# Setting Node.js

$docker pull node

Add new file hello.js

Console.log(“hello”)

sudo docker run –it –rm –name = HelloWorld –v “$PWD”:/usr/src/app

–w /usr/src/app node node hello.js

The following points need to be noted about the above command −

* The **–rm** option is used to remove the container after it is run.
* We are giving a name to the container called “HelloWorld”.
* We are mentioning to map the volume in the container which is **/usr/src/app** to our current present working directory. This is done so that the node container will pick up our HelloWorld.js script which is present in our working directory on the Docker Host.
* The **–w** option is used to specify the working directory used by Node.js.
* The first node option is used to specify to run the node image.
* The second node option is used to mention to run the node command in the node container.
* And finally we mention the name of our script.

# Setting MongoDB

$docker pull mongo

sudo docker run -it -d mongo

The following points can be noted about the above command −

* The **–it** option is used to run the container in interactive mode.
* The **–d** option is used to run the container as a daemon process.
* And finally we are creating a container from the Mongo image.

Take a note of the following points −

* The name of the container is **tender\_poitras**. This name will be different since the name of the containers keep on changing when you spin up a container. But just make a note of the container which you have launched.

sudo docker run –it –link=tender\_poitras:mongo mongo /bin/bash

The following points can be noted about the above command −

* The **–it** option is used to run the container in interactive mode.
* We are now linking our new container to the already launched MongoDB server container. Here, you need to mention the name of the already launched container.
* We are then specifying that we want to launch the Mongo container as our client and then run the **bin/bash** shell in our new container.

Now it’s time to connect to the MongoDB server from the client container. We can do this via the following command −

mongo 172.17.0.2:27017

The following points need to be noted about the above command

* The **mongo** command is the client **mongo** command that is used to connect to a MongoDB database.
* The IP and port number is what you get when you use the **env**command.
* use demo

Setting NGINX

$docker pull nginx

$sudo docker run –p 8080:80 –d nginx

we will create a simple **HelloWorld.html** file and host it in our **nginx** container.

Let’s first create an HTML file called **HelloWorld.html**

Let’s then run the following Docker command.

sudo docker run –p 8080:80 –v

“$PWD”:/usr/share/nginx/html:ro –d nginx

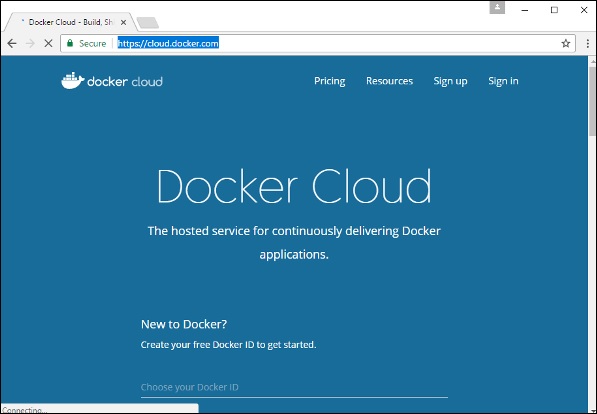
# Docker - Cloud

The Docker Cloud is a service provided by Docker in which you can carry out the following operations −

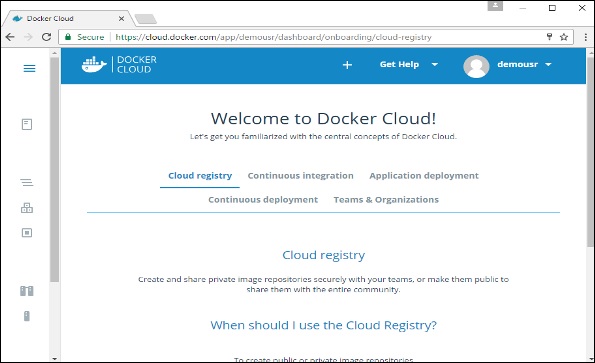
* **Nodes** − You can connect the Docker Cloud to your existing cloud providers such as Azure and AWS to spin up containers on these environments.
* **Cloud Repository** − Provides a place where you can store your own repositories.
* **Continuous Integration** − Connect with **Github** and build a continuous integration pipeline.
* **Application Deployment** − Deploy and scale infrastructure and containers.
* **Continuous Deployment** − Can automate deployments.

## Getting started

You can go to the following link to getting started with Docker Cloud − <https://cloud.docker.com/>



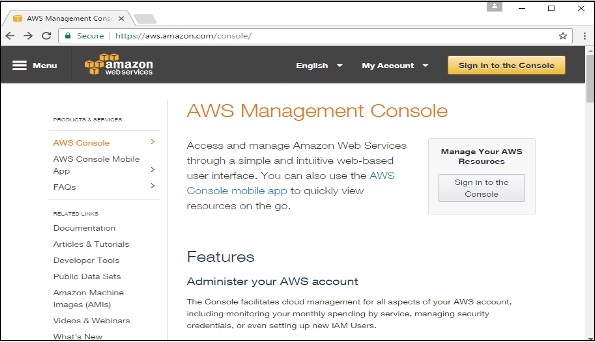
Once logged in, you will be provided with the following basic interface −



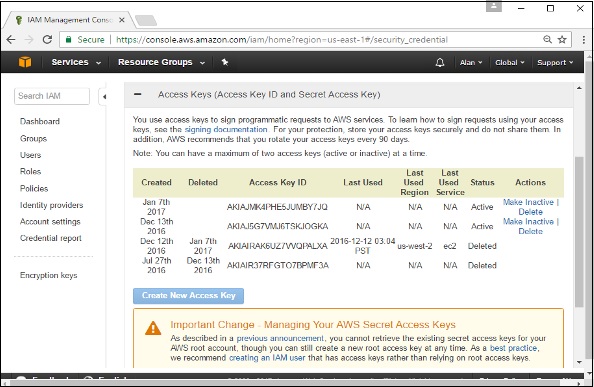
## Connecting to the Cloud Provider

The first step is to connect to an existing cloud provider. The following steps will show you how to connect with an Amazon Cloud provider.

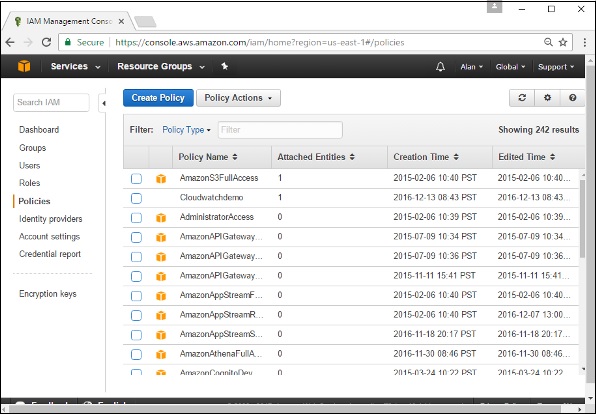
**Step 1** − The first step is to ensure that you have the right AWS keys. This can be taken from the **aws** console. Log into your **aws** account using the following link − <https://aws.amazon.com/console/>



**Step 2** − Once logged in, go to the Security Credentials section. Make a note of the access keys which will be used from Docker Hub.



**Step 3** − Next, you need to create a policy in **aws** that will allow Docker to view EC2 instances. Go to the profiles section in **aws**. Click the **Create Policy**button.



**Step 4** − Click on ‘Create Your Own Policy’ and give the policy name as **dockercloudpolicy** and the policy definition as shown below.

{

"Version": "2012-10-17",

"Statement": [ {

"Action": [

"ec2:\*",

"iam:ListInstanceProfiles"

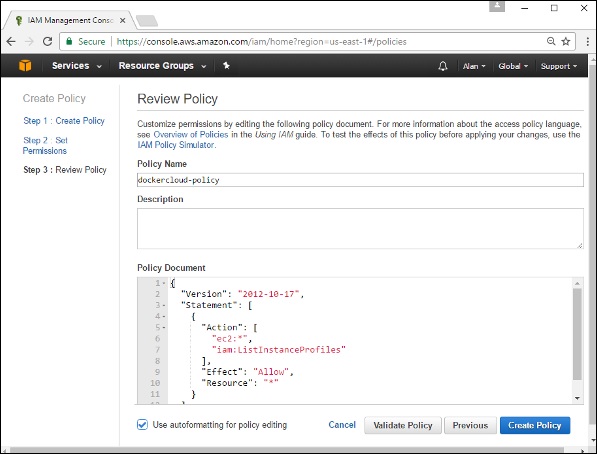
],

"Effect": "Allow",

"Resource": "\*"

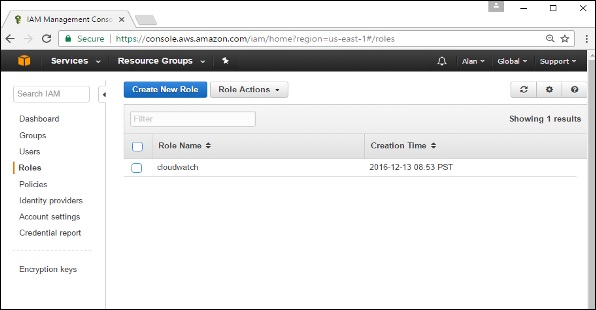
} ]

}

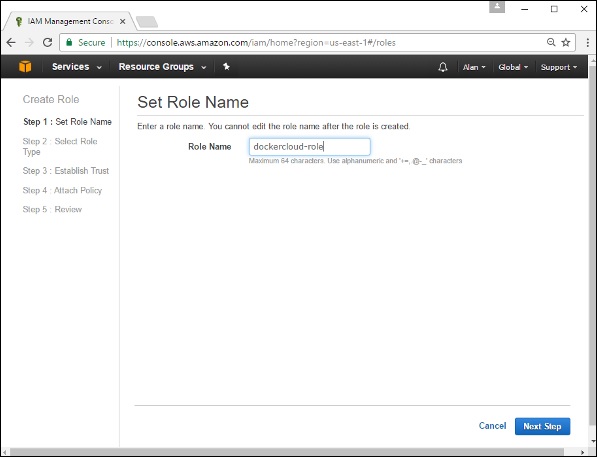


Next, click the **Create Policy** button

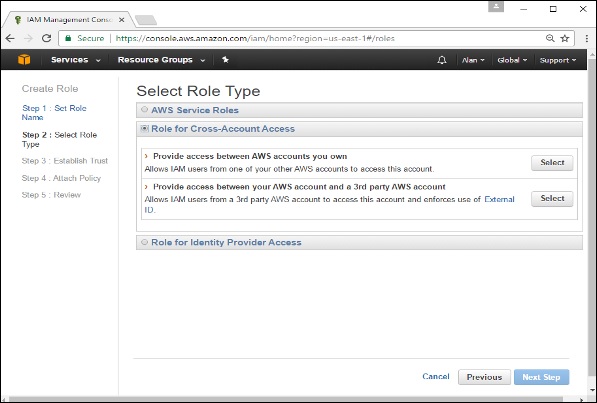
**Step 5** − Next, you need to create a **role** which will be used by Docker to spin up nodes on AWS. For this, go to the **Roles** section in AWS and click the **Create New Role** option.



**Step 6** − Give the name for the role as **dockercloud-role**.

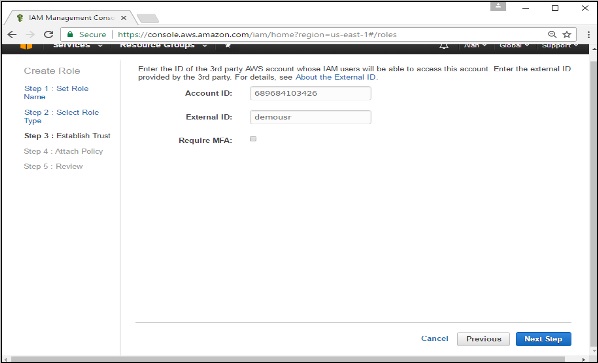


**Step 7** − On the next screen, go to ‘Role for Cross Account Access’ and select “Provide access between your account and a 3rd party AWS account".

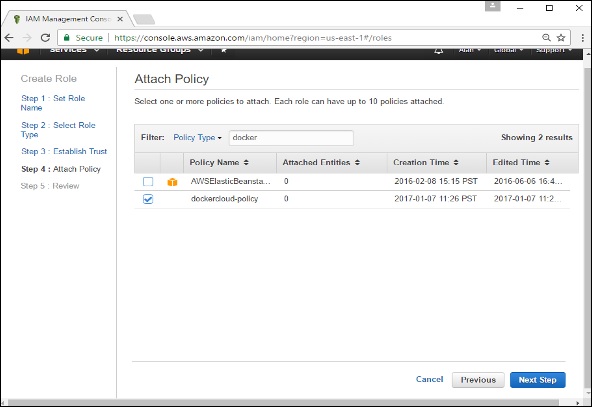


**Step 8** − On the next screen, enter the following details −

* In the Account ID field, enter the ID for the Docker Cloud service: 689684103426.
* In the External ID field, enter your Docker Cloud username.

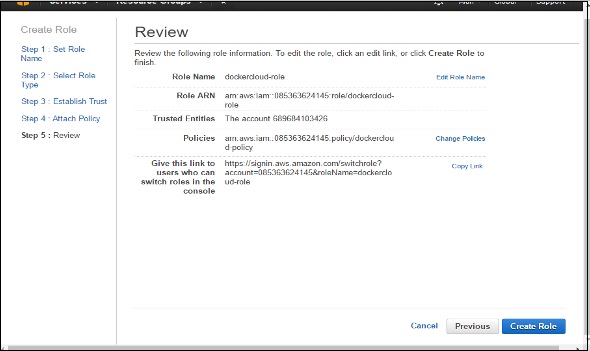


**Step 9** − Then, click the **Next Step** button and on the next screen, attach the policy which was created in the earlier step.

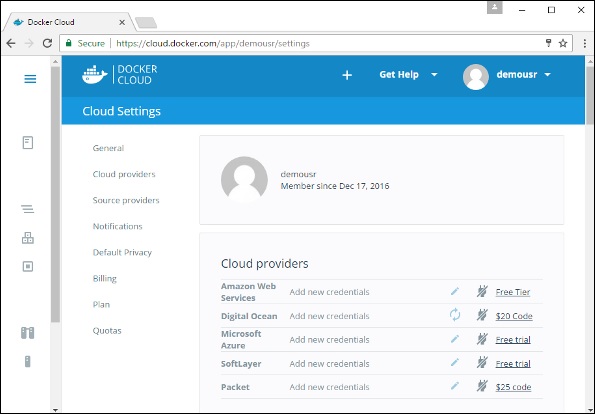


**Step 10** − Finally, on the last screen when the role is created, make sure to copy the **arn** role which is created.

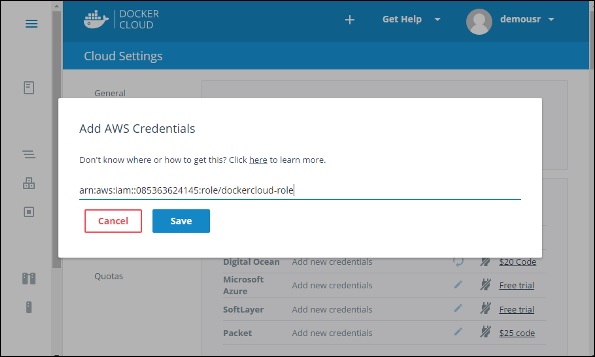
arn:aws:iam::085363624145:role/dockercloud-role



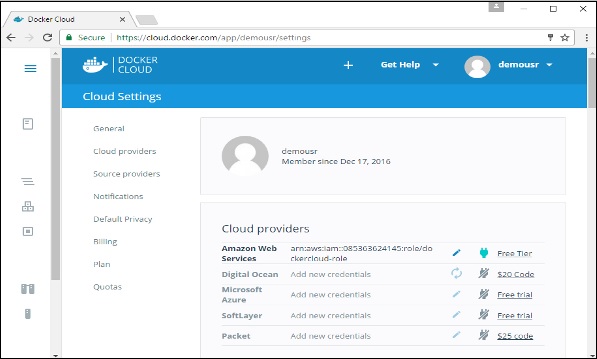
**Step 11** − Now go back to **Docker Cloud**, select **Cloud Providers**, and click the **plug symbol** next to Amazon Web Services.



Enter the **arn** role and click the **Save** button.



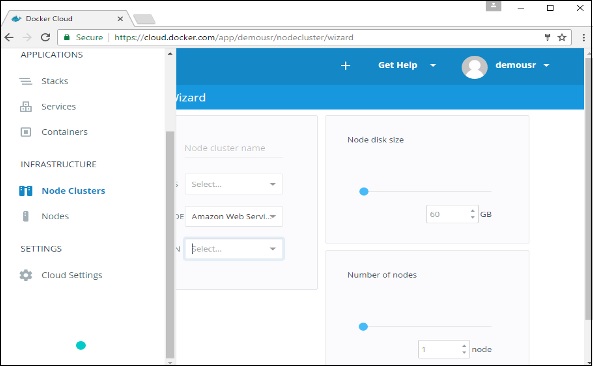
Once saved, the integration with AWS would be complete.



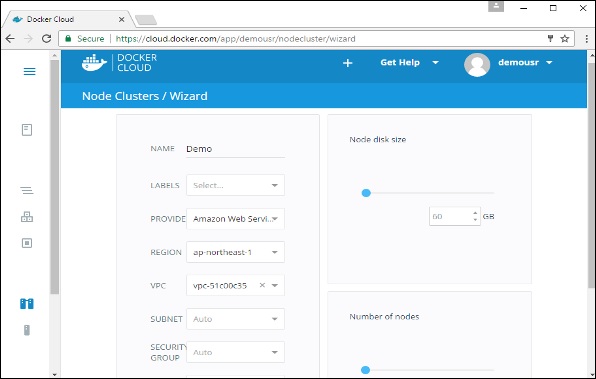
## Setting Up Nodes

Once the integration with AWS is complete, the next step is to setup a node. Go to the Nodes section in Docker Cloud. Note that the setting up of nodes will automatically setup a node cluster first.

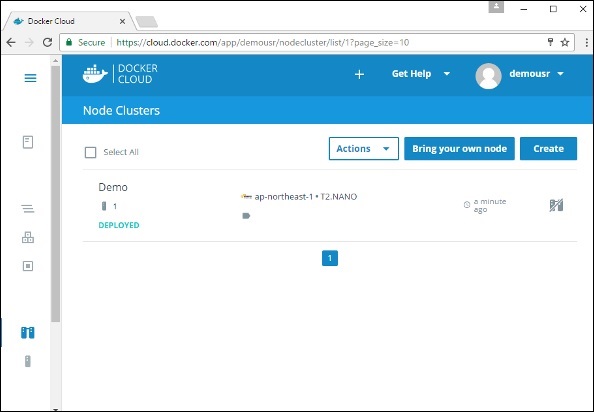
**Step 1** − Go to the Nodes section in Docker Cloud.



**Step 2** − Next, you can give the details of the nodes which will be setup in AWS.



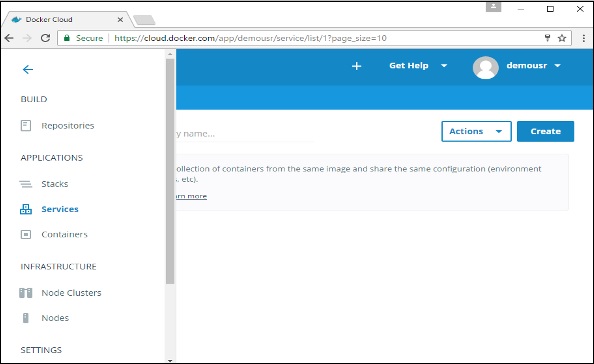
You can then click the Launch Node cluster which will be present at the bottom of the screen. Once the node is deployed, you will get the notification in the Node Cluster screen.



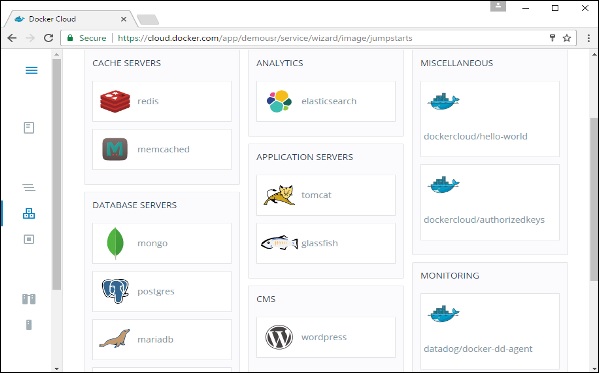
## Deploying a Service

The next step after deploying a node is to deploy a service. To do this, we need to perform the following steps.

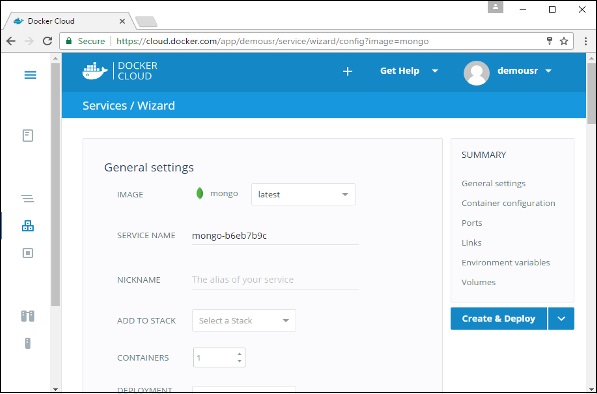
**Step 1** − Go to the **Services Section** in Docker Cloud. Click the **Create**button



**Step 2** − Choose the Service which is required. In our case, let’s choose **mongo**.



**Step 3** − On the next screen, choose the **Create & Deploy** option. This will start deploying the **Mongo** container on your node cluster.



Once deployed, you will be able to see the container in a running state.

