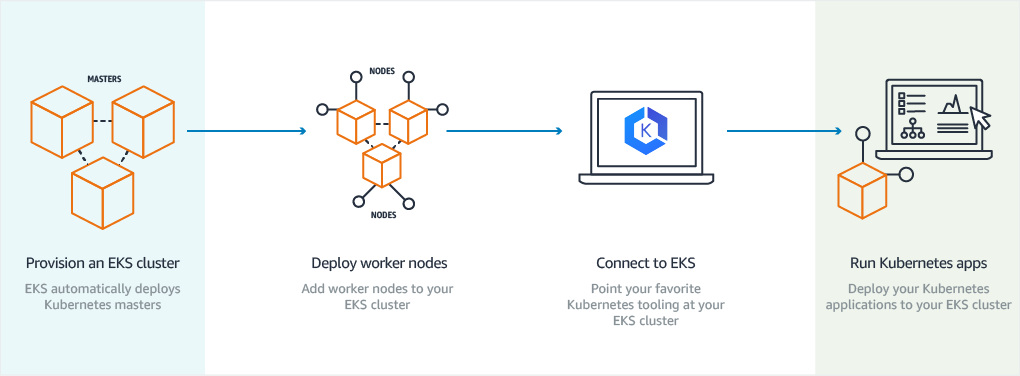
**Deployed WordPress and MySql on the top of EKS Cluster.**

In this task we are creating a multi-node K8S cluster on the top of AWS cloud using Elastic Kubernetes Service (EKS) and deploying WordPress which is an open-source content management system for publishing web content using MySQL database. We will use AWS ELB (Elastic Load Balancing), EBS (Elastic Block Store) and EFS (Elastic File System) for load balancing and storage management.



* **Manage Control Plane:** Amazon EKS provides a scalable and highly-available control plane that runs across multiple AWS availability zones. The Amazon EKS service automatically manages the availability and scalability of the Kubernetes API servers and the etcd persistence layer for each cluster. Amazon EKS runs the Kubernetes control plane across three Availability Zones in order to ensure high availability, and it automatically detects and replaces unhealthy masters.
* **Manage Worker Nodes:**Amazon EKS lets you create, update, or terminate worker nodes for your cluster with a single command. Managed node groups run nodes using the latest EKS-optimized AMIs in your AWS account while updates and terminations gracefully drain nodes to ensure your applications stay available.
* **Load Balancing:**Amazon EKS supports Elastic Load Balancer including Application Load Balancer (ALB), Network Load Balancer (NLB), and Classic Load Balancer.
* **ServerLess Compute:** EKS supports AWS Fargate to run your Kubernetes applications using serverless compute. Fargate removes the need to provision and manage servers, lets us specify and pay for resources per application, and improves security through application isolation by design.

## Prerequisite

* AWS account
* AWS CLI installed
* eksctl installed
* kubectl installed
* *HELM*
* *Tiller*

**AWS CLI:**The AWS Command Line Interface (CLI) is a unified tool to manage your AWS services.

***eksctl:****eksctl is a simple CLI tool for creating clusters on EKS - Amazon's new managed Kubernetes service for EC2.*

***kubectl:****We require kubectl command as a client to connect with Kubernetes master from our local system.*

***HELM:****HELM is like a package installer as in RedHat Linux we use yum or dnf to install packages. In Kubernetes, packages are known as Charts. HELM is used as a client-side command to install packages on Kubernetes Cluster.*

***Tiller:****Tiller act as server-side from where HELM is installing packages/charts.*

***We have to download all the software like eksctl, kubectl, HELM, and Tiller and put all this in one file and add this directory to our system variable path.***

For connecting to AWS EKS, we have the following ways such as WebUI, CLI and terraform code also. Here we are using CLI , but for using AWS from CLI we required secret key and access key of AWS, IAM user.

**STEP 1: AWS Configure**

>>aws configure

**STEP 2: Creating Cluster**

**code to create cluster**

apiVersion: eksctl.io/v1alpha5

kind: ClusterConfig

metadata:

  name: abdcluster

  region: ap-south-1

nodeGroups:

   - name: ng1

     desiredCapacity: 2

     instanceType: t2.micro

     ssh:

        publicKeyName: abdkey

   - name: ng2

     desiredCapacity: 1

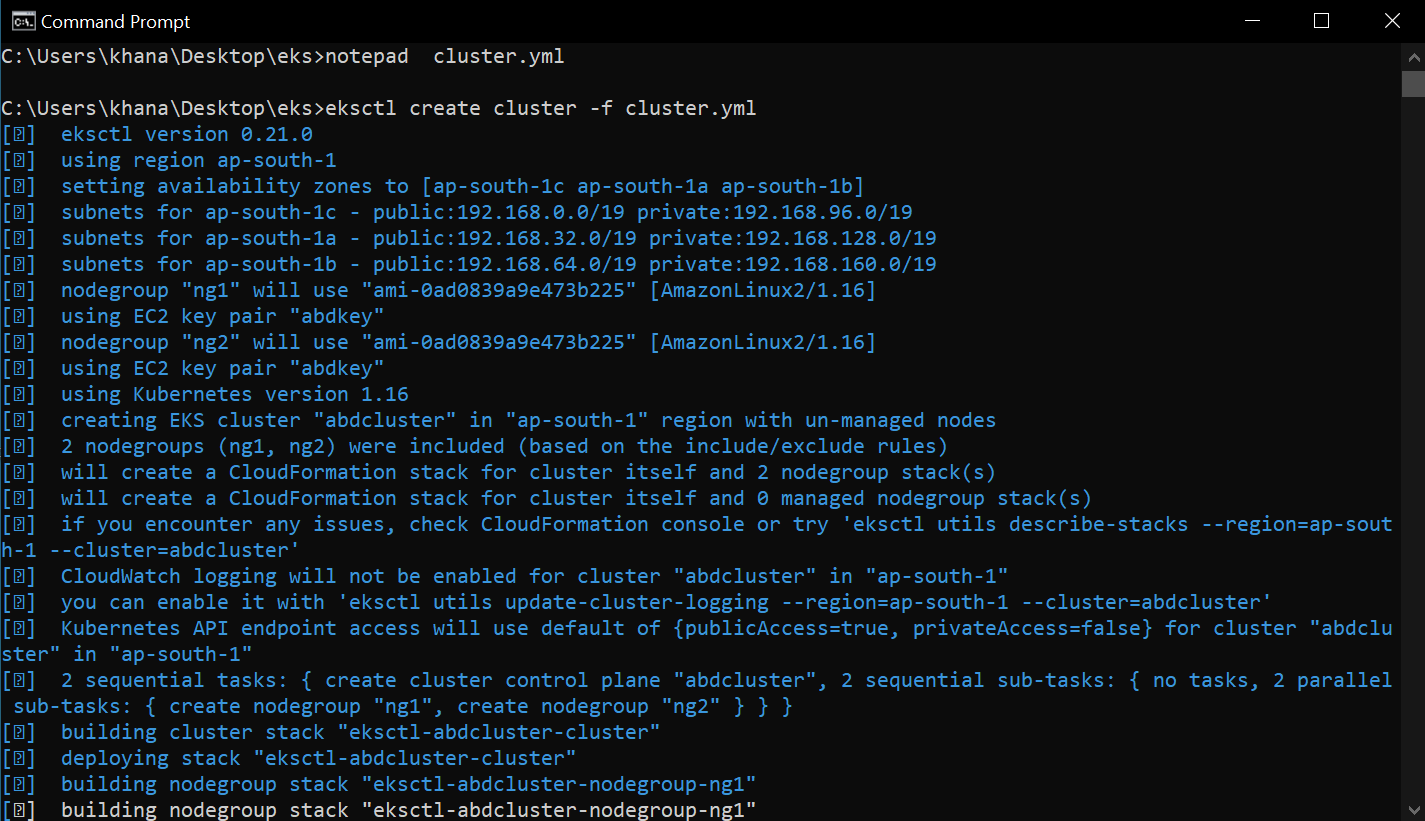
     instanceType: t2.small

     ssh:

        publicKeyName: abdkey

**Below is the command to create a cluster or to run the above file.**

**eksctl create cluster -f cluster.yml**

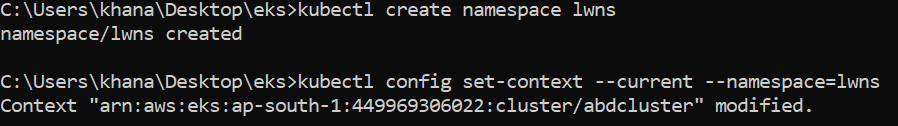


We have created a yml file to launch a cluster having 2 node groups. One of them having one EC2-instance of the type t2.small and the other with two EC2-instances of type t2.micro. One thing to note here is you must decide the desired capacity and instance type carefully because there are limited network interfaces that can be added to an instance type. And thus there is a limited number of pods that can be launched on a slave node/instance (out of which some pods for eg. core DNS, will be deployed at the time of launch of the cluster).Also add a key pair to the ec2 instance.

**STEP 3: Update kubeconfig and created a new namespace**

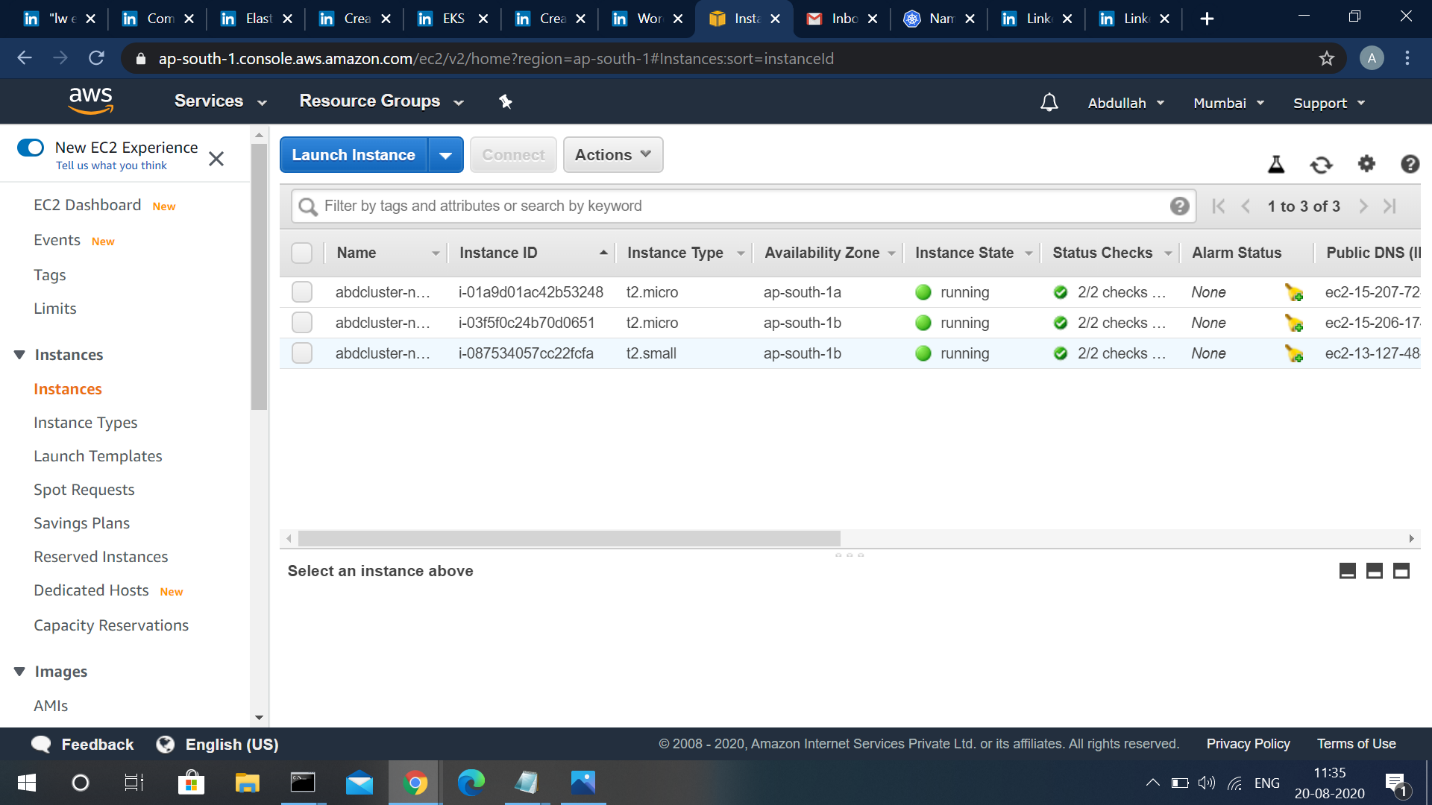
A namespace is a way to group services for an application. So, a separate workspace is being created named lwns

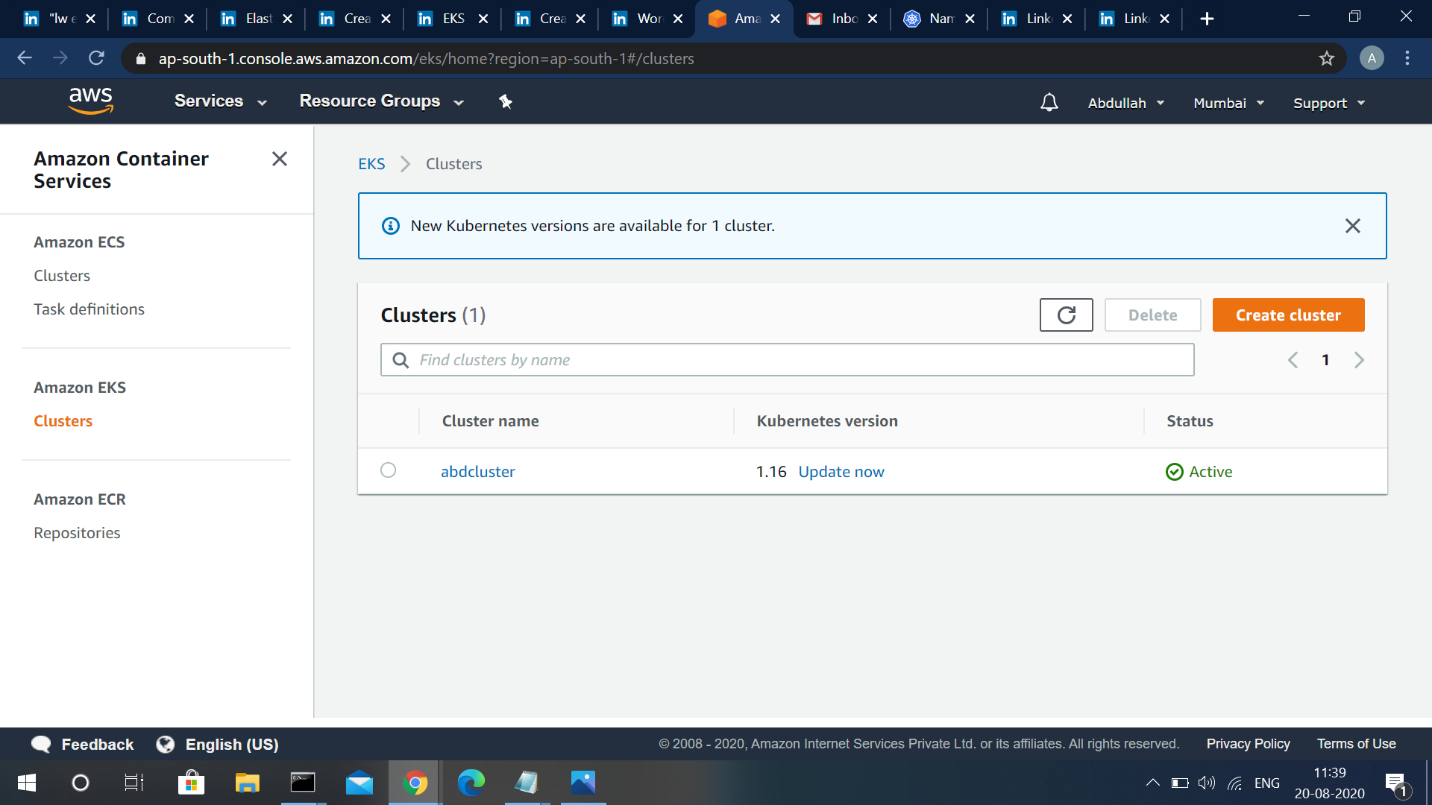
No alt text provided for this image

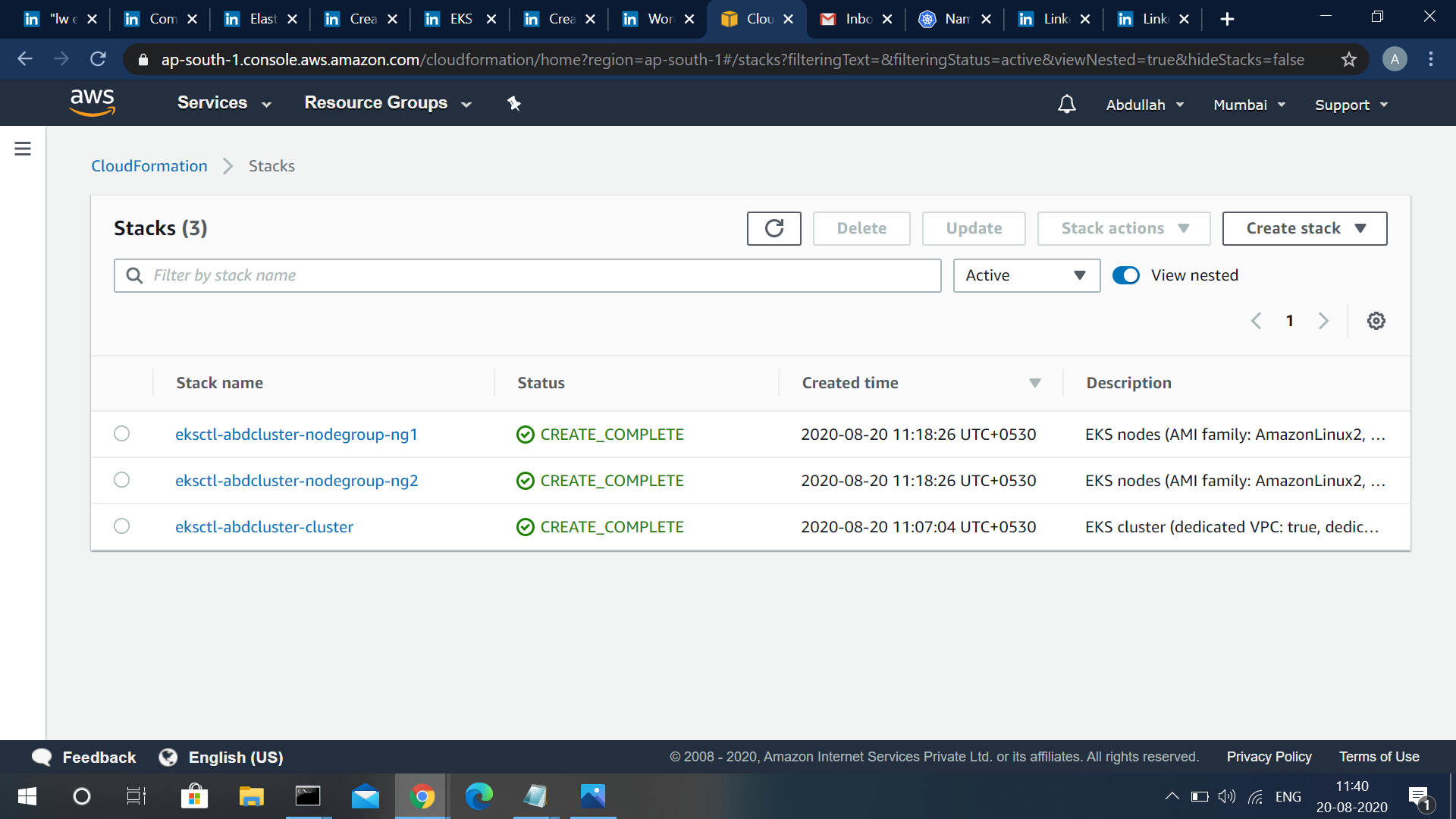


Now, this cluster can be managed by Kubectl commands as well. create, delete, deploy we can do all this by Kubeclt cmd .

We can check what ever cluster we instances we have created on AWS by GUI(which provides a simple way to create and manage different AWS resources) or also CLI.







**STEP 3: Create PVC**

A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator. It is a resource in the cluster just like a node is a cluster resource.

*Code to create PVC*

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

  name: pv-volume

spec:

  storageClassName: lwns

  accessModes:

    - ReadWriteOnce

  resources:

    requests:

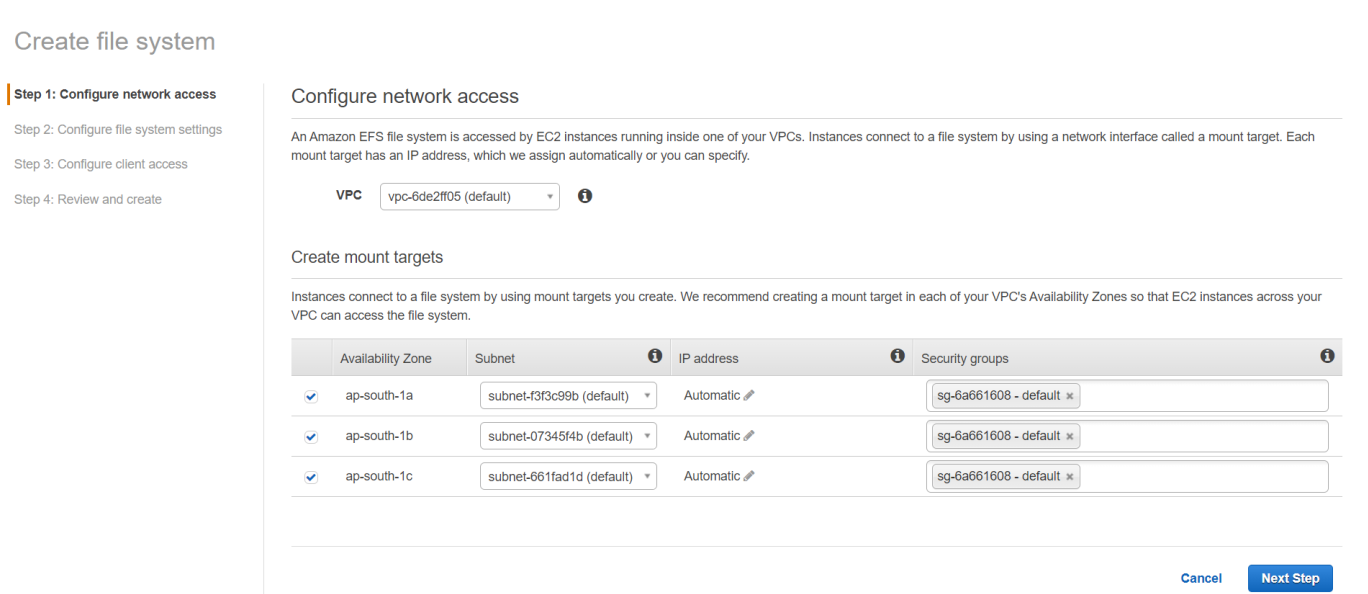
  storage: 10Gi

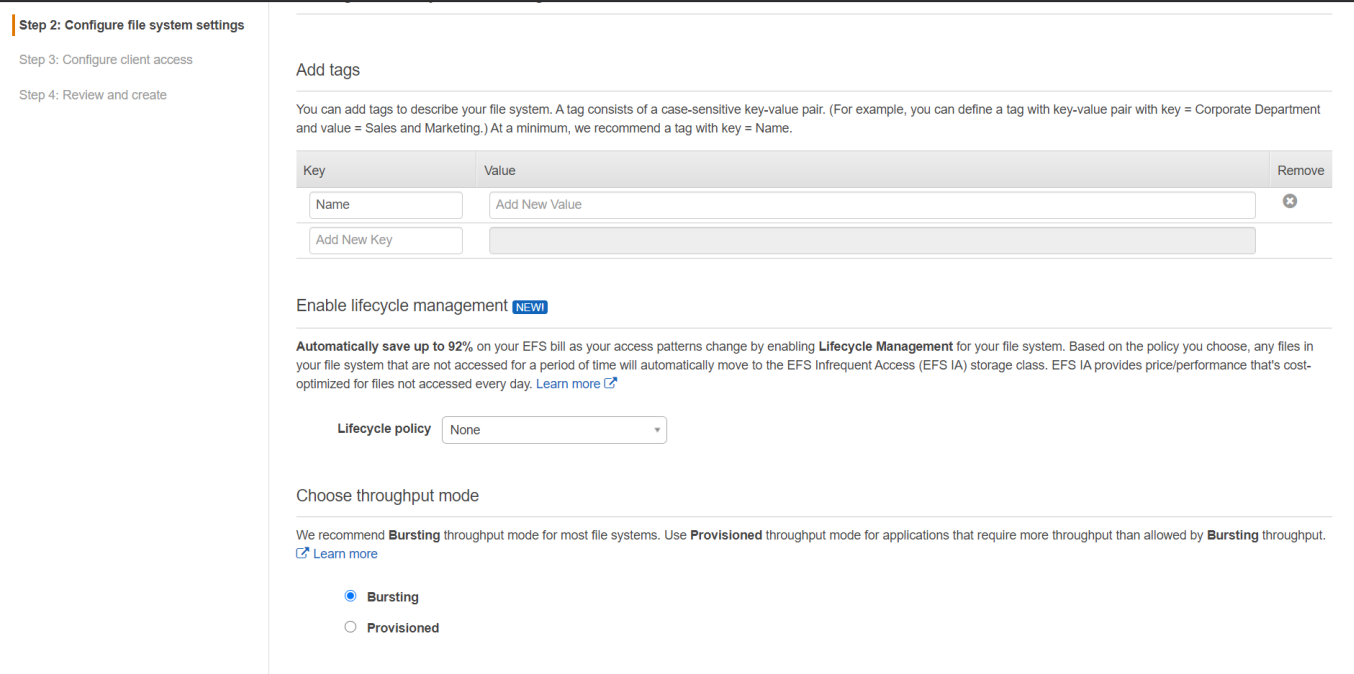
cmd to run the code

kubectl create -f pvc.yml

**STEP 5: Create File System and EFS**

Now when our cluster been set-up, we can create a file system using Amazon EFS, so that we can maintain consistency and make the data of WordPress and Database(MySQL) persistent.





Now we have created the EFS file system through GUI,We are going to connect it to the PVC. Following is the code for it:

kind: Deployment

apiVersion: apps/v1

metadata:

  name: efs-provisioner

spec:

  selector:

    matchLabels:

      app: efs-provisioner

  replicas: 1

  strategy:

    type: Recreate

  template:

    metadata:

      labels:

        app: efs-provisioner

    spec:

      containers:

        - name: efs-provisioner

          image: quay.io/external\_storage/efs-provisioner:v0.1.0

          env:

            - name: FILE\_SYSTEM\_ID

              value: fs-c2e44e76

            - name: AWS\_REGION

              value: ap-south-1

            - name: PROVISIONER\_NAME

              value: lw-course/aws-efs

          volumeMounts:

            - name: pv-volume

              mountPath: /persistentvolumes

      volumes:

        - name: pv-volume

          nfs:

            server: fs-c2e44e76.efs.ap-south-1.amazonaws.com

            path: /

STEP 6: Grant Permision-BRAC

Role-based access control (RBAC) is a method of restricting network access based on the roles of individual users within an enterprise. RBAC lets employees have access rights only to the information they need to do their jobs and prevents them from accessing information that doesn't pertain to them.

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRoleBinding

metadata:

  name: nfs-provisioner-role-binding

subjects:

  - kind: ServiceAccount

    name: default

    namespace: lwns

roleRef:

  kind: ClusterRole

  name: cluster-admin

  apiGroup: rbac.authorization.k8s.io

**STEP 7: Creating a Storage Class**

We need to create a storage class from where PV get the storage for PVC.

kind: StorageClass

apiVersion: storage.k8s.io/v1

metadata:

  name: aws-efs

provisioner: lw-course/aws-efs

---

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

  name: efs-wordpress

  annotations:

    volume.beta.kubernetes.io/storage-class: "aws-efs"

spec:

  accessModes:

    - ReadWriteMany

  resources:

    requests:

      storage: 10Gi

---

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

  name: efs-mysql

  annotations:

    volume.beta.kubernetes.io/storage-class: "aws-efs"

spec:

  accessModes:

    - ReadWriteMany

  resources:

    requests:

      storage: 10Gi

**STEP 8: MySql and WordPress Pods**

MySQL is a database management system that is used by WordPress to store and retrieve all your blog information. WordPress is the most popular way to create a blog or website. Here, we use it as a front-end.

code for Mysql pod

apiVersion: v1

kind: Service

metadata:

  name: wordpress-mysql

  labels:

    app: wordpress

spec:

  ports:

    - port: 3306

  selector:

    app: wordpress

    tier: mysql

  clusterIP: None

---

apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2

kind: Deployment

metadata:

  name: wordpress-mysql

  labels:

    app: wordpress

spec:

  selector:

    matchLabels:

      app: wordpress

      tier: mysql

  strategy:

    type: Recreate

  template:

    metadata:

      labels:

        app: wordpress

        tier: mysql

    spec:

      containers:

      - image: mysql:5.6

        name: mysql

        env:

        - name: MYSQL\_ROOT\_PASSWORD

          valueFrom:

            secretKeyRef:

              name: mysql-pass

              key: password

        ports:

        - containerPort: 3306

          name: mysql

        volumeMounts:

        - name: mysql-persistent-storage

          mountPath: /var/lib/mysql

      volumes:

      - name: mysql-persistent-storage

        persistentVolumeClaim:

          claimName: efs-mysql

code for WordPress pod

apiVersion: v1

kind: Service

metadata:

  name: wordpress

  labels:

    app: wordpress

spec:

  ports:

    - port: 80

  selector:

    app: wordpress

    tier: frontend

  type: LoadBalancer

---

apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2

kind: Deployment

metadata:

  name: wordpress

  labels:

    app: wordpress

spec:

  selector:

    matchLabels:

      app: wordpress

      tier: frontend

  strategy:

    type: Recreate

  template:

    metadata:

      labels:

        app: wordpress

        tier: frontend

    spec:

      containers:

      - image: wordpress:4.8-apache

        name: wordpress

        env:

        - name: WORDPRESS\_DB\_HOST

          value: wordpress-mysql

        - name: WORDPRESS\_DB\_PASSWORD

          valueFrom:

            secretKeyRef:

              name: mysql-pass

              key: password

        ports:

        - containerPort: 80

          name: wordpress

        volumeMounts:

        - name: wordpress-persistent-storage

          mountPath: /var/www/html

      volumes:

      - name: wordpress-persistent-storage

        persistentVolumeClaim:

          claimName: efs-wordpress

STEP 9: DEPLOY

Now we can automate all the above step through a singe, kustomization file. here we can execute all the above code in a manner that we want, through a single line of cmd(kubectl create -k .).

apiVersion: kustomize.config.k8s.io/v1beta1

kind: Kustomization

secretGenerator:

- name: mysql-pass

  literals:

  - password=redhat

resources:

  - create-efs-provisioner.yaml

  - create-rbac.yaml

  - create-storage.yaml

  - deploy-mysql.yaml

  - deploy-wordpress.yaml

cmd to run this file

kubectl create -k .

and finally we can accesses our site

