[Write](https://medium.com/new-story?source=post_page---top_nav_layout_nav-----------------------------------------)



[Home](https://medium.com/?source=post_page--------------------------------------------)

[Library](https://medium.com/me/lists?source=post_page--------------------------------------------)

[Profile](https://medium.com/@dk1078451?source=post_page--------------------------------------------)

[Stories](https://medium.com/me/stories/drafts?source=post_page--------------------------------------------)

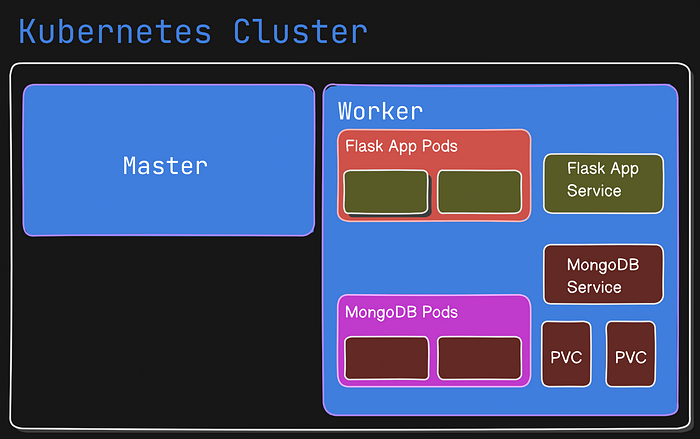
[Stats](https://medium.com/me/stats?source=post_page--------------------------------------------)

Get unlimited access to the best of Medium for less than $1/week.

[Become a member](https://medium.com/plans?source=upgrade_membership---post_top_nav_upsell-----------------------------------------)

Deepanshu





**How to Deploy a Microservices Application on AWS EC2 using Kubernetes: A Step-by-Step Guide**

Mastering Microservices Deployment with Kubernetes on AWS EC2: A Step-by-Step Guide

[[](https://ajitfawade.medium.com/?source=post_page---byline--63b845544d39---------------------------------------)](https://ajitfawade.medium.com/?source=post_page---byline--63b845544d39---------------------------------------)

[Ajit Fawade](https://ajitfawade.medium.com/?source=post_page---byline--63b845544d39---------------------------------------)

·

Follow

14 min read

·

Oct 9, 2023

71

2

Hieveryone, welcome to my #90DaysOfDevOps blog series where I share my daily learning and progress on DevOps.

Today, I’m going to show you how to deploy a microservices application on AWS EC2 using Kubernetes, which is a popular platform for managing containerized applications.

The blog post will consist of the following sections:

* Setting up AWS EC2 Instances: In this section, I will show you how to create two t2.medium instances on AWS EC2, one for the master node and one for the worker node. I will also show you how to configure the security groups and SSH keys for these instances.
* Installing Kubernetes on AWS EC2 Instances: In this section, I will show you how to install Kubernetes on both instances using kubeadm, kubelet, and kubectl. I will also show you how to initialize the master node and join the worker node to the cluster.
* Deploying the Microservices Application on Kubernetes: In this section, I will show you how to deploy a microservices application that consists of a MongoDB database and a taskmaster service. The taskmaster service is a web app that allows you to create and manage tasks. The app is written in Flask and uses MongoDB as the database. I will show you how to deploy the persistent volume and persistent volume claim, the MongoDB database, the ClusterIP service for MongoDB, the taskmaster service, and the NodePort service for taskmaster.

*By the end of this blog post, you will have a working microservices application running on AWS EC2 using Kubernetes. You will also learn some important concepts and commands related to Kubernetes, such as pods, services, deployments, etc.*

I hope you are excited to learn more about this topic. If you are ready, let’s begin! 🚀

**Setting up AWS EC2 Instances**

In this section, we will create two t2.medium instances on AWS EC2, one for the master node and one for the worker node. We will also configure the security groups and SSH keys for these instances.

**Creating Two**t2.medium**Instances**

To create two t2.medium instances on AWS EC2, follow these steps:

1. Log in to your AWS console and go to the EC2 dashboard.
2. Click on the Launch Instance button.
3. Choose Ubuntu Server 22.04 LTS (HVM) as the AMI.
4. Choose t2.medium as the instance type.
5. Click on Next until you reach the Configure Security Group page.
6. Create a new security group with the following rules:

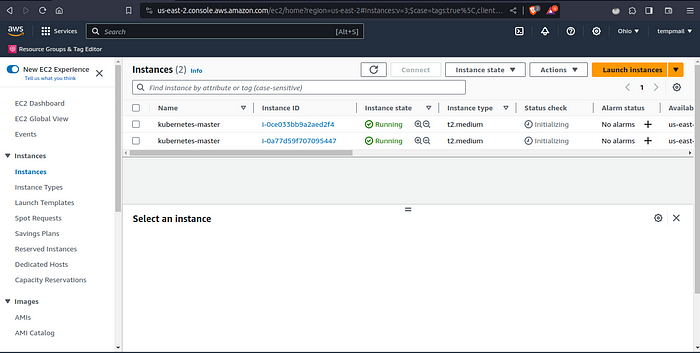
* Allow SSH from anywhere (port 22)
* Allow TCP from anywhere (port 80)
* Allow TCP from anywhere (port 5000)
* Allow TCP from anywhere (port 6443)
* Allow all traffic from within the security group (port range 0–65535)

7. Click on the Review and Launch button.

8. Create a new key pair or use an existing one and download it.

9. Click on the Launch Instances button.

You should see something like this:



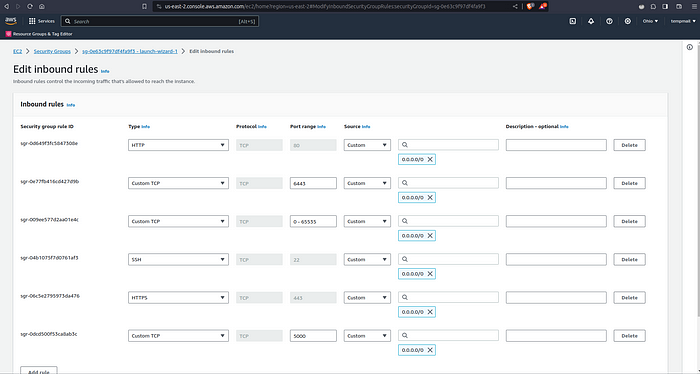
This shows that you have successfully launched two instances on AWS EC2.

**Configuring Security Groups and SSH Keys**

To configure the security groups and SSH keys for our instances, follow these steps:

1. Go to the Instances page on the EC2 dashboard and select one of your instances.
2. Click on Actions > Networking > Change Security Groups.
3. Select the security group that you created in the previous step and click on the Assign Security Groups button.
4. Repeat the same steps for the other instance.

You should see something like this:



This shows that you have successfully assigned the same security group to both instances.

5. Go to your terminal and change the permissions of your key pair file by running:

chmod 400 ~/.ssh/mykey.pem

Replace ~/.ssh/mykey.pem with the path to your key pair file.

This will make sure that only you can read and write to your key pair file.

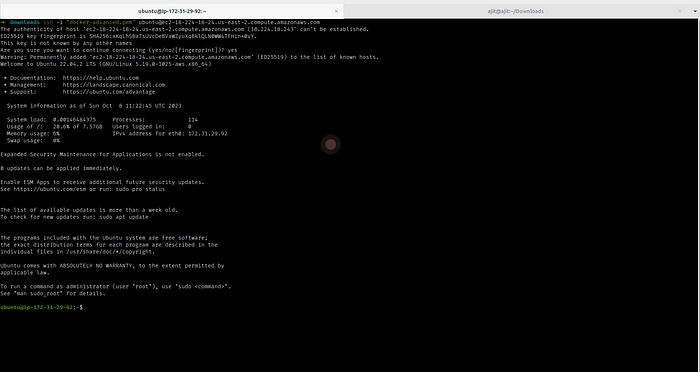
6. SSH into one of your instances by running:

ssh -i ~/.ssh/mykey.pem ubuntu@<instance-ip>

Replace ~/.ssh/mykey.pem with the path to your key pair file and <instance-ip> with the public IPv4 address of your instance.

This will establish a secure connection to your instance using your key pair.

You should see something like this:



This shows that you have successfully logged into your instance.

7. Repeat the same steps for the other instance.

You should now have two terminals connected to your instances, one for the master node and one for the worker node.

You have successfully set up two AWS EC2 instances for our Kubernetes cluster. In the next section, we will install Kubernetes on these instances.

**Installing Kubernetes on AWS EC2 Instances**

In this section, we will install Kubernetes on both instances using kubeadm, kubelet, and kubectl. Kubeadm is a tool that helps us bootstrap a Kubernetes cluster. Kubelet is an agent that runs on each node and communicates with the master node. Kubectl is a command-line tool that allows us to interact with the cluster.

**Installing Docker on Both Instances**

Before we install Kubernetes, we need to install Docker on both instances. Docker is a software that allows us to run containers, which are isolated environments that contain our applications and their dependencies.

To install Docker on both instances, follow these steps:

1. SSH into one of your instances by running:

ssh -i ~/.ssh/mykey.pem ubuntu@<instance-ip>

Replace ~/.ssh/mykey.pem with the path to your key pair file and <instance-ip> with the public IPv4 address of your instance.

2. Update the package index by running:

sudo apt update

3. Install Docker by running:

sudo apt install docker.io -y

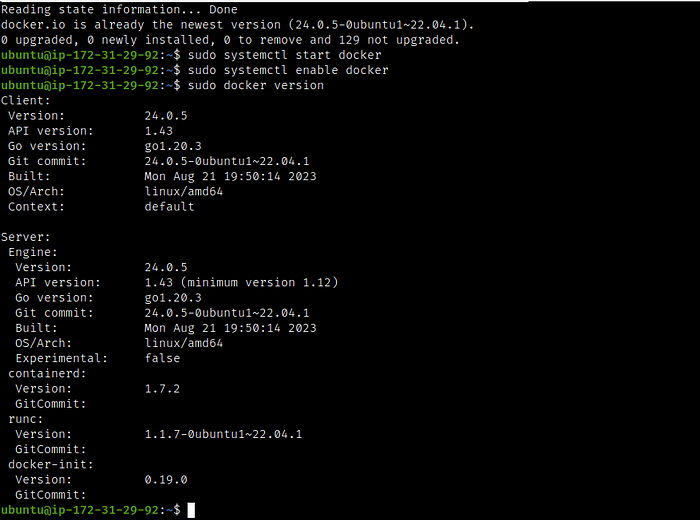
4. Start and enable the Docker service by running:

sudo systemctl start docker  
sudo systemctl enable docker

5. Verify that Docker is installed and running by running:

sudo docker version

You should see something like this:



This shows that you have successfully installed Docker on your instance.

6. Add current user to the docker group

sudo usermod -aG docker $USER

7. Repeat the same steps for the other instance.

You should now have Docker installed and running on both instances.

Try restarting your instances before proceeding further. This will make sure that the docker is installed and the current user is added to the docker group.

**Installing**kubeadm**,**kubelet**, and**kubectl**on Both Instances**

To install kubeadm, kubelet, and kubectl on both instances, follow these steps:

1. SSH into one of your instances by running:

ssh -i ~/.ssh/mykey.pem ubuntu@<instance-ip>

Replace ~/.ssh/mykey.pem with the path to your key pair file and <instance-ip> with the public IPv4 address of your instance.

2. Add the Kubernetes apt repository by running:

sudo apt update  
sudo apt install apt-transport-https ca-certificates curl -y  
curl -fsSL "https://packages.cloud.google.com/apt/doc/apt-key.gpg" | sudo gpg --dearmor -o /etc/apt/trusted.gpg.d/kubernetes-archive-keyring.gpg  
echo 'deb https://packages.cloud.google.com/apt kubernetes-xenial main' | sudo tee /etc/apt/sources.list.d/kubernetes.list

3. Install kubeadm, kubelet, and kubectl by running:

sudo apt update  
sudo apt install kubeadm=1.20.0-00 kubectl=1.20.0-00 kubelet=1.20.0-00 -y

4. Verify that kubeadm, kubelet, and kubectl are installed by running:

kubeadm version  
kubelet --version  
kubectl version --client

You should see something like this:



This shows that you have successfully installed kubeadm, kubelet, and kubectl on your instance.

5. Repeat the same steps for the other instance.

You should now have kubeadm, kubelet, and kubectl installed on both instances.

**Initializing the Master Node**

To initialize the master node, follow these steps:

1. SSH into the instance that you want to use as the master node by running:

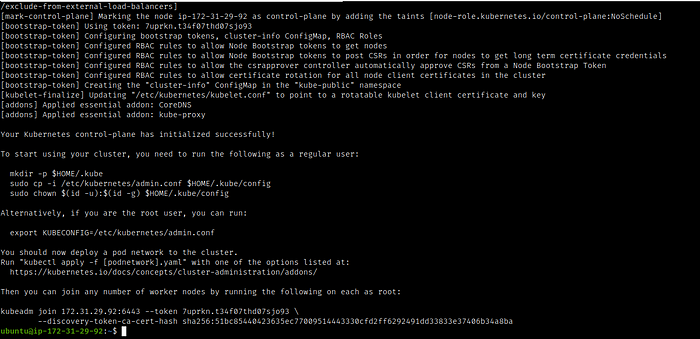
ssh -i ~/.ssh/mykey.pem ubuntu@<master-node-ip>

Replace ~/.ssh/mykey.pem with the path to your key pair file and <master-node-ip> with the public IPv4 address of your master node instance.

2. Initialize the master node by running:

sudo kubeadm init

This command will take some time to complete and will output a lot of information. At the end of the output, you should see something like this:



This shows that you have successfully initialized the master node.

3. Copy the join command that is displayed at the end of the output. You will need this command later to join the worker node to the cluster.

4. Configure your user account to use kubectl by running:

mkdir -p $HOME/.kube  
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
sudo chown $(id -u):$(id -g) $HOME/.kube/config

This will create a .kube directory in your home directory and copy the cluster configuration file to it. This will allow you to use kubectl to interact with the cluster.

5. Install a pod network add-on by running:

kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

This will install Flannel, a pod network add-on that provides a layer of networking for the pods in the cluster. Flannel will use the pod network CIDR that we specified earlier when initializing the master node.

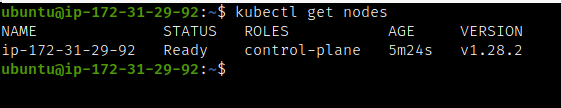
6. Generate a token for worker nodes to join:

sudo kubeadm token create --print-join-commandsudo kubeadm token create --print-join-command

7. Verify that the master node is ready by running:

kubectl get nodes

You should see something like this:



This shows that you have one node in your cluster, which is the master node, and it is ready to run pods.

You have successfully initialized the master node. In the next section, we will join the worker node to the cluster.

**Joining the Worker Node to the Cluster**

In this section, we will join the worker node to the cluster using the join command that we copied earlier from the master node. This will allow us to run pods on both nodes and distribute the workload.

To join the worker node to the cluster, follow these steps:

1. SSH into the instance that you want to use as the worker node by running:

ssh -i ~/.ssh/mykey.pem ubuntu@<worker-node-ip>

Replace ~/.ssh/mykey.pem with the path to your key pair file and <worker-node-ip> with the public IPv4 address of your worker node instance.

2. Run the following command

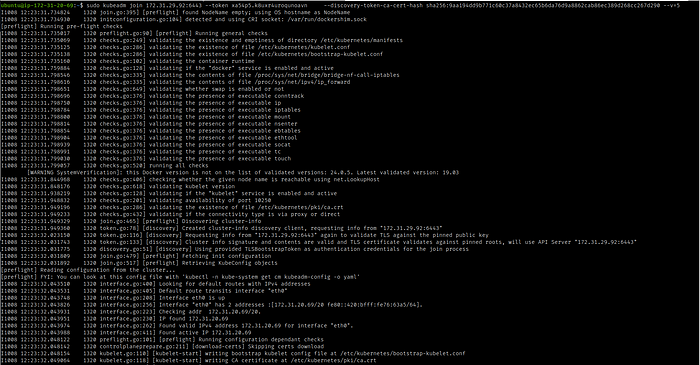
sudo kubeadm reset pre-flight checks

3. Join the worker node to the cluster by running the join command that you copied from the master node. The command should look something like this:

sudo kubeadm join 172.31.29.92:6443 --token thclqt.0m4dzsh5m6haswf5 --discovery-token-ca-cert-hash sha256:51bc85440423635ec77009514443330cfd2ff6292491dd33833e37406b34a8ba

Replace <master-node-ip> with the public IPv4 address of your master node instance, <token> with the token that was generated by kubeadm, and <hash> with the hash of the CA certificate.

This command will take some time to complete and will output some information. At the end of the output, you should see something like this:

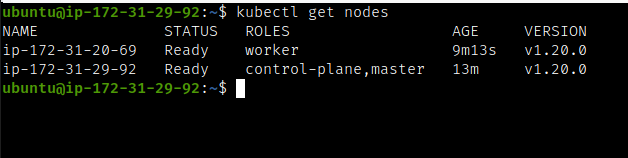


This shows that you have successfully joined the worker node to the cluster.

4. Verify that the worker node is ready by running:

kubectl get nodes

You should see something like this:



This shows that you have two nodes in your cluster, one master and one worker, and both are ready to run pods.

You have successfully joined the worker node to the cluster. In the next section, we will deploy a microservices application on Kubernetes.

**Deploying the Microservices Application on Kubernetes**

In this section, we will deploy a microservices application on Kubernetes that consists of a MongoDB database and a taskmaster service. The taskmaster service is a web app that allows you to create and manage tasks. The app is written in Node.js and uses MongoDB as the database.

To deploy the microservices application on Kubernetes, follow these steps:

**Cloning the GitHub Repository**

To clone the GitHub repository that contains the source code and the configuration files for our microservices application, follow these steps:

1. SSH into the master node by running:

ssh -i ~/.ssh/mykey.pem ubuntu@<master-node-ip>

Replace ~/.ssh/mykey.pem with the path to your key pair file and <master-node-ip> with the public IPv4 address of your master node instance.

2. Install git by running:

sudo apt install git -y

3. Clone the repository that contains the configuration files for our MongoDB database by running:

git clone https://github.com/ajitfawade/microservices-k8s.git

This will create a directory named microservices-k8s in your current directory.

**Deploying the Persistent Volume and Persistent Volume Claim**

To deploy the persistent volume and persistent volume claim for our MongoDB database, follow these steps:

1. Go to the mongodb-k8s directory by running:

cd microservices-k8s/flask-api/k8s

2. Apply the YAML file that defines the persistent volume by running:

kubectl apply -f mongo-pv.yml

This will create a persistent volume object named mongodb-pv that allocates 1 GB of storage on the host path /data/mongodb.

3. Apply the YAML file that defines the persistent volume claim by running:

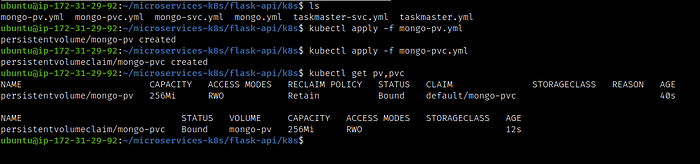
kubectl apply -f mongo-pvc.yml

This will create a persistent volume claim object named mongodb-pvc that requests 1 GB of storage from the persistent volume.

4. Verify that the persistent volume and persistent volume claim are created and bound by running:

kubectl get pv,pvc

You should see something like this:



This shows that you have one persistent volume named mongo-pv and one persistent volume claim named mongo-pvc.

You have successfully deployed the persistent volume and persistent volume claim for our MongoDB database. In the next section, we will deploy the MongoDB database itself.

**Deploying the MongoDB Database**

To deploy the MongoDB database for our microservices application, follow these steps:

1. Apply the YAML file that defines the deployment by running:

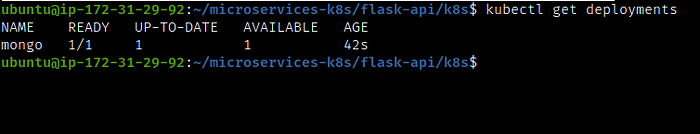
kubectl apply -f mongo.yml

This will create a deployment object named mongo-deployment that creates one replica of a pod that runs the mongo:4.4.6 image. The deployment has the label app: mongodb. The pod mounts the persistent volume claim mongodb-pvc to the path /data/db and uses the secret mongodb-secret to set the environment variables MONGO\_INITDB\_ROOT\_USERNAME and MONGO\_INITDB\_ROOT\_PASSWORD.

2. Verify that the deployment is created and running by running:

kubectl get deployments

You should see something like this:

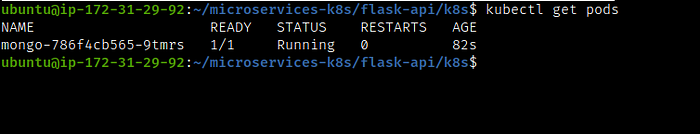


This shows that you have one deployment named mongo that has one replica ready and up-to-date.

3. Verify that the pod is running by running:

kubectl get pods

You should see something like this:



This shows that you have one pod named mongodb-786f4cb565-9tmrs that is running and ready.

You have successfully deployed the MongoDB database for our microservices application. In the next section, we will deploy the ClusterIP service for MongoDB.

**Deploying the ClusterIP Service for MongoDB**

To deploy the ClusterIP Service for MongoDB, follow these steps:

1. Apply the YAML file that defines the service by running:

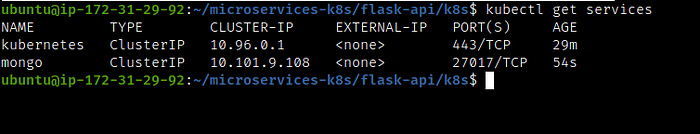
kubectl apply -f mongo-svc.yml

This will create a service object named mongo that exposes port 27017 of our pods as port 27017 on an internal IP address. The service selects the pods that have the label app: mongo. The type: ClusterIP specifies that this is a ClusterIP Service.

2. Verify that the service is created by running:

kubectl get services

You should see something like this:



This shows that you have one ClusterIP Service named mongo that exposes port 27017 of your pods as port 27017 on an internal IP address 10.101.9.108.

You have successfully deployed the ClusterIP Service for MongoDB. This will allow our taskmaster service to communicate with our MongoDB database within the cluster. In the next section, we will deploy the taskmaster service itself.

**Deploying the Taskmaster Service**

To deploy the taskmaster service for our microservices application, follow these steps:

1. Apply the YAML file that defines the deployment by running:

kubectl apply -f taskmaster.yml

This will create a deployment object named taskmaster-deployment that creates two replicas of pods that run the ajitfawade14/first-repo image. The deployment has the label app: taskmaster. The pods use the environment variable MONGO\_URL to connect to the MongoDB database via the ClusterIP Service mongodb-service.

2. Verify that the deployment is created and running by running:

kubectl get deployments

You should see something like this:

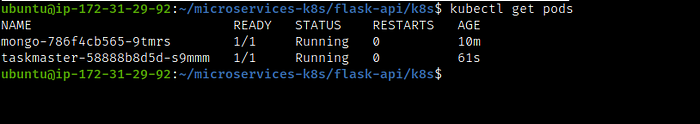


This shows that you have one deployment named taskmaster-deployment that has two replicas ready and up-to-date.

3. Verify that the pods are running by running:

kubectl get pods

You should see something like this:



This shows that you have two pods named taskmaster-<random-string> that are running and ready.

You have successfully deployed the taskmaster service for our microservices application. In the next section, we will deploy the NodePort Service for taskmaster. This will allow us to access our web app from outside the cluster.

**Deploying the NodePort Service for Taskmaster**

To deploy the NodePort Service for taskmaster, follow these steps:

1. Apply the YAML file that defines the service by running:

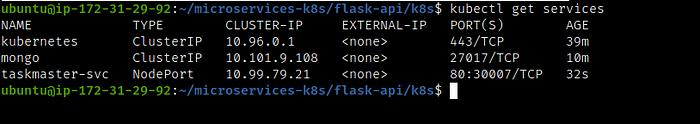
kubectl apply -f taskmaster-svc.yml

This will create a service object named taskmaster that exposes port 5000 of our pods as port 30007 on each node’s IP address. The service selects the pods that have the label app: taskmaster. The type: NodePort specifies that this is a NodePort Service.

2. Verify that the service is created by running:

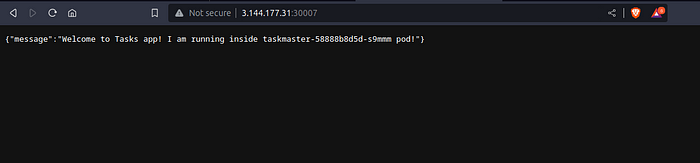
kubectl get services

You should see something like this:



This shows that you have one NodePort Service named taskmaster that exposes port 5000 of your pods as port 30007 on each node’s IP address.

3. Access the taskmaster web app from your browser by going to http://<node-ip>:30007/. You can use any node’s IP address, either the master or the worker. You should see something like this:



This shows that you have successfully accessed your taskmaster web app via the NodePort Service from outside your cluster.

You have successfully deployed the NodePort Service for taskmaster. This completes our microservices application deployment on Kubernetes.

**Summary and Conclusion**

In this blog post, we learned how to deploy a microservices application on AWS EC2 using Kubernetes, which is a popular platform for managing containerized applications. We covered the following topics:

* Setting up AWS EC2 Instances: We created two t2.medium instances on AWS EC2, one for the master node and one for the worker node. We also configured the security groups and SSH keys for these instances.
* Installing Kubernetes on AWS EC2 Instances: We installed Kubernetes on both instances using kubeadm, kubelet, and kubectl. We also initialized the master node and joined the worker node to the cluster.
* Deploying the Microservices Application on Kubernetes: We deployed a microservices application that consists of a MongoDB database and a taskmaster service. The taskmaster service is a web app that allows us to create and manage tasks. The app is written in Node.js and uses MongoDB as the database. We deployed the persistent volume and persistent volume claim, the MongoDB database, the ClusterIP service for MongoDB, the taskmaster service, and the NodePort service for taskmaster.

By the end of this blog post, we had a working microservices application running on AWS EC2 using Kubernetes. We also learned some important concepts and commands related to Kubernetes, such as pods, services, deployments, etc.

I hope you enjoyed this blog post and learned something new. If you have any questions or feedback, please feel free to leave a comment below.

If you want to follow my journey of learning DevOps, you can check out my [GitHub](https://github.com/ajitfawade) and my [LinkedIn](https://www.linkedin.com/in/ajitfawade/) profile.

Thank you for reading and stay tuned for more!