CLOUD TRAIN

ACCELERATE YOUR GROWTH

MODULE 10

DEVOPS ON AWS

AWS Workshop

Contact us

TO ACCELERATE YOUR CAREER GROWTH

For questions and more details:

please call @ +91 98712 72900, or

visit https://www.thecloudtrain.com/, or

email at support@thecloudtrain.com, or

WhatsApp us @ +91 98712 72900

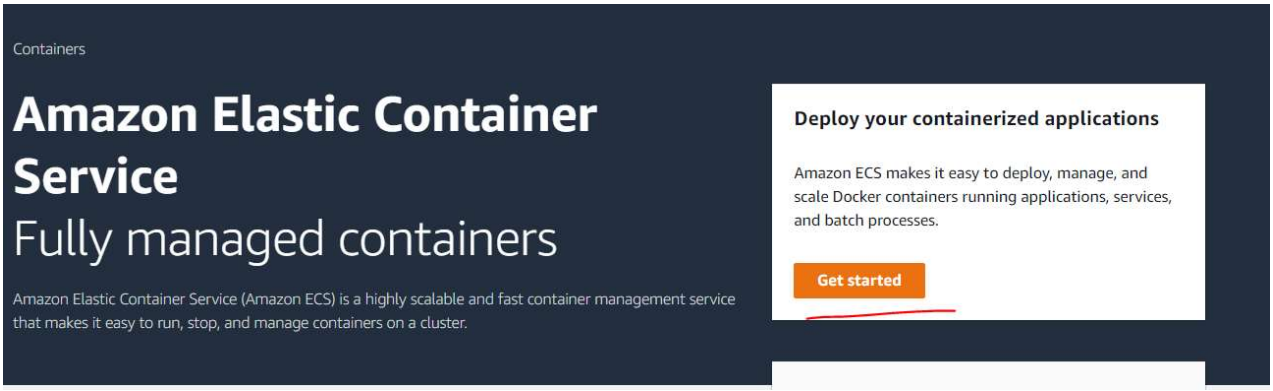
CONTAINERIZATION IN AWS

Deploy a containerized application in ECS

Set up your first run with Amazon ECS Using Fargate

The Amazon ECS first run wizard will guide you through creating a cluster and launching a sample web application. In this step, you will enter the Amazon ECS console and launch the wizard.

Step 1. Open https://console.aws.amazon.com/ecs/home#/firstRun?p=gsrc&c=ho_ddc for ECS home page



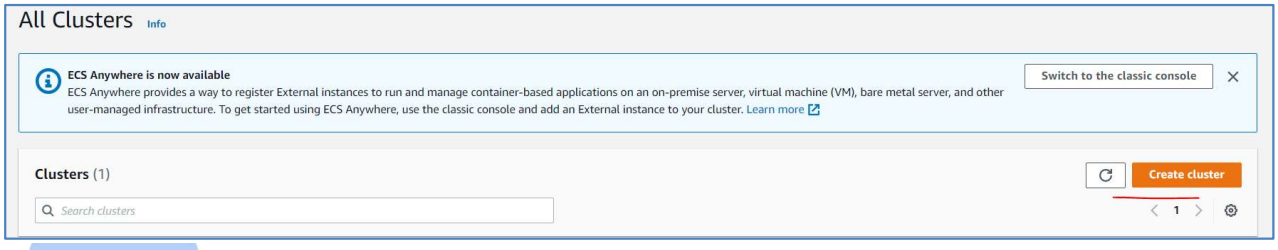
Create a Fargate Cluster.

Let's return to the AWS management console for this step.

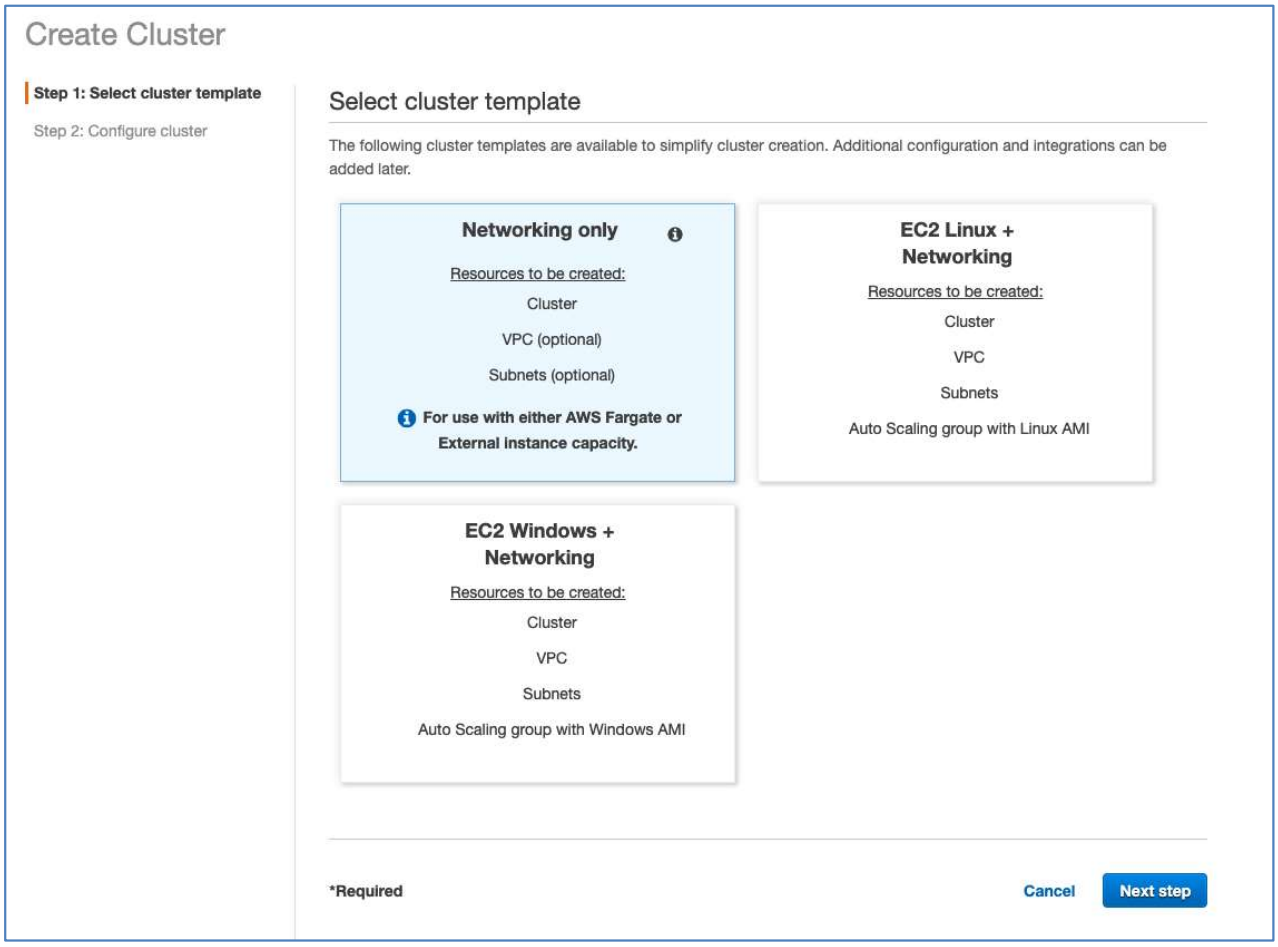
Step 1. Search for **Elastic Container Service** and select **Elastic Container Service**.

Step 2. From the left menu select **Clusters**

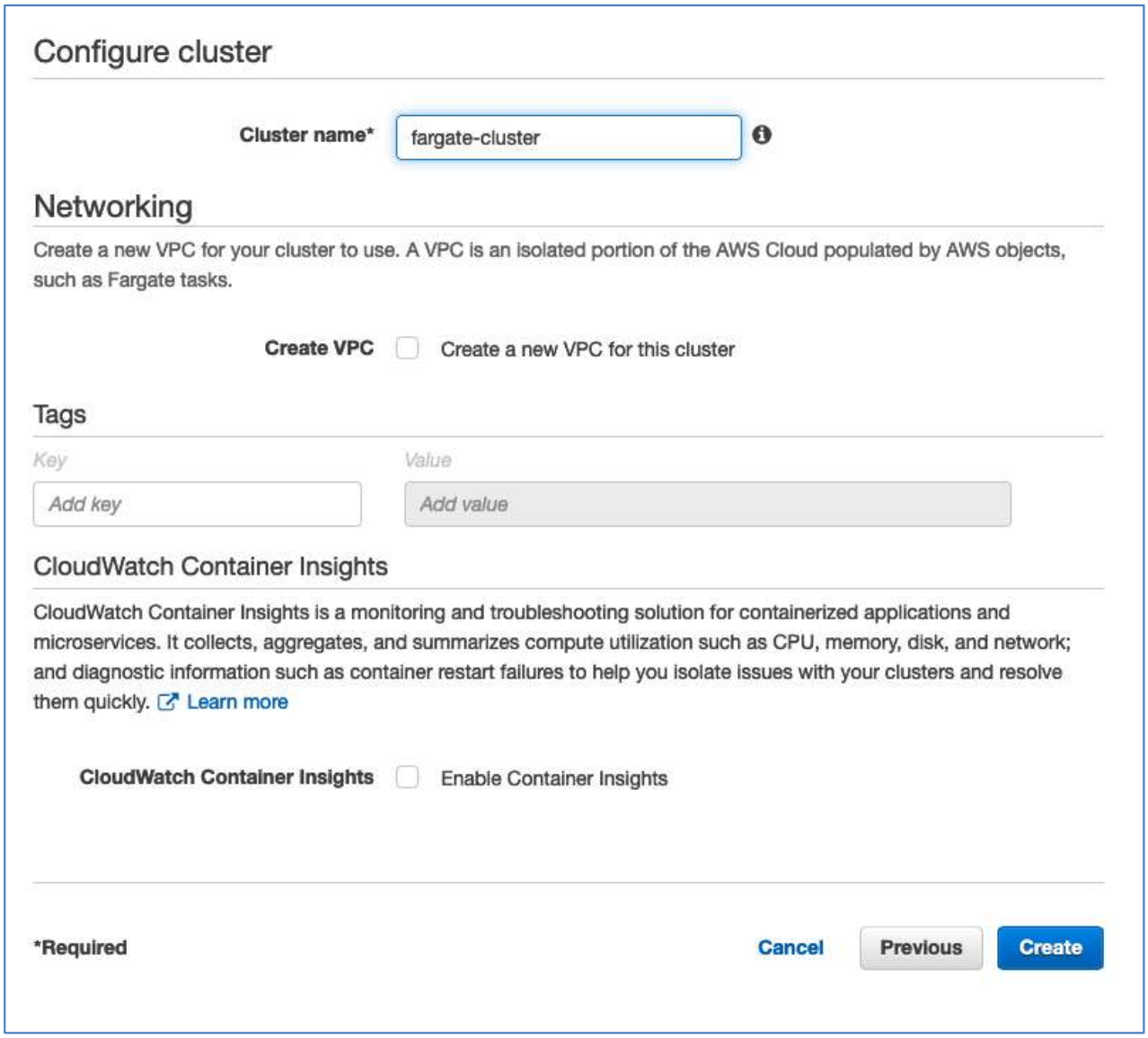
Step 3. Select **Create cluster**



Step 4. Under **Select cluster template** we are going to select networking only. We don't need an ec2 instances in our cluster because Fargate will take care of spinning the up when we start our task, and spinning them down when we stop our task.



Step 5. Let's name the cluster **fargate-cluster**, and the rest we can leave as is.



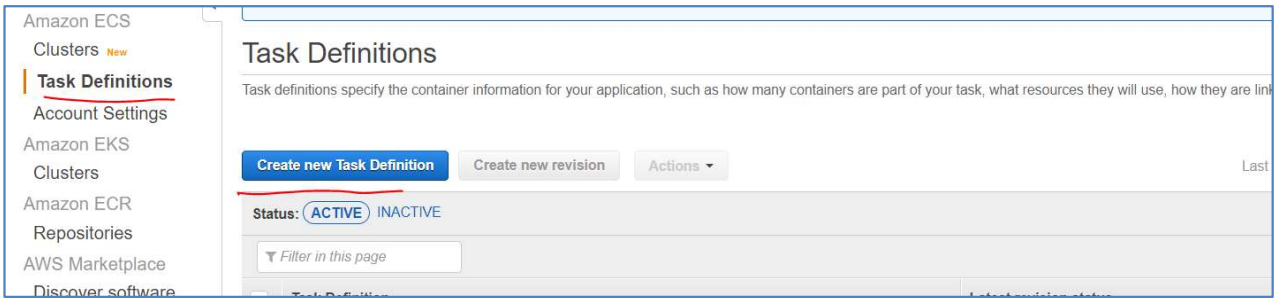
Step 6. Select **Create**

Step 7. Click **View Cluster**.

Create an ECS Task

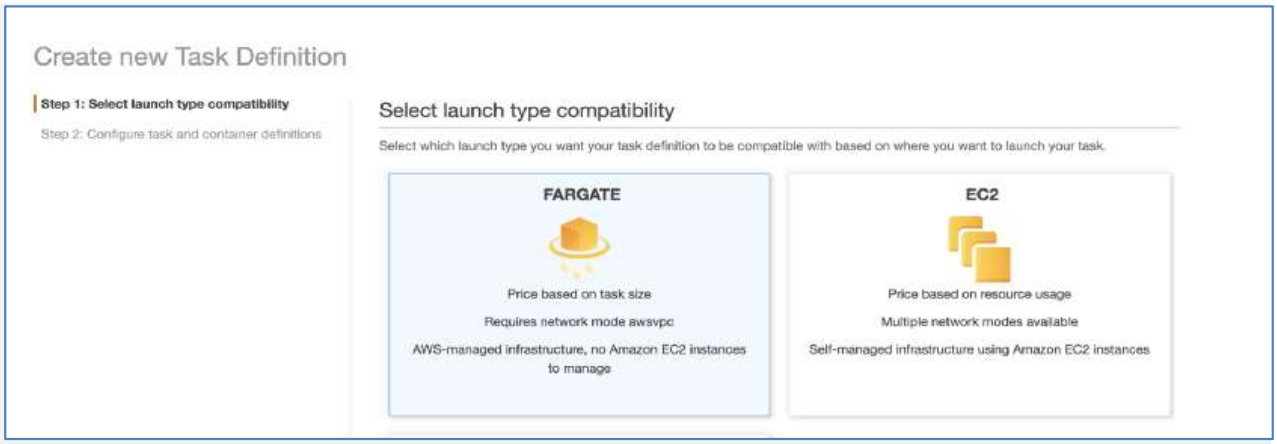
The ECS Task is the action that takes our image and deploys it to a container. To create an ECS Task lets go back to the ECS page and do the following:

Step 1. Select **Task Definitions** from the left menu. Then select **Create new Task Definition**



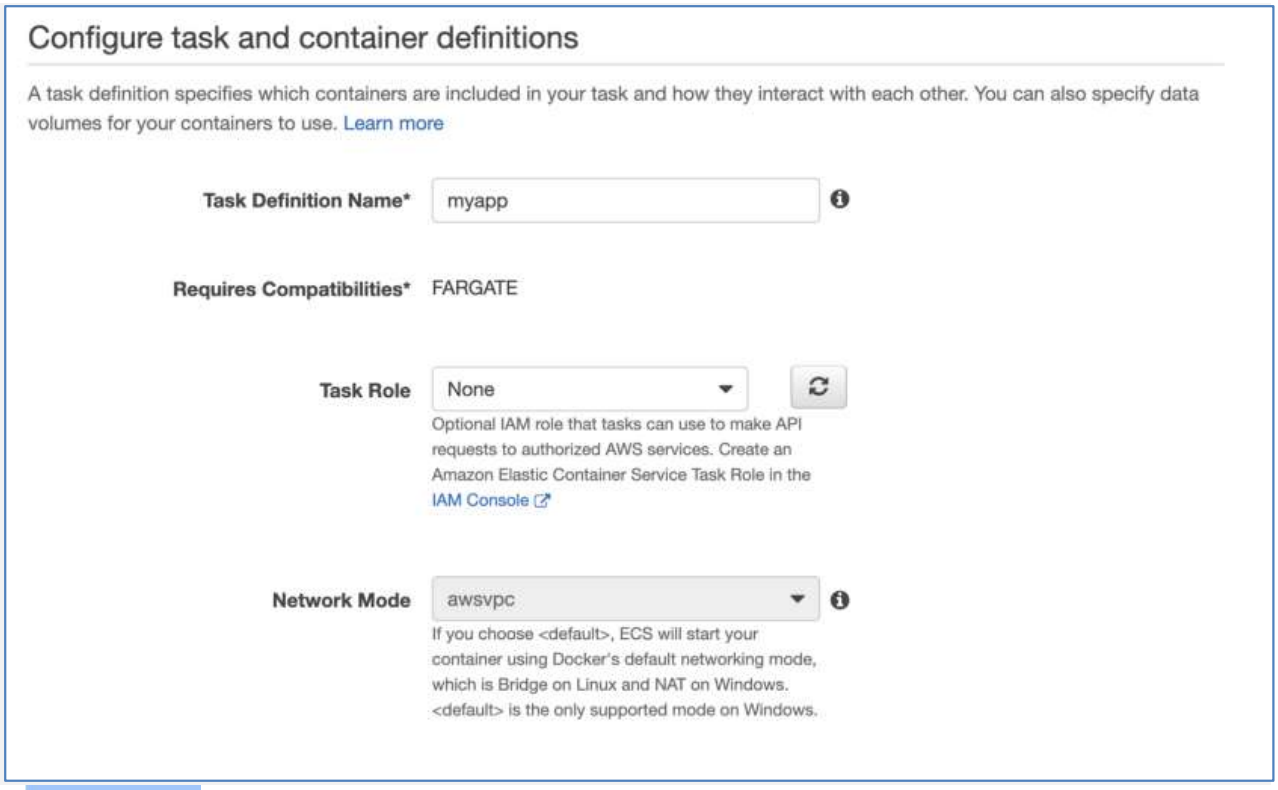
Step 2. Select **Fargate**

Step 3. Select **Next Step**



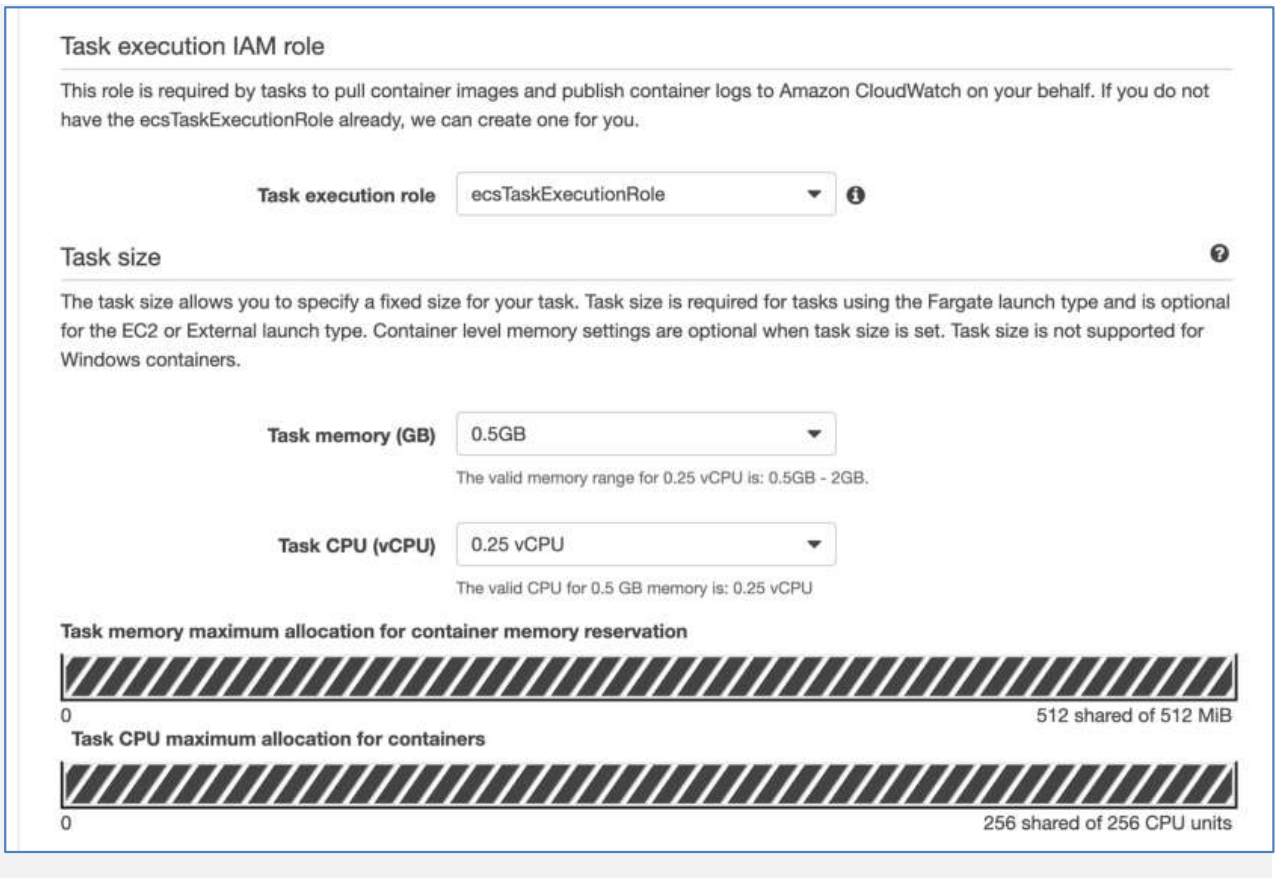
Step 4. Enter a name for the task. I am going to use **myapp**.

Step 5. Leave **Task Role** and **Network Mode** set to their default values.

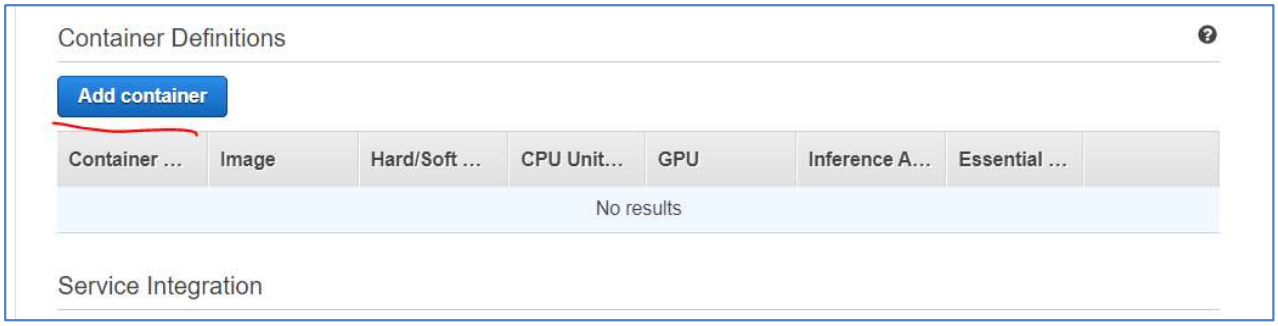


Step 6. Leave **Task Execution Role** set to its default.

Step 7. For Task memory and Task CPU select the minimum values. We only need minimal resources for this test.



Step 8. Under Container definition select Add Container.

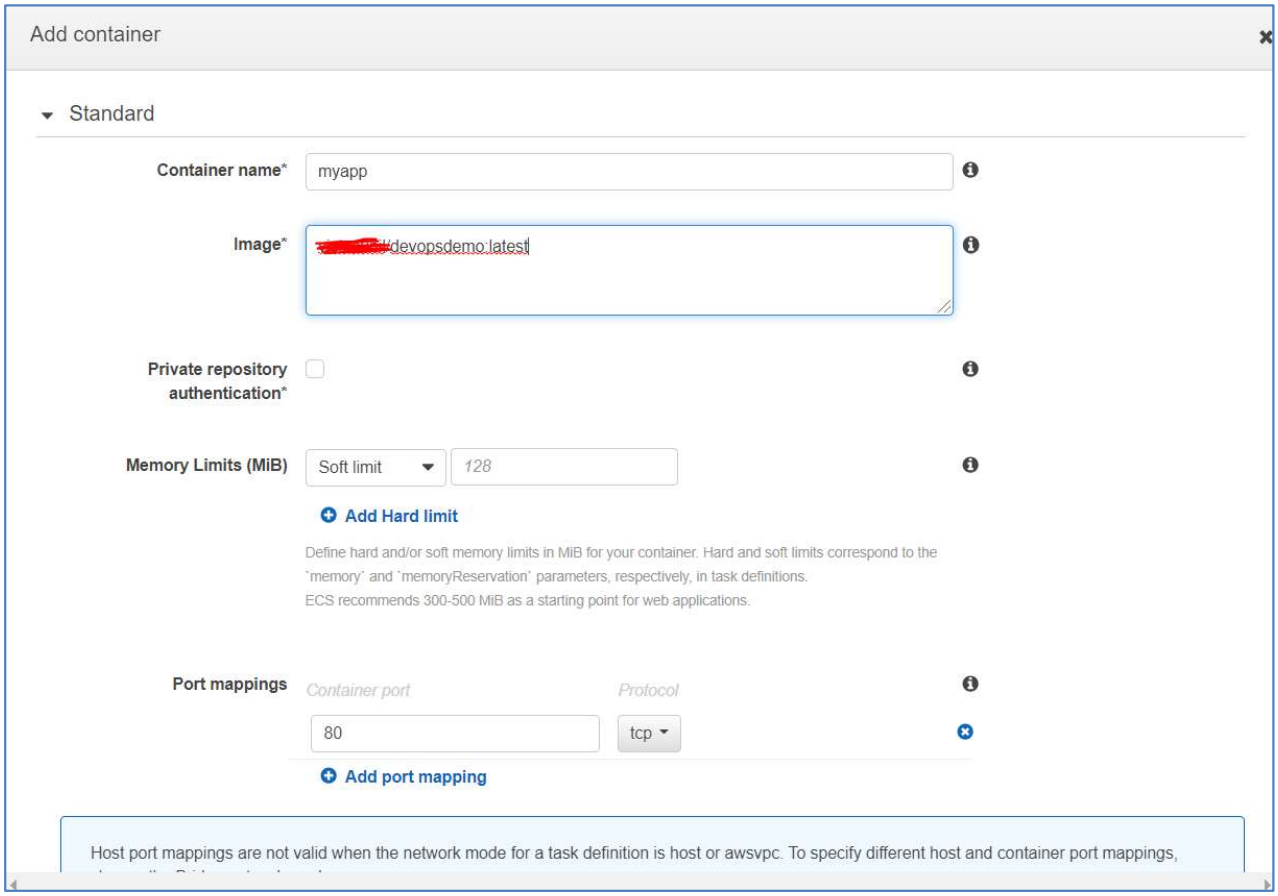


Step 9. Enter a Container name. I will use **myapp** again.

Step 10. In the Image box enter the ARN of our image. You will want to copy and paste this from the ECR dashboard if you haven't already or use any docker image URL from docker hub, say **vistasunil/devopsdemo:tagname**

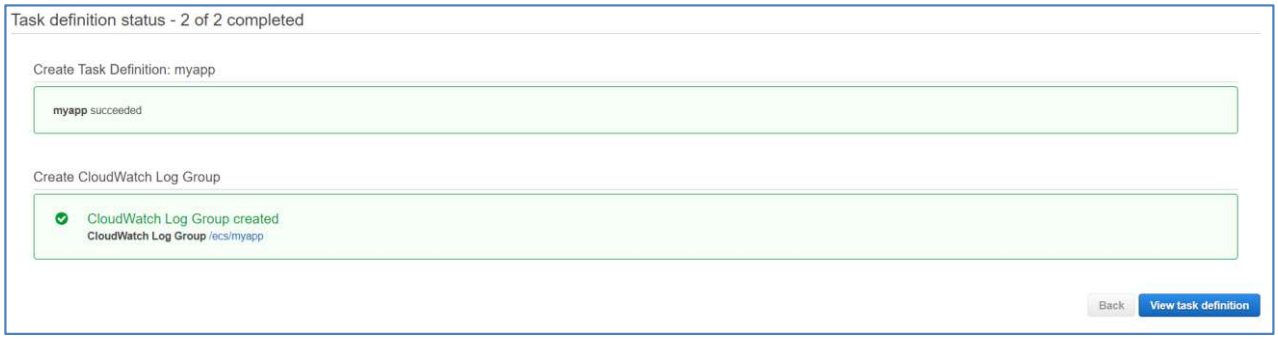
Step 11. We can keep the Memory Limit to 128Mb

Step 12. In port mappings you will notice that we can't actually map anything. Whatever port we enter here will be opened on the instance and will map to the same port on container. We will use 80 because that is where our demo app listens.

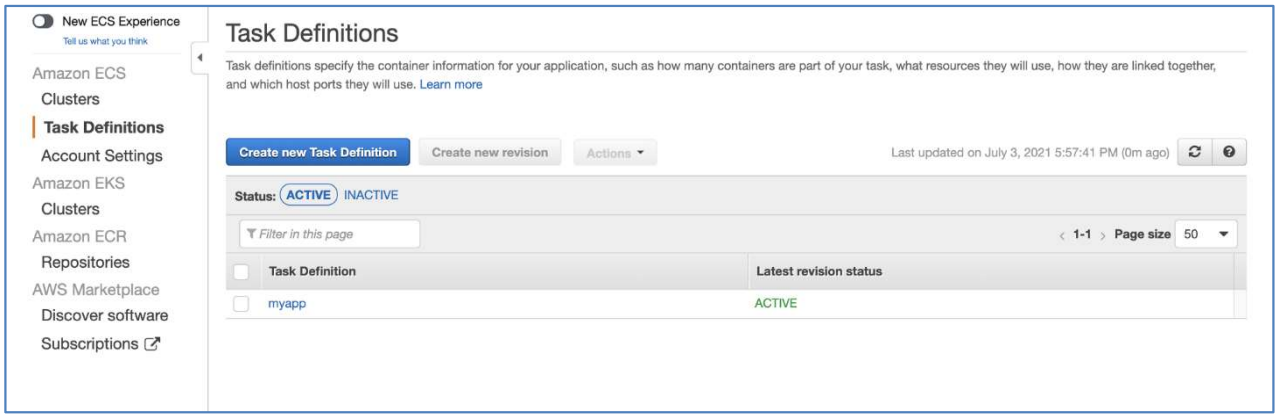


Step 13. Leave everything else set to its default value and click **Add** in the lower left corner of the dialog.

Step 14. Leave everything else in the Configure task and container definitions page as is and select **Create** in the lower right corner of the page.



Step 15. Go back to the ECS page, select Task Definitions and we should see our new task with a status of ACTIVE.

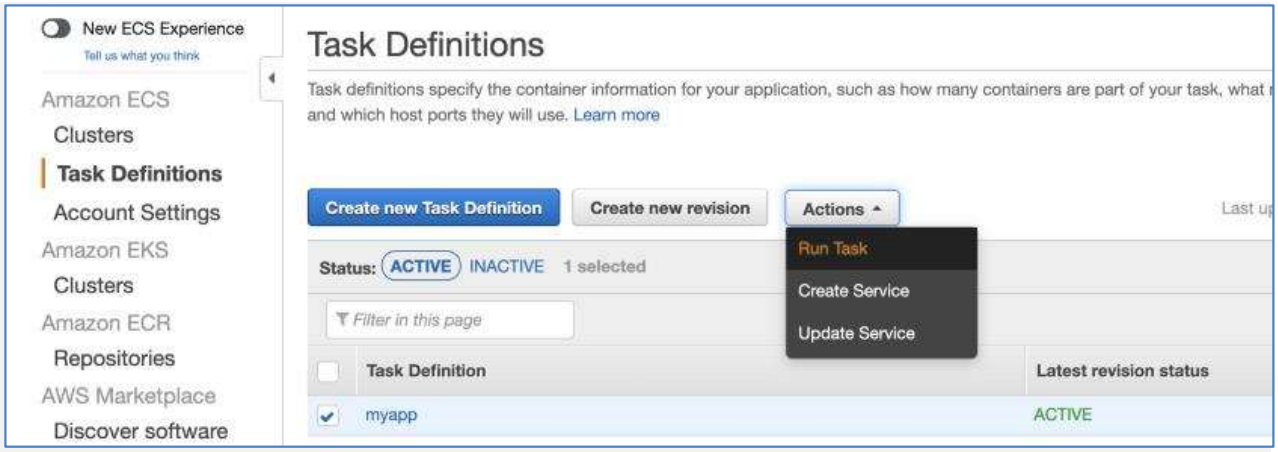


Run the ECS Task

This is the moment we have all been waiting for.

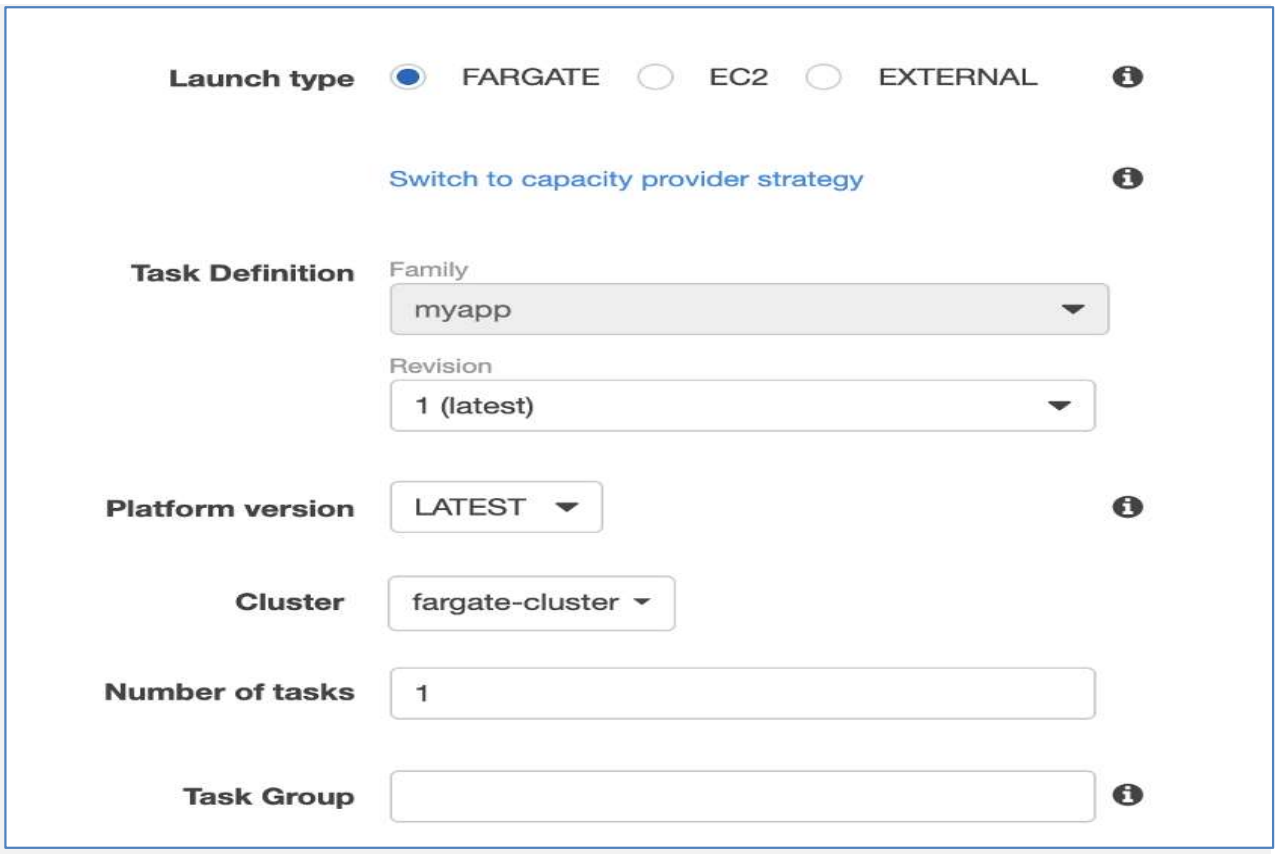
Step 1. Select the task in the Task definition list

Step 2. Click **Actions** and select **Run Task**



Step 3. For **Launch type**: select **Fargate**

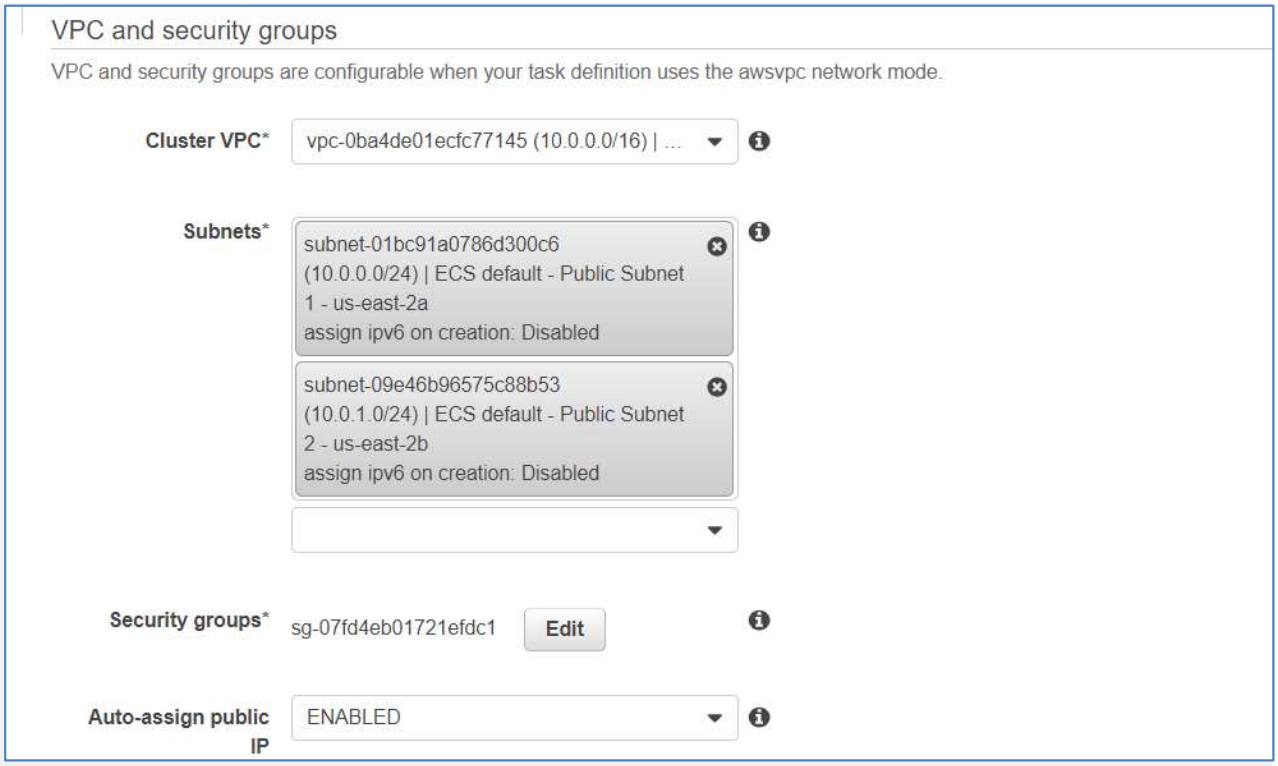
Step 4. Make sure **Cluster**: is set to the **fargate-cluster** we created earlier.



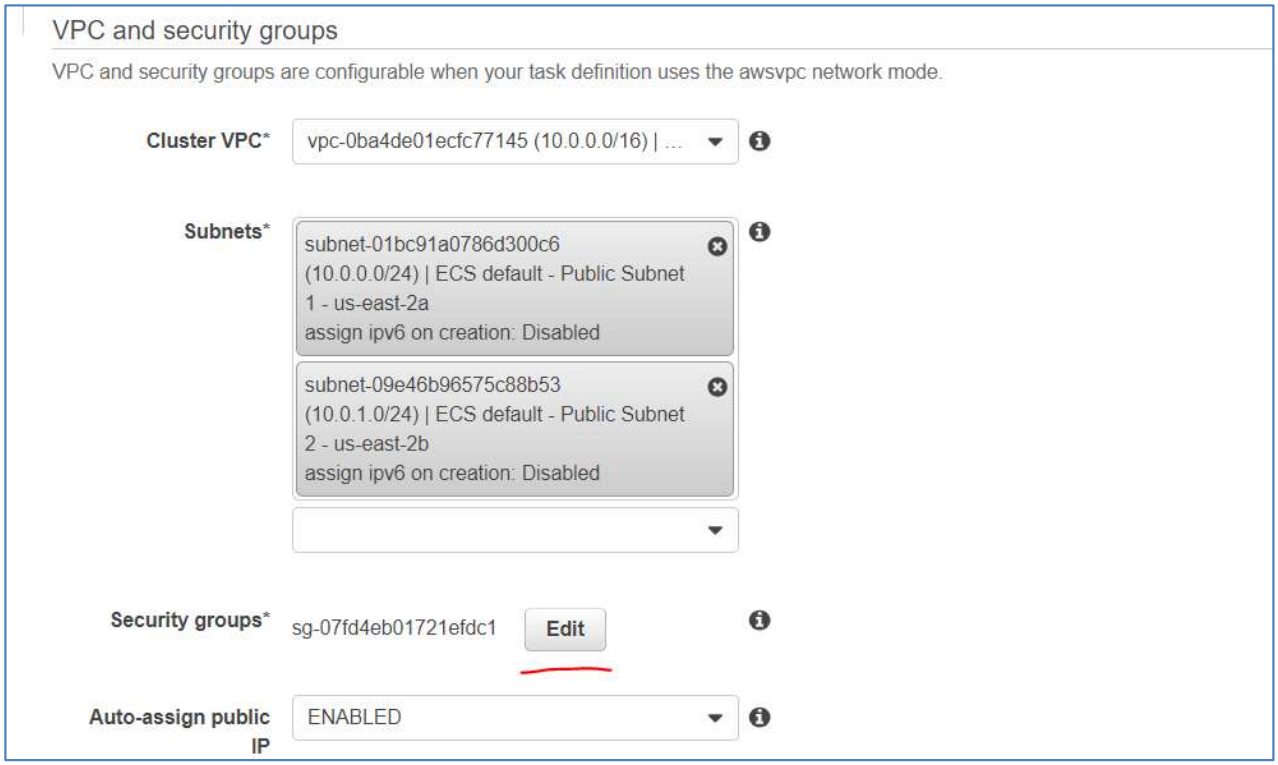
Step 5. Cluster VPC select a vpc from the list. If you are building a custom app this should be the vpc assigned to any other AWS services you will need to access from your instance. For our app, any will do.

Step 6. Add at least one subnet.

Step 7. **Auto-assign public IP** should be set to **ENBABLED**

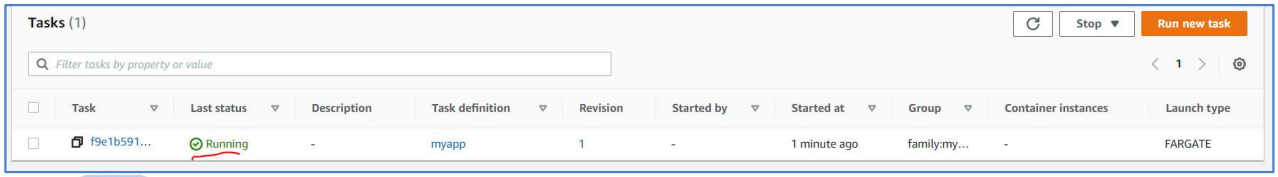


Step 8. **Edit the security group**. By default, the security group created by Run Task only allows incoming connections on port 80. Because our app listens on port 80, and we opened port 80 on our container, we don't need to open any additional port. But if you have any other port then click on Edit next to the security group name and add a Custom TCP rule that opens port 80.

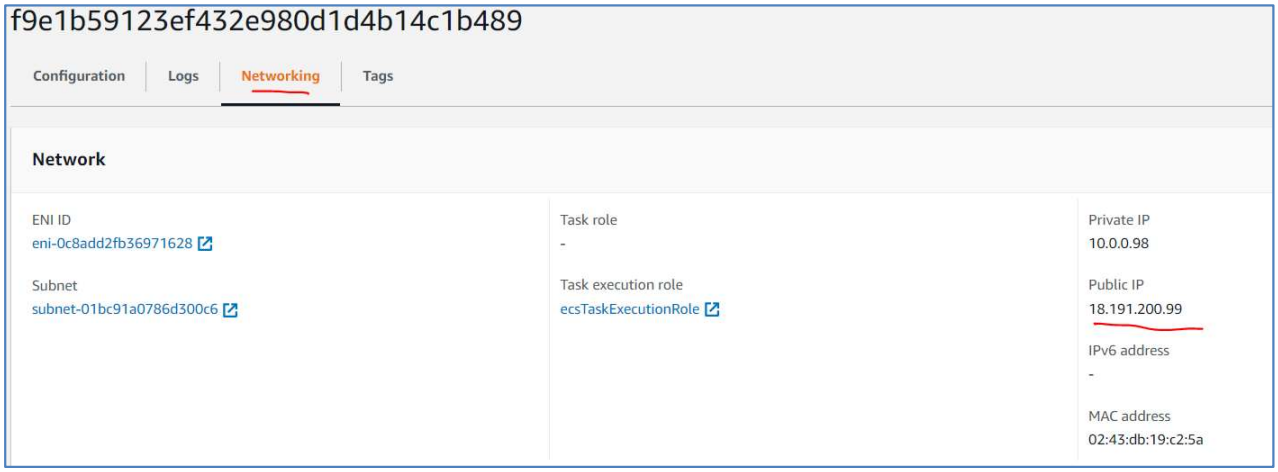


Step 9. And finally, run the task by clicking **Run Task** in the Right left corner of the page.

Step 10. Your app on ECS will be up and running in few moments



Step 11. Click on the task name and open it. Go to **Networking** tab and copy the public IP as below:



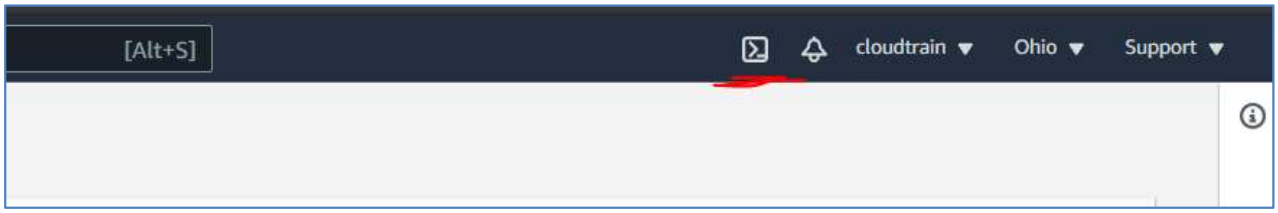
Step 12. Now, open this public IP like ***http:<public_IP>/devopsIQ*** in the browser and you shall be able to view your website running on ECS.

Congratulations!! You have successfully deployed a Website on ECS containers using Fargate cluster.

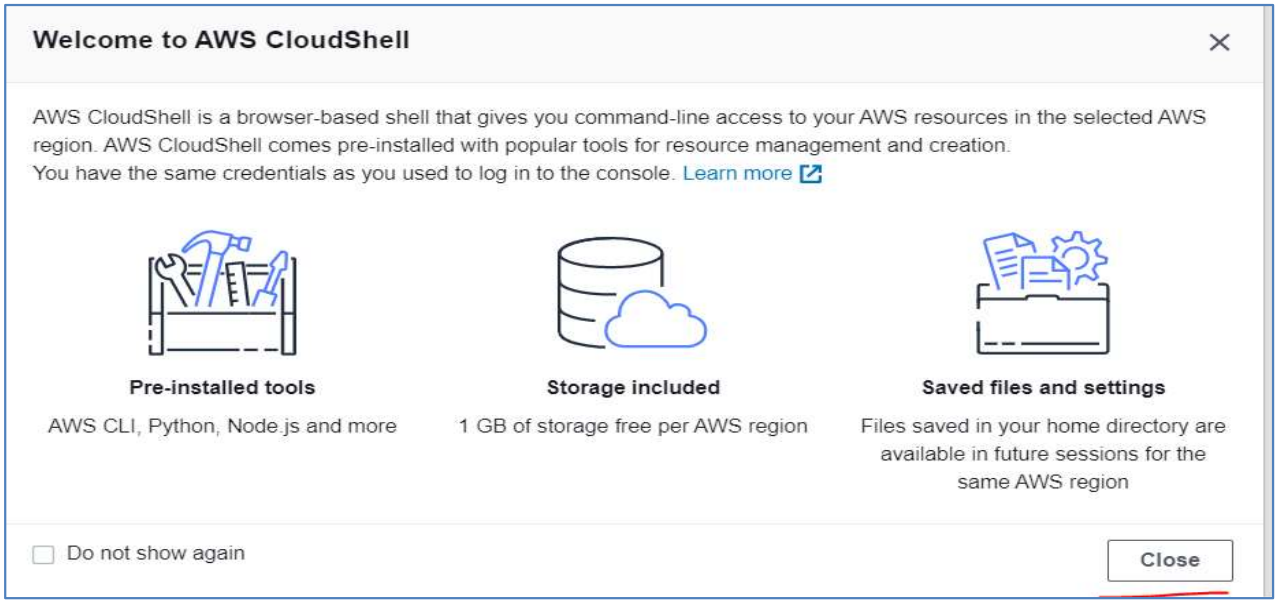
Setup an EKS cluster and deploy a Kubernetes application on it

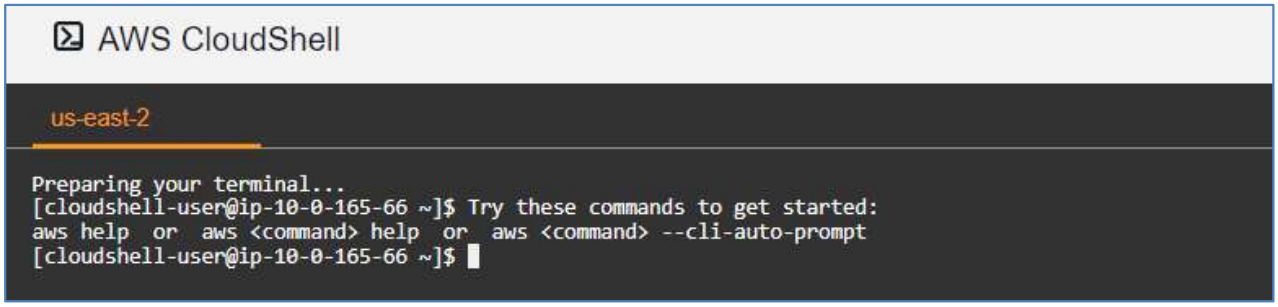
Prerequisites

Before starting this tutorial, you must install and configure the following tools and resources that you need to create and manage an Amazon EKS cluster. We will use **cloudshell** for this tutorial:



This will open a terminal in browser itself:





- **AWS CLI** – A command line tool for working with AWS services, including Amazon EKS. This guide requires that you use version 2.2.22 or later or 1.20.6 or later. **For us this step is already done as part of previous tutorial.**
- **kubectl** – A command line tool for working with Kubernetes clusters. This guide requires that you use version 1.21 or later. Install it using below command:

  Windows:

```
curl -o kubectl.exe https://amazon-eks.s3.us-west-2.amazonaws.com/1.21.2/2021-07-05/bin/windows/amd64/kubectl.exe
```

  Linux:

```
curl -o kubectl https://amazon-eks.s3-us-west-2.amazonaws.com/1.21.2/2021-07-05/bin/linux/amd64/kubectl

sudo mv kubectl /usr/local/bin

sudo chmod 755 /usr/local/bin/kubectl
```

```
[cloudshell-user@ip-10-0-165-66 ~]$ curl -o kubectl https://amazon-eks.s3-us-west-2.amazonaws.com/1.21.2/2021-07-05/bin/linux/amd64/kubectl
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 44.2M  100 44.2M    0     0  24.7M      0  0:00:01  0:00:01 --:--:-- 24.7M
[cloudshell-user@ip-10-0-165-66 ~]$ sudo mv kubectl /usr/local/bin
[cloudshell-user@ip-10-0-165-66 ~]$ sudo chmod 755 /usr/local/bin/kubectl
```

```
[cloudshell-user@ip-10-0-165-66 ~]$ kubectl version
Client Version: version.Info{Major:"1", Minor:"21+", GitVersion:"v1.21.2-13+d2965f0db10712", GitCommit:"d2965f0db1071203c6f5bc662c2827c71fc8b20d", GitTre
linux/amd64"}
```

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

Create your Amazon EKS cluster

To create your cluster

Step 1. Create an Amazon VPC with public and private subnets that meets Amazon EKS requirements. You can replace ***example values*** with your own, but we recommend using the example values in this tutorial.

```
aws cloudformation create-stack  --region us-west-2   --stack-name my-eks-vpc-stack  --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml
```

```
[cloudshell-user@ip-10-0-165-66 ~]$ aws cloudformation create-stack  --region us-west-2  --stack-name my-eks-vpc-stack  --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml
{
    "StackId": "arn:aws:cloudformation:us-west-2:090322976386:stack/my-eks-vpc-stack/92802010-f6a1-11eb-a398-02f4eabf161f"
}
```

Step 2. Create a cluster IAM role and attach the required Amazon EKS IAM managed policy to it. Kubernetes clusters managed by Amazon EKS make calls to other AWS services on your behalf to manage the resources that you use with the service.

a. Copy the following contents to a file named ***cluster-role-trust-policy.json***.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Principal": {
    "Service": "eks.amazonaws.com"
   },
   "Action": "sts:AssumeRole"
  }
 ]
}
```

```
[cloudshell-user@ip-10-0-165-66 ~]$ vi cluster-role-trust-policy.json
[cloudshell-user@ip-10-0-165-66 ~]$ cat cluster-role-trust-policy.json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. Create the role.

```
aws iam create-role --role-name myAmazonEKSClusterRole --assume-role-poli
cy-document file://"cluster-role-trust-policy.json"
```

```
[cloudshell-user@ip-10-0-165-66 ~]$ aws iam create-role --role-name myAmazonEKSClusterRole --assume-role-policy-document file://"cluster-role-trust-policy.json"
{
    "Role": {
        "Path": "/",
        "RoleName": "myAmazonEKSClusterRole",
        "RoleId": "AROARKB5LH2BPR2MEOKSD",
        "Arn": "arn:aws:iam::090322976386:role/myAmazonEKSClusterRole",
        "CreateDate": "2021-08-06T10:38:17+00:00",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "eks.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        }
    }
}
```

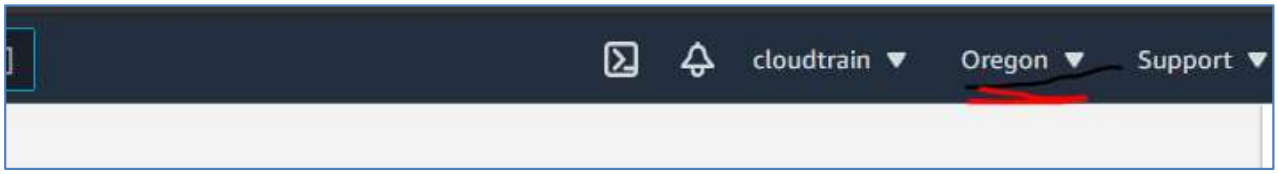
c. Attach the required Amazon EKS managed IAM policy to the role.

```
aws iam attach-role-policy  --policy-arn arn:aws:iam::aws:policy/AmazonEK
SClusterPolicy   --role-name myAmazonEKSClusterRole
```

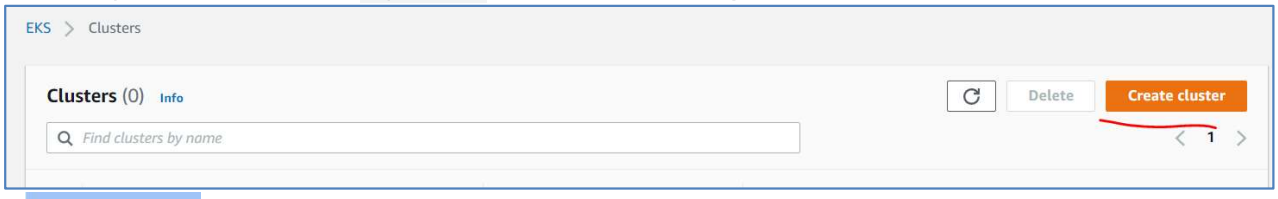
```
[cloudshell-user@ip-10-0-165-66 ~]$ aws iam attach-role-policy  --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy   --role-name myAmazonEKSClusterRole
[cloudshell-user@ip-10-0-165-66 ~]$
```

Step 3. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.

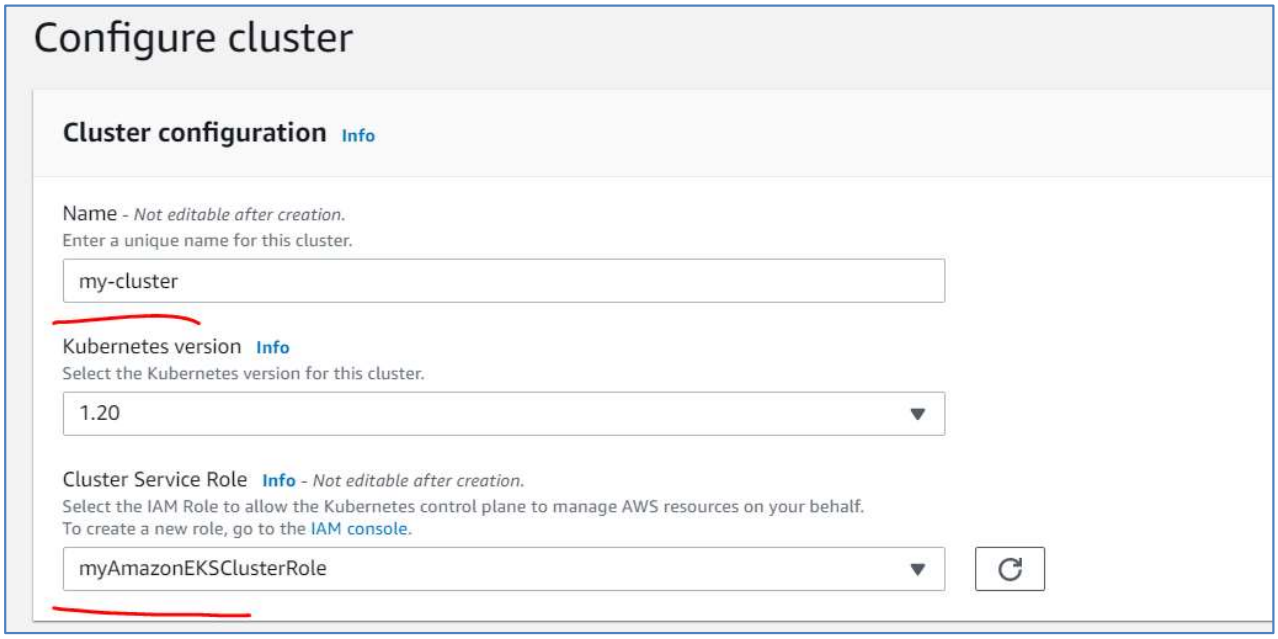
Make sure that the Region selected in the top right of your console is **Oregon**. If not, select the drop-down next to the Region name and select **US West (Oregon) us-west-2** . Though you can create a cluster in any Amazon EKS supported Region, in this tutorial, it's created in **US West (Oregon) us-west-2**.



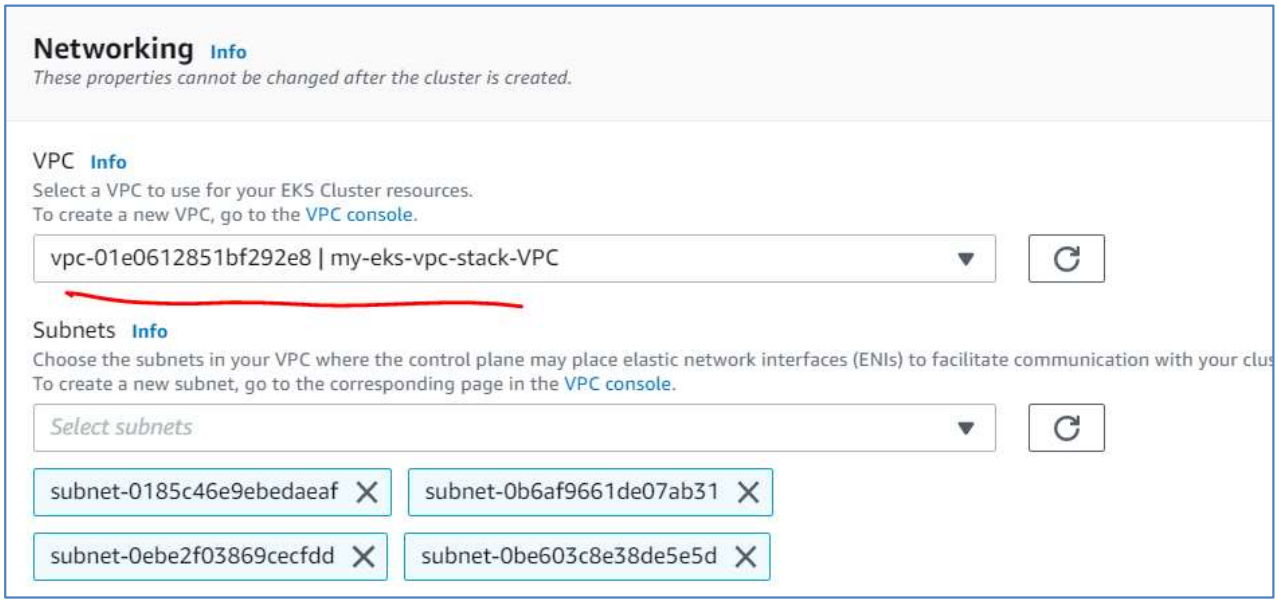
Step 4. Select **Create cluster**. If you don't see this option, in the **Create EKS cluster** box, enter a name for your cluster, such as my-cluster, and select **Next step**.



Step 5. On the **Configure cluster** page enter a name for your cluster, such as ***my-cluster*** and select ***myAmazonEKSClusterRole*** for **Cluster Service Role**. Leave the remaining settings at their default values and select **Next**.

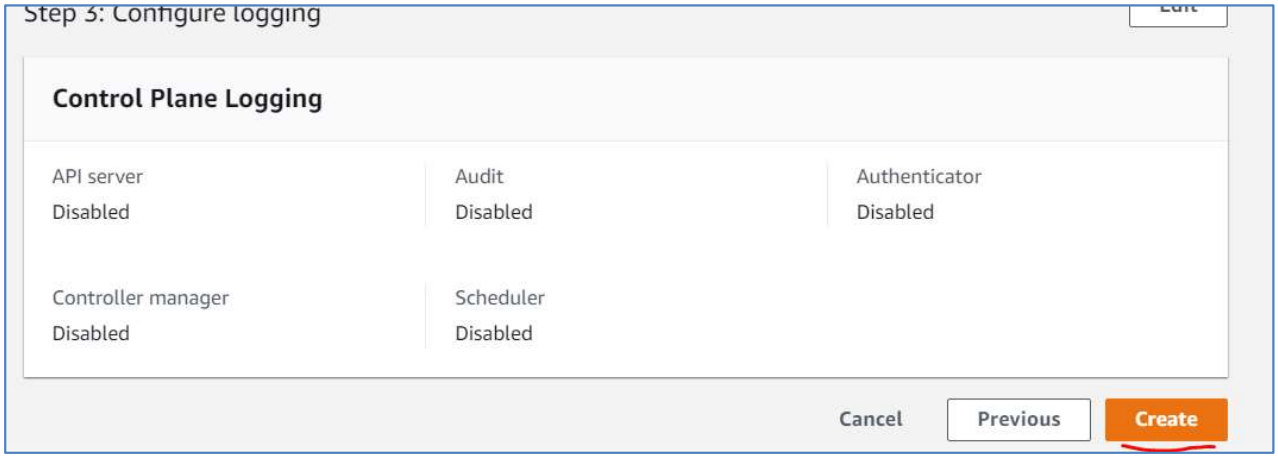


Step 6. On the **Specify networking** page, select ***vpc-00x0000x000x0x000 | my-eks-vpc-stack-VPC*** from the **VPC** drop down list. Leave the remaining settings at their default values and select **Next**.



Step 7. On the **Configure logging** page, select **Next**.

Step 8. On the **Review and create** page, select **Create**.



To the right of the cluster's name, the cluster status is **Creating** for several minutes until the cluster provisioning process completes. Don't continue to the next step until the status is **Active**.

Note: You might receive an error that one of the Availability Zones in your request doesn't have sufficient capacity to create an Amazon EKS cluster. If this happens, the error output contains the Availability Zones that can support a new cluster. Retry creating your cluster with at least two subnets that are located in the supported Availability Zones for your account.

Configure your computer to communicate with your cluster

In this section, you create a `kubeconfig` file for your cluster. The settings in this file enable the `kubectl` CLI to communicate with your cluster.

To configure your computer to communicate with your cluster

Step 1. Create or update a **kubeconfig** file for your cluster. If necessary, replace ***us-west-2*** with the Region that you created your cluster in.

```
aws eks update-kubeconfig --region us-west-2  --name my-cluster
```

By default, the `config` file is created in `~/.kube` or the new cluster's configuration is added to an existing `config` file in `~/.kube`.

```
[cloudshell-user@ip-10-0-165-66 ~]$ aws eks update-kubeconfig --region us-west-2  --name my-cluster
Added new context arn:aws:eks:us-west-2:090322976386:cluster/my-cluster to /home/cloudshell-user/.kube/config
```

Step 2. Test your configuration.

```
kubectl get svc
```

Output

```
[cloudshell-user@ip-10-0-165-66 ~]$ kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   5m24s
```

Create nodes

You can create a cluster with one of the following node types. To learn more about each type, see Amazon EKS nodes. After your cluster is deployed, you can add other node types.

- **Fargate – Linux** – Select this type if you want to run Linux applications on AWS Fargate.
- **Managed nodes – Linux** – Select this type if you want to run Amazon Linux applications on Amazon EC2 instances. Create a Fargate profile. When Kubernetes pods are deployed with criteria that matches the criteria defined in the profile, the pods are deployed to Fargate.

To create a Fargate profile

Step 1. Create an IAM role and attach the required Amazon EKS IAM managed policy to it. When your cluster creates pods on Fargate infrastructure, the components running on the Fargate infrastructure need to make calls to AWS APIs on your behalf to do things like pull container images from Amazon ECR or route logs to other AWS services. The Amazon EKS pod execution role provides the IAM permissions to do this.

Step 2. Copy the following contents to a file named ***pod-execution-role-trust-policy.json***.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Principal": {
    "Service": "eks-fargate-pods.amazonaws.com"
   },
   "Action": "sts:AssumeRole"
  }
 ]
}
```

```
[cloudshell-user@ip-10-0-165-66 ~]$ vi pod-execution-role-trust-policy.json
[cloudshell-user@ip-10-0-165-66 ~]$ cat pod-execution-role-trust-policy.json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks-fargate-pods.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Step 3. Create a pod execution IAM role.

```
aws iam create-role  --role-name myAmazonEKSFargatePodExecutionRole  --as
sume-role-policy-document file://"pod-execution-role-trust-policy.json"
```

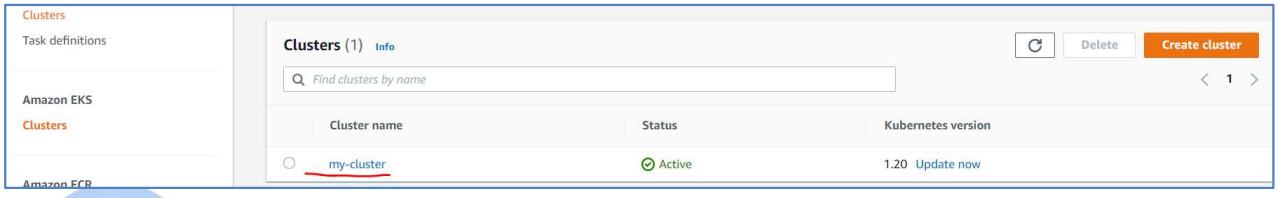
```
[cloudshell-user@ip-10-0-165-66 ~]$ aws iam create-role  --role-name myAmazonEKSFargatePodExecutionRole  --assume-role-policy-document file://"pod-ex
ecution-role-trust-policy.json"
```

Step 4. Attach the required Amazon EKS managed IAM policy to the role.

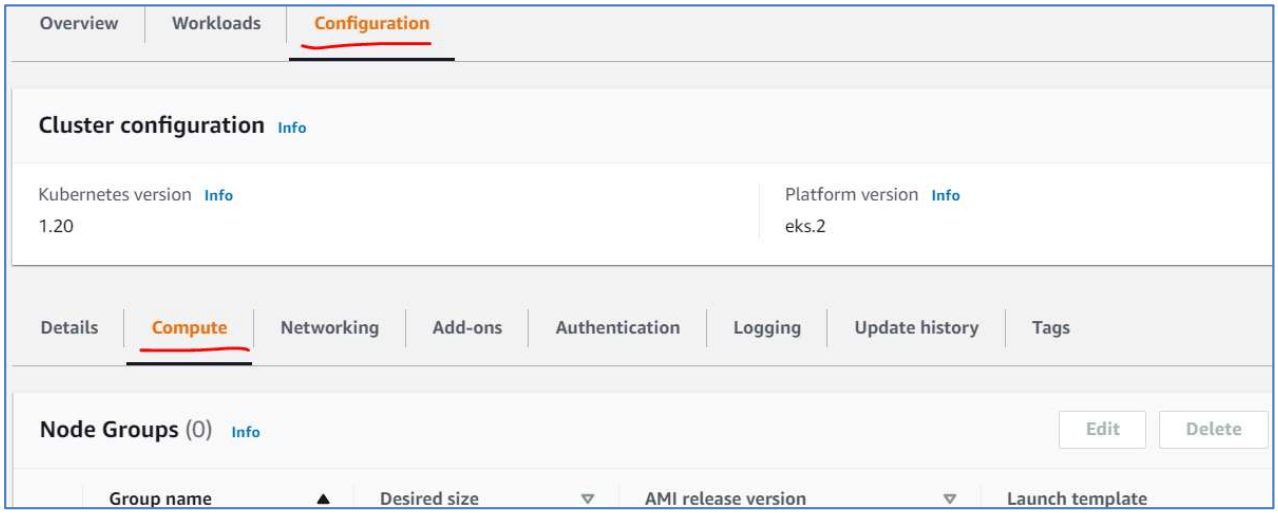
```
aws iam attach-role-policy  --policy-arn arn:aws:iam::aws:policy/AmazonEK
SFargatePodExecutionRolePolicy --role-name myAmazonEKSFargatePodExecution
Role
```

```
[cloudshell-user@ip-10-0-165-66 ~]$ aws iam attach-role-policy  --policy-arn arn:aws:iam::aws:policy/AmazonEKSFargatePodExecutionRolePolicy --role-na
me myAmazonEKSFargatePodExecutionRole
[cloudshell-user@ip-10-0-165-66 ~]$
```

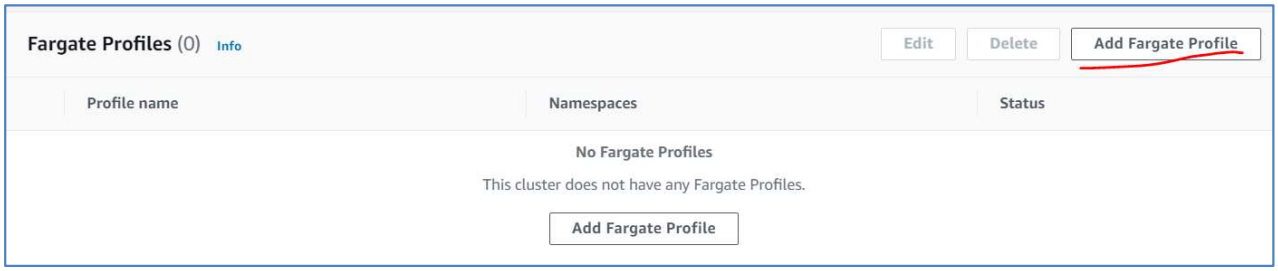
Step 5. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.



Step 6. Choose the cluster to create a Fargate profile for and select the **Configuration** tab, then the **Compute** tab.



Step 7. Under **Fargate profiles**, choose **Add Fargate profile**.

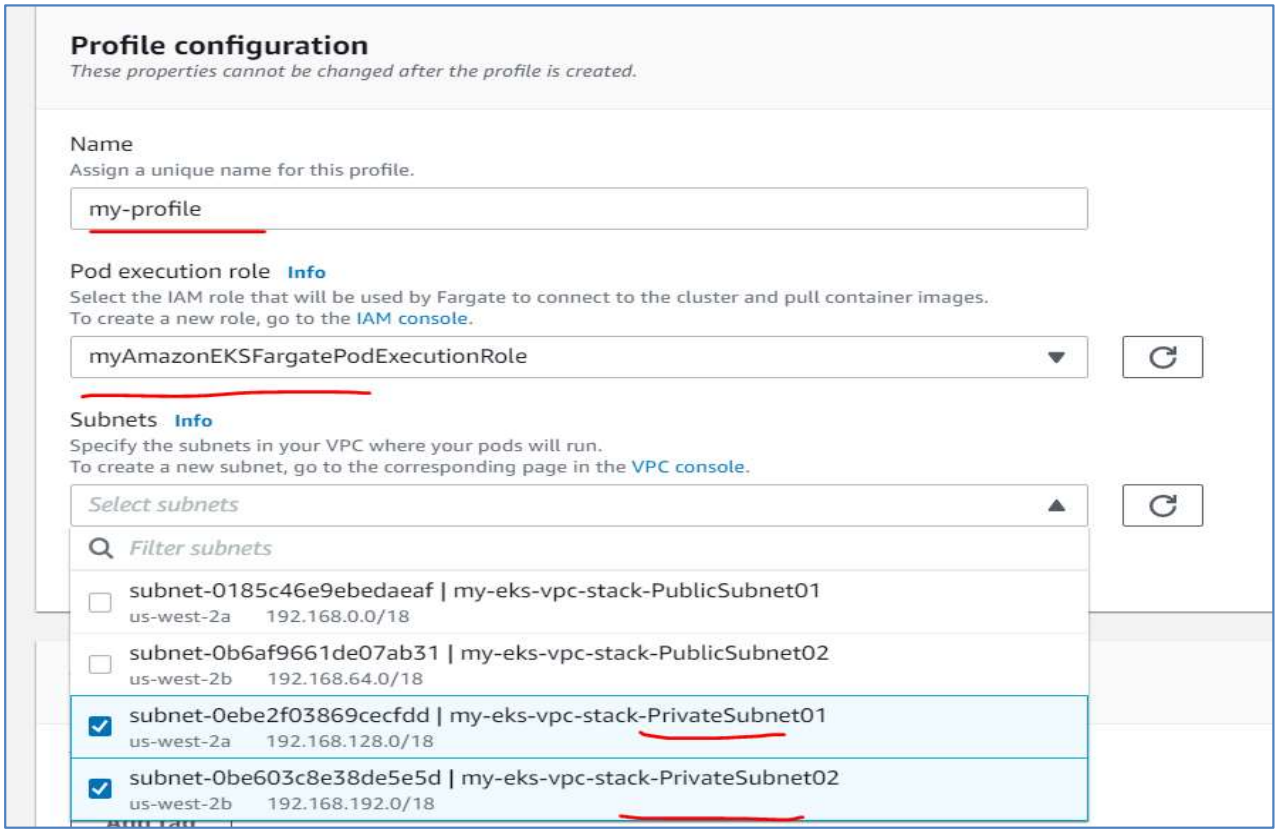


Step 8. On the **Configure Fargate profile** page, enter the following information and choose **Next**.

a) For **Name**, enter a unique name for your Fargate profile, such as ***my-profile***.

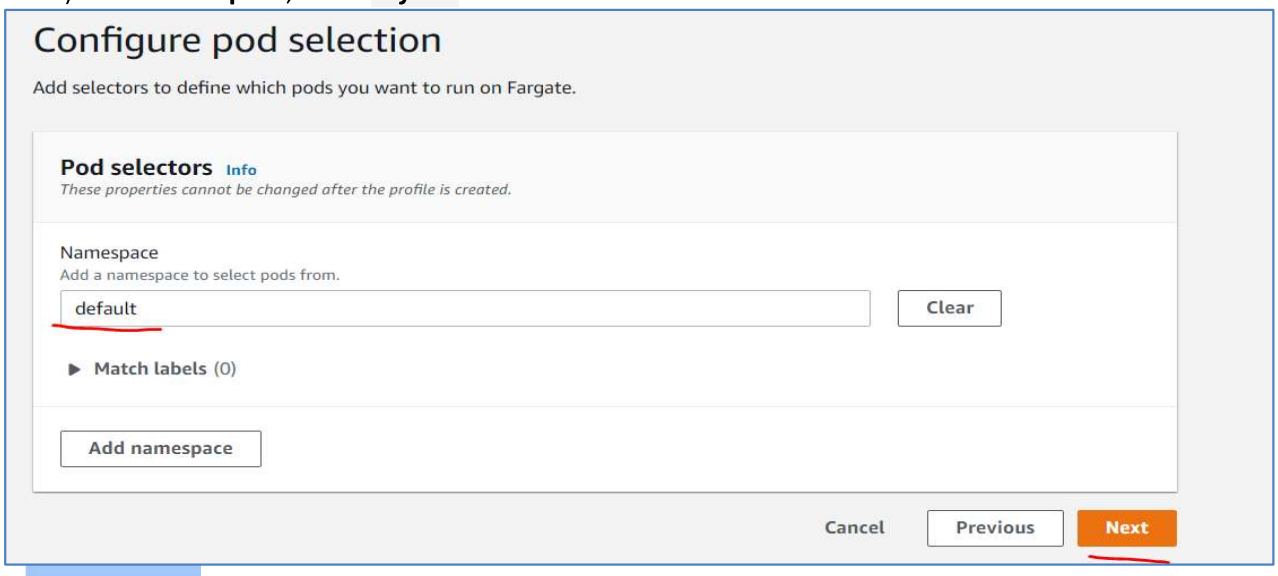
b) For **Pod execution role**, choose the ***myAmazonEKSFargatePodExecutionRole*** role that you created in step one.

c) Select the **Subnets** dropdown and unselect any subnet with Public in its name. Only private subnets are supported for pods running on Fargate.

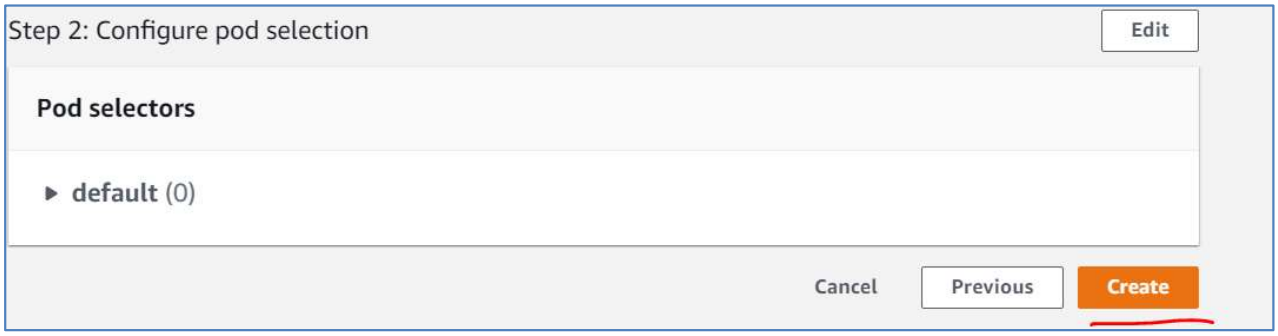


Step 9. On the **Configure pods selection** page, enter the following information and choose **Next**.

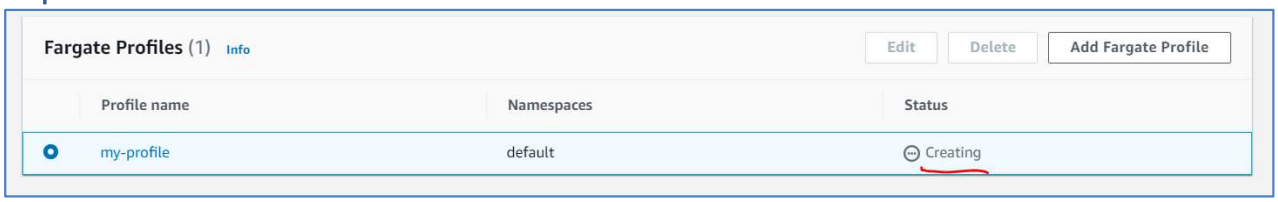
a) For **Namespace**, enter ***default*** and click **Next**.



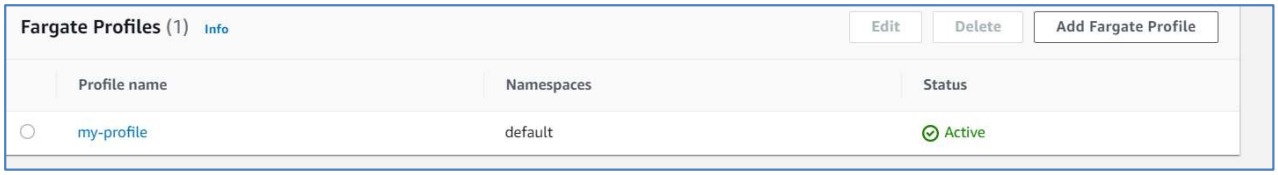
Step 10. On the **Review and create** page, review the information for your Fargate profile and choose **Create**.



Step 11. It will take some time to create. Check the status here:



Step 12. Wait until it becomes active:



View resources

You can view your nodes and Kubernetes workloads.

To view your nodes

Step 1. In the left pane, select **Clusters**, and then in the list of **Clusters**, select the name of the cluster that you created, such as ***my-cluster***.

Step 2. On the **Overview** tab, you see the list of **Nodes** that were deployed for the cluster. You can select the name of a node to see more information about it. Initially you wont see anything until some app is deployed

Step 3. On the **Workloads** tab of the cluster, you see a list of the workloads that are deployed by default to an Amazon EKS cluster. You can select the name of a workload to see more information about it. Initially you wont see anything until some app is deployed

To deploy a sample application

Step 1. (Optional) Create a Kubernetes namespace for the sample app, only if using other than default in **Fargate** profile

```
kubectl create namespace <my-namespace>
```

Step 2. Create a Kubernetes service and deployment.

a. Save the following contents to a file named **k8s-app.yaml** on your computer. If you're deploying to AWS Fargate pods, then make sure that the value for **namespace** matches the namespace that you defined in your AWS Fargate profile. This sample deployment will pull a container image from a public repository, deploy three replicas of it to your cluster, and create a Kubernetes service with its own IP address that can be accessed from within the cluster only. To access the service from outside the cluster, you need to deploy a network load balancer or ALB Ingress Controller.

```
apiVersion: v1
kind: Service
metadata:
 name: my-service
 namespace: default
 labels:
   app: my-app
spec:
 selector:
   app: my-app
 ports:
  - protocol: TCP
    port: 80
    targetPort: 80
---
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: my-deployment
 namespace: default
 labels:
  app: my-app
spec:
 replicas: 3
 selector:
  matchLabels:
   app: my-app
 template:
  metadata:
   labels:
    app: my-app
  spec:
   affinity:
    nodeAffinity:
     requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: beta.kubernetes.io/arch
          operator: In
          values:
          - amd64
          - arm64
   containers:
   - name: nginx
     image: public.ecr.aws/z9d2n7e1/nginx:1.19.5
     ports:
     - containerPort: 80
```

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  namespace: default
  labels:
    app: my-app
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
  namespace: default
  labels:
    app: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: beta.kubernetes.io/arch
                operator: In
                values:
                - amd64
                - arm64
      containers:
      - name: nginx
        image: public.ecr.aws/z9d2n7e1/nginx:1.19.5
        ports:
        - containerPort: 80
```

Step 3. Deploy the application.

```
kubectl apply -f k8s-app.yaml
```

```
[cloudshell-user@ip-10-0-165-66 ~]$ kubectl apply -f k8s-app.yaml
service/my-service created
deployment.apps/my-deployment created
```

Step 4. View all resources that exist in the my-namespace namespace.

```
kubectl get all
```

Output

```
[cloudshell-user@ip-10-0-165-66 ~]$ kubectl get all
NAME                                 READY   STATUS    RESTARTS   AGE
pod/my-deployment-5dd5dfd6b9-7vn6m   1/1     Running   0          9m50s
pod/my-deployment-5dd5dfd6b9-bfhw6   1/1     Running   0          9m50s
pod/my-deployment-5dd5dfd6b9-tmwsw   1/1     Running   0          9m50s

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.100.0.1      <none>        443/TCP   81m
service/my-service   ClusterIP   10.100.109.12   <none>        80/TCP    9m51s

NAME                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/my-deployment   3/3     3            3           9m52s

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/my-deployment-5dd5dfd6b9   3         3         3       9m52s
```

Step 5. View the details of the deployed service.

```
kubectl describe service my-service
```

Abbreviated output

```
[cloudshell-user@ip-10-0-165-66 ~]$ kubectl describe service my-service
Name:              my-service
Namespace:         default
Labels:            app=my-app
Annotations:       <none>
Selector:          app=my-app
Type:              ClusterIP
IP Families:       <none>
IP:                10.100.109.12
IPs:               10.100.109.12
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         192.168.203.25:80,192.168.220.117:80,192.168.220.236:80
Session Affinity:  None
Events:            <none>
```

In the output, the value for IP: is a unique IP address that can be reached from any pod within the cluster.

Step 6. View the details of one of the pods that was deployed.

```
kubectl describe pod my-deployment-5dd5dfd6b9-7vn6m
```

Abbreviated output

```
Volumes:
  default-token-fwpt9:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-fwpt9
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                 node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason   Age   From     Message
  ----    ------   ----  ----     -------
  Normal  Pulling  12m   kubelet  Pulling image "public.ecr.aws/z9d2n7e1/nginx:1.19.5"
  Normal  Pulled   12m   kubelet  Successfully pulled image "public.ecr.aws/z9d2n7e1/nginx:1.19.5" in 5.277814795s
  Normal  Created  12m   kubelet  Created container nginx
  Normal  Started  12m   kubelet  Started container nginx
```

Step 7. Execute a shell on one of the pods:

```
kubectl exec -it my-deployment-5dd5dfd6b9-7vn6m -- /bin/bash
```

Step 8. View the DNS resolver configuration file.

```
cat /etc/resolv.conf
```

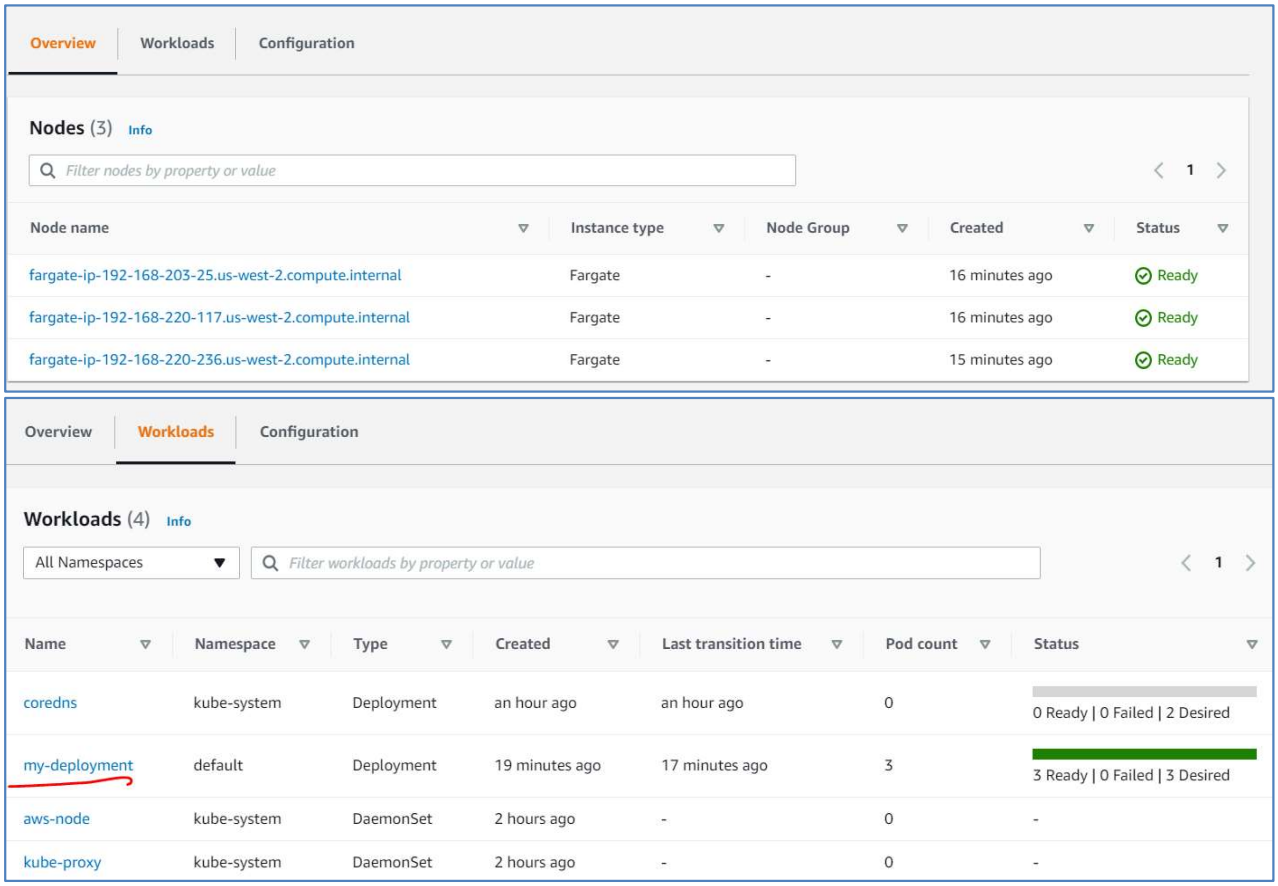
Output

```
root@my-deployment-5dd5dfd6b9-7vn6m:/# cat /etc/resolv.conf
search default.svc.cluster.local svc.cluster.local cluster.local us-west-2.compute.internal
nameserver 10.100.0.10
options ndots:5
```

In the previous output, the value for **nameserver** is the cluster's nameserver and is automatically assigned as the name server for any pod deployed to the cluster.

Step 9. Disconnect from the pod by typing **exit**.

Step 10. You can see nodes added on **Overview** tab of cluster and workload deployed under **Workloads** tab:



Step 11. Remove the sample service, deployment, pods, and namespace.

```
kubectl delete -f k8s-app.yaml
```

```
[cloudshell-user@ip-10-0-165-66 ~]$ kubectl delete -f k8s-app.yaml
service "my-service" deleted
deployment.apps "my-deployment" deleted
```

Delete your cluster and nodes

After you've finished with the cluster and nodes that you created for this tutorial, you should clean up by deleting the cluster and nodes.

To delete your cluster and nodes

Step 1. Delete all node groups and Fargate profiles.

a. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.

b. In the left navigation, select **Clusters**, and then in the list of clusters, select the name of the cluster that you want to delete.

c. Select the **Configuration** tab. On the **Compute** tab, select:

- The node group that you created in a previous step and select **Delete**. Enter the name of the node group, and then select **Delete**.
- The **Fargate Profile** that you created in a previous step and select **Delete**. Enter the name of the profile, and then select **Delete**.

Step 2. Delete the cluster.

a. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.

b. Select the cluster to delete and choose **Delete**.

c. On the delete cluster confirmation screen, choose **Delete**.

Step 3. Delete the VPC AWS CloudFormation stack that you created in this guide.

a. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.

b. Select the VPC stack to delete, and choose **Delete**.

c. On the **Delete Stack** confirmation screen, choose **Delete stack**.

Step 4. Delete the IAM roles that you created.

a. Open the IAM console at https://console.aws.amazon.com/iam/.

b. In the left navigation pane, select **Roles**.

c. Select the ***myAmazonEKSClusterRole*** from the list. Select **Delete role**, and then select **Yes, Delete**. Delete the ***myAmazonEKSFargatePodExecutionRole*** or ***myAmazonEKSNodeRole*** role that you created and the ***myAmazonEKSCNIRole*** role, if you created one.

CI/CD

Setup an AWS CodePipeline using CodeCommit, CodeBuild and CodeDeploy

In this tutorial, you use CodePipeline to deploy code maintained in a CodeCommit repository to a single Amazon EC2 instance. Your pipeline is triggered when you push a change to the CodeCommit repository. The pipeline deploys your changes to an Amazon EC2 instance using CodeDeploy as the deployment service.

The pipeline has two stages:

- A source stage (**Source**) for your CodeCommit source action.
- A deployment stage (**Deploy**) for your CodeDeploy deployment action.

The easiest way to get started with AWS CodePipeline is to use the **Create Pipeline** wizard in the CodePipeline console.

Note

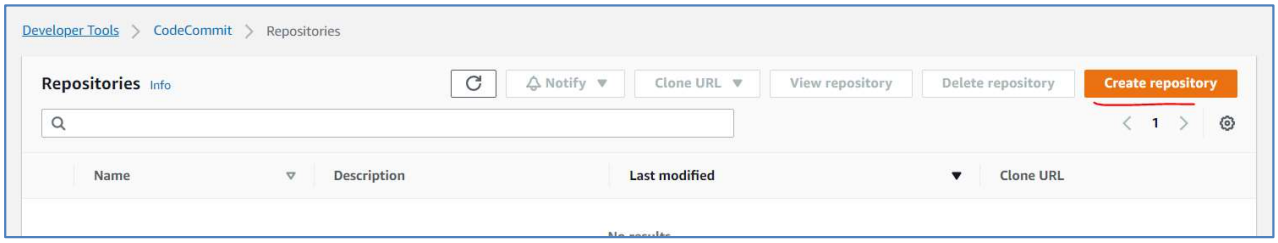
Before you begin, make sure you've set up your Git client to work with CodeCommit. For instructions, see Setting up for CodeCommit.

Create a CodeCommit repository

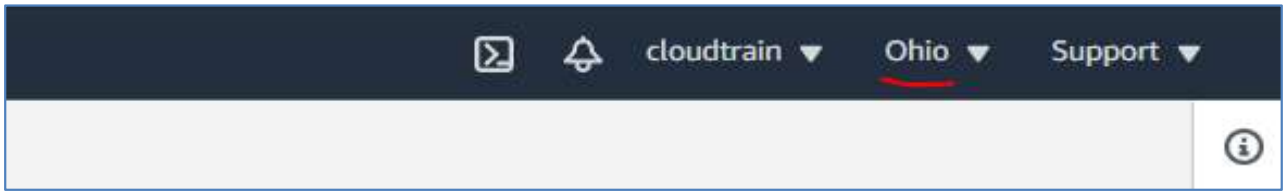
First, you create a repository in CodeCommit. Your pipeline gets source code from this repository when it runs. You also create a local repository where you maintain and update code before you push it to the CodeCommit repository.

To create a CodeCommit repository

Step 1. Open the CodeCommit console at https://console.aws.amazon.com/codecommit/.



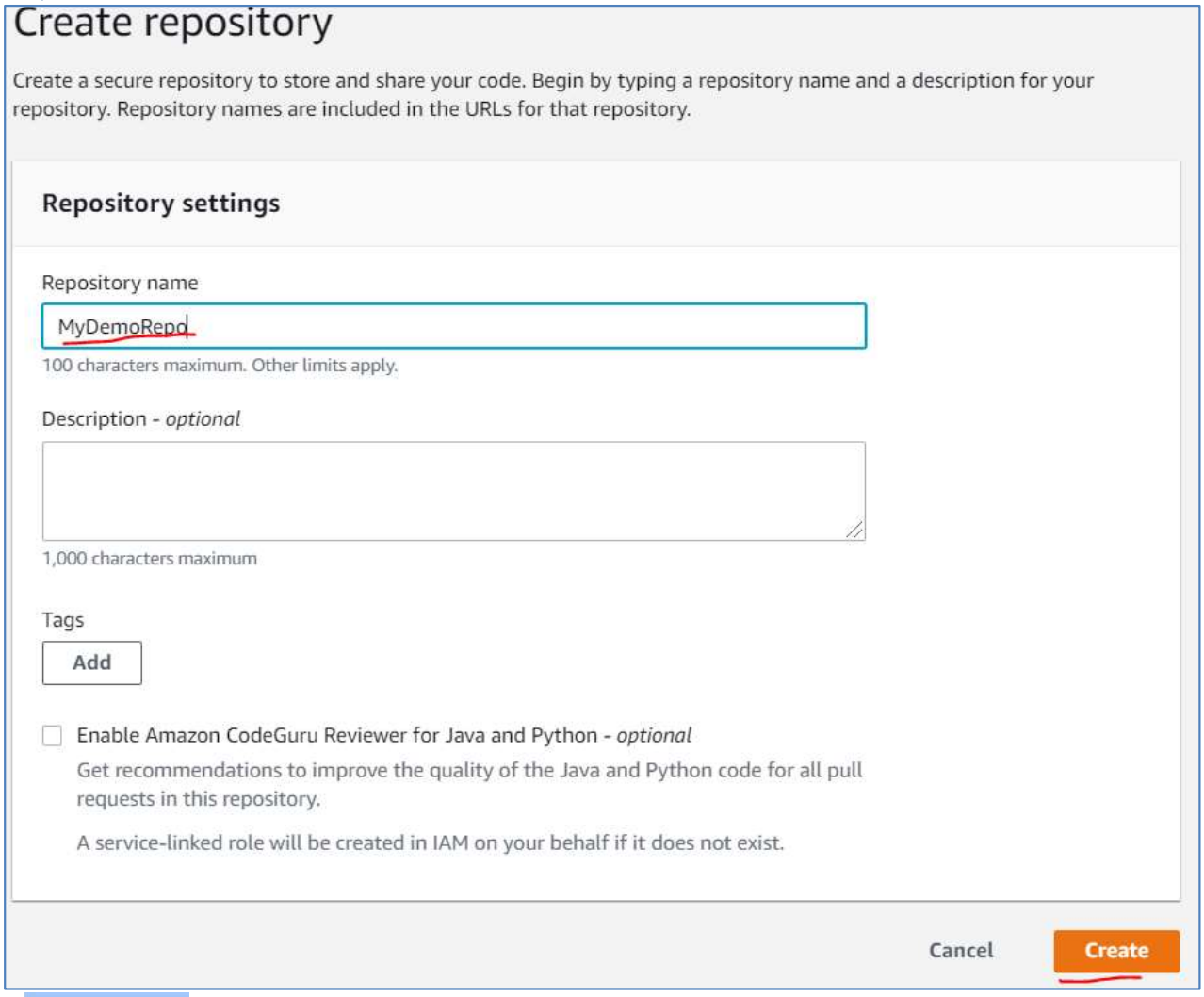
Step 2. In the Region selector, choose the AWS Region where you want to create the repository and pipeline.



Step 3. On the **Repositories** page, choose **Create repository**.

Step 4. On the **Create repository** page, in **Repository name**, enter a name for your repository (for example, **MyDemoRepo**).

Step 5. Choose **Create**.



Note: The remaining steps in this tutorial use **MyDemoRepo** for the name of your CodeCommit repository. If you choose a different name, be sure to use it throughout this tutorial.

To set up a local repository (Optional- Not required for this exercise)

In this step, you set up a local repository to connect to your remote CodeCommit repository.

Note: You are not required to set up a local repository. You can also use the console to upload files as described in **Add sample code to your CodeCommit repository.**

Step 1. With your new repository open in the console, choose **Clone URL** on the top right of the page, and then choose **Clone SSH**. The address to clone your Git repository is copied to your clipboard.

Step 2. In your terminal or command line, navigate to a local directory where you'd like your local repository to be stored. In this tutorial, we use **/tmp**.

Step 3. Run the following command to clone the repository, replacing the SSH address with the one you copied in the previous step. This command creates a directory called **MyDemoRepo**. You copy a sample application to this directory.

```
git clone ssh://git-codecommit.us-west-2.amazonaws.com/v1/repos/MyDemoRepo
```

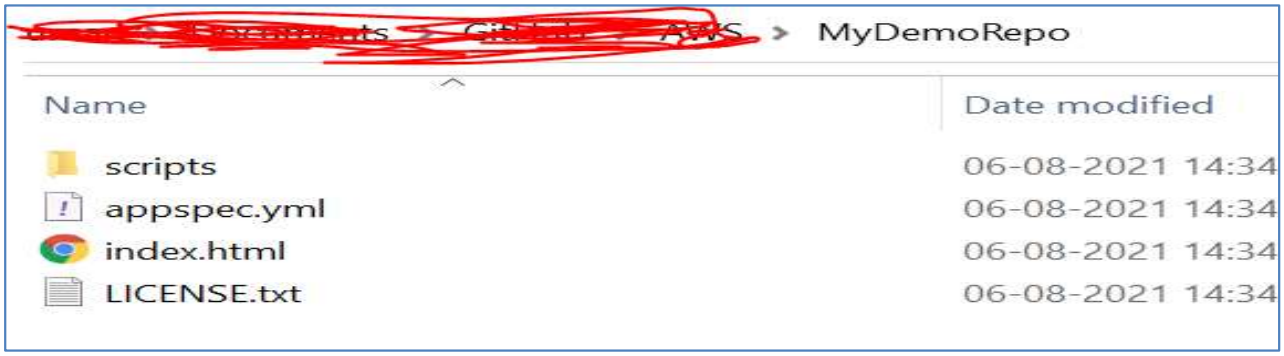
Add sample code to your CodeCommit repository

In this step, you download code for a sample application that was created for a CodeDeploy sample walkthrough, and add it to your CodeCommit repository.

Step 1. Download the following file: SampleApp_Linux.zip: (URL: https://docs.aws.amazon.com/codepipeline/latest/userguide/samples/SampleApp_Linux.zip)

Step 2. Unzip the files from SampleApp_Linux.zip into the local directory you created earlier (for example, **/tmp/MyDemoRepo** or **c:\temp\MyDemoRepo**).

Be sure to place the files directly into your local repository. Do not include a **SampleApp_Linux** folder. On your local Linux, macOS, or Unix machine, for example, your directory and file hierarchy should look like this:



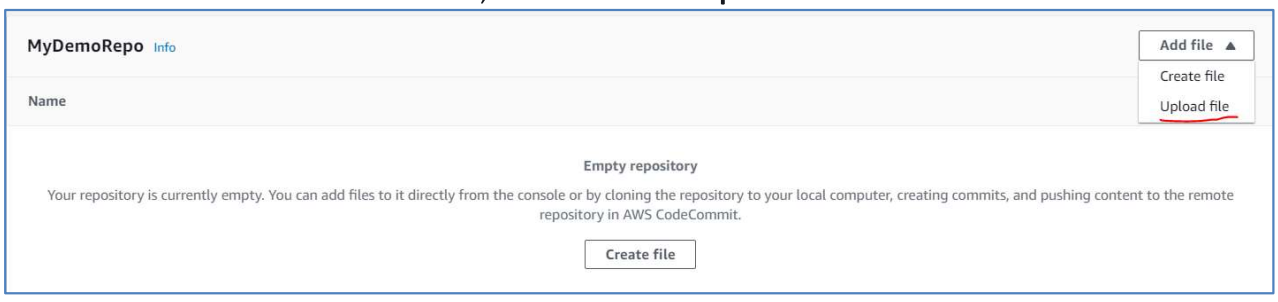

```
└-- MyDemoRepo
    |-- appspec.yml
    |-- index.html
    |-- LICENSE.txt
    └-- scripts
        |-- install_dependencies
        |-- start_server
        └-- stop_server
```

Step 3. To upload files to your repository, use one of the following methods.

a. To use the CodeCommit console to upload your files:

i. Open the CodeCommit console, and choose your repository from the **Repositories** list.

ii. Choose **Add file**, and then choose **Upload file**.



iii. Select **Choose file**, and then browse for your file. To add a file under a folder, choose **Create file** and then enter the folder name with the file name, such as **scripts/install_dependencies**. Paste the file contents into the new file.

MyDemoRepo Info

```
1 #!/bin/bash
2 yum install -y httpd
```

Commit changes to main

File name
For example, file.txt

scripts/install_dependencies

MyDemoRepo/scripts/install_dependencies

Author name

test

Email address

test

Upload a file

MyDemoRepo Info

Name	Size	Actions

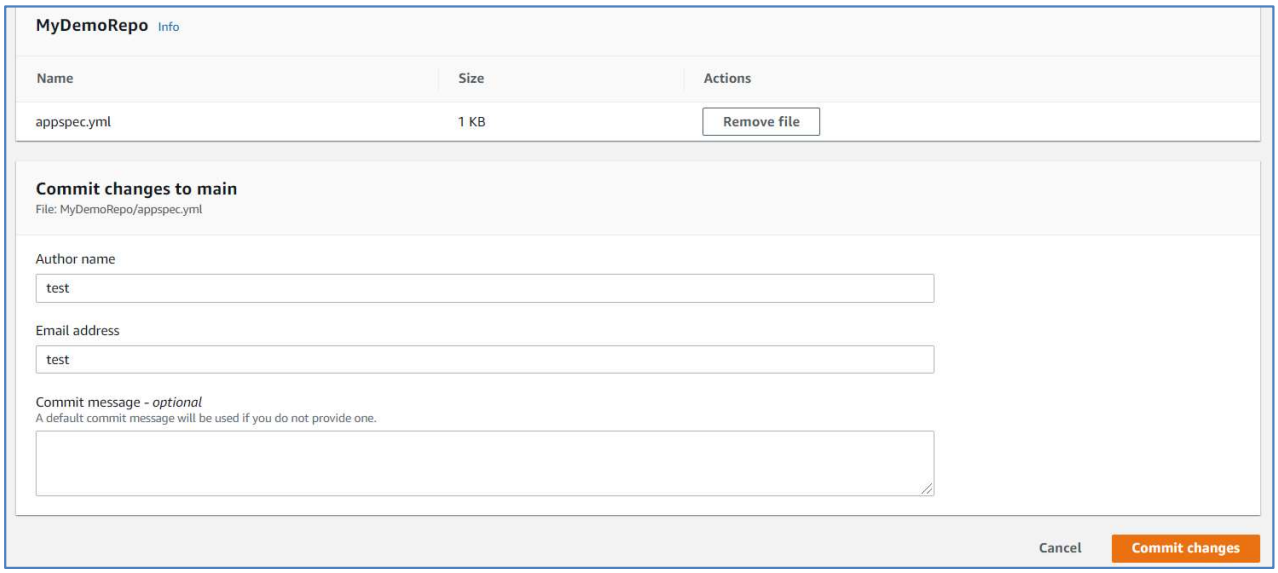
Upload file

Choose a file to upload.

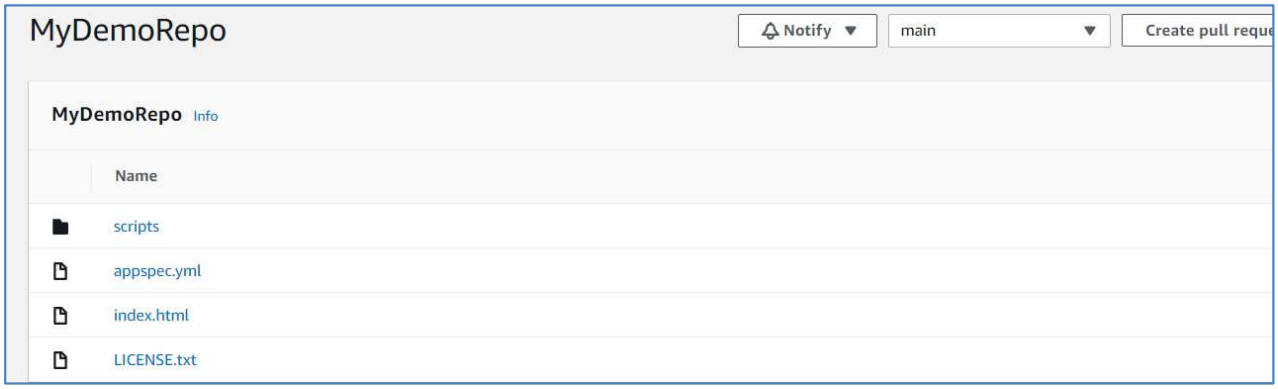
Choose file

iv. Commit the change by entering your user name and email address. Enter anything here for this exercise

v. Choose Commit changes.



vi. Repeat this step for each file. Your repository contents should look like this:




```
|-- appspec.yml
|-- index.html
|-- LICENSE.txt
└-- scripts
    |-- install_dependencies
    |-- start_server
    └-- stop_server
```

b. To use git commands to upload your files (Optional – **Not used in this exercise**):

i. Change directories to your local repo:

(For Linux, macOS, or Unix)

```
cd /tmp/MyDemoRepo
```

(For Windows)

```
cd c:\temp\MyDemoRepo
```

ii. Run the following command to stage all of your files at once:

```
git add -A
```

iii. Run the following command to commit the files with a commit message:

```
git commit -m "Sample application files"
```

iv. Run the following command to push the files from your local repo to your CodeCommit repository:

```
git push
```

Step 4. The files you downloaded and added to your local repo have now been added to the main branch in your CodeCommit **MyDemoRepo** repository and are ready to be included in a pipeline.

Create an EC2 Linux instance and install the CodeDeploy agent

In this step, you create the EC2 instance where you deploy a sample application. As part of this process, you install the CodeDeploy agent on the EC2 instance. The CodeDeploy agent is a software package that enables an instance to be used in CodeDeploy deployments. You also attach an IAM role to the instance (known as an *instance role*) to allow it to fetch files that the CodeDeploy agent uses to deploy your application.

To create an instance role

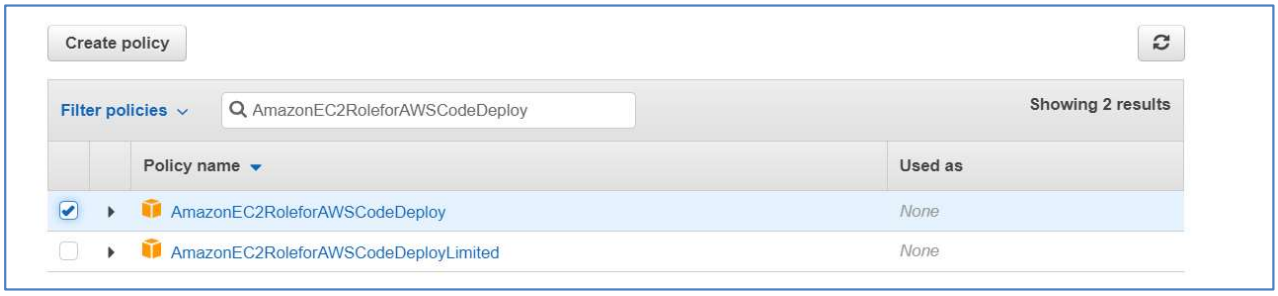
Step 1. Open the IAM console at https://console.aws.amazon.com/iam/).

Step 2. From the console dashboard, choose **Roles**.

Step 3. Choose **Create role**.

Step 4. Under **Select type of trusted entity**, select **AWS service**. Under **Choose a use case**, select **EC2**. Under **Select your use case**, choose **EC2**. Choose **Next: Permissions**.

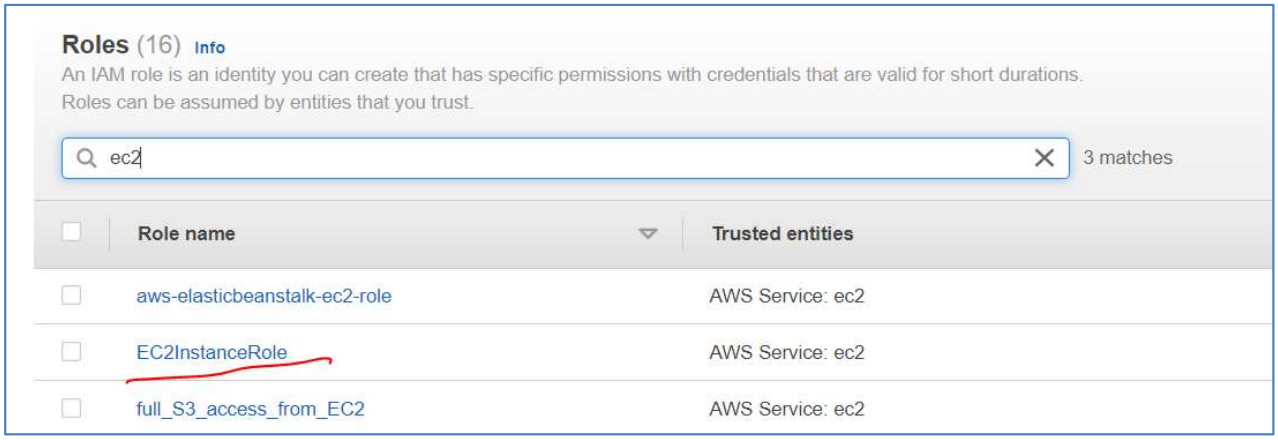
Step 5. Search for and select the policy named **AmazonEC2RoleforAWSCodeDeploy**, and then choose **Next: Tags**.



Step 6. Choose **Next: Review**. Enter a name for the role (for example, **EC2InstanceRole**).

Note: Make a note of your role name for the next step. You choose this role when you are creating your instance.

Step 7. Choose **Create role**.



To launch an instance

Step 1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

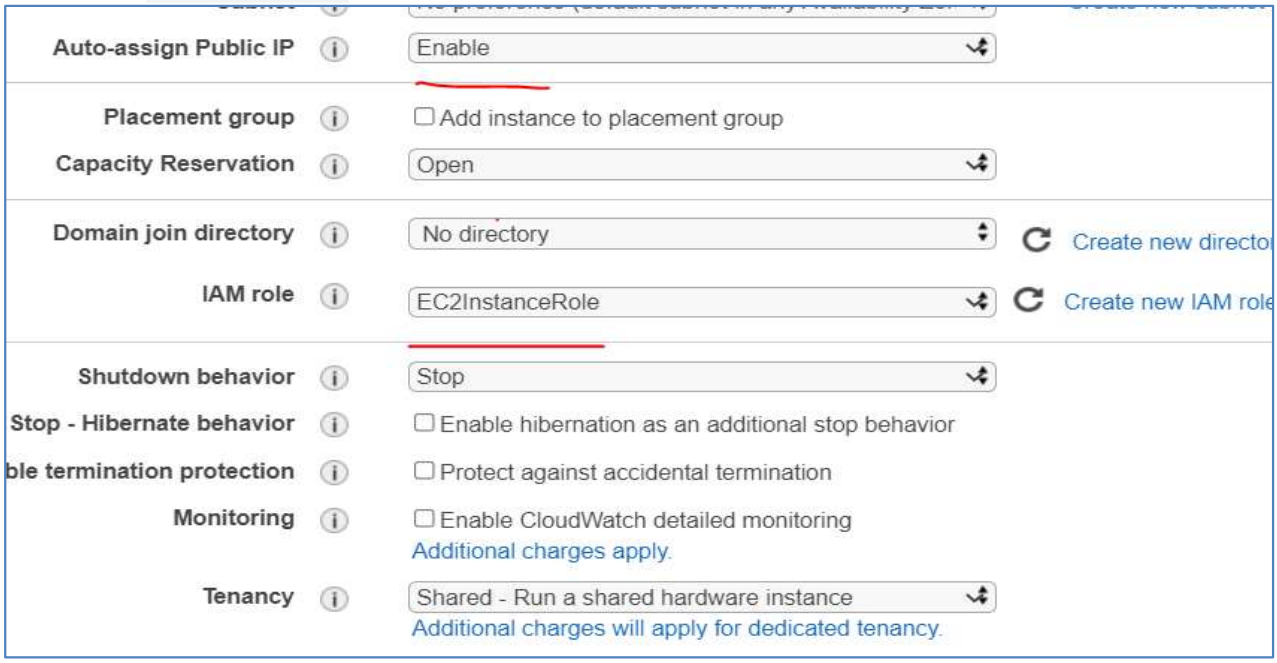
Step 2. From the console dashboard, choose **Launch instance**, and select **Launch instance** from the options that pop up.

Step 3. On **Step 1: Choose an Amazon Machine Image (AMI)**, locate **Amazon Linux 2 AMI (HVM), SSD Volume Type**, and then choose **Select**. (This AMI is labeled "Free tier eligible" and can be found at the top of the list.)

Step 4. On the **Step 2: Choose an Instance Type** page, choose the free tier eligible t2.micro type as the hardware configuration for your instance, and then choose **Next: Configure Instance Details**.

Step 5. On the **Step 3: Configure Instance Details** page, do the following:

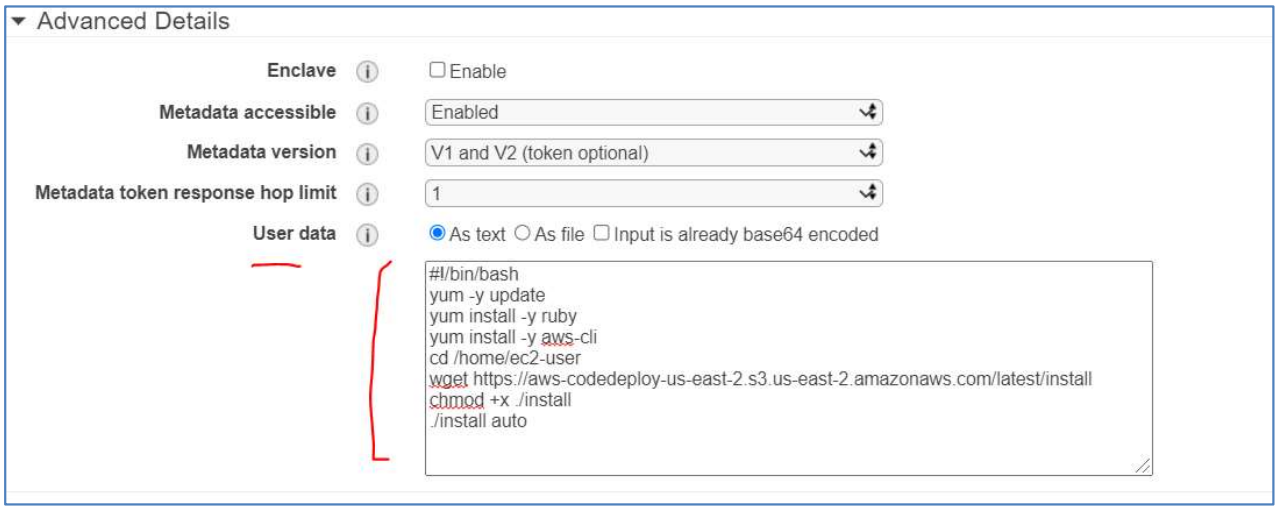
- In **Number of instances**, enter 1.
- In **Auto-assign Public IP**, choose **Enable**.
- In **IAM role**, choose the IAM role you created in the previous procedure (for example, **EC2InstanceRole**).



- Expand **Advanced Details**, and in the **User data** field, enter the following:

```
#!/bin/bash
yum -y update
yum install -y ruby
yum install -y aws-cli
cd /home/ec2-user
wget https://aws-codedeploy-us-east-2.s3.us-east-2.amazonaws.com/latest/install
chmod +x ./install
./install auto
```

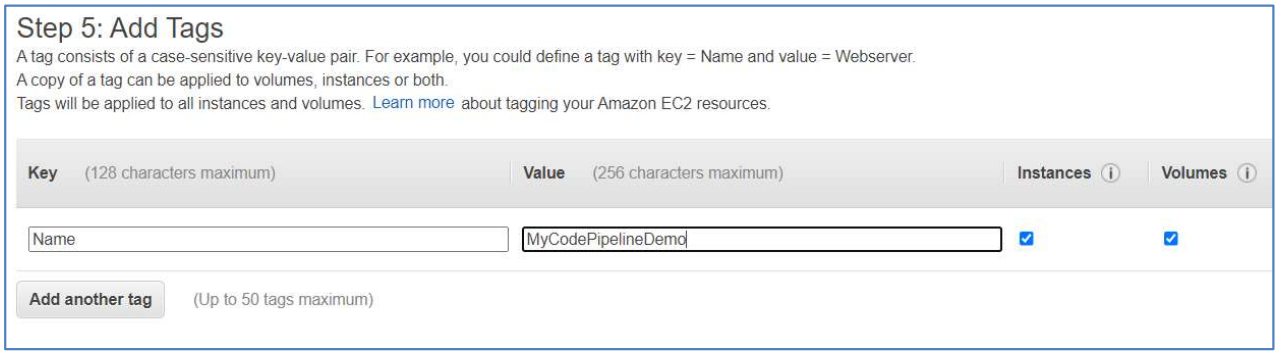
This code installs the CodeDeploy agent on your instance as it is created.



- Leave the rest of the items on the **Step 3: Configure Instance Details** page unchanged. Choose **Next: Add Storage**.

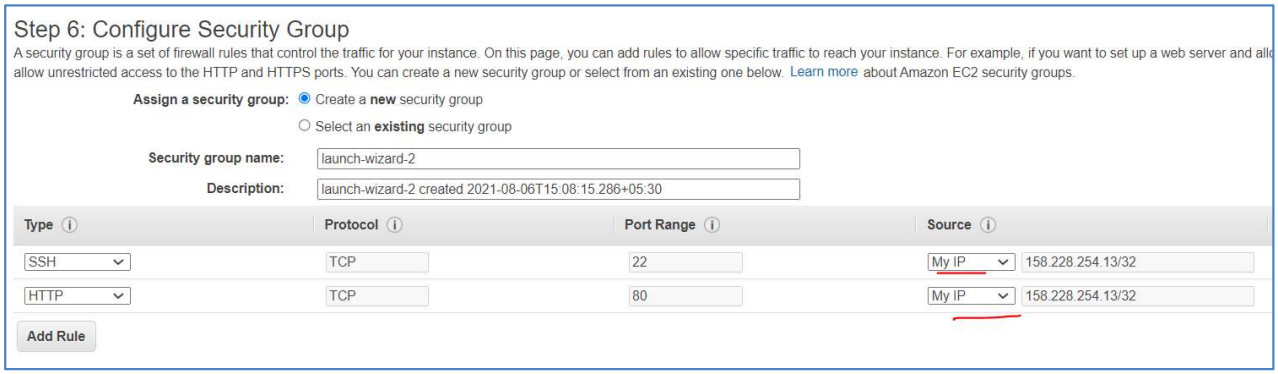
Step 6. Leave the **Step 4: Add Storage** page unchanged, and then choose **Next: Add Tags**.

Step 7. Choose **Add Tag**. In **Key**, enter **Name**, and in **Value**, enter **MyCodePipelineDemo**. Choose **Next: Configure Security Group**. Later, you create a CodeDeploy application that deploys the sample application to this instance. CodeDeploy selects instances to deploy based on the tags that are attached to instances.



Step 8. On the **Step 6: Configure Security Group** page, do the following:

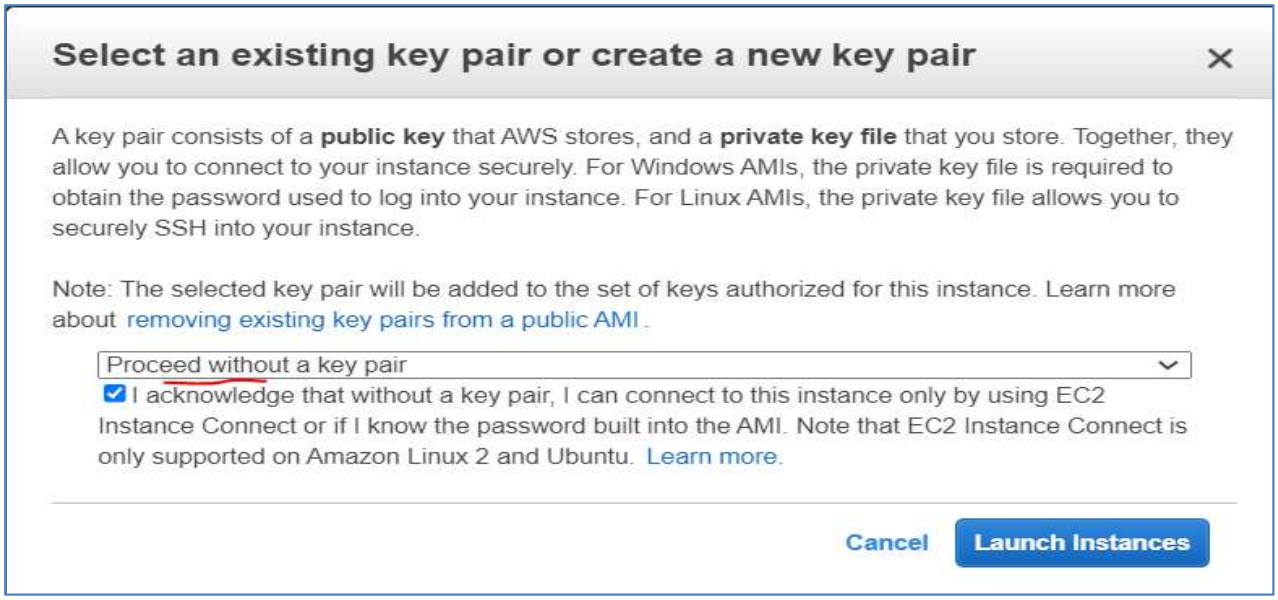
- Next to **Assign a security group**, choose **Create a new security group**.
- In the row for **SSH**, under **Source**, choose **My IP**.
- Choose **Add Rule**, choose **HTTP**, and then under **Source**, choose **My IP**.



Step 9. Choose **Review and Launch**.

Step 10. On the **Review Instance Launch** page, choose **Launch**. When prompted for a key pair, choose **Proceed without a key pair**.

Note: For the purposes of this tutorial, you can proceed without a key pair. To use SSH to connect to your instances, create or use a key pair.



When you are ready, select the acknowledgment check box, and then choose **Launch Instances**.

Step 11. Choose **View Instances** to close the confirmation page and return to the console.

Step 12. You can view the status of the launch on the **Instances** page. When you launch an instance, its initial state is pending. After the instance starts, its state changes to running, and it receives a public DNS name. (If the **Public DNS** column is not displayed, choose the **Show/Hide** icon, and then select **Public DNS**.)

Step 13. It can take a few minutes for the instance to be ready for you to connect to it. View the information in the **Status Checks** column to see if your instance has passed its status checks.

Create an application in CodeDeploy

In CodeDeploy, an *application* is a resource that contains the software application you want to deploy. Later, you use this application with CodePipeline to automate deployments of the sample application to your Amazon EC2 instance.

First, you create a role that allows CodeDeploy to perform deployments. Then, you create a CodeDeploy application.

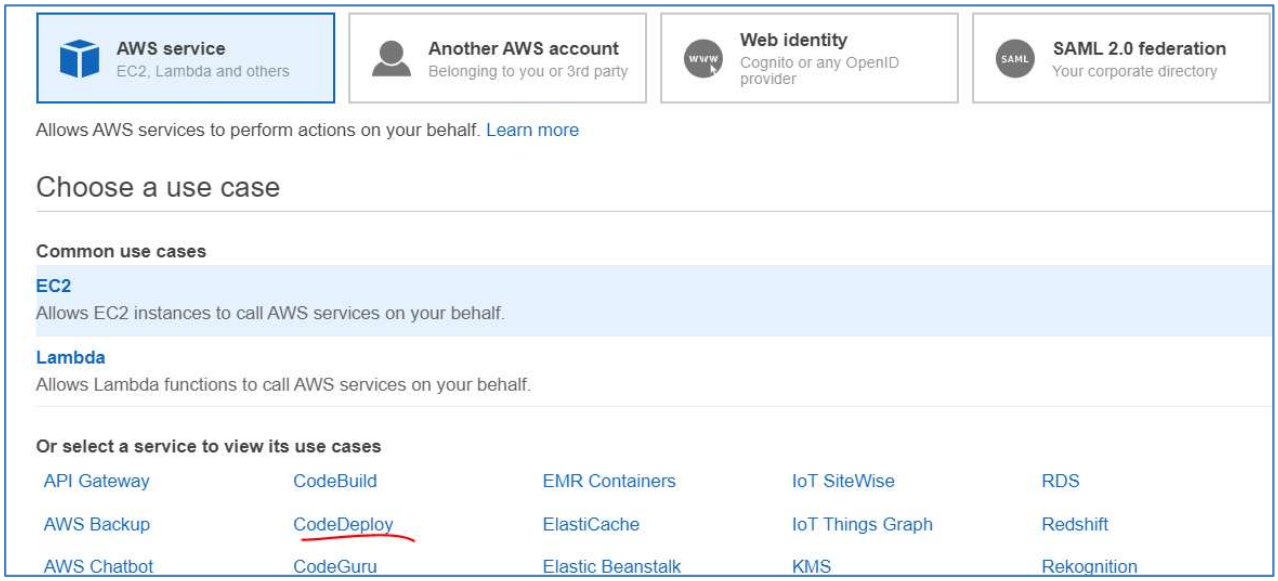
To create a CodeDeploy service role

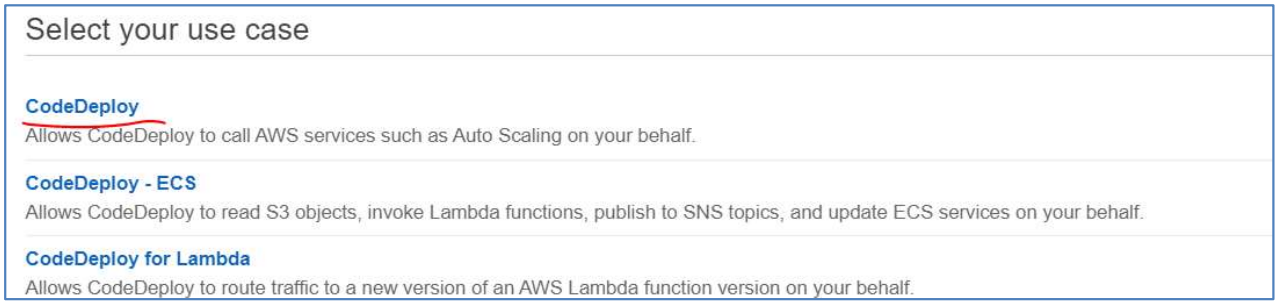
Step 1. Open the IAM console at https://console.aws.amazon.com/iam/).

Step 2. From the console dashboard, choose **Roles**.

Step 3. Choose **Create role**.

Step 4. Under **Select type of trusted entity**, select **AWS service**. Under **Choose a use case**, select **CodeDeploy**. Under **Select your use case**, choose **CodeDeploy**. Choose **Next: Permissions**. The ***AWSCodeDeployRole*** managed policy is already attached to the role.





Step 5. Choose **Next: Tags**, and **Next: Review**.

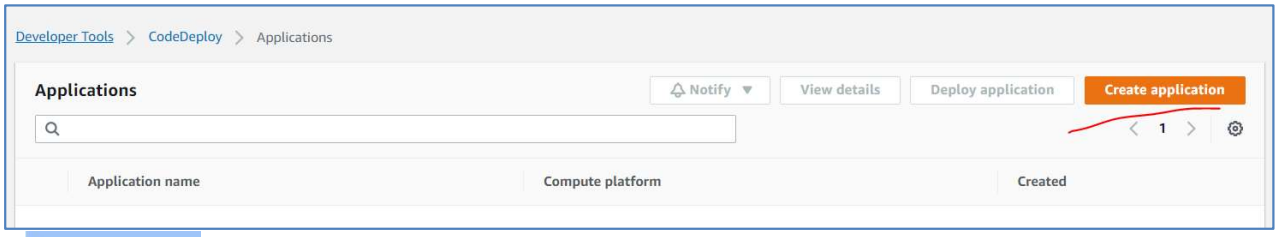
Step 6. Enter a name for the role (for example, **CodeDeployRole**), and then choose **Create role**.

To create an application in CodeDeploy

Step 1. Open the CodeDeploy console at https://console.aws.amazon.com/codedeploy.

Step 2. If the **Applications** page does not appear, on the AWS CodeDeploy menu, choose **Applications**.

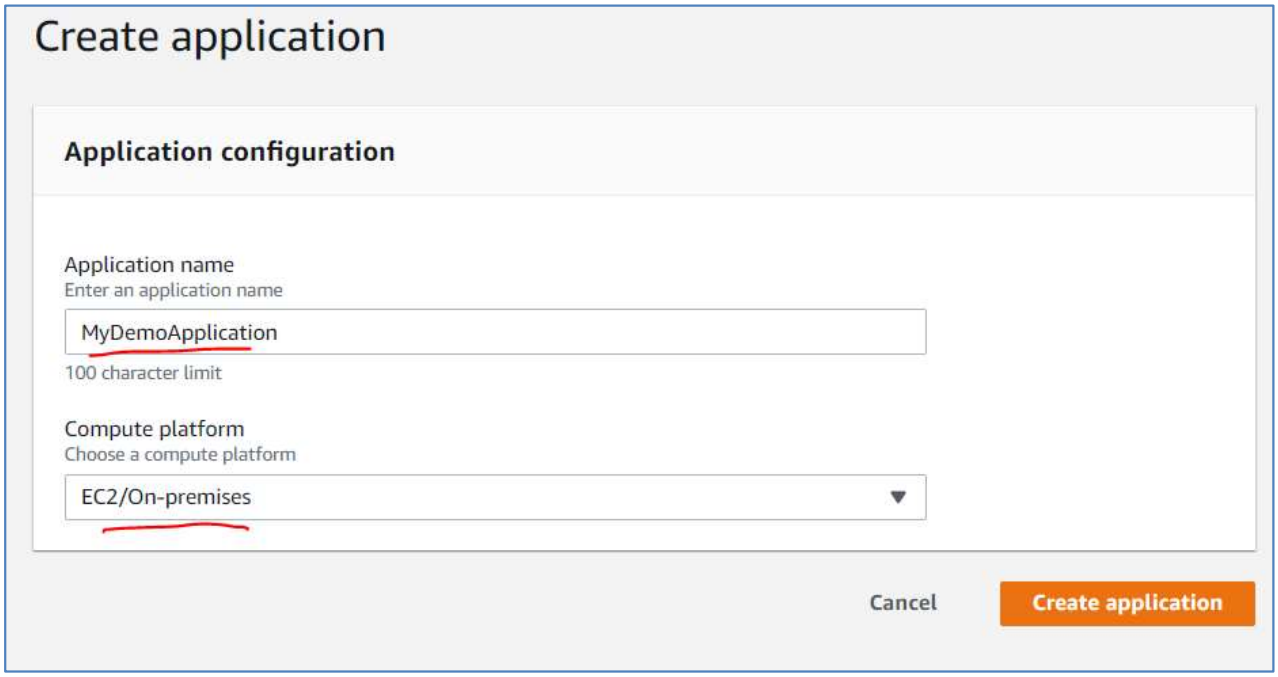
Step 3. Choose **Create application**.



Step 4. In **Application name**, enter **MyDemoApplication**.

Step 5. In **Compute Platform**, choose **EC2/On-premises**.

Step 6. Choose **Create application**.

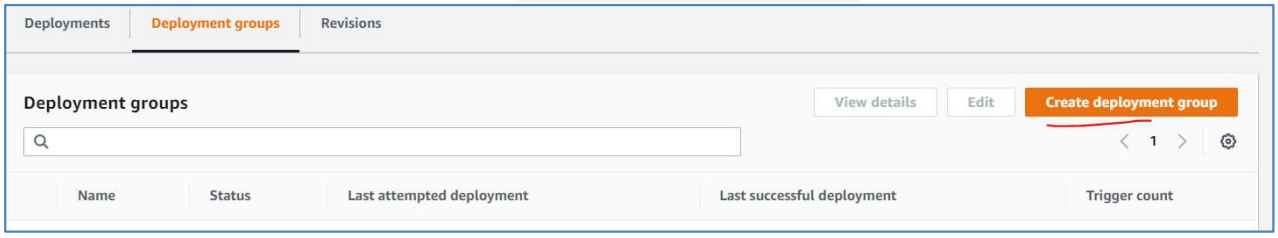


To create a deployment group in CodeDeploy

A *deployment group* is a resource that defines deployment-related settings like which instances to deploy to and how fast to deploy them.

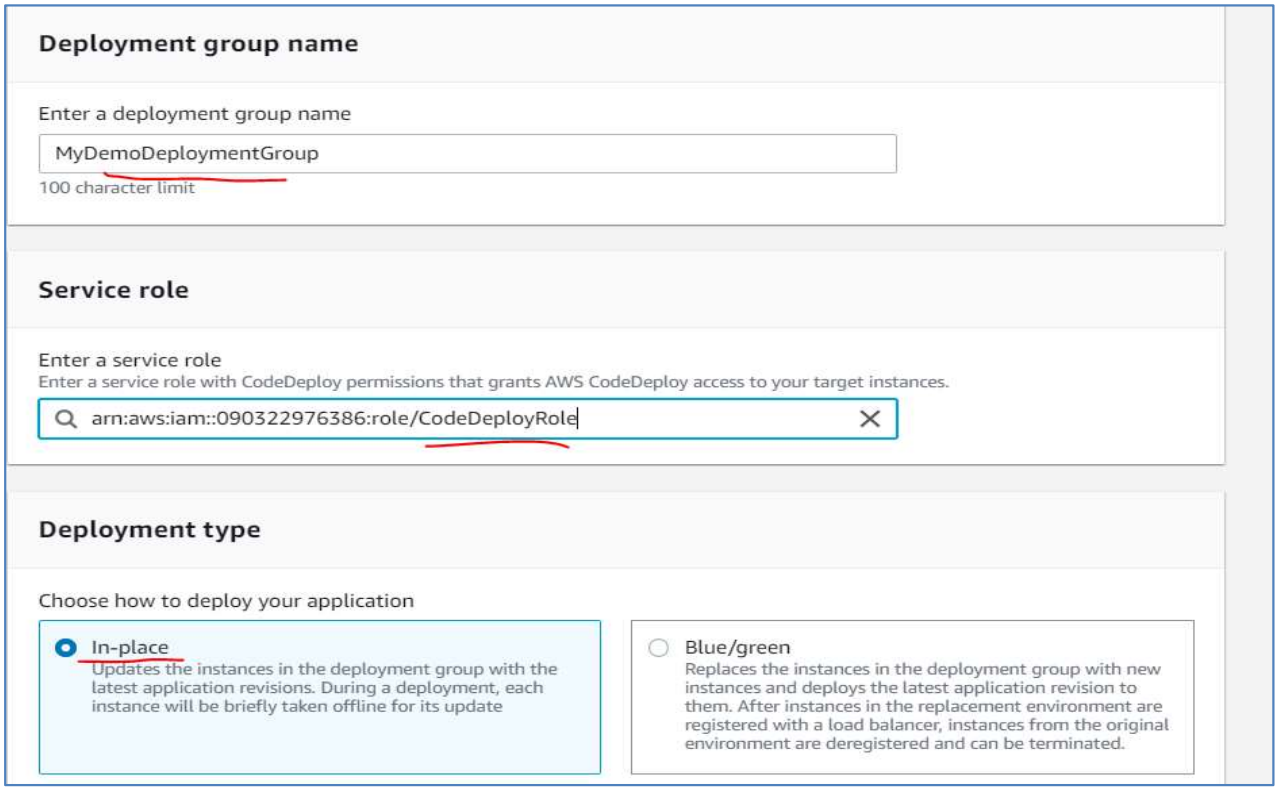
Step 1. On the page that displays your application, choose **Create deployment group**.

Step 2. In **Deployment group name**, enter **MyDemoDeploymentGroup**.



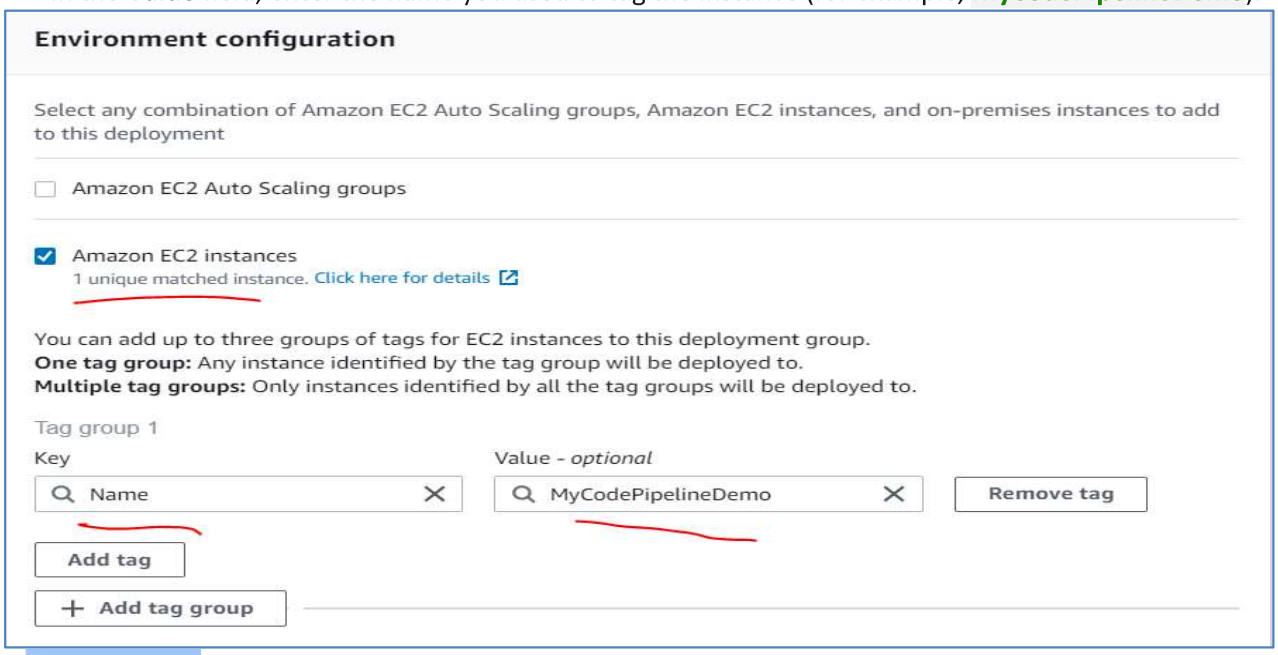
Step 3. In **Service Role**, choose the service role you created earlier (for example, **CodeDeployRole**).

Step 4. Under **Deployment type**, choose **In-place**.



Step 5. Under **Environment configuration**, choose **Amazon EC2 Instances**. In the **Key** field, enter **Name**.

In the **Value** field, enter the name you used to tag the instance (for example, **MyCodePipelineDemo**).

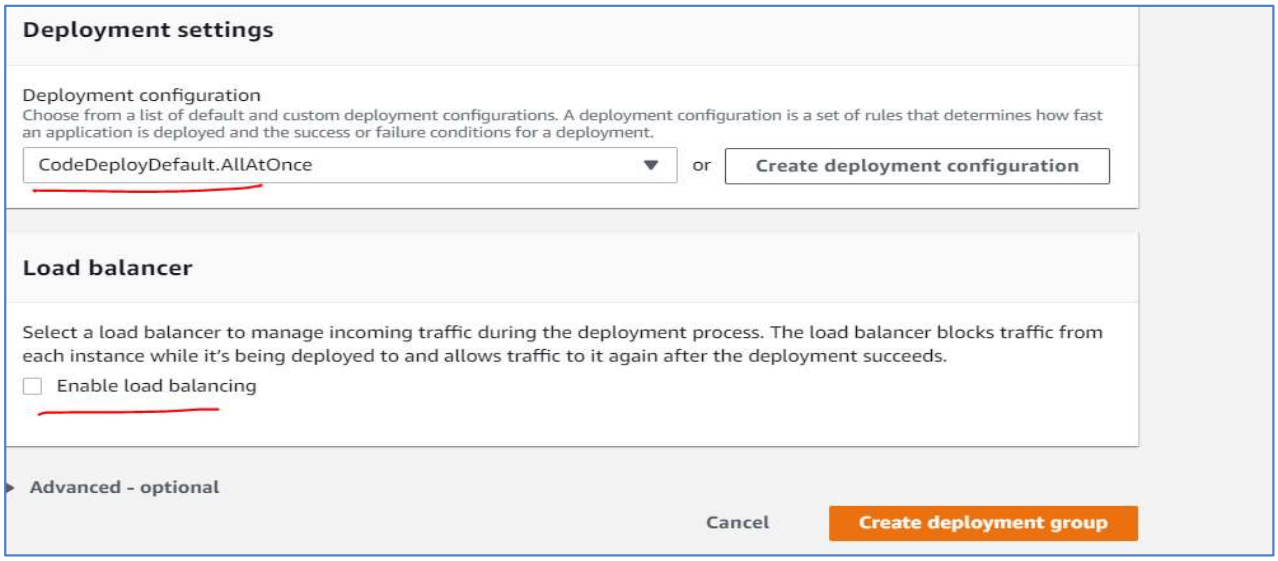


Step 6. Under **Deployment configuration**, choose CodeDeployDefault.OneAtaTime.

Step 7. Under **Load Balancer**, make sure **Enable load balancing** is not selected. You do not need to set up a load balancer or choose a target group for this example.

Step 8. Expand the **Advanced** section. Under **Alarms**, if any alarms are listed, choose **Ignore alarm configuration**.

Step 9. Choose **Create deployment group**.



Create your first pipeline in CodePipeline

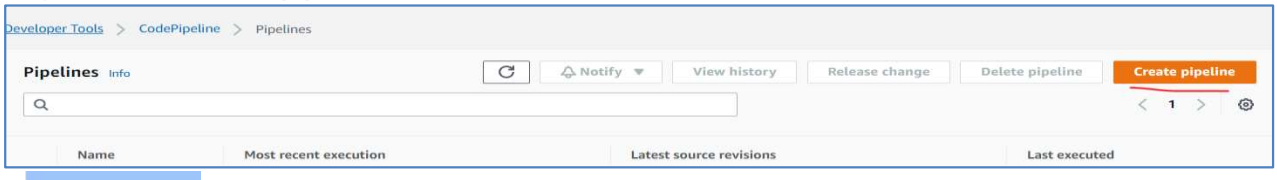
You're now ready to create and run your first pipeline. In this step, you create a pipeline that runs automatically when code is pushed to your CodeCommit repository.

To create a CodePipeline pipeline

Step 1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home or

Open the CodePipeline console at https://console.aws.amazon.com/codepipeline/.

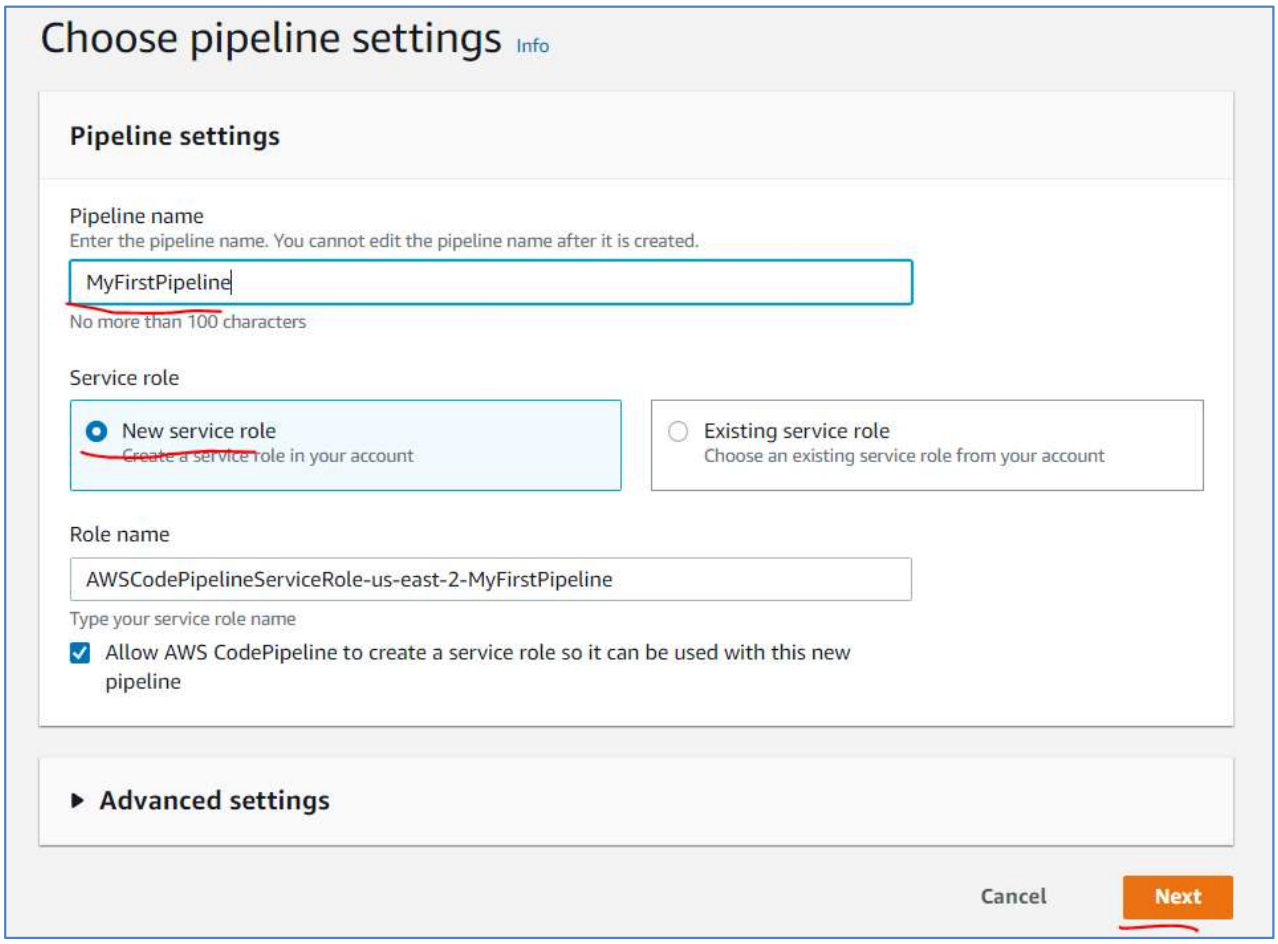
Step 2. Choose **Create pipeline**.



Step 3. In **Step 1: Choose pipeline settings**, in **Pipeline name**, enter **MyFirstPipeline**.

Step 4. In **Service role**, choose **New service role** to allow CodePipeline to create a service role in IAM.

Step 5. Leave the settings under **Advanced settings** at their defaults, and then choose **Next**.

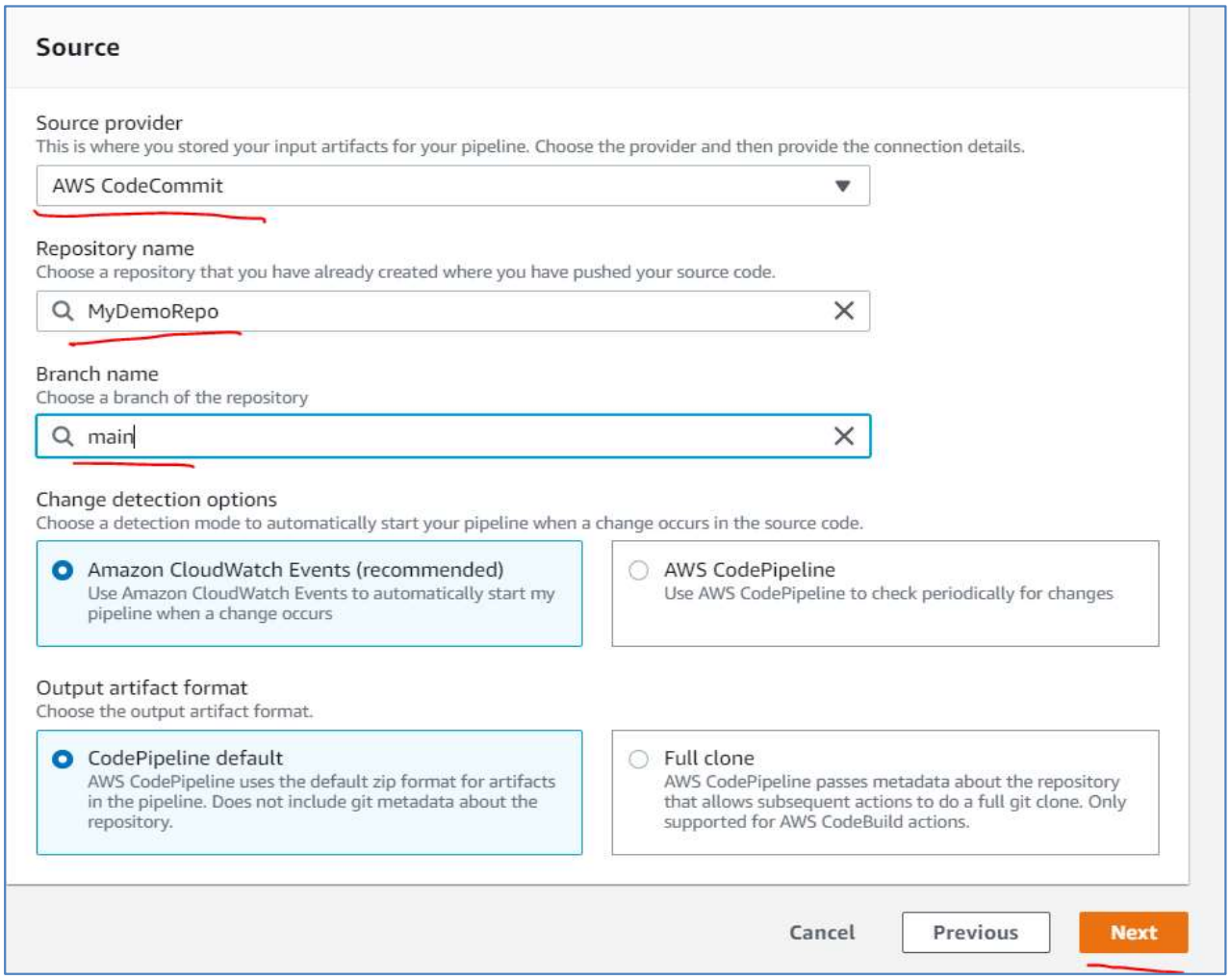


Step 6. In **Step 2: Add source stage**, in **Source provider**, choose **AWS CodeCommit**. In **Repository name**, choose the name of the CodeCommit repository you created in **Create a CodeCommit repository**. In **Branch name**, choose main, and then choose **Next step**.

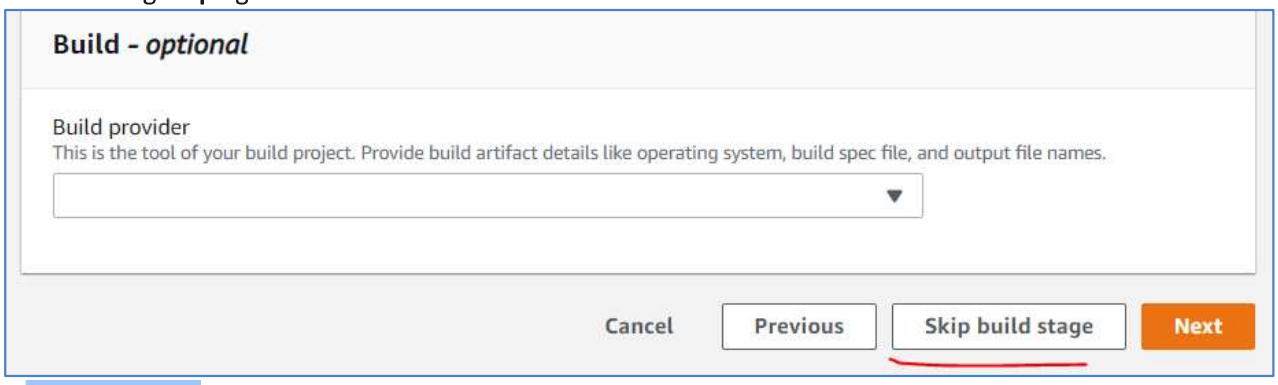
After you select the repository name and branch, a message displays the Amazon CloudWatch Events rule to be created for this pipeline.

Under **Change detection options**, leave the defaults. This allows CodePipeline to use Amazon CloudWatch Events to detect changes in your source repository.

Choose **Next**.

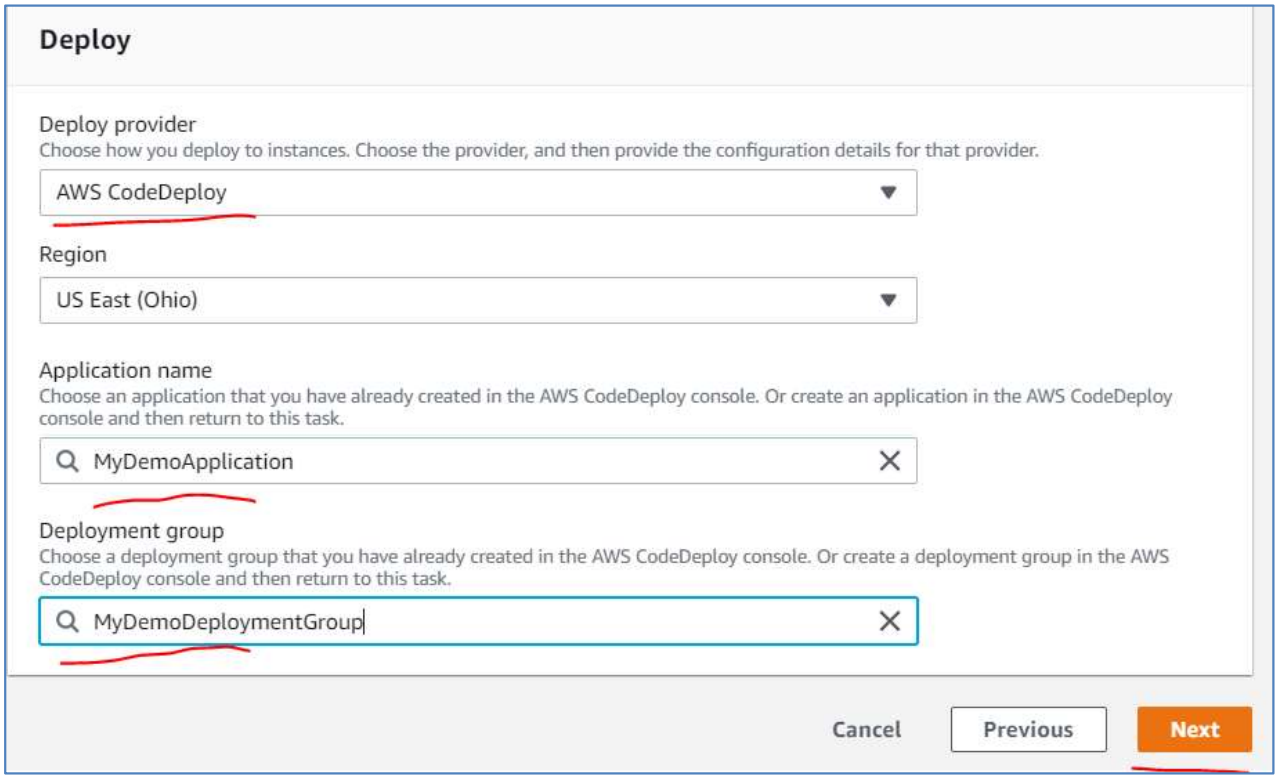


Step 7. In **Step 3: Add build stage**, choose **Skip build stage**, and then accept the warning message by choosing **Skip** again. Choose **Next**.

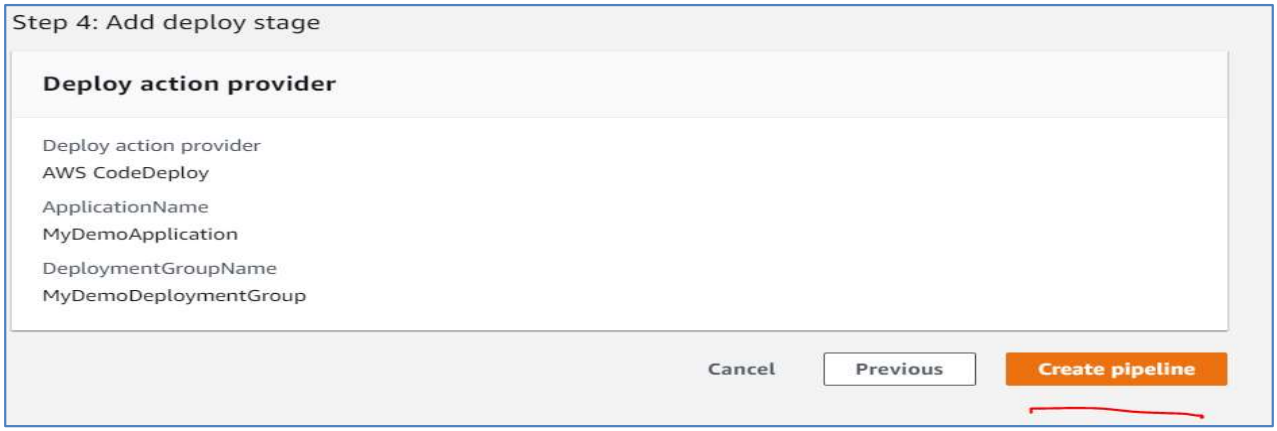


Note: In this tutorial, you are deploying code that requires no build service, so you can skip this step. However, if your source code needs to be built before it is deployed to instances, you can configure CodeBuild in this step.

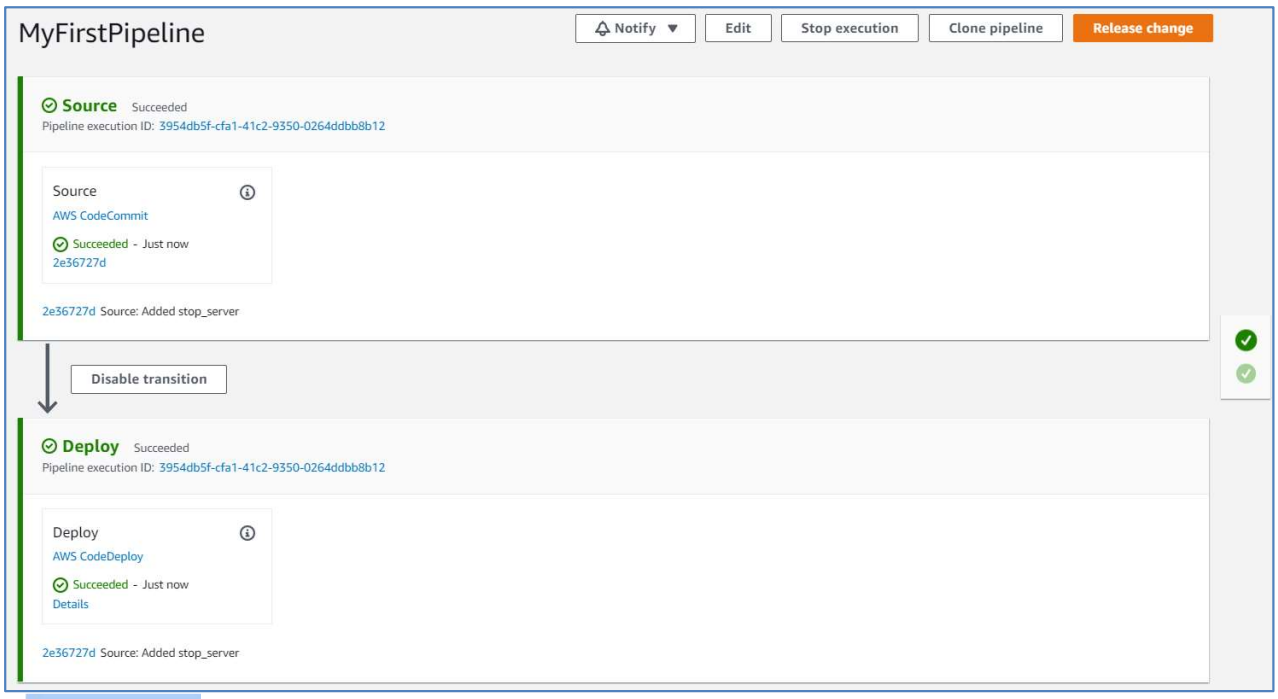
Step 8. In **Step 4: Add deploy stage**, in **Deploy provider**, choose **AWS CodeDeploy**. In **Application name**, choose **MyDemoApplication**. In **Deployment group**, choose **MyDemoDeploymentGroup**, and then choose **Next step**.



Step 9. In **Step 5: Review**, review the information, and then choose **Create pipeline**.



Step 10. The pipeline starts running after it is created. It downloads the code from your CodeCommit repository and creates a CodeDeploy deployment to your EC2 instance. You can view progress and success and failure messages as the CodePipeline sample deploys the webpage to the Amazon EC2 instance in the CodeDeploy deployment.



Congratulations! You just created a simple pipeline in CodePipeline.

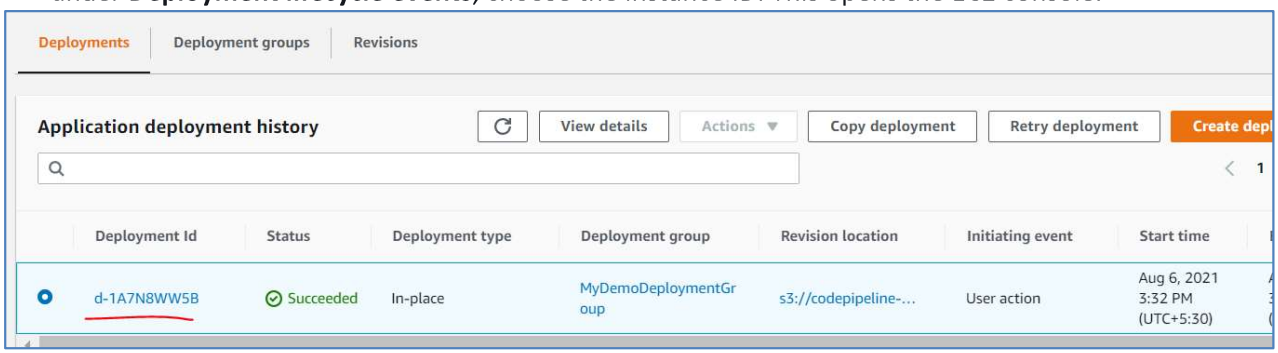
Next, you verify the results.

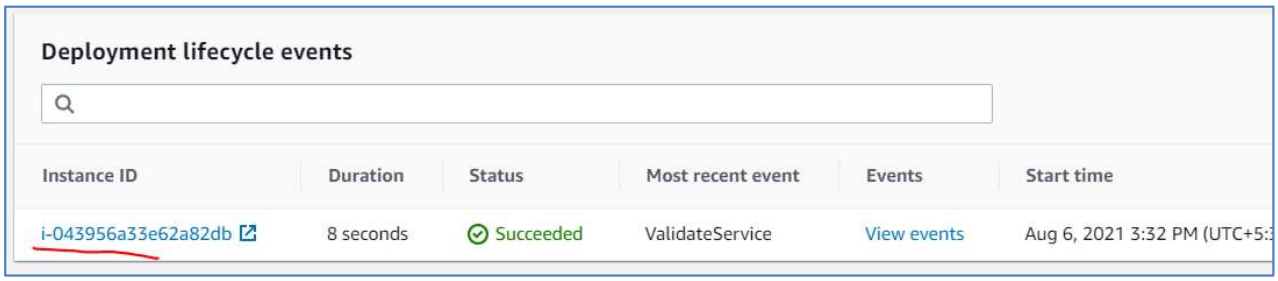
To verify that your pipeline ran successfully

Step 1. View the initial progress of the pipeline. The status of each stage changes from **No executions yet** to **In Progress**, and then to either **Succeeded** or **Failed**. The pipeline should complete the first run within a few minutes.

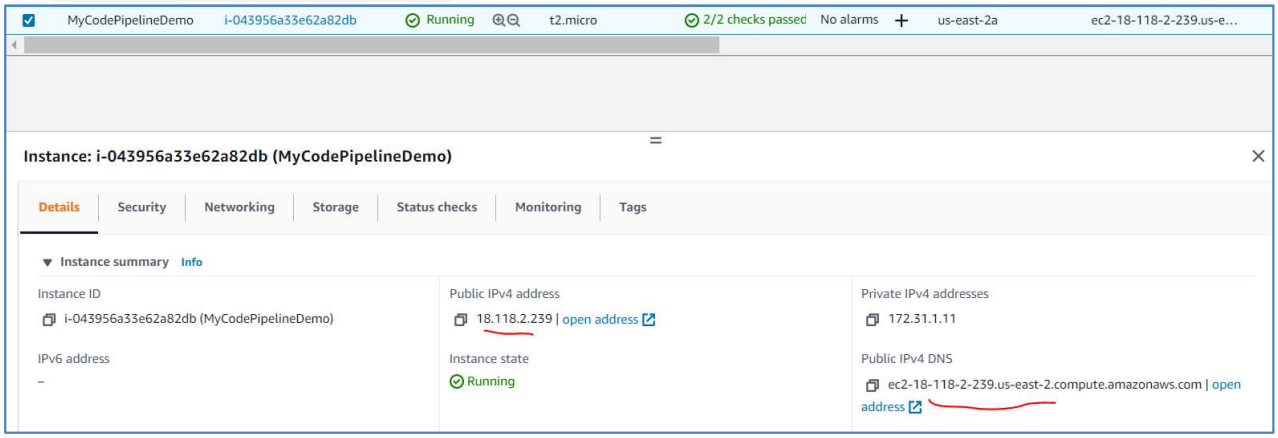
Step 2. After **Succeeded** is displayed for the pipeline status, in the status area for the **Deploy** stage, choose **AWS CodeDeploy**. This opens the CodeDeploy console.

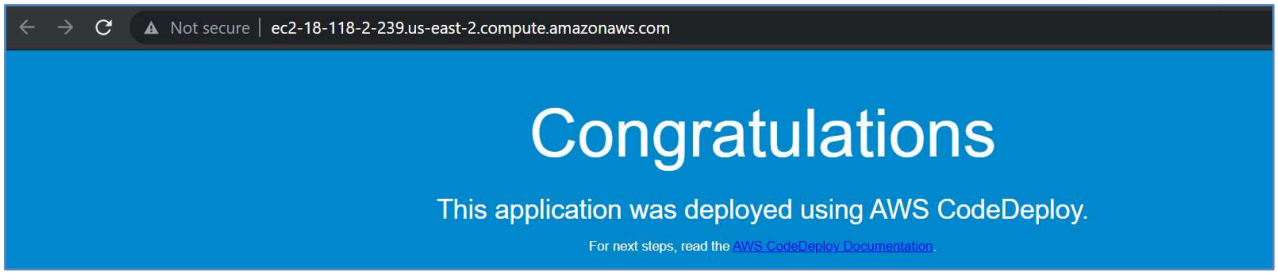
Step 3. On the **Deployments** tab, choose the deployment ID. On the page for the deployment, under **Deployment lifecycle events**, choose the instance ID. This opens the EC2 console.





Step 4. On the **Description** tab, in **Public DNS**, copy the address (for example, ec2-192-0-2-1.us-west-2.compute.amazonaws.com), and then paste it into the address bar of your web browser.





This is the sample application you downloaded and pushed to your CodeCommit repository.