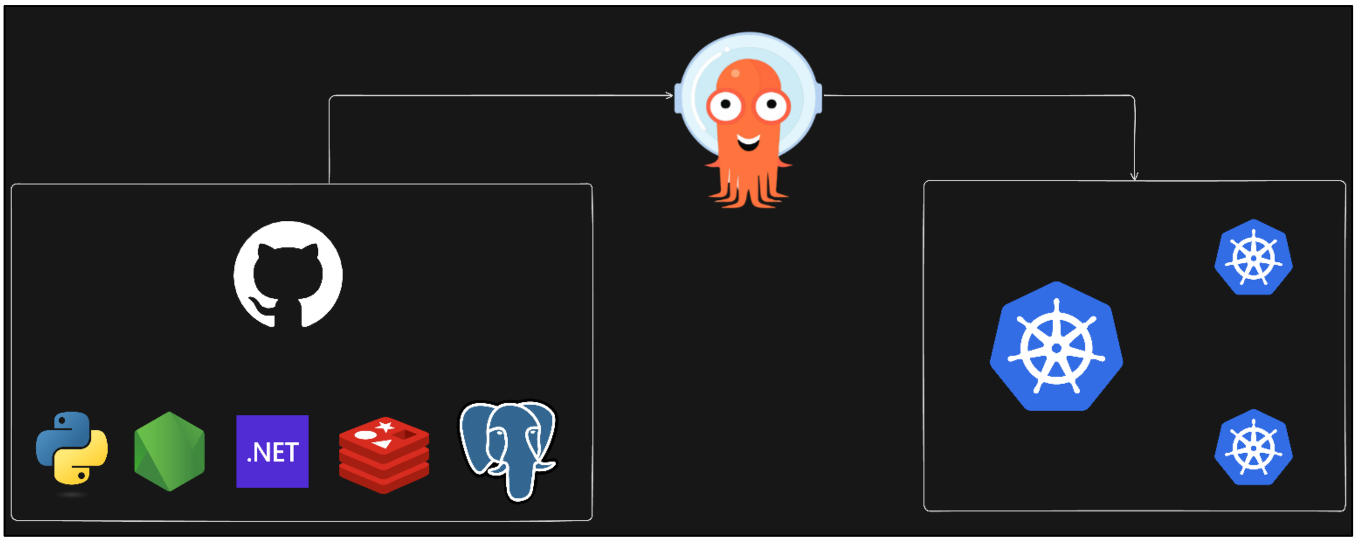
Automated Deployment of Scalable Applications on AWS EC2 with Kubernetes and Argo CD

This project focuses on deploying scalable applications on AWS EC2 using Kubernetes for orchestration and Argo CD for continuous delivery. Deployments are efficiently managed through the Kubernetes dashboard, ensuring optimal resource utilization and seamless scaling.

Architecture:



In this project, the following steps will be covered:

- Launch an AWS EC2 instance.
- Install Docker and Kind.
- Create a Kubernetes cluster using Kind.
- Install and access kubectl.
- Set up the Kubernetes Dashboard.
- Install and configure Argo CD.
- Connect and manage your Kubernetes cluster with Argo CD.

This project consists of several key components working together:

1. **A front-end web app in Python**: This web application allows users to vote between two options. It serves as the interface where users interact and submit their votes.
2. **A Redis instance**: Redis is used as a message broker to collect new votes from users. It temporarily stores these votes before they are processed by other services in the system.
3. **A .NET worker**: This worker service consumes the votes from Redis, processes them, and then stores the results in a database for further retrieval and analysis.

4. **A Postgres database**: The .NET worker stores the processed votes in a Postgres database. The database is backed by a Docker volume to ensure that the data persists even if the container is restarted.
5. **A Node.js web app**: This application displays the real-time results of the voting process. It fetches data from the Postgres database and updates the displayed results as votes are recorded.

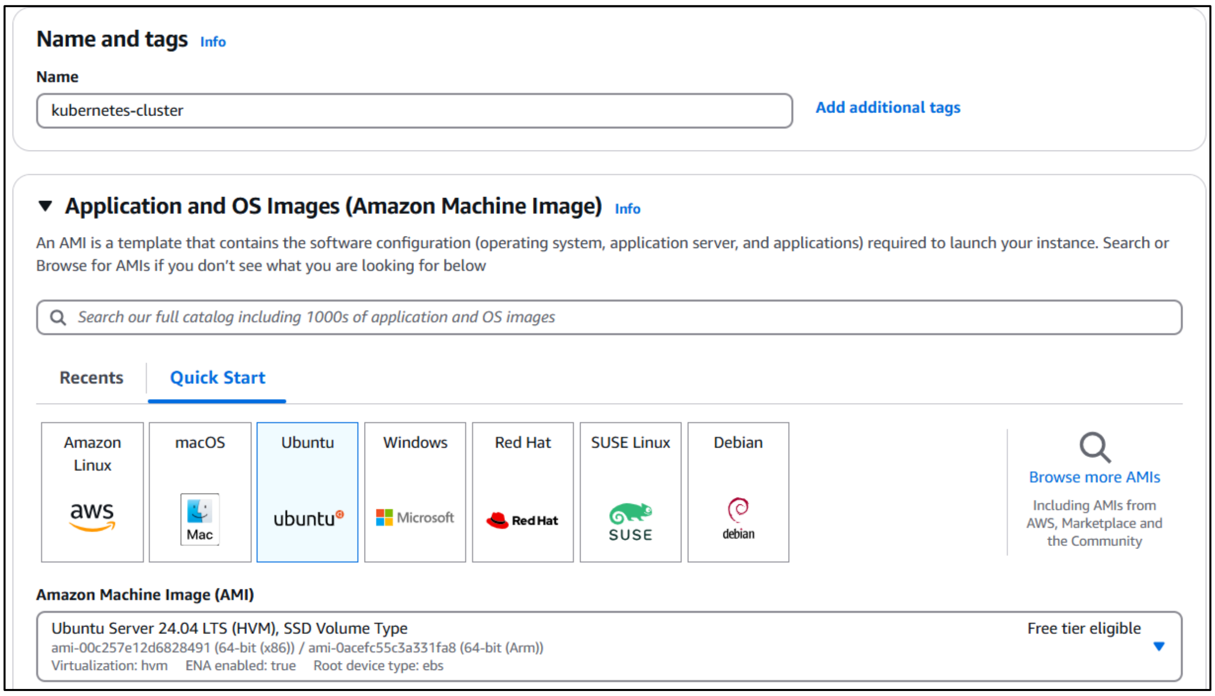
These components interact seamlessly to provide a full-stack voting system where users can cast votes, vote data is stored, and results are displayed in real-time.

Preliminary Steps: Preparing the Project from the External GitHub Repository

- Before starting with the deployment and other project-related tasks, you need to clone the project from the external GitHub repository available at **https://github.com/LondheShubham153/k8s-kind-voting-app.git**. This repository contains the application that we will deploy. After cloning it to your local computer, you'll push the project to your own GitHub account. Once it's in your repository, we will proceed to execute the project from there.
- Once we have successfully cloned the repository to our local computer and pushed it to our own GitHub account, we can proceed with the next step: **Launching an AWS EC2 instance**. This instance will serve as the server where we will deploy and manage our application, setting up the necessary environment for Docker, Kubernetes, and Argo CD.

Launch an AWS EC2 instance

- **Enter the Name of the Instance**, eg: kubernetes-cluster
- Choose **Ubuntu Server 24.04 LTS (HVM)** under **Amazon Machine Image(AMI)**


Name and tags Info

Name

kubernetes-cluster

Add additional tags

Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

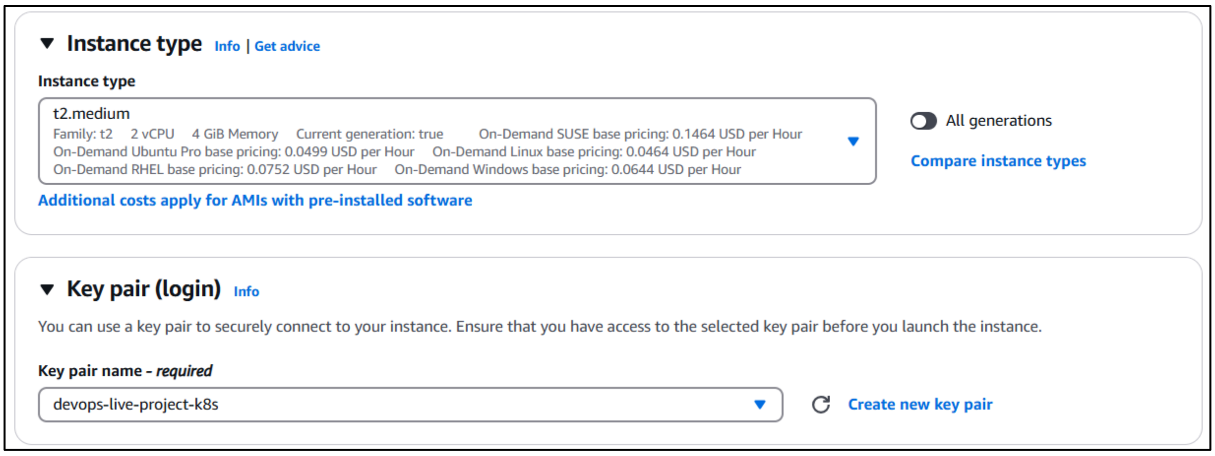
Amazon Machine Image (AMI)

Ubuntu Server 24.04 LTS (HVM), SSD Volume Type
ami-00c257e12d6828491 (64-bit (x86)) / ami-0acefc55c3a331fa8 (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs

Free tier eligible

- Choose **t2.medium** under **Instance type**.

- Under **Key pair (login)**, give your key pair name or create a new key pair eg: devops-live-project-k8s is my keypair.



Instance type Info | Get advice

Instance type

t2.medium
Family: t2 2 vCPU 4 GiB Memory Current generation: true On-Demand SUSE base pricing: 0.1464 USD per Hour
On-Demand Ubuntu Pro base pricing: 0.0499 USD per Hour On-Demand Linux base pricing: 0.0464 USD per Hour
On-Demand RHEL base pricing: 0.0752 USD per Hour On-Demand Windows base pricing: 0.0644 USD per Hour

All generations
Compare instance types

Additional costs apply for AMIs with pre-installed software

Key pair (login) Info

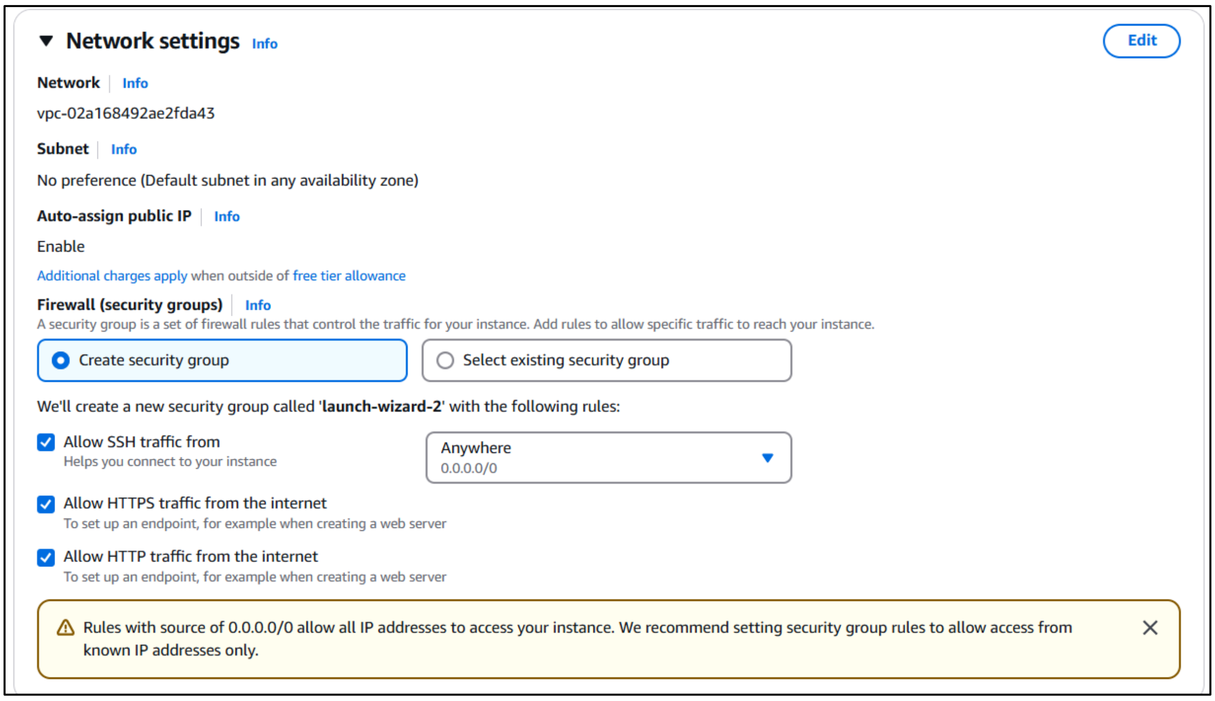
You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*

devops-live-project-k8s

Create new key pair

- In the network settings, choose the default VPC and subnet, and enable the option to auto-assign a public IP. For the firewall security groups, create a new security group and configure the following rules: allow **SSH** traffic from anywhere (port 22) to enable secure remote access, allow **HTTP** traffic from the internet (port 80) to support standard web traffic, and allow **HTTPS** traffic from the internet (port 443) for secure, encrypted web connections.



Network settings Info

Edit

Network | Info
vpc-02a168492ae2fda43

Subnet | Info
No preference (Default subnet in any availability zone)

Auto-assign public IP | Info
Enable

Additional charges apply when outside of free tier allowance

Firewall (security groups) | Info
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

● Create security group ○ Select existing security group

We'll create a new security group called '**launch-wizard-2**' with the following rules:

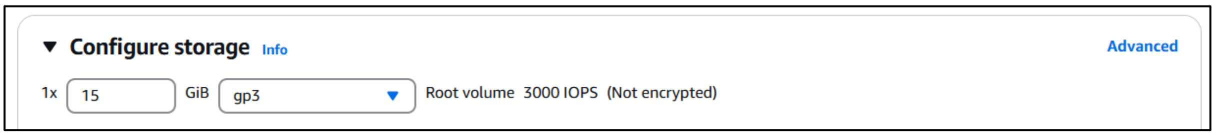
✓ Allow SSH traffic from — Anywhere 0.0.0.0/0
Helps you connect to your instance

✓ Allow HTTPS traffic from the internet
To set up an endpoint, for example when creating a web server

✓ Allow HTTP traffic from the internet
To set up an endpoint, for example when creating a web server

Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

- Configure the storage settings by setting the root volume size to **15 GiB** to ensure sufficient space for the application and related dependencies.

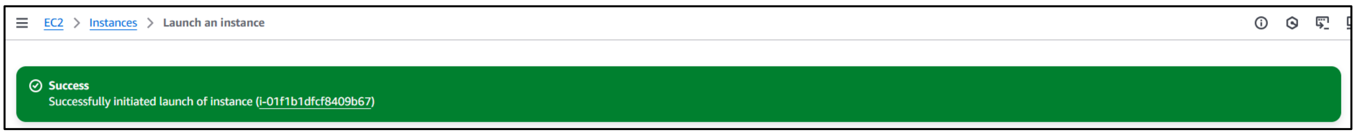


Configure storage Info

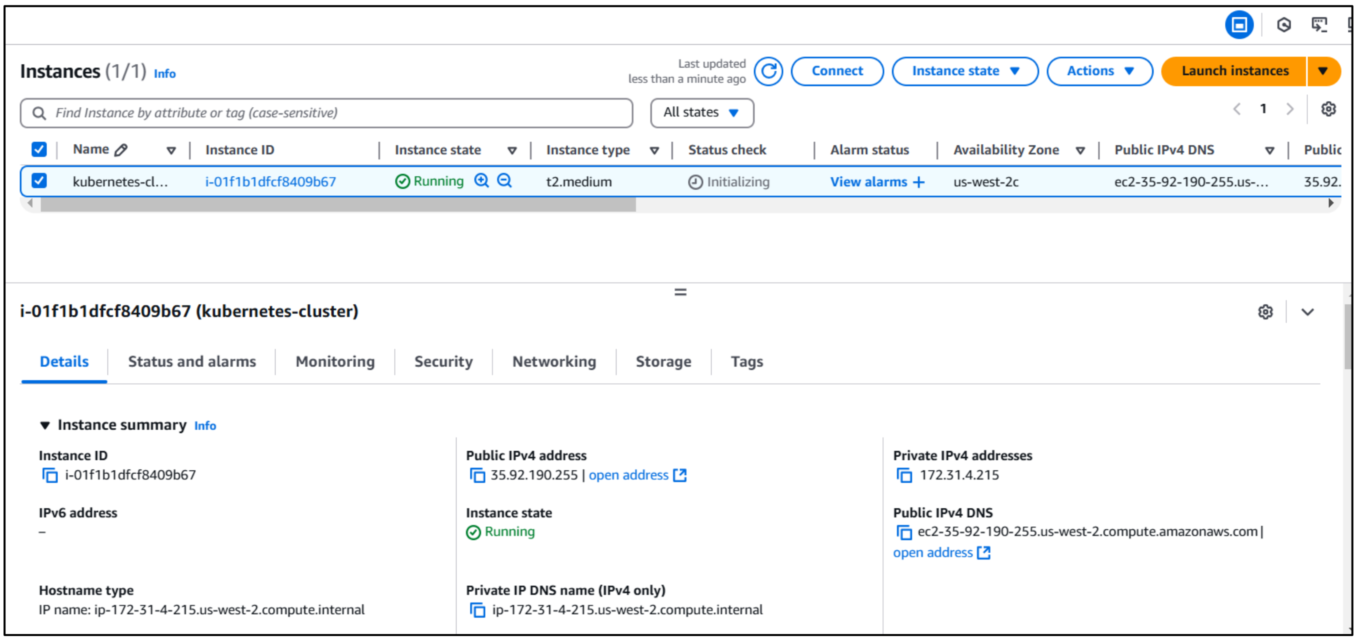
Advanced

1x 15 GiB gp3 Root volume 3000 IOPS (Not encrypted)

- After finalizing the storage configuration and reviewing all settings, proceed to **launch the EC2 instance**.


EC2 > Instances > Launch an instance

Success
Successfully initiated launch of instance (i-01f1b1dfcf8409b67)



Instances (1/1) Info

Last updated less than a minute ago

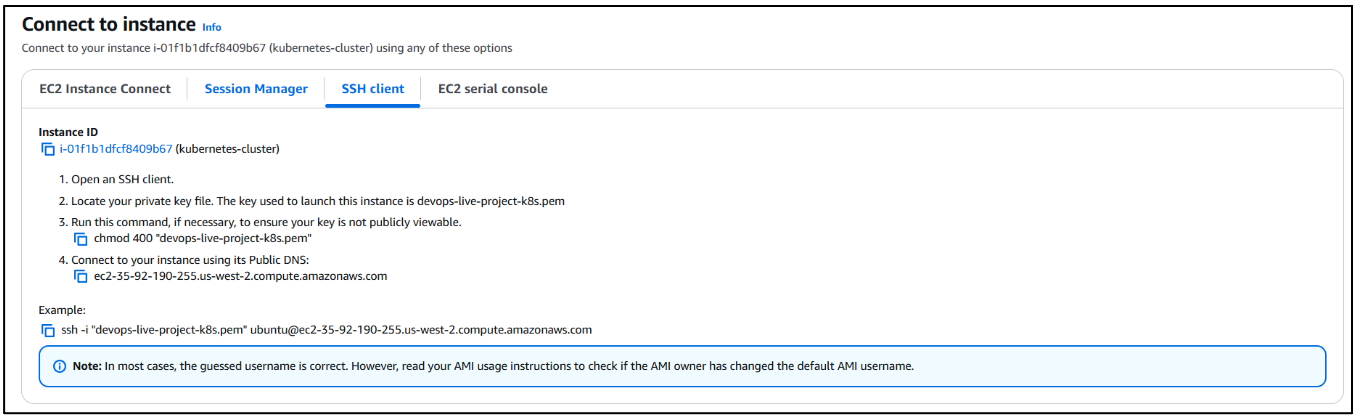
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public
kubernetes-cl...	i-01f1b1dfcf8409b67	Running	t2.medium	Initializing	View alarms +	us-west-2c	ec2-35-92-190-255.us-...	35.92.

i-01f1b1dfcf8409b67 (kubernetes-cluster)

Details | Status and alarms | Monitoring | Security | Networking | Storage | Tags

Instance summary Info

Instance ID: i-01f1b1dfcf8409b67
Public IPv4 address: 35.92.190.255 | open address
Private IPv4 addresses: 172.31.4.215
IPv6 address: –
Instance state: Running
Public IPv4 DNS: ec2-35-92-190-255.us-west-2.compute.amazonaws.com | open address
Hostname type: IP name: ip-172-31-4-215.us-west-2.compute.internal
Private IP DNS name (IPv4 only): ip-172-31-4-215.us-west-2.compute.internal


Connect to instance Info

Connect to your instance i-01f1b1dfcf8409b67 (kubernetes-cluster) using any of these options

EC2 Instance Connect | Session Manager | SSH client | EC2 serial console

Instance ID
i-01f1b1dfcf8409b67 (kubernetes-cluster)

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is devops-live-project-k8s.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
   chmod 400 "devops-live-project-k8s.pem"
4. Connect to your instance using its Public DNS:
   ec2-35-92-190-255.us-west-2.compute.amazonaws.com

Example:
ssh -i "devops-live-project-k8s.pem" ubuntu@ec2-35-92-190-255.us-west-2.compute.amazonaws.com

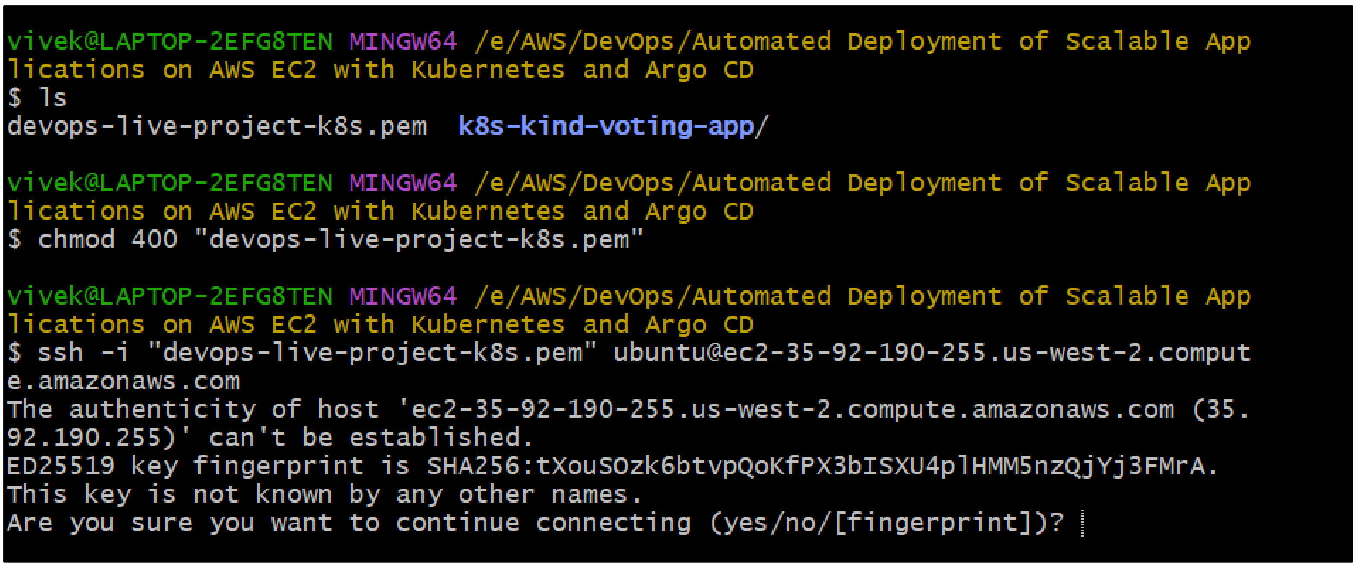
Note: In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

- To connect to the EC2 instance using an SSH client, open the **Git Bash terminal** on your local computer. Since we've already created the key pair named **`devops-live-project-k8s`** during the instance setup and downloaded it to our local machine, navigate to the directory where the key pair file (`devops-live-project-k8s.pem`) is stored. From there, initiate the SSH connection to the EC2 instance, allowing you to access the instance's command line interface.
- Run the following command in the **Git Bash terminal** to set the correct permissions for your key pair file, ensuring it is **not publicly viewable**:

```
chmod 400 "devops-live-project-k8s.pem"
```

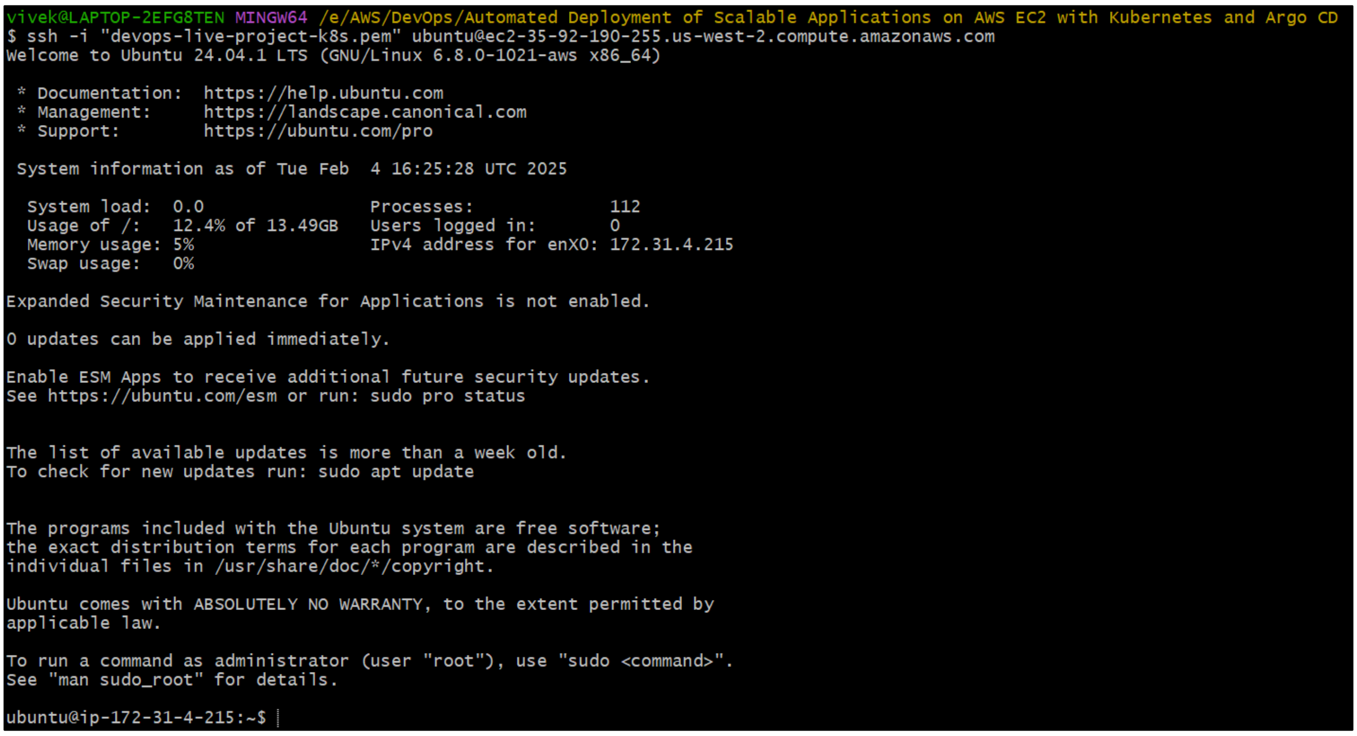
This command makes the file read-only for the owner and removes all permissions for others, which is a security requirement by AWS to prevent unauthorized access. Without these restricted permissions, the SSH connection to your EC2 instance may be refused. Make sure to execute this command in the directory where the key pair file is stored or provide the full file path if it's located elsewhere.

- To connect to your EC2 instance, run the command `ssh -i "devops-live-project-k8s.pem" ubuntu@ec2-35-92-190-255.us-west-2.compute.amazonaws.com` in the Git Bash terminal. This command uses SSH to establish a secure connection, with the `-i` flag specifying the private key file (`devops-live-project-k8s.pem`) required for authentication. The `ubuntu@` part indicates that you're logging in as the default Ubuntu user, and the provided public DNS (`ec2-35-92-190-255.us-west-2.compute.amazonaws.com`) directs the connection to your specific EC2 instance. Upon running the command, you may be prompted to confirm the connection;



```
vivek@LAPTOP-2EFG8TEN MINGW64 /e/AWS/DevOps/Automated Deployment of Scalable App
lications on AWS EC2 with Kubernetes and Argo CD
$ ls
devops-live-project-k8s.pem  k8s-kind-voting-app/

vivek@LAPTOP-2EFG8TEN MINGW64 /e/AWS/DevOps/Automated Deployment of Scalable App
lications on AWS EC2 with Kubernetes and Argo CD
$ chmod 400 "devops-live-project-k8s.pem"

vivek@LAPTOP-2EFG8TEN MINGW64 /e/AWS/DevOps/Automated Deployment of Scalable App
lications on AWS EC2 with Kubernetes and Argo CD
$ ssh -i "devops-live-project-k8s.pem" ubuntu@ec2-35-92-190-255.us-west-2.comput
e.amazonaws.com
The authenticity of host 'ec2-35-92-190-255.us-west-2.compute.amazonaws.com (35.
92.190.255)' can't be established.
ED25519 key fingerprint is SHA256:tXouSOzk6btvpQoKfPX3bISXU4plHMM5nzQjYj3FMrA.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

type `yes` to proceed and gain access to the instance's command-line interface.



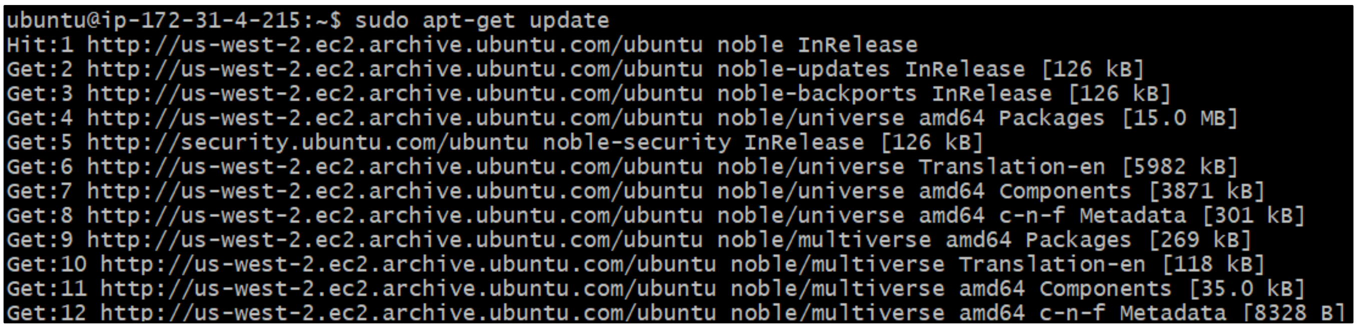
```
vivek@LAPTOP-2EFG8TEN MINGW64 /e/AWS/DevOps/Automated Deployment of Scalable Applications on AWS EC2 with Kubernetes and Argo CD
$ ssh -i "devops-live-project-k8s.pem" ubuntu@ec2-35-92-190-255.us-west-2.compute.amazonaws.com
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-1021-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Tue Feb  4 16:25:28 UTC 2025

  System load:  0.0               Processes:             112
  Usage of /:   12.4% of 13.49GB  Users logged in:       0
  Memory usage: 5%                IPv4 address for enX0: 172.31.4.215
  Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-4-215:~$
```

Install Docker and Kind.

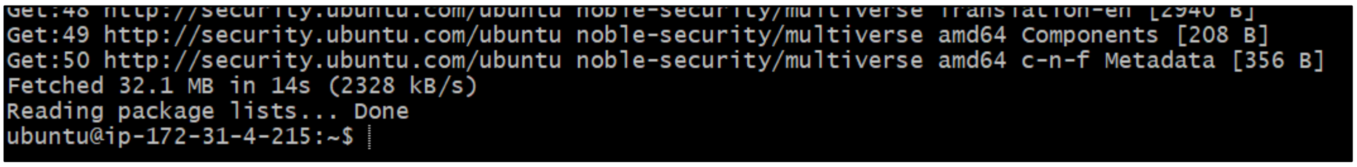
- Now that we are connected to our EC2 instance, the next step is to **install Docker**, which is essential for running containerized applications. Since **Kind (Kubernetes IN Docker)** operates by creating Kubernetes clusters inside Docker containers, Docker must be installed and running on the instance before we proceed with installing Kind.
- To set up Docker on your EC2 instance, follow these steps:

1. **Update the package list** to ensure the latest repository information:

```
sudo apt-get update
```



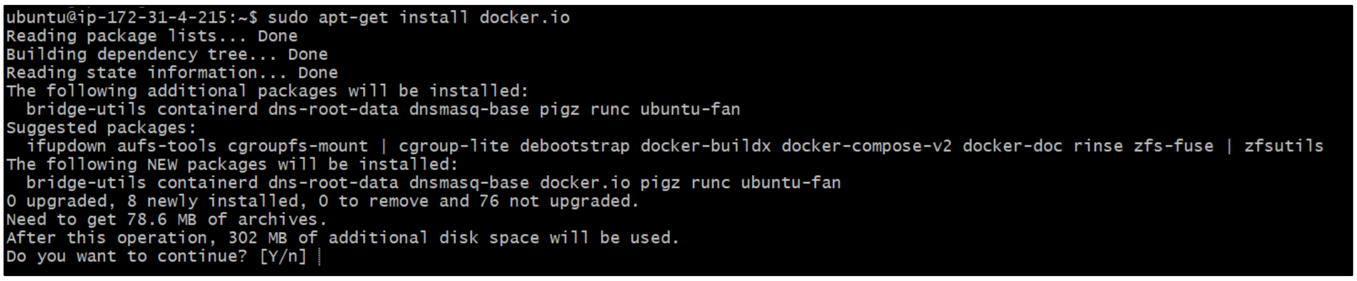
```
ubuntu@ip-172-31-4-215:~$ sudo apt-get update
Hit:1 http://us-west-2.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://us-west-2.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://us-west-2.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:4 http://us-west-2.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]
Get:5 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:6 http://us-west-2.ec2.archive.ubuntu.com/ubuntu noble/universe Translation-en [5982 kB]
Get:7 http://us-west-2.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Components [3871 kB]
Get:8 http://us-west-2.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 c-n-f Metadata [301 kB]
Get:9 http://us-west-2.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [269 kB]
Get:10 http://us-west-2.ec2.archive.ubuntu.com/ubuntu noble/multiverse Translation-en [118 kB]
Get:11 http://us-west-2.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Components [35.0 kB]
Get:12 http://us-west-2.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 c-n-f Metadata [8328 B]
```



```
Get:48 http://security.ubuntu.com/ubuntu noble-security/multiverse Translation-en [2940 B]
Get:49 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Components [208 B]
Get:50 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 c-n-f Metadata [356 B]
Fetched 32.1 MB in 14s (2328 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-4-215:~$
```

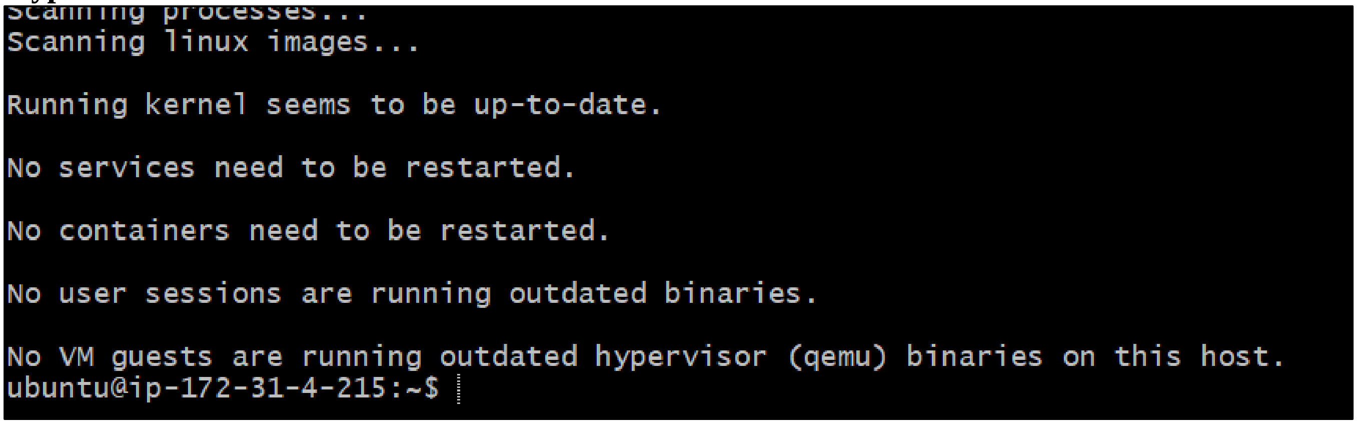
2. **Install Docker** by running the following command:

```
sudo apt-get install docker.io
```



```
ubuntu@ip-172-31-4-215:~$ sudo apt-get install docker.io
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-buildx docker-compose-v2 docker-doc rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io pigz runc ubuntu-fan
0 upgraded, 8 newly installed, 0 to remove and 76 not upgraded.
Need to get 78.6 MB of archives.
After this operation, 302 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

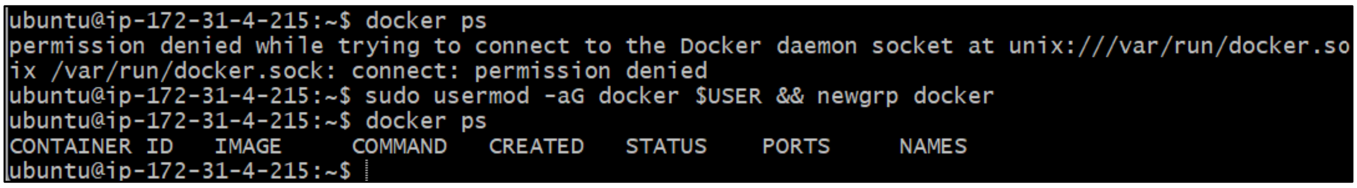
Type Y



```
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-4-215:~$
```

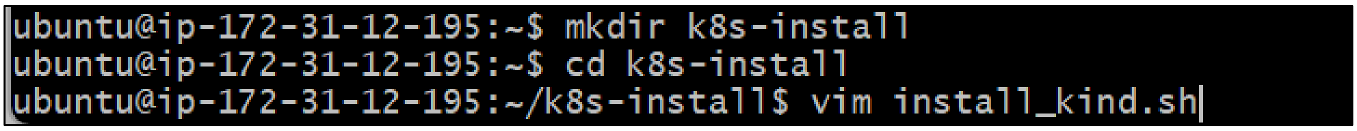
After completing these steps, Docker will be installed and ready for use on your EC2 instance.

- You can check the containers running in Docker with the command `docker ps`. However, if you encounter the error **"permission denied"** while trying to connect to the Docker daemon at `unix:///var/run/docker.sock`, it means your user does not have the necessary permissions to access the Docker socket.
- To resolve this, run the command `sudo usermod -aG docker $USER && newgrp docker`.
  The `sudo usermod -aG docker $USER` part of the command adds your user ($USER) to the docker group, enabling your user to execute Docker commands.
  The `&&` operator ensures that the second command only runs if the first one succeeds.
  The `newgrp` docker command refreshes the group membership in the current session, so you don't have to log out and log back in for the changes to take effect.
  After running this command, you should have the necessary permissions to interact with Docker without encountering the "permission denied" error.
- Once you run the necessary command to add your user to the Docker group and refresh the group membership, you should be able to execute the `docker ps` command without any issues. The command will now run successfully and display the list of available containers running on your system.



```
ubuntu@ip-172-31-4-215:~$ docker ps
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.so
ix /var/run/docker.sock: connect: permission denied
ubuntu@ip-172-31-4-215:~$ sudo usermod -aG docker $USER && newgrp docker
ubuntu@ip-172-31-4-215:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu@ip-172-31-4-215:~$
```

- Now that Docker is set up on your EC2 instance, you can proceed with installing Kind (Kubernetes IN Docker) using your shell script. Kind allows you to create a local Kubernetes cluster inside Docker containers, which is a great way to run and test Kubernetes clusters without needing a full setup.
- To run the shell script for installing Kind, follow these steps:
  - Setup a folder named **`k8s-install`** with the command `mkdir k8s-install`.
  - Access into the folder with `cd k8s-install`,
  - Create the **`install_kind.sh`** file by running `vim install_kind.sh` inside the folder.

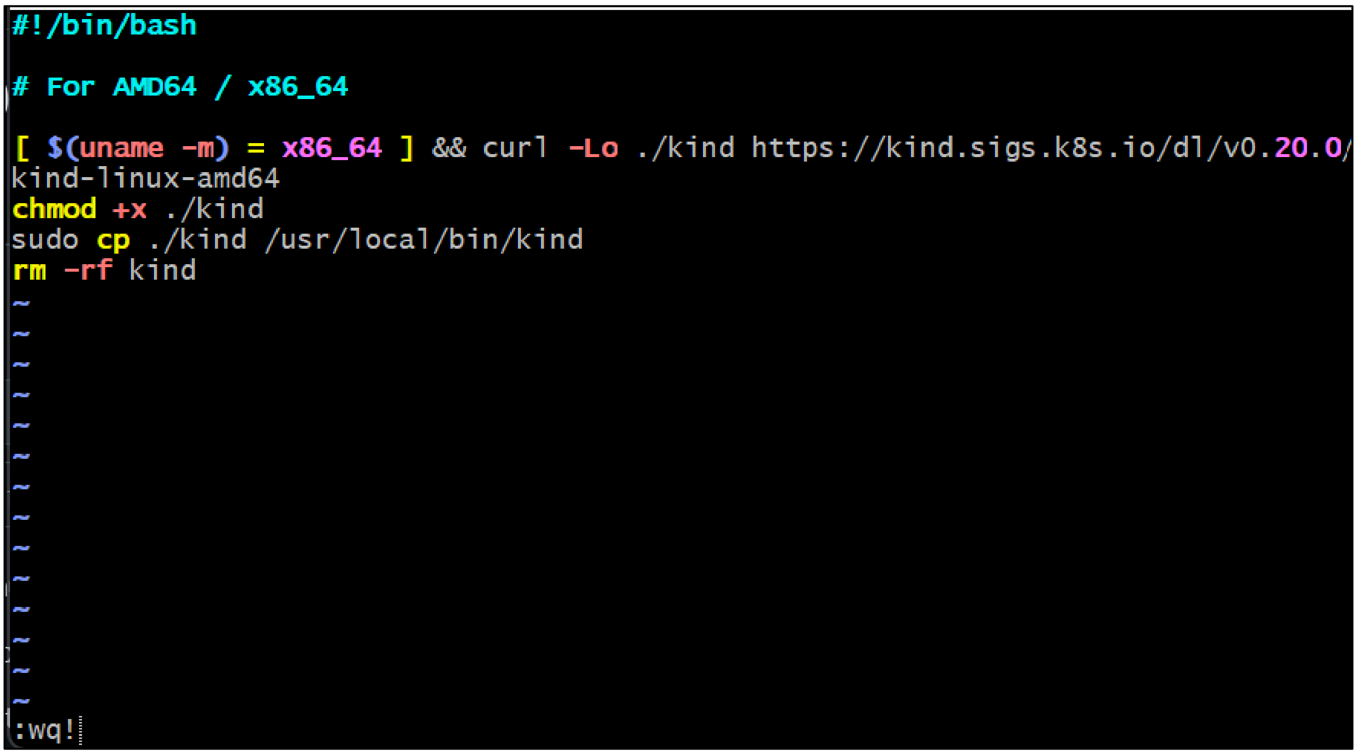


```
ubuntu@ip-172-31-12-195:~$ mkdir k8s-install
ubuntu@ip-172-31-12-195:~$ cd k8s-install
ubuntu@ip-172-31-12-195:~/k8s-install$ vim install_kind.sh
```

  - Paste the shell script, which you have copied from the GitHub repository for installing Kind, into the file. the script is as follows:

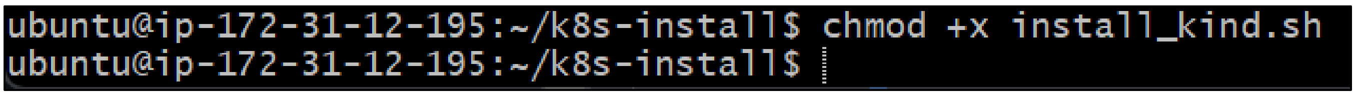
```
#!/bin/bash

# For AMD64 / x86_64

[ $(uname -m) = x86_64 ] && curl -Lo ./kind
https://kind.sigs.k8s.io/dl/v0.20.0/kind-linux-amd64
chmod +x ./kind
sudo cp ./kind /usr/local/bin/kind
rm -rf kind
```

- Save the script and exit vim.



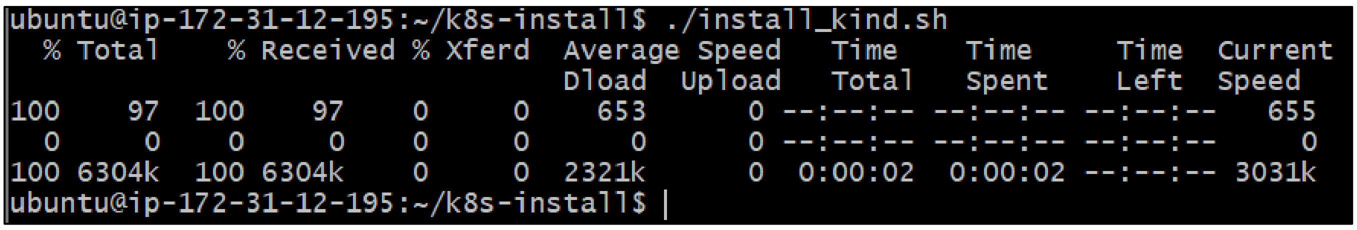
```
#!/bin/bash

# For AMD64 / x86_64

[ $(uname -m) = x86_64 ] && curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.20.0/
kind-linux-amd64
chmod +x ./kind
sudo cp ./kind /usr/local/bin/kind
rm -rf kind
~
~
~
~
~
~
~
~
~
~
~
~
~
~
:wq!
```

- Grant executable permissions using `chmod +x install_kind.sh`



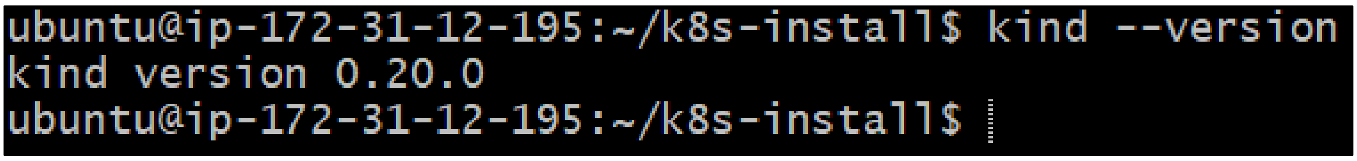
```
ubuntu@ip-172-31-12-195:~/k8s-install$ chmod +x install_kind.sh
ubuntu@ip-172-31-12-195:~/k8s-install$
```

- Execute the script with `./install_kind.sh`.



```
ubuntu@ip-172-31-12-195:~/k8s-install$ ./install_kind.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    97  100    97    0     0    653      0 --:--:-- --:--:-- --:--:--   655
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 6304k  100 6304k    0     0  2321k      0  0:00:02  0:00:02 --:--:-- 3031k
ubuntu@ip-172-31-12-195:~/k8s-install$
```

- You can verify the installed version of Kind by running the command `kind --version`.



```
ubuntu@ip-172-31-12-195:~/k8s-install$ kind --version
kind version 0.20.0
ubuntu@ip-172-31-12-195:~/k8s-install$
```

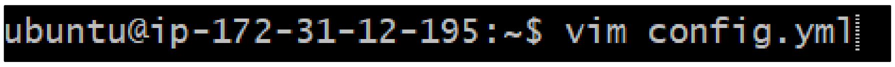
Create a Kubernetes cluster using Kind.

- Now, we can create the Kubernetes cluster in Kind using the `config.yml` file available in the GitHub repository.
- The script inside the file is as follows:

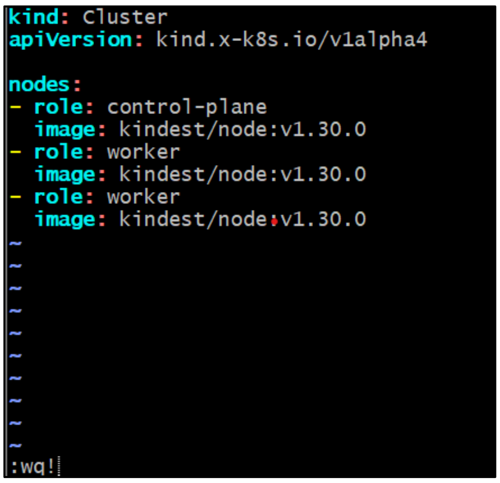
```
kind: Cluster      # Specifies that the resource being defined is a Kubernetes Cluster.
apiVersion: kind.x-k8s.io/v1alpha4   # Defines the API version for Kind configuration compatibility.

nodes:             # Begins the list of nodes to be created in the cluster.
- role: control-plane         # Defines the first node as the control plane, responsible for managing the cluster.
  image: kindest/node:v1.30.0 # Specifies the Kubernetes version (v1.30.0) for the control plane node.

- role: worker     # Defines the second node as a worker node to run application workloads.
  image: kindest/node:v1.30.0 # Specifies the Kubernetes version (v1.30.0) for the worker node.

- role: worker     # Defines the third node as an additional worker node for better scalability.
  image: kindest/node:v1.30.0 # Specifies the Kubernetes version (v1.30.0) for the additional worker node.
```

This configuration sets up a multi-node Kubernetes cluster with one control plane and two worker nodes, all running Kubernetes version **v1.30.0**.

- Create the `config.yml` file in your EC2 instance by running the command `vim config.yml`, then paste the Kubernetes cluster configuration script mentioned above into the file. Afterward, save and exit Vim to proceed with creating the cluster using this configuration.



```
ubuntu@ip-172-31-12-195:~$ vim config.yml
```

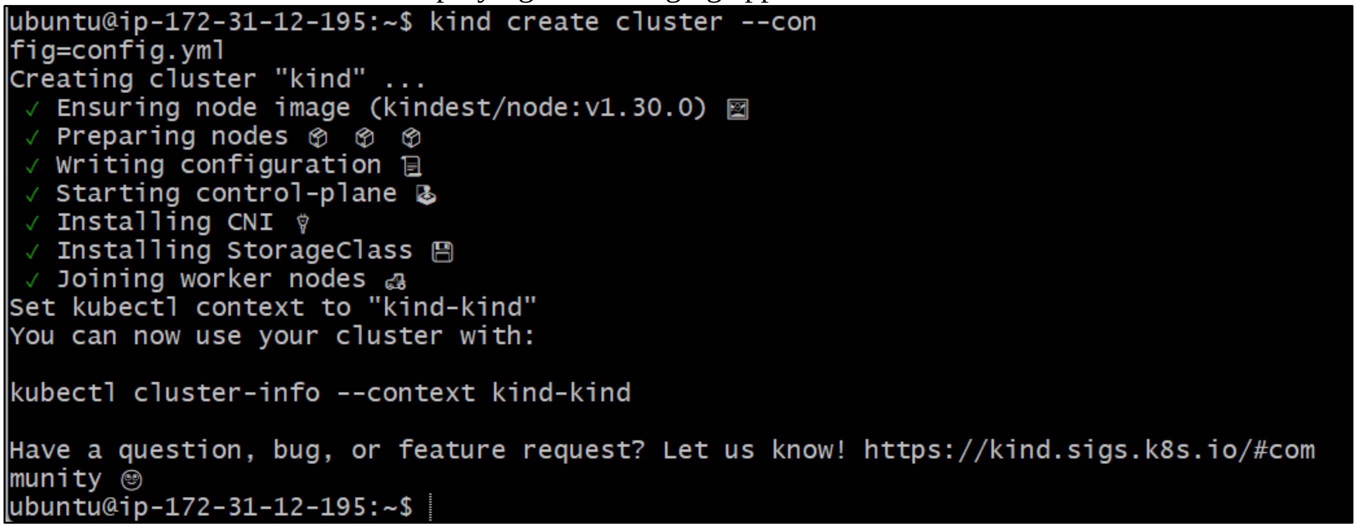


```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4

nodes:
- role: control-plane
  image: kindest/node:v1.30.0
- role: worker
  image: kindest/node:v1.30.0
- role: worker
  image: kindest/node:v1.30.0
~
~
~
~
~
~
~
~
~
~
:wq!
```

- Now that the `config.yml` file is set up, you can create a 3-node Kubernetes cluster using Kind by executing the command:

  `kind create cluster --config=config.yml.`

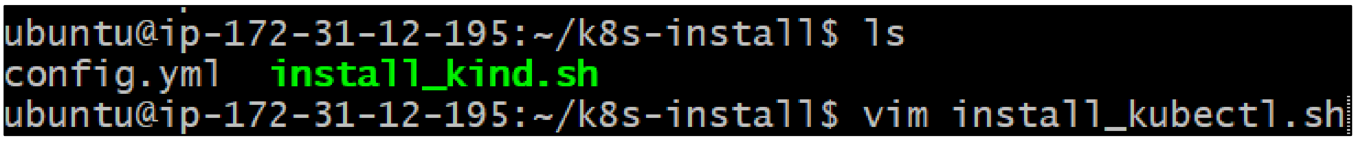
  This command directs Kind to read the configuration details from the `config.yml` file and set up the cluster accordingly. It will create one control plane node and two worker nodes, all running the specified Kubernetes version, providing a functional multi-node cluster environment for deploying and managing applications.



```
ubuntu@ip-172-31-12-195:~$ kind create cluster --con
fig=config.yml
Creating cluster "kind" ...
 ✓ Ensuring node image (kindest/node:v1.30.0) 🖼
 ✓ Preparing nodes 📦 📦 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
 ✓ Joining worker nodes 🚜
Set kubectl context to "kind-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-kind

Have a question, bug, or feature request? Let us know! https://kind.sigs.k8s.io/#com
munity 🙂
ubuntu@ip-172-31-12-195:~$
```

Install and access kubectl.

- To interact with and manage the Kubernetes cluster and its nodes, we need to use **kubectl**, the command-line tool for Kubernetes. Therefore, the next step is to install **kubectl** on our EC2 instance, which will allow us to run commands to deploy applications, inspect cluster resources, and manage the cluster efficiently.
- Now that the Kubernetes cluster is set up using Kind, the next step is to install kubectl, the command-line tool used to interact with Kubernetes clusters. It allows you to deploy applications, inspect and manage cluster resources, and view logs efficiently. To install kubectl, follow these steps:
- To run the shell script for installing kubectl, follow these steps:
  - Create the **`install_kubectl.sh`** file by running `vim install_kubectl.sh` inside the folder.



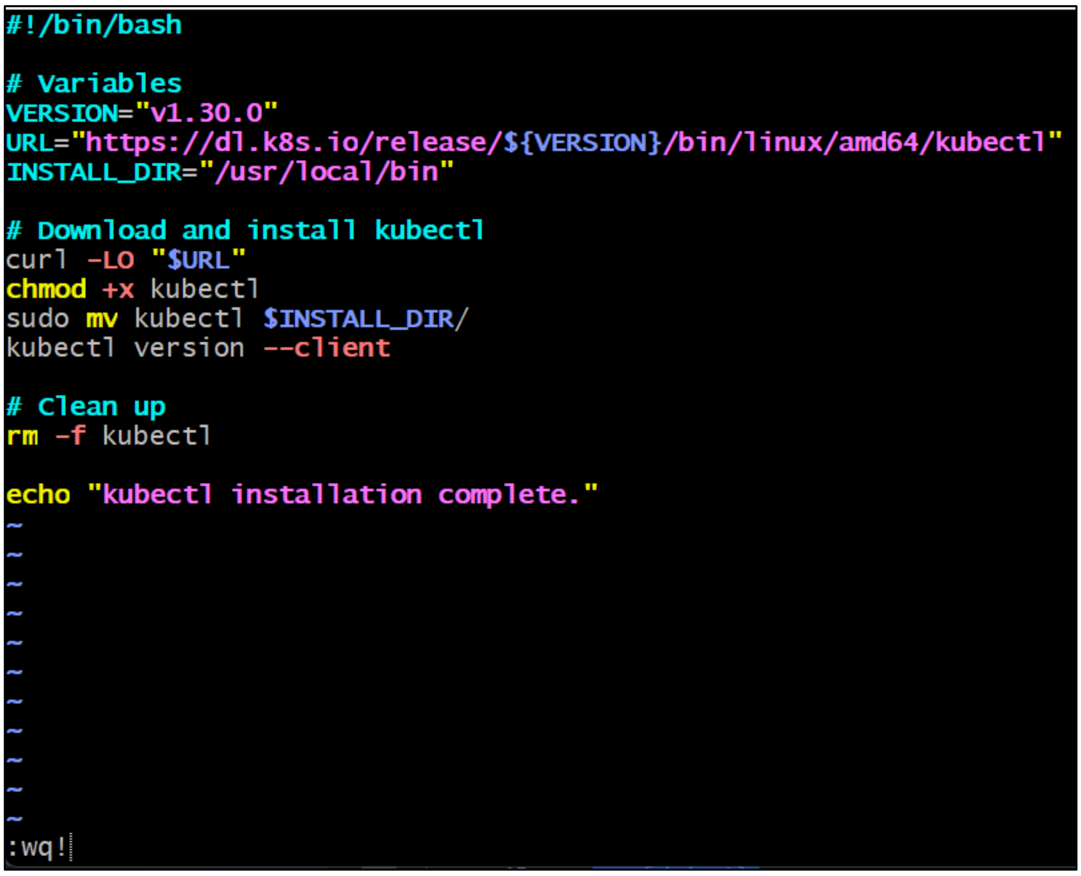
```
ubuntu@ip-172-31-12-195:~/k8s-install$ ls
config.yml   install_kind.sh
ubuntu@ip-172-31-12-195:~/k8s-install$ vim install_kubectl.sh
```

  - Paste the shell script, which you have copied from the GitHub repository for installing kubectl, into the file. the script is as follows:

```
#!/bin/bash

# Variables
VERSION="v1.30.0"
```

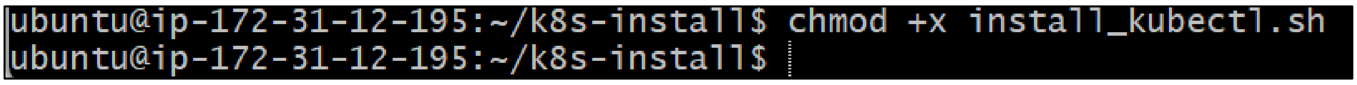
```
URL="https://dl.k8s.io/release/${VERSION}/bin/linux/amd64/kubec
tl"
INSTALL_DIR="/usr/local/bin"

# Download and install kubectl
curl -LO "$URL"
chmod +x kubectl
sudo mv kubectl $INSTALL_DIR/
kubectl version --client

# Clean up
rm -f kubectl

echo "kubectl installation complete."
```

- Save the script and exit vim.



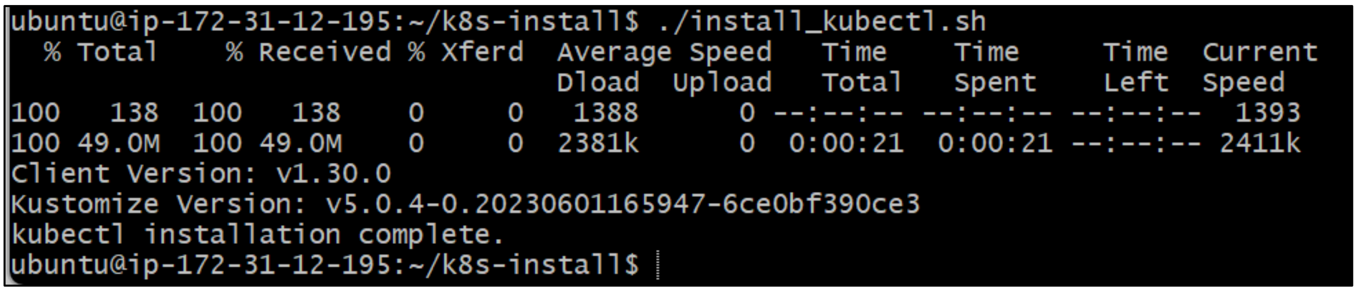
```
#!/bin/bash

# Variables
VERSION="v1.30.0"
URL="https://dl.k8s.io/release/${VERSION}/bin/linux/amd64/kubectl"
INSTALL_DIR="/usr/local/bin"

# Download and install kubectl
curl -LO "$URL"
chmod +x kubectl
sudo mv kubectl $INSTALL_DIR/
kubectl version --client

# Clean up
rm -f kubectl

echo "kubectl installation complete."
~
~
~
~
~
~
~
~
~
~
~
:wq!
```

- Grant executable permissions using `chmod +x install_kubectl.sh`



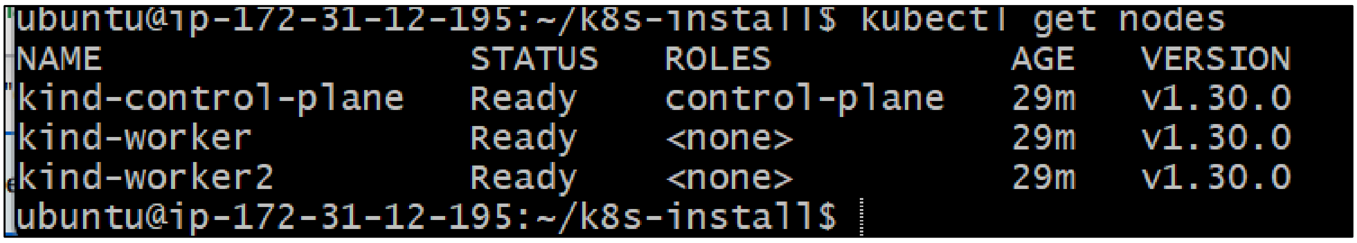
```
ubuntu@ip-172-31-12-195:~/k8s-install$ chmod +x install_kubectl.sh
ubuntu@ip-172-31-12-195:~/k8s-install$
```

- Execute the script with `./install_kubectl.sh`.



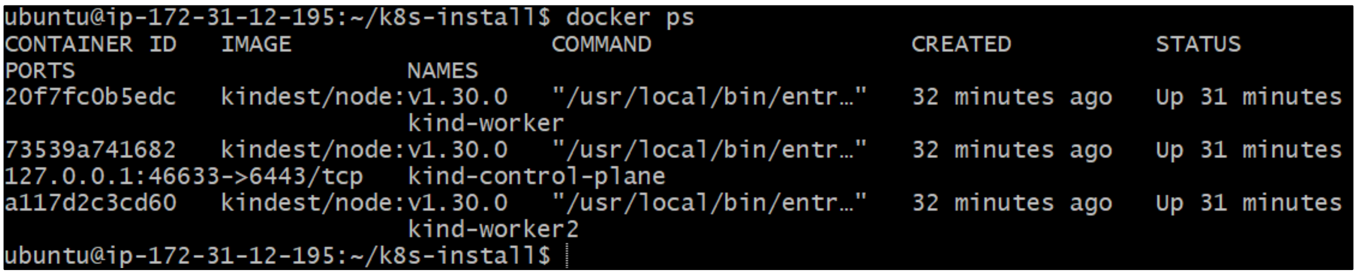
```
ubuntu@ip-172-31-12-195:~/k8s-install$ ./install_kubectl.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   138  100   138    0     0   1388      0 --:--:-- --:--:-- --:--:--  1393
100 49.0M  100 49.0M    0     0  2381k      0  0:00:21  0:00:21 --:--:-- 2411k
Client Version: v1.30.0
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
kubectl installation complete.
ubuntu@ip-172-31-12-195:~/k8s-install$
```

- Now, if you execute the command `kubectl get nodes`, you will observe the three nodes that were created in the Kubernetes cluster using Kind.



```
ubuntu@ip-172-31-12-195:~/k8s-install$ kubectl get nodes
NAME                 STATUS   ROLES           AGE   VERSION
kind-control-plane   Ready    control-plane   29m   v1.30.0
kind-worker          Ready    <none>          29m   v1.30.0
kind-worker2         Ready    <none>          29m   v1.30.0
ubuntu@ip-172-31-12-195:~/k8s-install$
```

- Execute the command `docker ps` to observe the Kind containers that have been created, which allow Kubernetes to run inside Docker.



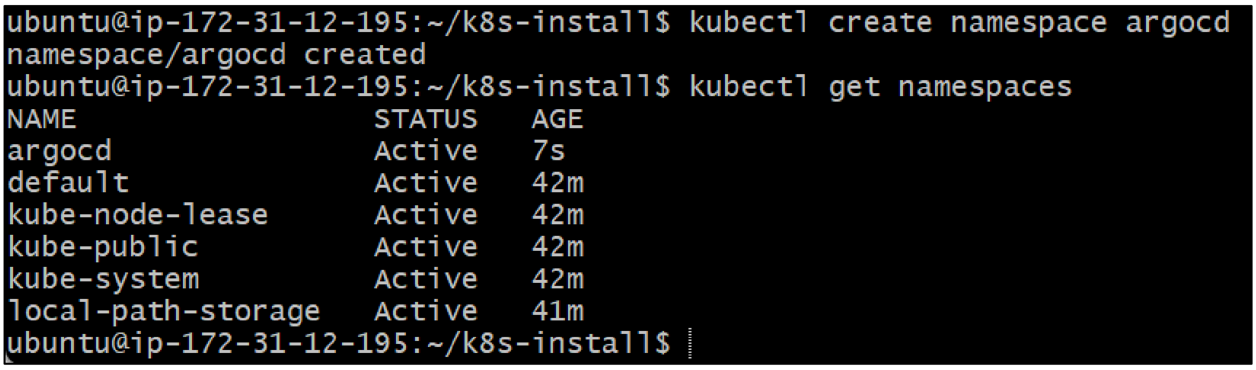
```
ubuntu@ip-172-31-12-195:~/k8s-install$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS
PORTS                       NAMES
20f7fc0b5edc   kindest/node:v1.30.0   "/usr/local/bin/entr…"   32 minutes ago   Up 31 minutes
                            kind-worker
73539a741682   kindest/node:v1.30.0   "/usr/local/bin/entr…"   32 minutes ago   Up 31 minutes
127.0.0.1:46633->6443/tcp   kind-control-plane
a117d2c3cd60   kindest/node:v1.30.0   "/usr/local/bin/entr…"   32 minutes ago   Up 31 minutes
                            kind-worker2
ubuntu@ip-172-31-12-195:~/k8s-install$
```

Install and configure Argo CD.

Now, let's proceed with setting up **Argo CD**. Argo CD is a declarative, GitOps continuous delivery tool for Kubernetes that enables the management of applications in a Kubernetes cluster.

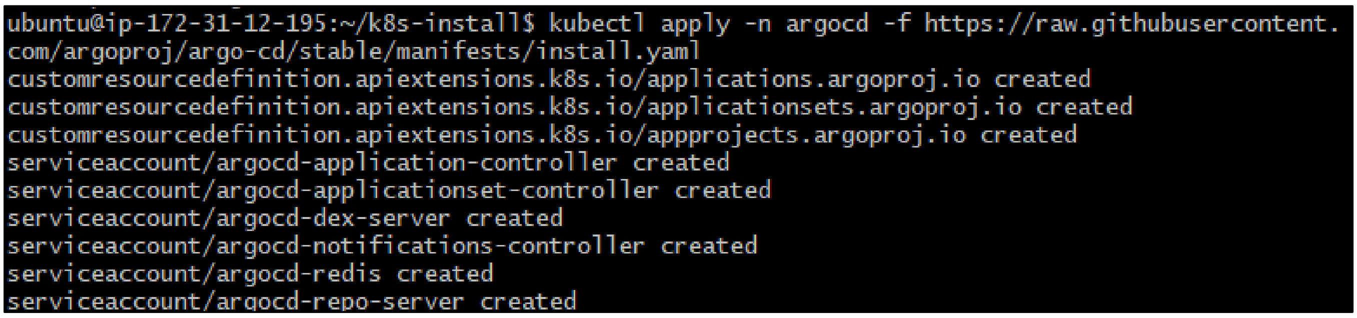
We can install Argo CD by carrying out the following steps:

- **Create a namespace for Argo CD**:
  First, create a dedicated namespace for Argo CD using the command `kubectl create namespace argocd`.



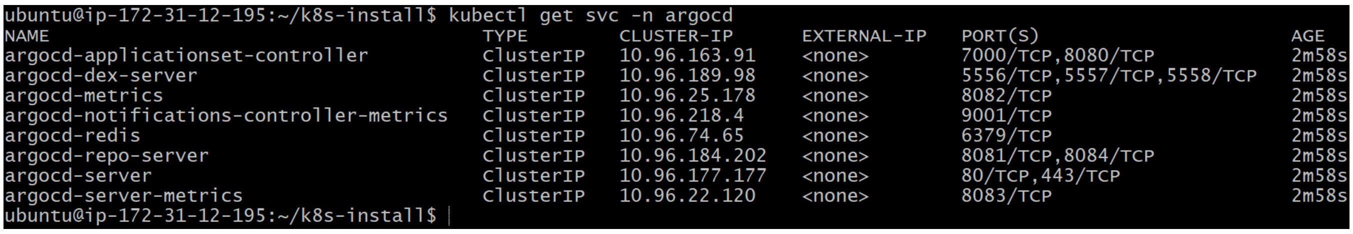
```
ubuntu@ip-172-31-12-195:~/k8s-install$ kubectl create namespace argocd
namespace/argocd created
ubuntu@ip-172-31-12-195:~/k8s-install$ kubectl get namespaces
NAME                 STATUS   AGE
argocd               Active   7s
default              Active   42m
kube-node-lease      Active   42m
kube-public          Active   42m
kube-system          Active   42m
local-path-storage   Active   41m
ubuntu@ip-172-31-12-195:~/k8s-install$
```

- **Apply the Argo CD manifest**:
  Deploy Argo CD by applying the official Argo CD manifest with the command `kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml`.



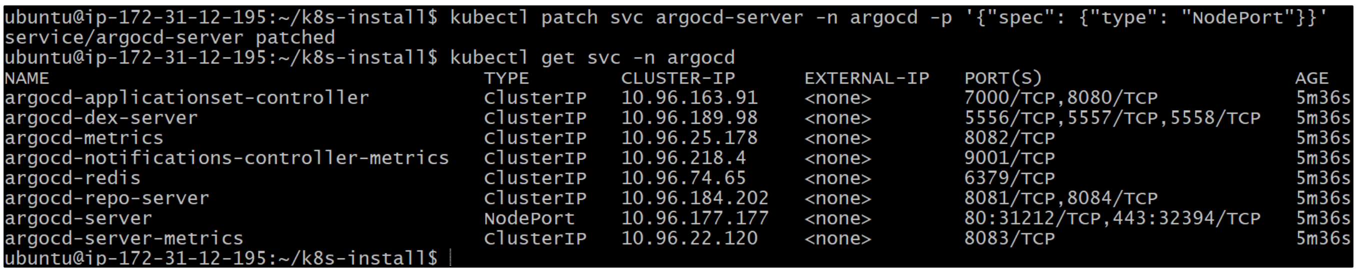
```
ubuntu@ip-172-31-12-195:~/k8s-install$ kubectl apply -n argocd -f https://raw.githubusercontent.
com/argoproj/argo-cd/stable/manifests/install.yaml
customresourcedefinition.apiextensions.k8s.io/applications.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/applicationsets.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/appprojects.argoproj.io created
serviceaccount/argocd-application-controller created
serviceaccount/argocd-applicationset-controller created
serviceaccount/argocd-dex-server created
serviceaccount/argocd-notifications-controller created
serviceaccount/argocd-redis created
serviceaccount/argocd-repo-server created
```

- **Check services in the Argo CD namespace**:
  To verify that the services are running in the `argocd` namespace, execute the command `kubectl get svc -n argocd`.



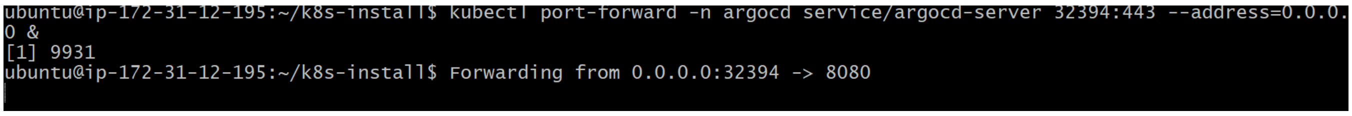
```
ubuntu@ip-172-31-12-195:~/k8s-install$ kubectl get svc -n argocd
NAME                                      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
argocd-applicationset-controller          ClusterIP   10.96.163.91    <none>        7000/TCP,8080/TCP            2m58s
argocd-dex-server                         ClusterIP   10.96.189.98    <none>        5556/TCP,5557/TCP,5558/TCP   2m58s
argocd-metrics                            ClusterIP   10.96.25.178    <none>        8082/TCP                     2m58s
argocd-notifications-controller-metrics   ClusterIP   10.96.218.4     <none>        9001/TCP                     2m58s
argocd-redis                              ClusterIP   10.96.74.65     <none>        6379/TCP                     2m58s
argocd-repo-server                        ClusterIP   10.96.184.202   <none>        8081/TCP,8084/TCP            2m58s
argocd-server                             ClusterIP   10.96.177.177   <none>        80/TCP,443/TCP               2m58s
argocd-server-metrics                     ClusterIP   10.96.22.120    <none>        8083/TCP                     2m58s
ubuntu@ip-172-31-12-195:~/k8s-install$
```

- **Expose the Argo CD server using NodePort**:
  Expose the Argo CD server service using NodePort to make it accessible from outside the cluster with the command `kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "NodePort"}}'`



```
ubuntu@ip-172-31-12-195:~/k8s-install$ kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "NodePort"}}'
service/argocd-server patched
ubuntu@ip-172-31-12-195:~/k8s-install$ kubectl get svc -n argocd
NAME                                      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
argocd-applicationset-controller          ClusterIP   10.96.163.91    <none>        7000/TCP,8080/TCP            5m36s
argocd-dex-server                         ClusterIP   10.96.189.98    <none>        5556/TCP,5557/TCP,5558/TCP   5m36s
argocd-metrics                            ClusterIP   10.96.25.178    <none>        8082/TCP                     5m36s
argocd-notifications-controller-metrics   ClusterIP   10.96.218.4     <none>        9001/TCP                     5m36s
argocd-redis                              ClusterIP   10.96.74.65     <none>        6379/TCP                     5m36s
argocd-repo-server                        ClusterIP   10.96.184.202   <none>        8081/TCP,8084/TCP            5m36s
argocd-server                             NodePort    10.96.177.177   <none>        80:31212/TCP,443:32394/TCP   5m36s
argocd-server-metrics                     ClusterIP   10.96.22.120    <none>        8083/TCP                     5m36s
ubuntu@ip-172-31-12-195:~/k8s-install$
```

- **Forward ports to access the Argo CD server**:
  Finally, forward the necessary ports to access the Argo CD web interface by running `kubectl port-forward -n argocd service/argocd-server 32394:443 --address=0.0.0.0 &`

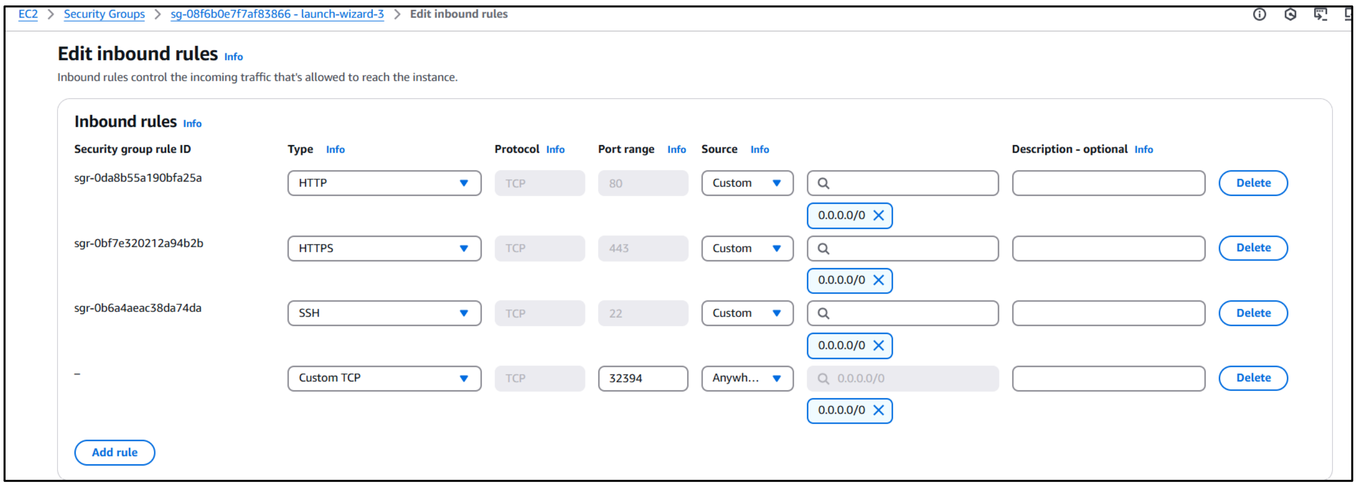


```
ubuntu@ip-172-31-12-195:~/k8s-install$ kubectl port-forward -n argocd service/argocd-server 32394:443 --address=0.0.0.
0 &
[1] 9931
ubuntu@ip-172-31-12-195:~/k8s-install$ Forwarding from 0.0.0.0:32394 -> 8080
```

By following these steps, you will have Argo CD set up and accessible through port `32394` on your EC2 instance's public IP.

- **I**f the connection hangs, you can stop any active port forwarding sessions and restart the process.
  First, stop any existing port forwarding sessions by running the command:
  `pkill -f "kubectl port-forward"`
  Then, restart the port forwarding by executing the command:
  `kubectl port-forward -n argocd service/argocd-server 32394:443 --address=0.0.0.0 &`
  This will help to re-establish the connection and ensure that the Argo CD UI is accessible.

- If you are not able to access Argo CD, make sure that the required port (32394) is enabled in the security group of your EC2 instance. You can do this by modifying the inbound rules of the security group to allow traffic on port 8443 from your IP or from anywhere, depending on your security requirements. This ensures that external requests can reach the Argo CD server running on your instance.



EC2 > Security Groups > sg-08f6b0e7f7af83866 - launch-wizard-3 > Edit inbound rules

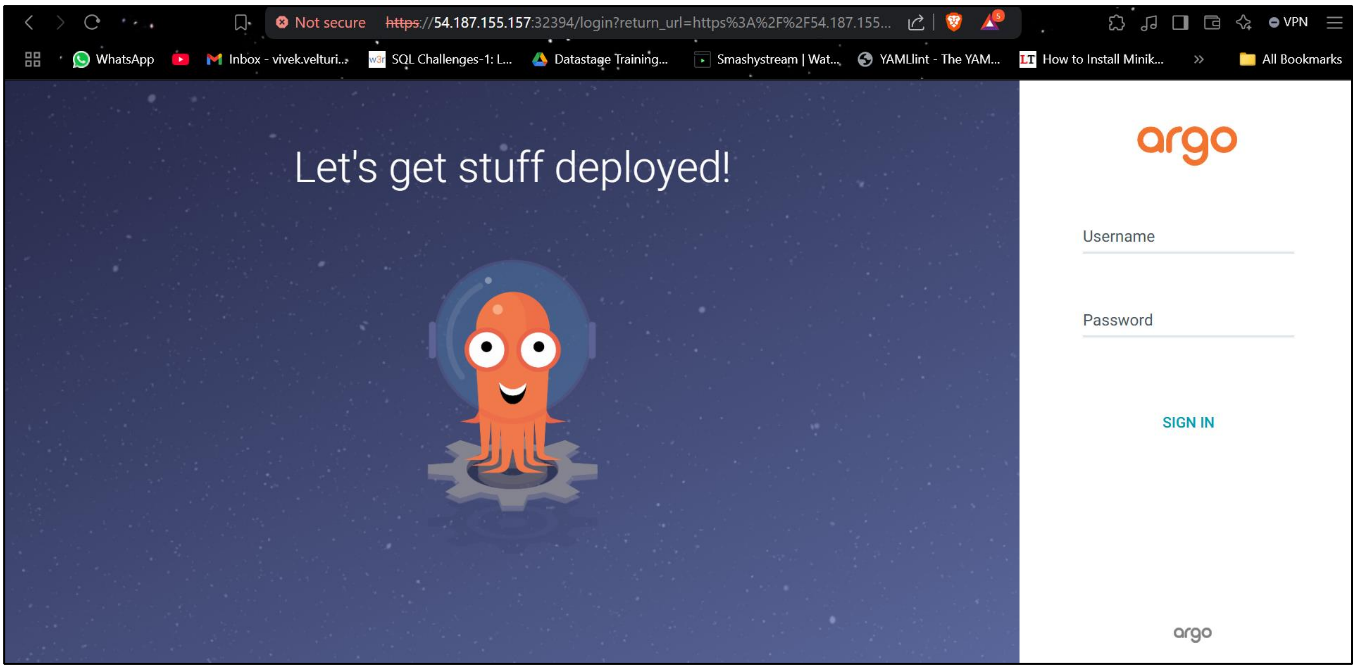
Edit inbound rules Info

Inbound rules control the incoming traffic that's allowed to reach the instance.

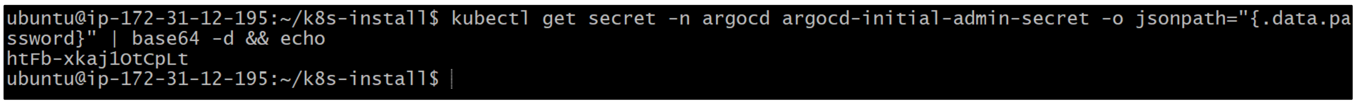
Inbound rules Info

Security group rule ID	Type	Protocol	Port range	Source		Description - optional	
sgr-0da8b55a190bfa25a	HTTP	TCP	80	Custom	0.0.0.0/0		Delete
sgr-0bf7e320212a94b2b	HTTPS	TCP	443	Custom	0.0.0.0/0		Delete
sgr-0b6a4aeac38da74da	SSH	TCP	22	Custom	0.0.0.0/0		Delete
–	Custom TCP	TCP	32394	Anywh...	0.0.0.0/0		Delete

Add rule

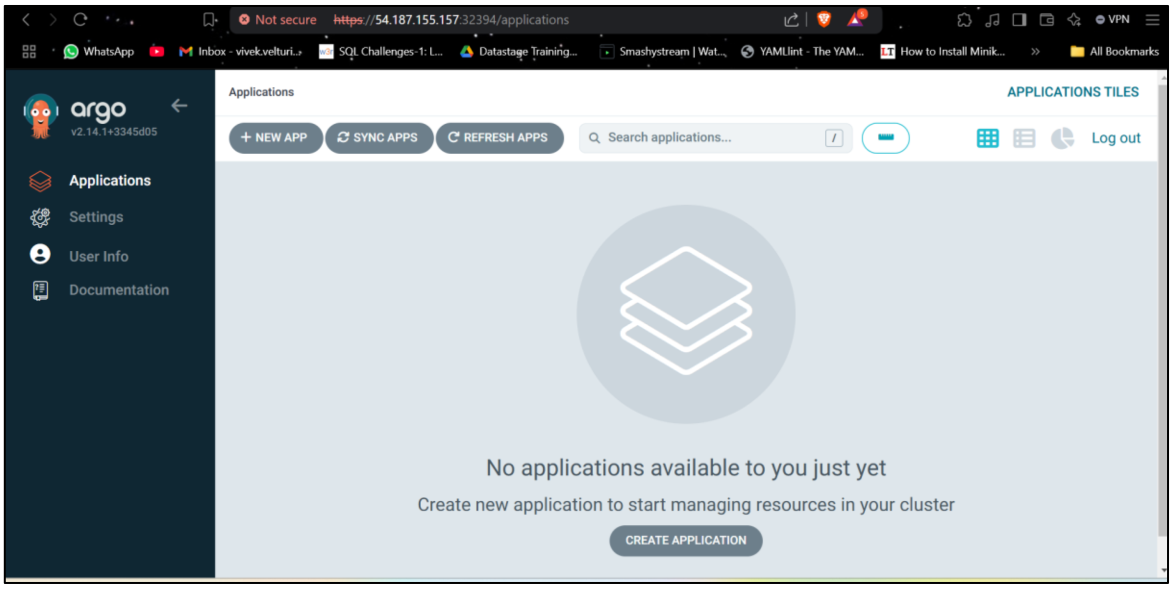

Not secure https://54.187.155.157:32394/login?return_url=https%3A%2F%2F54.187.155...
Let's get stuff deployed!
argo
Username
Password
SIGN IN
argo

- To log in to the Argo CD web interface, the default **username** is **`admin`**, and you'll need to retrieve the **initial admin password** generated during the Argo CD installation. This password is stored as a Kubernetes secret in the `argocd` namespace.
  To obtain it, run the following command:
  `kubectl get secret -n argocd argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d && echo`



```
ubuntu@ip-172-31-12-195:~/k8s-install$ kubectl get secret -n argocd argocd-initial-admin-secret -o jsonpath="{.data.pa
ssword}" | base64 -d && echo
htFb-xkaj1otCpLt
ubuntu@ip-172-31-12-195:~/k8s-install$ 
```

This command fetches the `argocd-initial-admin-secret`, extracts the encoded password, decodes it from Base64, and displays it in the terminal. Once you have the password, you can log in to the Argo CD UI using `admin` as the username and the retrieved password for secure access.



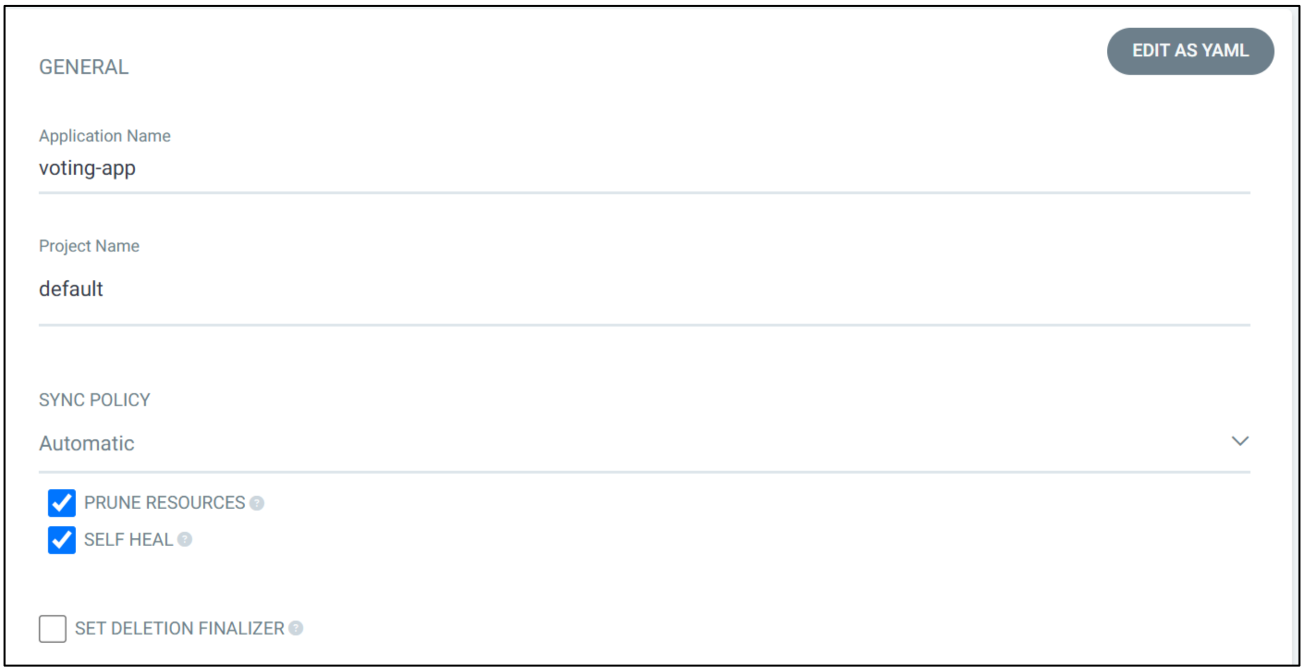
No applications available to you just yet

Create new application to start managing resources in your cluster

Connect and manage your Kubernetes cluster with Argo CD.

- Inside the Argo CD web interface, navigate to the **"Settings"** section to verify the cluster configuration. Ensure that the cluster is set to **default**, as Argo CD has been installed in the default Kubernetes cluster created using Kind. This ensures seamless management and deployment of applications within the correct cluster environment.
- In the Argo CD web interface, navigate to the **"Applications"** section from the sidebar. Click on the **"New App"** button to begin creating a new application. This will open a form where you can configure the application details, such as the application name, project, repository URL, and the target cluster for deployment.
- While creating a new application in Argo CD, fill in the required details as follows:
  - **Application Name:** `voting-app`
  - **Project Name:** `default`
  - **Sync Policy:** Set to **Automatic**
  - **Enable Options:** Check both **"Prune Resources"** and **"Self Heal"** to ensure that Argo CD automatically manages resource cleanup and maintains the desired state.
  - **Source Repository URL:** `https://github.com/VivekVelturi/k8s-kind-voting-app.git`, which contains all the deployment YAML files.
  - **Revision:** Change from the default `HEAD` to the `main` branch, as all configurations are maintained there.
  - **Path:** `k8s-specifications`, which is the folder in the repository where the deployment manifests are located.
  - **Destination Cluster:** Select the cluster defined in the **Settings** section of Argo CD, which corresponds to the Kubernetes cluster where Argo CD is installed.
  - **Namespace:** Set the namespace to **`default`** to ensure that the application resources are deployed within the default Kubernetes namespace.

After completing these configurations, click on **"Create"** to deploy the `voting-app`. Argo CD will handle the synchronization and ensure the application is deployed and managed efficiently within the cluster.



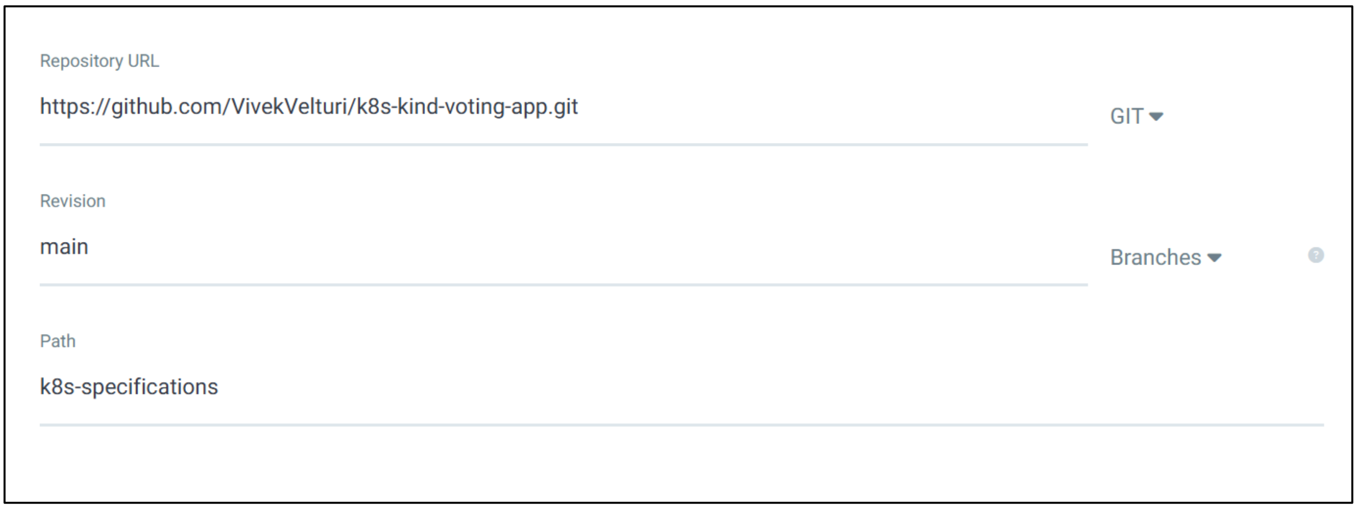
GENERAL

Application Name
voting-app

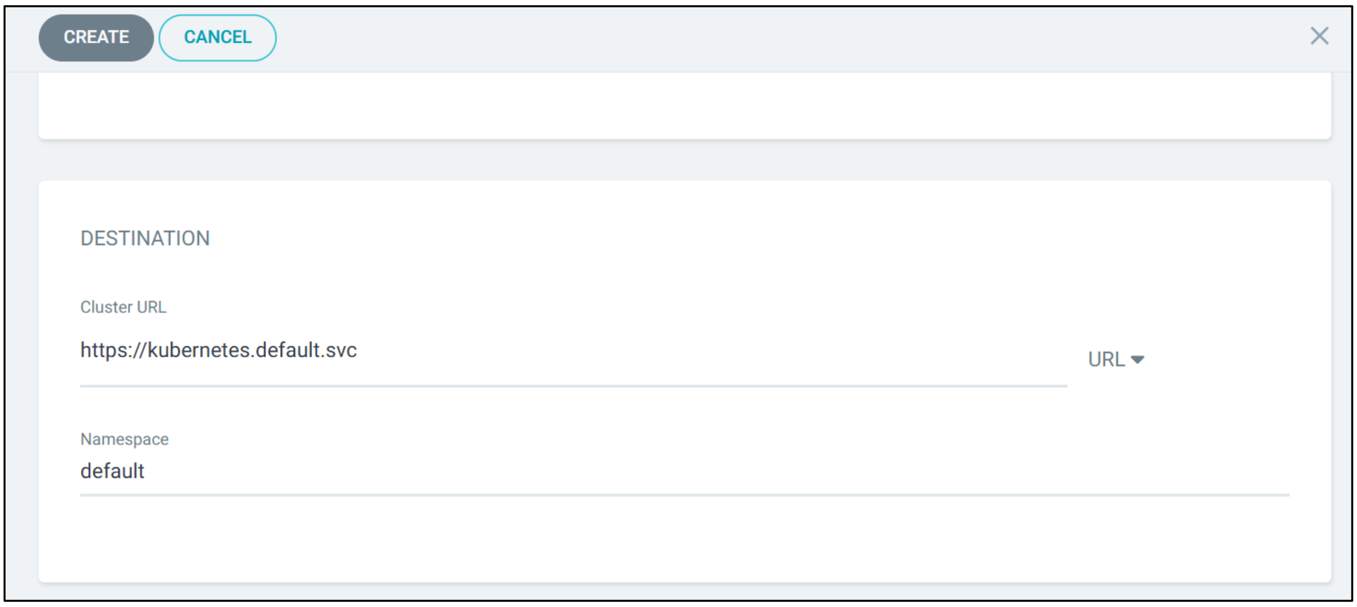
Project Name
default

SYNC POLICY
Automatic

[✓] PRUNE RESOURCES
[✓] SELF HEAL
[ ] SET DELETION FINALIZER

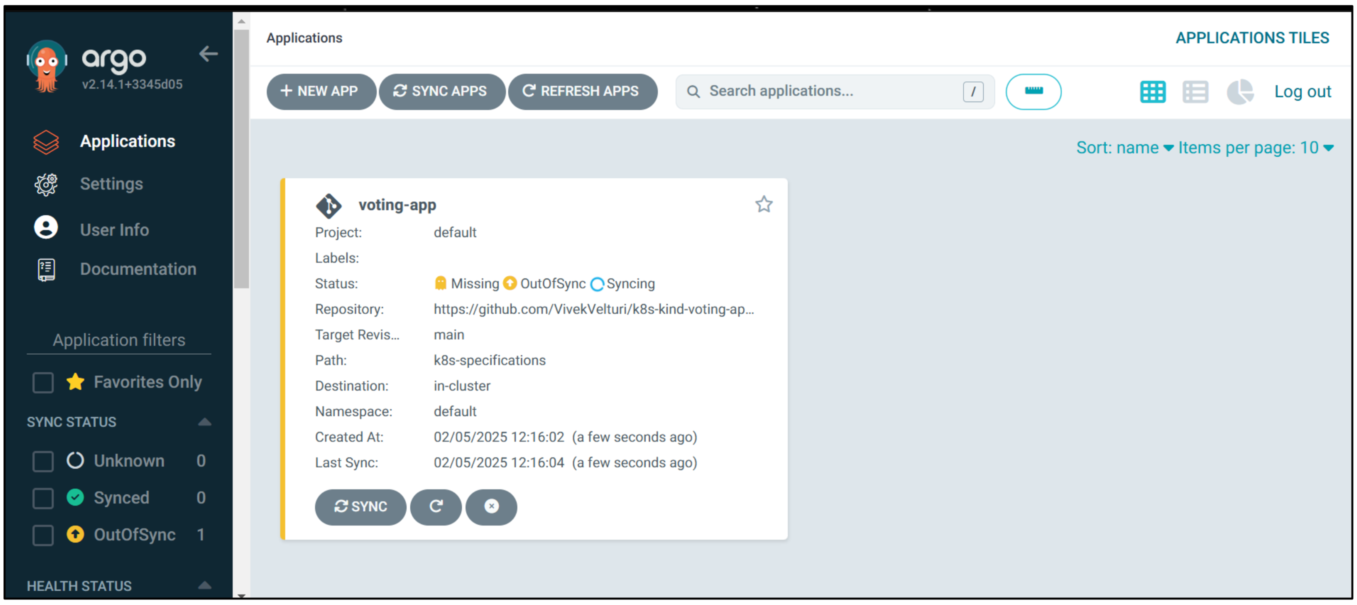


Repository URL
https://github.com/VivekVelturi/k8s-kind-voting-app.git
GIT
Revision
main
Branches
Path
k8s-specifications



CREATE CANCEL

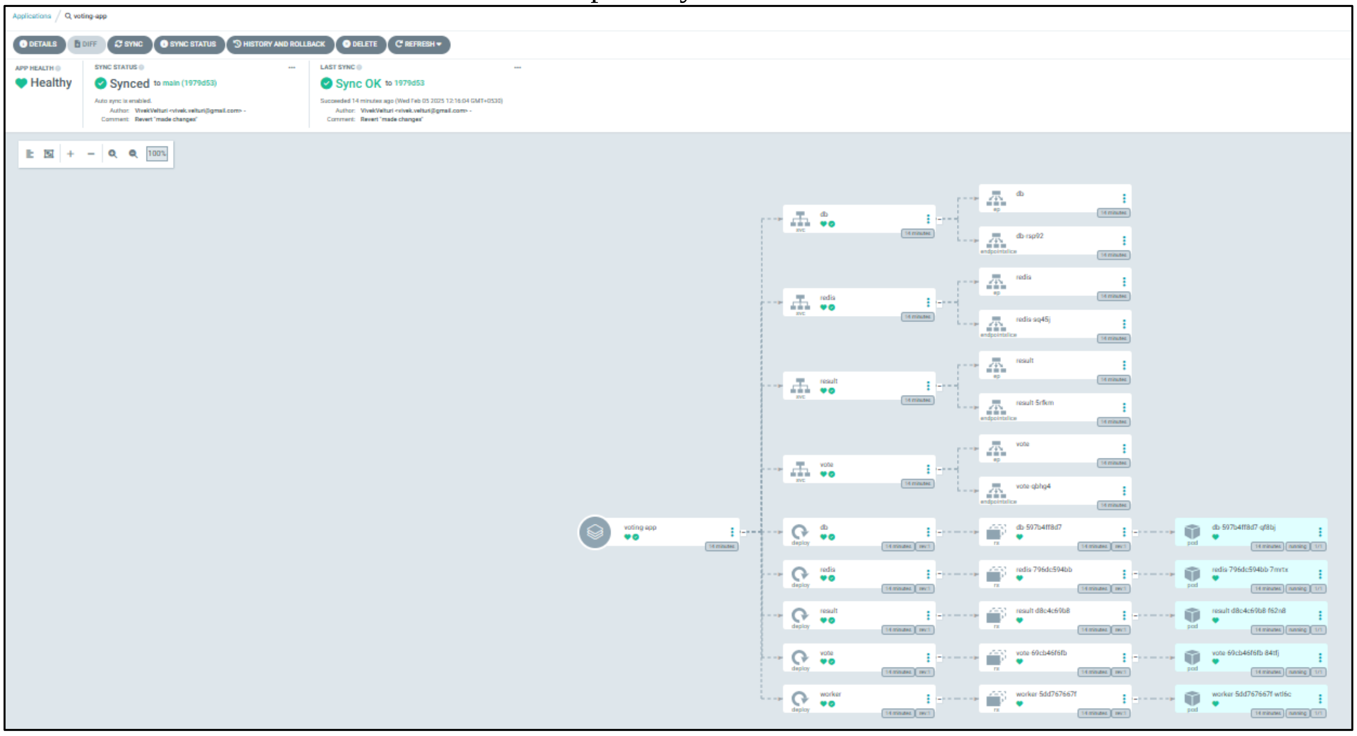
DESTINATION
Cluster URL
https://kubernetes.default.svc
URL
Namespace
default



voting-app

Project:	default
Labels:	
Status:	Missing OutOfSync Syncing
Repository:	https://github.com/VivekVelturi/k8s-kind-voting-ap...
Target Revis...	main
Path:	k8s-specifications
Destination:	in-cluster
Namespace:	default
Created At:	02/05/2025 12:16:02 (a few seconds ago)
Last Sync:	02/05/2025 12:16:04 (a few seconds ago)

- Click on the **voting-app** application in the Argo CD UI, and you will see that it is periodically syncing from the repository. It will display the status of the resources and create the pods as specified in the `k8s-specifications` folder of the repository. This process ensures that the desired state of the application is maintained according to the Kubernetes manifests defined in the repository.



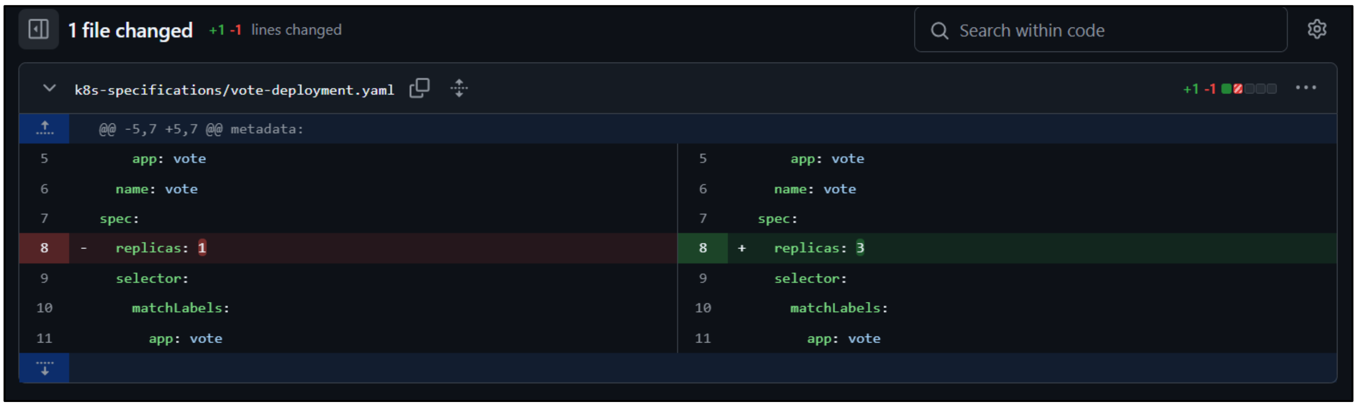
Applications / voting-app

APP HEALTH: Healthy

SYNC STATUS: Synced to main (1979d53)
Auto sync is enabled.
Author: VivekVelturi <vivek.velturi@gmail.com> -
Comment: Revert "made changes"

LAST SYNC: Sync OK to 1979d53
Succeeded 14 minutes ago (Wed Feb 05 2025 12:16:04 GMT+0530)
Author: VivekVelturi <vivek.velturi@gmail.com> -
Comment: Revert "made changes"

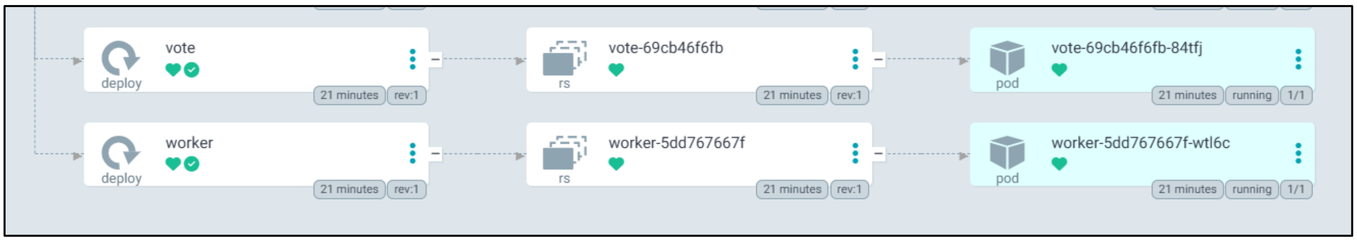
- To observe the sync and changes in Argo CD, we will edit the replica count in the `vote.deployment` YAML file located inside the `k8s-specifications` folder in our GitHub repository. Change the number of replicas from `1` to `3` in the `spec.replicas` section of the file. After saving and committing the changes, go to the Argo CD UI, select the **voting-app** application, and click on the **Sync** button. This will trigger Argo CD to pull the latest changes from the repository, and you will observe the updated replica count, which will scale from 1 to 3, being reflected in the application.



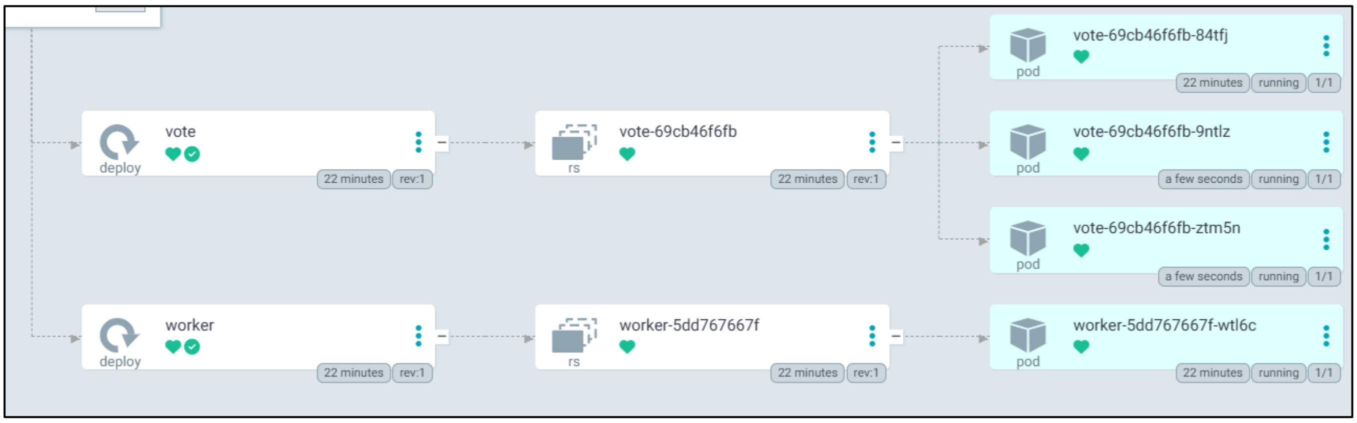
1 file changed +1 -1 lines changed

k8s-specifications/vote-deployment.yaml +1 -1

```
@@ -5,7 +5,7 @@ metadata:
      app: vote
    name: vote
  spec:
-   replicas: 1
+   replicas: 3
    selector:
      matchLabels:
        app: vote
```

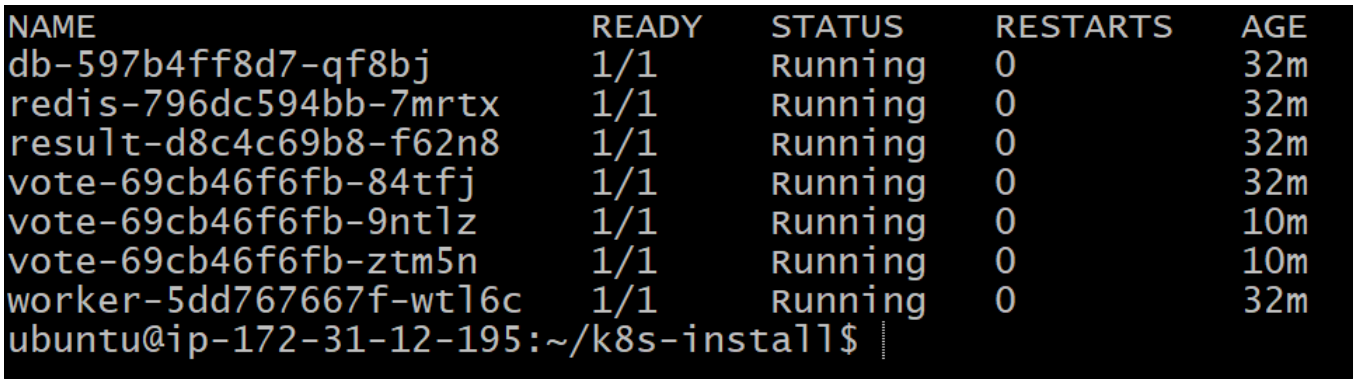


***Before Editing**: Screenshot showing the original YAML file with the replica count set to 1*



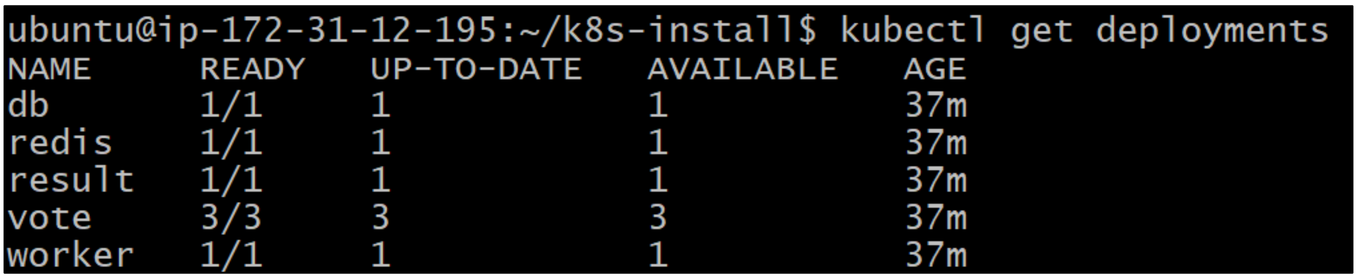
***After Editing**: Screenshot showing the updated YAML file with the replica count changed to 3.*

- Now, if we execute the command `kubectl get pods` in our instance, we will see the pods created as per the specifications defined in the YAML files located in the **`k8s-specifications`** folder in our GitHub repository. These pods will be deployed according to the configuration changes, such as the updated replica count and other settings, that were synced through Argo CD.



```
NAME                        READY   STATUS    RESTARTS   AGE
db-597b4ff8d7-qf8bj         1/1     Running   0          32m
redis-796dc594bb-7mrtx      1/1     Running   0          32m
result-d8c4c69b8-f62n8      1/1     Running   0          32m
vote-69cb46f6fb-84tfj       1/1     Running   0          32m
vote-69cb46f6fb-9ntlz       1/1     Running   0          10m
vote-69cb46f6fb-ztm5n       1/1     Running   0          10m
worker-5dd767667f-wtl6c     1/1     Running   0          32m
ubuntu@ip-172-31-12-195:~/k8s-install$
```

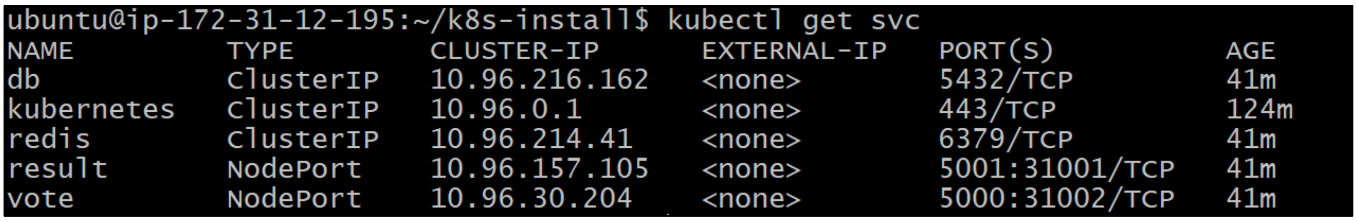
- Now, if you execute the command `kubectl get deployments` in our instance, This command will show the updated state of the deployment based on the latest changes applied in the repository and synced via Argo CD.



```
ubuntu@ip-172-31-12-195:~/k8s-install$ kubectl get deployments
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
db       1/1     1            1           37m
redis    1/1     1            1           37m
result   1/1     1            1           37m
vote     3/3     3            3           37m
worker   1/1     1            1           37m
```

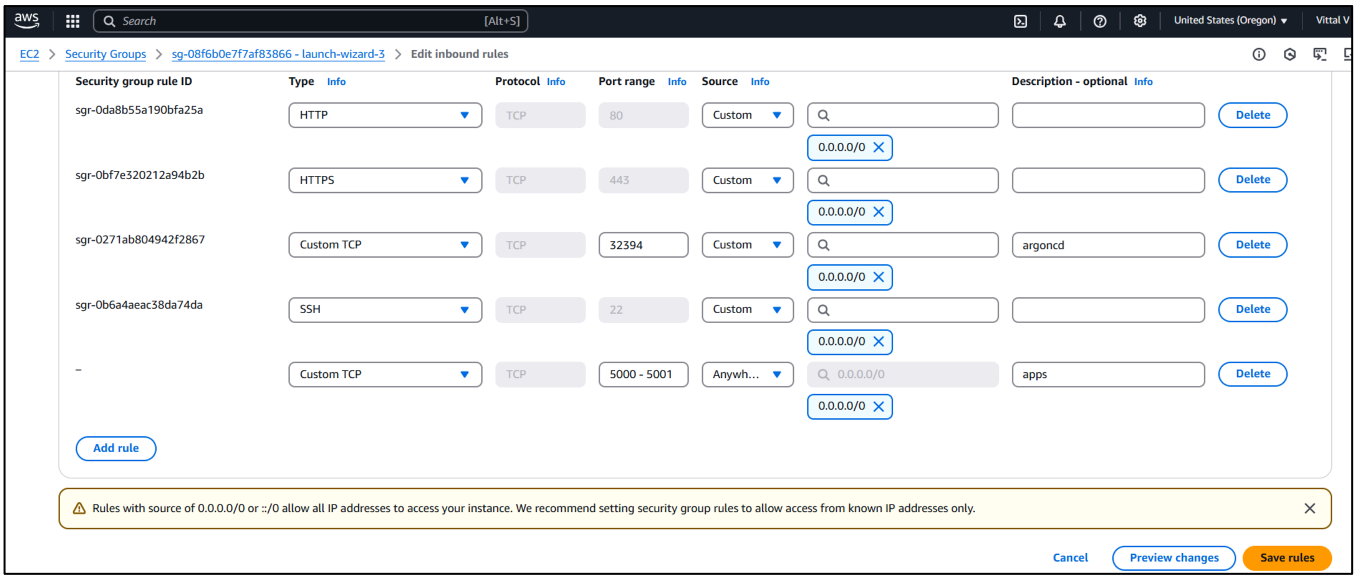
- In order to access the results and votes of our app in the browser, we need to find the ports on which the services are exposed. You can retrieve this information by executing the command `kubectl get svc`. This will display the services running in the Kubernetes cluster, along with their associated ports.

  Look for the **NodePort** type service (if configured) and note down the port number under the **PORT(S)** column. This port will allow you to access the application externally from the browser by using the EC2 instance's public IP address along with the specified port number.



```
ubuntu@ip-172-31-12-195:~/k8s-install$ kubectl get svc
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
db            ClusterIP   10.96.216.162   <none>        5432/TCP         41m
kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP          124m
redis         ClusterIP   10.96.214.41    <none>        6379/TCP         41m
result        NodePort    10.96.157.105   <none>        5001:31001/TCP   41m
vote          NodePort    10.96.30.204    <none>        5000:31002/TCP   41m
```

- Once you have identified the ports using the `kubectl get svc` command, make sure to open those ports in the security group of your EC2 instance to allow inbound traffic. You can do this by updating the inbound rules in your instance's security group through the AWS Management Console. This will ensure that your app is accessible from your browser by allowing traffic on the required ports.

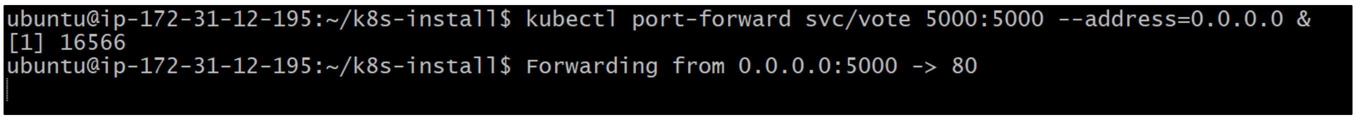


EC2 > Security Groups > sg-08f6b0e7f7af83866 - launch-wizard-3 > Edit inbound rules

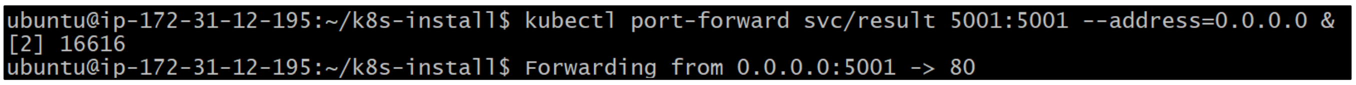
Security group rule ID	Type	Protocol	Port range	Source		Description - optional
sgr-0da8b55a190bfa25a	HTTP	TCP	80	Custom	0.0.0.0/0	
sgr-0bf7e320212a94b2b	HTTPS	TCP	443	Custom	0.0.0.0/0	
sgr-0271ab804942f2867	Custom TCP	TCP	32394	Custom	0.0.0.0/0	argoncd
sgr-0b6a4aeac38da74da	SSH	TCP	22	Custom	0.0.0.0/0	
–	Custom TCP	TCP	5000 - 5001	Anywh...	0.0.0.0/0	apps

Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

- To access the services running in your Kubernetes cluster, you need to map the appropriate ports.
  You can do this by executing the following commands.
  For the voting service, use
  `kubectl port-forward svc/vote 5000:5000 --address=0.0.0.0 &`
  and for the result service, use
  `kubectl port-forward svc/result 5001:5001 --address=0.0.0.0 &`
  These commands will forward the ports `5000` and `5001` from the services in the cluster to your local machine. The `--address=0.0.0.0` flag ensures that the services can be accessed from any address on the local machine.

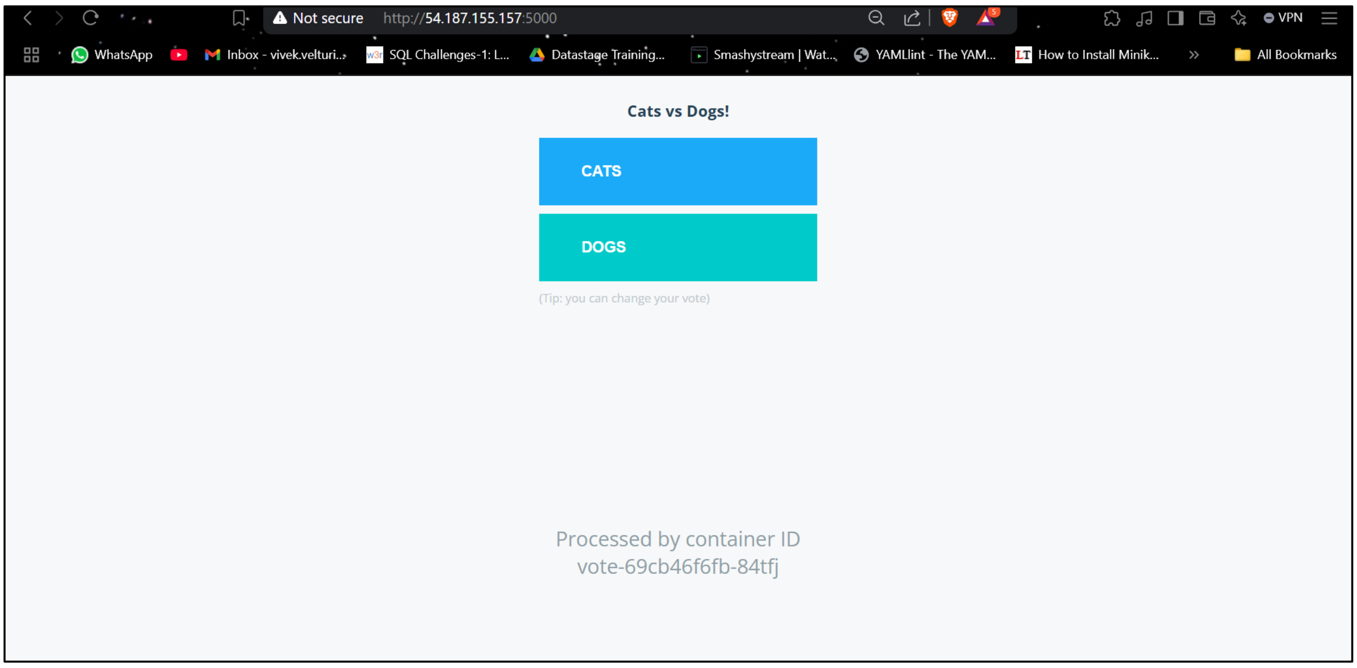


```
ubuntu@ip-172-31-12-195:~/k8s-install$ kubectl port-forward svc/vote 5000:5000 --address=0.0.0.0 &
[1] 16566
ubuntu@ip-172-31-12-195:~/k8s-install$ Forwarding from 0.0.0.0:5000 -> 80
```

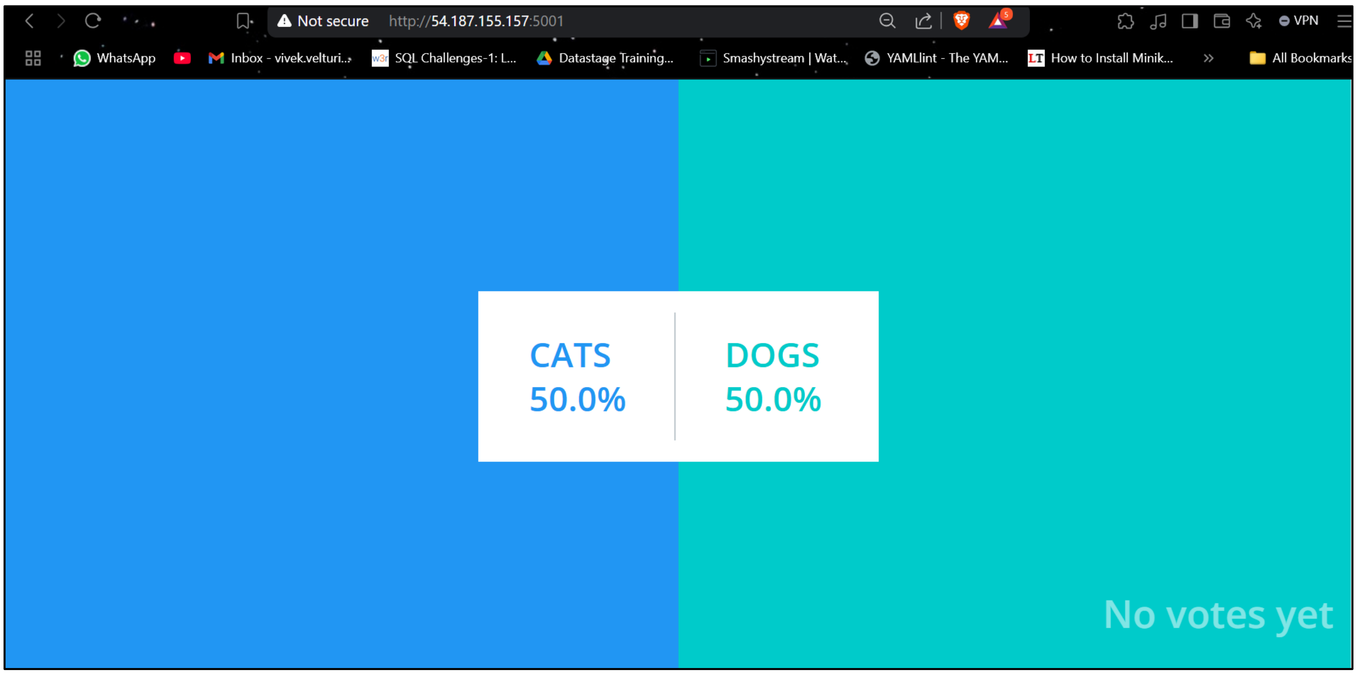


```
ubuntu@ip-172-31-12-195:~/k8s-install$ kubectl port-forward svc/result 5001:5001 --address=0.0.0.0 &
[2] 16616
ubuntu@ip-172-31-12-195:~/k8s-install$ Forwarding from 0.0.0.0:5001 -> 80
```

- Now that we have mapped the ports and configured the necessary inbound rules in the security group, you can access the services by opening the public IP of the EC2 instance along with the respective ports for the vote and result services. By navigating to the public IP with the ports `5000` for the voting service and `5001` for the result service in the browser, you will be able to see the application in action.

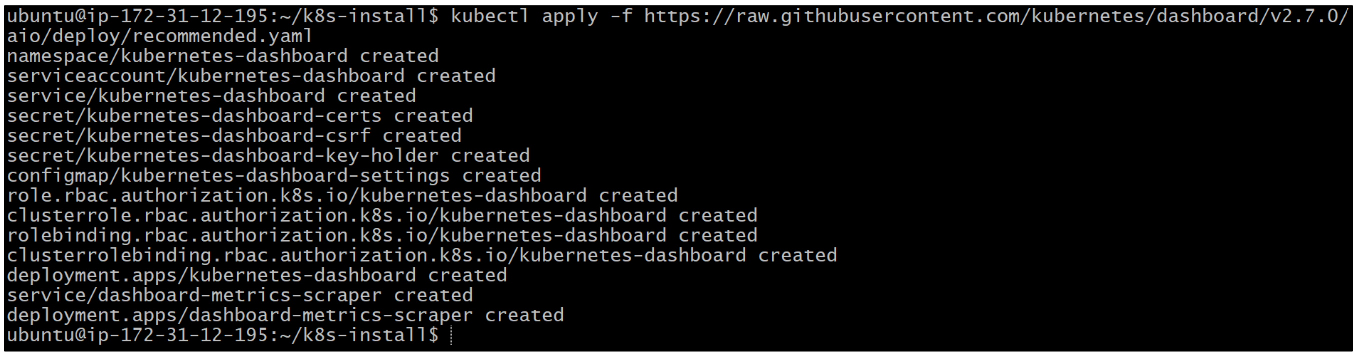


Processed by container ID
vote-69cb46f6fb-84tfj



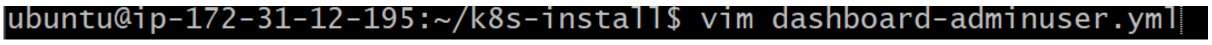
Set up the Kubernetes Dashboard.

- To install the Kubernetes Dashboard, you can deploy it by running the following command:
  kubectl apply -f
  https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/recommended.yaml
  This command will apply the recommended Kubernetes dashboard manifest, which will deploy the necessary resources to run the Kubernetes Dashboard within your cluster. Once the resources are deployed, you will be able to access the dashboard to manage and monitor your Kubernetes cluster.



```
ubuntu@ip-172-31-12-195:~/k8s-install$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/
aio/deploy/recommended.yaml
namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf created
secret/kubernetes-dashboard-key-holder created
configmap/kubernetes-dashboard-settings created
role.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
deployment.apps/kubernetes-dashboard created
service/dashboard-metrics-scraper created
deployment.apps/dashboard-metrics-scraper created
ubuntu@ip-172-31-12-195:~/k8s-install$
```

- To set up the Kubernetes Dashboard,
  - create the **dashboard-adminuser.yml** file on your EC2 instance by running the command `vim dashboard.yml`.

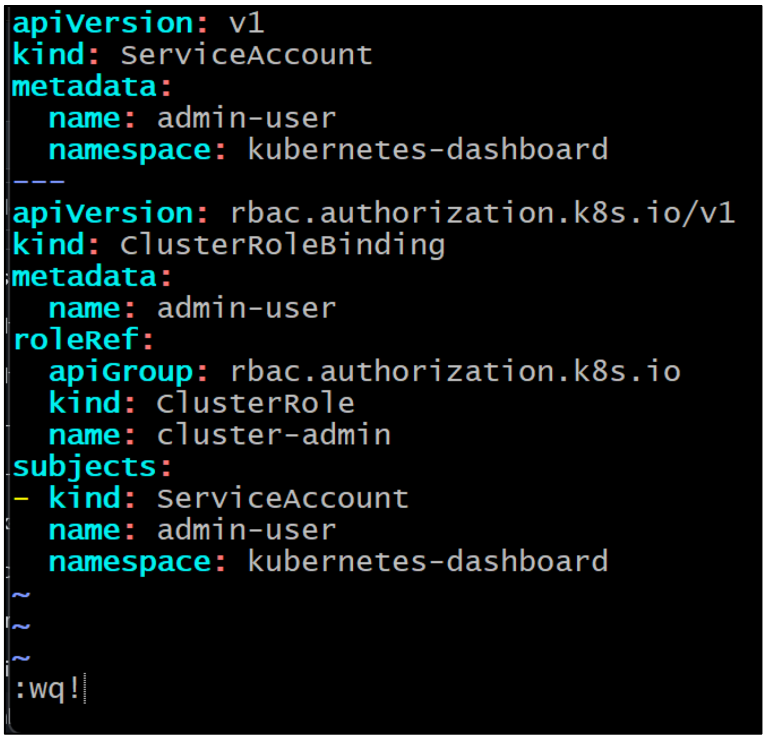


```
ubuntu@ip-172-31-12-195:~/k8s-install$ vim dashboard-adminuser.yml
```

  - Then, paste the Kubernetes Dashboard configuration script from the GitHub repository into the file.
    The Script is as follows:

```
apiVersion: v1  # Defines the API version used for the object. Here, it's v1, which is standard for Kubernetes objects like ServiceAccount and ClusterRoleBinding.
kind: ServiceAccount  # Specifies that the resource being created is a ServiceAccount. A ServiceAccount provides an identity for processes that run in a Pod.
metadata:  # Contains metadata about the object, such as name, namespace, and labels.
  name: admin-user  # The name of the ServiceAccount being created, which will be used for authentication purposes.
  namespace: kubernetes-dashboard  # The namespace where the ServiceAccount will be created. In this case, it's the 'kubernetes-dashboard' namespace, which is the default namespace for Kubernetes Dashboard.
---
apiVersion: rbac.authorization.k8s.io/v1  # Specifies the API version for RBAC resources, which control access and permissions in Kubernetes.
```

```
kind: ClusterRoleBinding  # Specifies that the resource being created is a ClusterRoleBinding, which binds a ClusterRole to a user or set of users (ServiceAccount, in this case).
metadata:  # Contains metadata for the ClusterRoleBinding object.
  name: admin-user  # The name of the ClusterRoleBinding, which links the 'admin-user' ServiceAccount to the 'cluster-admin' ClusterRole.
roleRef:  # Defines the ClusterRole that is being granted to the ServiceAccount.
  apiGroup: rbac.authorization.k8s.io  # The API group for RBAC resources, indicating that this ClusterRoleBinding will reference an RBAC ClusterRole.
  kind: ClusterRole  # Specifies that the role being referenced is a ClusterRole (as opposed to a Role, which is scoped to a specific namespace).
  name: cluster-admin  # The name of the ClusterRole being referenced. 'cluster-admin' grants superuser privileges at the cluster level.
subjects:  # Specifies the user(s) (or ServiceAccount(s)) who will be granted the permissions defined in the ClusterRole.
- kind: ServiceAccount  # Specifies that the subject of the ClusterRoleBinding is a ServiceAccount.
  name: admin-user  # The name of the ServiceAccount to which the permissions will be granted, matching the name defined in the first part of the YAML.
  namespace: kubernetes-dashboard  # The namespace in which the ServiceAccount exists, aligning with the namespace defined earlier.
```

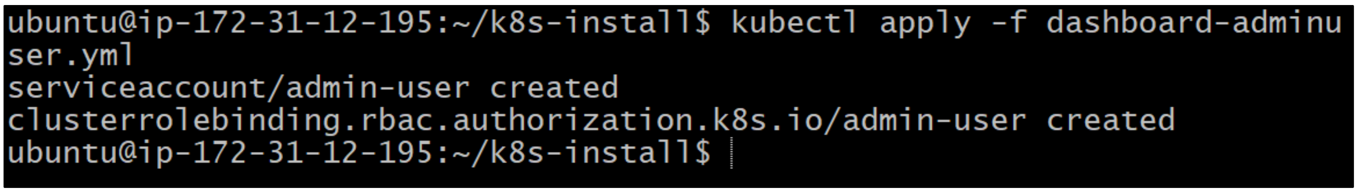


```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kubernetes-dashboard
~
~
~
:wq!
```

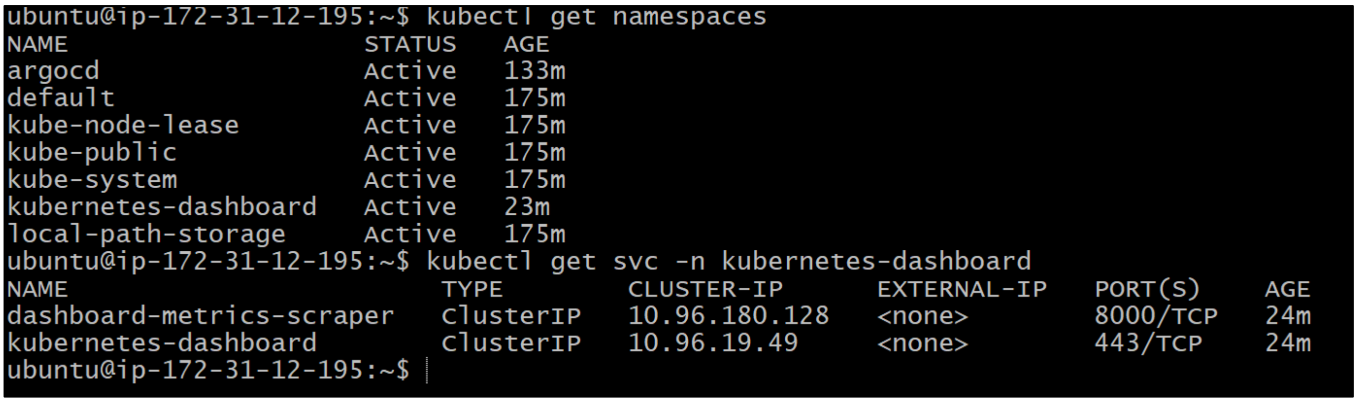
- Afterward, save and exit Vim to complete the setup.
- With the **dashboard.yml** file in place, you can deploy the Kubernetes Dashboard by executing the following command:

```
kubectl apply -f dashboard-adminuser.yml
```

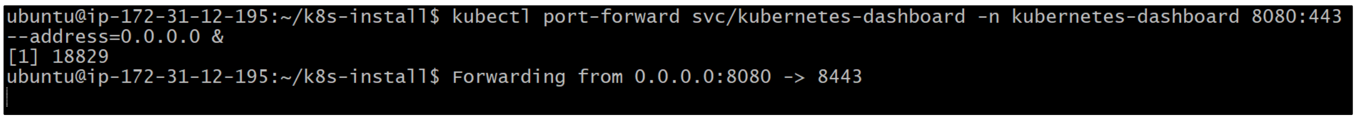
This command will deploy the Kubernetes Dashboard as a set of resources in your cluster, allowing you to manage and monitor your Kubernetes cluster via a web-based UI.


```
ubuntu@ip-172-31-12-195:~/k8s-install$ kubectl apply -f dashboard-adminu
ser.yml
serviceaccount/admin-user created
clusterrolebinding.rbac.authorization.k8s.io/admin-user created
ubuntu@ip-172-31-12-195:~/k8s-install$
```

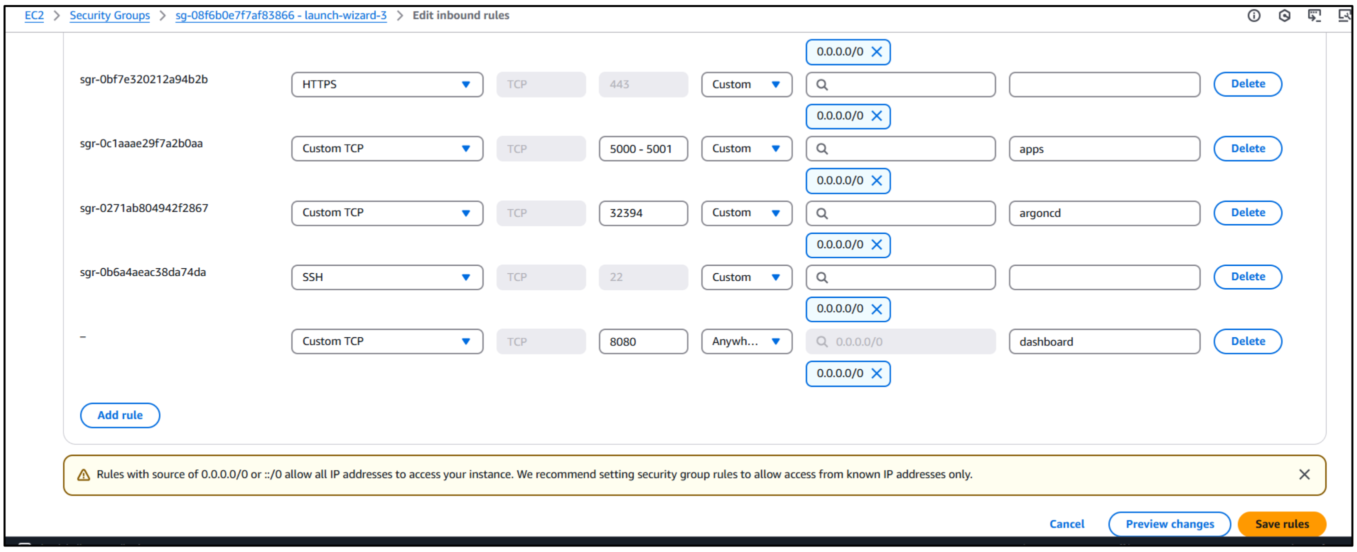
- To find the port where the Kubernetes Dashboard can be accessed, first run the command:
  `kubectl get svc -n kubernetes-dashboard` .This will list the services running in the `kubernetes-dashboard` namespace, including the `kubernetes-dashboard` service. Look for the `port` or `NodePort` associated with the service, which is the port you can use to access the dashboard from your browser.


```
ubuntu@ip-172-31-12-195:~$ kubectl get namespaces
NAME                   STATUS   AGE
argocd                 Active   133m
default                Active   175m
kube-node-lease        Active   175m
kube-public            Active   175m
kube-system            Active   175m
kubernetes-dashboard   Active   23m
local-path-storage     Active   175m
ubuntu@ip-172-31-12-195:~$ kubectl get svc -n kubernetes-dashboard
NAME                        TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
dashboard-metrics-scraper   ClusterIP   10.96.180.128   <none>        8000/TCP   24m
kubernetes-dashboard        ClusterIP   10.96.19.49     <none>        443/TCP    24m
ubuntu@ip-172-31-12-195:~$
```

- To access the Kubernetes Dashboard, you first need to forward the necessary port by running the command:
  `kubectl port-forward svc/kubernetes-dashboard -n kubernetes-dashboard 8080:443 --address=0.0.0.0 &` .This command forwards port 443 of the Kubernetes Dashboard service to port 8080 on your EC2 instance, allowing you to access the dashboard externally

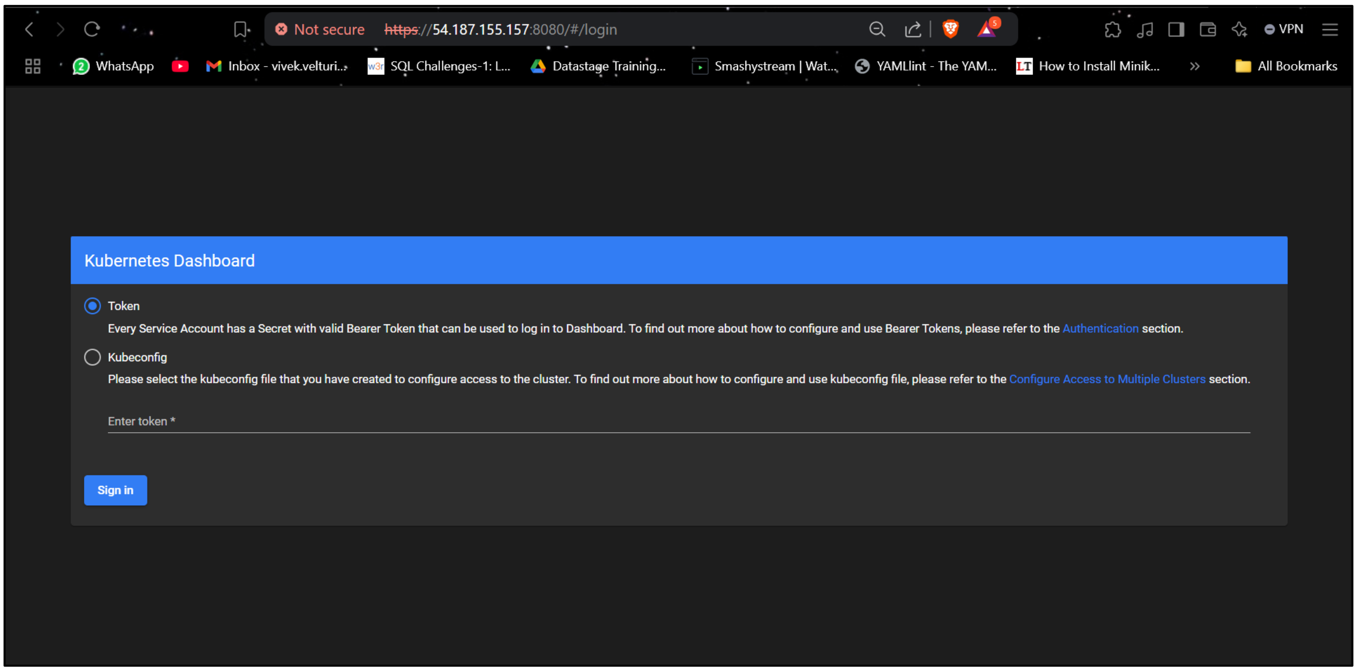

```
ubuntu@ip-172-31-12-195:~/k8s-install$ kubectl port-forward svc/kubernetes-dashboard -n kubernetes-dashboard 8080:443
--address=0.0.0.0 &
[1] 18829
ubuntu@ip-172-31-12-195:~/k8s-install$ Forwarding from 0.0.0.0:8080 -> 8443
```

- Now, to access the Kubernetes Dashboard via your EC2 instance's public IP, you need to ensure that port 8080 is open in the security group associated with your instance. To do this, go to the AWS Management Console, navigate to EC2, and under "Security Groups," select the security group associated with your instance. In the "Inbound rules" section, click "Edit inbound rules," then add a new rule for TCP port 8080. Set the source as "Anywhere" (0.0.0.0/0) if you want it to be publicly accessible. Save the changes, and once the rule is added, you should be able to access the Kubernetes Dashboard via the EC2 instance's public IP at port 8080.


EC2 > Security Groups > sg-08f6b0e7f7af83866 - launch-wizard-3 > Edit inbound rules

sgr-0bf7e320212a94b2b	HTTPS	TCP	443	Custom	0.0.0.0/0		Delete
sgr-0c1aaae29f7a2b0aa	Custom TCP	TCP	5000 - 5001	Custom	0.0.0.0/0	apps	Delete
sgr-0271ab804942f2867	Custom TCP	TCP	32394	Custom	0.0.0.0/0	argoncd	Delete
sgr-0b6a4aeac38da74da	SSH	TCP	22	Custom	0.0.0.0/0		Delete
–	Custom TCP	TCP	8080	Anywh...	0.0.0.0/0	dashboard	Delete

Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

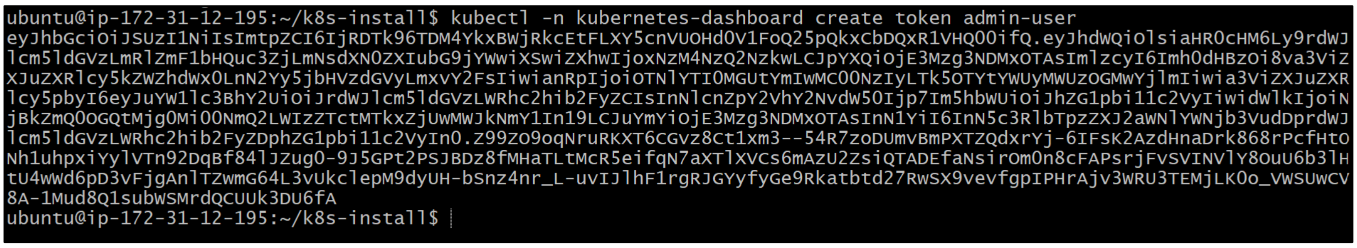

Not secure ~~https~~://54.187.155.157:8080/#/login

Kubernetes Dashboard

- [✓ Token]
  Every Service Account has a Secret with valid Bearer Token that can be used to log in to Dashboard. To find out more about how to configure and use Bearer Tokens, please refer to the Authentication section.
- Kubeconfig
  Please select the kubeconfig file that you have created to configure access to the cluster. To find out more about how to configure and use kubeconfig file, please refer to the Configure Access to Multiple Clusters section.

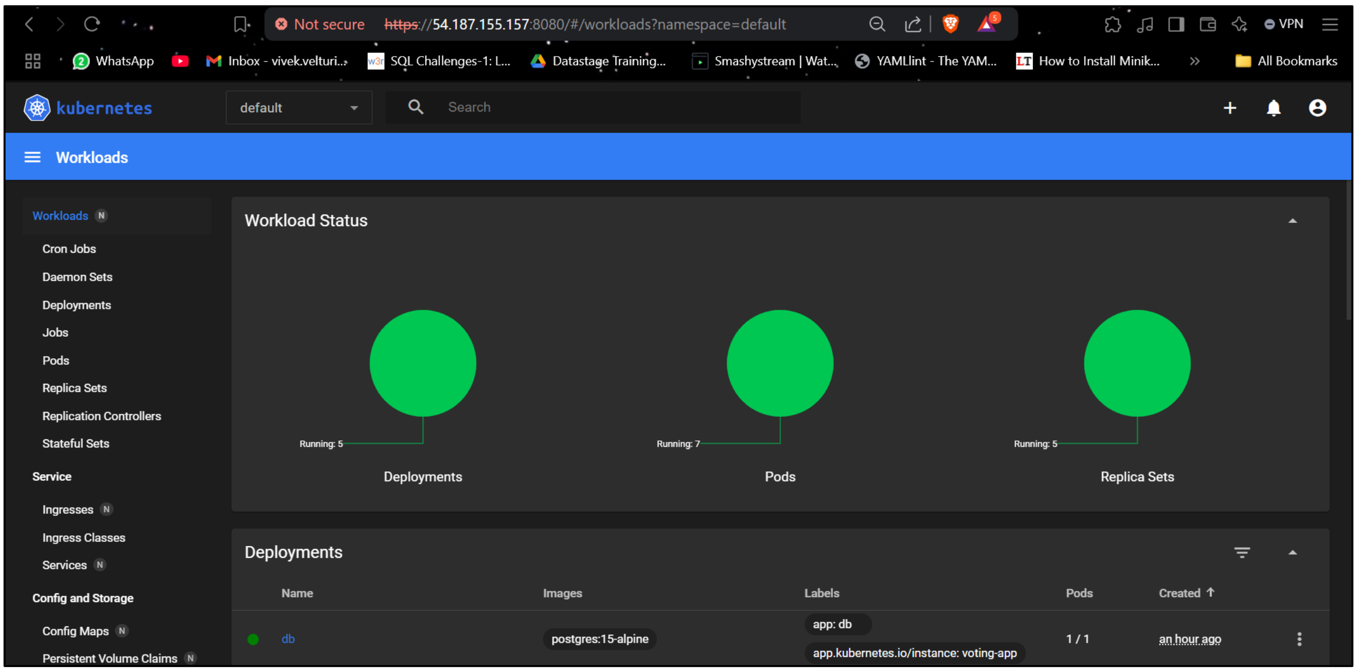
Enter token *

- Now, to create a token for accessing the Kubernetes Dashboard, run the command : `kubectl -n kubernetes-dashboard create token admin-user`.
  This will generate a token for the `admin-user` ServiceAccount within the `kubernetes-dashboard` namespace.
  The token grants administrative access to the Kubernetes Dashboard as specified by the `cluster-admin` role binding.
  Once executed, the generated token will be displayed in the terminal, and you can use it to log in to the Dashboard with full administrative privileges.

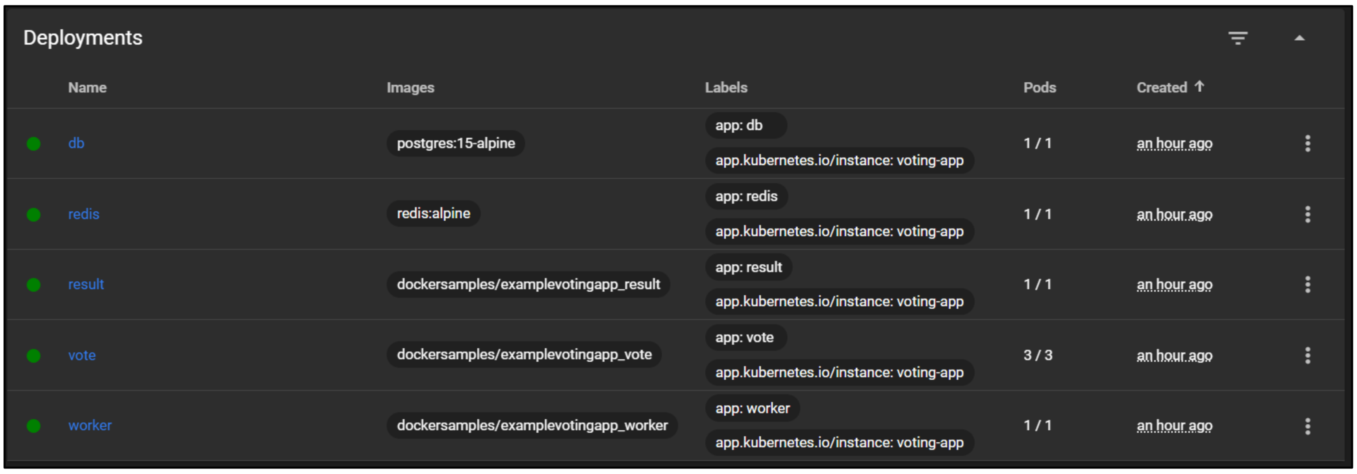

```
ubuntu@ip-172-31-12-195:~/k8s-install$ kubectl -n kubernetes-dashboard create token admin-user
eyJhbGciOiJSUzI1NiIsImtpZCI6IjRDTk96TDM4YkxBWjRkcEtFLXY5cnVUOHd0V1FoQ25pQkxCbDQxR1VHQ00ifQ.eyJhdWQiOlsiaHR0cHM6Ly9rdWJ
lcm5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiXSwiZXhwIjoxNzM4NzQ2NzkwLCJpYXQiOjE3Mzg3NDMxOTAsImlzcyI6Imh0dHBzOi8va3ViZ
XJuZXRlcy5kZWZhdWx0LnN2Yy5jbHVzdGVyLmxvY2FsIiwianRpIjoiOTNlYTI0MGUtYmIwMC00NzIyLTk5OTYtYWUyMWUzOGMwYjlmIiwia3ViZXJuZXR
lcy5pbyI6eyJuYW1lc3BhY2UiOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIsInNlcnZpY2VhY2NvdW50Ijp7Im5hbWUiOiJhZG1pbi11c2VyIiwidWlkIjoiN
jBkZmQ0OGQtMjg0Mi00NmQ2LWIzZTctMTkxZjUwMWJkNmY1In19LCJuYmYiOjE3Mzg3NDMxOTAsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDprdWJ
lcm5ldGVzLWRhc2hib2FyZDphZG1pbi11c2VyIn0.Z99ZO9oqNruRKXT6CGvz8Ct1xm3--54R7zoDUmvBmPXTZQdxrYj-6IFsK2AzdHnaDrk868rPcfHtO
Nh1uhpxiYylVTn92DqBf84lJZug0-9J5GPt2PSJBDz8fMHaTLtMcR5eifqN7aXTlXVCs6mAzU2ZsiQTADEfaNsirOm0n8cFAPsrjFvSVINVlY8OuU6b3lH
tU4wWd6pD3vFjgAnlTZwmG64L3vUkclepM9dyUH-bSnz4nr_L-uvIJlhF1rgRJGYyfyGe9Rkatbtd27RwSX9vevfgpIPHrAjv3WRU3TEMjLKOo_VWSUwCV
8A-1Mud8Q1subWSMrdQCUUk3DU6fA
ubuntu@ip-172-31-12-195:~/k8s-install$
```

Kubernetes Dashboard Overview

- After successfully installing and configuring the Kubernetes Dashboard, we have logged in using the generated token. The following screenshots showcase different sections of the dashboard, providing a visual overview of the Kubernetes cluster's components and resources. These include the main dashboard interface, details of the running pods, active deployments, and replica sets. This graphical interface offers an intuitive way to monitor, manage, and troubleshoot the cluster, making it easier to visualize resource allocation, deployment status, and overall cluster health.

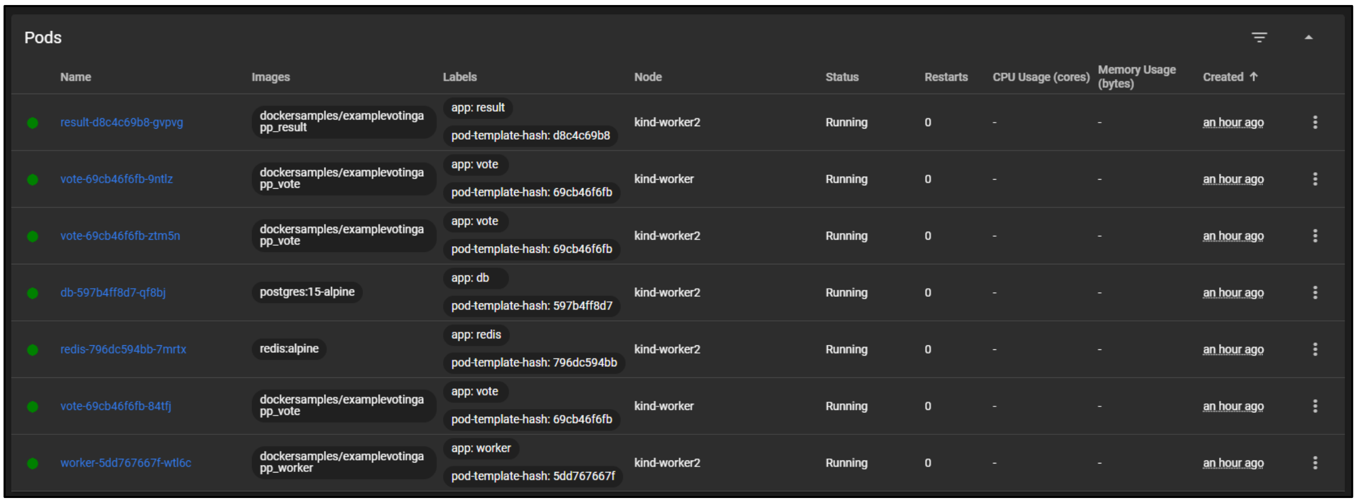


Name	Images	Labels	Pods	Created ↑
db	postgres:15-alpine	app: db app.kubernetes.io/instance: voting-app	1 / 1	an hour ago



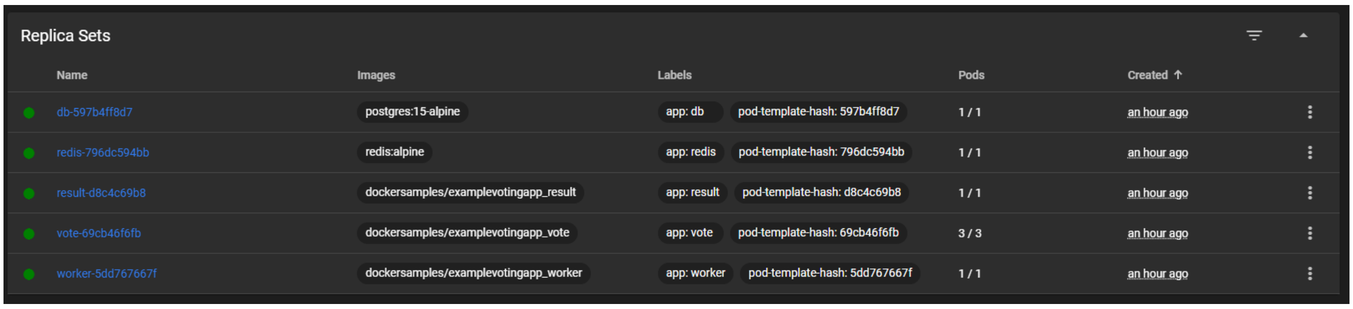
Deployments

Name	Images	Labels	Pods	Created ↑
db	postgres:15-alpine	app: db app.kubernetes.io/instance: voting-app	1 / 1	an hour ago
redis	redis:alpine	app: redis app.kubernetes.io/instance: voting-app	1 / 1	an hour ago
result	dockersamples/examplevotingapp_result	app: result app.kubernetes.io/instance: voting-app	1 / 1	an hour ago
vote	dockersamples/examplevotingapp_vote	app: vote app.kubernetes.io/instance: voting-app	3 / 3	an hour ago
worker	dockersamples/examplevotingapp_worker	app: worker app.kubernetes.io/instance: voting-app	1 / 1	an hour ago



Pods

	Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created ↑
●	result-d8c4c69b8-gvpvg	dockersamples/examplevotingapp_result	app: result pod-template-hash: d8c4c69b8	kind-worker2	Running	0	-	-	an hour ago
●	vote-69cb46f6fb-9ntlz	dockersamples/examplevotingapp_vote	app: vote pod-template-hash: 69cb46f6fb	kind-worker	Running	0	-	-	an hour ago
●	vote-69cb46f6fb-ztm5n	dockersamples/examplevotingapp_vote	app: vote pod-template-hash: 69cb46f6fb	kind-worker2	Running	0	-	-	an hour ago
●	db-597b4ff8d7-qf8bj	postgres:15-alpine	app: db pod-template-hash: 597b4ff8d7	kind-worker2	Running	0	-	-	an hour ago
●	redis-796dc594bb-7mrtx	redis:alpine	app: redis pod-template-hash: 796dc594bb	kind-worker2	Running	0	-	-	an hour ago
●	vote-69cb46f6fb-84tfj	dockersamples/examplevotingapp_vote	app: vote pod-template-hash: 69cb46f6fb	kind-worker	Running	0	-	-	an hour ago
●	worker-5dd767667f-wtl6c	dockersamples/examplevotingapp_worker	app: worker pod-template-hash: 5dd767667f	kind-worker2	Running	0	-	-	an hour ago



Replica Sets

	Name	Images	Labels	Pods	Created ↑
●	db-597b4ff8d7	postgres:15-alpine	app: db pod-template-hash: 597b4ff8d7	1 / 1	an hour ago
●	redis-796dc594bb	redis:alpine	app: redis pod-template-hash: 796dc594bb	1 / 1	an hour ago
●	result-d8c4c69b8	dockersamples/examplevotingapp_result	app: result pod-template-hash: d8c4c69b8	1 / 1	an hour ago
●	vote-69cb46f6fb	dockersamples/examplevotingapp_vote	app: vote pod-template-hash: 69cb46f6fb	3 / 3	an hour ago
●	worker-5dd767667f	dockersamples/examplevotingapp_worker	app: worker pod-template-hash: 5dd767667f	1 / 1	an hour ago

In this project, we successfully demonstrated the **Automated Deployment of Scalable Applications on AWS EC2 with Kubernetes and Argo CD**.

Starting from launching an EC2 instance, we configured Docker, installed Kind to create a multi-node Kubernetes cluster, and deployed applications seamlessly using Argo CD. We also integrated the Kubernetes Dashboard to provide a visual interface for managing and monitoring cluster resources effectively.

Through this setup, we showcased how scalable applications can be deployed, managed, and synchronized automatically from a GitHub repository, ensuring efficient CI/CD workflows. The ability to monitor application health, manage pods, deployments, and replica sets through both Argo CD and the Kubernetes Dashboard highlights the power of automation and container orchestration in modern cloud environments.

This project not only demonstrates the deployment of scalable applications but also lays a strong foundation for managing cloud-native applications with ease, flexibility, and robustness.