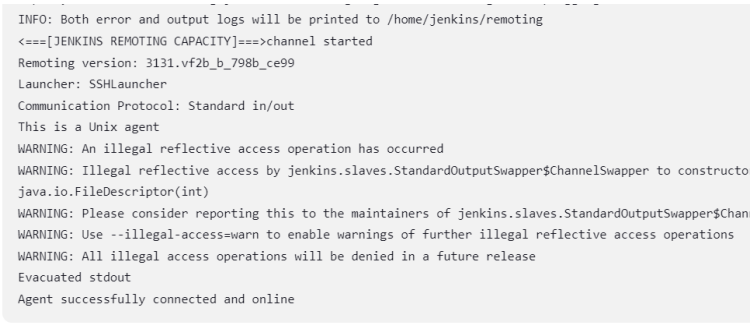
-----END RSA PRIVATE KEY-----

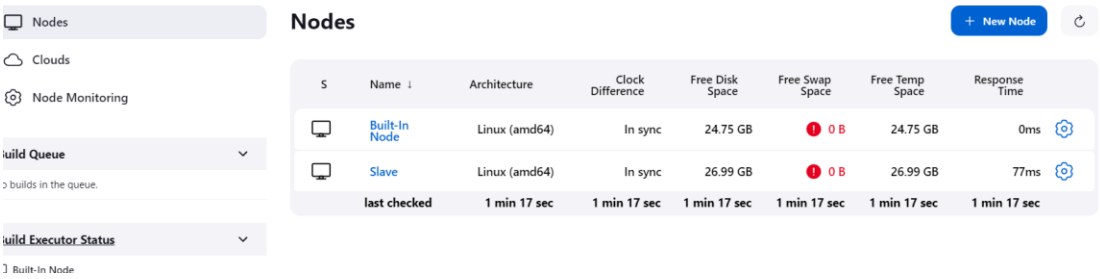


Click **Save**.
Select **Host key verification strategy** as "**manually trusted key verification strategy**".



Click **Save**.
Click on launch agent..make sure it connects to agent node.





Now you can kick start building the jobs, you will see Jenkins master runs jobs in slave nodes.

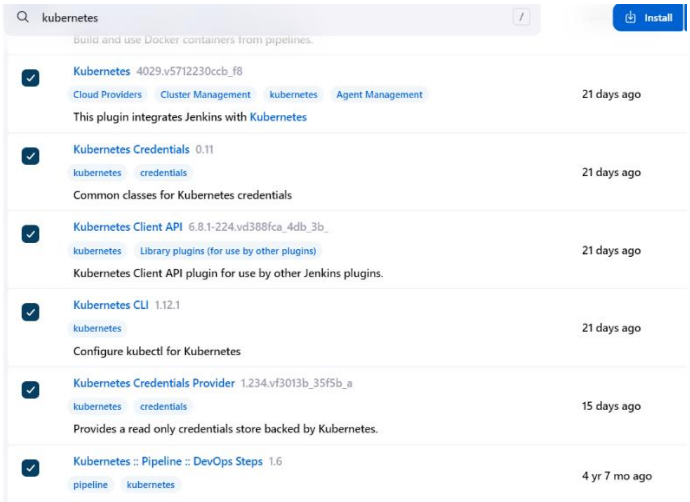
3. Docker, Docker pipeline and Kubectl CLI plug-ins are installed in Jenkins.

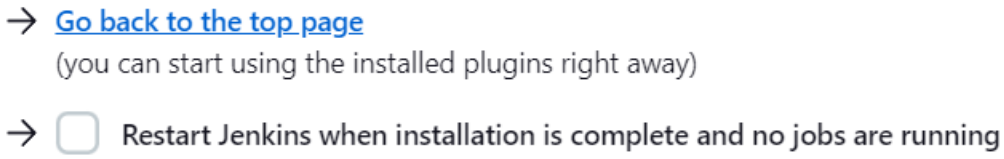
Managing Jenkins > Plugins

Available Plugins

Select and install the following:

Docker
Docker Pipeline
Kubectl CLI





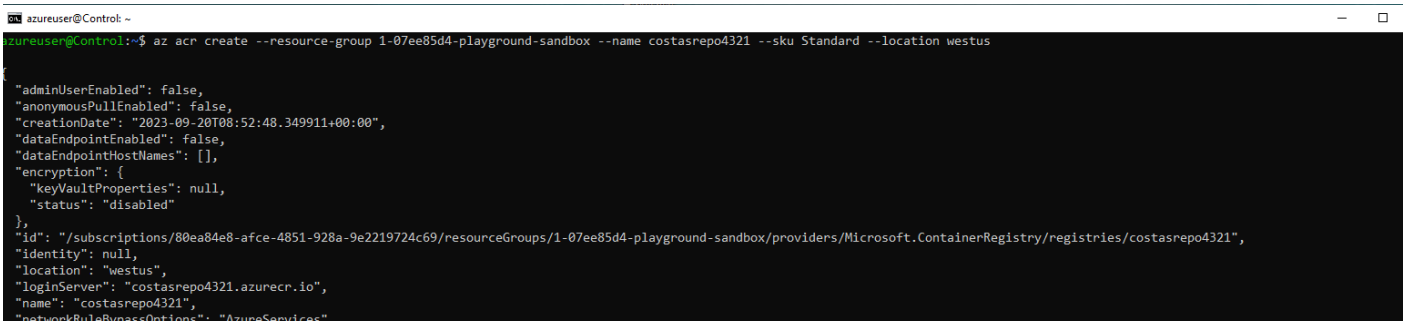
4. Azure container registry is setup in https://portal.azure.com

Run the below command to create your own private container registry using Azure Container Registry (ACR).

Make sure below green marked registry name is unique.

az acr create --resource-group myResourceGroup --name myacrrepo4321 --sku Standard --location southcentralus

az acr create --resource-group 1-07ee85d4-playground-sandbox --name costasrepo4321 --sku Standard --location westus



Connect to the cluster

az aks get-credentials --resource-group myResourceGroup --name myAKSCluster --overwrite-existing

az aks get-credentials --resource-group 1-07ee85d4-playground-sandbox --name myAKSCluster --overwrite-existing

To verify the connection to your cluster, use the kubectl get command to return a list of the cluster nodes.

kubectl get nodes

```
AK-DevOps-01:~ devopscoaching$  kubectl get nodes
NAME                          STATUS   ROLES   AGE   VERSION
aks-nodepool1-26195414-0      Ready    agent   13m   v1.17.11
aks-nodepool1-26195414-1      Ready    agent   13m   v1.17.11
```

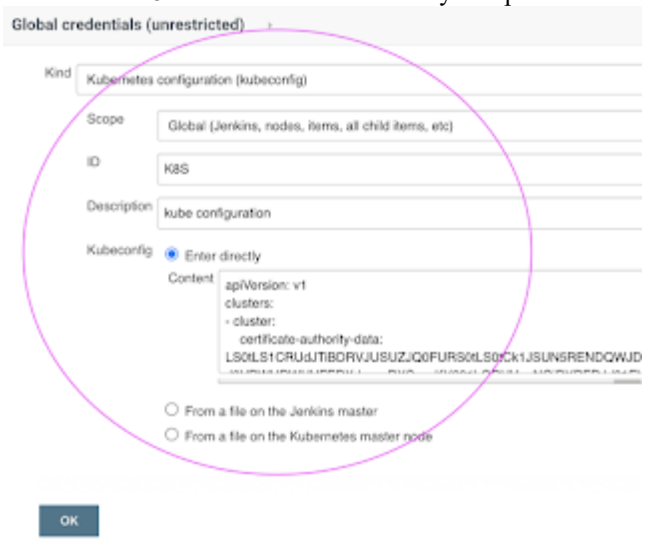
List all deployments in a specific namespace
kubectl get deployments --all-namespaces=true

```
AK-DevOps-01:~ devopscoaching$ kubectl get deployments --all-namespaces=true
NAMESPACE     NAME                READY   UP-TO-DATE   AVAILABLE   AGE
kube-system   coredns             2/2     2            2           6m25s
kube-system   coredns-autoscaler  1/1     1            1           6m25s
kube-system   metrics-server      1/1     1            1           6m25s
kube-system   omsagent-rs         1/1     1            1           6m25s
kube-system   tunnelfront         1/1     1            1           6m24s
AK-DevOps-01:~ devopscoaching$
```

azureuser@Control: ~

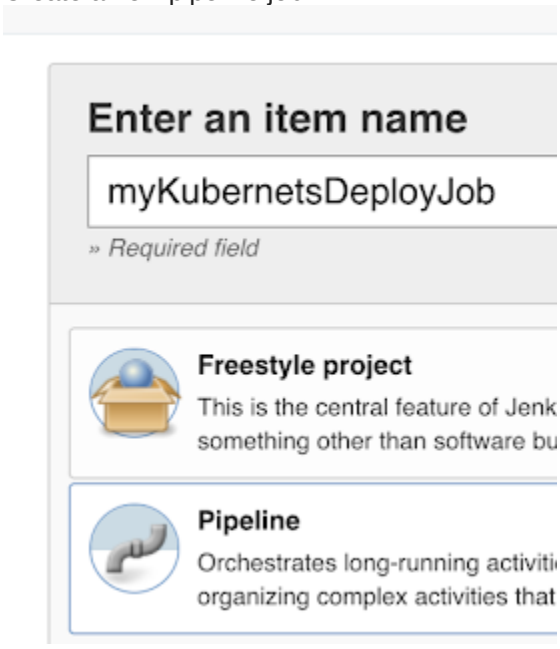
```
azureuser@Control:~$  az aks get-credentials --resource-group 1-07ee85d4-playground-sandbox --name myAKSCluster --overwrite-existing
Merged "myAKSCluster" as current context in /home/azureuser/.kube/config
azureuser@Control:~$ kubectl get nodes
NAME                                STATUS   ROLES   AGE   VERSION
aks-nodepool1-35897272-vmss000000   Ready    agent   15m   v1.26.6
aks-nodepool1-35897272-vmss000001   Ready    agent   15m   v1.26.6
azureuser@Control:~$
azureuser@Control:~$ kubectl get deployments --all-namespaces=true
NAMESPACE          NAME                   READY   UP-TO-DATE   AVAILABLE   AGE
gatekeeper-system  gatekeeper-audit       1/1     1            1           7m25s
gatekeeper-system  gatekeeper-controller  2/2     2            2           7m25s
kube-system        azure-policy           1/1     1            1           7m26s
kube-system        azure-policy-webhook   1/1     1            1           7m25s
kube-system        coredns                2/2     2            2           16m
kube-system        coredns-autoscaler     1/1     1            1           16m
kube-system        konnectivity-agent     2/2     2            2           16m
kube-system        metrics-server         2/2     2            2           16m
azureuser@Control:~$
azureuser@Control:~$
```

Enter ID as K8S and choose enter directly and paste the above file content and save.



Step # 3 - Create a pipeline in Jenkins

Create a new pipeline job.



Step # 4 - Copy the pipeline code from below

Make sure you change red highlighted values below:
Your docker user id should be updated.

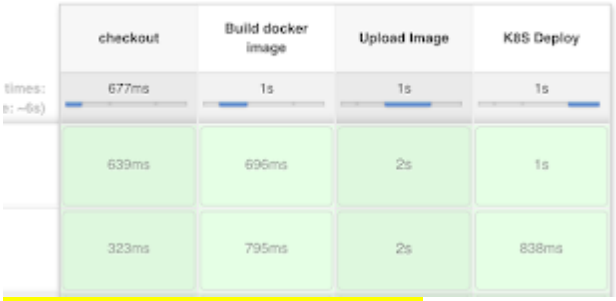
your registry credentials ID from Jenkins from step # 1 should be copied

```
pipeline {
    agent {
        label 'myslave'
    }
    environment {
    //once you sign up for Docker hub, use that user_id here
    registry = "your_docker_hub_user_id/mypython-app"
    //- update your credentials ID after creating credentials for connecting to Docker Hub
    registryCredential = 'dockerhub'
    dockerImage = ''
  }
  stages {

    stage ('checkout') {
        steps {
        checkout([$class: 'GitSCM', branches: [[name: '*/master']], doGenerateSubmoduleConfigurations:
false, extensions: [], submoduleCfg: [], userRemoteConfigs: [[url:
'https://github.com/akannan1087/myPythonDockerRepo']]])
        }
    }

    stage ('Build docker image') {
        steps {
            script {
            dockerImage = docker.build registry
            }
        }
    }

     // Uploading Docker images into Docker Hub
  stage('Upload Image') {
   steps{
      script {
        docker.withRegistry( '', registryCredential ) {
        dockerImage.push()
        }
      }
    }
  }

  stage ('K8S Deploy') {
      steps {
          script {
              kubernetesDeploy(
                  configs: 'k8s-deployment.yaml',
                  kubeconfigId: 'K8S',
                  enableConfigSubstitution: true
                  )

          }
      }
  }
```

```
    }
}
```

Step # 5 - Build the pipeline

Once you create the pipeline and changes values per your Docker user id and credentials ID, click on



Step # 6 - Verify deployments to K8S

kubectl get pods

```
ubuntu@ip-172-31-2-128:~$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
gk1-controller-h8dck                1/1     Running   3          23h
my-php-deployment-8cd555cd6-bz5bv   1/1     Running   0          4m2s
my-php-deployment-8cd555cd6-wbhlk   1/1     Running   0          4m2s
```

kubectl get deployments

```
ubuntu@ip-172-31-2-128:~$ kubectl get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
my-php-deployment   2/2     2            2           7m27s
ubuntu@ip-172-31-2-128:~$
```

kubectl get services

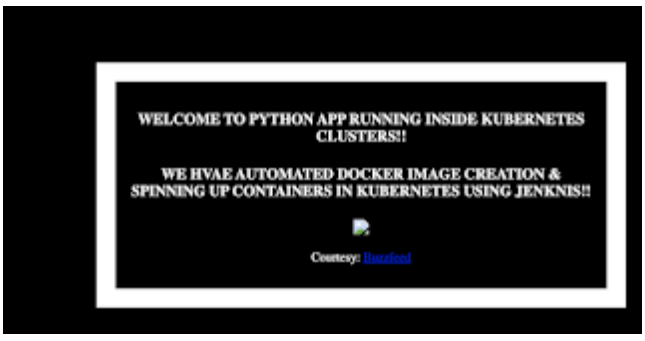
```
ubuntu@ip-172-31-2-128:~$ kubectl get services
NAME             TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
gk-angular-app   NodePort       10.103.50.84    <none>        8885:32004/TCP   23h
kubernetes       ClusterIP      10.96.0.1       <none>        443/TCP          26h
php-app-svc      LoadBalancer   10.97.39.217    <pending>     5000:30314/TCP   3m31s
ubuntu@ip-172-31-2-128:~$
```

Steps # 7 - Access Python App in K8S cluster

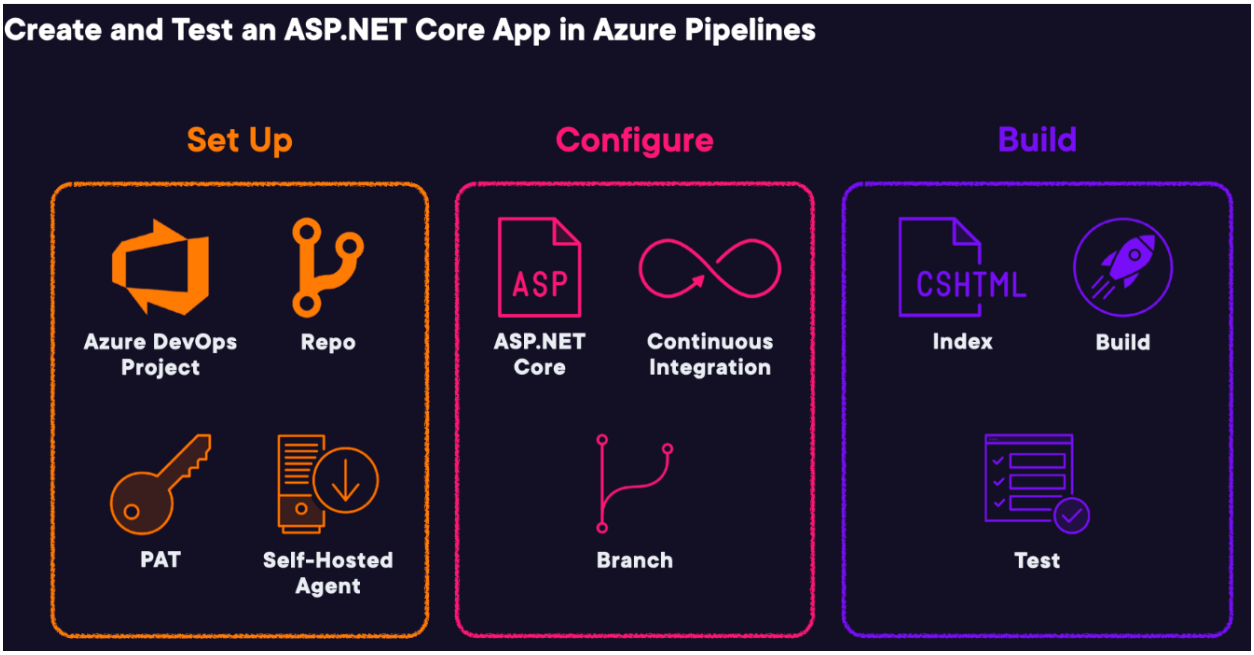
Once build is successful, go to browser and enter master or worker node public ip address along with port number mentioned above

http://master_or_worker_node_public_ipaddress:port_no_from_above

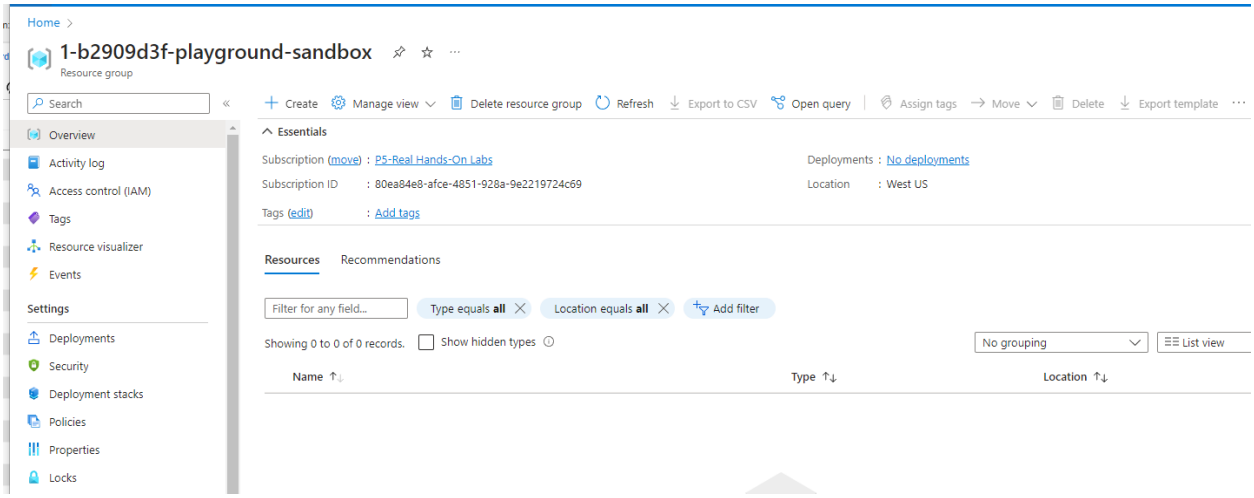
You should see page like below:



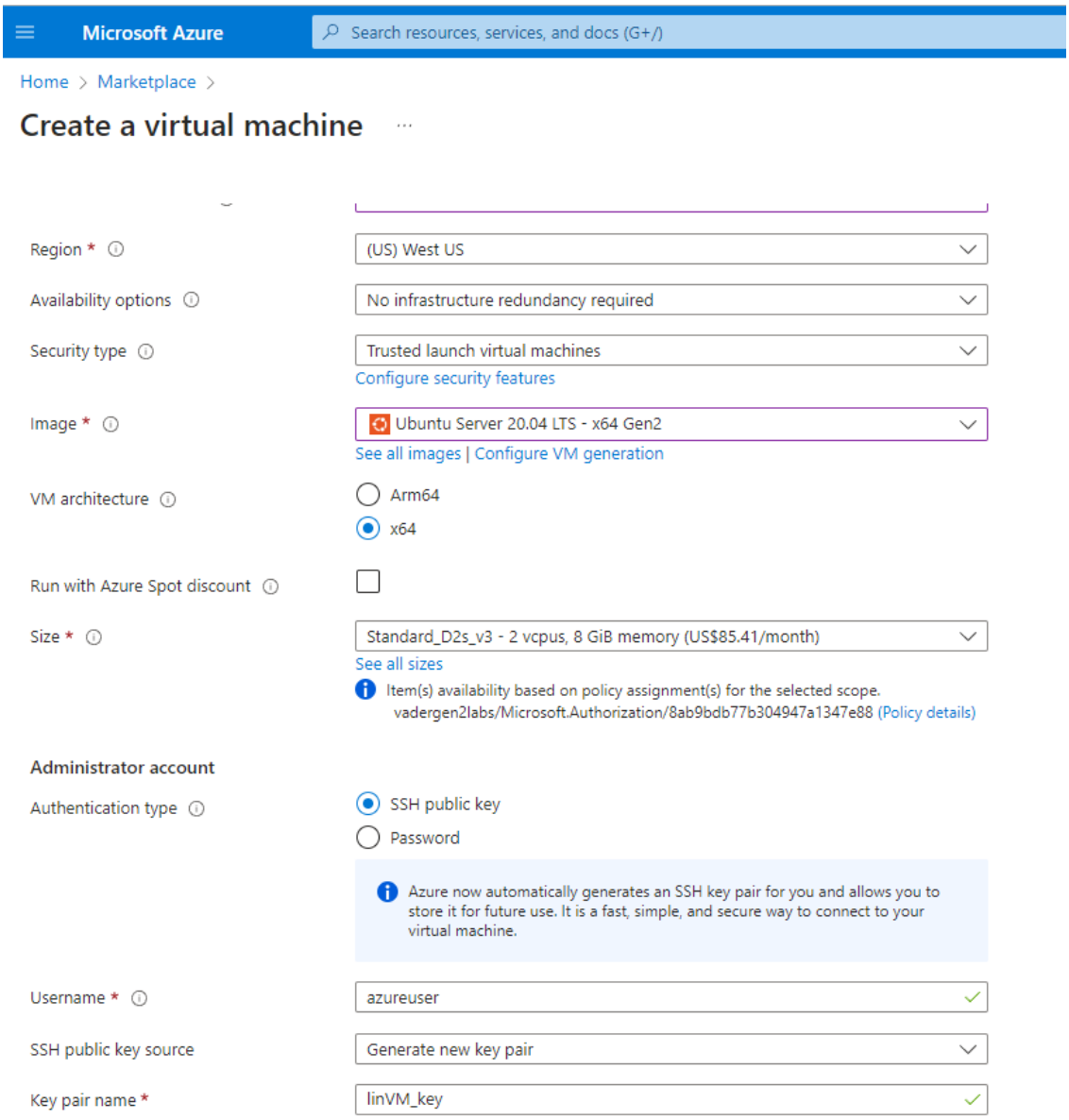
Create and Test an ASP.NET Core App in Azure Pipelines



Create the VM (and other resources using ARM)



Go to the **Market Place** and locate an Ubuntu image



Ensure that the **Region** is the same as the Resource group.

NOTE: Ubuntu 20.04 LTS was the only option.

Choose the **Size** Standard D2 with 2 CPUs as shown above.

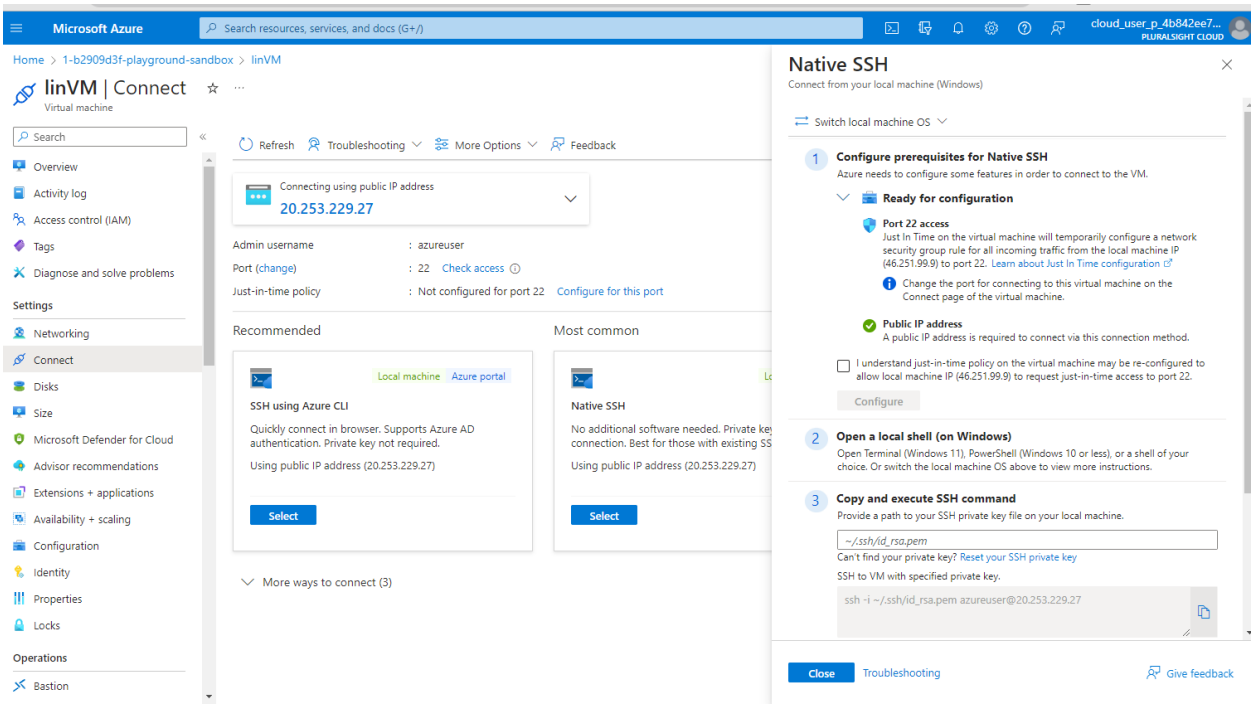
For **Authentication** type, choose **SSH Public Key** and download the key pair locally to your system.

Go to end and create the ubuntu virtual machine.

NOTE: When you complete creating the VM all the other associated resources will also be created.

Navigate to the Virtual machine **linVM** within the resource group and select the link.

On the left hand menu, navigate to Settings and select **connect > Native SSH**



In a command prompt on your system type the following:

ssh -i <path to key pair>/<key pair name>.pem <username created earlier>@<public IP address of VM>

Once logged in, proceed to create the Pipeline within Azure DEVOPS.

Create the Azure DEVOPS Pipeline

In the main search bar, search for and select **Azure DevOps organizations**.

Scroll down and click **My Azure DevOps Organizations**.

Select your country/region from the dropdown list.

Click **Continue**.

Select **Create new organization > Continue > Continue** (complete the required CAPTCHA).

Set the following values to create a project:

Project name: **ASP_NET**

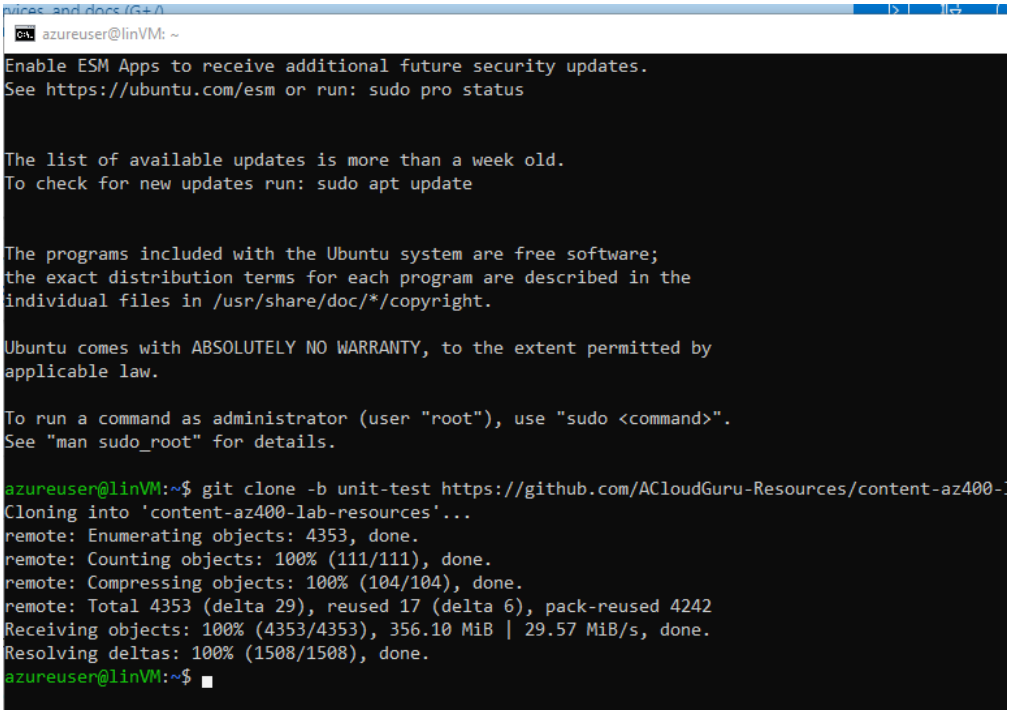
Visibility: **Private**

Click + **Create project**.

Obtain and Push Project Code

In the Linux VM, pull code from the unit-test branch using the following URL:

git clone -b unit-test https://github.com/ACloudGuru-Resources/content-az400-lab-resources.git



List the contents of the directory with **ls**

Observe the folder is cloned to the VM.

Change into the directory shown:

cd content-az400-lab-resources/

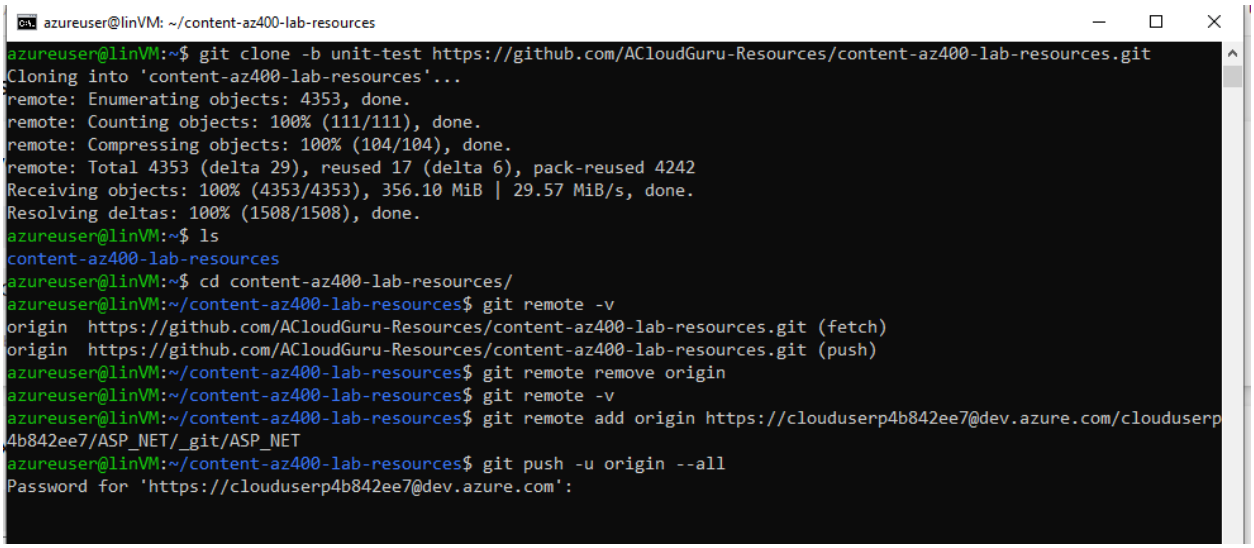
View the remote origin:

git remote -v

It is currently the ACG GitHub repo.

Remove the remote origin:

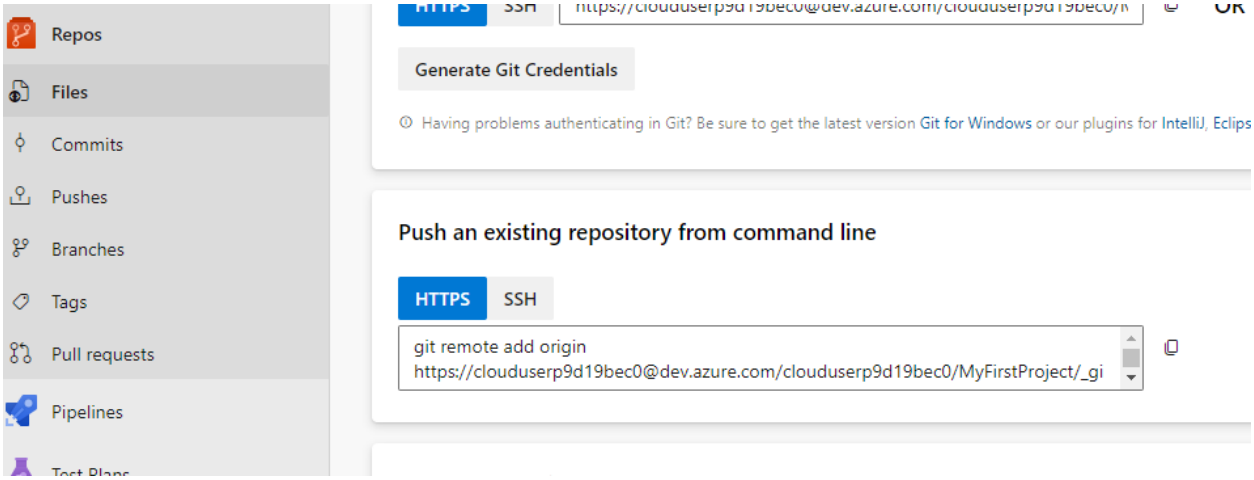
git remote remove origin

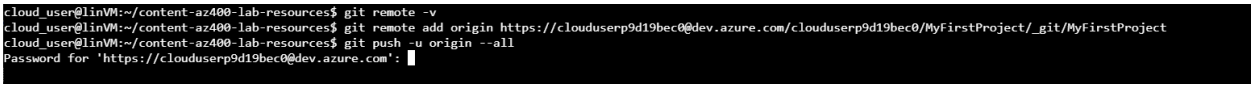


If you use the **git remote -v** command again you will find that nothing comes up!

Go back to **Azure DevOps**. From the left menu, click **Repos**.

Under **Push an existing repository** from command line, copy the commands.





Return to the Linux VM and paste in the commands. You will be asked for a personal access token.

Create Personal Access Token

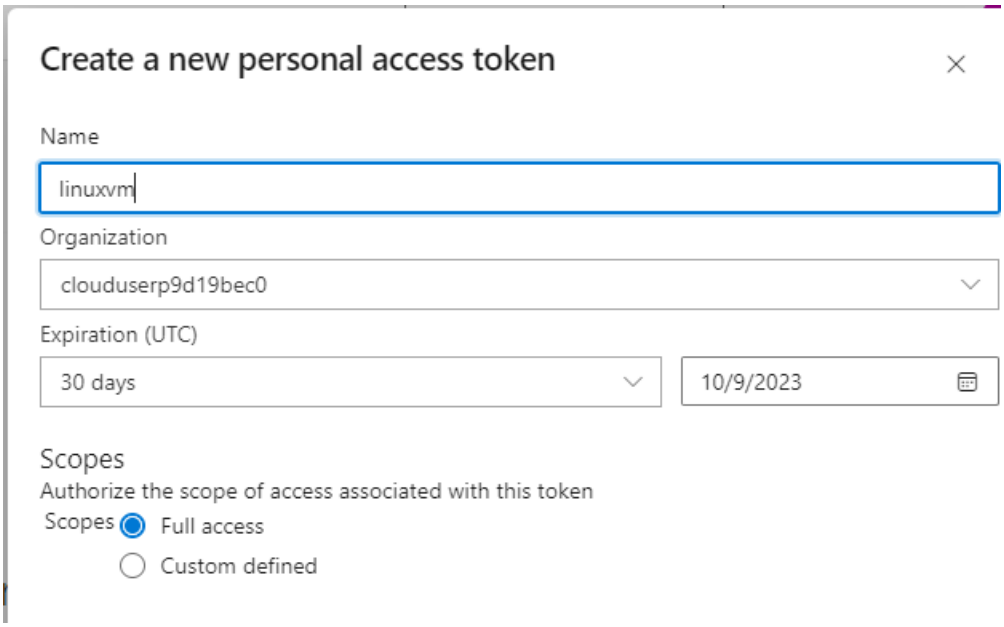
Back in Azure DevOps, in the upper right-hand corner, click the **User settings icon > Personal access tokens**.

Click + **New Token** and set the following values:

Name: **linuxvm**

Scopes: **Full access**

Click **Create**.



It will create a string like below:

das5yco2kl474qngkvp3fikjev6vtxgymaverwa7kasm5nq3xfkq

Copy the **linuxvm** token and save it in a safe location.

Click **Close**.

```
azureuser@linVM: ~/content-az400-lab-resources
emote: Compressing objects: 100% (104/104), done.
emote: Total 4353 (delta 29), reused 17 (delta 6), pack-reused 4242
eceiving objects: 100% (4353/4353), 356.10 MiB | 29.57 MiB/s, done.
esolving deltas: 100% (1508/1508), done.
zureuser@linVM:~$ ls
ontent-az400-lab-resources
zureuser@linVM:~$ cd content-az400-lab-resources/
zureuser@linVM:~/content-az400-lab-resources$ git remote -v
rigin  https://github.com/ACloudGuru-Resources/content-az400-lab-resources.git (fetch)
rigin  https://github.com/ACloudGuru-Resources/content-az400-lab-resources.git (push)
zureuser@linVM:~/content-az400-lab-resources$ git remote remove origin
zureuser@linVM:~/content-az400-lab-resources$ git remote -v
zureuser@linVM:~/content-az400-lab-resources$ git remote add origin https://clouduserp4b842ee7@dev.azure.com/clouduserp
b842ee7/ASP_NET/_git/ASP_NET
zureuser@linVM:~/content-az400-lab-resources$ git push -u origin --all
assword for 'https://clouduserp4b842ee7@dev.azure.com':
numerating objects: 184, done.
ounting objects: 100% (184/184), done.
elta compression using up to 2 threads
ompressing objects: 100% (146/146), done.
riting objects: 100% (184/184), 1.20 MiB | 7.96 MiB/s, done.
otal 184 (delta 39), reused 150 (delta 25)
emote: Analyzing objects... (184/184) (212 ms)
emote: Validating commits... (8/8) done (1 ms)
emote: Storing packfile... done (97 ms)
emote: Storing index... done (44 ms)
o https://dev.azure.com/clouduserp4b842ee7/ASP_NET/_git/ASP_NET
* [new branch]      unit-test -> unit-test
ranch 'unit-test' set up to track remote branch 'unit-test' from 'origin'.
zureuser@linVM:~/content-az400-lab-resources$
```

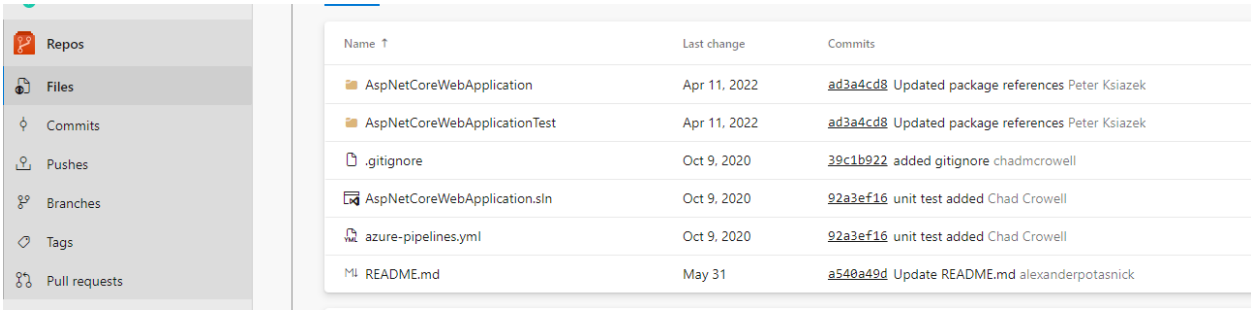
Return to the Linux VM and paste in the access token as above.

If you now use **git remote -v** you will see the git hub code again.

In **Azure DevOps**, click the **Azure DevOps** icon in the upper-left corner.

Select your project.

From the left-hand navigation menu, click **Repos**. Verify that the code was pushed to Azure Repos.



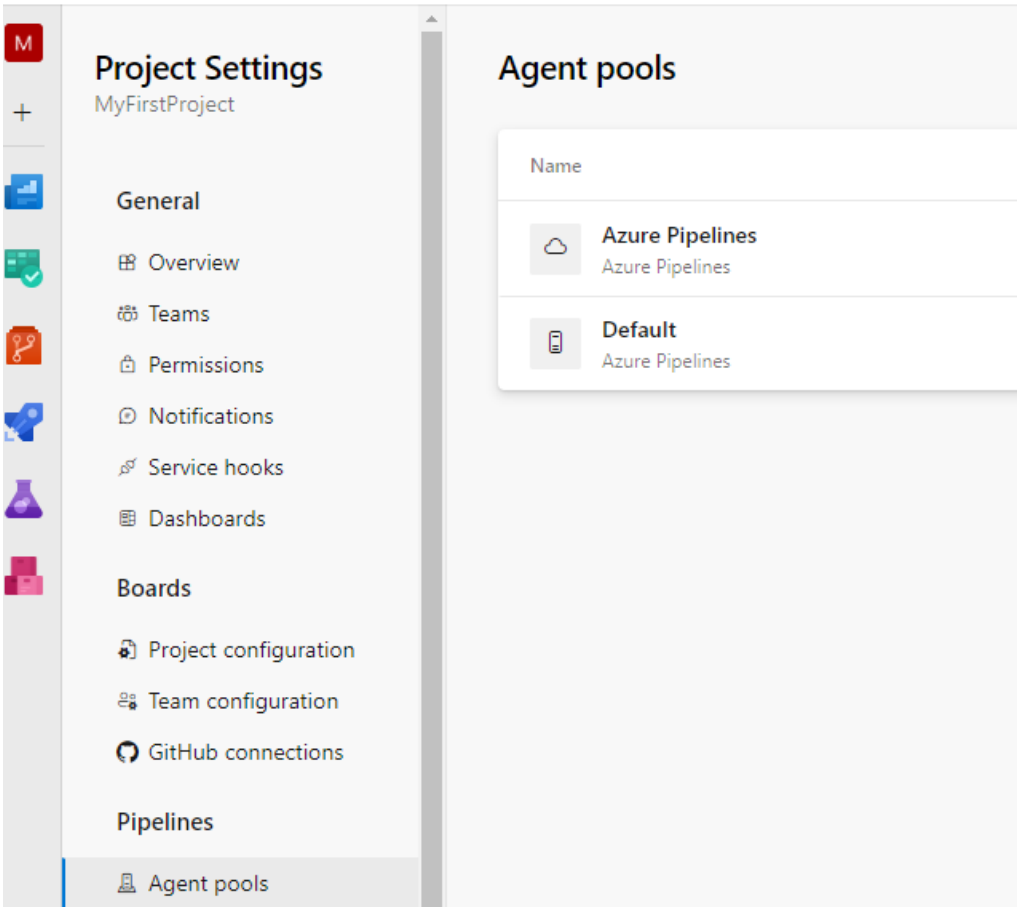
Create a New Azure DevOps Pipeline

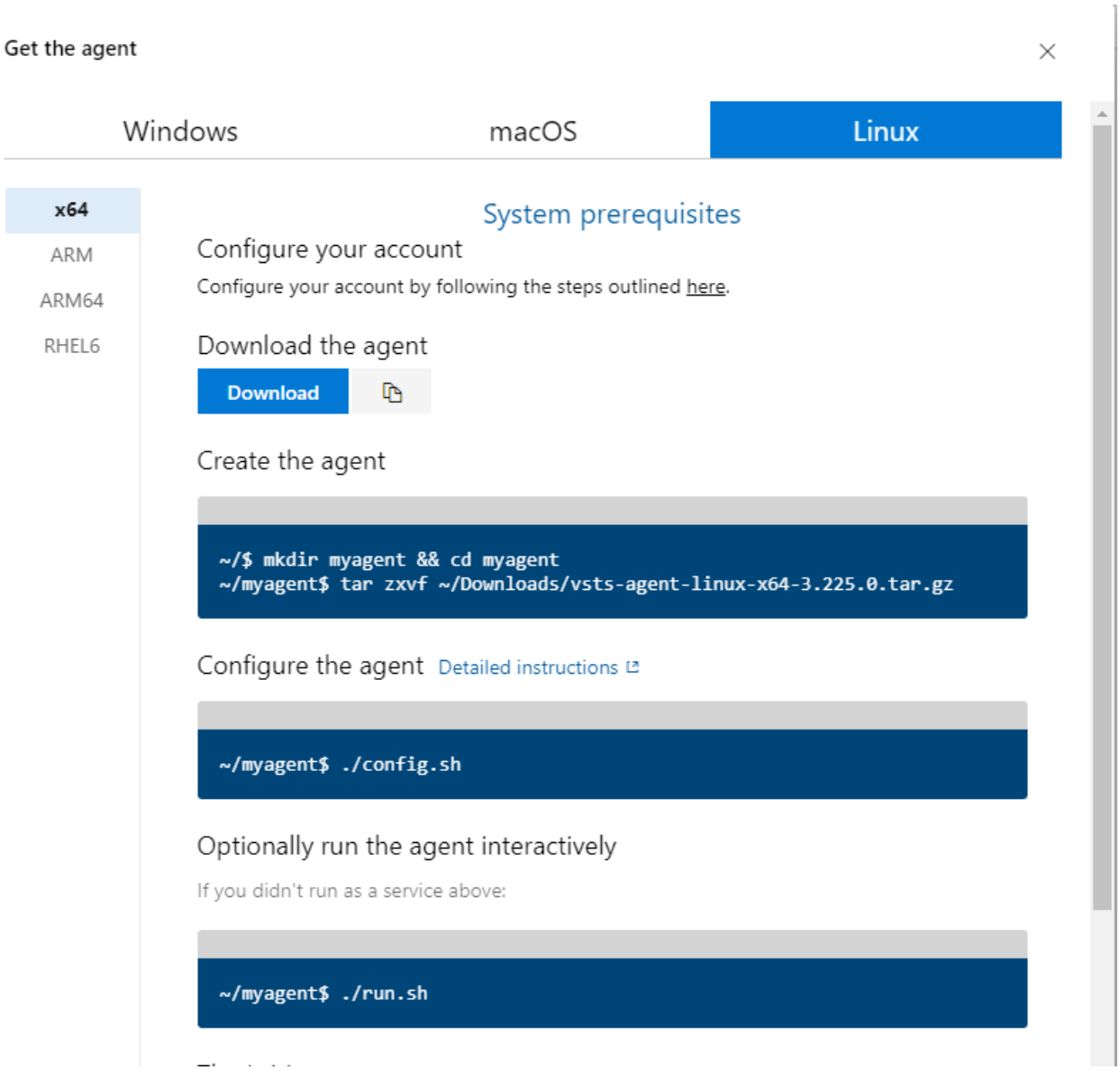
At the top of the page, click the **Azure DevOps** icon, then select the project.

On the bottom left, click **Project settings**.

Scroll down the left-hand pane and under **Pipelines**, select **Agent pools**.

Select the **Default** pool.





Click **New agent**.

Select the **Linux** tab.

Click the Copy icon, next to Download to copy the agent URL to your clipboard.

Paste it to a text file to use in the following steps.

https://vstsagentpackage.azureedge.net/agent/3.225.0/vsts-agent-linux-x64-3.225.0.tar.gz

Keep the **Linux** tab and popup open.

Go back to the Linux VM, and change directory to the root: **cd ~**

Make and navigate to a new directory called Downloads:

mkdir Downloads && cd Downloads

Run **wget** against the agent URL you just copied:

wget <AGENT_URL>

wget https://vstsagentpackage.azureedge.net/agent/3.225.0/vsts-agent-linux-x64-3.225.0.tar.gz

This will download the agent.

Navigate back to the root again by using **cd ~**.

Create and navigate to another directory called myagent:

mkdir myagent && cd myagent

Now copy the file downloaded from **downloads** to the **myagent folder**

example:

cp /home/cloud_user/Downloads/vsts-agent-linux-x64-3.225.0.tar.gz /home/cloud_user/myagent

NOTE: To work out the paths to use above use the command **pwd**.

Unzip the package you just downloaded. You can copy this same command from the popup in Azure DevOps to get the full package and version number:

tar zxvf <PACKAGE>

example:

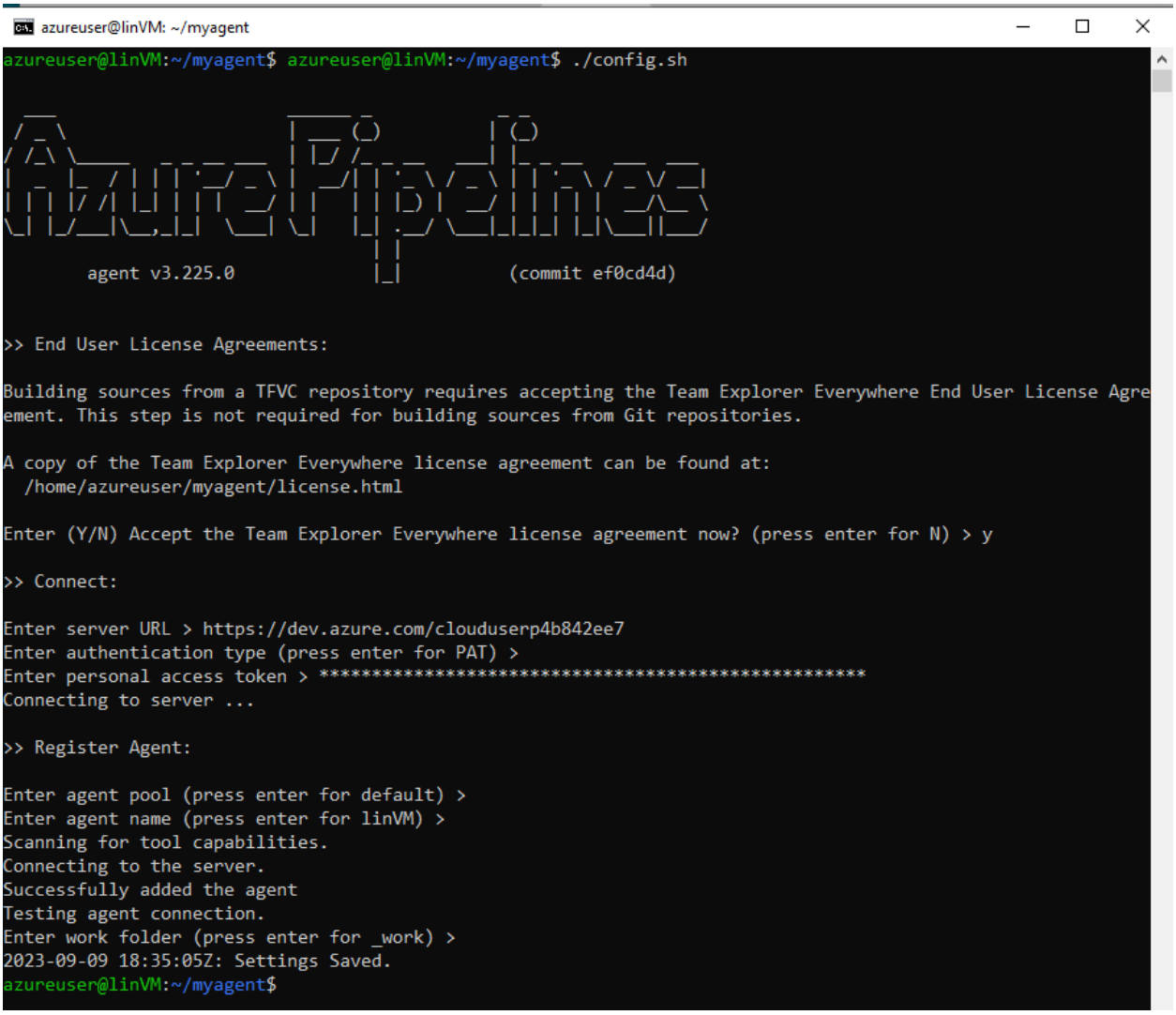
tar zxvf vsts-agent-linux-x64-3.225.0.tar.gz

```
cloud_user@linVM:~/myagent$ ls
bin  config.sh  env.sh  externals  license.html  run-docker.sh  run.sh  vsts-agent-linux-x64-3.225.0.tar.gz
cloud_user@linVM:~/myagent$
```

Review the contents of the directory (**ls**). Observe all of the files and folders for the agent.

Configure the Agent

Run the configuration script with the **config.sh** file:



./config.sh

Type **y** to accept the license agreement.

When prompted for your server URL, go back to Azure DevOps and grab the URL from your browser. It will look something like https://dev.azure.com/clouduser..., followed by some numbers (i.e., copy everything up to and not including /ASP_NET).

Paste this in the terminal.

Example:

https://dev.azure.com/clouduserp4b842ee7

NOTE: even adding / after the above URL will cause a failure and you will have start again!

Press **Enter** to accept the default authentication type of **PAT**.

For the personal access token option, copy and paste the personal access token you created and saved earlier in this lab.

example:

das5yco2kl474qngkvp3fikjev6vtxgymaverwa7kasm5nq3xfkq

NOTE: this sandbox environment was deleted after creating this lab. Hence the above PAT is useless.

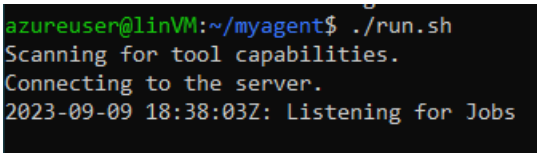
Press **Enter** to accept the default agent pool.

Press **Enter** to accept the default name.

Press **Enter** to accept the default work folder.

Run the agent interactively so that it is always listening for any jobs by using the following bash script.

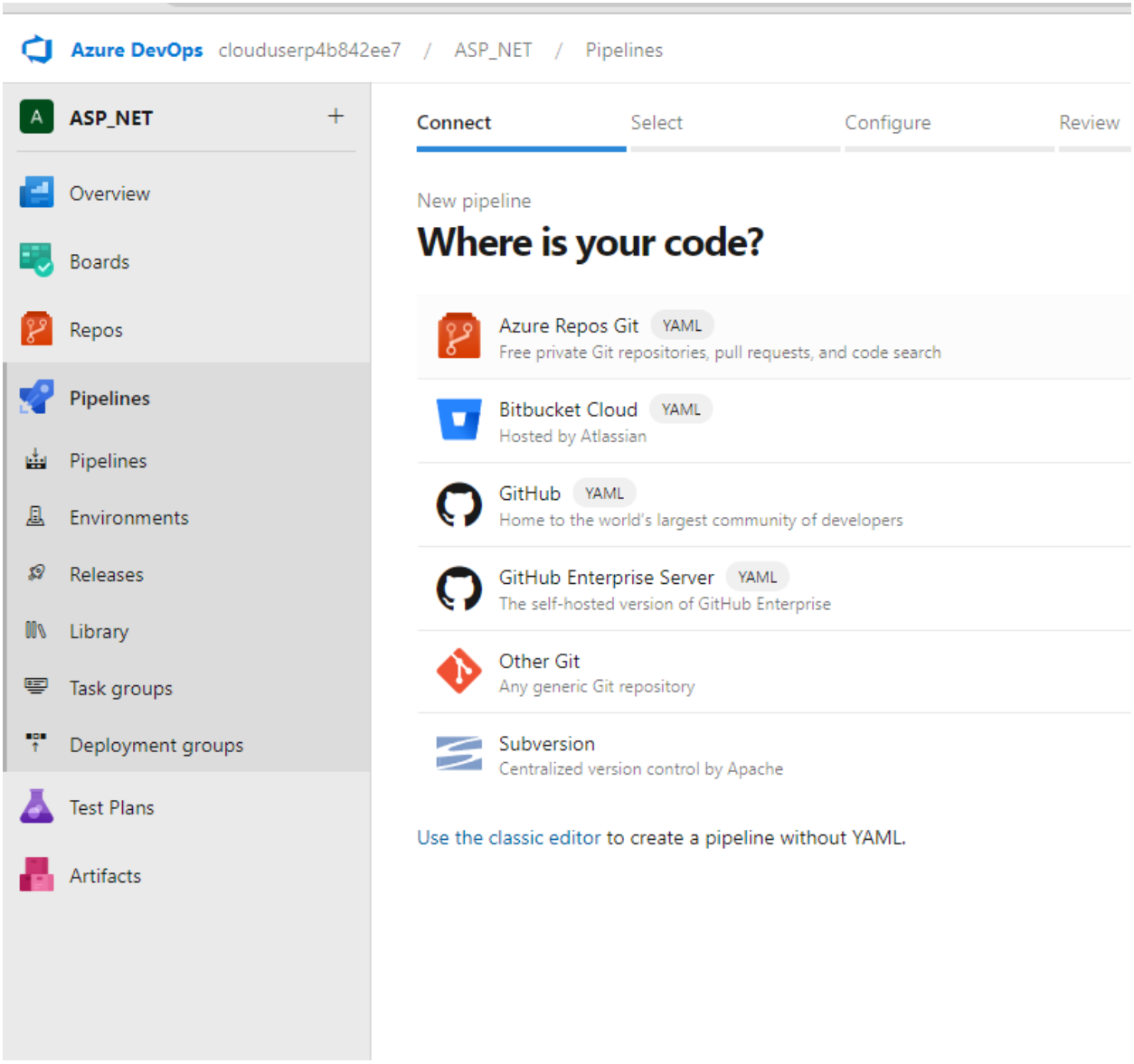
./run.sh



Create the New Pipeline

Go back to Azure DevOps, and in the left-hand navigation menu, click **Pipelines**.

Click **Create Pipeline.**



Click Use the **classic editor**.

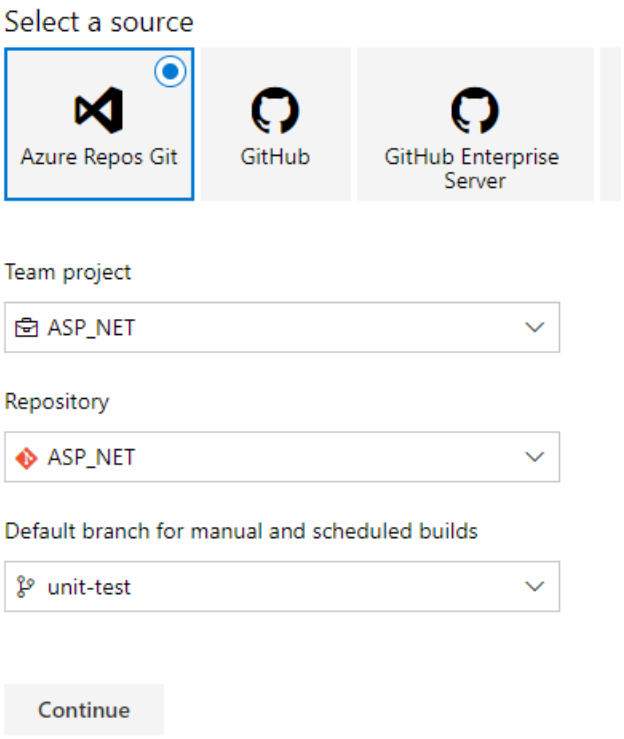
Under Select a source, select **Azure Repos Git** and make the following selections:

Team project: **ASP_NET**

Repository: **ASP_NET**

Default branch: **unit-test**

Click **Continue**.



Scroll down and select the **ASP.NET Core** template.

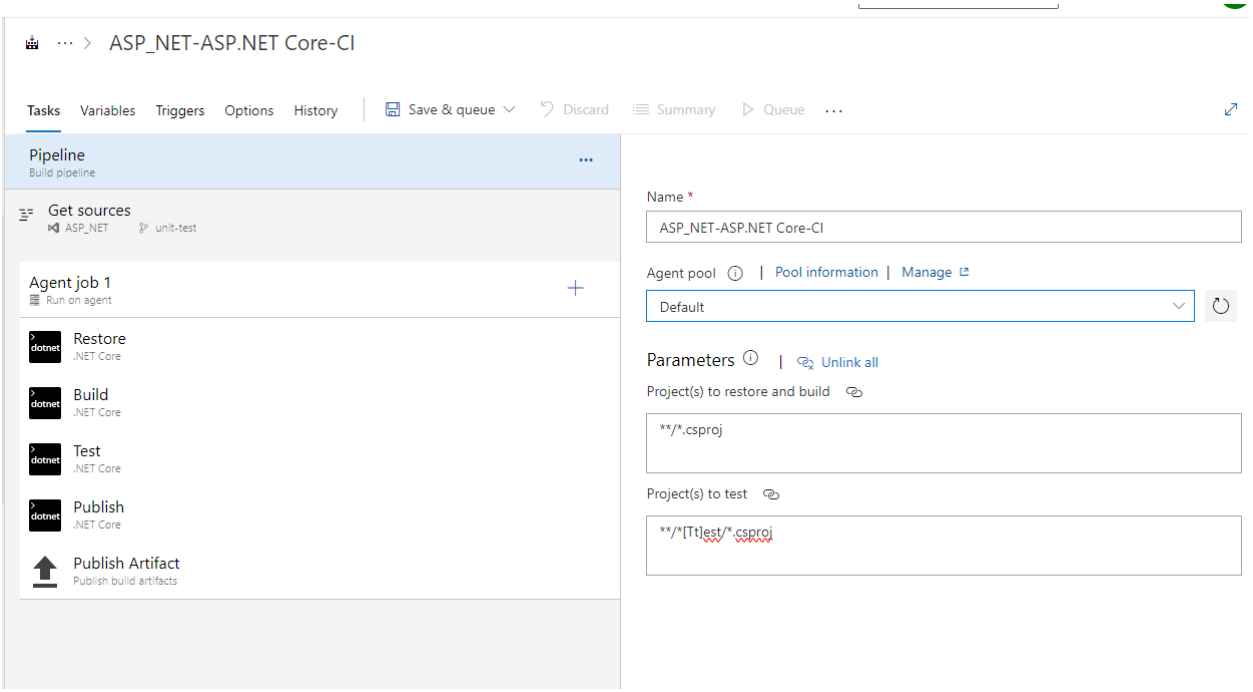
Click **Apply**.

On the right panel, for **Agent pool**, select **Default**.

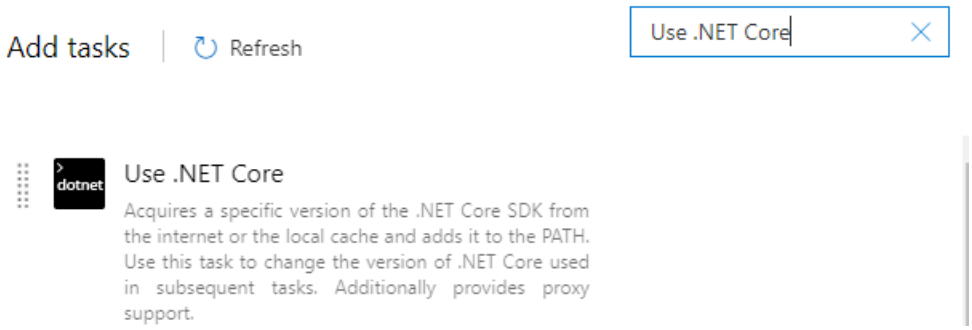
For Project(s) to test, enter: ****/*[Tt]est/*.csproj**.

NOTE: Please avoid typos as this will cause the job to fail!

Next to Agent job 1, click the **+** icon to add a new task.



Search for and click Add for the **Use .NET Core** task.

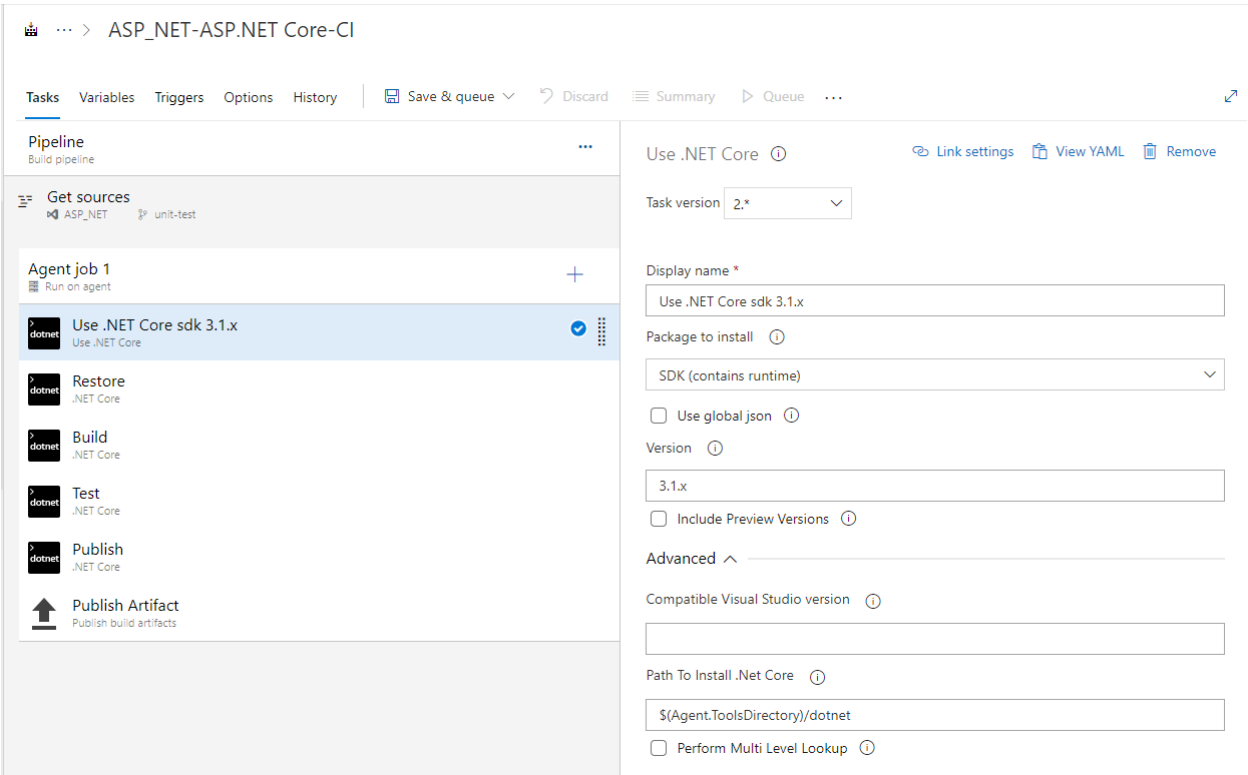


NOTE: Please ensure you use this legacy version.

In the task list for Agent job 1, drag and move the **Use .NET Core task** to the top.

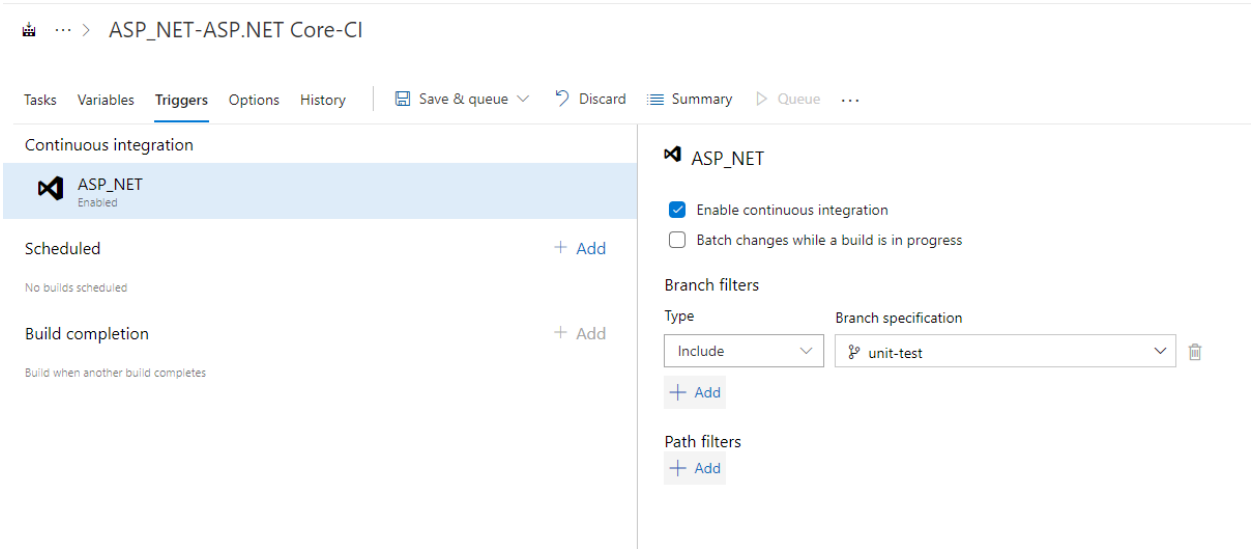
With the **Use .NET Core** task still selected, change the Version on the right to **3.1.x**.

At the top, click **Save & queue > Save > Save**.



Select the **Triggers** tab and check the box for **Enable continuous integration** on the right panel.

At the top, click **Save & queue > Save > Save**.



Update the Branch

On the left menu, select **Repos**.

Select the **AspNetCoreWebApplication** folder on the right.

Navigate through the folders to **Views > Home**.

Open the **Index.cshtml** file.

To the right of the code pane, click Edit.

Make a minor change in the **<h1> heading. After Congratulations! </h1>**, add You Did It!.

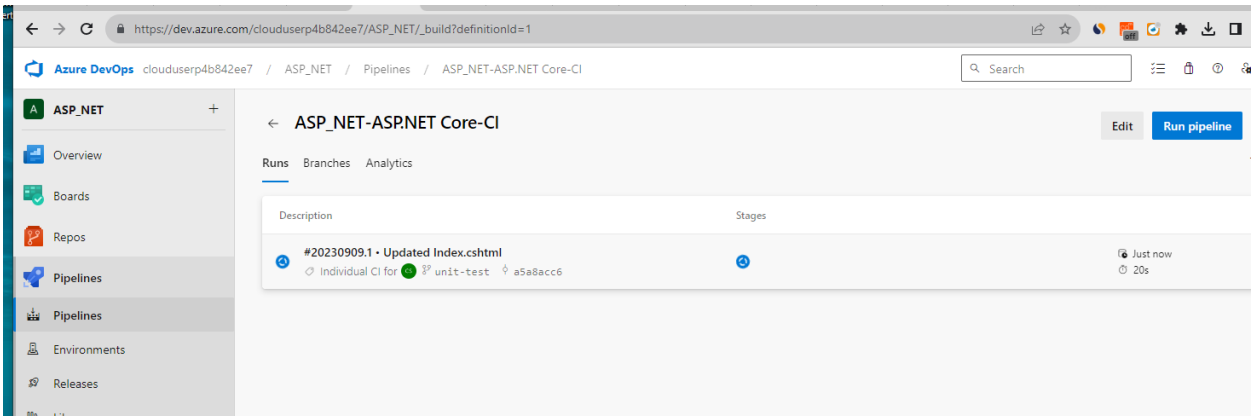
<h1> Congratulations! You Did It!</h1>

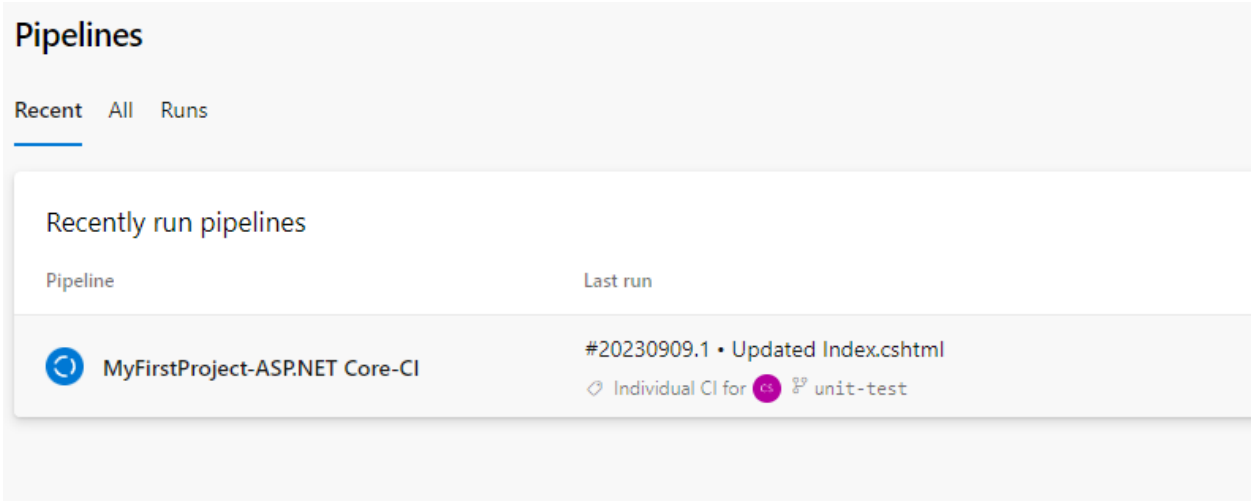
At the top, click **Commit > Commit**.

Run the Pipeline and Verify Success

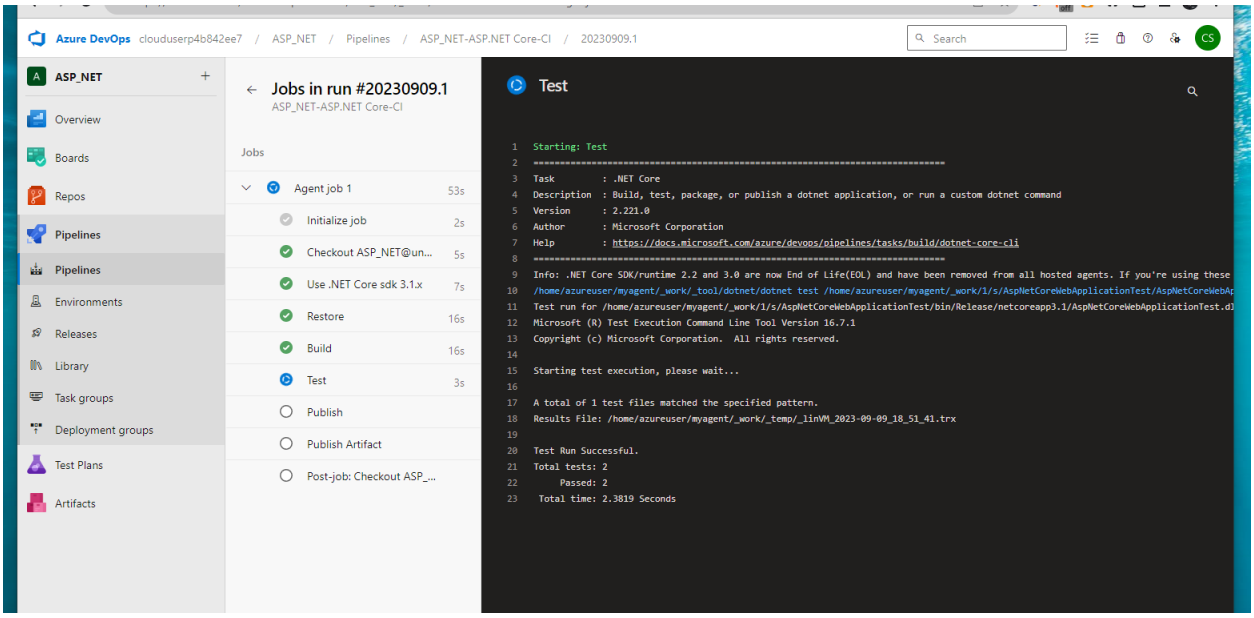
From the left navigation menu, select **Pipelines**.

Select the pipeline shown to open it. Then, select the currently running job.



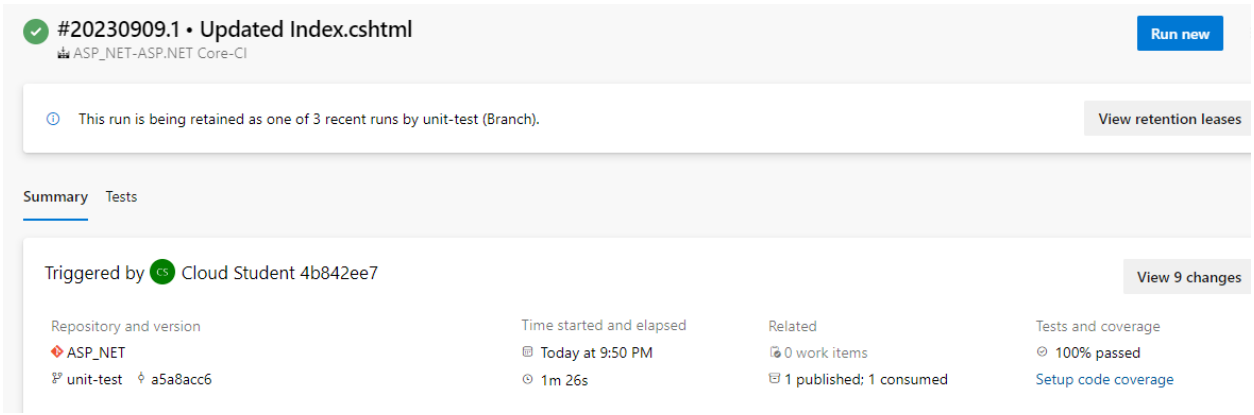


Select **Agent job 1** to view the progress.

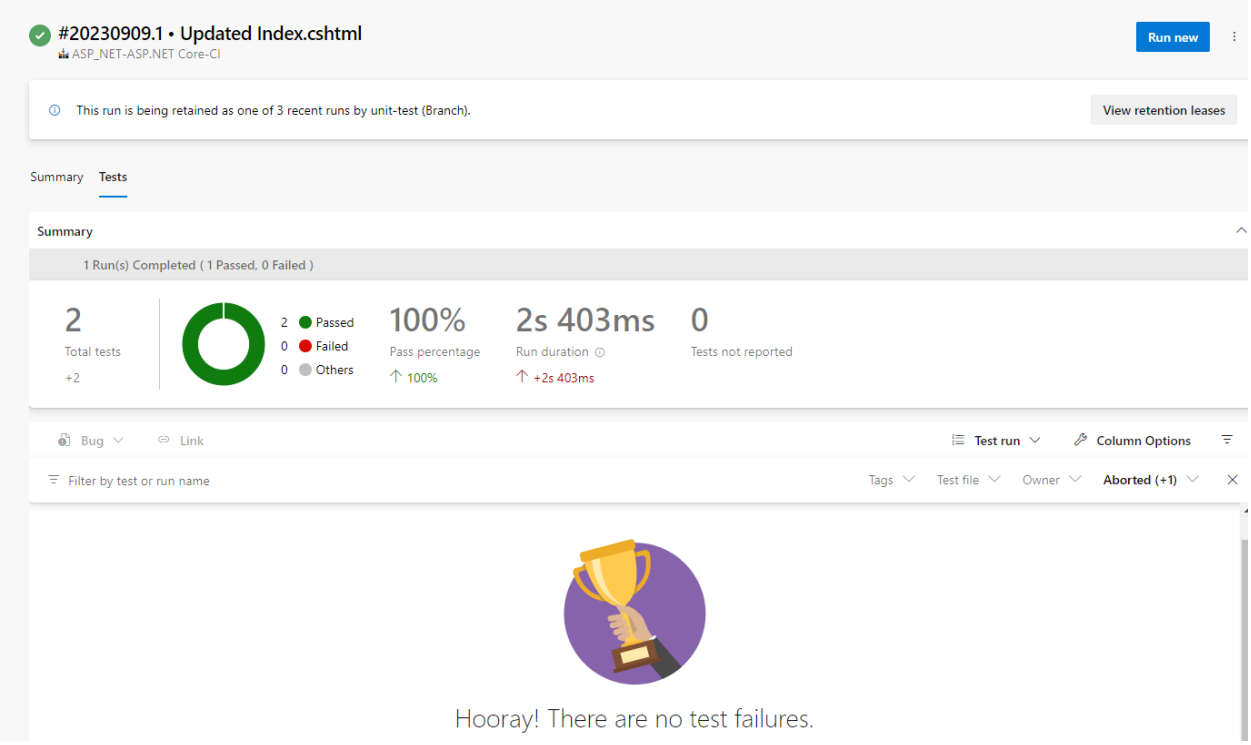


It will take a few minutes for the job to complete.

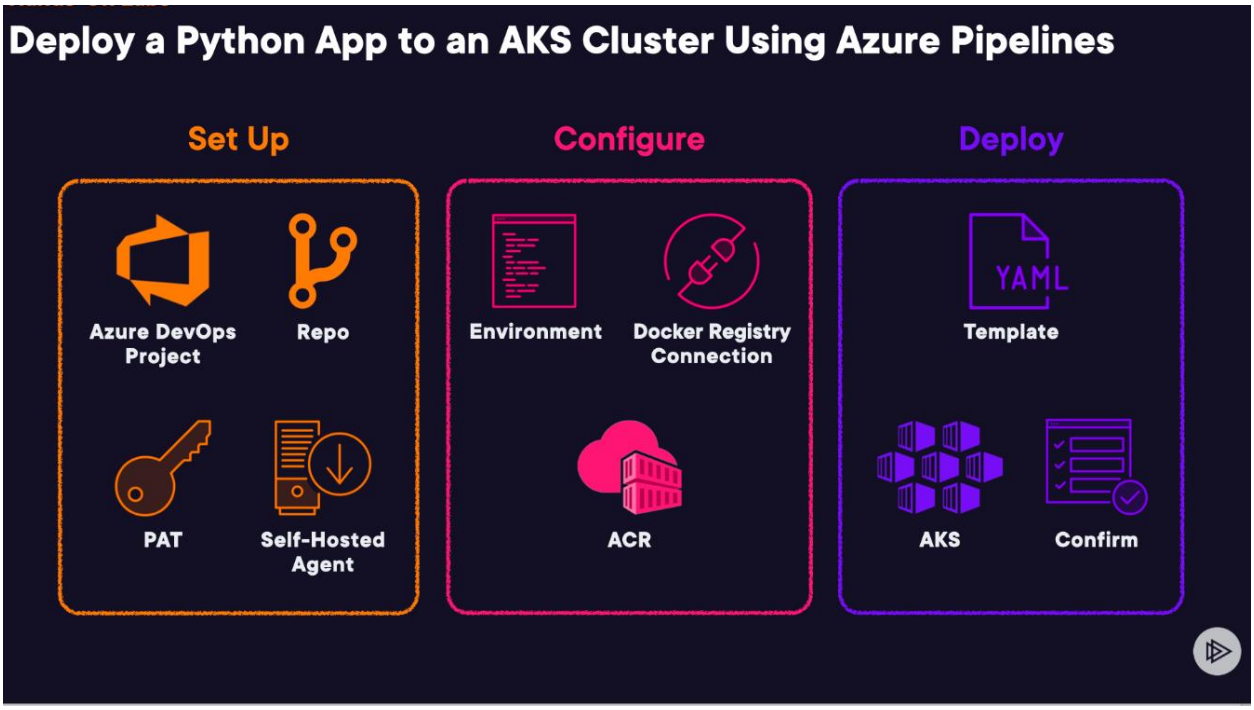
Once complete, click the back arrow above the job progress.



Select the **Tests** tab to verify success.



Deploy a Python App to an AKS Cluster Using Azure Pipelines



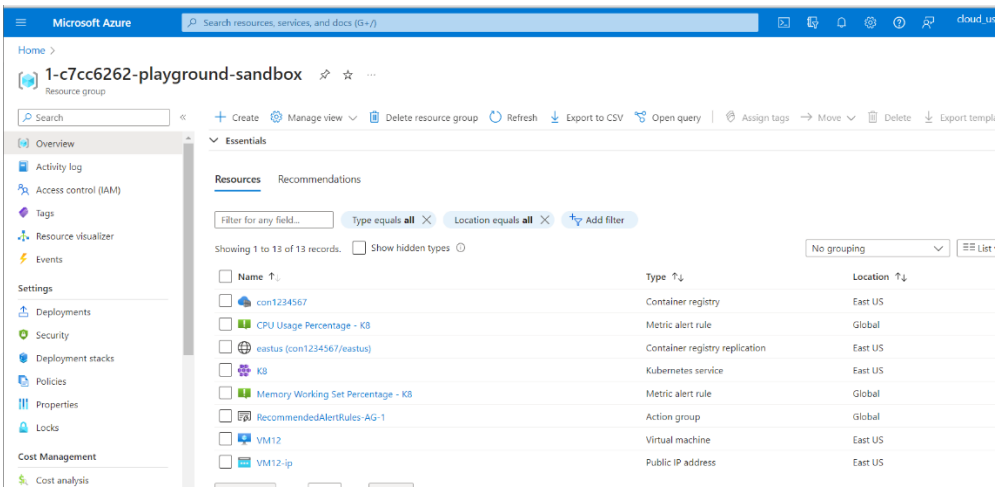
Pre steps

1. Create a Linux virtual machine (Ubuntu)

2. Create a Container Registry

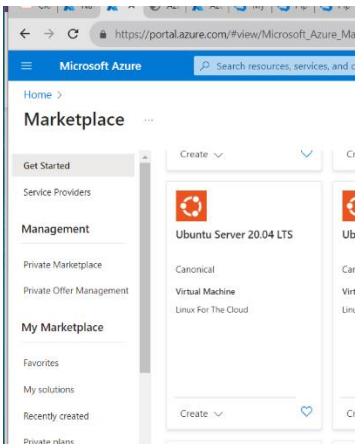
3. Create a Kubernetes service.

4.Setting up and configuring Docker on the Ubuntu server.

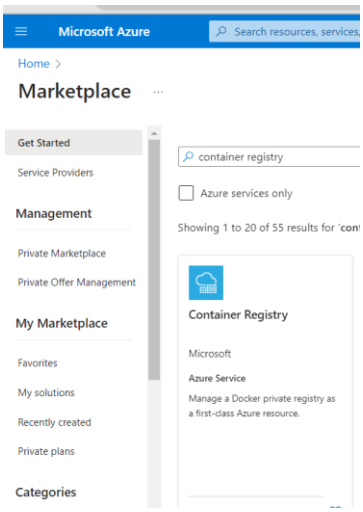


Login to the Azure portal sandbox and create the following resources we see above:

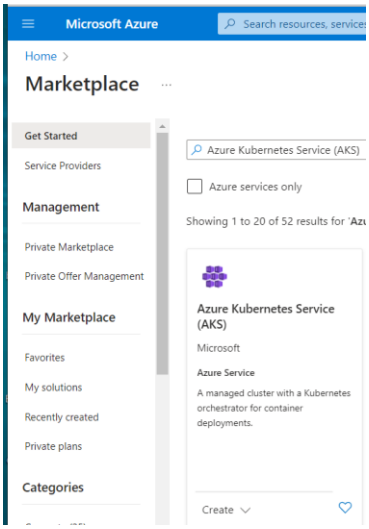
1. Create a Linux virtual machine (Ubuntu)



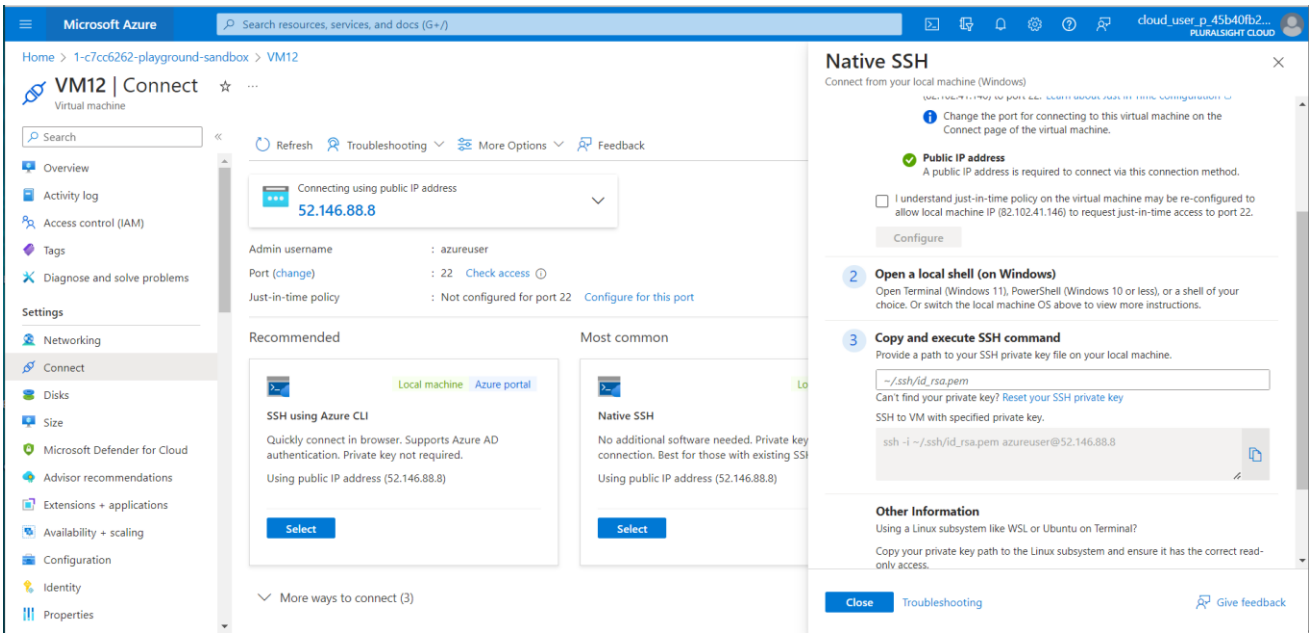
2. Create a Container Registry



3. Create a Kubernetes service.



Now, SSH into the virtual machine with the keypair



ssh -i <key_pair>@<Public IP Address>

cat /etc/os-release

```
azureuser@VM12:~$ cat /etc/os-release
NAME="Ubuntu"
VERSION="20.04.6 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.6 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
azureuser@VM12:~$
```

4.Setting up and configuring Docker on the Ubuntu server.

sudo apt update

**sudo apt install -y **
**apt-transport-https **
**ca-certificates **
**curl **
software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt update

sudo apt install docker-ce

docker --version

sudo systemctl enable docker
sudo systemctl start docker

whoami
azureuser

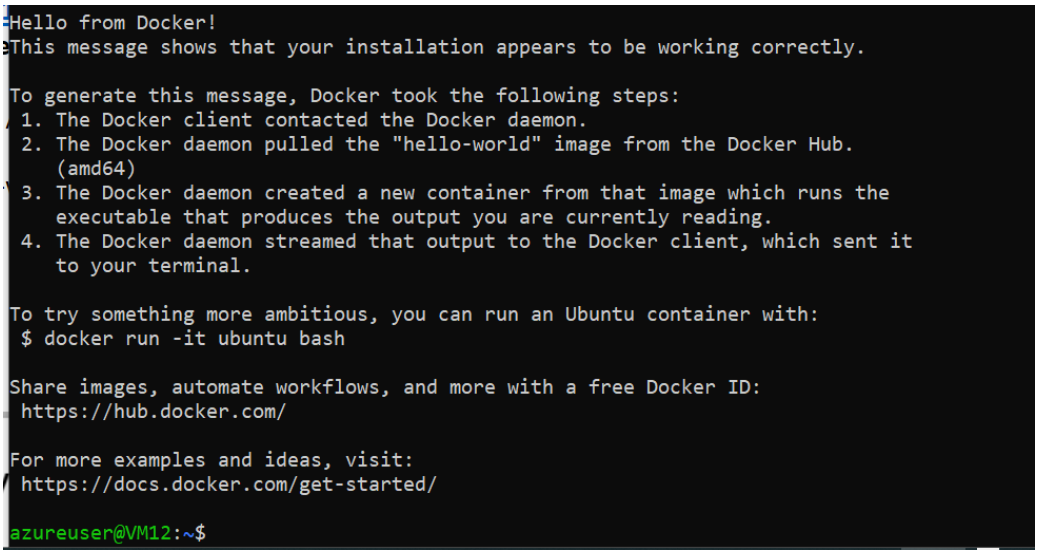
sudo usermod -aG docker azureuser
sudo systemctl restart docker

docker run hello-world

permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fvar%2Frun%2Fdocker.sock/v1.24/images/create?fromImage=tiangolo%2Fuwsgi-nginx-flask&tag=python3.6": dial unix /var/run/docker.sock: connect: permission denied

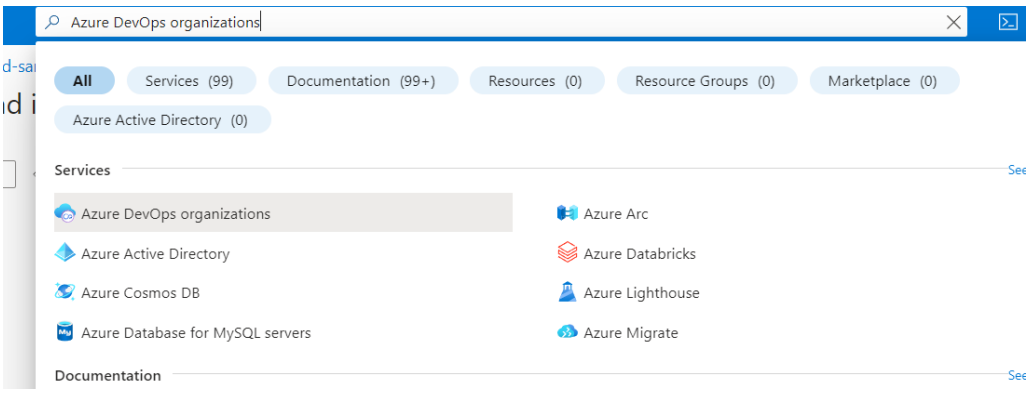
Run the following:
sudo chmod 666 /var/run/docker.sock

Now run the following hello-world command again.
docker run hello-world



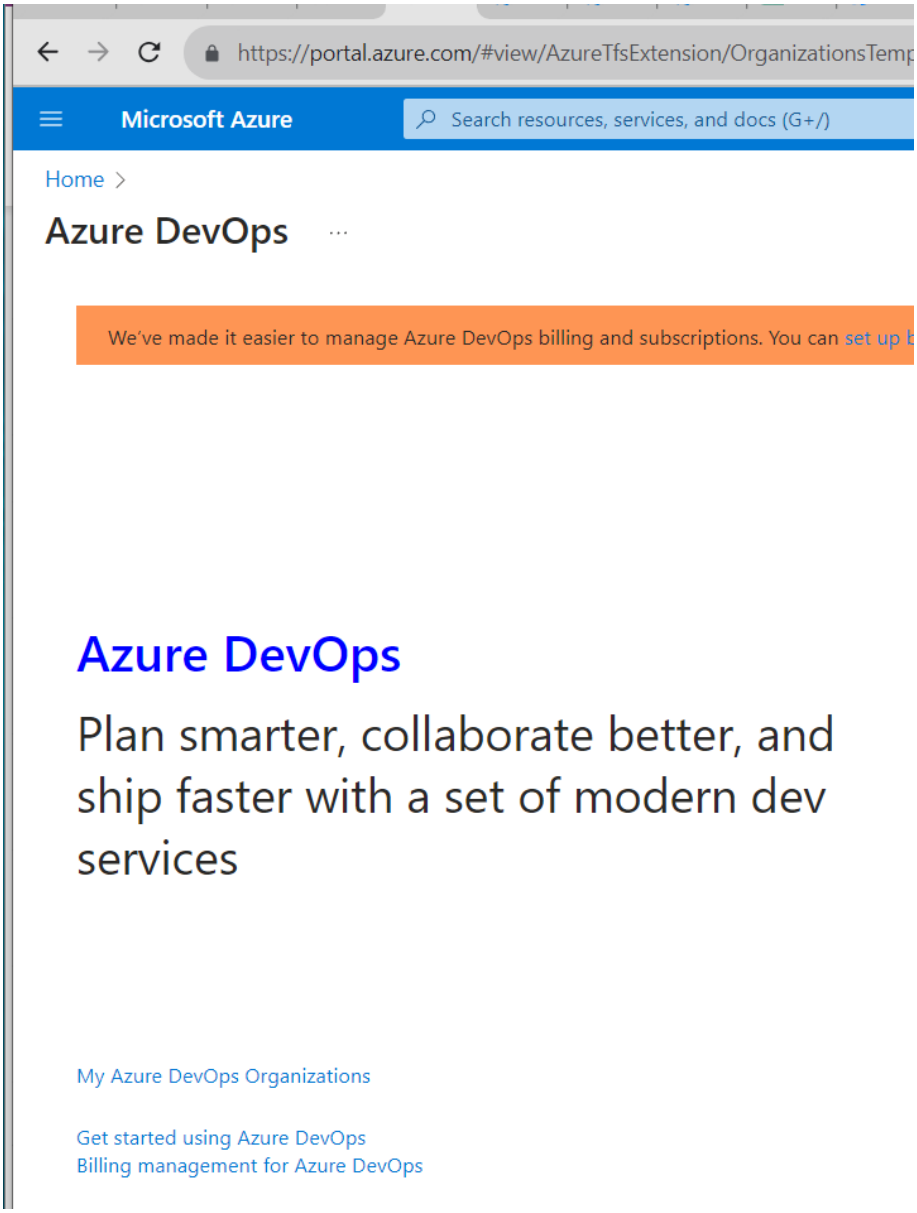
Now we have completed the pre steps lets now proceed to setting up the Azure pipeline.

Create an Azure DevOps Organization



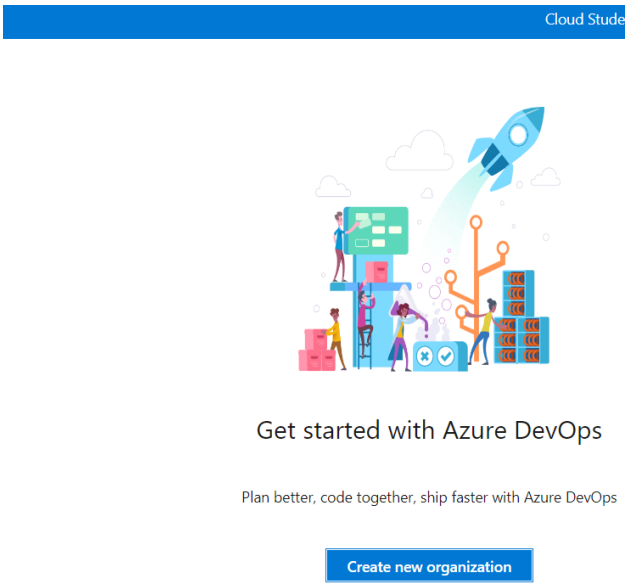
In the main search bar, search for and select **Azure DevOps organizations**.

Scroll down and click **My Azure DevOps Organizations**.



Select your **country/region** from the dropdown list.

Select Create **new organization** > **Continue** > **Continue** (complete the required **CAPTCHA**).



Set the following values to create a project:

Project name: **Dev**
Visibility: **Private**
Click + **Create project**.

Import Code and Set Up the Environment

In the sidebar menu, select **Repos**.

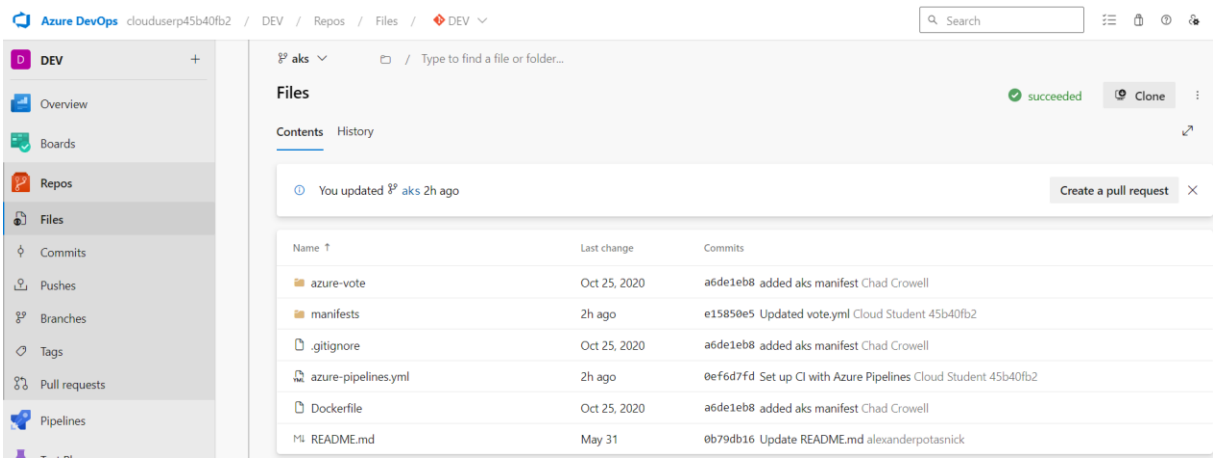
In the Import a repository section, click **Import**.

In the Clone URL field, enter the following repository:

https://github.com/ACloudGuru-Resources/content-az400-lab-resources.git

Click **Import**.

After the code import completes successfully, the Files page should open automatically.



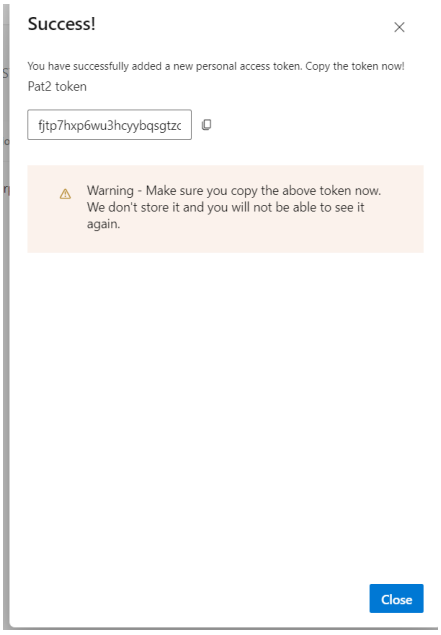
At the top of the page, use the dropdown to switch the branch from **DSC** to **aks**.

You should now see the files you need for the lab as above.

Create Personal Access Token
In the upper right-hand corner, click the **User settings icon > Personal access tokens**.

Click + **New Token** and set the following values:

Name: **pat1**
Scopes: **Full access**
Click **Create**.



Copy the **pat1** token and save it in a safe location.

Click **Close**.

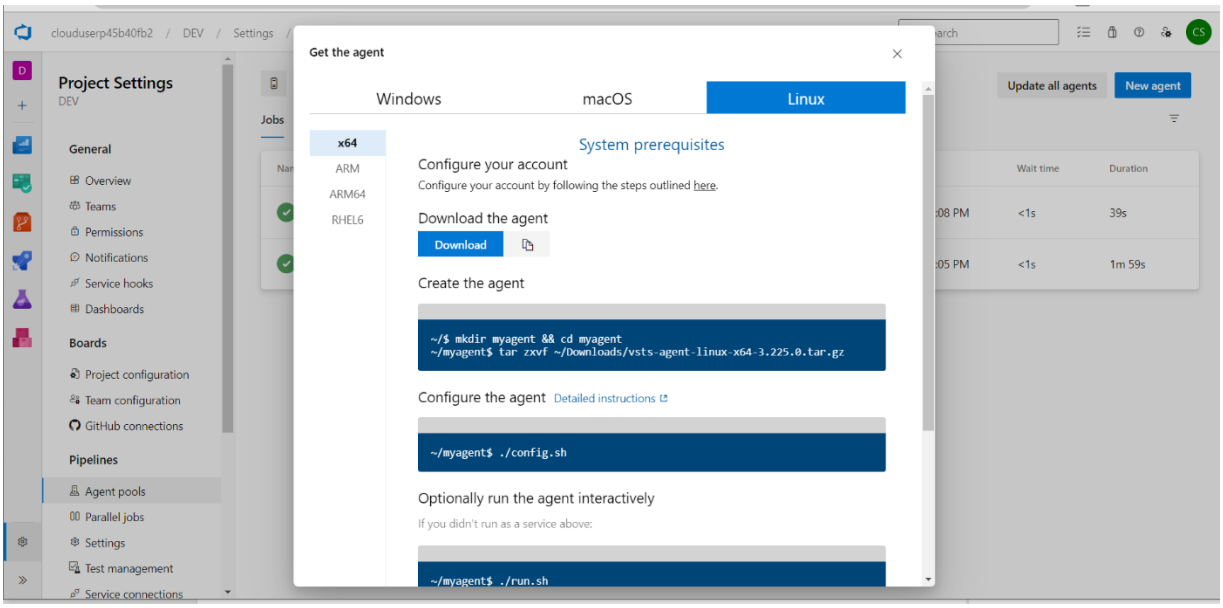
Download Self-Hosted Agent
At the top of the page, click the **Azure DevOps icon**, then select the **project**.

On the bottom left, click **Project settings**.

Scroll down the left-hand pane and under **Pipelines**, select **Agent pools**.

Select the **Default pool**.

Click **New agent**.



Select the **Linux** tab.

Click the Copy icon, next to **Download** to copy the agent URL to your clipboard.
Paste it to a text file to use in the following steps. Keep this tab and popup open.

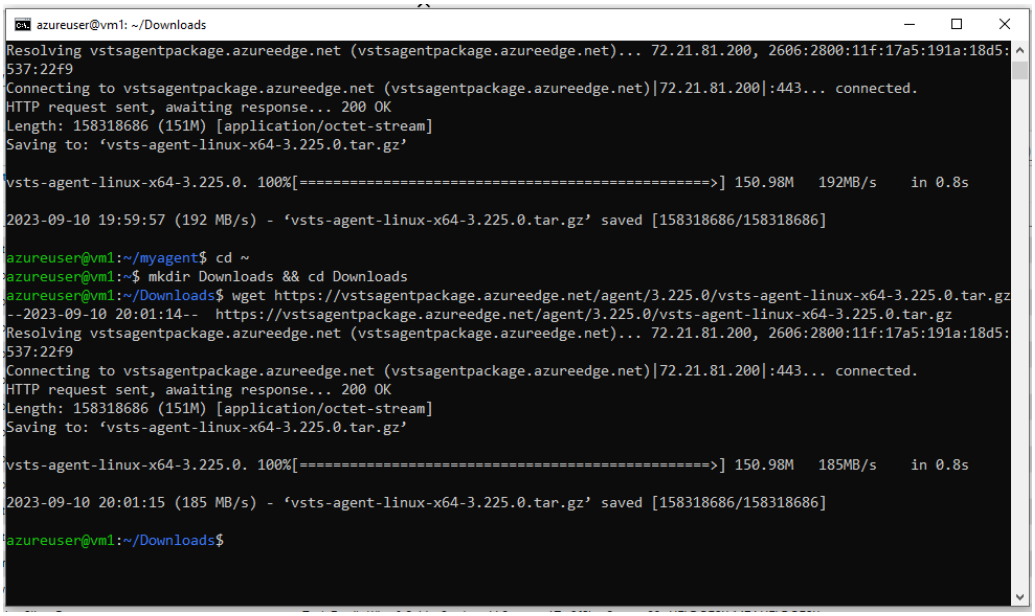
For example:
https://vstsagentpackage.azureedge.net/agent/3.225.0/vsts-agent-linux-x64-3.225.0.tar.gz

Setup the agent on the virtual machine

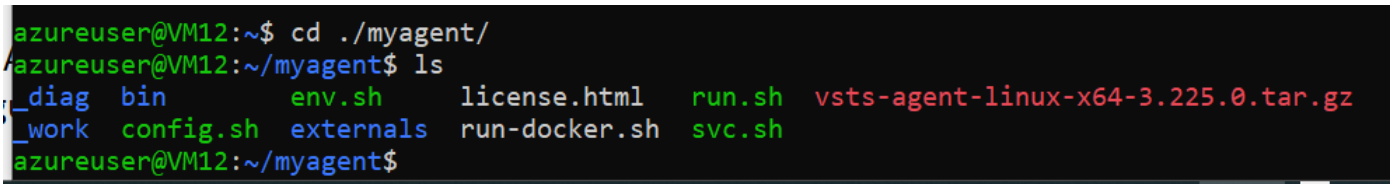
Ensure the directory is set to the root:
cd ~

Create and navigate to another directory called myagent:
mkdir myagent && cd myagent

Download the tar file within the agent directory.
wget <AGENT_URL>
wget https://vstsagentpackage.azureedge.net/agent/3.225.0/vsts-agent-linux-x64-3.225.0.tar.gz

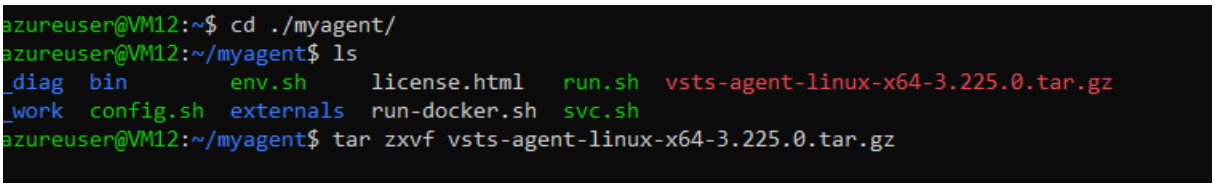


Unzip the package you just downloaded with the following command:



tar zxvf <PACKAGE>

You can also copy this same command from the popup in Azure DevOps to get the full package and version number:

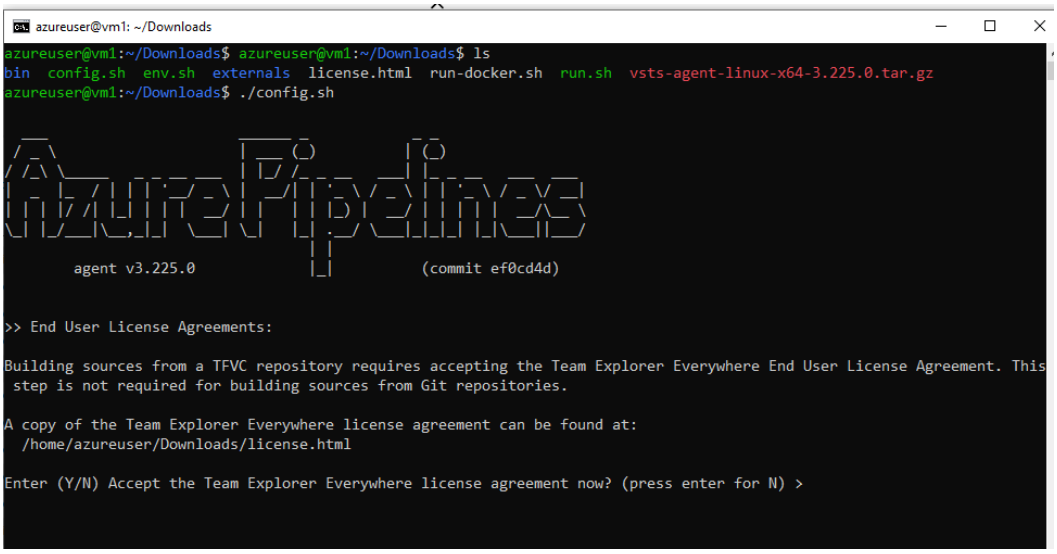


Review the contents of the directory (**ls**). Observe all the files and folders for the agent.

Configure the Agent
Run the configuration script with the following command:

./config.sh

Type y to accept the license agreement.



When prompted for your server URL, go back to Azure DevOps and grab the URL from your browser. It will look something like https://dev.azure.com/clouduser..., followed by some characters (i.e., copy everything up to but not including /dev).

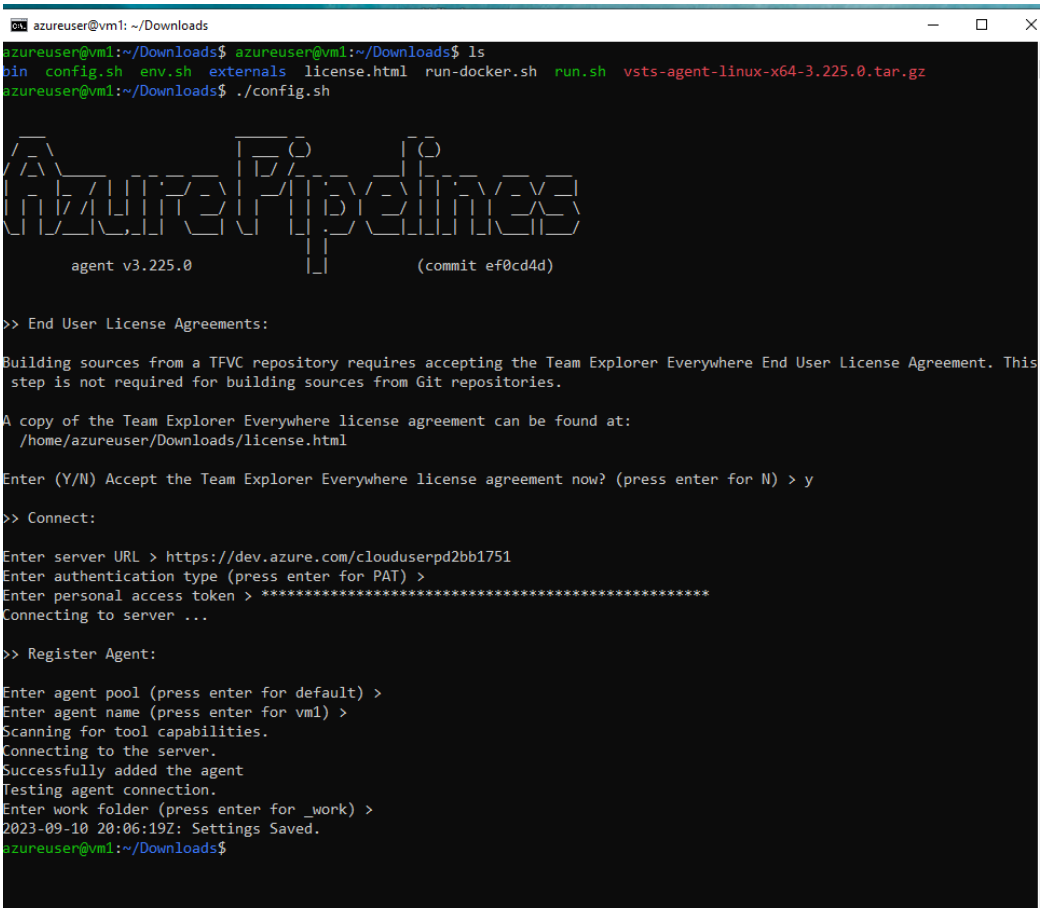
For example:

https://dev.azure.com/clouduserp45b40fb2

Paste this in the terminal

Press **Enter** to accept the default authentication type of **PAT**.

For personal access token, copy and paste the personal access token you created and saved earlier.



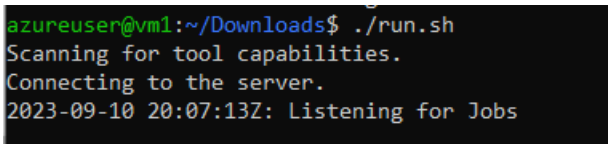
Press Enter to accept the default **agent pool**.

Press Enter to accept the default **name**.

Press Enter to accept the default **work folder**.

Now run the agent interactively so that it is always listening for any jobs:

./run.sh

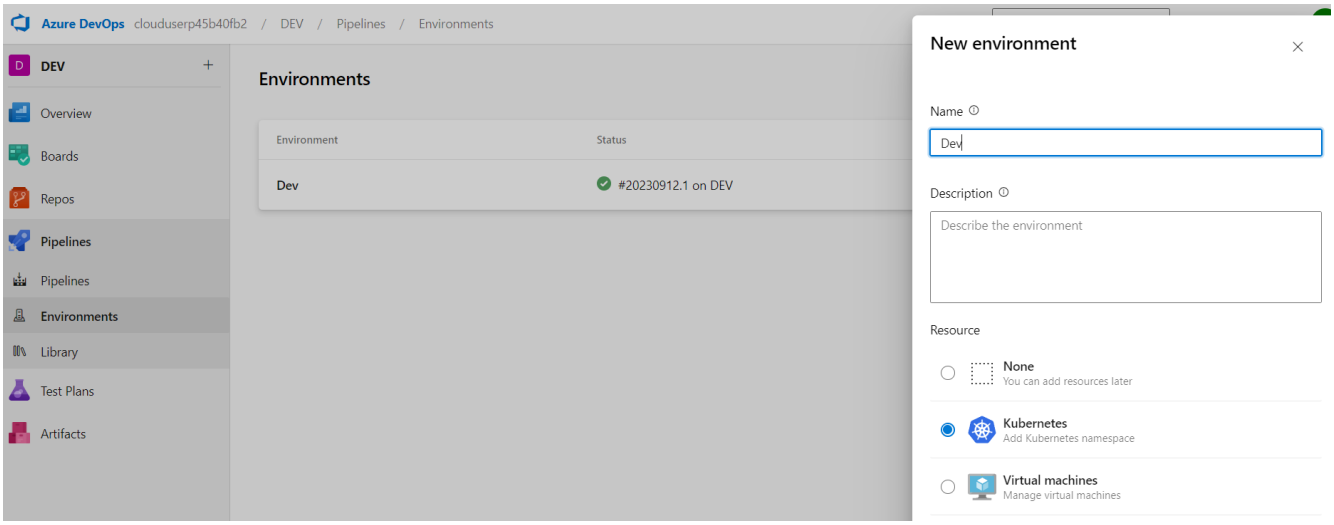


Now let's go back to the DevOps portal

Create Pipeline Environment

In the sidebar menu, select **Pipelines**, then select **Environments**.
Click **Create environment**.
Fill in the environment details:

Name: Enter **dev**.
Resource: Select **Kubernetes**.
Click **Next**.
Ensure the resource details auto-populate, then click **Validate and create**.



Create a Service Connection and an ACR
In the bottom-left corner, click **Project Settings**.

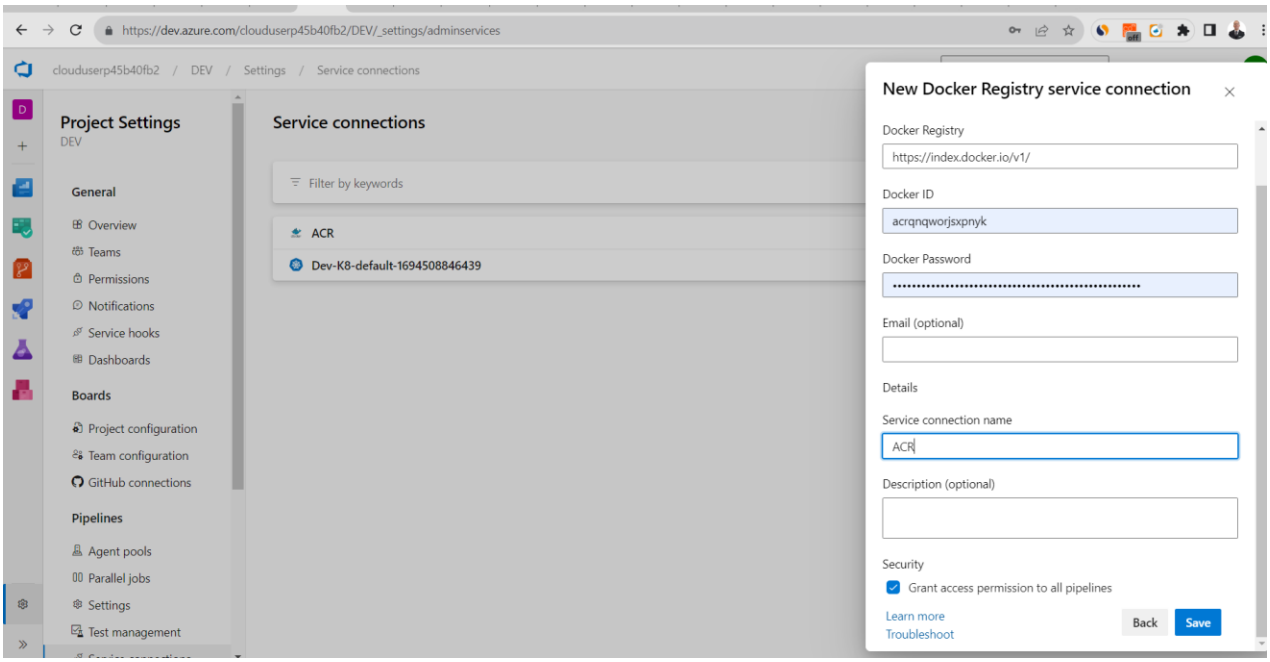
This opens the settings in a new tab.

In the sidebar menu, select **Service connections**.

In the top right, click **New service connection**.

In the pane on the right, select **Docker Registry**, then click **Next**.

You'll need to gather some information from an **Azure Container Registry (ACR)** before you finish creating your service connection.

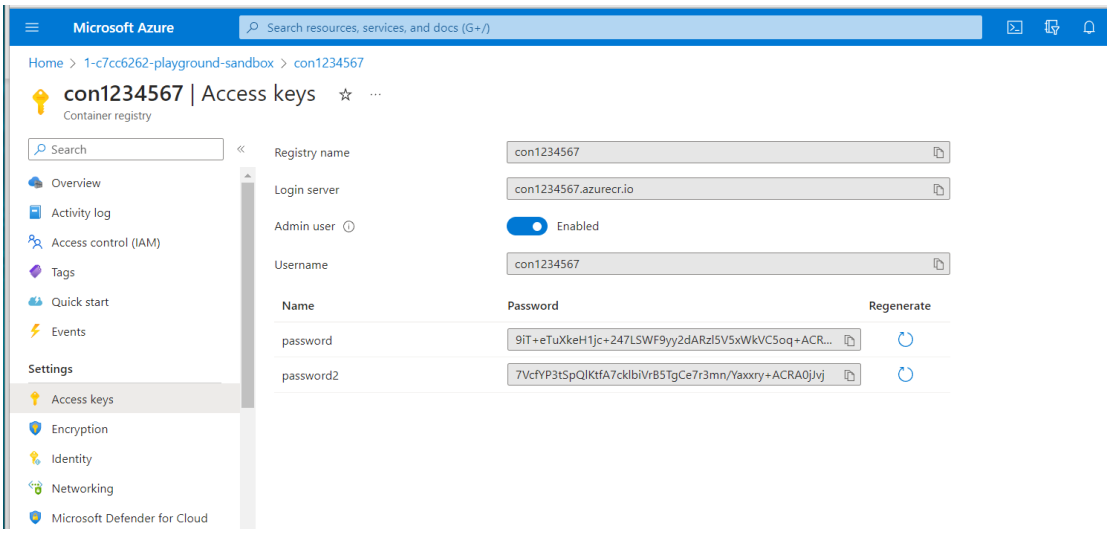


Navigate back to your Azure DevOps tab.

Navigate to **All resources** using the hamburger menu in the top-left corner.

Select the link for the **Azure container registry (ACR)**.

In the ACR's sidebar menu, select **Access keys** (under **Settings**).
You'll use these details for your service connection.



Add the New Docker Registry service connection details:

Fill in the Docker Registry:
From the Access keys page, copy the **Login server**.

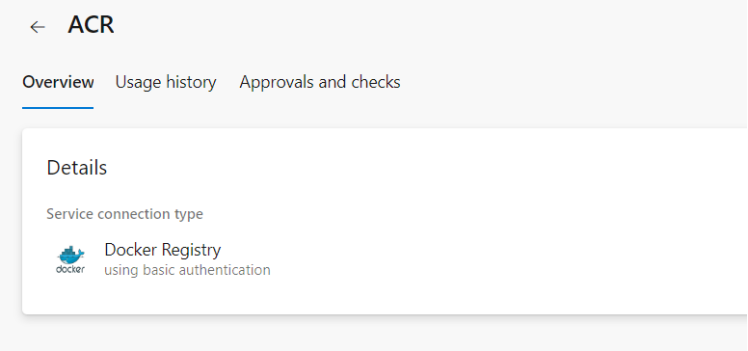
Navigate to the service connections tab and paste the login server into the **Docker Registry** field using the format https://<LOGIN_SERVER>.

Fill in the Docker ID:
Navigate to the ACR tab and copy the **Username**.
Navigate to the service connections tab and paste the username into the **Docker ID** field.

Fill in the Docker Password:
Navigate to the ACR tab and copy either **password**.
Navigate to the service connections tab and paste the password into the **Docker Password** field.

In the Service connection name field, enter **ACR**.

Below Security, check the **Grant access permission to all pipelines** checkbox.
Click **Save**.



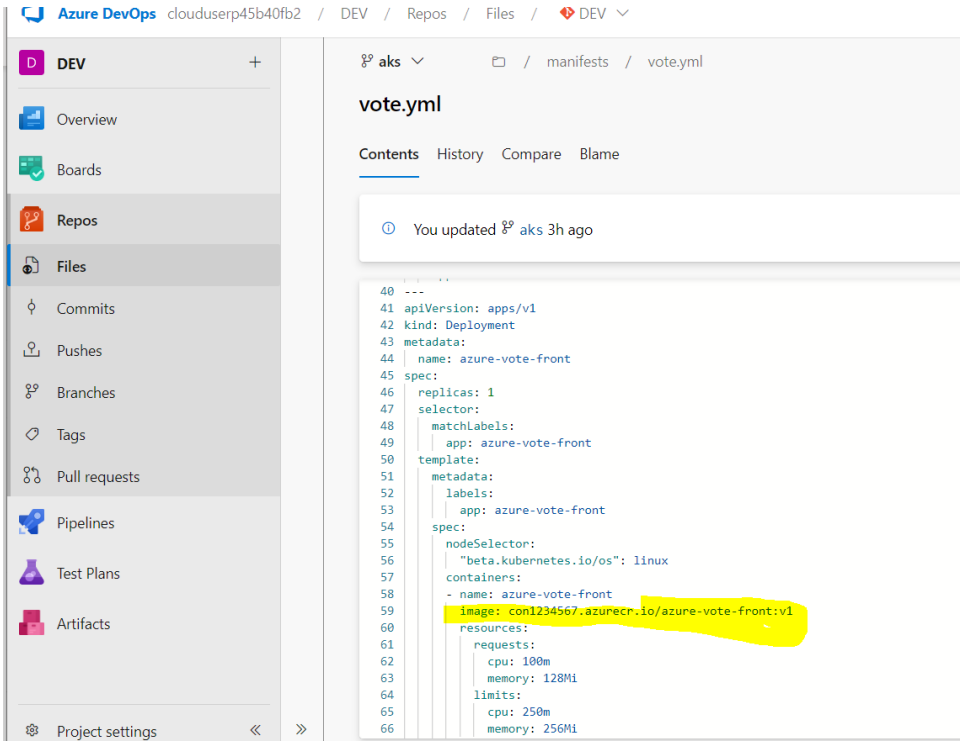
Create the CI/CD Pipeline
Update the Deployment Image
From the service connections tab, use the sidebar menu to select **Repos**.

Select the **manifests** folder and open the **vote.yml** file.

In the top right, click **Edit**.

Update the deployment image:

Navigate to the **ACR** tab and copy the **Login server** again.
Navigate to the **vote.yaml** tab and replace the container image on line 59 with the Azure ACR DNS registry name, following the format **<LOGIN_SERVER>/azure-vote-front:v1**.



Note: The ACR DNS name ends with azurecr.io.

In the top right, click **Commit**.

In the Commit pane, ensure the Branch name is set to **aks**, then click **Commit**.

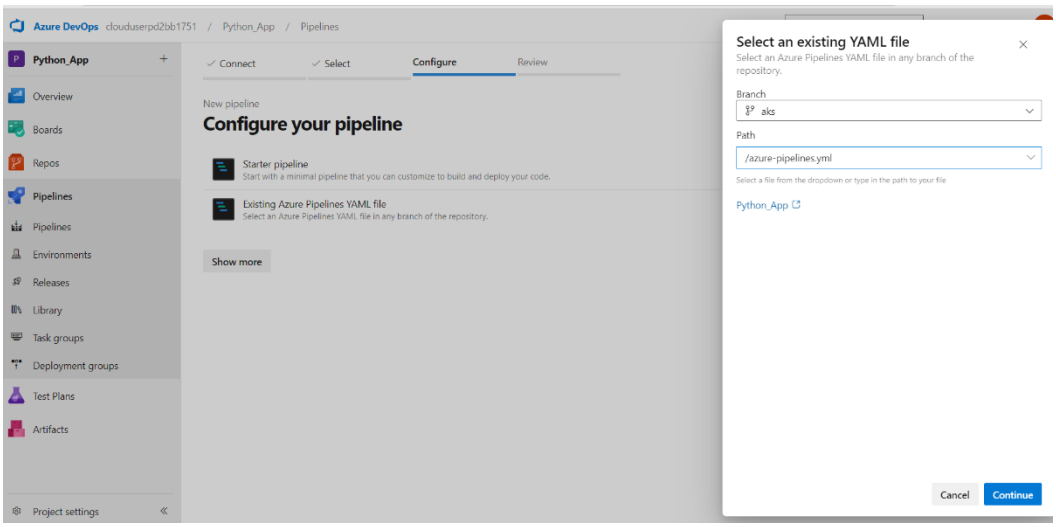
Create the Pipeline
In the sidebar menu, select **Pipelines**.

Click **Create Pipeline**.

For the code location, select **Azure Repos Git**.

For the repository, select the dev repo.

Select **Existing Azure Pipelines YAML file**.

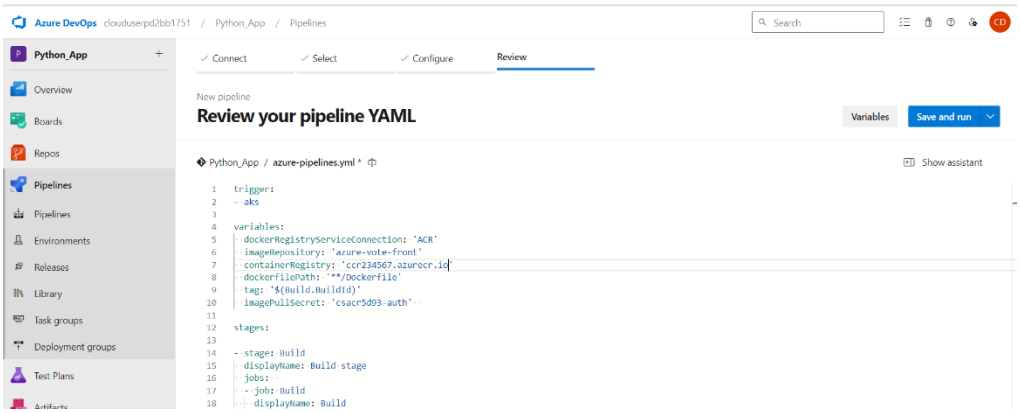


Fill in the YAML file details:

Branch: Select **aks**.
Path: Select **/azure-pipelines.yml**.
Click **Continue**.

Update the **azure-pipelines.yml** code:

Navigate to the **ACR** tab and copy the **Login server** again.
Navigate to the **pipeline** tab and replace the value for container Registry with your copied Azure ACR DNS registry name.



In the top-right corner, click **Save and run**.

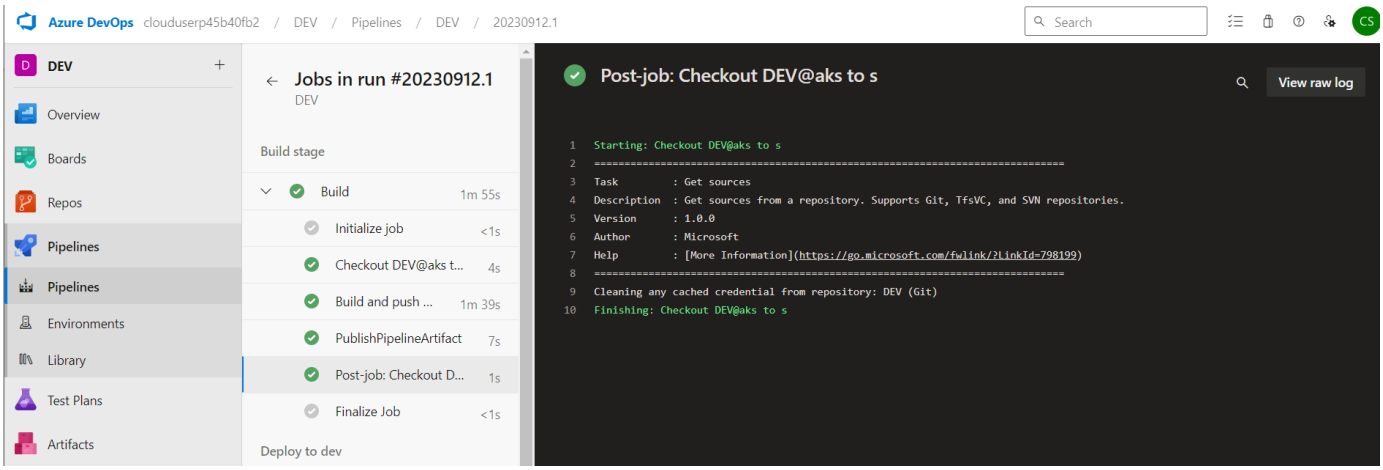
In the pane on the right, leave the default settings and click **Save and run**.

After the pipeline is triggered, select **Build stage** to monitor the build progress.

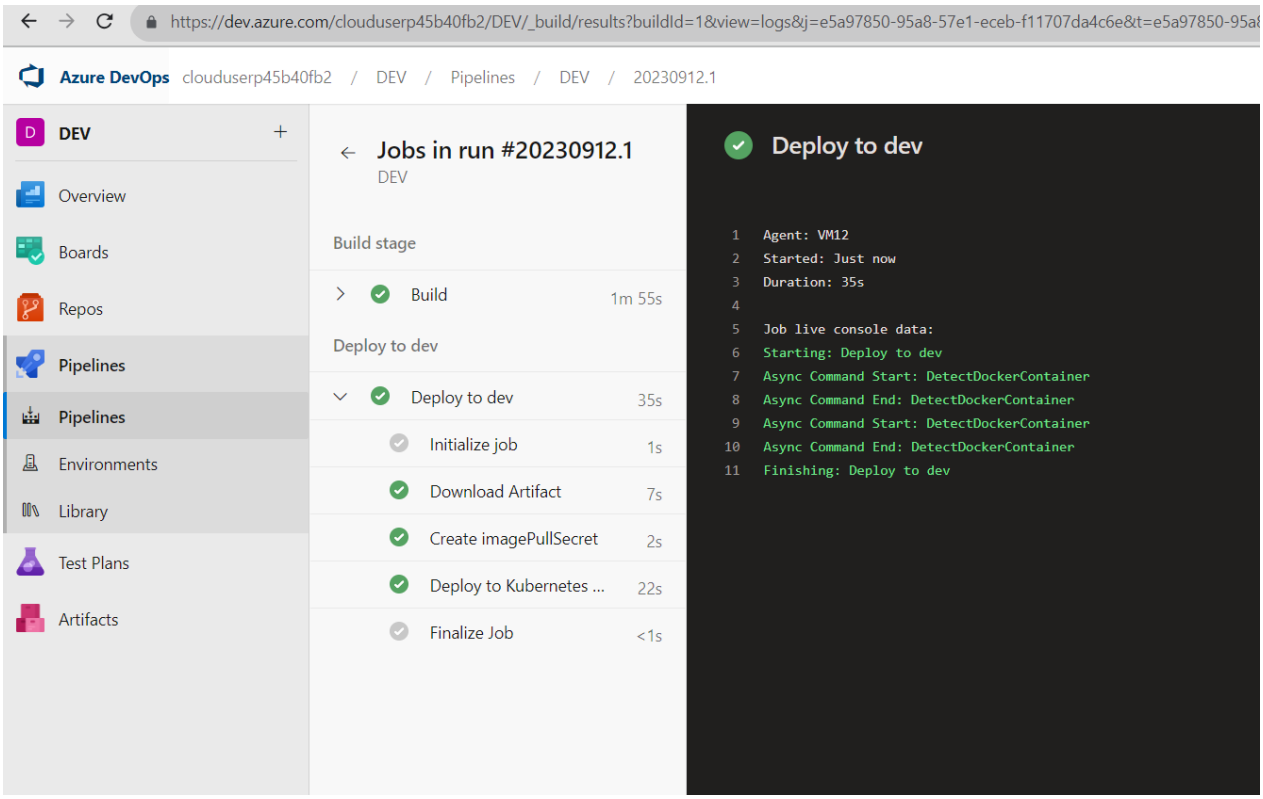
Note the banner in the terminal indicating that the pipeline needs additional permissions to continue the build.

Click **View** to the right of the banner.

Click **Permit** to the right of the requested permissions and confirm the selection.



After the build has completed, select **Deploy to dev**.

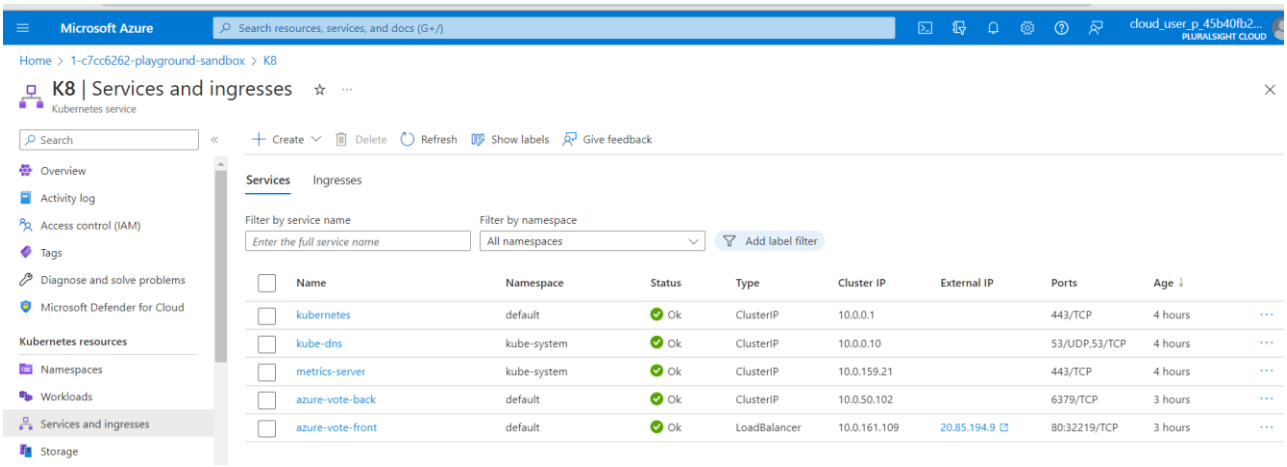


Note the banner in the terminal indicating that the pipeline needs additional permissions to continue the deployment. If the banner does not display. Click the back arrow above the deployment to go back to the job summary. Then, click View in the yellow notification box.

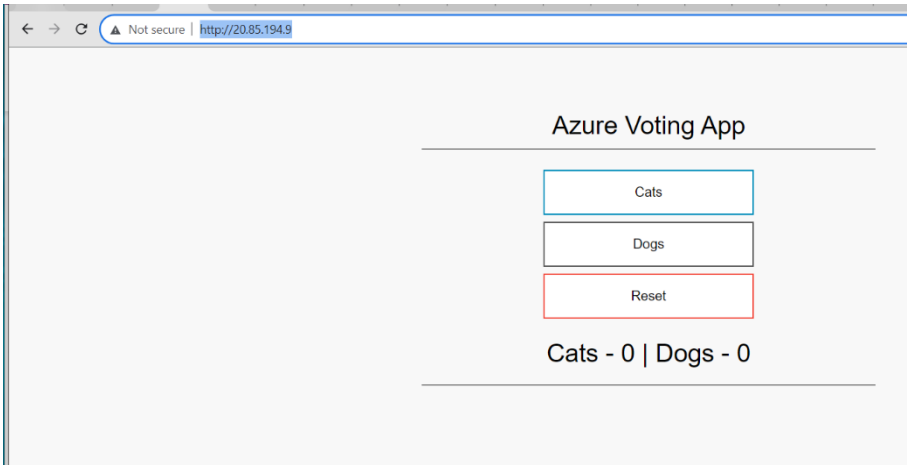
Like before, allow all permissions. The deployment will take a few minutes to complete.

Access the **AKS Cluster**
Navigate to the **ACR** tab.
At the top, click **All resources**.
Select the link for the **Kubernetes service** cluster. (You can ignore any errors that appear as they are related to restrictions).
On the left menu, under **Kubernetes resources**, select **Services and ingresses.**
All services in the cluster are displayed. Observe the azure-vote-front service has an **External IP**

Click the link for the IP.



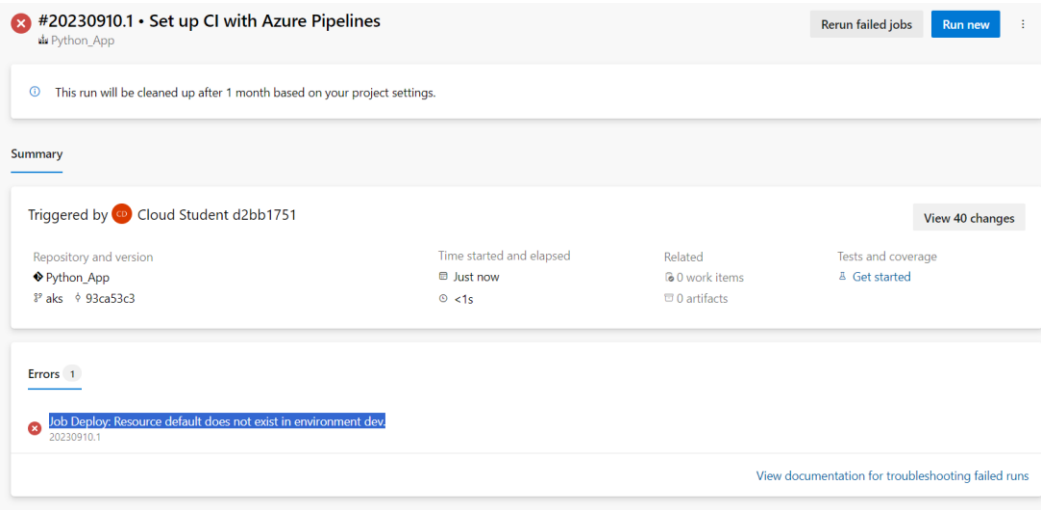
The app opens in a new tab where you can click to vote on cats or dogs.



Addendum

Errors to avoid

1) The pipeline gives this error!



Solution: Is your Docker installed and properly configured on your virtual machine? Please follow the instructions above in the pre steps!

2) The pipeline build gives this error!

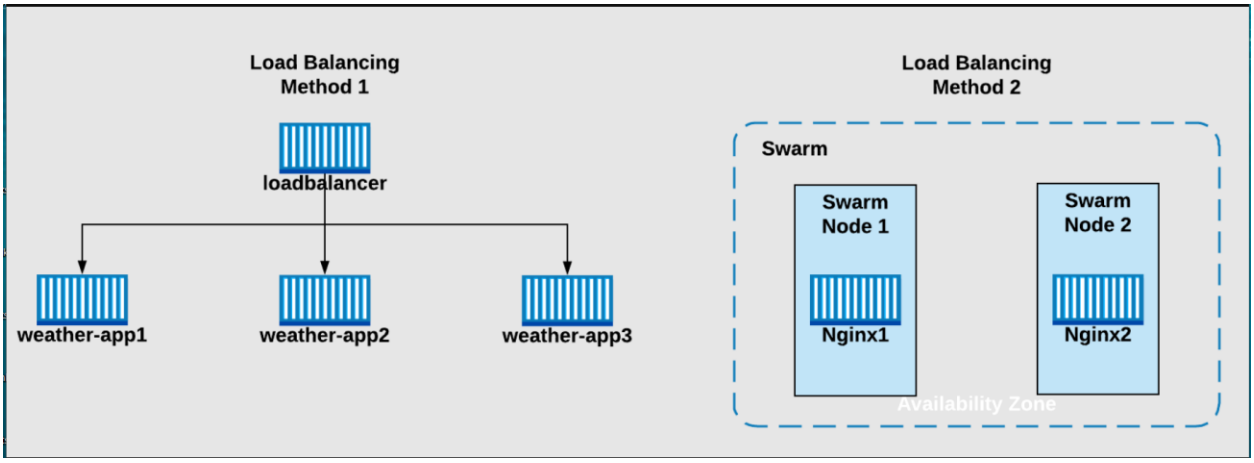
##[error]ERROR: permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get "http://%2Fvar%2Frun%2Fdocker.sock/_ping": dial unix /var/run/docker.sock: connect: permission denied
##[error]The process '/usr/bin/docker' failed with exit code 1

Solution: As indicated above, this is a permission issue related to your Docker on your virtual machine. Instructions for assigning permissions so that the DevOps pipeline can be

deployed to the virtual machine are above in pre steps.

Note: if the hello world test above fails with a similar message. Please do not proceed further with the Pre steps until this permission error is resolved. Otherwise, the pipeline Will fail!

Load Balancing Containers (Docker Swarm and Docker Compose)



Solution
This is as follows:

Prerequisites.

Before you begin, make sure you have:

Two CentOS 7 servers; one to be the master and the other to be the worker.

Master and Worker

[root@ip-10-0-1-4 ~]# **cat /etc/os-release**

NAME="CentOS Linux"
VERSION="7 (Core)"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="7"
PRETTY_NAME="CentOS Linux 7 (Core)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:7"
HOME_URL="https://www.centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"

CENTOS_MANTISBT_PROJECT="CentOS-7"
CENTOS_MANTISBT_PROJECT_VERSION="7"
REDHAT_SUPPORT_PRODUCT="centos"
REDHAT_SUPPORT_PRODUCT_VERSION="7"

You will need to have the following inbound port rules set.

Port **22** and **8080** need to be open.

SSH access to both servers with sudo privileges.

When setting up both server instances in the cloud, you will need to generate two .PEM key pairs to gain remote access to.

Installation and Configuration on the Master Node

Step 1: Update the System

SSH into the master server and update the system packages:

sudo yum update -y

Step 2: Install Docker

Install Docker on the master server:

sudo yum install -y docker

Step 3: Start and Enable Docker

Start and enable the Docker service:

sudo systemctl start docker

sudo systemctl enable docker

Step 4: Install Docker Compose

Install Docker Compose on the master node:

sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-

$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

Installation and Configuration on the Worker Node

Step 1: Update the System

SSH into the master server and update the system packages:

sudo yum update -y

Step 2: Install Docker

Install Docker on the master server:

sudo yum install -y docker

Step 3: Start and Enable Docker

Start and enable the Docker service:

sudo systemctl start docker

sudo systemctl enable docker

Step 4: Install Docker Compose

Install Docker Compose on the master node:

sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-

$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

Work on the Master node

ssh cloud_user@PUBLIC_IP_ADDRESS

Become the root user:

sudo su -

Create a Docker Compose file

Look at the directories available to you.

ls

```
[root@ip-10-0-1-103 ~]# ls
anaconda-ks.cfg  lb-challenge  swarm-token.txt
[root@ip-10-0-1-103 ~]# pwd
```

Change to the lb-challenge directory:

cd lb-challenge

Use the VIM editor to create the **docker-compose.yml** file.

vi docker-compose.yml

Use **Ctrl + :**

type **set paste**

Use the letter I for insert

```
version: '3.2'
services:
  weather-app1:
      build: ./weather-app
      tty: true
      networks:
       - frontend
  weather-app2:
      build: ./weather-app
      tty: true
      networks:
       - frontend
  weather-app3:
      build: ./weather-app
      tty: true
      networks:
       - frontend

  loadbalancer:
      build: ./load-balancer
      image: nginx
      tty: true
      ports:
       - '80:80'
      networks:
       - frontend

networks:
  frontend:
```

Now copy paste or type the above in the format as above.

When you have completed the above do the following to save and exit the file:

Ctrl + :
wq <enter>

Update nginx.conf
Change to the load-balancer directory:

cd load-balancer

Edit the nginx.conf file:

vi nginx.conf

The contents of your **nginx.conf** file should look like the following:

```
events { worker_connections 1024; }

http {
  upstream localhost {
    server weather-app1:3000;
    server weather-app2:3000;
    server weather-app3:3000;
  }
  server {
    listen 80;
    server_name localhost;
    location / {
      proxy_pass http://localhost;
      proxy_set_header Host $host;
    }
  }
}
```

Note: Please ensure that the format is as above. Otherwise the **nginx** file will not be processed!
Once done, save and exit with **wq** <enter>

Execute docker-compose up
Change directories:

cd ../

Execute the docker-compose up command:

docker-compose up --build -d

```
Compose does not use swarm mode to deploy services to multiple nodes in a swarm. All containers will be scheduled on the cur

To deploy your application across the swarm, use `docker stack deploy`.

Creating network "lb-challenge_frontend" with the default driver
Building weather-app1
Step 1/10 : FROM node
latest: Pulling from library/node
167b8a53ca45: Download complete
b47a222d28fa: Download complete
debce5f9f3a9: Download complete
1d7ca7cd2e06: Downloading [======>                                            ]  29.14MB/211MB
94c7791033e8: Download complete
72ab0dfaf5cb: Downloading [========>                                          ]  7.796MB/47.86MB
3316ed2852d4: Download complete
ef5505406bea: Waiting
```

```
Step 10/10 : CMD ./bin/www
 ---> Using cache
 ---> 746f221ee7ca
Successfully built 746f221ee7ca
Successfully tagged lb-challenge_weather-app3:latest
Building loadbalancer
Step 1/4 : FROM nginx
latest: Pulling from library/nginx
a803e7c4b030: Pull complete
8b625c47d697: Pull complete
4d3239651a63: Pull complete
0f816efa513d: Pull complete
01d159b8db2f: Pull complete
5fb9a81470f3: Pull complete
9b1e1e7164db: Pull complete
Digest: sha256:32da30332506740a2f7c34d5dc70467b7f14ec67d912703568daff790ab3f
Status: Downloaded newer image for nginx:latest
 ---> 61395b4c586d
Step 2/4 : COPY nginx.conf /etc/nginx/nginx.conf
 ---> e9a8196303dc
Step 3/4 : EXPOSE 80
 ---> Running in 065c58de4883
Removing intermediate container 065c58de4883
 ---> 4ceaad870e19
Step 4/4 : CMD ["nginx", "-g", "daemon off;"]
 ---> Running in ad2713a8c142
Removing intermediate container ad2713a8c142
 ---> b72755fd3d7c
Successfully built b72755fd3d7c
Successfully tagged nginx:latest
Creating lb-challenge_weather-app3_1 ... done
Creating lb-challenge_weather-app1_1 ... done
Creating lb-challenge_loadbalancer_1 ... done
Creating lb-challenge_weather-app2_1 ... done
[root@ip-10-0-1-4 lb-challenge]#
```

Now let's check our work:

docker ps

Copy the public IP address of the Swarm master and paste it into a new tab in your browser.

Create a Docker service using Docker Swarm

Change to the root directory:

```
cd ~/
```

View the contents of swarm-token.txt:

cat swarm-token.txt

```
[root@ip-10-0-1-4 lb-challenge]# cd ~/
[root@ip-10-0-1-4 ~]# cat swarm-token.txt
Swarm initialized: current node (yt05f9gtvto20vp2s209mjy5i) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-17j7h3wn62fkvit6zevjebaarb2o5imtrmjux7dwbonvnid4h1-3if43dsktfc7re2c8mezyl0i1 10.0.1.4:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

[root@ip-10-0-1-4 ~]#
```

Copy the docker swarm join command from the previous step.

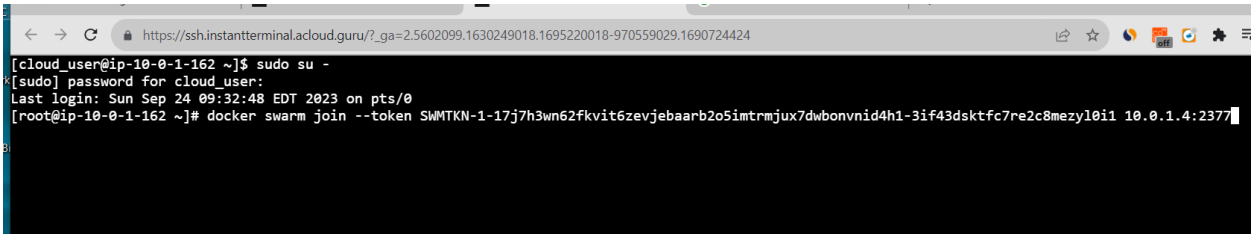
Note: You can also initiate this process external to the script by running the following on the Master:

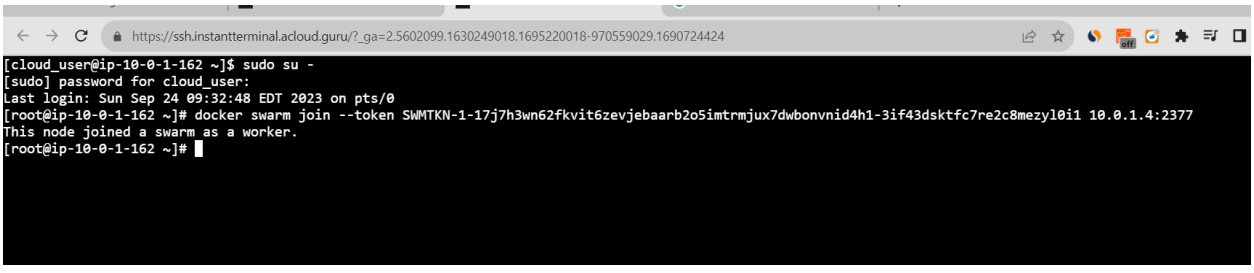
sudo docker swarm init

On Swarm worker node

Execute the command that was copied from the previous step.

i.e., the string docker **swarm join –token <token> <IP address>:<port>**
docker swarm join --token SWMTKN-1-17j7h3wn62fkvit6zevjebaarb2o5imtrmjux7dwbonvnid4h1-3if43dsktfc7re2c8mezyl0i1 10.0.1.4:2377

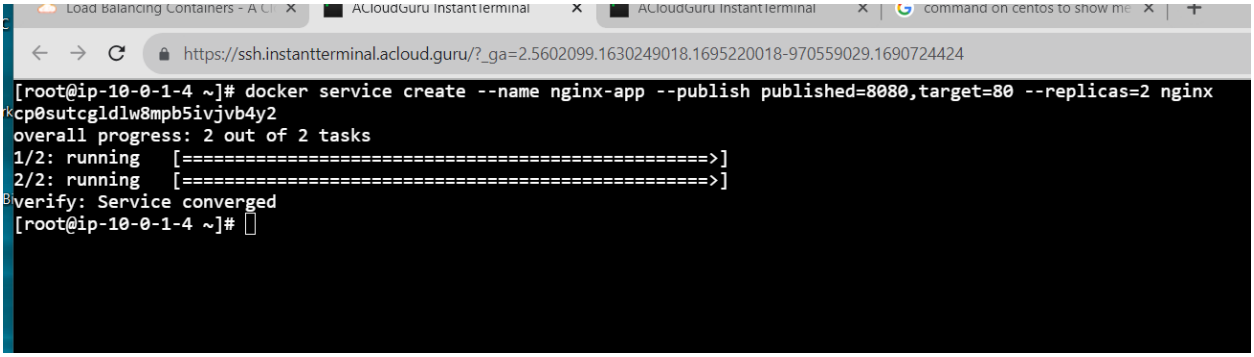




On Swarm master

Create a Docker service by executing the following command:

docker service create --name nginx-app --publish published=8080,target=80 --replicas=2 nginx



Note: the port 8080 used here. Please ensure that its open.

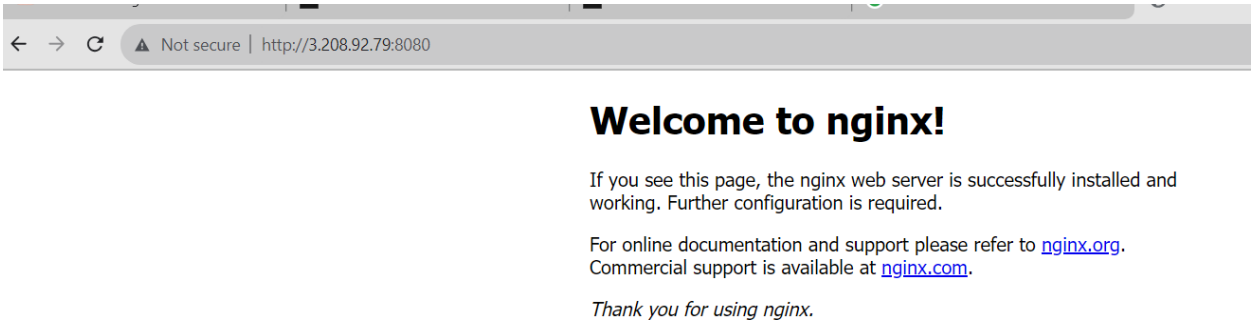
Verify this completed successfully by running the following command on both servers:

docker ps

Note: If you wish to see all containers (whether running or shutdown) use the **docker ps -a** command.

Verify that the default nginx page loads in the browser:

PUBLIC_IP_ADDRESS:8080



Addendum
How to download files used for this project between your local machine and Centos.

```
[root@ip-10-0-1-103 ~]# ls
anaconda-ks.cfg  lb-challenge  swarm-token.txt
[root@ip-10-0-1-103 ~]# pwd
```

The files, due to the permissions, can only be worked on by someone with root account access. Hence you can only see the above with the root account.

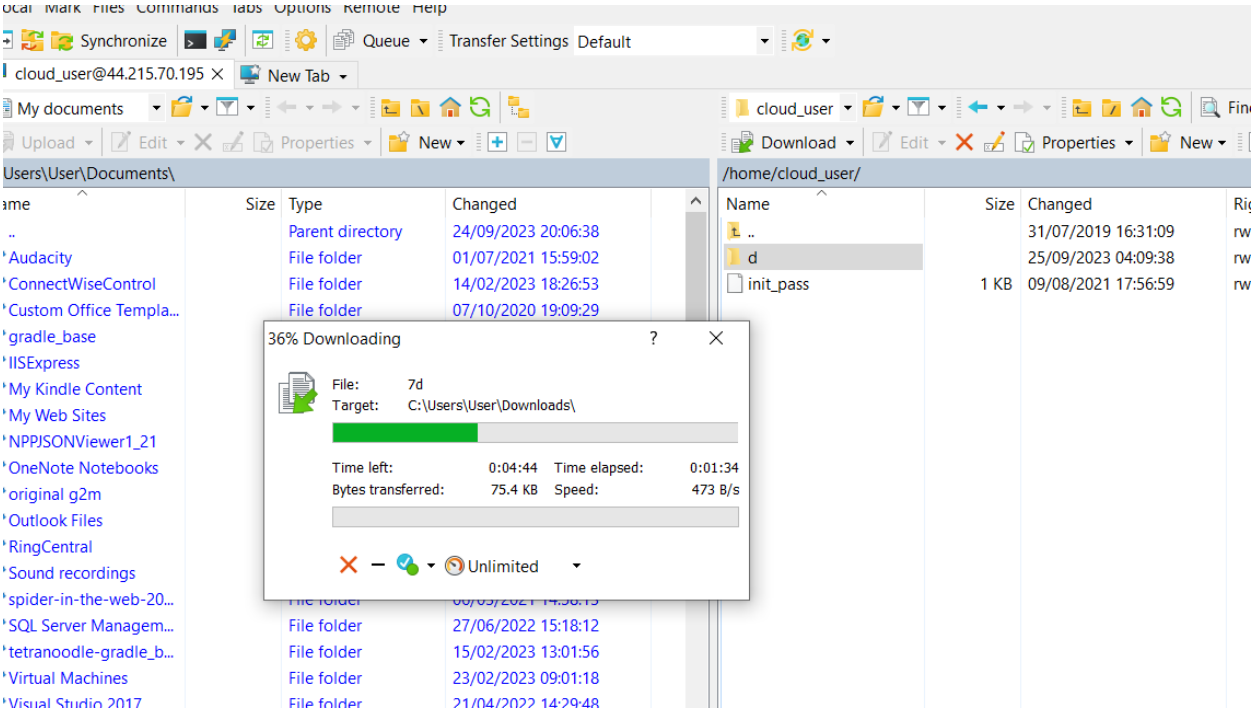
You may, however, wish to download the files and folders to work on them from your own local machine.

To make matter more complicated, when using WinSCP, you are unable to login as a root user. Hence, you will not be able to download the files!

Rather than playing with both the permissions and commands like **sudo visudo** that can weaken security you can instead copy the files from the root profile to a non-root profile as follows:

```
[root@ip-10-0-1-103 ~]# cp -r /root/lb-challenge/ /home/cloud_user/d
[root@ip-10-0-1-103 ~]# cp -r /root/anaconda-ks.cfg /home/cloud_user/d
[root@ip-10-0-1-103 ~]# cp -r /root/swarm-token.txt /home/cloud_user/d
[root@ip-10-0-1-103 ~]#
```

Note: pwd is helpful in constructing the folder paths.
cp == copy
-r == recursive command. Good when you have subfolders.
/path/source/ /path/destination

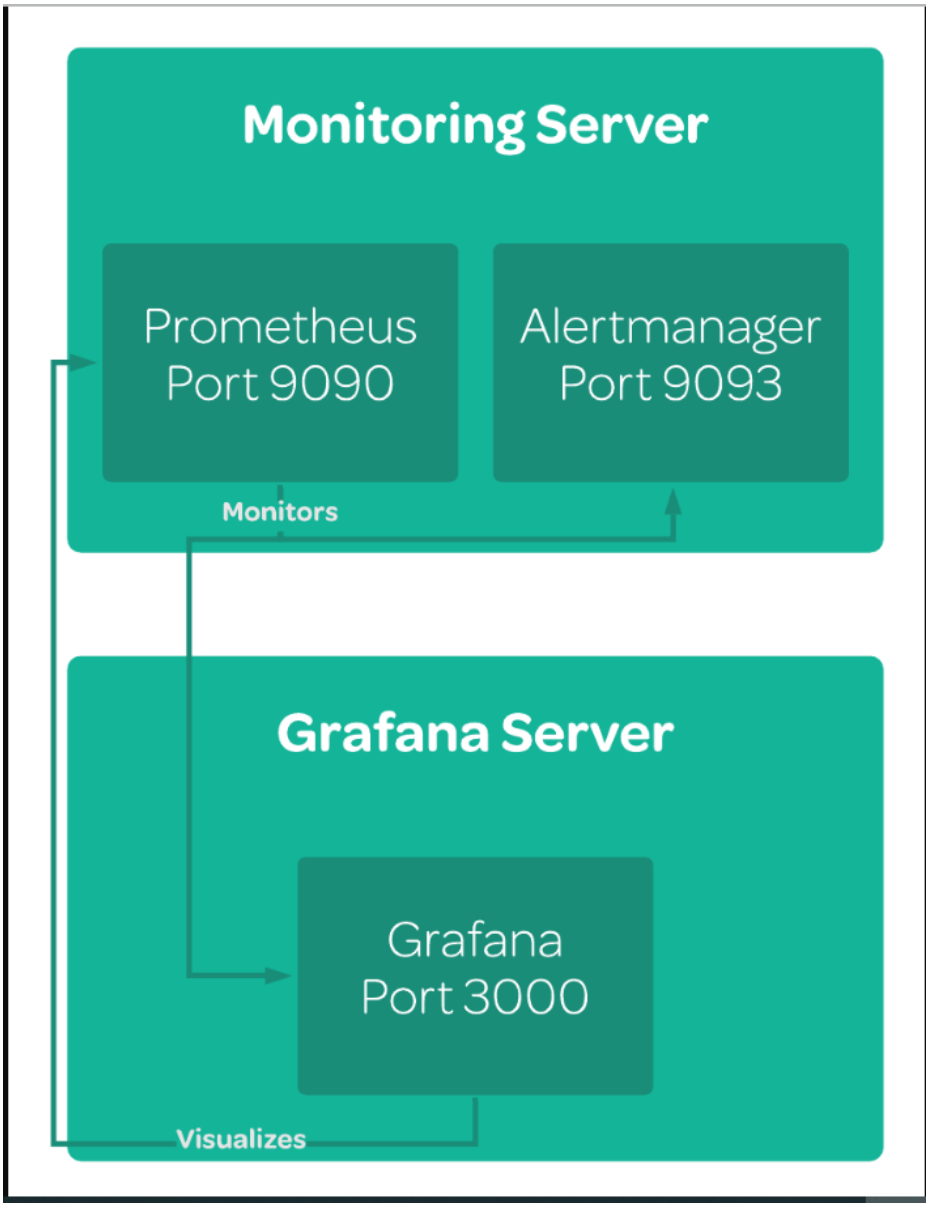


If you have any ‘stubborn’ files that wont download. No problem!

```
[root@ip-10-0-1-103 ~]# cat anaconda-ks.cfg
```

Use the cat command! Copy and paste the text locally.

Setting Up Prometheus and Adding Endpoints



Solution
Prerequisites:

You need two Linux servers.

One for installing Prometheus and the Alert Manager
The other for Grafana.

Call one **Monitoring** and the other **Grafana**.

As the following commands will, likely, run on any distro I am agnostic on which one you may choose.

For this project I will be using Centos 7

Logging In
NOTE: It's probably a good idea to have two or three terminals open, one for each node.

Install Docker and Kubernetes on All Servers
Most of these commands need to be run on each of the nodes. Pay attention though. Down at Step 10, we are going to do a little bit on just the master, and down at Step 15 we'll run something on just the nodes. There are notes down there, just be watching for them.

1 - Once we have logged in, we need to elevate privileges using sudo:
sudo su

2 - Disable SELinux:
setenforce 0
sed -i --follow-symlinks 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/sysconfig/selinux

3 - Enable the br_netfilter module for cluster communication:
modprobe br_netfilter
echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables

4 - Ensure that the Docker dependencies are satisfied:
yum install -y yum-utils device-mapper-persistent-data lvm2

5 - Add the Docker repo and install Docker:
yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
yum install -y docker-ce

6 - Set the cgroup driver for Docker to systemd, reload systemd, then enable and start Docker:
sed -i '/^ExecStart/ s/$/ --exec-opt native.cgroupdriver=systemd/'
/usr/lib/systemd/system/docker.service
systemctl daemon-reload
systemctl enable docker --now

7 - Add the Kubernetes repo:
cat << EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=0
repo_gpgcheck=0
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

EOF

8 - Install Kubernetes v1.14.0:
yum install -y kubelet-1.14.0-0 kubeadm-1.14.0-0 kubectl-1.14.0-0 kubernetes-cni-0.7.5

9 - Enable the kubelet service. The kubelet service will fail to start until the cluster is initialized, this is expected:
systemctl enable kubelet

Note: Complete the following section on the **MASTER ONLY!**

10 - Initialize the cluster using the IP range for Flannel:
kubeadm init --pod-network-cidr=10.244.0.0/16

11 - Copy the kubeadmn join command that is in the output. We will need this later.
NOTE: see **Addendum** for an example of the whole output from using the **kubeadm init** command.
12 - Exit sudo, copy the admin.conf to your home directory, and take ownership.
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

13 - Deploy Flannel:
kubectl apply -f https://raw.githubusercontent.com/flannel-io/flannel/master/Documentation/kube-flannel-old.yaml

14 - Check the cluster state:
kubectl get pods --all-namespaces

Note: Complete the following steps on the **NODES ONLY!**

15 - Run the join command that you copied earlier, this requires running the command prefaced with sudo on the nodes (if we hadn't run sudo su to begin with).

Then we'll check the nodes from the **master**.

kubectl get nodes

Create and Scale a Deployment Using kubectl

Note: These commands will only be run on the **master node.**

16 - Create a simple deployment:
kubectl create deployment nginx --image=nginx

17 - Inspect the pod:
kubectl get pods

18 - Scale the deployment:

kubectl scale deployment nginx --replicas=4

19 - Inspect the pods. We should have four now:
kubectl get pods

Configure permissions for Docker on both Master and node/s

sudo chmod 666 /var/run/docker.sock

docker run hello-world

Also install the following on both Master and node/s

Install the Git package:

sudo yum install git -y

Install Python3 on both Master and node/s:

NOTE: It is important to ensure that both **PIP** and **botocore boto3** libraries are installed on this system. Please follow these steps:
sudo yum update
sudo yum install python3
sudo yum install python3-pip
pip install botocore boto3
python3 –version

Install Java 11 on both Master and node/s
sudo apt install default-jdk -y

Please, ensure that the above ports are open externally and between each other.

Log in to the monitoring server

ssh cloud_user@<MONITORING_SERVER_PUBLIC_IP_ADDRESS>

Note: When copying and pasting code into Vim as you will later, first enter

:set paste (and then i to enter insert mode) to avoid adding unnecessary spaces and hashes.

Set Up Prometheus
On the monitoring server, create a user for Prometheus:

sudo useradd --no-create-home --shell /bin/false prometheus

```
cloud_user@mon:~$ sudo useradd --no-create-home --shell /bin/false prometheus
[sudo] password for cloud_user:
```

Create the needed directories:

sudo mkdir /etc/prometheus
sudo mkdir /var/lib/Prometheus

```
cloud_user@mon:~$ sudo mkdir /etc/prometheus
cloud_user@mon:~$ sudo mkdir /var/lib/prometheus
```

Set the ownership of the /var/lib/prometheus directory:
sudo chown prometheus:prometheus /var/lib/Prometheus

```
cloud_user@mon:~$ sudo chown prometheus:prometheus /var/lib/prometheus
```

Download Prometheus:

cd /tmp/
wget https://github.com/prometheus/prometheus/releases/download/v2.7.1/prometheus-

2.7.1.linux-amd64.tar.gz

```
cloud_user@mon:/tmp$ wget https://github.com/prometheus/prometheus/releases/download/v2.7.1/prometheus-2.7.1.linux-amd64.tar.gz
--2023-09-26 07:58:37--  https://github.com/prometheus/prometheus/releases/download/v2.7.1/prometheus-2.7.1.linux-amd64.tar.gz
Resolving github.com (github.com)... 140.82.113.3
Connecting to github.com (github.com)|140.82.113.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/6838921/926b3400-2563-11e9-8088-63bcd1a7b246?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20230926%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20230926T075838Z&X-Amz-Expires=300&X-Amz-Signature=1d39eda3807ab9947a7b0249eea36747ce68187a46c14988f18337b500d87edb&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=6838921&response-content-disposition=attachment%3B%20filename%3Dprometheus-2.7.1.linux-amd64.tar.gz&response-content-type=application%2Foctet-stream [following]
--2023-09-26 07:58:37--  https://objects.githubusercontent.com/github-production-release-asset-2e65be/6838921/926b3400-2563-11e9-8088-63bcd1a7b246?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20230926%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20230926T075838Z&X-Amz-Expires=300&X-Amz-Signature=1d39eda3807ab9947a7b0249eea36747ce68187a46c14988f18337b500d87edb&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=6838921&response-content-disposition=attachment%3B%20filename%3Dprometheus-2.7.1.linux-amd64.tar.gz&response-content-type=application%2Foctet-stream
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.109.133, 185.199.110.133, 185.199.108.133, ...
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 38342763 (37M) [application/octet-stream]
Saving to: 'prometheus-2.7.1.linux-amd64.tar.gz'

prometheus-2.7.1.linux-amd64.tar.gz      100%[=====================================================================================>]  36.57M   181MB/s    in 0.2s

2023-09-26 07:58:37 (181 MB/s) - 'prometheus-2.7.1.linux-amd64.tar.gz' saved [38342763/38342763]

cloud_user@mon:/tmp$
```

Extract the files:

tar -xvf prometheus-2.7.1.linux-amd64.tar.gz

```
cloud_user@mon:~$ ls
cloud_user@mon:~$ cd /tmp/
cloud_user@mon:/tmp$ ls
prometheus-2.7.1.linux-amd64.tar.gz  systemd-private-0e2b4d0243044376a327d441faa9ad51-systemd-resolved.service-Crrxcz
snap-private-tmp                     systemd-private-0e2b4d0243044376a327d441faa9ad51-systemd-timesyncd.service-kQt5eu
cloud_user@mon:/tmp$ tar -xvf prometheus-2.7.1.linux-amd64.tar.gz
```

```
cloud_user@mon:~$ ls
cloud_user@mon:~$ cd /tmp/
cloud_user@mon:/tmp$ ls
prometheus-2.7.1.linux-amd64.tar.gz  systemd-private-0e2b4d0243044376a327d441faa9ad51-systemd-resolved.service-Crrxcz
snap-private-tmp                     systemd-private-0e2b4d0243044376a327d441faa9ad51-systemd-timesyncd.service-kQt5eu
cloud_user@mon:/tmp$ tar -xvf prometheus-2.7.1.linux-amd64.tar.gz
prometheus-2.7.1.linux-amd64/
prometheus-2.7.1.linux-amd64/LICENSE
prometheus-2.7.1.linux-amd64/prometheus.yml
prometheus-2.7.1.linux-amd64/consoles/
prometheus-2.7.1.linux-amd64/consoles/node-cpu.html
prometheus-2.7.1.linux-amd64/consoles/node.html
prometheus-2.7.1.linux-amd64/consoles/prometheus.html
prometheus-2.7.1.linux-amd64/consoles/prometheus-overview.html
prometheus-2.7.1.linux-amd64/consoles/node-overview.html
prometheus-2.7.1.linux-amd64/consoles/node-disk.html
prometheus-2.7.1.linux-amd64/consoles/index.html.example
prometheus-2.7.1.linux-amd64/console_libraries/
prometheus-2.7.1.linux-amd64/console_libraries/prom.lib
prometheus-2.7.1.linux-amd64/console_libraries/menu.lib
prometheus-2.7.1.linux-amd64/NOTICE
prometheus-2.7.1.linux-amd64/promtool
prometheus-2.7.1.linux-amd64/prometheus
cloud_user@mon:/tmp$
```

Move the configuration file and set the owner to the prometheus user:

cd prometheus-2.7.1.linux-amd64/
sudo mv console* /etc/prometheus
sudo mv prometheus.yml /etc/prometheus
sudo chown -R prometheus:prometheus /etc/prometheus

```
cloud_user@mon:/tmp$ cd prometheus-2.7.1.linux-amd64/
cloud_user@mon:/tmp/prometheus-2.7.1.linux-amd64$ sudo mv console* /etc/prometheus
[sudo] password for cloud_user:
cloud_user@mon:/tmp/prometheus-2.7.1.linux-amd64$ sudo mv prometheus.yml /etc/prometheus
cloud_user@mon:/tmp/prometheus-2.7.1.linux-amd64$ sudo chown -R prometheus:prometheus /etc/prometheus
cloud_user@mon:/tmp/prometheus-2.7.1.linux-amd64$
```

Move the binaries and set the owner:

sudo mv prometheus /usr/local/bin/
sudo mv promtool /usr/local/bin/

sudo chown prometheus:prometheus /usr/local/bin/prometheus
sudo chown prometheus:prometheus /usr/local/bin/promtool

Change back to your main directory, and then create the service file:

```
cloud_user@mon:/tmp/prometheus-2.7.1.linux-amd64$ sudo mv prometheus /usr/local/bin/
cloud_user@mon:/tmp/prometheus-2.7.1.linux-amd64$ sudo mv promtool /usr/local/bin/
cloud_user@mon:/tmp/prometheus-2.7.1.linux-amd64$ sudo chown prometheus:prometheus /usr/local/bin/prometheus
cloud_user@mon:/tmp/prometheus-2.7.1.linux-amd64$ sudo chown prometheus:prometheus /usr/local/bin/promtool
cloud_user@mon:/tmp/prometheus-2.7.1.linux-amd64$
```

cd
sudo vim /etc/systemd/system/prometheus.service

```
cloud_user@mon:/tmp/prometheus-2.7.1.linux-amd64$ cd
cloud_user@mon:~$ sudo vim /etc/systemd/system/prometheus.service
cloud_user@mon:~$
```

Add the following to the file:

[Unit]
Description=Prometheus
Wants=network-online.target
After=network-online.target

[Service]
User=prometheus
Group=prometheus
Type=simple
**ExecStart=/usr/local/bin/prometheus **
**--config.file /etc/prometheus/prometheus.yml **
**--storage.tsdb.path /var/lib/prometheus/ **
**--web.console.templates=/etc/prometheus/consoles **
--web.console.libraries=/etc/prometheus/console_libraries

[Install]

WantedBy=multi-user.target

Save and exit the file by pressing Escape followed by :**wq.**

```
[Unit]
Description=Prometheus
Wants=network-online.target
After=network-online.target

[Service]
User=prometheus
Group=prometheus
Type=simple
ExecStart=/usr/local/bin/prometheus \
    --config.file /etc/prometheus/prometheus.yml \
    --storage.tsdb.path /var/lib/prometheus/ \
    --web.console.templates=/etc/prometheus/consoles \
    --web.console.libraries=/etc/prometheus/console_libraries

[Install]
WantedBy=multi-user.target
~
~
```

Start Prometheus and make sure it automatically starts on boot:

sudo systemctl start prometheus
sudo systemctl enable Prometheus

```
cloud_user@mon:~$ sudo systemctl start prometheus
cloud_user@mon:~$ sudo systemctl enable prometheus
Created symlink /etc/systemd/system/multi-user.target.wants/prometheus.service → /etc/systemd/system/prometheus.service.
cloud_user@mon:~$
```

Check Prometheus's status to ensure everything is working correctly:

sudo systemctl status prometheus

```
cloud_user@mon:~$ sudo systemctl status prometheus
● prometheus.service - Prometheus
   Loaded: loaded (/etc/systemd/system/prometheus.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2023-09-26 08:16:54 UTC; 1min 5s ago
 Main PID: 31758 (prometheus)
    Tasks: 9 (limit: 1104)
   CGroup: /system.slice/prometheus.service
           └─31758 /usr/local/bin/prometheus --config.file /etc/prometheus/prometheus.yml --storage.tsdb.path /var/lib/prometheus/ --web.console.templates=/etc/pr

Sep 26 08:16:55 mon prometheus[31758]: level=info ts=2023-09-26T08:16:55.052864138Z caller=main.go:303 build_context="(go=go1.11.5, user=root@f9f82868fc43, date=2
Sep 26 08:16:55 mon prometheus[31758]: level=info ts=2023-09-26T08:16:55.052929744Z caller=main.go:304 host_details="(Linux 5.4.0-1092-aws #100~18.04.2-Ubuntu SMP
Sep 26 08:16:55 mon prometheus[31758]: level=info ts=2023-09-26T08:16:55.052969338Z caller=main.go:305 fd_limits="(soft=1024, hard=4096)"
Sep 26 08:16:55 mon prometheus[31758]: level=info ts=2023-09-26T08:16:55.053000373Z caller=main.go:306 vm_limits="(soft=unlimited, hard=unlimited)"
Sep 26 08:16:55 mon prometheus[31758]: level=info ts=2023-09-26T08:16:55.054567195Z caller=web.go:416 component=web msg="Start listening for connections" address=
Sep 26 08:16:55 mon prometheus[31758]: level=info ts=2023-09-26T08:16:55.054517709Z caller=main.go:620 msg="Starting TSDB ..."
Sep 26 08:16:55 mon prometheus[31758]: level=info ts=2023-09-26T08:16:55.080045567Z caller=main.go:635 msg="TSDB started"
Sep 26 08:16:55 mon prometheus[31758]: level=info ts=2023-09-26T08:16:55.080134169Z caller=main.go:695 msg="Loading configuration file" filename=/etc/prometheus/p
Sep 26 08:16:55 mon prometheus[31758]: level=info ts=2023-09-26T08:16:55.081213092Z caller=main.go:722 msg="Completed loading of configuration file" filename=/etc
Sep 26 08:16:55 mon prometheus[31758]: level=info ts=2023-09-26T08:16:55.081242996Z caller=main.go:589 msg="Server is ready to receive web requests."
lines 1-18/18 (END)
```

Set Up Alertmanager:

Still in the monitoring server, create the alertmanager system user:

sudo useradd --no-create-home --shell /bin/false alertmanager

Create the needed directories:

sudo mkdir /etc/alertmanager

```
cloud_user@mon:~$ sudo useradd --no-create-home --shell /bin/false alertmanager
cloud_user@mon:~$ sudo mkdir /etc/alertmanager
cloud_user@mon:~$
```

Download Alertmanager:

cd /tmp/
wget "https://github.com/prometheus/alertmanager/releases/download/v0.16.1/alertmanager-0.16.1.linux-amd64.tar.gz"

```
cloud_user@mon:~$ cd /tmp/
cloud_user@mon:/tmp$ wget "https://github.com/prometheus/alertmanager/releases/download/v0.16.1/alertmanager-0.16.1.linux-amd64.tar.gz"
--2023-09-26 08:23:36--  https://github.com/prometheus/alertmanager/releases/download/v0.16.1/alertmanager-0.16.1.linux-amd64.tar.gz
Resolving github.com (github.com)... 140.82.113.4
Connecting to github.com (github.com)|140.82.113.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/11452538/5abdb580-257e-11e9-8be2-57c655a5e02d?X-Amz-Algorithm=AWS4-HMAC-SHA
256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20230926%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20230926T082337Z&X-Amz-Expires=300&X-Amz-Signature=db8f876fe7cc7a1
0f5e6fd09cfe2b998b56900cd121658712eac2a542ece47c1&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=11452538&response-content-disposition=attachment%3B%20filen
ame%3Dalertmanager-0.16.1.linux-amd64.tar.gz&response-content-type=application%2Foctet-stream [following]
--2023-09-26 08:23:36--  https://objects.githubusercontent.com/github-production-release-asset-2e65be/11452538/5abdb580-257e-11e9-8be2-57c655a5e02d?X-Amz-Algorith
m=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20230926%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20230926T082337Z&X-Amz-Expires=300&X-Amz-Signature=
db8f876fe7cc7a10f5e6fd09cfe2b998b56900cd121658712eac2a542ece47c1&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=11452538&response-content-disposition=attach
ment%3B%20filename%3Dalertmanager-0.16.1.linux-amd64.tar.gz&response-content-type=application%2Foctet-stream
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.108.133, 185.199.111.133, 185.199.110.133, ...
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 19070056 (18M) [application/octet-stream]
Saving to: ‘alertmanager-0.16.1.linux-amd64.tar.gz’

alertmanager-0.16.1.linux-amd64.tar.gz   100%[=========================================================================>]  18.19M  --.-KB/s    in 0.1s

2023-09-26 08:23:36 (169 MB/s) - ‘alertmanager-0.16.1.linux-amd64.tar.gz’ saved [19070056/19070056]

cloud_user@mon:/tmp$
```

Extract the files:

tar -xvf alertmanager-0.16.1.linux-amd64.tar.gz

```
cloud_user@mon:/tmp$ tar -xvf alertmanager-0.16.1.linux-amd64.tar.gz
```

```
cloud_user@mon:/tmp$ tar -xvf alertmanager-0.16.1.linux-amd64.tar.gz
alertmanager-0.16.1.linux-amd64/
alertmanager-0.16.1.linux-amd64/NOTICE
alertmanager-0.16.1.linux-amd64/amtool
alertmanager-0.16.1.linux-amd64/LICENSE
alertmanager-0.16.1.linux-amd64/alertmanager
alertmanager-0.16.1.linux-amd64/alertmanager.yml
cloud_user@mon:/tmp$
```

cd alertmanager-0.16.1.linux-amd64/

```
cloud_user@mon:/tmp$ cd alertmanager-0.16.1.linux-amd64/
cloud_user@mon:/tmp/alertmanager-0.16.1.linux-amd64$
```

Move the binaries:

sudo mv alertmanager /usr/local/bin/
sudo mv amtool /usr/local/bin/

```
cloud_user@mon:/tmp/alertmanager-0.16.1.linux-amd64$ sudo mv alertmanager /usr/local/bin/
cloud_user@mon:/tmp/alertmanager-0.16.1.linux-amd64$ sudo mv amtool /usr/local/bin/
cloud_user@mon:/tmp/alertmanager-0.16.1.linux-amd64$
```

Set the ownership of the binaries:

sudo chown -R alertmanager:alertmanager /usr/local/bin/alertmanager

sudo chown -R alertmanager:alertmanager /usr/local/bin/amtool

```
cloud_user@mon:/tmp/alertmanager-0.16.1.linux-amd64$ sudo chown -R alertmanager:alertmanager /usr/local/bin/alertmanager
cloud_user@mon:/tmp/alertmanager-0.16.1.linux-amd64$ sudo chown -R alertmanager:alertmanager /usr/local/bin/amtool
cloud_user@mon:/tmp/alertmanager-0.16.1.linux-amd64$
```

Move the configuration file into the /etc/alertmanager directory:

sudo mv alertmanager.yml /etc/alertmanager/

```
cloud_user@mon:/tmp/alertmanager-0.16.1.linux-amd64$ sudo mv alertmanager.yml /etc/alertmanager/
cloud_user@mon:/tmp/alertmanager-0.16.1.linux-amd64$
```

Set the ownership of the /etc/alertmanager directory:

sudo chown -R alertmanager:alertmanager /etc/alertmanager/

```
cloud_user@mon:/tmp/alertmanager-0.16.1.linux-amd64$ sudo chown -R alertmanager:alertmanager /etc/alertmanager/
cloud_user@mon:/tmp/alertmanager-0.16.1.linux-amd64$
```

Create the alertmanager.service file for systemd:

sudo vim /etc/systemd/system/alertmanager.service

Add the following to the file:

```
[Unit]
Description=Alertmanager
Wants=network-online.target
After=network-online.target

[Service]
User=alertmanager
Group=alertmanager
Type=simple
WorkingDirectory=/etc/alertmanager/
ExecStart=/usr/local/bin/alertmanager \
    --config.file=/etc/alertmanager/alertmanager.yml

[Install]
WantedBy=multi-user.target
```

Save and exit the file by pressing Escape followed by :**wq**.

```
[Unit]
Description=Alertmanager
Wants=network-online.target
After=network-online.target

[Service]
User=alertmanager
Group=alertmanager
Type=simple
WorkingDirectory=/etc/alertmanager/
ExecStart=/usr/local/bin/alertmanager \
    --config.file=/etc/alertmanager/alertmanager.yml

[Install]
WantedBy=multi-user.target
~
~
~
```

Stop Prometheus, change back to your main directory, and then update the Prometheus configuration file to use Alertmanager:

sudo systemctl stop prometheus
cd
sudo vim /etc/prometheus/prometheus.yml

```
cloud_user@mon:/tmp/alertmanager-0.16.1.linux-amd64$ sudo systemctl stop prometheus
cloud_user@mon:/tmp/alertmanager-0.16.1.linux-amd64$ cd
cloud_user@mon:~$ sudo vim /etc/prometheus/prometheus.yml
cloud_user@mon:~$
```

Un-comment alertmanager, and change alertmanager to localhost (so it will look like the following):

```
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      - localhost:9093
```

```
# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      # - alertmanager:9093
```

```
# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      - alertmanager:9093
```

Save and exit the file by pressing Escape followed by **:wq**.

Reload systemd, and then start the prometheus and alertmanager services:

sudo systemctl daemon-reload
sudo systemctl start prometheus
sudo systemctl start alertmanager

```
cloud_user@mon:~$ sudo systemctl daemon-reload
cloud_user@mon:~$ sudo systemctl start prometheus
cloud_user@mon:~$ sudo systemctl start alertmanager
cloud_user@mon:~$
```

Make sure alertmanager starts on boot:

sudo systemctl enable alertmanager

```
cloud_user@mon:~$ sudo systemctl enable alertmanager
Created symlink /etc/systemd/system/multi-user.target.wants/alertmanager.service → /etc/systemd/system/alertmanager.service.
cloud_user@mon:~$
```

Set Up Grafana

In a new terminal window, log in to the grafana server via SSH using the credentials provided on the lab page:

ssh cloud_user@<GRAFANA_SERVER_PUBLIC_IP_ADDRESS>

Install the prerequisite package:

sudo apt-get install libfontconfig
Y

```
cloud_user@grafana:~$ sudo apt-get install libfontconfig
[sudo] password for cloud_user:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'libfontconfig1' instead of 'libfontconfig'
The following packages were automatically installed and are no longer required:
  linux-aws-5.4-headers-5.4.0-1060 linux-headers-5.4.0-1060-aws linux-image-5.4.0-1060-aws linux-modules-5.4.0-1060-aws
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  fontconfig-config fonts-dejavu-core
The following NEW packages will be installed:
  fontconfig-config fonts-dejavu-core libfontconfig1
0 upgraded, 3 newly installed, 0 to remove and 26 not upgraded.
Need to get 1233 kB of archives.
After this operation, 4015 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Download and install Grafana using the .deb package provided on the Grafana download page:

cd /tmp/
wget https://dl.grafana.com/oss/release/grafana_5.4.3_amd64.deb

```
cloud_user@grafana:~$ cd /tmp/
cloud_user@grafana:/tmp$ wget https://dl.grafana.com/oss/release/grafana_5.4.3_amd64.deb
--2023-09-26 08:45:29--  https://dl.grafana.com/oss/release/grafana_5.4.3_amd64.deb
Resolving dl.grafana.com (dl.grafana.com)... 146.75.38.217, 2a04:4e42:78::729
Connecting to dl.grafana.com (dl.grafana.com)|146.75.38.217|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 55405690 (53M) [application/x-debian-package]
Saving to: ‘grafana_5.4.3_amd64.deb’

grafana_5.4.3_amd64.deb               100%[=====================================================================================>]  52.84M   166MB/s    in 0.3s

2023-09-26 08:45:30 (166 MB/s) - ‘grafana_5.4.3_amd64.deb’ saved [55405690/55405690]

cloud_user@grafana:/tmp$ sudo dpkg -i grafana_5.4.3_amd64.deb
```

sudo dpkg -i grafana_5.4.3_amd64.deb

```
cloud_user@grafana:/tmp$ sudo dpkg -i grafana_5.4.3_amd64.deb
Selecting previously unselected package grafana.
(Reading database ... 149179 files and directories currently installed.)
Preparing to unpack grafana_5.4.3_amd64.deb ...
Unpacking grafana (5.4.3) ...
Setting up grafana (5.4.3) ...
Adding system user `grafana' (UID 111) ...
Adding new user `grafana' (UID 111) with group `grafana' ...
Not creating home directory `/usr/share/grafana'.
### NOT starting on installation, please execute the following statements to configure grafana to start automatically using systemd
 sudo /bin/systemctl daemon-reload
 sudo /bin/systemctl enable grafana-server
### You can start grafana-server by executing
 sudo /bin/systemctl start grafana-server
Processing triggers for systemd (237-3ubuntu10.57) ...
Processing triggers for ureadahead (0.100.0-21) ...
cloud_user@grafana:/tmp$
```

Start Grafana:

sudo systemctl start grafana-server

Ensure Grafana starts at boot:

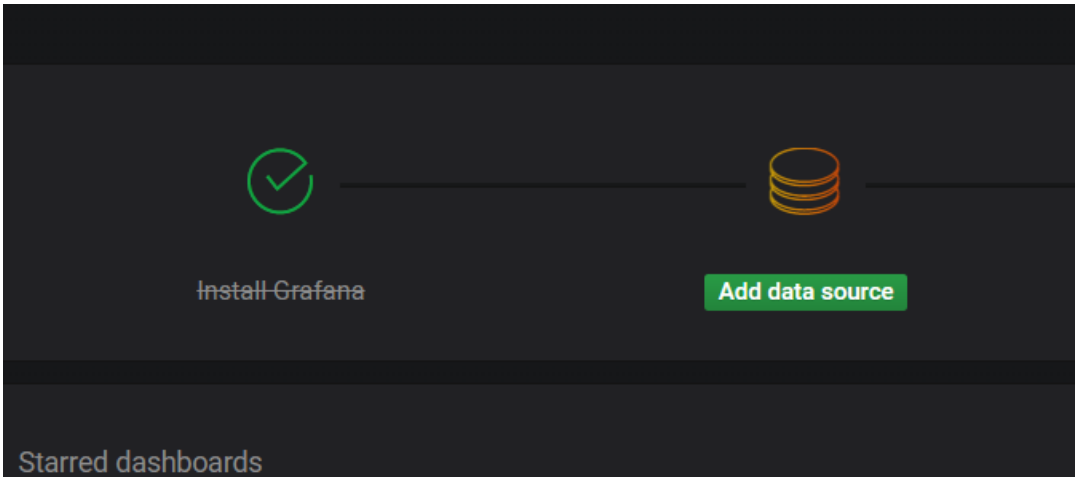
sudo systemctl enable grafana-server

```
cloud_user@grafana:/tmp$ sudo systemctl start grafana-server
cloud_user@grafana:/tmp$ sudo systemctl enable grafana-server
Synchronizing state of grafana-server.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable grafana-server
Created symlink /etc/systemd/system/multi-user.target.wants/grafana-server.service → /usr/lib/systemd/system/grafana-server.service.
cloud_user@grafana:/tmp$
```

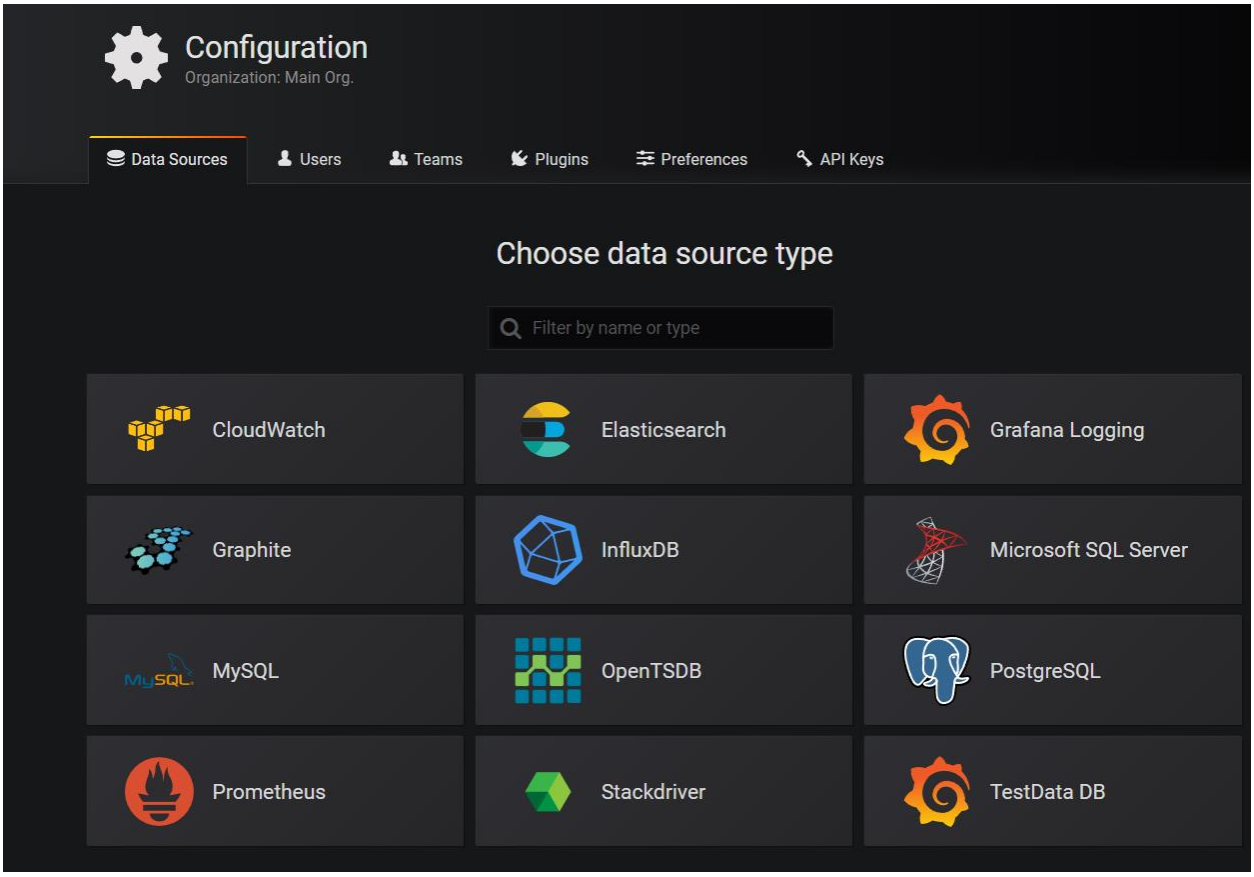
Using the Grafana IP address listed on the lab page, access Grafana's web UI by navigating to **<GRAFANA_IP_ADDRESS>:3000** in a new browser tab.

Log in with the username **admin** and the password **admin**. Reset the password when prompted.

Add a Data Source

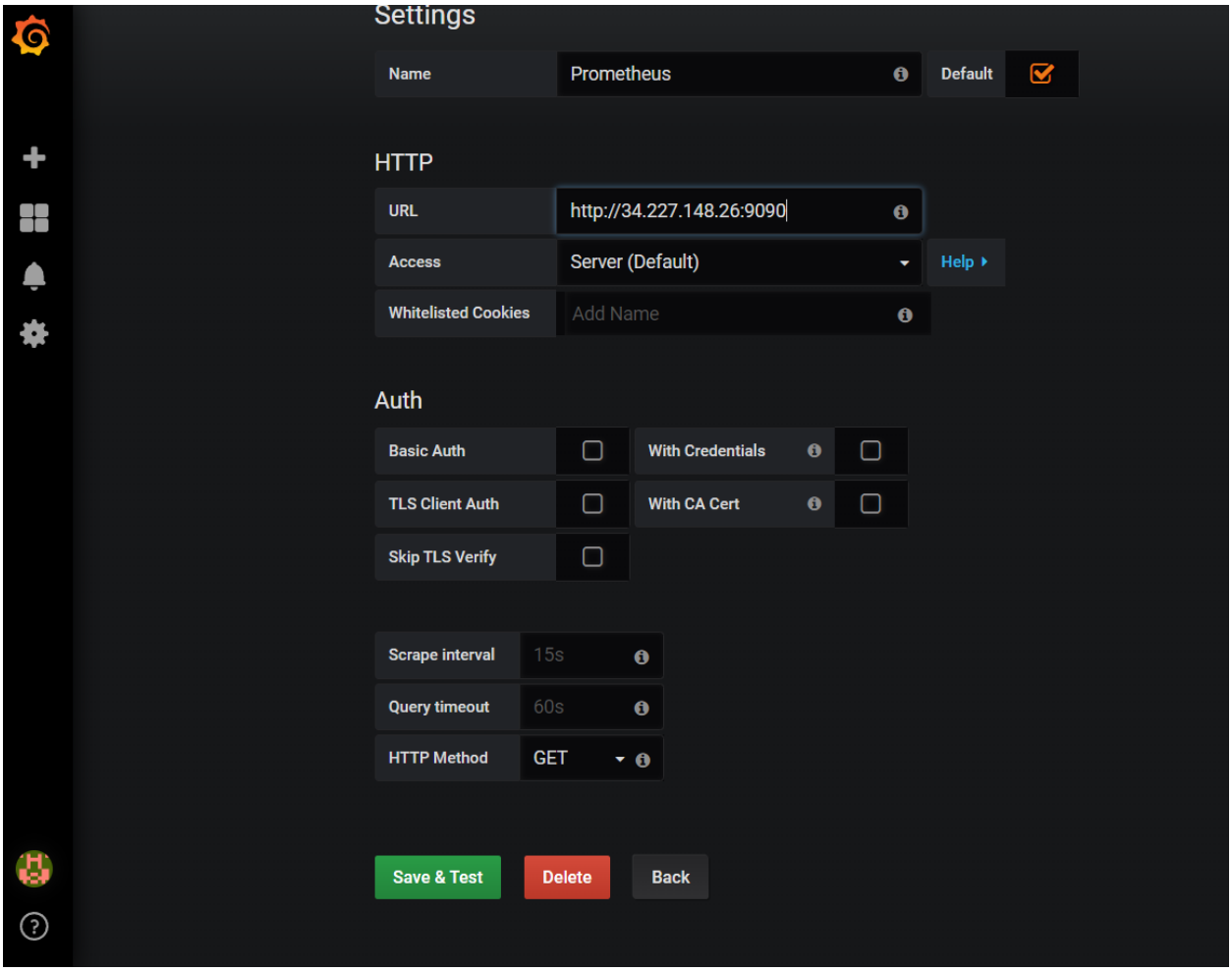


Click **Add data source** on the home page.

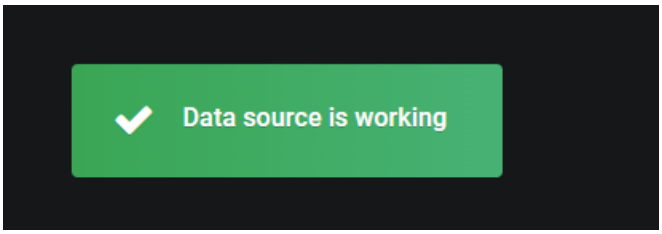


Select Prometheus.

In the HTTP section, set the URL to http://<MONITORING_IP_ADDRESS>:9090
(using the public IP of the monitoring server provided on the lab page).



Click **Save & Test**.



Add Endpoints
Back in the **monitoring server** terminal, open the Prometheus configuration file:

sudo vim /etc/prometheus/prometheus.yml

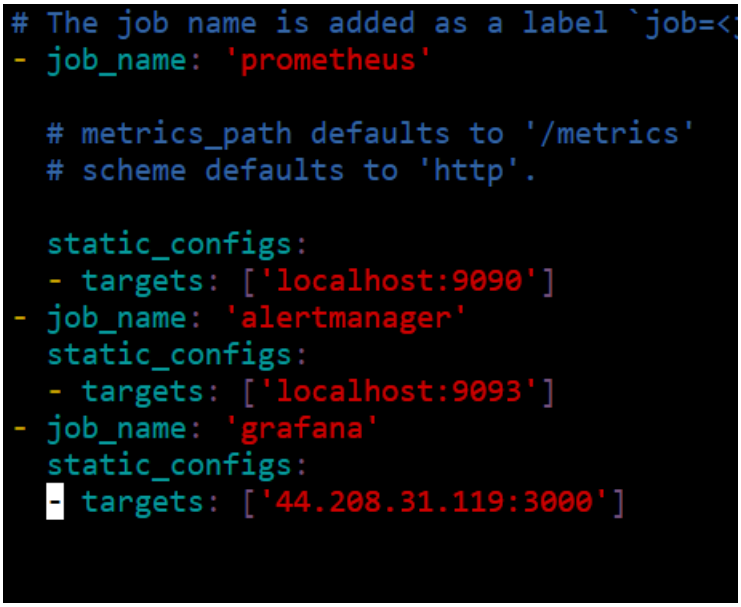
At the end of the file, at the bottom of the scrape_configs section, add the Alertmanager endpoint (make sure it aligns with the - job_name: 'prometheus' line):

- job_name: 'alertmanager'
static_configs:
- targets: ['localhost:9093']

Beneath what you just added, add the Grafana endpoint (using the public IP address of the grafana server):

- job_name: 'grafana'
static_configs:
- targets: ['<GRAFANA_IP_ADDRESS>:3000']

Save and exit the file by pressing Escape followed by :**wq**.



Restart Prometheus:

sudo systemctl restart Prometheus

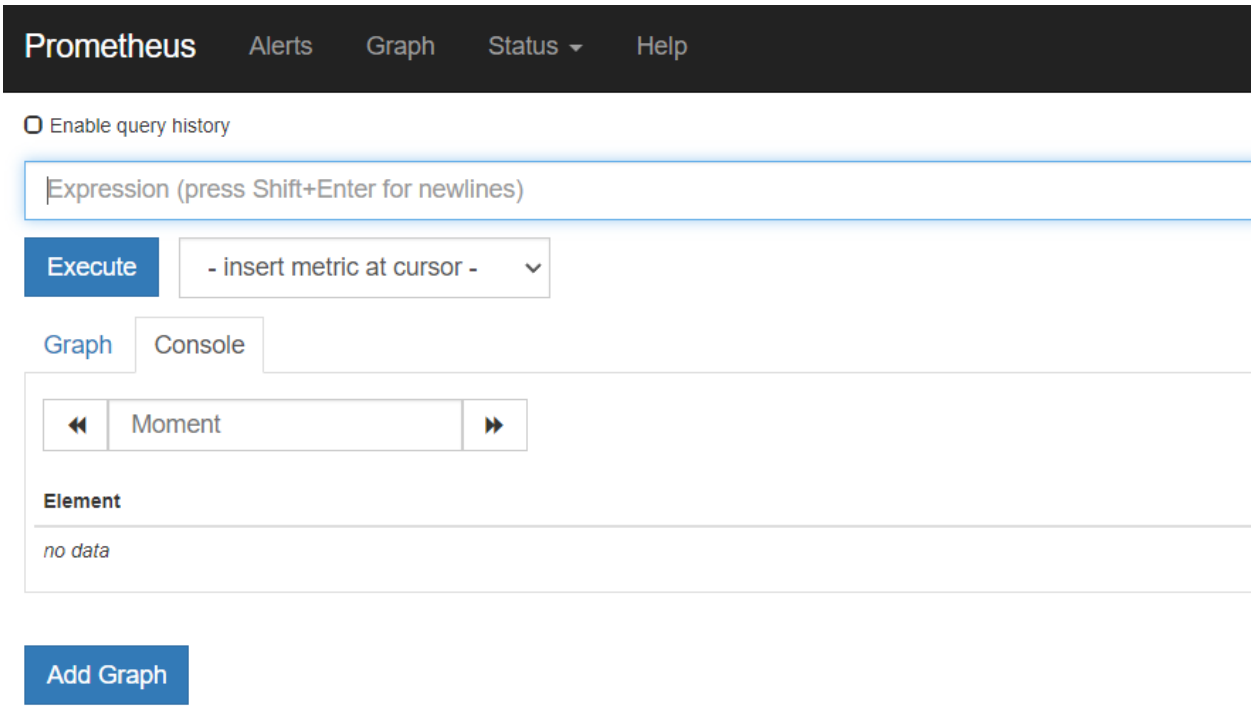
Check its status:

sudo systemctl status prometheus

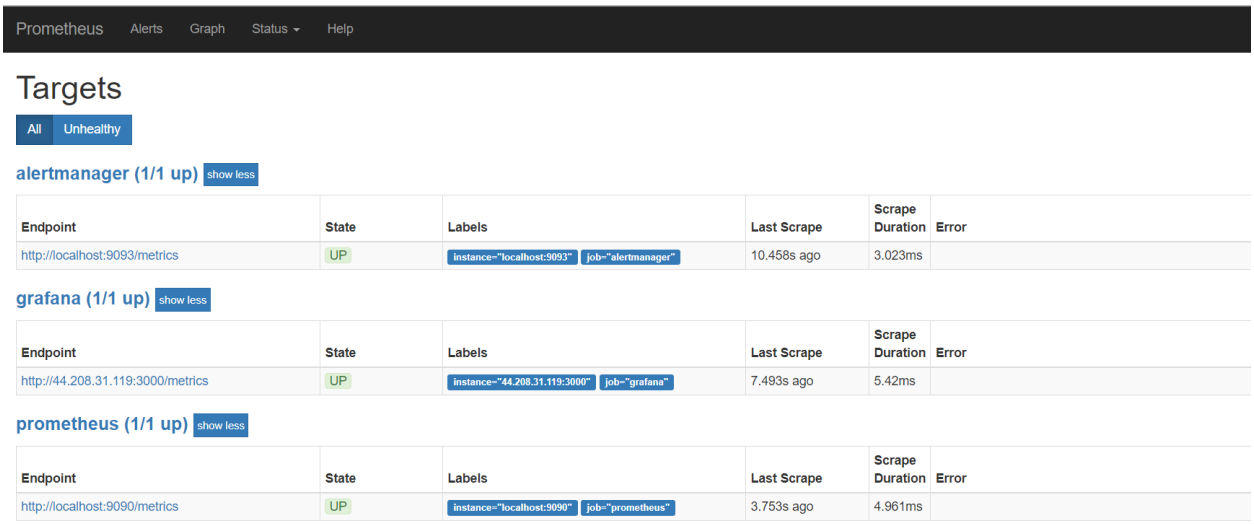
```
cloud_user@mon:~$ sudo systemctl restart prometheus
cloud_user@mon:~$ sudo systemctl status prometheus
● prometheus.service - Prometheus
   Loaded: loaded (/etc/systemd/system/prometheus.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2023-09-26 09:12:31 UTC; 22s ago
 Main PID: 32261 (prometheus)
    Tasks: 10 (limit: 1104)
   CGroup: /system.slice/prometheus.service
           └─32261 /usr/local/bin/prometheus --config.file /etc/prometheus/prometheus.yml --storage.tsdb.path /var/lib/prometheus/ --web.console.templates=/etc/

Sep 26 09:12:31 mon prometheus[32261]: level=info ts=2023-09-26T09:12:31.631910232Z caller=main.go:304 host_details="(Linux 5.4.0-1092-aws #100~18.04.2-Ubuntu S
Sep 26 09:12:31 mon prometheus[32261]: level=info ts=2023-09-26T09:12:31.631952175Z caller=main.go:305 fd_limits="(soft=1024, hard=4096)"
Sep 26 09:12:31 mon prometheus[32261]: level=info ts=2023-09-26T09:12:31.631983667Z caller=main.go:306 vm_limits="(soft=unlimited, hard=unlimited)"
Sep 26 09:12:31 mon prometheus[32261]: level=info ts=2023-09-26T09:12:31.636602338Z caller=main.go:620 msg="Starting TSDB ..."
Sep 26 09:12:31 mon prometheus[32261]: level=info ts=2023-09-26T09:12:31.636671328Z caller=web.go:416 component=web msg="Start listening for connections" addres
Sep 26 09:12:31 mon prometheus[32261]: level=info ts=2023-09-26T09:12:31.661077185Z caller=main.go:635 msg="TSDB started"
Sep 26 09:12:31 mon prometheus[32261]: level=info ts=2023-09-26T09:12:31.661155465Z caller=main.go:695 msg="Loading configuration file" filename=/etc/prometheus
Sep 26 09:12:31 mon prometheus[32261]: level=info ts=2023-09-26T09:12:31.6621346Z caller=main.go:722 msg="Completed loading of configuration file" filename=/etc
Sep 26 09:12:31 mon prometheus[32261]: level=info ts=2023-09-26T09:12:31.662173397Z caller=main.go:589 msg="Server is ready to receive web requests."
Sep 26 09:12:36 mon prometheus[32261]: level=error ts=2023-09-26T09:12:36.640107365Z caller=notifier.go:481 component=notifier alertmanager=http://alertmanager:
lines 1-18/18 (END)
```

Using the public IP address of the monitoring server, navigate to the Prometheus web UI in a new browser tab:

http://<MONITORING_IP_ADDRESS>:9090.

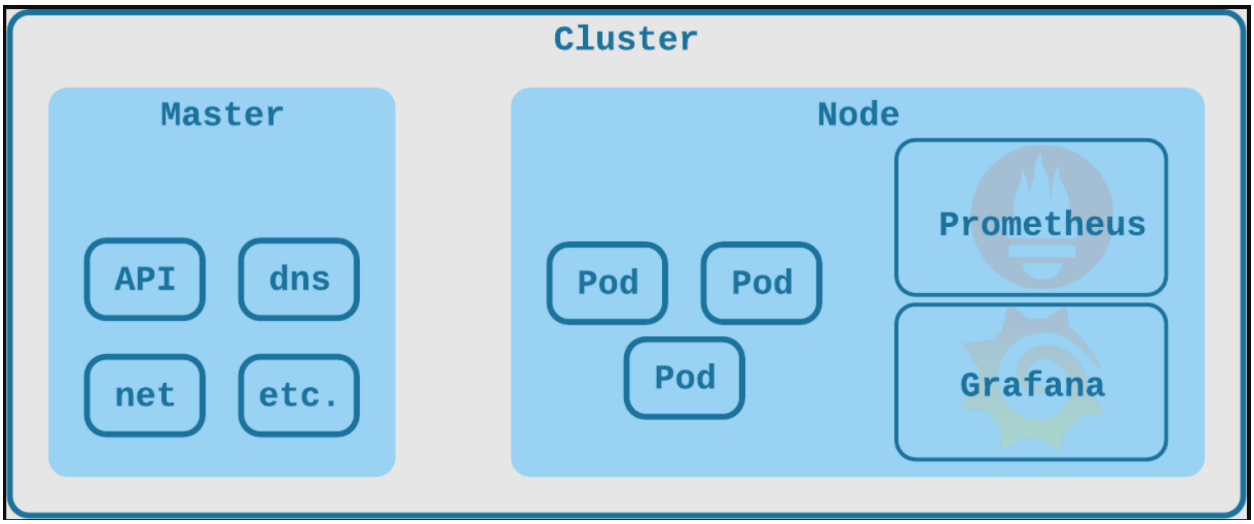


Click **Status > Targets**.



Ensure all three endpoints are listed on the Targets page.

Monitoring in Kubernetes with Prometheus and Grafana



Creating a Kubernetes Cluster
Introduction
We will install and configure a Kubernetes cluster consisting of 1 master and 1 or 2 nodes. Once the installation and configuration are complete, we will have a 2-node Kubernetes cluster that uses Flannel as the network overlay.

We will be using Centos 7.x for all virtual machines.

Logging In
NOTE: It's probably a good idea to have two or three terminals open, one for each node.

Install Docker and Kubernetes on All Servers
Most of these commands need to be run on each of the nodes. Pay attention though. Down at Step 10, we are going to do a little bit on just the master, and down at Step 15 we'll run something on just the nodes. There are notes down there, just be watching for them.

1 - Once we have logged in, we need to elevate privileges using sudo:
sudo su

2 - Disable SELinux:
setenforce 0
sed -i --follow-symlinks 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/sysconfig/selinux

3 - Enable the br_netfilter module for cluster communication:
modprobe br_netfilter
echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables

4 - Ensure that the Docker dependencies are satisfied:
yum install -y yum-utils device-mapper-persistent-data lvm2

5 - Add the Docker repo and install Docker:
yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
yum install -y docker-ce

6 - Set the cgroup driver for Docker to systemd, reload systemd, then enable and start Docker:
sed -i '/^ExecStart/ s/$/ --exec-opt native.cgroupdriver=systemd/'
/usr/lib/systemd/system/docker.service
systemctl daemon-reload
systemctl enable docker --now

7 - Add the Kubernetes repo:
cat << EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=0
repo_gpgcheck=0
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF

8 - Install Kubernetes v1.14.0:
yum install -y kubelet-1.14.0-0 kubeadm-1.14.0-0 kubectl-1.14.0-0 kubernetes-cni-0.7.5

9 - Enable the kubelet service. The kubelet service will fail to start until the cluster is initialized, this is expected:
systemctl enable kubelet

Note: Complete the following section on the **MASTER ONLY!**

10 - Initialize the cluster using the IP range for Flannel:
kubeadm init --pod-network-cidr=10.244.0.0/16

11 - Copy the kubeadmn join command that is in the output. We will need this later.
NOTE: see **Addendum** for an example of the whole output from using the **kubeadm init** command.
12 - Exit sudo, copy the admin.conf to your home directory, and take ownership.
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

13 - Deploy Flannel:
kubectl apply -f https://raw.githubusercontent.com/flannel-io/flannel/master/Documentation/kube-flannel-old.yaml

14 - Check the cluster state:
kubectl get pods --all-namespaces

Note: Complete the following steps on the **NODES ONLY!**

15 - Run the join command that you copied earlier, this requires running the command prefaced with sudo on the nodes (if we hadn't run sudo su to begin with).

Then we'll check the nodes from the **master**.

kubectl get nodes

Create and Scale a Deployment Using kubectl

Note: These commands will only be run on the **master node.**

16 - Create a simple deployment:
kubectl create deployment nginx --image=nginx

17 - Inspect the pod:
kubectl get pods

18 - Scale the deployment:
kubectl scale deployment nginx --replicas=4

19 - Inspect the pods. We should have four now:
kubectl get pods

Configure permissions for Docker on both Master and node/s

sudo chmod 666 /var/run/docker.sock

docker run hello-world

Also install the following on both Master and node/s

Install the Git package:

sudo yum install git -y

Install Python3 on both Master and node/s:

NOTE: It is important to ensure that both **PIP** and **botocore boto3** libraries are installed on this system. Please follow these steps:
sudo yum update
sudo yum install python3
sudo yum install python3-pip
pip install botocore boto3
python3 –version

Install Java 11 on both Master and node/s
sudo apt install default-jdk -y

Install components to use Gradle on both Master and node/s

sudo yum update
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh | bash
nvm install node

A task to perform on the Node

git clone https://github.com/ACloudGuru-Resources/cicd-pipeline-train-schedule-pipelines.git

cd

./gradlew build

./gradlew npm_start

Install Helm v2.8.2 on the Master

Helm is a package manager for Kubernetes!

Helm 3 can be installed many ways. We will install Helm 2.8.2 using the scripts option.

1.Download the script:
Download Helm v2.8.2 for your operating system from the Helm GitHub releases page.

Replace the URL with the appropriate one for your OS:

wget https://get.helm.sh/helm-v2.8.2-linux-amd64.tar.gz
tar -zxvf helm-v2.8.2-linux-amd64.tar.gz

If you're using a different OS, adjust the URL accordingly.

2.Move Helm Binary to a Directory in Your PATH:

sudo mv linux-amd64/helm /usr/local/bin/

Move the Helm binary to a directory in your PATH so you can use it conveniently.

Initialize Helm with Tiller and the Specific Image:
helm init --stable-repo-url=https://charts.helm.sh/stable --wait --tiller-image ghcr.io/helm/tiller:v2.8.2

```
cloud_user@ip-10-0-1-101:~$ helm init --stable-repo-url=https://charts.helm.sh/stable --wait --tiller-image ghcr.io/helm/tiller:v2.8.2
Creating /home/cloud_user/.helm
Creating /home/cloud_user/.helm/repository
Creating /home/cloud_user/.helm/repository/cache
Creating /home/cloud_user/.helm/repository/local
Creating /home/cloud_user/.helm/plugins
Creating /home/cloud_user/.helm/starters
Creating /home/cloud_user/.helm/cache/archive
Creating /home/cloud_user/.helm/repository/repositories.yaml
Adding stable repo with URL: https://charts.helm.sh/stable
Adding local repo with URL: http://127.0.0.1:8879/charts
$HELM_HOME has been configured at /home/cloud_user/.helm.

Tiller (the Helm server-side component) has been installed into your Kubernetes Cluster.

Please note: by default, Tiller is deployed with an insecure 'allow unauthenticated users' policy.
For more information on securing your installation see: https://docs.helm.sh/using_helm/#securing-your-helm-installation
Happy Helming!
cloud_user@ip-10-0-1-101:~$
```

Install Prometheus in the Kubernetes Cluster
To do this, make sure you have cloned the Kubernetes charts repo:

cd ~/
git clone https://github.com/kubernetes/charts

```
cloud_user@ip-10-0-1-101:~$  git clone https://github.com/kubernetes/charts
Cloning into 'charts'...
remote: Enumerating objects: 129385, done.
remote: Total 129385 (delta 0), reused 0 (delta 0), pack-reused 129385
Receiving objects: 100% (129385/129385), 284.57 MiB | 59.74 MiB/s, done.
Resolving deltas: 100% (87191/87191), done.
Checking connectivity... done.
cloud_user@ip-10-0-1-101:~$
```

cd charts
git checkout efdcffe0b6973111ec6e5e83136ea74cdbe6527d

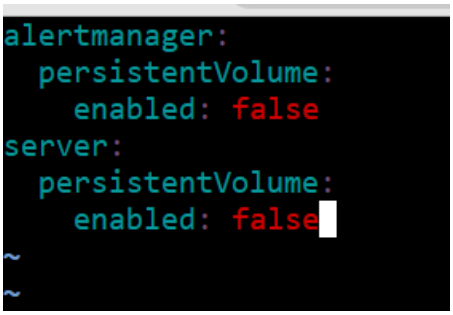
```
cloud_user@ip-10-0-1-101:~/charts$ git checkout efdcffe0b6973111ec6e5e83136ea74cdbe6527d
Note: checking out 'efdcffe0b6973111ec6e5e83136ea74cdbe6527d'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at efdcffe... [stable/prometheus] Ability to enable admin API (#5570)
cloud_user@ip-10-0-1-101:~/charts$
```

cd ../

Create a prometheus-values.yml file:

vi prometheus-values.yml
And paste in this content:

```
alertmanager:
  persistentVolume:
    enabled: false
server:
  persistentVolume:
    enabled: false
```



Save and close the file:

:wq

Use helm to install Prometheus with prometheus-values.yml:

helm install -f ~/prometheus-values.yml ~/charts/stable/prometheus --name prometheus --namespace prometheus

```
cloud_user@ip-10-0-1-101:~$  vi prometheus-values.yml
cloud_user@ip-10-0-1-101:~$  helm install -f ~/prometheus-values.yml ~/charts/stable/prometheus --name prometheus --namespace prometheus
```

```
NOTES:
The Prometheus server can be accessed via port 80 on the following DNS name from within your cluster:
prometheus-server.prometheus.svc.cluster.local


Get the Prometheus server URL by running these commands in the same shell:
  export POD_NAME=$(kubectl get pods --namespace prometheus -l "app=prometheus,component=server" -o jsonpath="{.items[0].metadata.name}")
  kubectl --namespace prometheus port-forward $POD_NAME 9090
#################################################################################
######   WARNING: Persistence is disabled!!! You will lose your data when   #####
######            the Server pod is terminated.                             #####
#################################################################################


The Prometheus alertmanager can be accessed via port 80 on the following DNS name from within your cluster:
prometheus-alertmanager.prometheus.svc.cluster.local


Get the Alertmanager URL by running these commands in the same shell:
  export POD_NAME=$(kubectl get pods --namespace prometheus -l "app=prometheus,component=alertmanager" -o jsonpath="{.items[0].metadata.name}")
  kubectl --namespace prometheus port-forward $POD_NAME 9093
#################################################################################
######   WARNING: Persistence is disabled!!! You will lose your data when   #####
######            the AlertManager pod is terminated.                       #####
#################################################################################


The Prometheus PushGateway can be accessed via port 9091 on the following DNS name from within your cluster:
prometheus-pushgateway.prometheus.svc.cluster.local


Get the PushGateway URL by running these commands in the same shell:
  export POD_NAME=$(kubectl get pods --namespace prometheus -l "app=prometheus,component=pushgateway" -o jsonpath="{.items[0].metadata.name}")
  kubectl --namespace prometheus port-forward $POD_NAME 9091

For more information on running Prometheus, visit:
https://prometheus.io/

cloud_user@ip-10-0-1-101:~$
```

We can see which pods are running in this new namespace with the command:

kubectl get pods -n prometheus

```
cloud_user@ip-10-0-1-101:~$  kubectl get pods -n prometheus
NAME                                             READY     STATUS    RESTARTS   AGE
prometheus-alertmanager-7b47d55bdf-7h4wd         2/2       Running   0          1m
prometheus-kube-state-metrics-6584885ccf-twdb2   1/1       Running   0          1m
prometheus-node-exporter-tk2td                   1/1       Running   0          1m
prometheus-pushgateway-66c9fdb48f-d4xch          1/1       Running   0          1m
prometheus-server-65d5cc8544-b224f               2/2       Running   0          1m
cloud_user@ip-10-0-1-101:~$
```

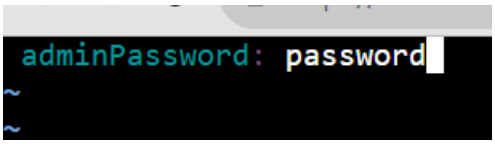
Install Grafana in the Kubernetes Cluster

Create a grafana-values.yml:

vi grafana-values.yml

And paste in this content (you will use this password to log in to Grafana):

adminPassword: password

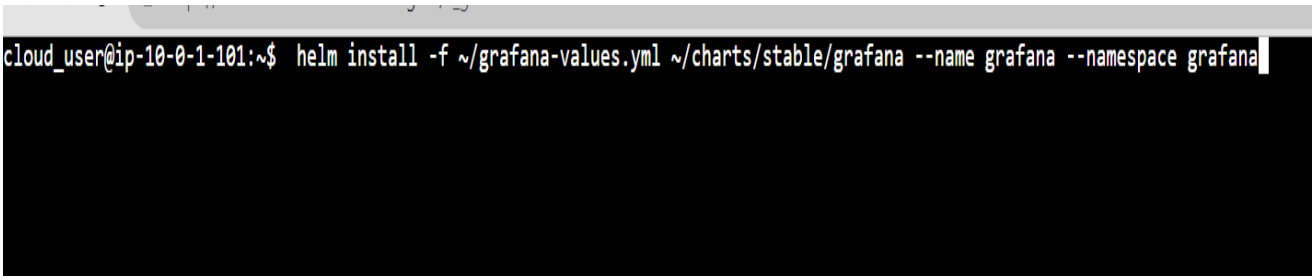


Save and close the file:

:wq

Use helm to install Grafana with grafana-values.yml:

helm install -f ~/grafana-values.yml ~/charts/stable/grafana --name grafana --namespace grafana



```
==> v1/ConfigMap
NAME                    DATA  AGE
grafana                 1     0s
grafana-dashboards-json 0     0s

==> v1/Service
NAME     TYPE       CLUSTER-IP    EXTERNAL-IP  PORT(S)  AGE
grafana  ClusterIP  10.104.74.80  <none>       80/TCP   0s

==> v1beta2/Deployment
NAME     DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
grafana  1        1        1           0          0s

==> v1/Pod(related)
NAME                     READY  STATUS            RESTARTS  AGE
grafana-d65c89cdc-778vg  0/1    ContainerCreating  0        0s


NOTES:
1. Get your 'admin' user password by running:

   kubectl get secret --namespace grafana grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo

2. The Grafana server can be accessed via port 80 on the following DNS name from within your cluster:

   grafana.grafana.svc.cluster.local

   Get the Grafana URL to visit by running these commands in the same shell:

     export POD_NAME=$(kubectl get pods --namespace grafana -l "app=grafana,component=" -o jsonpath="{.items[0].metadata.name}")
     kubectl --namespace grafana port-forward $POD_NAME 3000

3. Login with the password from step 1 and the username: admin
#################################################################################
######   WARNING: Persistence is disabled!!! You will lose your data when   #####
######           the Grafana pod is terminated.                             #####
#################################################################################

cloud_user@ip-10-0-1-101:~$
```

We can see which pods are running in this new namespace with the command:

kubectl get pods -n grafana

```
cloud_user@ip-10-0-1-101:~$ kubectl get pods -n grafana
NAME                       READY     STATUS    RESTARTS   AGE
grafana-d65c89cdc-778vg    0/1       Running   0          1m
cloud_user@ip-10-0-1-101:~$
```

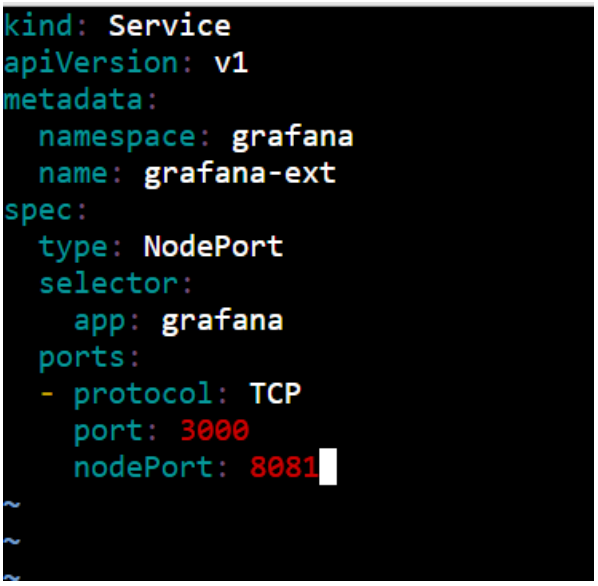
Deploy a NodePort Service to Provide External Access to Grafana

Make a file called grafana-ext.yml:

vi grafana-ext.yml

Paste in this content:

```
kind: Service
apiVersion: v1
metadata:
  namespace: grafana
  name: grafana-ext
spec:
  type: NodePort
  selector:
    app: grafana
  ports:
  - protocol: TCP
    port: 3000
    nodePort: 8081
```



Save and close the file:

:wq

Deploy the service:

kubectl apply -f ~/grafana-ext.yml

```
cloud_user@ip-10-0-1-101:~$  vi grafana-ext.yml
cloud_user@ip-10-0-1-101:~$  kubectl apply -f ~/grafana-ext.yml
service "grafana-ext" created
cloud_user@ip-10-0-1-101:~$
```

Log in to Grafana using the Kubernetes Node Public IP provided on the hands-on lab page:

<Grafana Public IP address>:8081

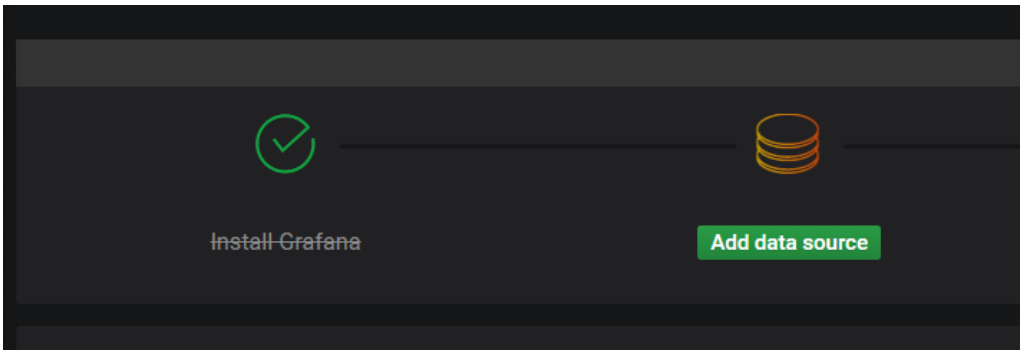
Log in using the following credentials that were set earlier:

Username: admin
Password: password

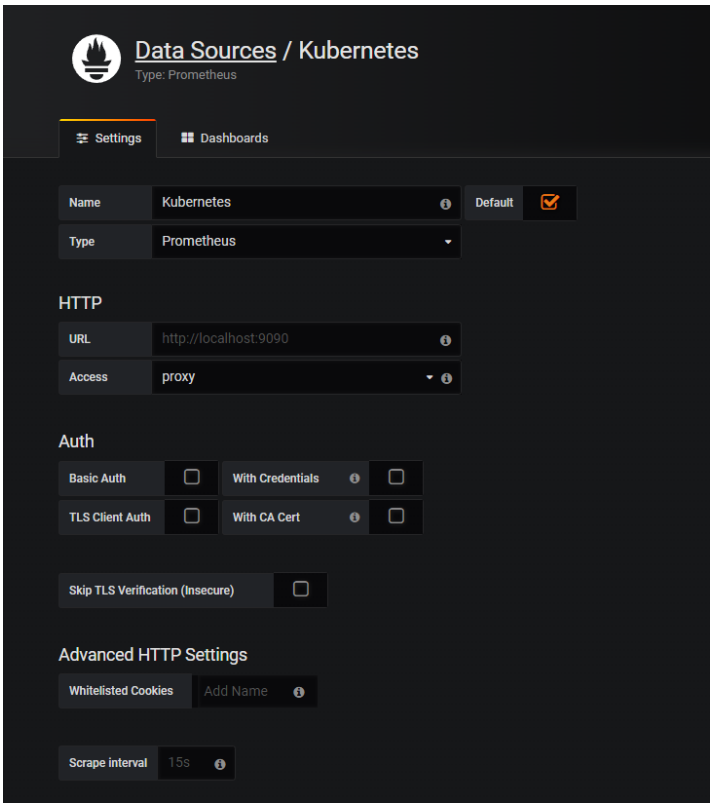
Create the Monitoring Dashboards

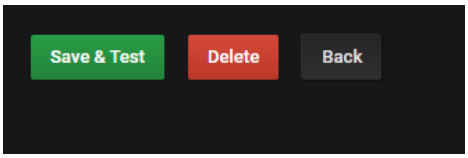
Add a Datasource for Prometheus

Click on **Add data source**



Name: Kubernetes
Type: Prometheus
URL: http://prometheus-server.prometheus.svc.cluster.local
Click Save & Test





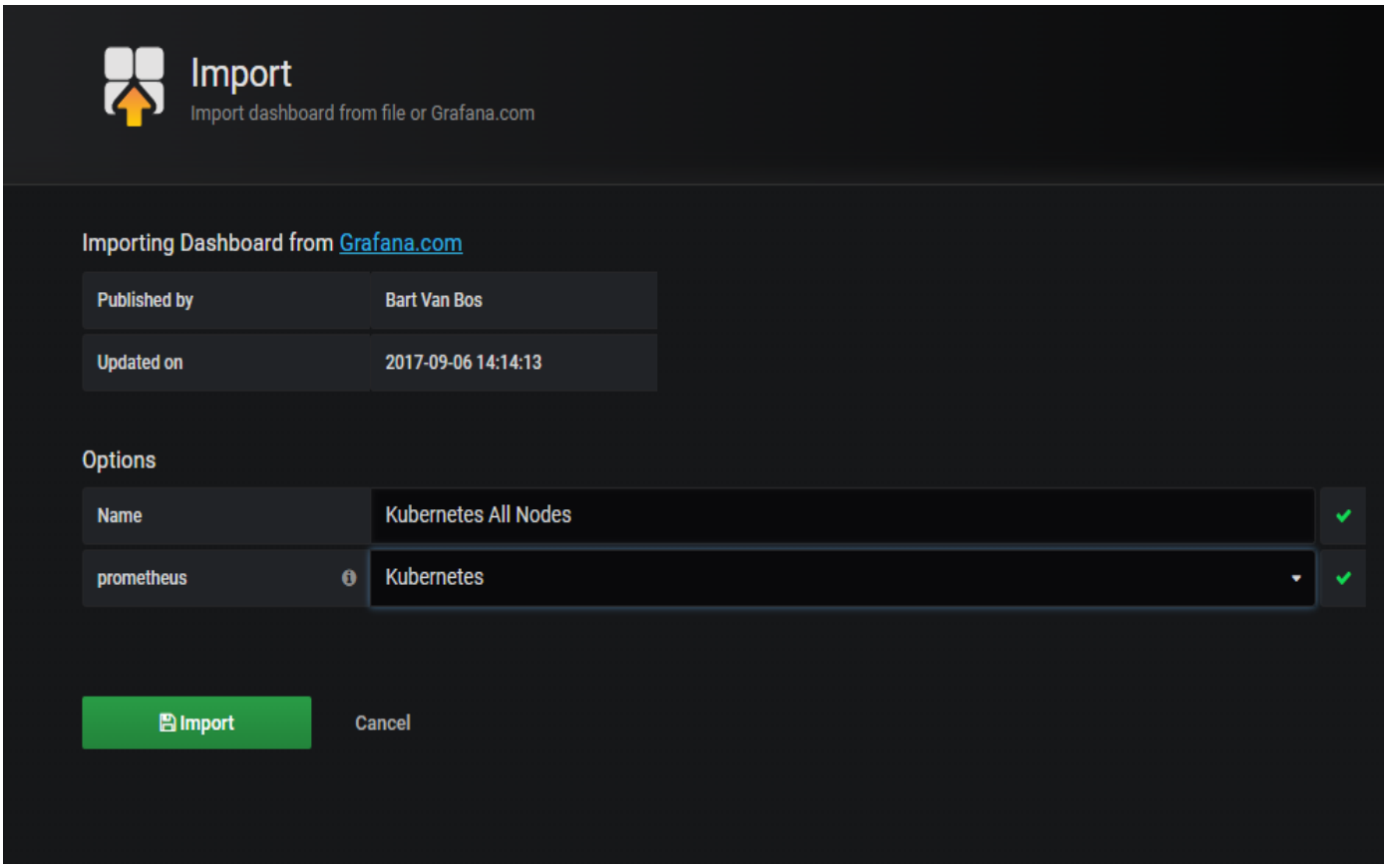
Add the Kubernetes All Nodes Community Dashboard

Hover your mouse over the **+** in the left sidebar and click on **Import**.

In the Grafana.com Dashboard field, provide the ID **3131**.

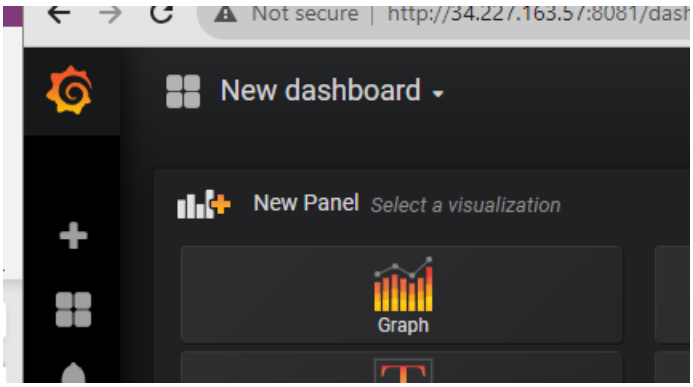
Click outside of the field to load information about the dashboard.

In the Options section:
prometheus: **Kubernetes**
Click **Import**.



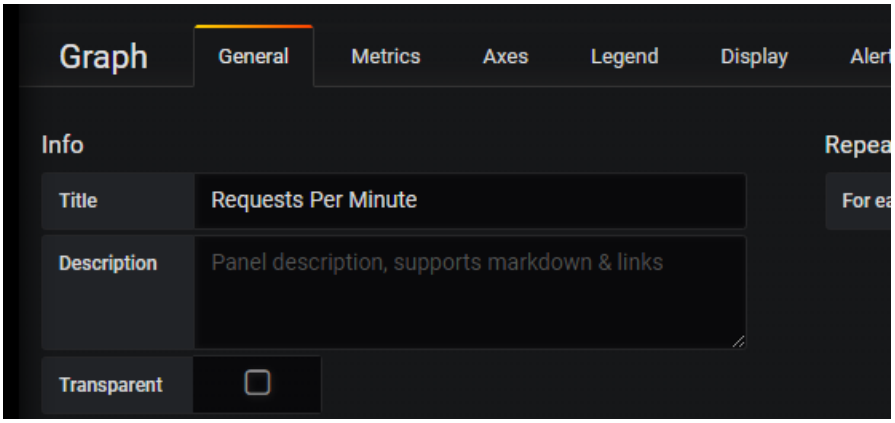
Hover your mouse over the **+** in the left sidebar and then click **Dashboard**.

Select the Graph panel.



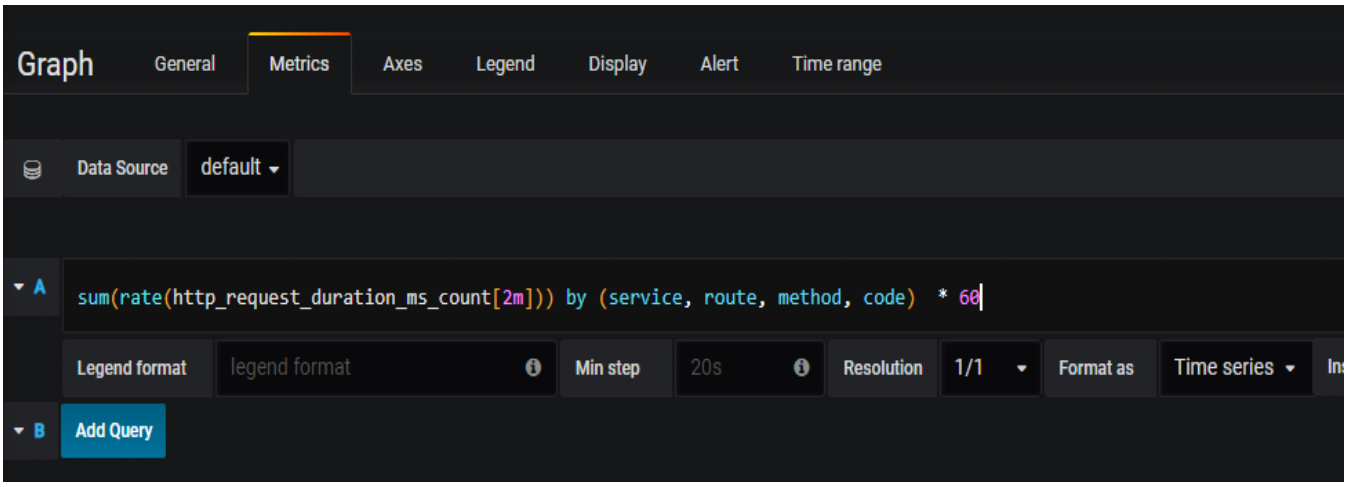
Hover over the Panel Title and click Edit.

In the General tab at the bottom of the screen, set the Title to Requests Per Minute.



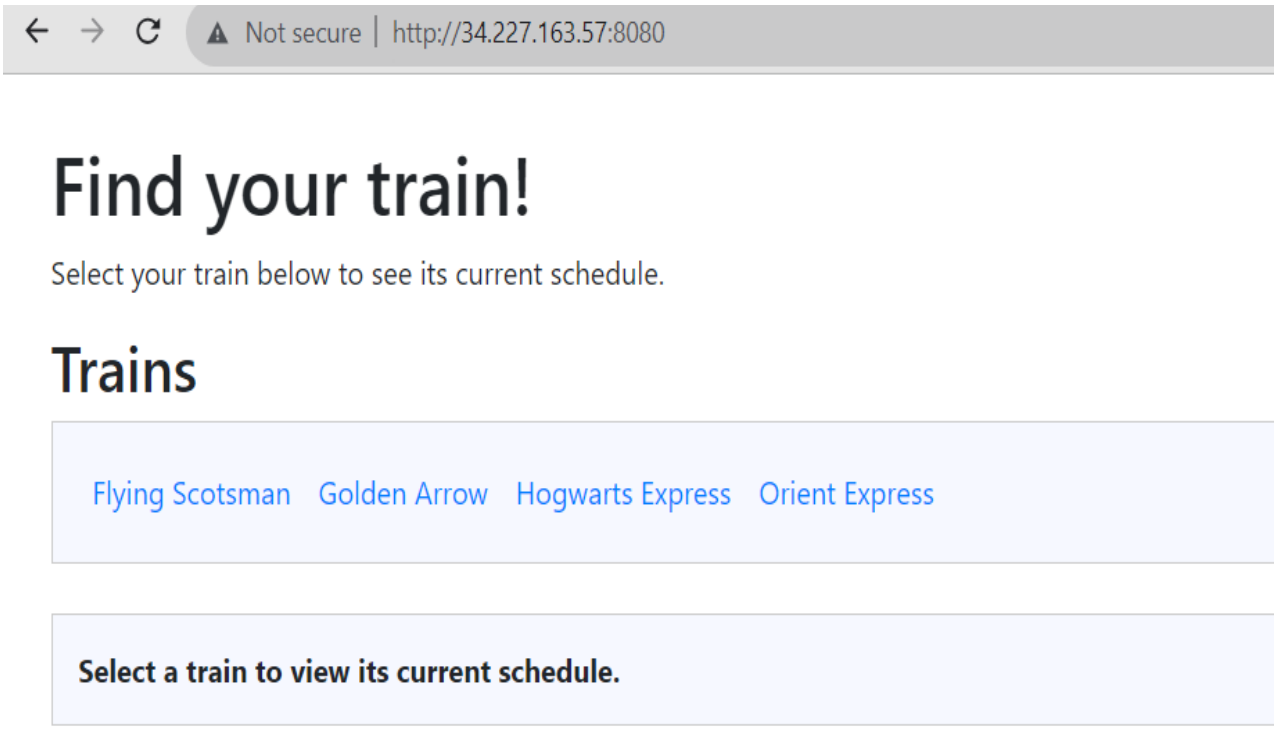
In the Metrics tab, paste in the following query:

sum(rate(http_request_duration_ms_count[2m])) by (service, route, method, code) * 60



With that in place, let's load our train-schedule app to give our graph some data:

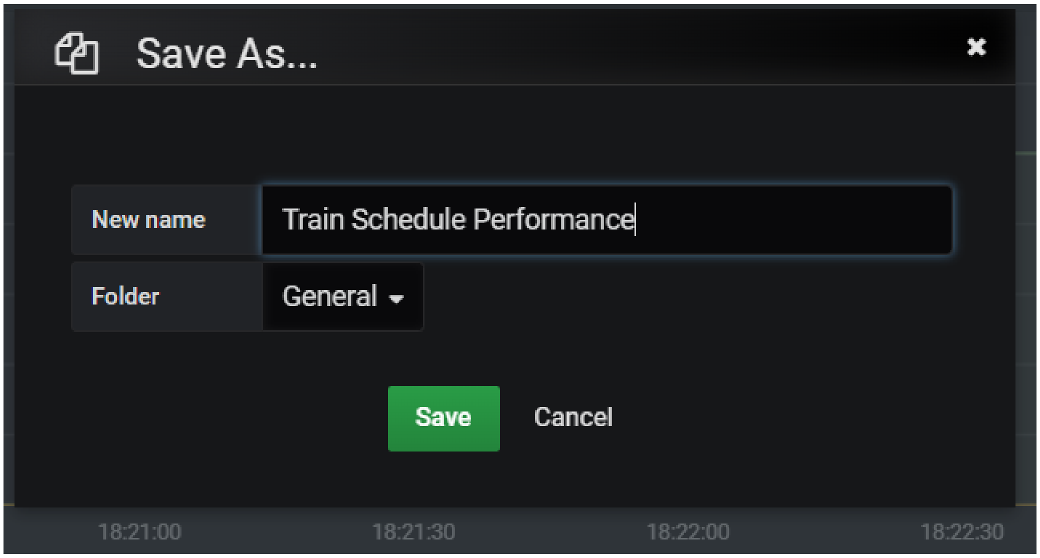
<Grafana Public IP Address>:8080



Refresh the page a few times.

Now, navigate back to the Grafana dashboard tab in your browser.
In the top-right of the page, click Last 6 hours and change the time selection to Last 5 minutes.

Click Back to dashboard in the top-right of the page.
Next, click Save dashboard, also in the top-right of the page.



Name the dashboard "**Train Schedule Performance**" and click **Save**.

//
Addendum
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

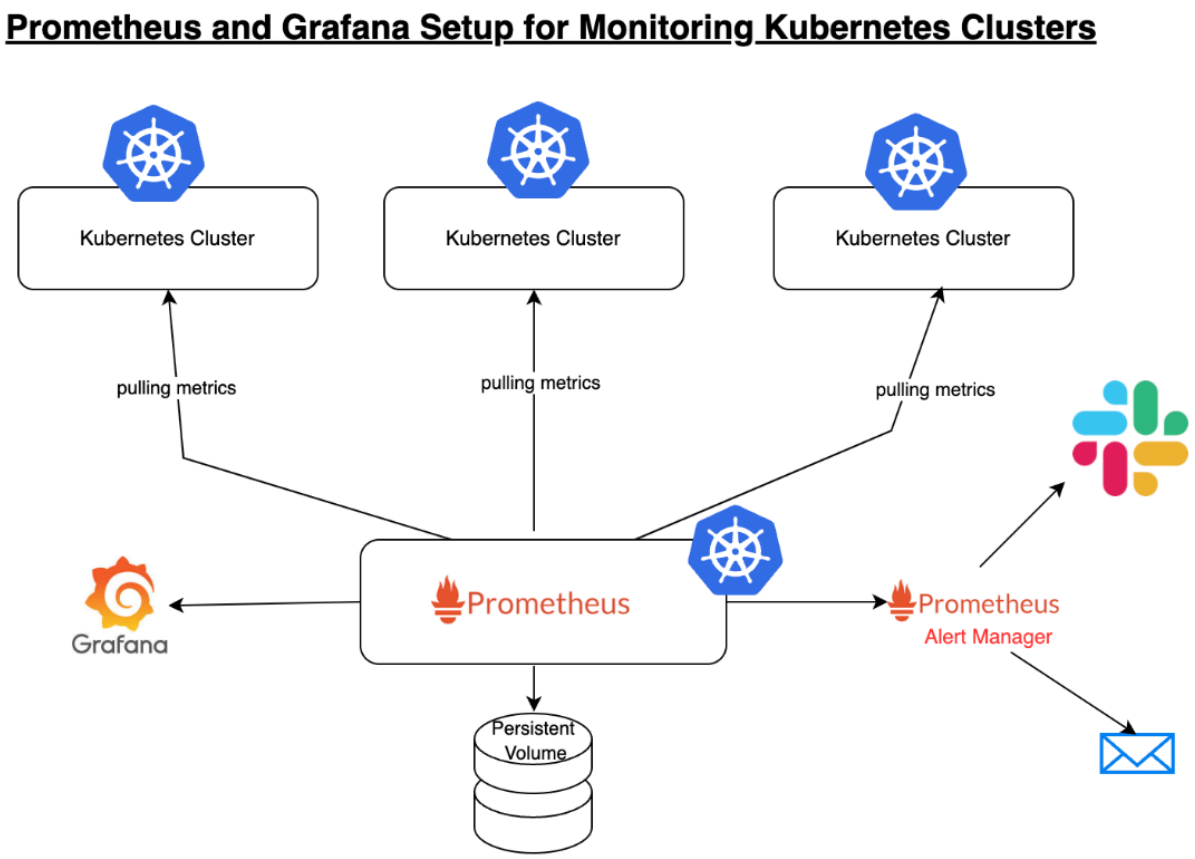
You should now deploy a pod network to the cluster.

Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

```
kubeadm join 172.31.27.197:6443 --token 2kla1r.ncp3uo7myontulj7 \
  --discovery-token-ca-cert-hash
sha256:228f75560859d6389ec4b6ac2b2b463f44bf42aa5f03c396596c8ec835e871ca
[root@9c438d91c5a44174a4a8c7a5cd9beb882c cloud_user]#
```
//

How to setup monitoring on an EKS Kubernetes Cluster using Prometheus and Grafana

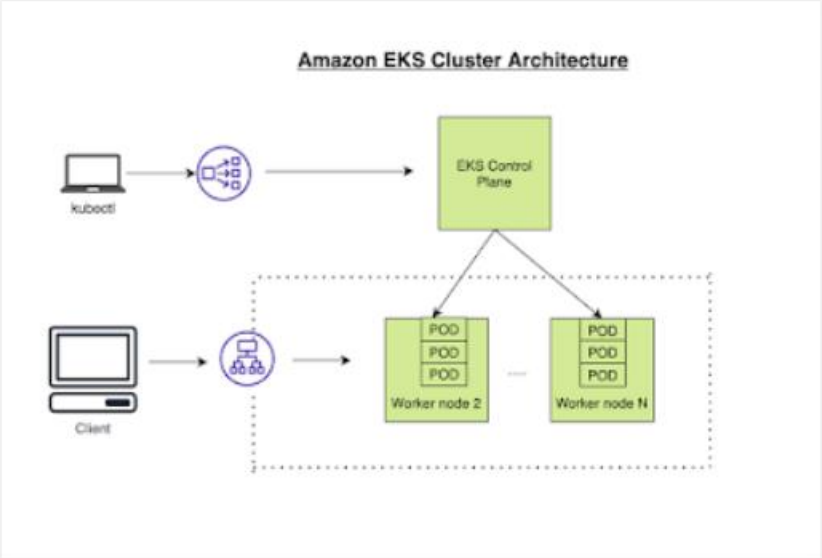


Pre-requisites:
EC2 instance to access EKS cluster.
EKS Cluster needs to be up and running.
Install Helm3

How to create EKS cluster in AWS cloud using eksctl.

What is Amazon EKS

Amazon EKS is a fully managed container orchestration service. EKS allows you to quickly deploy a production ready Kubernetes cluster in AWS, deploy and manage containerized applications more easily with a fully managed Kubernetes service.

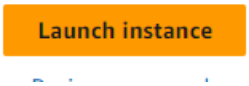


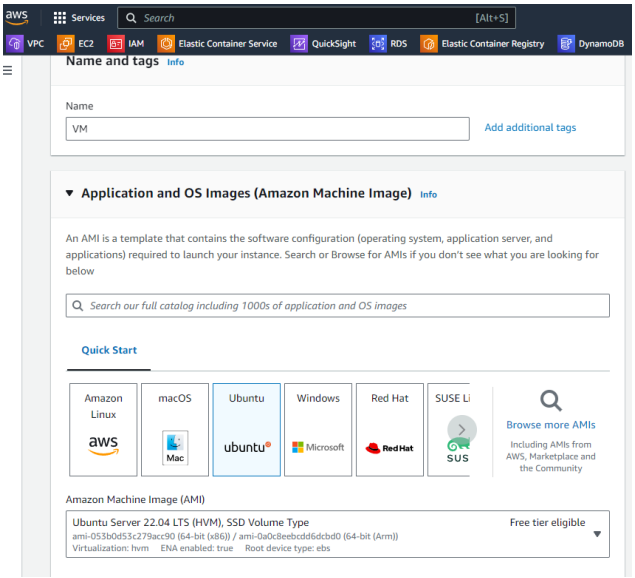
Pre-requisites:
An EC2 instance (Ubuntu 22.4 LTS)

Login to the AWS portal.

Go to **EC2**

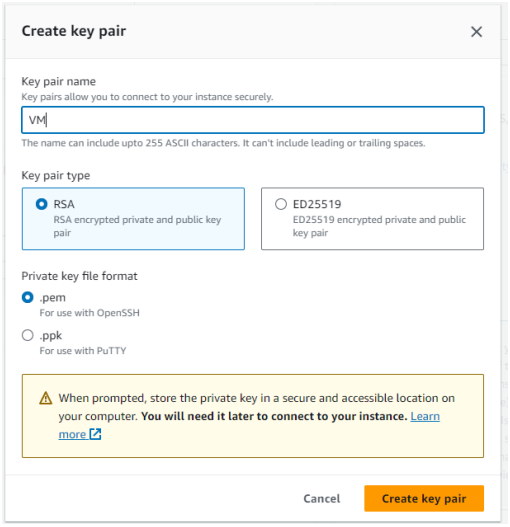
Launch Instance





Select Ubuntu

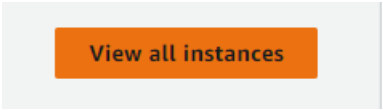
Create a new key pair.

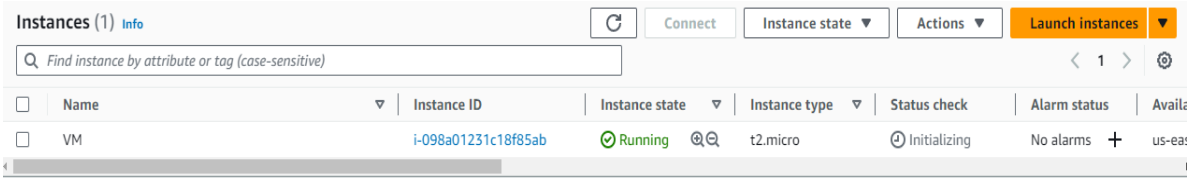


Click **Create key pair.**

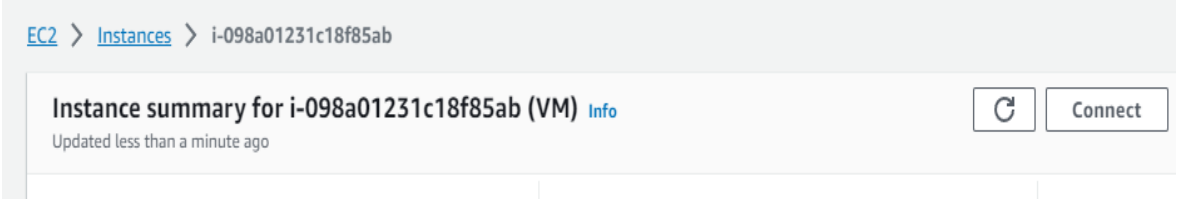
Create the Instance.

Now select **View all instances.**

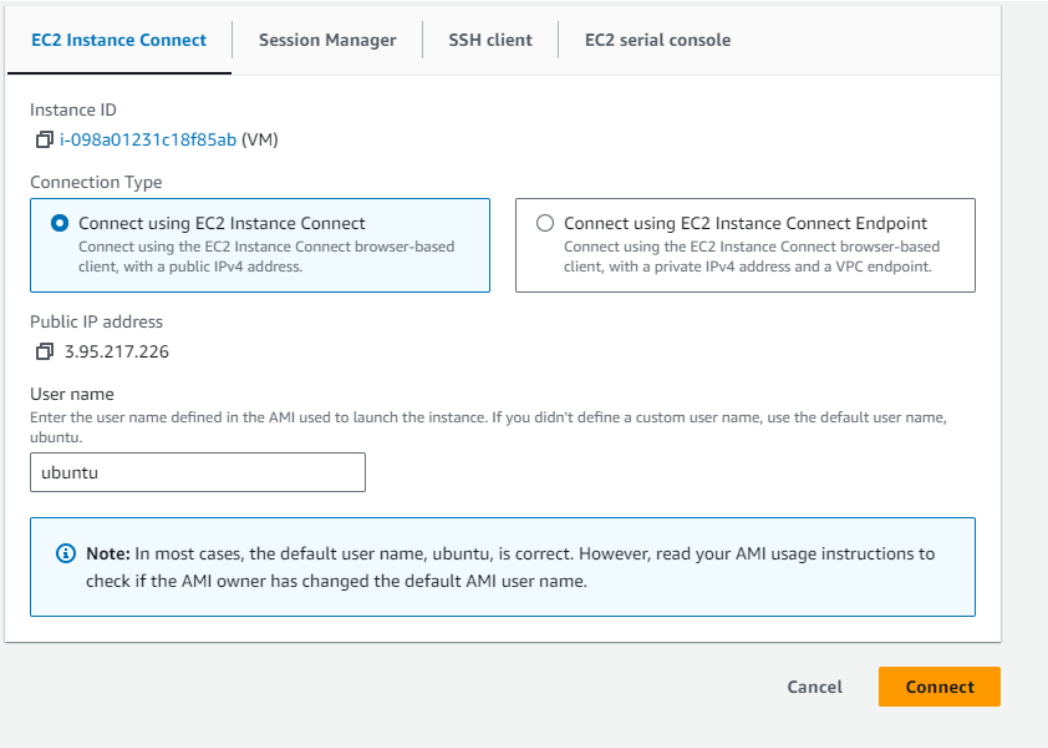




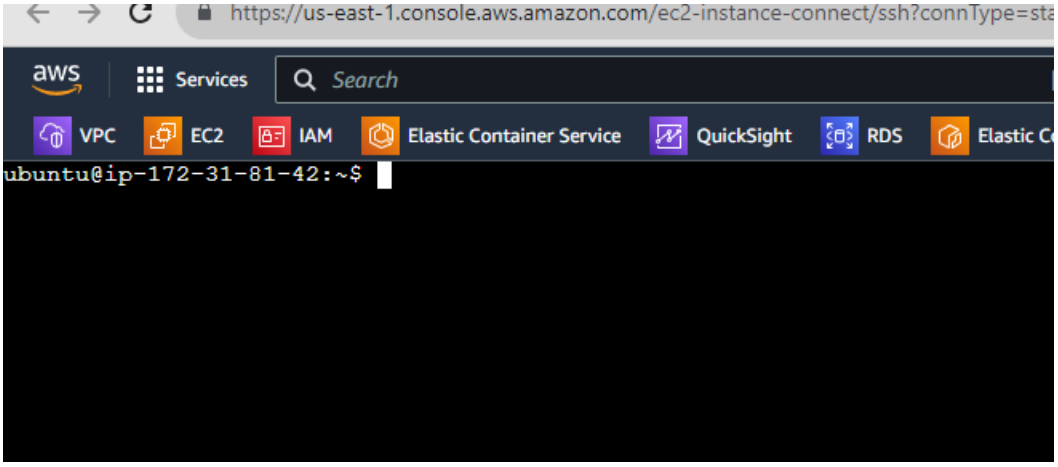
Select the instance on the left.



Select the **Connect** button on the right.

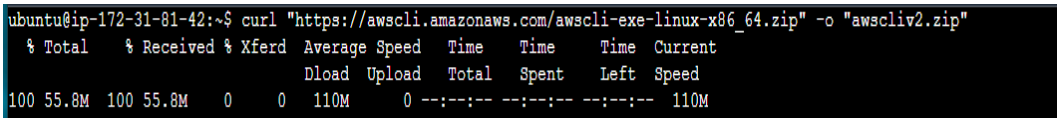


Click the **Connect** button.



Install AWS CLI – Command line tools for working with AWS services, including Amazon EKS.

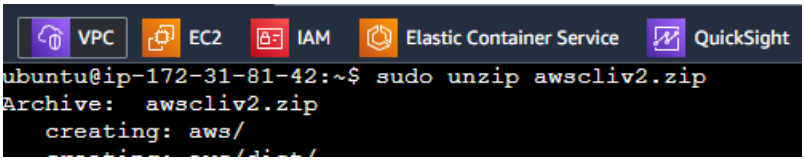
Download the AWS CLI
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"



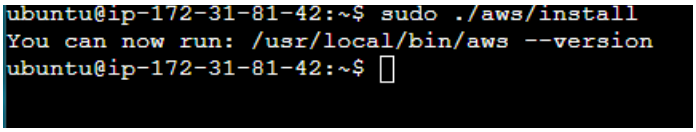
Install the unzip tool
sudo apt install unzip

```
ubuntu@ip-172-31-81-42:~$ sudo apt install unzip
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Suggested packages:
  zip
The following NEW packages will be installed:
  unzip
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 174 kB of archives.
After this operation, 385 kB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 unzip amd64 6.0-26ubuntu3.1 [174 kB]
Fetched 174 kB in 0s (6351 kB/s)
Selecting previously unselected package unzip.
(Reading database ... 64295 files and directories currently installed.)
Preparing to unpack .../unzip_6.0-26ubuntu3.1_amd64.deb ...
Unpacking unzip (6.0-26ubuntu3.1) ...
Setting up unzip (6.0-26ubuntu3.1) ...
Processing triggers for man-db (2.10.2-1) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-81-42:~$
```

Unzip the AWS CLI
sudo unzip awscliv2.zip



Install the AWS CLI tool
sudo ./aws/install



Check if the AWS CLI tool is installed.
aws --version

it should display the below output.

```
ubuntu@ip-172-31-81-42:~$ aws --version
aws-cli/2.13.22 Python/3.11.5 Linux/5.19.0-1025-aws exe/x86_64.ubuntu.22 prompt/off
ubuntu@ip-172-31-81-42:~$
```

Install eksctl – This is a command line tool for working with EKS clusters that automates many individual tasks.

The ***eksctl*** tool uses **CloudFormation** under the hood, creating one stack for the **EKS master control plane** and another stack for the **worker nodes**.

Download and extract the latest release of **eksctl** with the following command.

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

```
ubuntu@ip-172-31-81-42:~$ curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
ubuntu@ip-172-31-81-42:~$
```

Move the extracted binary to /usr/local/bin.

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

```
ubuntu@ip-172-31-81-42:~$ sudo mv /tmp/eksctl /usr/local/bin
ubuntu@ip-172-31-81-42:~$
```

Check if the eskctl is running.
eksctl version

```
ubuntu@ip-172-31-81-42:~$ eksctl version
0.159.0
ubuntu@ip-172-31-81-42:~$
```

Install kubectl – This is a command line tool for working with Kubernetes clusters.

Kubernetes uses a command line utility called kubectl for communicating with the cluster API server. It is tool for controlling Kubernetes clusters. kubectl looks for a file named config in the $HOME directory.

Downloading kubectl and placing it within a specific location.
sudo curl --silent --location -o /usr/local/bin/kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.6/2022-03-09/bin/linux/amd64/kubectl

```
ubuntu@ip-172-31-81-42:~$ sudo curl --silent --location -o /usr/local/bin/kubectl   https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.6/2022-03-09/bin/linux/amd64/kub
ectl
ubuntu@ip-172-31-81-42:~$
```

Making the kubectl executable.

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

```
ubuntu@ip-172-31-81-42:~$ sudo chmod +x /usr/local/bin/kubectl
ubuntu@ip-172-31-81-42:~$
```

Verifying if kubectl got installed

kubectl version --short --client

```
ubuntu@ip-172-31-81-42:~$ kubectl version --short --client
Client Version: v1.22.6-eks-7d68063
ubuntu@ip-172-31-81-42:~$
```

NOTE: The **–short** flag doesn't work on all Linux distributions.

Install Helm 3

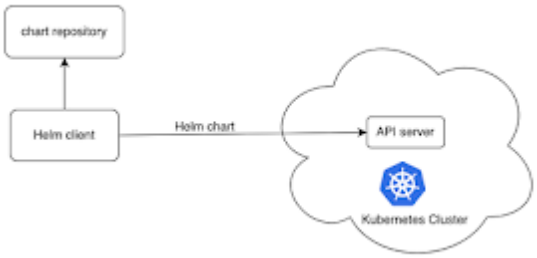
What is Helm and How to install Helm version 3?
Helm is a package manager for Kubernetes. Helm is the K8s equivalent of yum or apt. It accomplishes the same goals as Linux system package managers like APT or YUM: managing the installation of applications and dependencies behind the scenes and hiding the complexity from the user.

Why use Helm?
As the Kubernetes platform and ecosystem continued to expand, deploying one and only one Kubernetes configuration file (ie: a single YAML) was not the norm anymore. As number of K8S deployment files increased, how to manage those files? Helm solves that problem.

Helm Charts
Helm uses a packaging format called Charts. A Helm Chart is a collection of files that describe a set of Kubernetes resources. Helm Charts help you define, install, and upgrade even the most complex Kubernetes application. Charts are easy to create, version, share, and publish.

```
devopscoaching@DevOpsCoachMBP springboot % tree mychart
mychart
├── Chart.yaml
├── charts
├── templates
│   ├── NOTES.txt
│   ├── _helpers.tpl
│   ├── deployment.yaml
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── service.yaml
│   ├── serviceaccount.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml

4 directories, 10 files
```

Helm Kubernetes Integration



In Helm 3 there is no tiller component (a feature of earlier versions of Helm). The Helm client directly interacts with the Kubernetes API for the helm chart deployment.

Helm 3 can be installed in many ways. We will, however, install Helm 3 using scripts option.

Download scripts
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3

```
ubuntu@ip-172-31-81-42:~$ curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
ubuntu@ip-172-31-81-42:~$
```

Provide permission
sudo chmod 700 get_helm.sh

```
ubuntu@ip-172-31-81-42:~$ sudo chmod 700 get_helm.sh
ubuntu@ip-172-31-81-42:~$
```

Execute script to install
sudo ./get_helm.sh

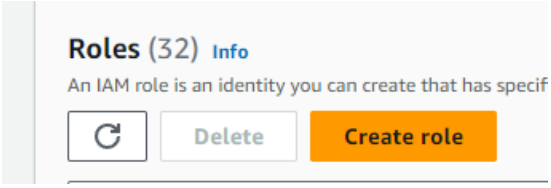
```
ubuntu@ip-172-31-81-42:~$ sudo ./get_helm.sh
Downloading https://get.helm.sh/helm-v3.13.0-linux-amd64.tar.gz
Verifying checksum... Done.
Preparing to install helm into /usr/local/bin
helm installed into /usr/local/bin/helm
ubuntu@ip-172-31-81-42:~$
```

Verify installation
helm version --client

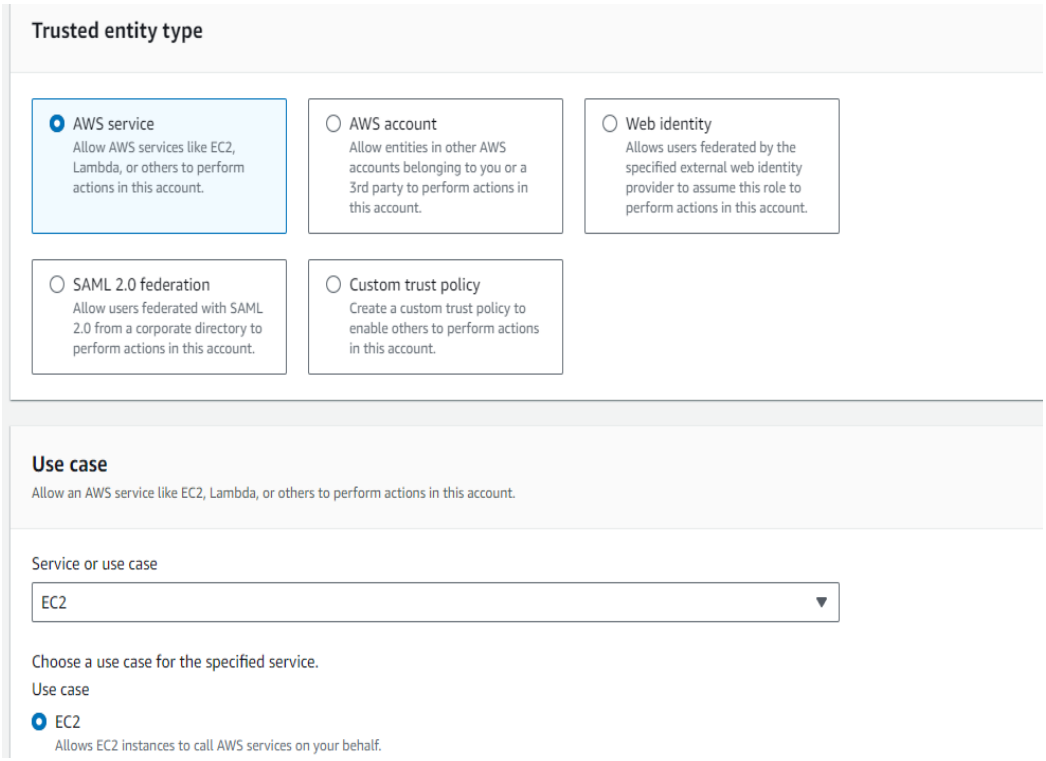
```
ubuntu@ip-172-31-81-42:~$ helm version --client
version.BuildInfo{Version:"v3.13.0", GitCommit:"825e86f6a7a38cef1112bfa606e4127a706749b1", GitTreeState:"clean", GoVersion:"go1.20.8"}
ubuntu@ip-172-31-81-42:~$
```

Create an IAM Role with Administrator Access
You need to create an **IAM** role with **AdministratorAccess** policy**.**

Go to AWS console, **IAM,** click on **Roles** and click **create role**

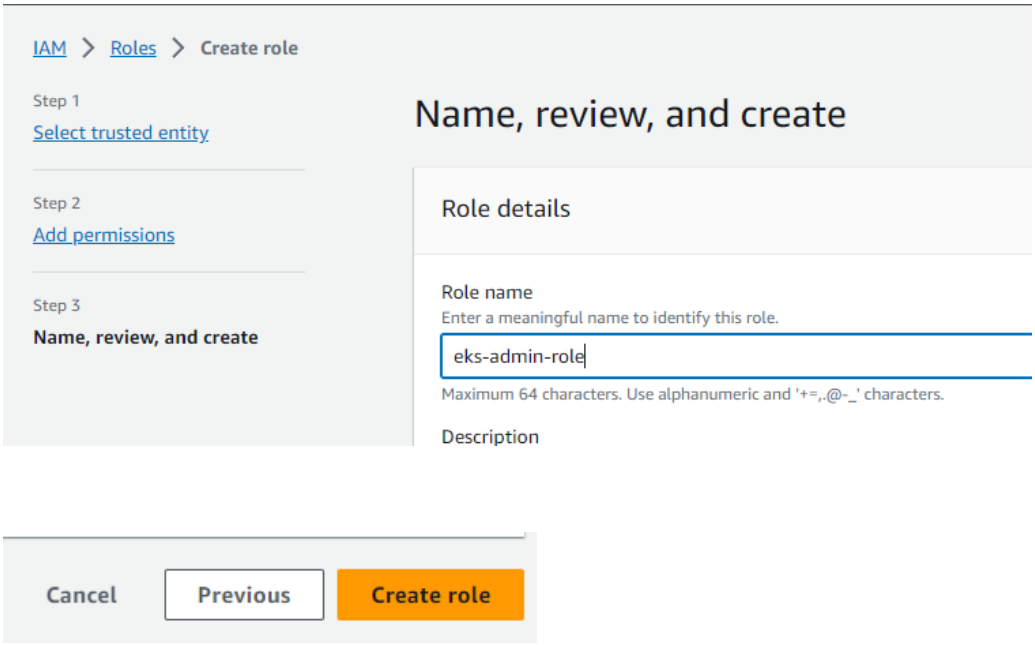


Select **AWS services**, Click **EC2**, Click on **Next** permissions.



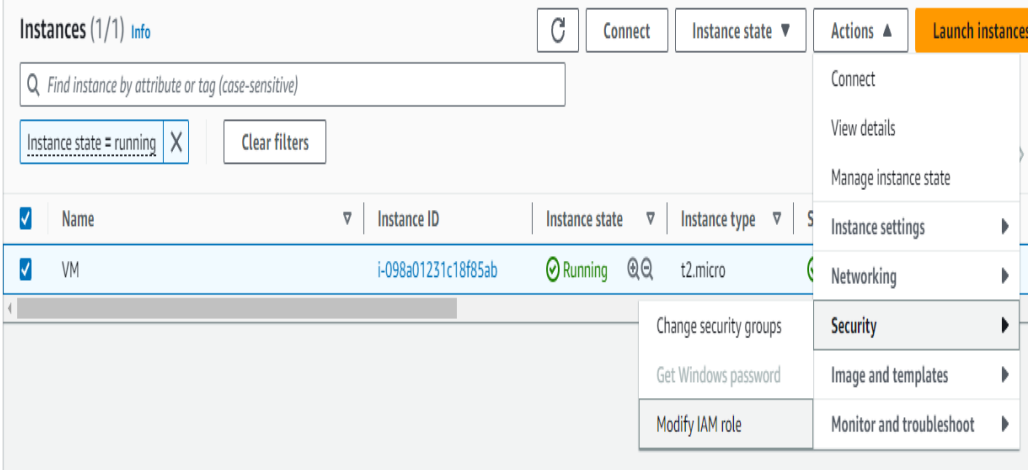
Now search for **AdministratorAccess** policy and click **Next**

Now give a role name and click **Create role**.



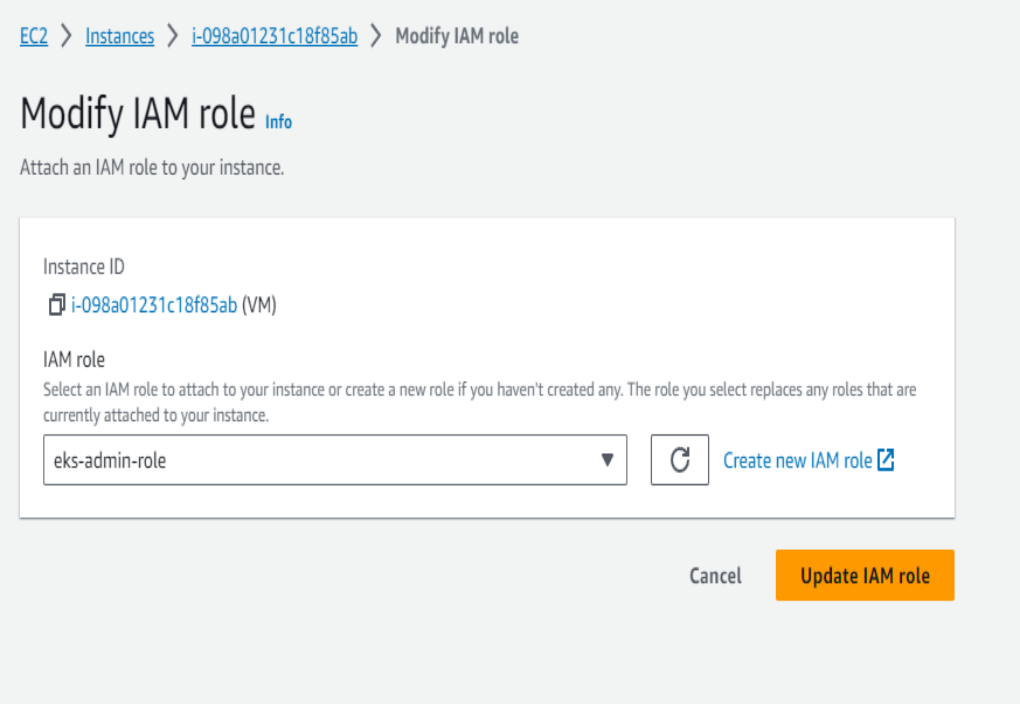
Assign the role to EC2 instance

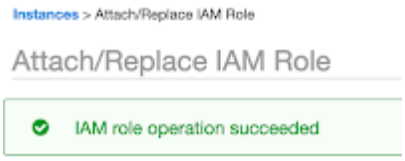
Go to **AWS** console, click on **EC2**, select the EC2 instance, Choose **Security**.
Click on **Modify IAM role**.



Choose the role you have created from the dropdown.

Select the role and click on **Apply**.



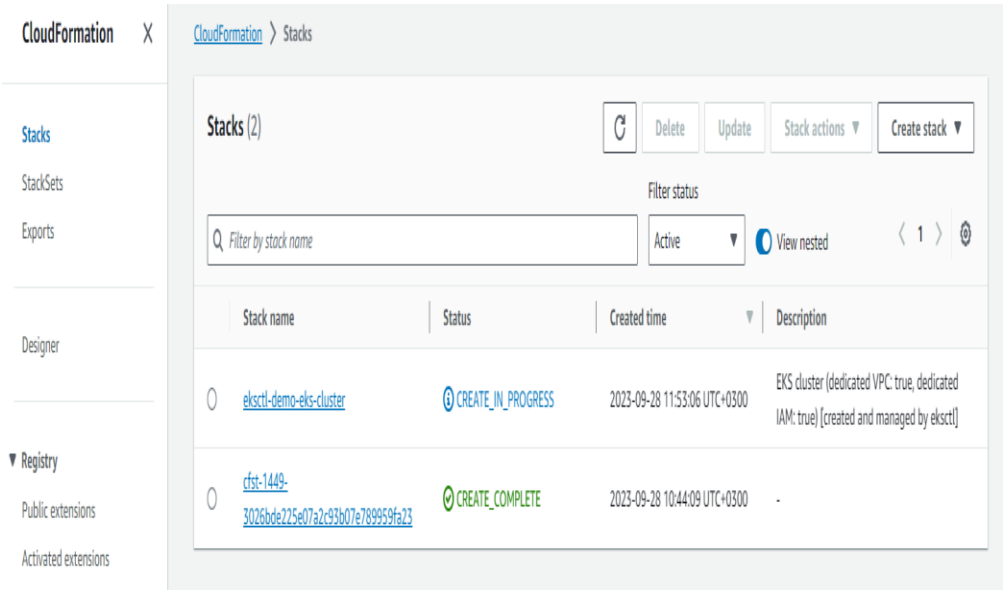


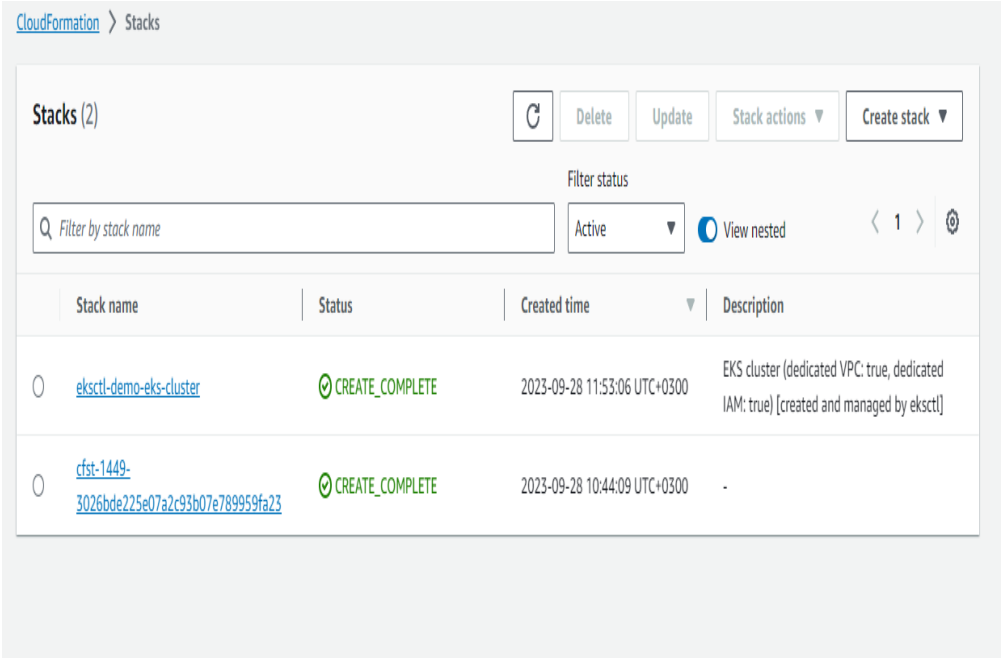
Create EKS Cluster with two worker nodes using eksctl

eksctl create cluster --name demo-eks --region us-east-1 --nodegroup-name my-nodes --node-type t3.small --managed --nodes 2

the above command should create a EKS cluster in AWS, it might take 15 to 20 mins. The ***eksctl*** tool uses CloudFormation under the hood, creating one stack for the EKS master control plane and another stack for the worker nodes.

```
ubuntu@ip-172-31-81-42:~$ eksctl create cluster --name demo-eks --region us-east-1 --nodegroup-name my-nodes --node-type t3.small --managed --nodes 2
2023-09-28 08:53:06 [ℹ]  eksctl version 0.159.0
2023-09-28 08:53:06 [ℹ]  using region us-east-1
2023-09-28 08:53:06 [ℹ]  skipping us-east-1e from selection because it doesn't support the following instance type(s): t3.small
2023-09-28 08:53:06 [ℹ]  setting availability zones to [us-east-1d us-east-1c]
2023-09-28 08:53:06 [ℹ]  subnets for us-east-1d - public:192.168.0.0/19 private:192.168.64.0/19
2023-09-28 08:53:06 [ℹ]  subnets for us-east-1c - public:192.168.32.0/19 private:192.168.96.0/19
2023-09-28 08:53:06 [ℹ]  nodegroup "my-nodes" will use "" [AmazonLinux2/1.25]
2023-09-28 08:53:06 [ℹ]  using Kubernetes version 1.25
2023-09-28 08:53:06 [ℹ]  creating EKS cluster "demo-eks" in "us-east-1" region with managed nodes
2023-09-28 08:53:06 [ℹ]  will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup
2023-09-28 08:53:06 [ℹ]  if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=us-east-1 --cluster=demo-eks'
2023-09-28 08:53:06 [ℹ]  Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "demo-eks" in "us-east-1"
2023-09-28 08:53:06 [ℹ]  CloudWatch logging will not be enabled for cluster "demo-eks" in "us-east-1"
2023-09-28 08:53:06 [ℹ]  you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=us-east-1 --clust
er=demo-eks'
2023-09-28 08:53:06 [ℹ]
2 sequential tasks: { create cluster control plane "demo-eks",
    2 sequential sub-tasks: {
        wait for control plane to become ready,
        create managed nodegroup "my-nodes",
    }
}
2023-09-28 08:53:06 [ℹ]  building cluster stack "eksctl-demo-eks-cluster"
2023-09-28 08:53:06 [ℹ]  deploying stack "eksctl-demo-eks-cluster"
```





The EKS cluster has been created.

eksctl get cluster --name demo-eks --region us-east-1

The above should confirm that EKS cluster is up and running.

```
ubuntu@ip-172-31-81-42:~$ eksctl get cluster --name demo-eks --region us-east-1
NAME            VERSION STATUS  CREATED              VPC                     SUBNETS                                                                                                                                 S
ECURITYGROUPS           PROVIDER
demo-eks        1.25    ACTIVE  2023-09-28T08:53:30Z    vpc-05fee4f22b2eb7f92   subnet-028e3389dd7a28274,subnet-059cb0cd9161466ae,subnet-0f5c03b32882d1e8f,subnet-0fd5d
41a07fbfc8de    sg-05c8b8d8016035fc7    EKS
ubuntu@ip-172-31-81-42:~$
```

Update Kube config by entering command below:

aws eks update-kubeconfig --name demo-eks --region us-east-1

```
ubuntu@ip-172-31-81-42:~$ aws eks update-kubeconfig --name demo-eks --region us-east-1
Added new context arn:aws:eks:us-east-1:594182463744:cluster/demo-eks to /home/ubuntu/.kube/config
ubuntu@ip-172-31-81-42:~$
```

Connect to EKS cluster using kubectl commands

To view the list of worker nodes as part of EKS cluster.

kubectl get nodes

```
ubuntu@ip-172-31-81-42:~$ kubectl get nodes
NAME                             STATUS   ROLES    AGE   VERSION
ip-192-168-28-73.ec2.internal    Ready    <none>   51s   v1.25.13-eks-43840fb
ip-192-168-52-210.ec2.internal   Ready    <none>   44s   v1.25.13-eks-43840fb
ubuntu@ip-172-31-81-42:~$
```

kubectl get ns

```
ubuntu@ip-172-31-81-42:~$ kubectl get ns
NAME              STATUS   AGE
default           Active   9m14s
kube-node-lease   Active   9m16s
kube-public       Active   9m16s
kube-system       Active   9m16s
ubuntu@ip-172-31-81-42:~$
```

Deploy Nginx on a Kubernetes Cluster
Let us run some apps to make sure they are deployed to Kubernetes cluster. The below command will create deployment:

kubectl create deployment nginx --image=nginx

```
ubuntu@ip-172-31-81-42:~$ kubectl create deployment nginx --image=nginx
deployment.apps/nginx created
ubuntu@ip-172-31-81-42:~$
```

View Deployments
kubectl get deployments

```
ubuntu@ip-172-31-81-42:~$ kubectl get deployments
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   1/1     1            1           45s
ubuntu@ip-172-31-81-42:~$
```

Errors during Cluster creation
If you are having issues when creating a cluster, try to delete the cluster by executing the below command and re-create it.

eksctl delete cluster --name demo-eks --region us-east-1

or Login to AWS console --> AWS Cloud formation --> delete the stack manually.

You can also delete the cluster under AWS console --> Elastic Kubernetes Service --> Clusters

Click on Delete cluster

Implementation steps
We need to add the Helm Stable Charts for your local client. Execute the below command:

helm repo add stable https://charts.helm.sh/stable

```
ubuntu@ip-172-31-81-42:~$ helm repo add stable https://charts.helm.sh/stable
"stable" has been added to your repositories
ubuntu@ip-172-31-81-42:~$
```

Add prometheus Helm repo
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

```
ubuntu@ip-172-31-81-42:~$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
"prometheus-community" has been added to your repositories
ubuntu@ip-172-31-81-42:~$
```

helm search repo prometheus-community

Prometheus and grafana helm chart moved to kube prometheus stack

```
ubuntu@ip-172-31-81-42:~$ helm search repo prometheus-community
NAME                                            CHART VERSION   APP VERSION     DESCRIPTION
prometheus-community/alertmanager               1.7.0           v0.26.0         The Alertmanager handles alerts sent by client ...
prometheus-community/alertmanager-snmp-notifier 0.1.2           v1.4.0          The SNMP Notifier handles alerts coming from Pr...
prometheus-community/jiralert                   1.6.0           v1.3.0          A Helm chart for Kubernetes to install jiralert
prometheus-community/kube-prometheus-stack      51.2.0          v0.68.0         kube-prometheus-stack collects Kubernetes manif...
prometheus-community/kube-state-metrics         5.13.0          2.10.0          Install kube-state-metrics to generate and expo...
prometheus-community/prom-label-proxy           0.6.0           v0.7.0          A proxy that enforces a given label in a given ...
prometheus-community/prometheus                 25.0.0          v2.47.0         Prometheus is a monitoring system and time seri...
prometheus-community/prometheus-adapter         4.5.0           v0.11.1         A Helm chart for k8s prometheus adapter
prometheus-community/prometheus-blackbox-exporter 8.4.0         v0.24.0         Prometheus Blackbox Exporter
prometheus-community/prometheus-cloudwatch-expo... 0.25.2       0.15.4          A Helm chart for prometheus cloudwatch-exporter
prometheus-community/prometheus-conntrack-stats... 0.5.7        v0.4.15         A Helm chart for conntrack-stats-exporter
prometheus-community/prometheus-consul-exporter 1.0.0           0.4.0           A Helm chart for the Prometheus Consul Exporter
prometheus-community/prometheus-couchdb-exporter 1.0.0          1.0             A Helm chart to export the metrics from couchdb...
prometheus-community/prometheus-druid-exporter  1.1.0           v0.11.0         Druid exporter to monitor druid metrics with Pr...
prometheus-community/prometheus-elasticsearch-e... 5.3.0         v1.6.0          Elasticsearch stats exporter for Prometheus
prometheus-community/prometheus-fastly-exporter 0.2.0           7.2.4           A Helm chart for the Prometheus Fastly Exporter
prometheus-community/prometheus-ipmi-exporter   0.1.0           1.6.1           This is an IPMI exporter for Prometheus.
prometheus-community/prometheus-json-exporter   0.7.1           v0.5.0          Install prometheus-json-exporter
prometheus-community/prometheus-kafka-exporter  2.6.0           v1.7.0          A Helm chart to export the metrics from Kafka i...
prometheus-community/prometheus-modbus-exporter 0.1.0           0.4.0           A Helm chart for prometheus-modbus-exporter
prometheus-community/prometheus-mongodb-exporter 3.4.0          0.39.0          A Prometheus exporter for MongoDB metrics
prometheus-community/prometheus-mysql-exporter  2.0.0           v0.15.0         A Helm chart for prometheus mysql exporter with...
prometheus-community/prometheus-nats-exporter   2.13.0          0.12.0          A Helm chart for prometheus-nats-exporter
prometheus-community/prometheus-nginx-exporter  0.1.1           0.11.0          A Helm chart for the Prometheus NGINX Exporter
prometheus-community/prometheus-node-exporter   4.23.1          1.6.1           A Helm chart for prometheus node-exporter
prometheus-community/prometheus-operator        9.3.2           0.38.1          DEPRECATED - This chart will be renamed. See ht...
prometheus-community/prometheus-operator-admiss... 0.7.0        0.68.0          Prometheus Operator Admission Webhook
prometheus-community/prometheus-operator-crds   6.0.0           v0.68.0         A Helm chart that collects custom resource defi...
prometheus-community/prometheus-pgbouncer-exporter 0.1.1        1.18.0          A Helm chart for prometheus pgbouncer-exporter
prometheus-community/prometheus-pingdom-exporter 2.5.0          20190610-1      A Helm chart for Prometheus Pingdom Exporter
prometheus-community/prometheus-pingmesh-exporter 0.3.0         v1.1.0          Prometheus Pingmesh Exporter
prometheus-community/prometheus-postgres-exporter 5.1.0         v0.14.0         A Helm chart for prometheus postgres-exporter
prometheus-community/prometheus-pushgateway     2.4.1           v1.6.1          A Helm chart for prometheus pushgateway
prometheus-community/prometheus-rabbitmq-exporter 1.8.1         v0.29.0         Rabbitmq metrics exporter for prometheus
prometheus-community/prometheus-redis-exporter  6.0.0           v1.54.0         Prometheus exporter for Redis metrics
prometheus-community/prometheus-smartctl-exporter 0.6.0         v0.11.0         A Helm chart for Kubernetes
```

Now create the Prometheus namespace
kubectl create namespace prometheus

```
ubuntu@ip-172-31-81-42:~$ kubectl create namespace prometheus

namespace/prometheus created
ubuntu@ip-172-31-81-42:~$
```

Install kube-prometheus-stack

Below is helm command to install kube-prometheus-stack. The helm repo kube-stack-prometheus (formerly prometheus-operator) comes with a grafana deployment embedded.

helm install stable prometheus-community/kube-prometheus-stack -n prometheus

```
ubuntu@ip-172-31-81-42:~$ helm install stable prometheus-community/kube-prometheus-stack -n prometheus

NAME: stable
LAST DEPLOYED: Thu Sep 28 09:15:30 2023
NAMESPACE: prometheus
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace prometheus get pods -l "release=stable"

Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create & configure Alertmanager and Prometheus instances using the Operator.
ubuntu@ip-172-31-81-42:~$
ubuntu@ip-172-31-81-42:~$
```

Lets check if prometheus and grafana pods are running already
kubectl get pods -n prometheus

```
ubuntu@ip-172-31-81-42:~$ kubectl get pods -n prometheus
NAME                                                     READY   STATUS    RESTARTS   AGE
alertmanager-stable-kube-prometheus-sta-alertmanager-0   2/2     Running   0          49s
prometheus-stable-kube-prometheus-sta-prometheus-0       2/2     Running   0          49s
stable-grafana-6c8fd88f56-zrcpr                          3/3     Running   0          54s
stable-kube-prometheus-sta-operator-75947cfc97-xjmmv     1/1     Running   0          54s
stable-kube-state-metrics-7ccdc6767c-xj26t               1/1     Running   0          54s
stable-prometheus-node-exporter-w646x                    1/1     Running   0          54s
stable-prometheus-node-exporter-x6w9w                    1/1     Running   0          54s
ubuntu@ip-172-31-81-42:~$
```

kubectl get svc -n prometheus

```
ubuntu@ip-172-31-81-42:~$ kubectl get svc -n prometheus
NAME                                     TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
alertmanager-operated                    ClusterIP   None             <none>        9093/TCP,9094/TCP,9094/UDP   90s
prometheus-operated                      ClusterIP   None             <none>        9090/TCP                     90s
stable-grafana                           ClusterIP   10.100.186.94    <none>        80/TCP                       95s
stable-kube-prometheus-sta-alertmanager  ClusterIP   10.100.254.197   <none>        9093/TCP,8080/TCP            95s
stable-kube-prometheus-sta-operator      ClusterIP   10.100.85.234    <none>        443/TCP                      95s
stable-kube-prometheus-sta-prometheus    ClusterIP   10.100.229.18    <none>        9090/TCP,8080/TCP            95s
stable-kube-state-metrics                ClusterIP   10.100.245.2     <none>        8080/TCP                     95s
stable-prometheus-node-exporter          ClusterIP   10.100.247.199   <none>        9100/TCP                     95s
ubuntu@ip-172-31-81-42:~$
```

This confirms that prometheus and grafana have been installed successfully using Helm.

In order to make prometheus and grafana available outside the cluster, we need ti use **LoadBalancer** or **NodePort** instead of **ClusterIP** within the service files.

Edit Prometheus Service

kubectl edit svc stable-kube-prometheus-sta-prometheus -n prometheus

```
  ipFamilyPolicy: SingleStack
  ports:
  - name: http-web
    port: 9090
    protocol: TCP
    targetPort: 9090
  - appProtocol: http
    name: reloader-web
    port: 8080
    protocol: TCP
    targetPort: reloader-web
  selector:
    app.kubernetes.io/name: prometheus
    operator.prometheus.io/name: stable-kube-prometheus-sta-prometheus
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer: {}
```

Edit Grafana Service

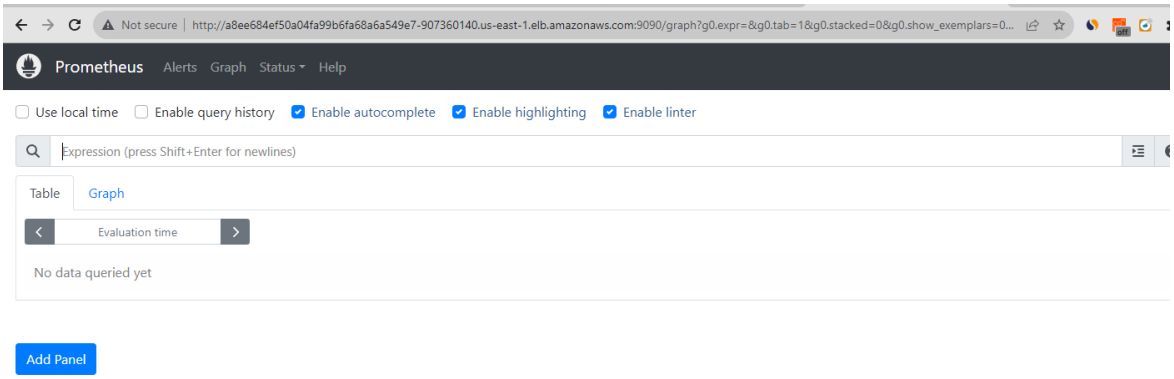
kubectl edit svc stable-grafana -n prometheus

```
  ipFamilyPolicy: SingleStack
  ports:
  - name: http-web
    port: 9090
    protocol: TCP
    targetPort: 9090
  - appProtocol: http
    name: reloader-web
    port: 8080
    protocol: TCP
    targetPort: reloader-web
  selector:
    app.kubernetes.io/name: prometheus
    operator.prometheus.io/name: stable-kube-prometheus-sta-prometheus
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer: {}
```

Verify if service is changed to **LoadBalancer** and also to get the **Load Balancer URL** for each application.

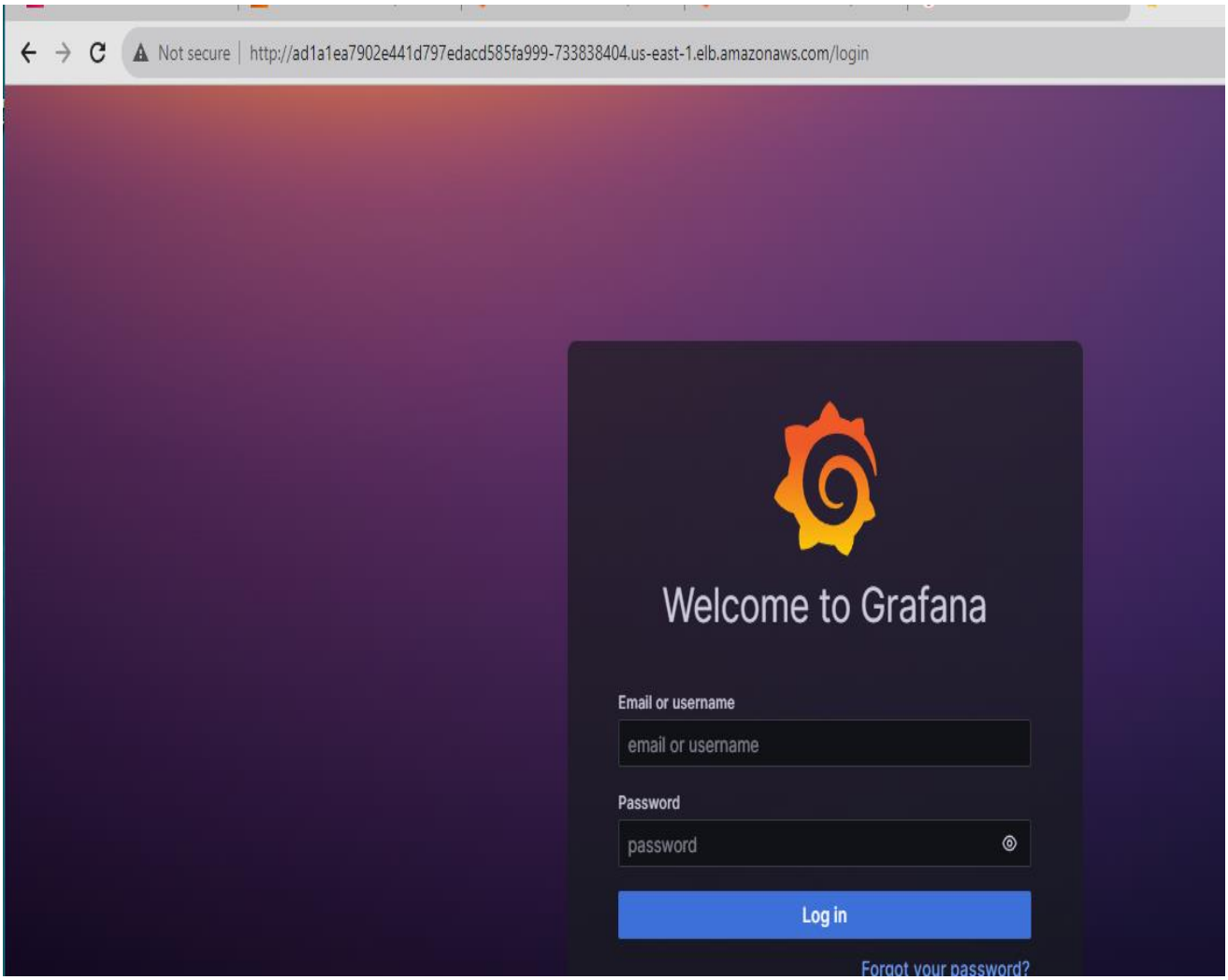
kubectl get svc -n prometheus

```
ubuntu@ip-172-31-81-42:~$ kubectl edit svc stable-grafana -n prometheus
service/stable-grafana edited
ubuntu@ip-172-31-81-42:~$ kubectl get svc -n prometheus
NAME                                     TYPE           CLUSTER-IP       EXTERNAL-IP                                                                PORT(S)
          AGE
alertmanager-operated                    ClusterIP      None             <none>                                                                     9093/TCP,9094/TCP,90
94/UDP    13m
prometheus-operated                      ClusterIP      None             <none>                                                                     9090/TCP
          13m
stable-grafana                           LoadBalancer   10.100.186.94    ad1a1ea7902e441d797edacd585fa999-733838404.us-east-1.elb.amazonaws.com     80:30905/TCP
          13m
stable-kube-prometheus-sta-alertmanager  ClusterIP      10.100.254.197   <none>                                                                     9093/TCP,8080/TCP
          13m
stable-kube-prometheus-sta-operator      ClusterIP      10.100.85.234    <none>                                                                     443/TCP
          13m
stable-kube-prometheus-sta-prometheus    LoadBalancer   10.100.229.18    a8ee684ef50a04fa99b6fa68a6a549e7-907360140.us-east-1.elb.amazonaws.com     9090:30489/TCP,8080:
30104/TCP   13m
stable-kube-state-metrics                ClusterIP      10.100.245.2     <none>                                                                     8080/TCP
          13m
stable-prometheus-node-exporter          ClusterIP      10.100.247.199   <none>                                                                     9100/TCP
          13m
ubuntu@ip-172-31-81-42:~$
```



Access Grafana UI in the browser

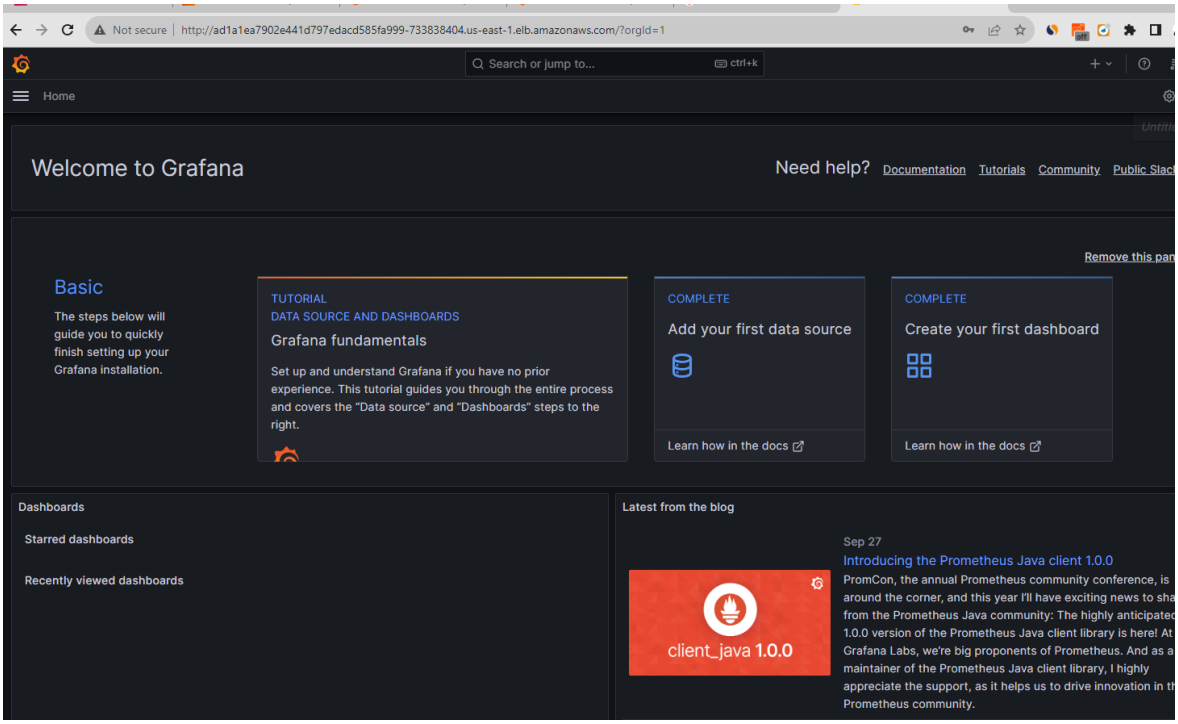
Get the URL after you run **kubectl get svc -n Prometheus** and place it in the browser.



You should see the above.

You can now login with the following details:

UserName: admin
Password: prom-operator



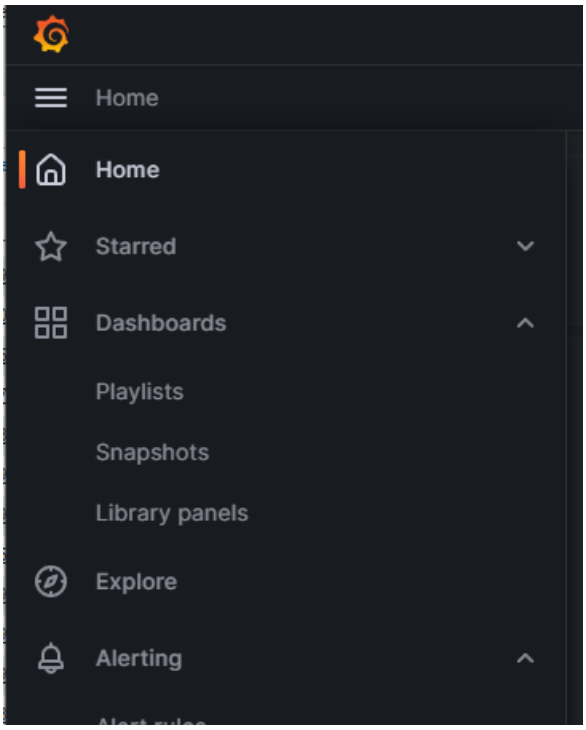
Create Dashboard in Grafana

In Grafana, we can create of various kinds of dashboards according to our needs.

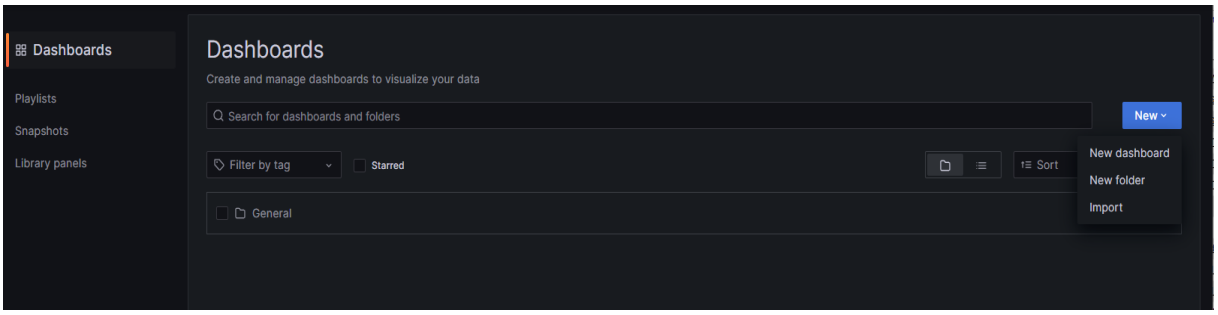
How to Create Kubernetes Monitoring Dashboard?

For creating a dashboard to monitor the cluster:

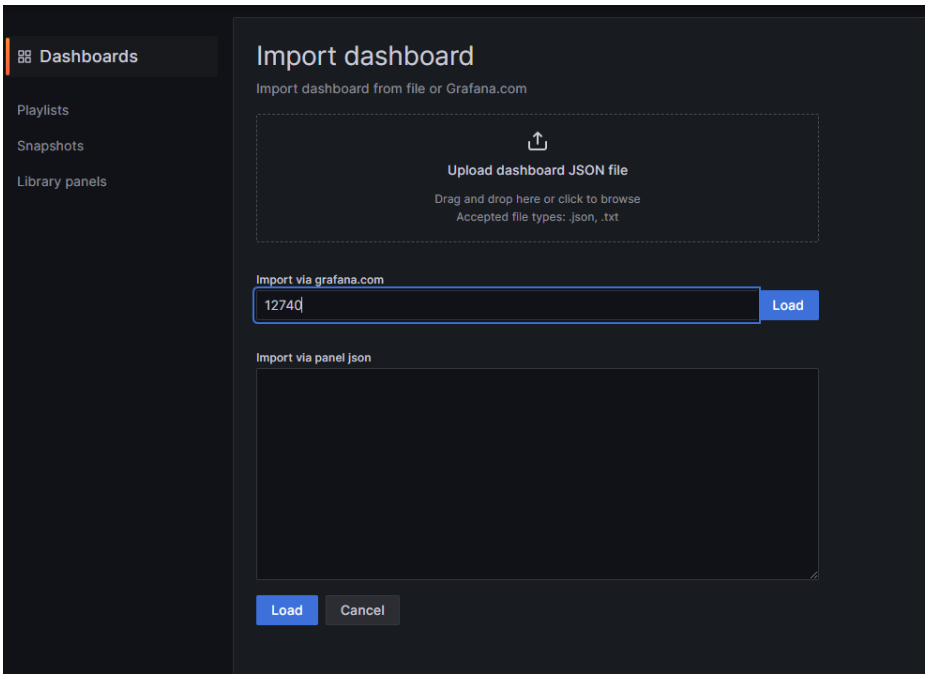
Click the hamburger at the top left than select **Dashboard**



Then on the right select **New > 'Import'**.

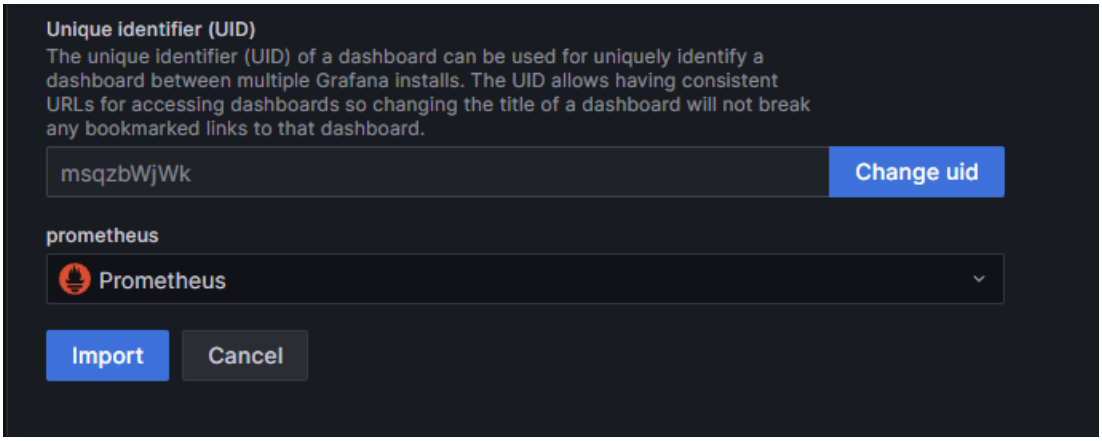


Enter **12740** dashboard id under Grafana.com Dashboard.



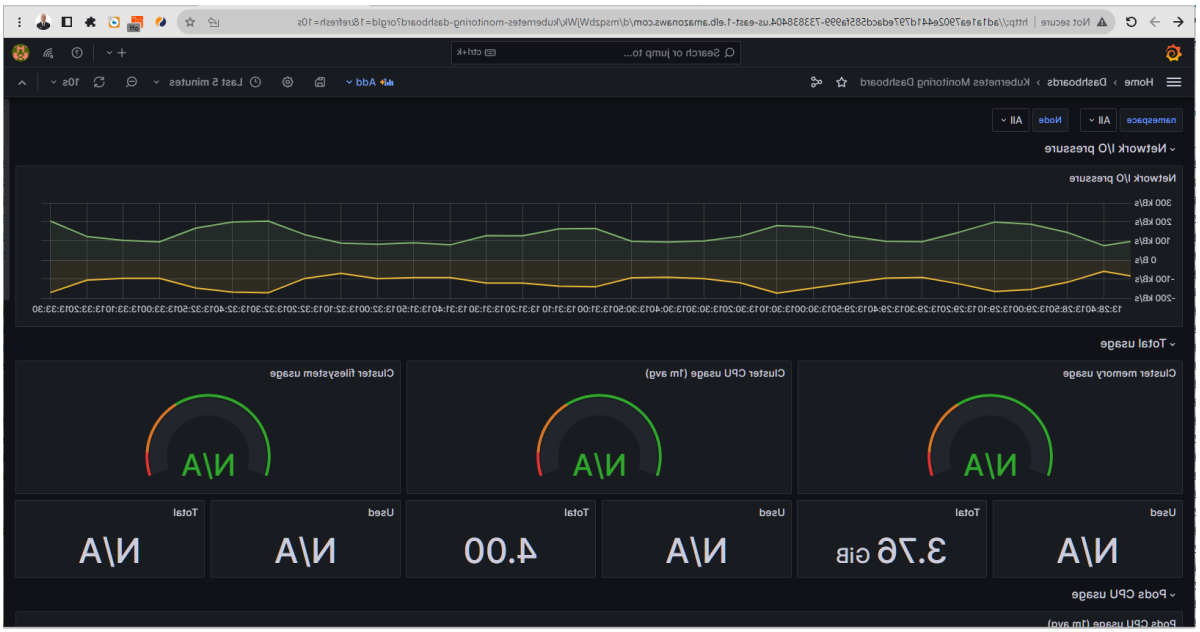
Click '**Load'**.

Select '**Prometheus'** as the endpoint under prometheus data sources drop down.



Click '**Import'**.

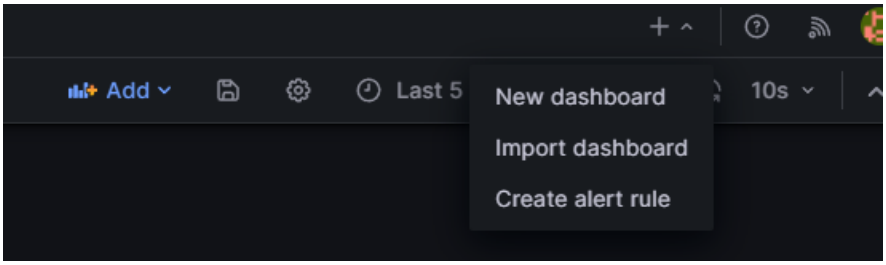
This will show monitoring dashboard for all cluster nodes.

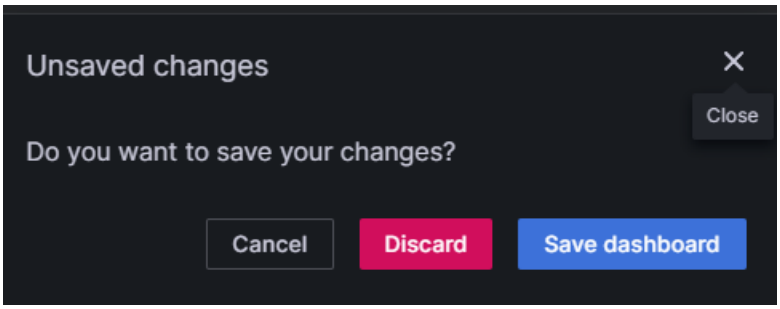


How to Create Kubernetes Cluster Monitoring Dashboard?

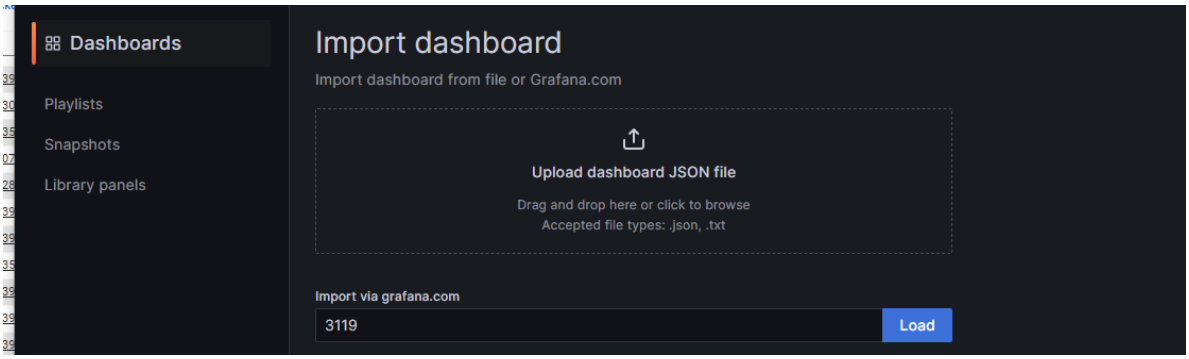
For creating a dashboard to monitor the cluster:

Click **'+'** button on the top right and select '**Import dashboard**'.



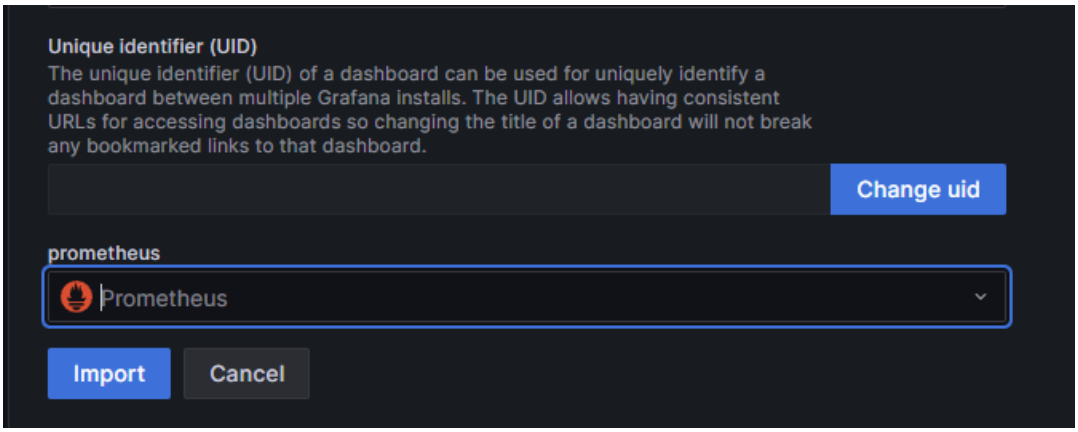


Enter **3119** dashboard id under Grafana.com Dashboard.



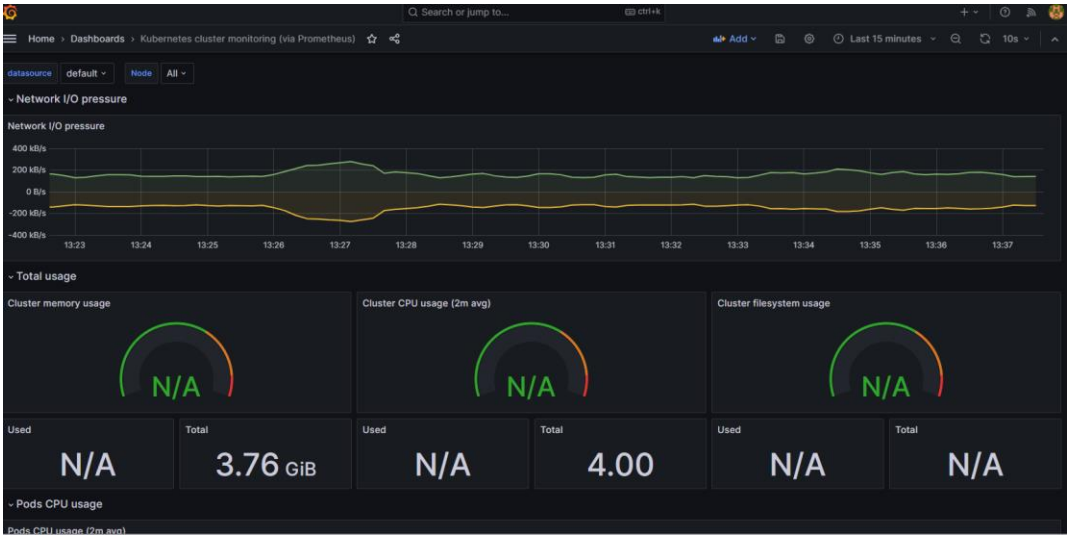
Click '**Load'**.

Select '**Prometheus'** as the endpoint under prometheus data sources drop down.



Click '**Import'**.

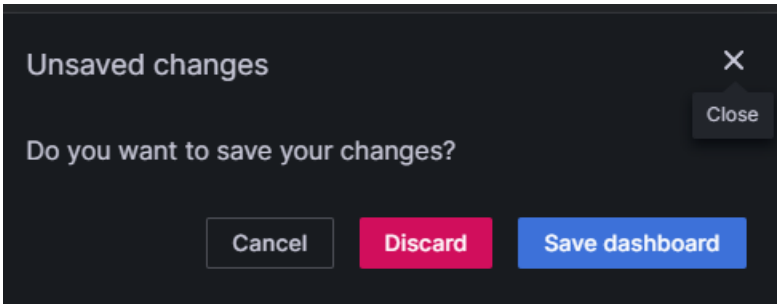
This will show a monitoring dashboard for all cluster nodes.



How to create a POD Monitoring Dashboard

For creating a dashboard to monitor the cluster:

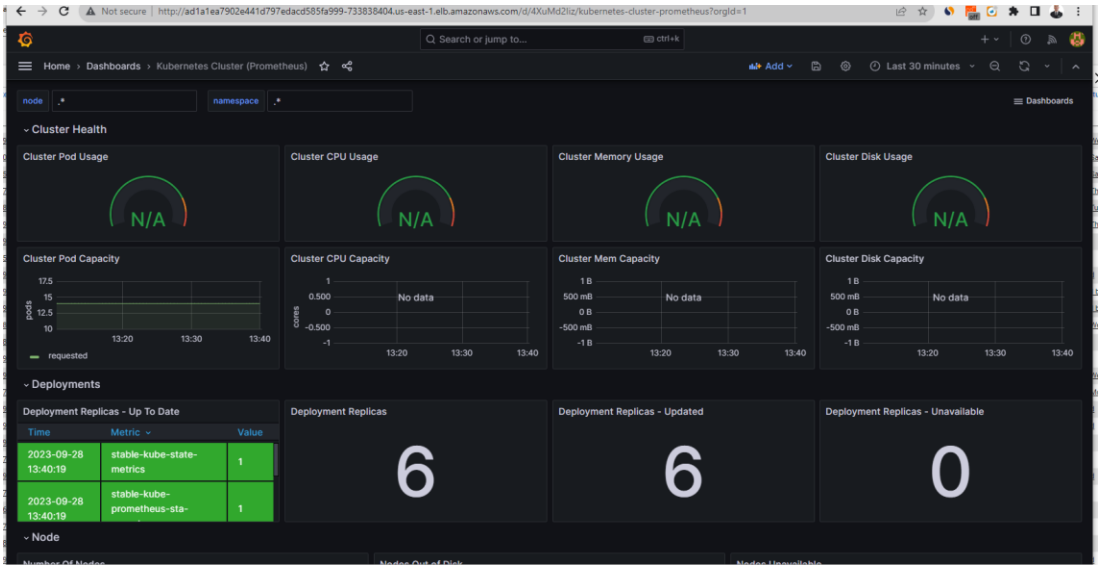
Click '**+**' button on the top right panel and select '**Import dashboard**'.



Enter **6417** dashboard id under Grafana.com **Dashboard**.
Click '**Load'**.

Select '**Prometheus'** as the endpoint under prometheus data sources drop down.
Click 'Import'.

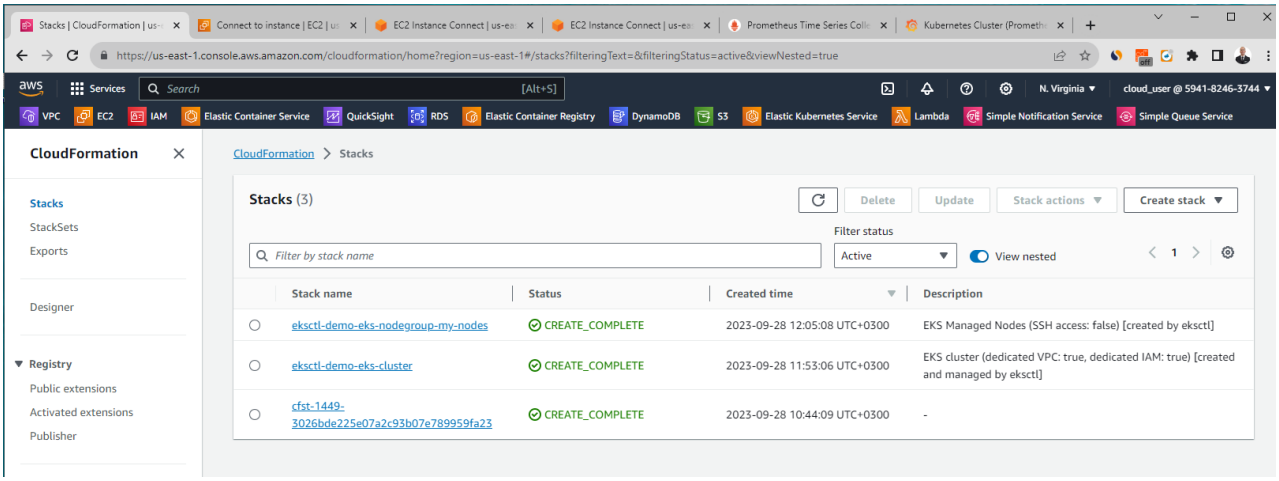
This will show a monitoring dashboard for all cluster nodes.



Cleanup EKS Cluster
Use the below command to delete EKS cluster to avoid being charged by AWS.

eksctl delete cluster --name demo-eks --region us-east-1

Alternatively, login to AWS console --> AWS Cloudformation --> delete the stack manually.



You can also delete the cluster under AWS console --> Elastic Kubernetes Service --> Clusters Click on Delete cluster.