



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

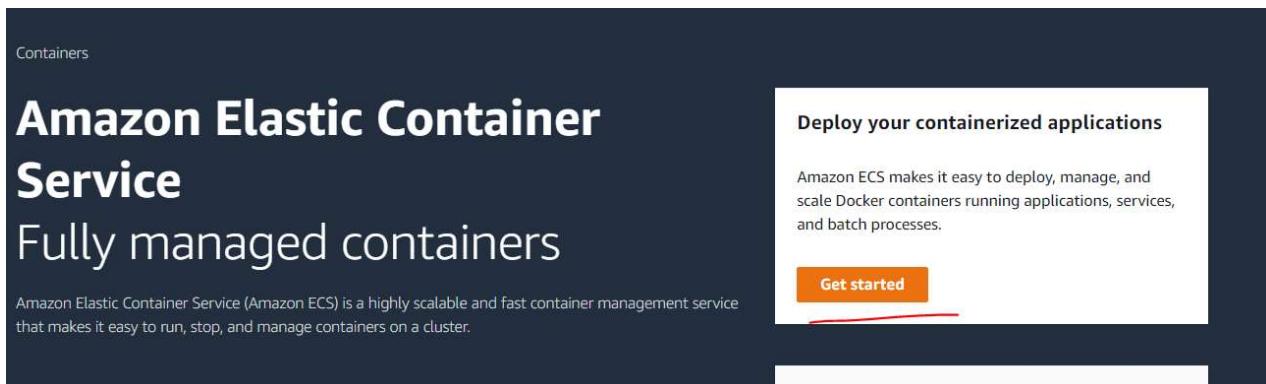
CONTAINERIZATION IN AWS

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=gsr&c=ho_ddc for ECS home page



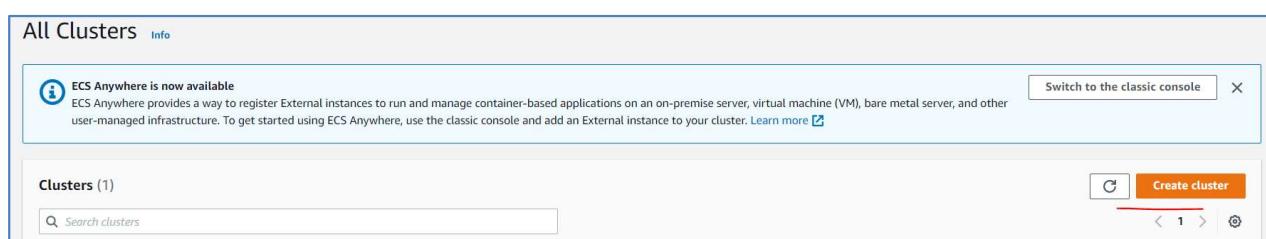
Create a Fargate Cluster.

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.

Create Cluster

Step 1: Select cluster template

Step 2: Configure cluster

Select cluster template

The following cluster templates are available to simplify cluster creation. Additional configuration and integrations can be added later.

Networking only ⓘ

Resources to be created:

- Cluster
- VPC (optional)
- Subnets (optional)

For use with either AWS Fargate or External instance capacity.

EC2 Linux + Networking ⓘ

Resources to be created:

- Cluster
- VPC
- Subnets

Auto Scaling group with Linux AMI

EC2 Windows + Networking ⓘ

Resources to be created:

- Cluster
- VPC
- Subnets

Auto Scaling group with Windows AMI

*Required

Cancel

Next step

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

Configure cluster

Cluster name* ⓘ

Networking

Create a new VPC for your cluster to use. A VPC is an isolated portion of the AWS Cloud populated by AWS objects, such as Fargate tasks.

Create VPC Create a new VPC for this cluster

Tags

Key	Value
<input type="text" value="Add key"/>	<input type="text" value="Add value"/>

CloudWatch Container Insights

CloudWatch Container Insights is a monitoring and troubleshooting solution for containerized applications and microservices. It collects, aggregates, and summarizes compute utilization such as CPU, memory, disk, and network; and diagnostic information such as container restart failures to help you isolate issues with your clusters and resolve them quickly.  [Learn more](#)

CloudWatch Container Insights Enable Container Insights

*Required

[Cancel](#)

[Previous](#)

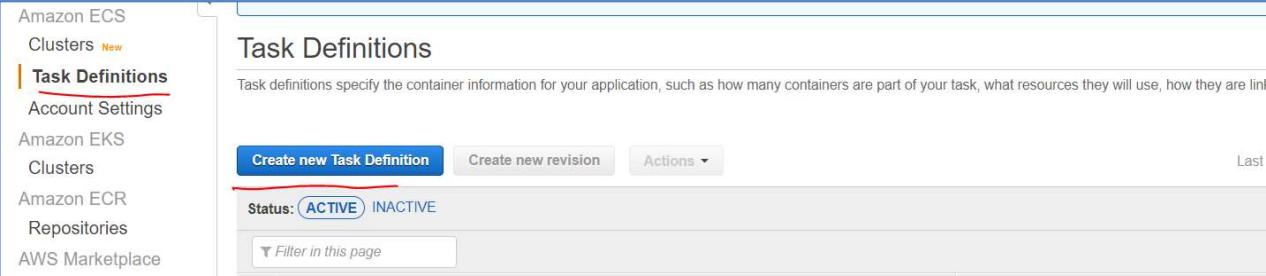
[Create](#)

Step 6. Select **Create**

The screenshot shows a success message for creating an ECS cluster. At the top, there are 'Back' and 'View Cluster' buttons. Below them, the text 'ECS status - 1 of 1 complete fargate-cluster' is displayed. A green box contains a checkmark icon and the text 'ECS cluster' followed by the message 'ECS Cluster fargate-cluster successfully created'.

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**

The screenshot shows the 'Task Definitions' page in the AWS Cloud Train interface. On the left, a sidebar lists 'Amazon ECS', 'Clusters New', 'Task Definitions' (which is selected and highlighted in red), 'Account Settings', 'Amazon EKS', 'Clusters', 'Amazon ECR', 'Repositories', 'AWS Marketplace', and 'Discover software'. The main area is titled 'Task Definitions' and contains a sub-header: 'Task definitions specify the container information for your application, such as how many containers are part of your task, what resources they will use, how they are lin'. Below this is a toolbar with 'Create new Task Definition' (highlighted in red), 'Create new revision', and 'Actions ▾'. A status filter 'Status: ACTIVE INACTIVE' is shown. At the bottom, there is a 'Filter in this page' input field and a 'Task Definitions' table.

Step 2. Select **Fargate**

Step 3. Select **Next Step**

Create new Task Definition

Step 1: Select launch type compatibility

Step 2: Configure task and container definitions

Select launch type compatibility

Select which launch type you want your task definition to be compatible with based on where you want to launch your task.

FARGATE



Price based on task size

Requires network mode awsvpc

AWS-managed infrastructure, no Amazon EC2 instances to manage

EC2



Price based on resource usage

Multiple network modes available

Self-managed infrastructure using Amazon EC2 instances

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.

Configure task and container definitions

A task definition specifies which containers are included in your task and how they interact with each other. You can also specify data volumes for your containers to use. [Learn more](#)

Task Definition Name*

myapp



Requires Compatibilities* FARGATE

Task Role

None



Optional IAM role that tasks can use to make API requests to authorized AWS services. Create an Amazon Elastic Container Service Task Role in the [IAM Console](#)

Network Mode

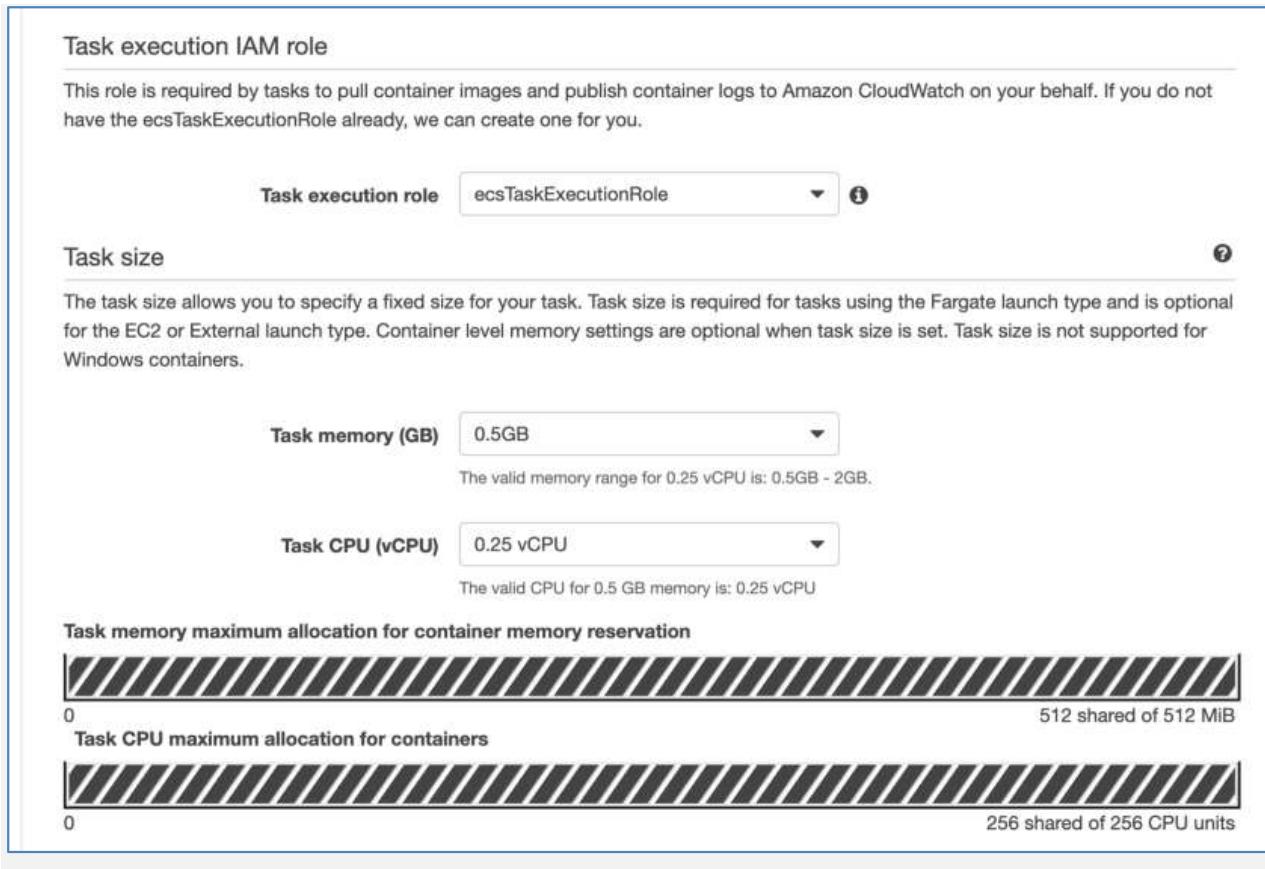
awsvpc



If you choose <default>, ECS will start your container using Docker's default networking mode, which is Bridge on Linux and NAT on Windows. <default> is the only supported mode on Windows.

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.



Task execution IAM role

This role is required by tasks to pull container images and publish container logs to Amazon CloudWatch on your behalf. If you do not have the `ecsTaskExecutionRole` already, we can create one for you.

Task execution role: `ecsTaskExecutionRole`

Task size

The task size allows you to specify a fixed size for your task. Task size is required for tasks using the Fargate launch type and is optional for the EC2 or External launch type. Container level memory settings are optional when task size is set. Task size is not supported for Windows containers.

Task memory (GB): 0.5GB

The valid memory range for 0.25 vCPU is: 0.5GB - 2GB.

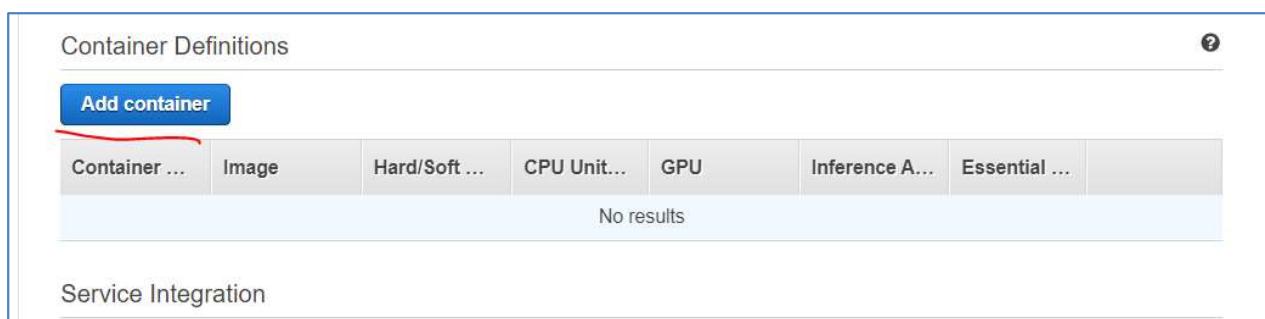
Task CPU (vCPU): 0.25 vCPU

The valid CPU for 0.5 GB memory is: 0.25 vCPU

Task memory maximum allocation for container memory reservation: 512 shared of 512 MiB

Task CPU maximum allocation for containers: 256 shared of 256 CPU units

Step 8. Under Container definition select Add Container.



Container Definitions

Add container

Container ...	Image	Hard/Soft ...	CPU Unit...	GPU	Inference A...	Essential ...
No results						

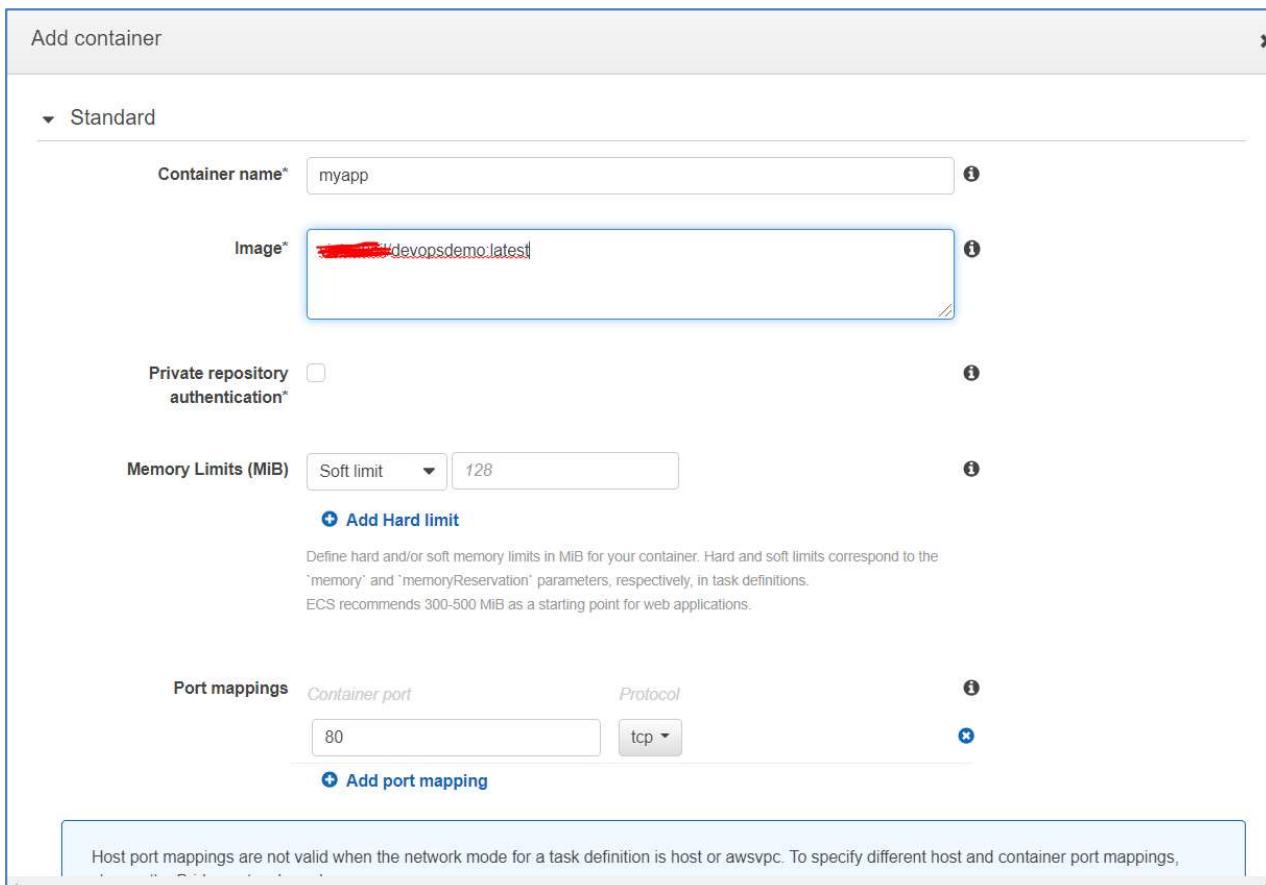
Service Integration

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.



The screenshot shows the 'Add container' dialog for AWS Lambda. The container name is set to 'myapp'. The image field contains the ARN of the Docker image, which has been redacted. The memory limit is set to 128 MB. Under 'Port mappings', a single mapping is defined with a container port of 80 and a protocol of TCP. A note at the bottom states that host port mappings are not valid when the network mode is 'host' or 'awsvpc'.

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.

Task definition status - 2 of 2 completed

Create Task Definition: myapp

myapp succeeded

Create CloudWatch Log Group

CloudWatch Log Group created
CloudWatch Log Group /ecs/myapp

[Back](#) [View task definition](#)

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

New ECS Experience [Tell us what you think](#)

Amazon ECS Clusters

Task Definitions

Account Settings

Amazon EKS Clusters

Amazon ECR Repositories

AWS Marketplace Discover software Subscriptions

Task Definitions

Task definitions specify the container information for your application, such as how many containers are part of your task, what resources they will use, how they are linked together, and which host ports they will use. [Learn more](#)

Create new Task Definition Create new revision Actions Last updated on July 3, 2021 5:57:41 PM (0m ago) [Edit](#) [Help](#)

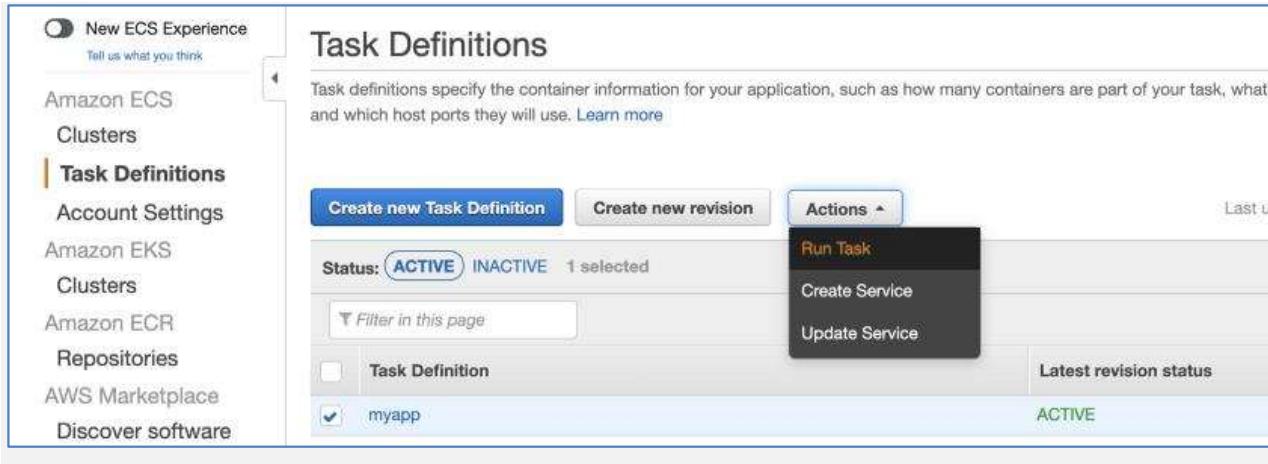
Status: ACTIVE INACTIVE	Filter in this page	< 1-1 > Page size 50
<input type="checkbox"/> Task Definition		Latest revision status
<input type="checkbox"/> myapp		ACTIVE

Run the ECS Task

This is the moment we have all been waiting for.

Step 1. Select the task in the Task definition list

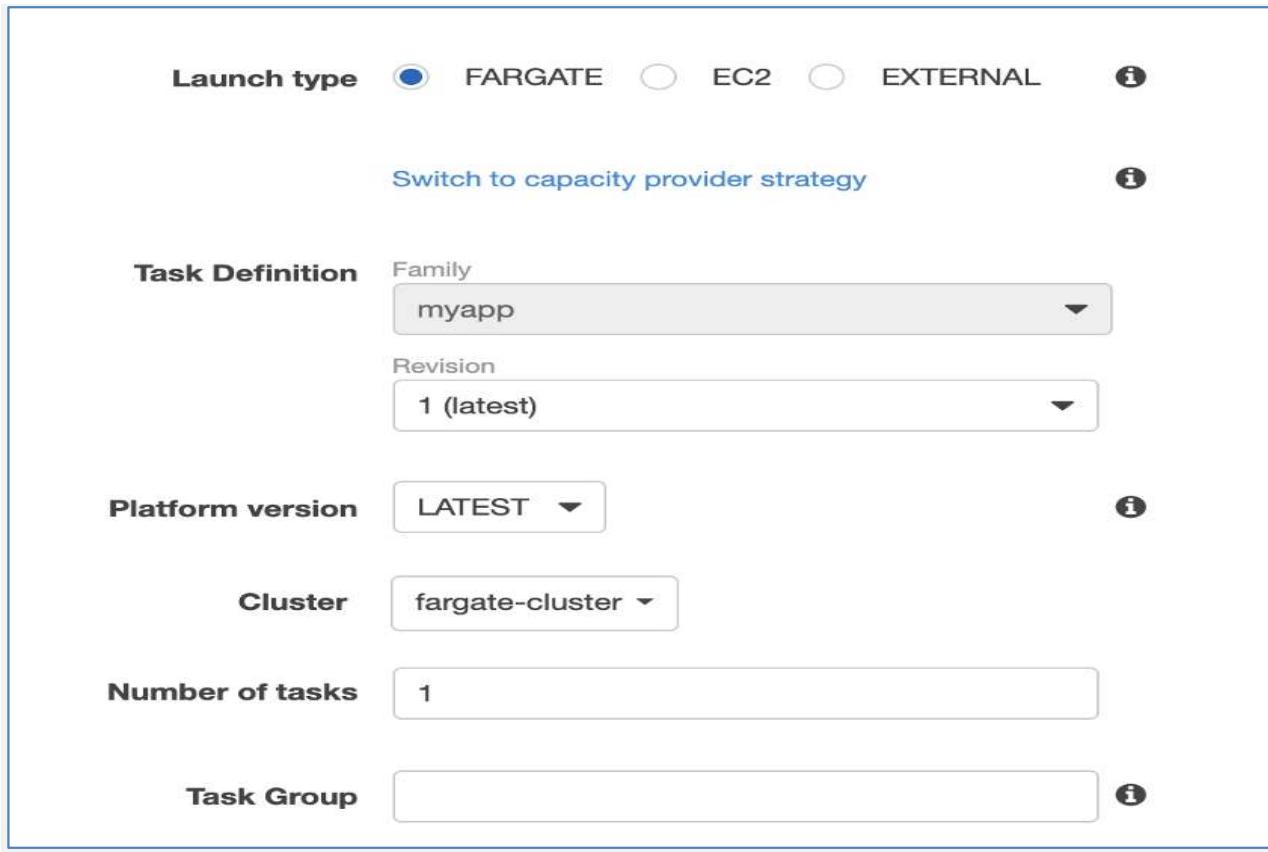
Step 2. Click Actions and select Run Task



The screenshot shows the AWS ECS Task Definitions page. On the left, there's a sidebar with links like 'New ECS Experience', 'Amazon ECS Clusters', 'Task Definitions' (which is selected), 'Account Settings', 'Amazon EKS Clusters', 'Amazon ECR Repositories', 'AWS Marketplace', and 'Discover software'. The main content area has a title 'Task Definitions' and a sub-instruction: 'Task definitions specify the container information for your application, such as how many containers are part of your task, what and which host ports they will use. Learn more'. Below this is a table with columns for 'Status' (ACTIVE), 'Actions' (dropdown menu), and 'Last updated'. The dropdown menu is open, showing 'Run Task' (highlighted in yellow), 'Create Service', and 'Update Service'. The table also includes a 'Filter in this page' input field and checkboxes for 'Task Definition' and 'myapp'. A status bar at the bottom right says 'Latest revision status ACTIVE'.

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

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



The screenshot shows the AWS Fargate Launch Configuration page. It has several fields:

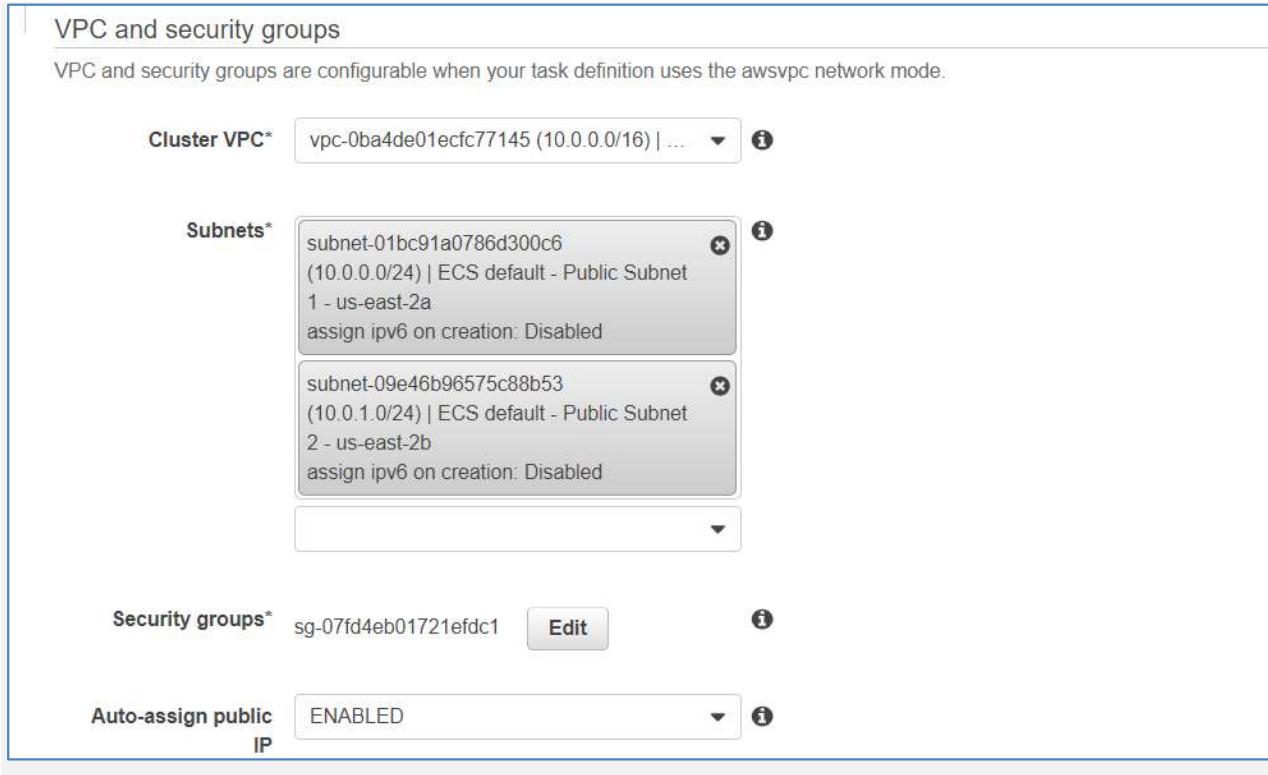
- Launch type:** FARGATE (radio button selected)
- Task Definition:** Family dropdown set to 'myapp'
- Platform version:** LATEST (dropdown)
- Cluster:** dropdown set to 'fargate-cluster'
- Number of tasks:** input field set to '1'
- Task Group:** input field (empty)

Each field has an informational icon (info icon) to its right.

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 **ENABLED**



VPC and security groups

VPC and security groups are configurable when your task definition uses the awsvpc network mode.

Cluster VPC* vpc-0ba4de01ecfc77145 (10.0.0.0/16) | ... 

Subnets*

- subnet-01bc91a0786d300c6 (10.0.0.0/24) | ECS default - Public Subnet
1 - us-east-2a
assign ipv6 on creation: Disabled 
- subnet-09e46b96575c88b53 (10.0.1.0/24) | ECS default - Public Subnet
2 - us-east-2b
assign ipv6 on creation: Disabled 

Security groups* sg-07fd4eb01721efdc1  

Auto-assign public IP **ENABLED** 

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.

VPC and security groups

VPC and security groups are configurable when your task definition uses the awsvpc network mode.

Cluster VPC* vpc-0ba4de01ecfc77145 (10.0.0.0/16) | ... 

Subnets*

- subnet-01bc91a0786d300c6 (10.0.0.0/24) | ECS default - Public Subnet
1 - us-east-2a
assign ipv6 on creation: Disabled
- subnet-09e46b96575c88b53 (10.0.1.0/24) | ECS default - Public Subnet
2 - us-east-2b
assign ipv6 on creation: Disabled

Security groups* sg-07fd4eb01721efdc1  

Auto-assign public IP ENABLED 

Inbound rules for security group

Type	Protocol	Port range	Source
HTTP	TCP	80	Anywhere



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

Tasks (1)

Task	Last status	Description	Task definition	Revision	Started by	Started at	Group	Container instances	Launch type
f9e1b591...	 Running	-	myapp	1	-	1 minute ago	family:my...	-	FARGATE



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

f9e1b59123ef432e980d1d4b14c1b489

Configuration	Logs	Networking	Tags
Network			
ENI ID eni-0c8add2fb36971628	Task role -	Private IP 10.0.0.98	
Subnet subnet-01bc91a0786d300c6	Task execution role ecsTaskExecutionRole	Public IP 18.191.200.99	IPv6 address -
		MAC address 02:43:db:19:c2:5a	

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.

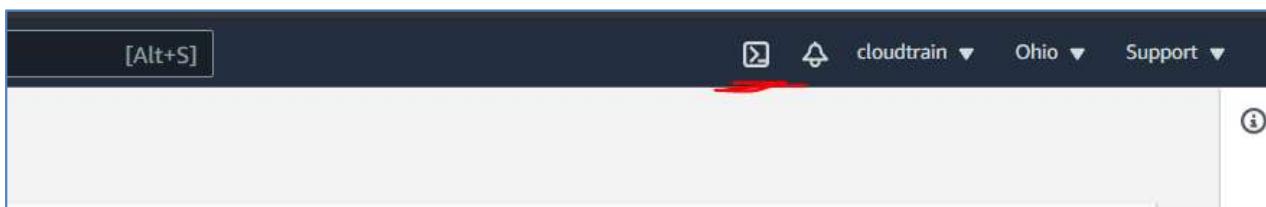


Congratulations!! You have successfully deployed a Website on ECS containers using Fargate cluster.

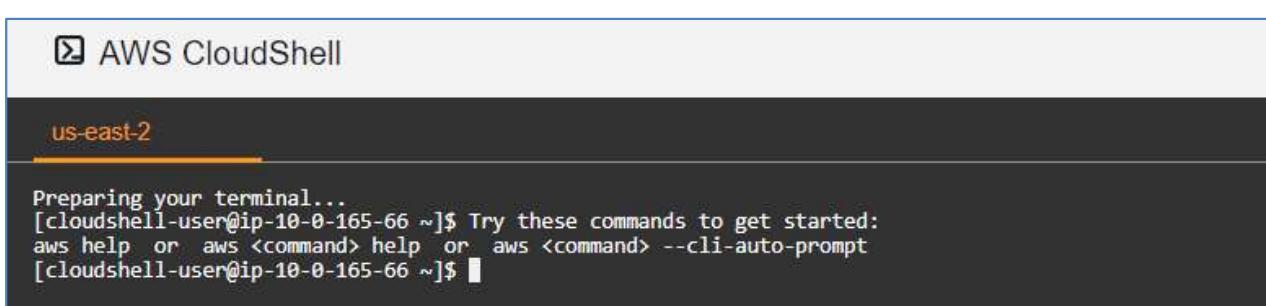
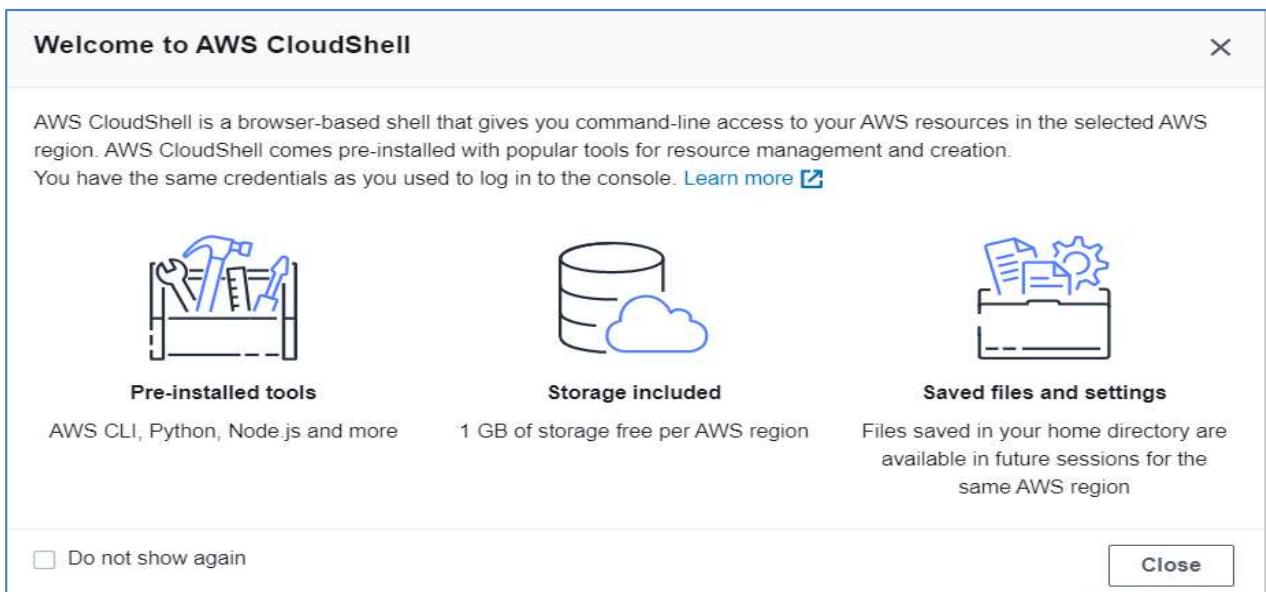
Setup an EKS cluster and deploy a Kubernetes application on it

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  Current  
          Dload  Upload   Total Spent   Left  Speed  
100 44.2M  100 44.2M    0     0  24.7M  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+d2965f0db1071203c6f5bc662c2827c71fc8b20d", GitTreeHash:"d2965f0db1071203c6f5bc662c2827c71fc8b20d", GitTreeHashSize:32, BuildDate:"2021-07-05T14:40:20Z", GoVersion:"go1.16.10+b4d5ca9", Compiler:"gc", Platform:"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.

- 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-policy-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": "ARDARKB5LH2BPR2MEOOKSD",
    "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"
        }
      ]
    }
  }
}
```

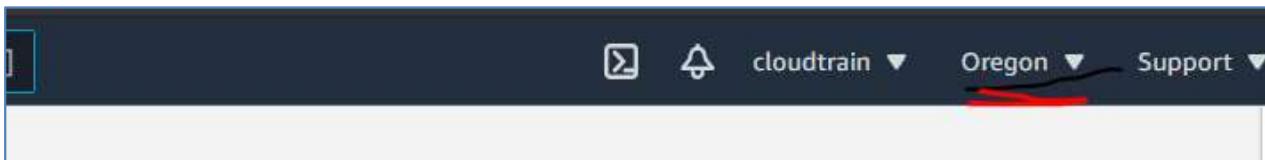
c. Attach the required Amazon EKS managed IAM policy to the role.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy --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**.

Configure cluster

Cluster configuration Info

Name - *Not editable after creation.*
Enter a unique name for this cluster.

Kubernetes version Info
Select the Kubernetes version for this cluster.

Cluster Service Role Info - *Not editable after creation.*
Select the IAM Role to allow the Kubernetes control plane to manage AWS resources on your behalf.
To create a new role, go to the [IAM console](#).

- 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**.

Networking Info

These properties cannot be changed after the cluster is created.

VPC Info

Select a VPC to use for your EKS Cluster resources.
To create a new VPC, go to the [VPC console](#).

Subnets Info

Choose the subnets in your VPC where the control plane may place elastic network interfaces (ENIs) to facilitate communication with your cluster.
To create a new subnet, go to the corresponding page in the [VPC console](#).

XXXX

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

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

Step 5: Configure logging

Control Plane Logging

API server	Audit	Authenticator
Disabled	Disabled	Disabled
Controller manager	Scheduler	
Disabled	Disabled	

Create 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.

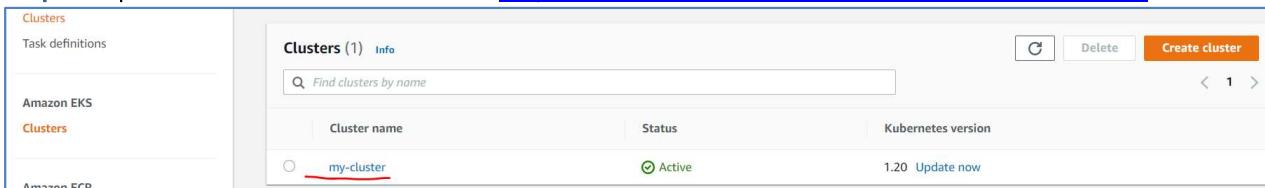
```
aws iam create-role --role-name myAmazonEKSFargatePodExecutionRole --assume-role-policy-document file:///"pod-execution-role-trust-policy.json"
```

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

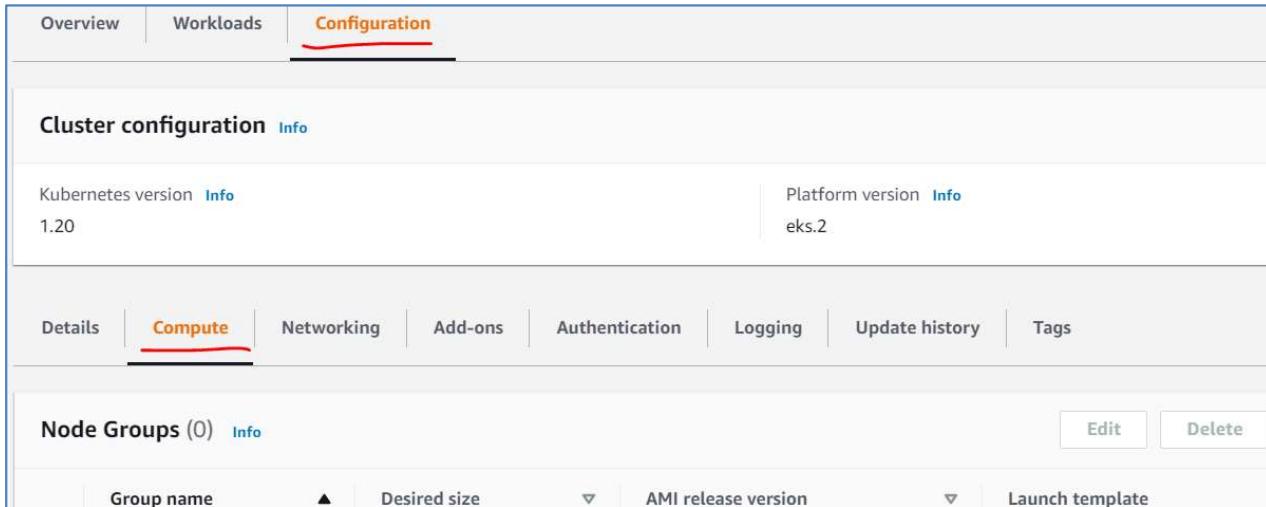
```
[cloudshell-user@ip-10-0-165-66 ~]$ aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/AmazonEKSFargatePodExecutionRolePolicy --role-name myAmazonEKSFargatePodExecutionRole
[cloudshell-user@ip-10-0-165-66 ~]$
```

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


The screenshot shows the AWS EKS Clusters page. On the left, there's a sidebar with 'Clusters' selected. The main area has a heading 'Clusters (1)'. Below it is a search bar with placeholder text 'Find clusters by name'. A table lists one cluster: 'Cluster name' is 'my-cluster', 'Status' is 'Active', and 'Kubernetes version' is '1.20 Update now'. There are buttons for 'Create cluster' and 'Delete' at the top right, and navigation arrows at the bottom right.

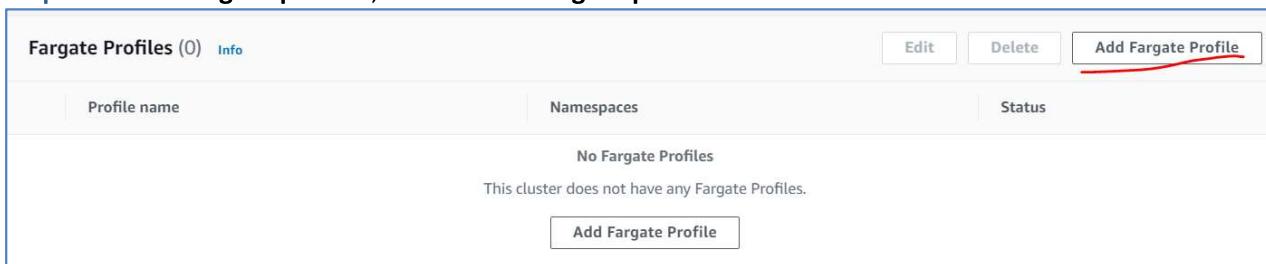
Cluster name	Status	Kubernetes version
my-cluster	Active	1.20 Update now

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



The screenshot shows the AWS EKS Cluster Configuration page. The top navigation bar has tabs for Overview, Workloads, and Configuration, with Configuration being the active tab. Below the tabs, the section title is "Cluster configuration" with an "Info" link. Underneath, it shows the Kubernetes version as 1.20 and the Platform version as eks.2. A horizontal navigation bar below these details includes tabs for Details, Compute (which is highlighted with a red underline), Networking, Add-ons, Authentication, Logging, Update history, and Tags. The Compute tab is currently selected. Below this, there is a section titled "Node Groups (0)" with an "Info" link, followed by a table header row with columns for Group name, Desired size, AMI release version, and Launch template. There are "Edit" and "Delete" buttons at the top right of this section.

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



The screenshot shows the Fargate Profiles page. At the top, it displays "Fargate Profiles (0)" with an "Info" link, and buttons for Edit, Delete, and Add Fargate Profile (which is highlighted with a red underline). Below this, there is a table with columns for Profile name, Namespaces, and Status. A message states "No Fargate Profiles" and "This cluster does not have any Fargate Profiles." At the bottom, there is a single "Add Fargate Profile" button.

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.

Profile configuration

These properties cannot be changed after the profile is created.

Name
Assign a unique name for this profile.

Pod execution role [Info](#)
Select the IAM role that will be used by Fargate to connect to the cluster and pull container images.
To create a new role, go to the [IAM console](#).

Subnets [Info](#)
Specify the subnets in your VPC where your pods will run.
To create a new subnet, go to the corresponding page in the [VPC console](#).

Select subnets ▾

Filter subnets

<input type="checkbox"/> subnet-0185c46e9ebedaaf my-eks-vpc-stack-PublicSubnet01 us-west-2a 192.168.0.0/18
<input type="checkbox"/> subnet-0b6af9661de07ab31 my-eks-vpc-stack-PublicSubnet02 us-west-2b 192.168.64.0/18
<input checked="" type="checkbox"/> subnet-0be2f03869cefdd my-eks-vpc-stack-PrivateSubnet01 us-west-2a 192.168.128.0/18
<input checked="" type="checkbox"/> subnet-0be603c8e38de5e5d my-eks-vpc-stack-PrivateSubnet02 us-west-2b 192.168.192.0/18

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

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

Configure pod selection

Add selectors to define which pods you want to run on Fargate.

Pod selectors [Info](#)

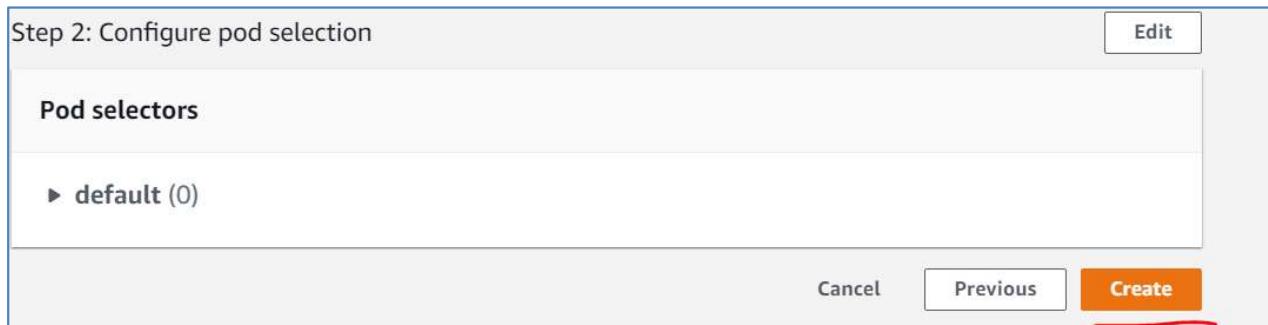
These properties cannot be changed after the profile is created.

Namespace

Add a namespace to select pods from.

► Match labels (0)

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



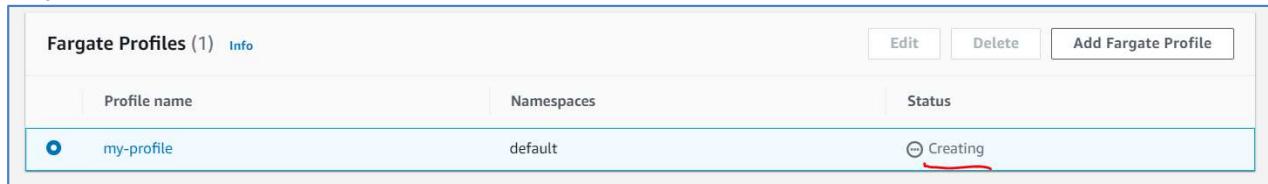
Step 2: Configure pod selection

Pod selectors

- ▶ default (0)

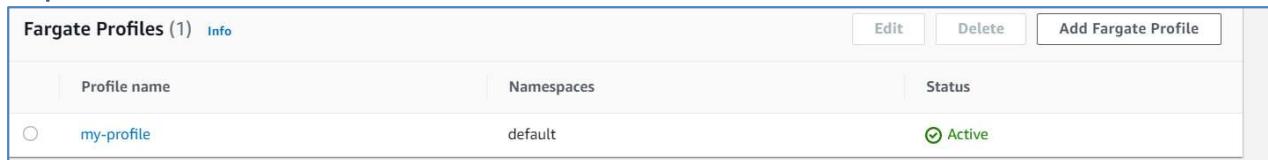
Cancel Previous **Create**

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



Fargate Profiles (1) Info	Edit	Delete	Add Fargate Profile
Profile name	Namespaces	Status	
my-profile	default	Creating	

Step 12. Wait until it becomes active:



Fargate Profiles (1) Info	Edit	Delete	Add Fargate Profile
Profile name	Namespaces	Status	
my-profile	default	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 won't 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 won't 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.

- 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
  ---
```

```
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-tmwsd          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
```

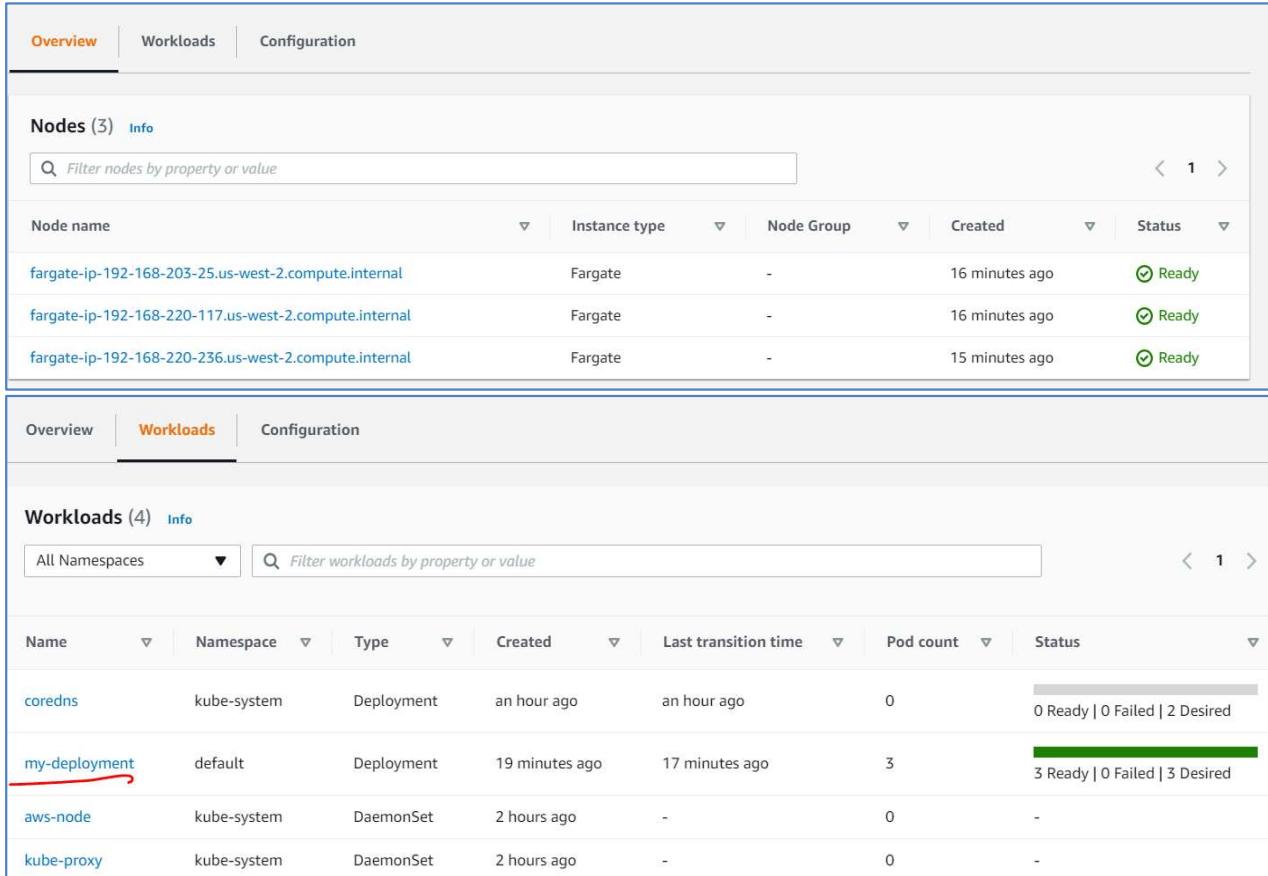
Output

```
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:



The screenshot shows two tabs of the AWS CloudWatch Metrics Insights interface:

- Overview**: Shows 3 nodes.
- Workloads**: Shows 4 workloads.

Nodes (3) Info

Node name	Instance type	Node Group	Created	Status
fargate-ip-192-168-203-25.us-west-2.compute.internal	Fargate	-	16 minutes ago	Ready
fargate-ip-192-168-220-117.us-west-2.compute.internal	Fargate	-	16 minutes ago	Ready
fargate-ip-192-168-220-236.us-west-2.compute.internal	Fargate	-	15 minutes ago	Ready

Workloads (4) Info

Name	Namespace	Type	Created	Last transition time	Pod count	Status
coredns	kube-system	Deployment	an hour ago	an hour ago	0	0 Ready 0 Failed 2 Desired
<u>my-deployment</u>	default	Deployment	19 minutes ago	17 minutes ago	3	3 Ready 0 Failed 3 Desired
aws-node	kube-system	DaemonSet	2 hours ago	-	0	-
kube-proxy	kube-system	DaemonSet	2 hours ago	-	0	-

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

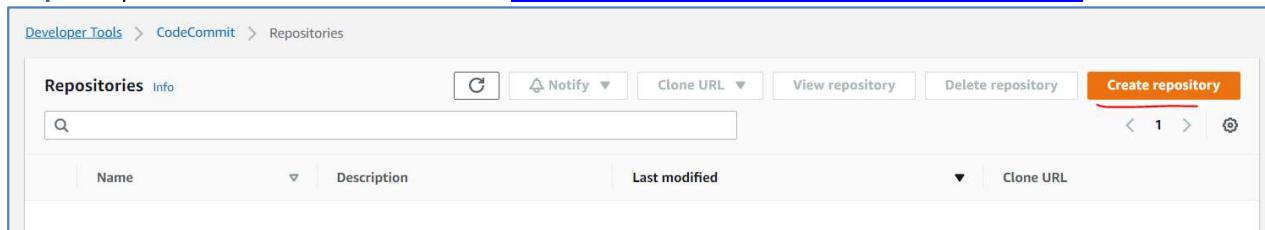
Before you begin, make sure you've set up your Git client to work with CodeCommit. For instructions, see [Setting up for CodeCommit](#).

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.

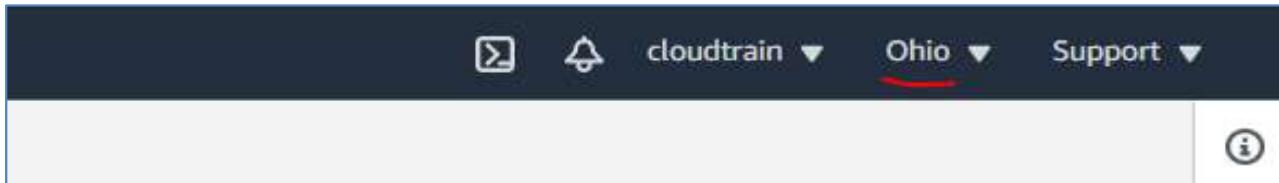
To create a CodeCommit repository

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



The screenshot shows the AWS CodeCommit console interface. At the top, there's a navigation bar with 'Developer Tools > CodeCommit > Repositories'. Below the navigation is a search bar and a 'Create repository' button, which is highlighted with a red border. The main area displays a table with columns for 'Name', 'Description', 'Last modified', and 'Clone URL'. There are no repositories listed in the table.

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**.

Create repository

Create a secure repository to store and share your code. Begin by typing a repository name and a description for your repository. Repository names are included in the URLs for that repository.

Repository settings

Repository name

100 characters maximum. Other limits apply.

Description - *optional*

1,000 characters maximum

Tags

[Add](#)

Enable Amazon CodeGuru Reviewer for Java and Python - *optional*

Get recommendations to improve the quality of the Java and Python code for all pull requests in this repository.

A service-linked role will be created in IAM on your behalf if it does not exist.

[Cancel](#) [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

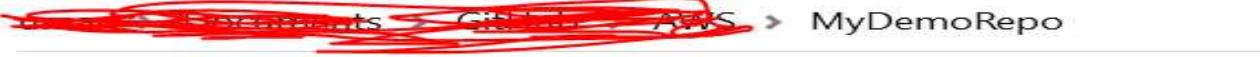
In this step, you download code for a sample application that was created for a CodeDeploy sample walkthrough, and add it to your CodeCommit repository.

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:



The screenshot shows a file listing for a repository named "MyDemoRepo". The files listed are:

Name	Date modified
scripts	06-08-2021 14:34
appspec.yml	06-08-2021 14:34
index.html	06-08-2021 14:34
LICENSE.txt	06-08-2021 14:34

Below the table, the directory structure is shown:

```
└-- 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**.



The screenshot shows the "MyDemoRepo" repository page. The repository is currently empty. The "Add file" button is highlighted.

Empty repository

Your repository is currently empty. You can add files to it directly from the console or by cloning the repository to your local computer, creating commits, and pushing content to the remote repository in AWS CodeCommit.

Create 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

MyDemoRepo/scripts/install_dependencies

Author name

Email address

Upload a file

MyDemoRepo [Info](#)

Name	Size	Actions
------	------	---------

Upload file

Choose a file to upload.

AWS Foundation Workshop

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

MyDemoRepo [Info](#)

Name	Size	Actions
appspec.yml	1 KB	Remove file

Commit changes to main
File: MyDemoRepo/appspec.yml

Author name

Email address

Commit message - optional
A default commit message will be used if you do not provide one.

[Cancel](#) [Commit changes](#)

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

MyDemoRepo

[Notify](#)
main
[Create pull request](#)

MyDemoRepo [Info](#)

Name
scripts
appspec.yml
index.html
LICENSE.txt

```

|-- 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):

AWS Foundation Workshop

- 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/>).

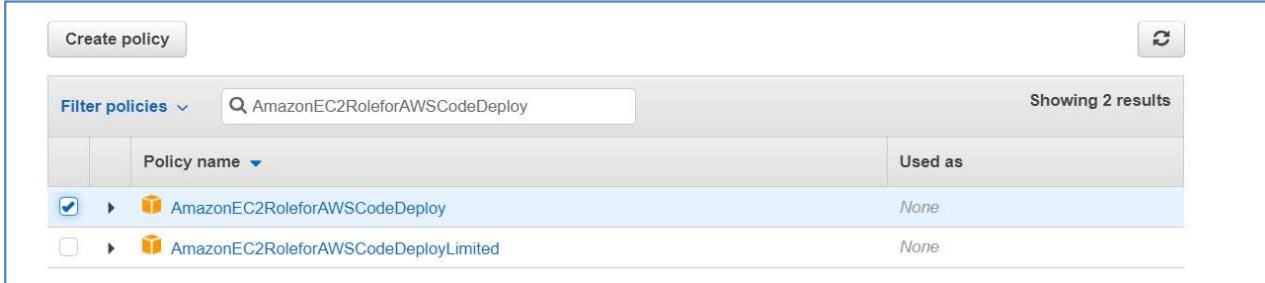
Step 2. From the console dashboard, choose **Roles**.

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**.



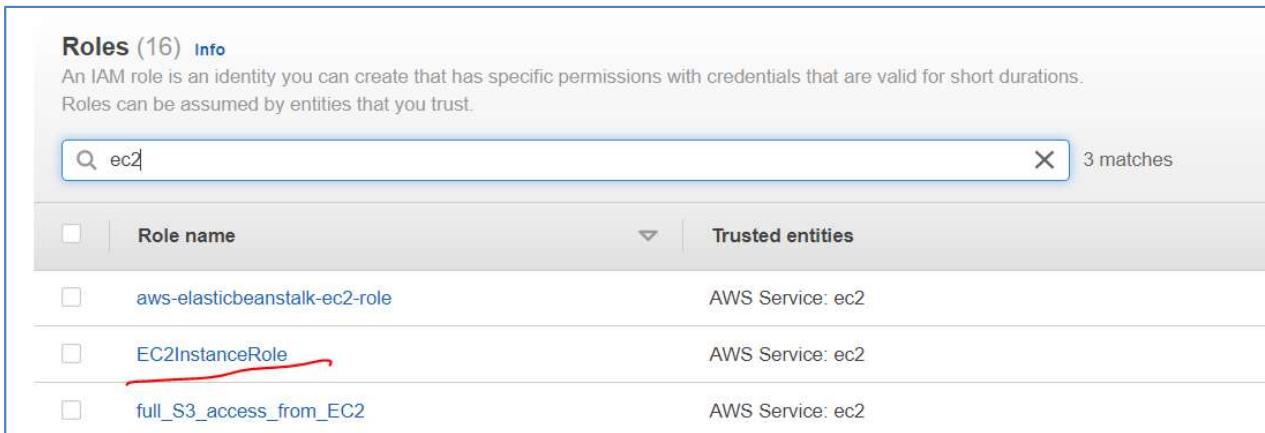
The screenshot shows a search interface for AWS IAM policies. A search bar at the top contains the text "AmazonEC2RoleforAWSCodeDeploy". Below the search bar, a table lists two results:

Policy name	Used as
<input checked="" type="checkbox"/> AmazonEC2RoleforAWSCodeDeploy	None
<input type="checkbox"/> AmazonEC2RoleforAWSCodeDeployLimited	None

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**.



The screenshot shows a list of IAM roles. A search bar at the top contains the text "ec2". The list includes:

Role name	Trusted entities
aws-elasticbeanstalk-ec2-role	AWS Service: ec2
<u>EC2InstanceRole</u>	AWS Service: ec2
full_S3_access_from_EC2	AWS Service: ec2

To launch an instance

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

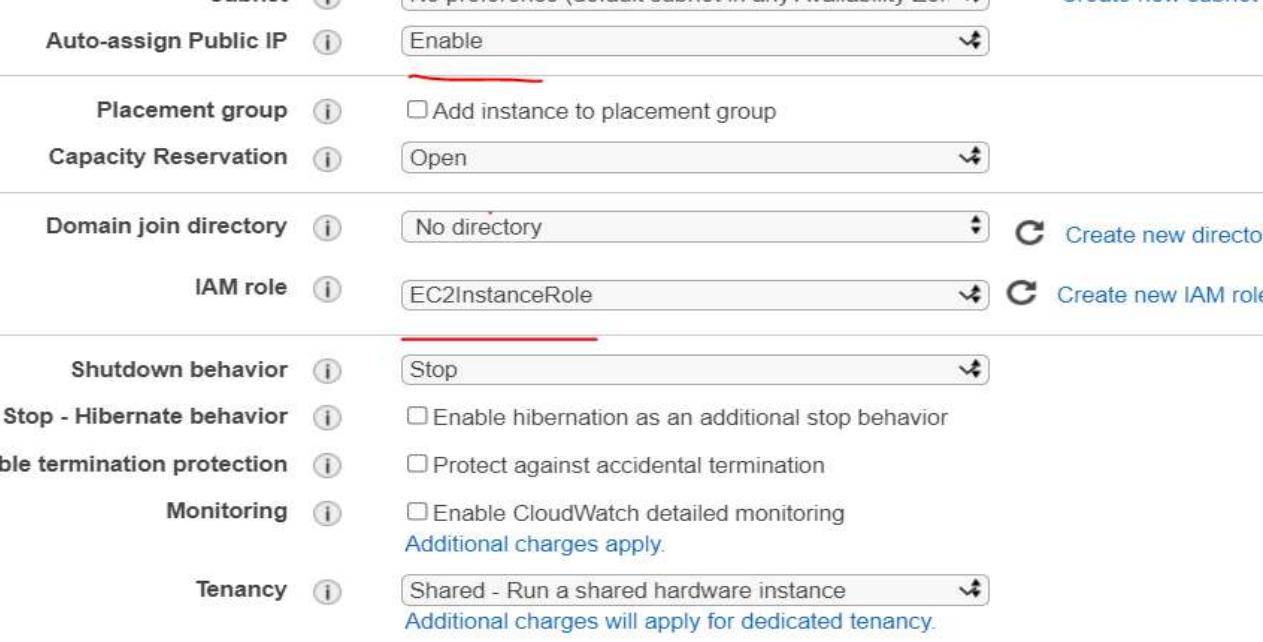
Step 2. From the console dashboard, choose **Launch instance**, and select **Launch instance** from the options that pop up.

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**).



The screenshot shows the 'Configure Instance Details' step of an AWS wizard. It lists several configuration options with dropdown menus and checkboxes:

- Auto-assign Public IP:** Set to **Enable**.
- Placement group:** Has a checkbox for 'Add instance to placement group' which is unchecked.
- Capacity Reservation:** Set to **Open**.
- Domain join directory:** Set to **No directory**.
- IAM role:** Set to **EC2InstanceRole**.
- Shutdown behavior:** Set to **Stop**.
- Stop - Hibernate behavior:** Has a checkbox for 'Enable hibernation as an additional stop behavior' which is unchecked.
- Termination protection:** Has a checkbox for 'Protect against accidental termination' which is unchecked.
- Monitoring:** Has a checkbox for 'Enable CloudWatch detailed monitoring' with a note 'Additional charges apply.'
- Tenancy:** Set to **Shared - Run a shared hardware instance**. A note 'Additional charges will apply for dedicated tenancy.' is shown below.

- 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.

▼ Advanced Details

Enclave  <input type="checkbox"/> Enable	
Metadata accessible  Enabled	
Metadata version  V1 and V2 (token optional)	
Metadata token response hop limit  1	
User data  <input checked="" type="radio"/> As text <input type="radio"/> As file <input type="checkbox"/> Input is already base64 encoded <pre>#!/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</pre>	

- 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 5: Add Tags

A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver.

A copy of a tag can be applied to volumes, instances or both.

Tags will be applied to all instances and volumes. Learn more about tagging your Amazon EC2 resources.

Key	(128 characters maximum)	Value	(256 characters maximum)	Instances 	Volumes 
Name		MyCodePipelineDemo		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input style="width: 150px; height: 20px; margin-bottom: 5px; border: 1px solid #ccc; padding: 2px; font-size: 10px; border-radius: 5px; background-color: #f0f0f0; color: #333; font-weight: bold; text-decoration: none; text-align: center; vertical-align: middle;" type="button" value="Add another tag"/> (Up to 50 tags maximum)					

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 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more about Amazon EC2 security groups.](#)

Assign a security group: Create a **new** security group
 Select an **existing** security group

Security group name:

Description:

Type	Protocol	Port Range	Source
SSH	TCP	22	My IP 158.228.254.13/32
HTTP	TCP	80	My IP 158.228.254.13/32

[Add Rule](#)

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.

Select an existing key pair or create a new key pair X

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

Proceed without a key pair

I acknowledge that without a key pair, I can connect to this instance only by using EC2 Instance Connect or if I know the password built into the AMI. Note that EC2 Instance Connect is only supported on Amazon Linux 2 and Ubuntu. [Learn more](#).

[Cancel](#) **Launch Instances**

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.

 AWS service EC2, Lambda and others	 Another AWS account Belonging to you or 3rd party	 Web identity Cognito or any OpenID provider	 SAML 2.0 federation Your corporate directory
--	---	---	--

Allows AWS services to perform actions on your behalf. [Learn more](#)

Choose a use case

Common use cases

EC2
Allows EC2 instances to call AWS services on your behalf.

Lambda
Allows Lambda functions to call AWS services on your behalf.

Or select a service to view its use cases

API Gateway	CodeBuild	EMR Containers	IoT SiteWise	RDS
AWS Backup	<u>CodeDeploy</u>	ElastiCache	IoT Things Graph	Redshift
AWS Chatbot	CodeGuru	Elastic Beanstalk	KMS	Rekognition

Select your use case

CodeDeploy
Allows CodeDeploy to call AWS services such as Auto Scaling on your behalf.

CodeDeploy - ECS
Allows CodeDeploy to read S3 objects, invoke Lambda functions, publish to SNS topics, and update ECS services on your behalf.

CodeDeploy for Lambda
Allows CodeDeploy to route traffic to a new version of an AWS Lambda function version on your behalf.

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

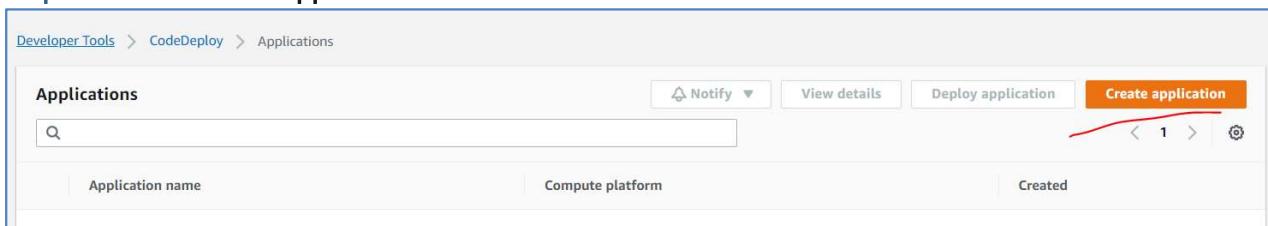
Step 6. Enter a name for the role (for example, **CodeDeployRole**), and then choose **Create role**.

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**.

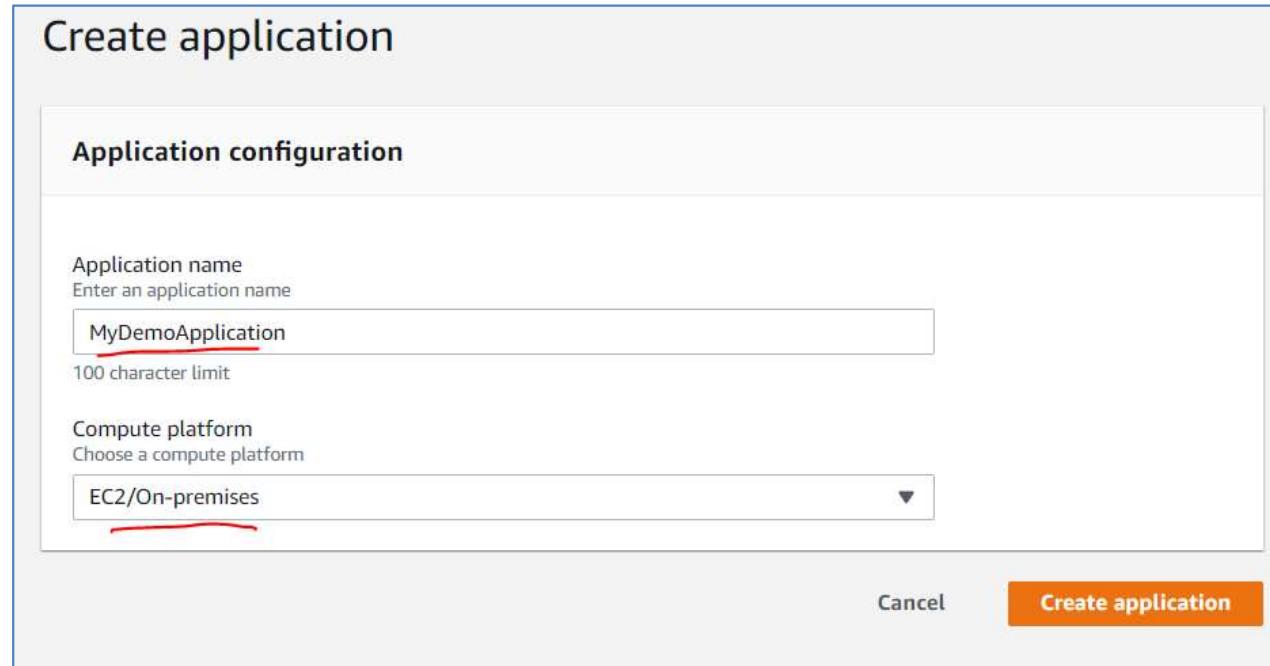


The screenshot shows the AWS CodeDeploy Applications page. The URL in the address bar is [Developer Tools > CodeDeploy > Applications](#). The main table has columns for Application name, Compute platform, and Created. A single row is visible. At the top right of the table, there is a red arrow pointing to the orange **Create application** button.

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

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

Step 6. Choose **Create application**.



Create application

Application configuration

Application name
Enter an application name
 MyDemoApplication

Compute platform
Choose a compute platform
 EC2/On-premises

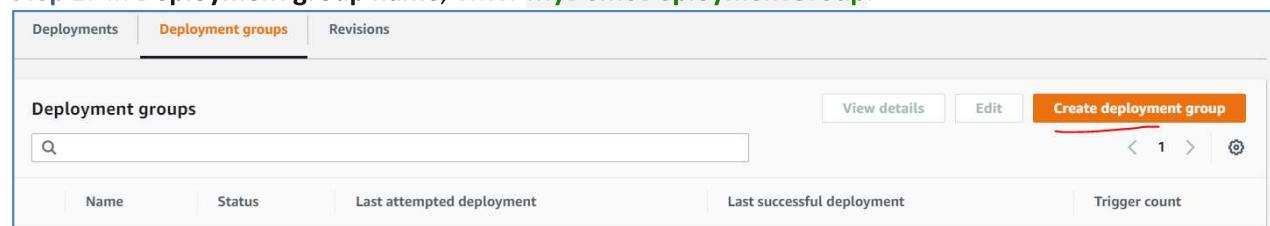
Create application

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**.



Deployments	Deployment groups	Revisions
	Deployment groups <input type="text" value="MyDemoDeploymentGroup"/> MyDemoDeploymentGroup	Create deployment group Create deployment group
	<input type="text"/>	< 1 > < 1 >
Name	Status	Last attempted deployment
		Last successful deployment
		Trigger count

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

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

Deployment group name

Enter a deployment group name
 100 character limit

Service role

Enter a service role
Enter a service role with CodeDeploy permissions that grants AWS CodeDeploy access to your target instances.

Deployment type

Choose how to deploy your application

In-place
Updates the instances in the deployment group with the latest application revisions. During a deployment, each instance will be briefly taken offline for its update

Blue/green
Replaces the instances in the deployment group with new instances and deploys the latest application revision to them. After instances in the replacement environment are registered with a load balancer, instances from the original environment are deregistered and can be terminated.

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**).

Environment configuration

Select any combination of Amazon EC2 Auto Scaling groups, Amazon EC2 instances, and on-premises instances to add to this deployment

Amazon EC2 Auto Scaling groups

Amazon EC2 instances
1 unique matched instance. [Click here for details](#)

You can add up to three groups of tags for EC2 instances to this deployment group.
One tag group: Any instance identified by the tag group will be deployed to.
Multiple tag groups: Only instances identified by all the tag groups will be deployed to.

Tag group 1

Key Value - optional [Remove tag](#)

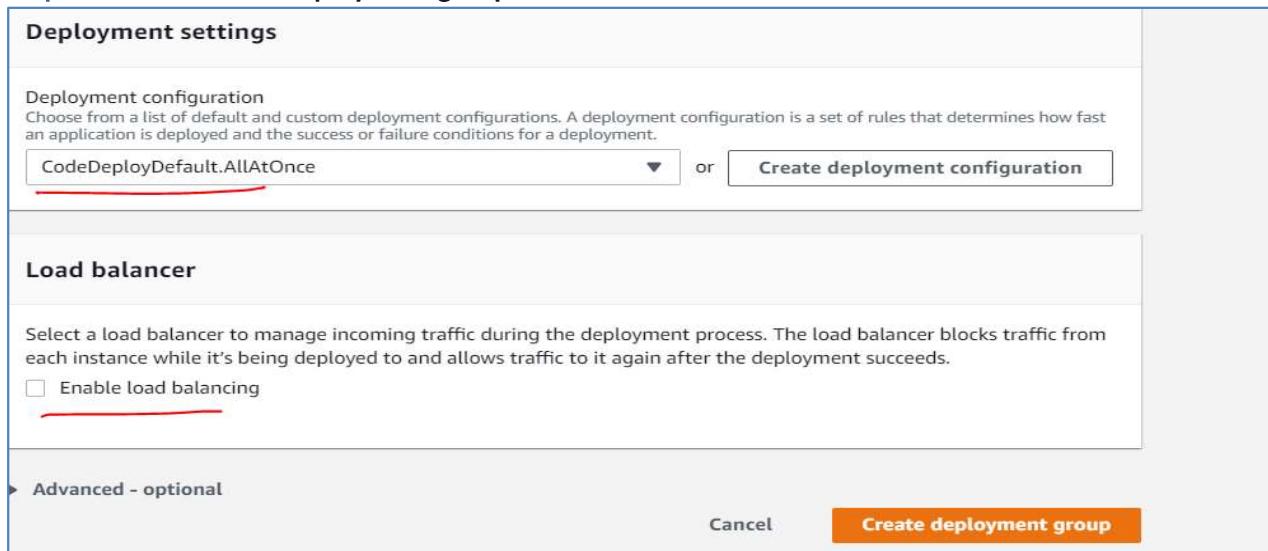
[Add tag](#) [+ Add tag group](#)

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**.



Deployment settings

Deployment configuration
Choose from a list of default and custom deployment configurations. A deployment configuration is a set of rules that determines how fast an application is deployed and the success or failure conditions for a deployment.

`CodeDeployDefault.AllAtOnce` or **Create deployment configuration**

Load balancer

Select a load balancer to manage incoming traffic during the deployment process. The load balancer blocks traffic from each instance while it's being deployed to and allows traffic to it again after the deployment succeeds.

Enable load balancing

▶ **Advanced - optional**

Create deployment group

Create your first pipeline in CodePipeline

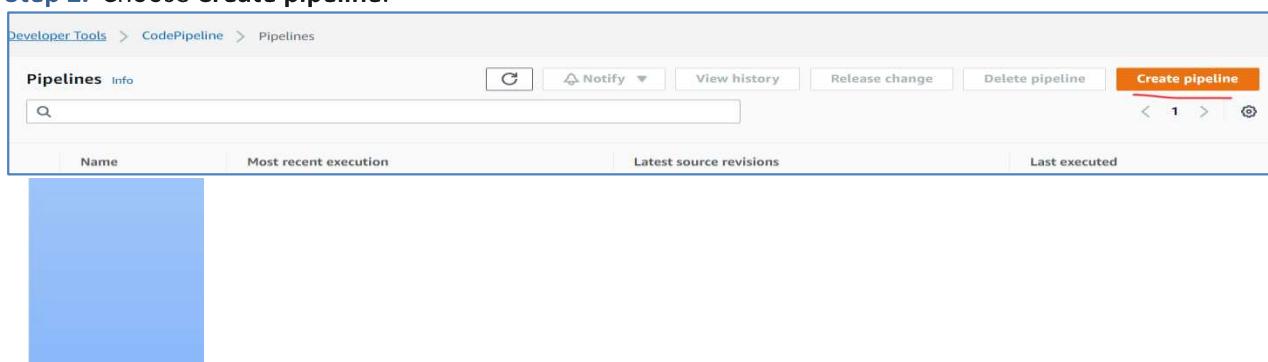
You're now ready to create and run your first pipeline. In this step, you create a pipeline that runs automatically when code is pushed to your CodeCommit repository.

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**.



Developer Tools > CodePipeline > Pipelines

Pipelines **Create pipeline**

Name	Most recent execution	Latest source revisions	Last executed

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**.

Choose pipeline settings Info

Pipeline settings

Pipeline name
Enter the pipeline name. You cannot edit the pipeline name after it is created.
 No more than 100 characters

Service role
 New service role Create a service role in your account Existing service role Choose an existing service role from your account

Role name

Type your service role name
 Allow AWS CodePipeline to create a service role so it can be used with this new pipeline

► Advanced settings

Cancel **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**.

Source

Source provider
This is where you stored your input artifacts for your pipeline. Choose the provider and then provide the connection details.

AWS CodeCommit

Repository name
Choose a repository that you have already created where you have pushed your source code.

MyDemoRepo

Branch name
Choose a branch of the repository

main

Change detection options
Choose a detection mode to automatically start your pipeline when a change occurs in the source code.

Amazon CloudWatch Events (recommended)
Use Amazon CloudWatch Events to automatically start my pipeline when a change occurs

AWS CodePipeline
Use AWS CodePipeline to check periodically for changes

Output artifact format
Choose the output artifact format.

CodePipeline default
AWS CodePipeline uses the default zip format for artifacts in the pipeline. Does not include git metadata about the repository.

Full clone
AWS CodePipeline passes metadata about the repository that allows subsequent actions to do a full git clone. Only supported for AWS CodeBuild actions.

Cancel Previous 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**.

Build - optional

Build provider
This is the tool of your build project. Provide build artifact details like operating system, build spec file, and output file names.

Cancel Previous Skip build stage Next

Skip build stage



Your pipeline will not include a build stage. Are you sure you want to skip this stage?

Cancel

Skip

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.

Deploy

Deploy provider

Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.

AWS CodeDeploy

Region

US East (Ohio)

Application name

Choose an application that you have already created in the AWS CodeDeploy console. Or create an application in the AWS CodeDeploy console and then return to this task.

MyDemoApplication 

Deployment group

Choose a deployment group that you have already created in the AWS CodeDeploy console. Or create a deployment group in the AWS CodeDeploy console and then return to this task.

MyDemoDeploymentGroup 

Cancel

Previous

Next

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

Step 4: Add deploy stage

Deploy action provider

Deploy action provider
AWS CodeDeploy

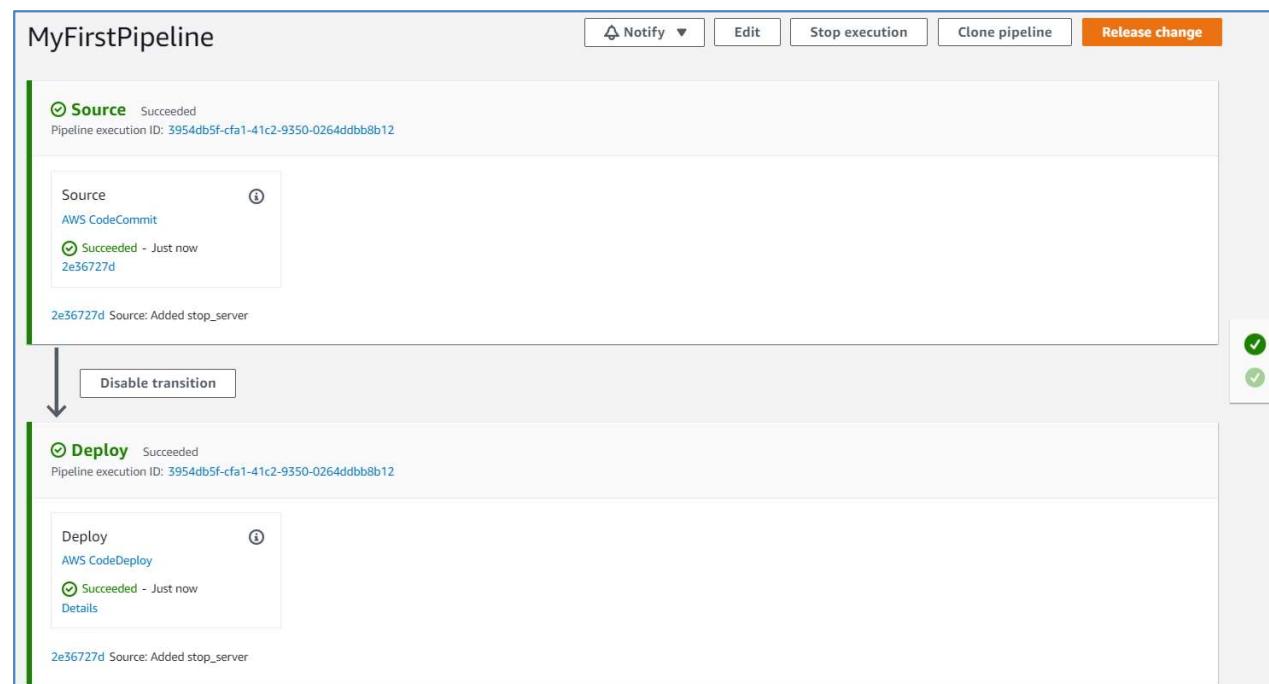
ApplicationName
MyDemoApplication

DeploymentGroupName
MyDemoDeploymentGroup

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.



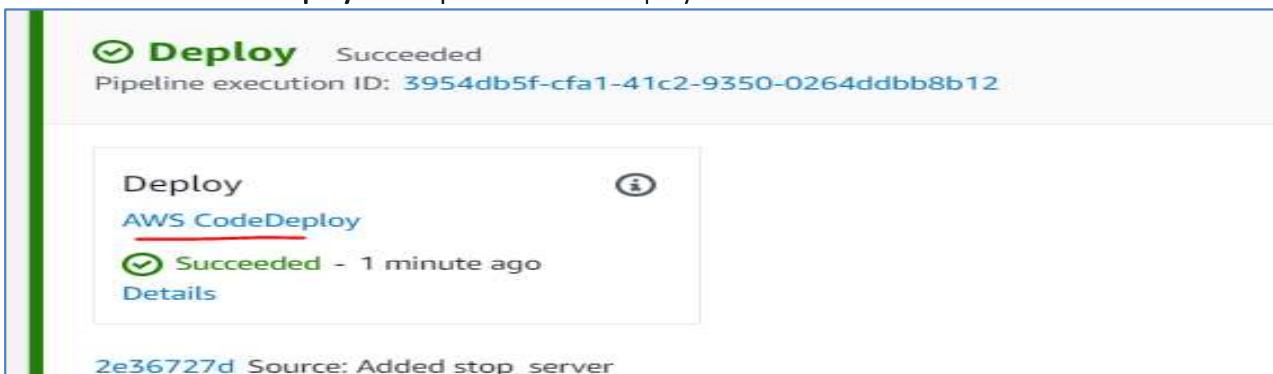
Congratulations! You just created a simple pipeline in CodePipeline.

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.

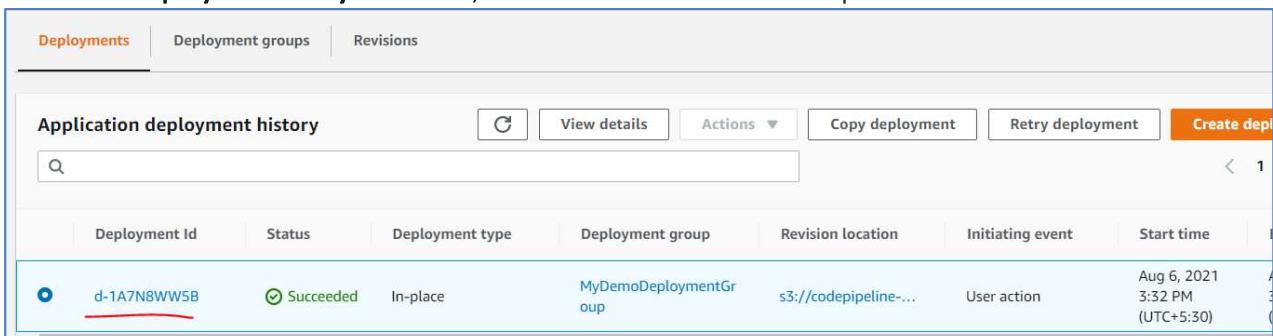
Step 2. After **Succeeded** is displayed for the pipeline status, in the status area for the **Deploy** stage, choose **AWS CodeDeploy**. This opens the CodeDeploy console.



The screenshot shows the AWS CodePipeline console. At the top, it displays the pipeline execution ID: **3954db5f-cfa1-41c2-9350-0264ddb8b12**. Below this, the **Deploy** stage is shown with the status **Succeeded** and a timestamp of **1 minute ago**. A red underline highlights the **AWS CodeDeploy** link. At the bottom of the stage card, there is a link labeled **Details**.

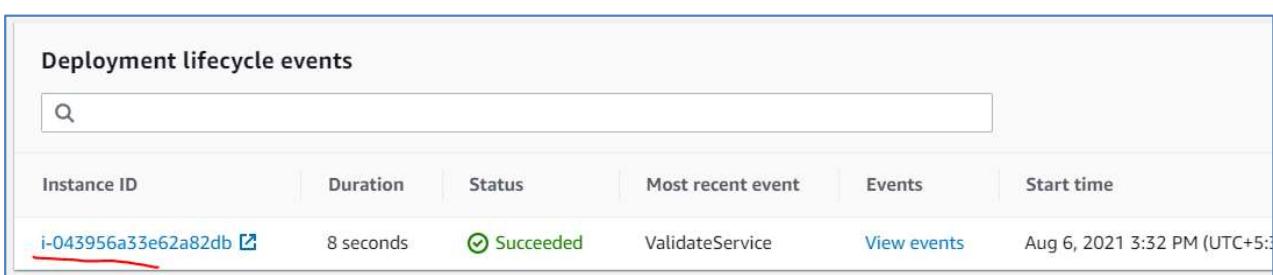
Below the stage card, the deployment ID **2e36727d** is listed with the source note **Source: Added stop_server**.

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.



The screenshot shows the **Deployments** tab in the AWS CodeDeploy console. It lists a single deployment entry:

Deployment Id	Status	Deployment type	Deployment group	Revision location	Initiating event	Start time
d-1A7N8WW5B	Succeeded	In-place	MyDemoDeploymentGroup	s3://codepipeline-...	User action	Aug 6, 2021 3:32 PM (UTC+5:30)



The screenshot shows the **Deployment lifecycle events** tab in the AWS CodeDeploy console. It lists a single event entry:

Instance ID	Duration	Status	Most recent event	Events	Start time
i-043956a33e62a82db	8 seconds	Succeeded	ValidateService	View events	Aug 6, 2021 3:32 PM (UTC+5:30)

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.



The screenshot shows the CloudWatch Metrics Insights pipeline named "MyCodePipelineDemo". It displays the following details:

- Instance ID: i-043956a33e62a82db (MyCodePipelineDemo)
- Public IPv4 address: 18.118.2.239 (with a red underline under the IP address)
- Private IPv4 addresses: 172.31.1.11
- Public IPv4 DNS: ec2-18-118-2-239.us-east-2.compute.amazonaws.com (with a red underline under the IP address)
- Instance state: Running



The screenshot shows a web browser window with the following content:

Not secure | ec2-18-118-2-239.us-east-2.compute.amazonaws.com

Congratulations

This application was deployed using AWS CodeDeploy.

For next steps, read the [AWS CodeDeploy Documentation](#).

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