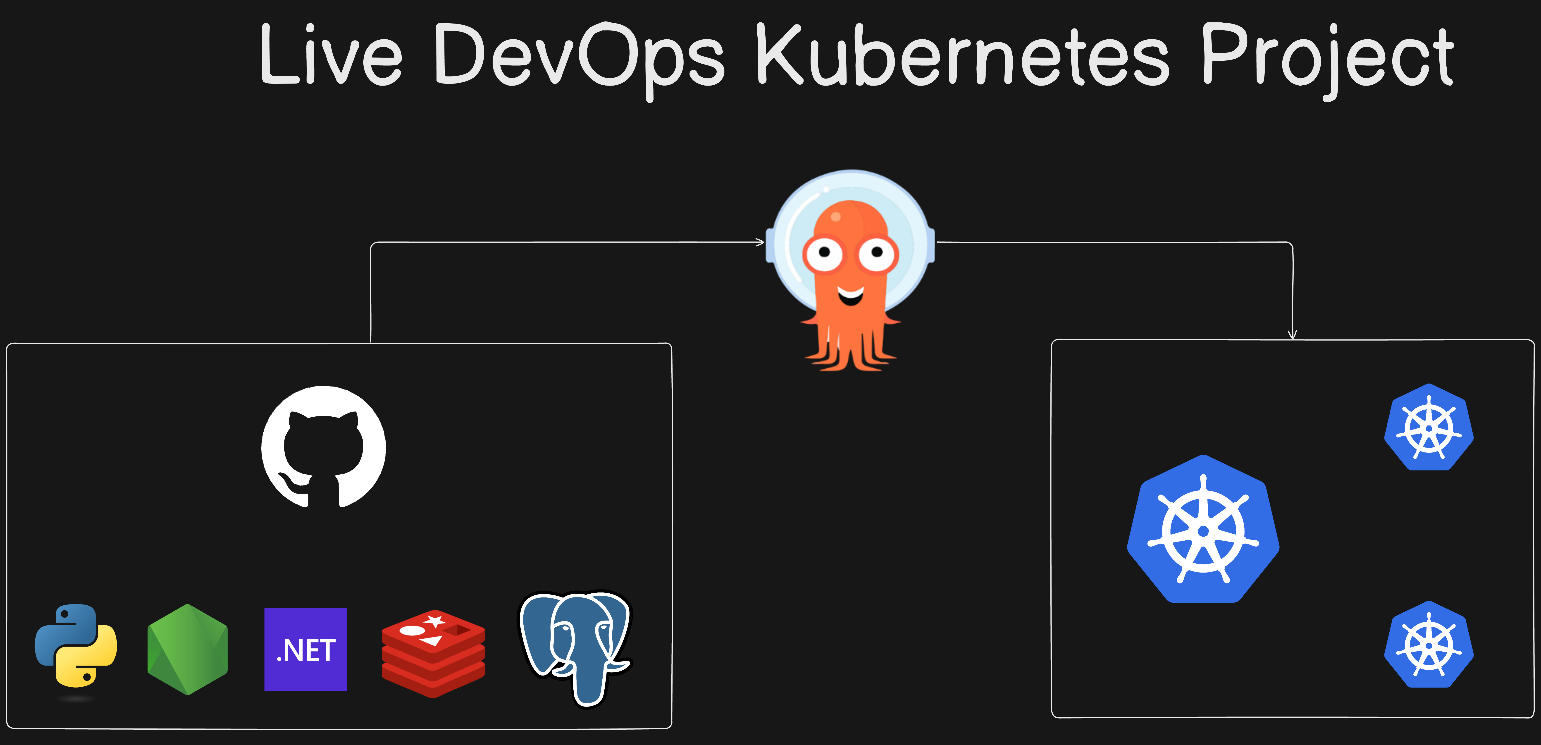
# Automated Deployment of Scalable Applications on AWS EC2 with Kubernetes and Argo CD

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.

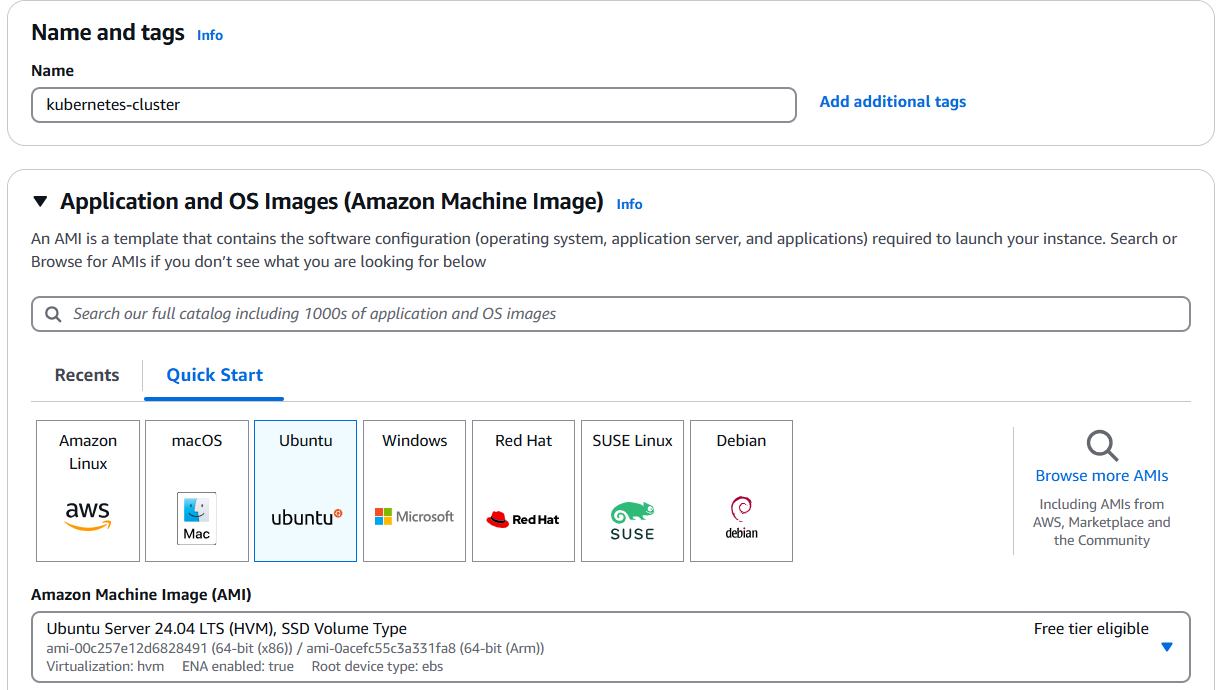
These components interact seamlessly to provide a full-stack voting system where users can cast votes, vote data is stored, and results are displayed in real-time.

**Preliminary Steps: Preparing the Project from the External GitHub Repository**

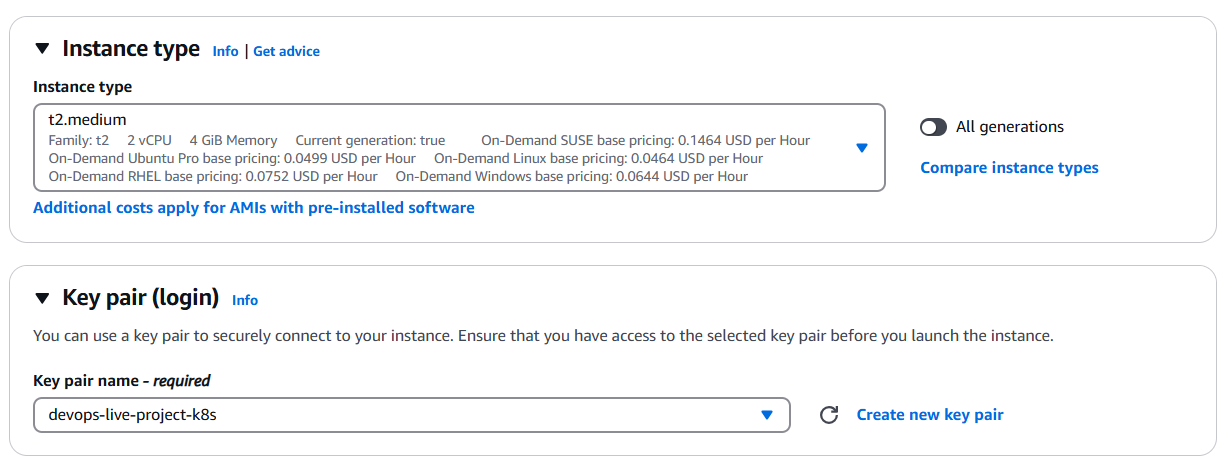
* Before starting with the deployment and other project-related tasks, you need to clone the project from the external GitHub repository available at **https://github.com/LondheShubham153/k8s-kind-voting-app.git**. This repository contains the application that we will deploy. After cloning it to your 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**

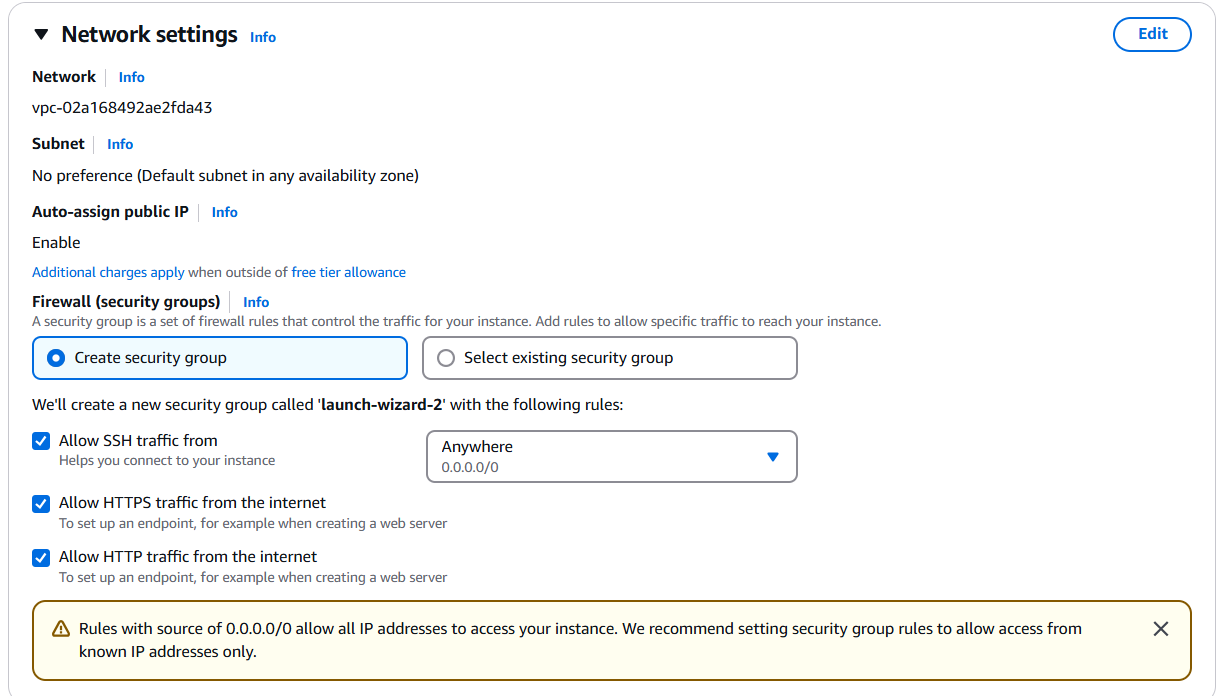
* **Enter the Name of the Instance**, eg: kubernetes-cluster
* Choose **Ubuntu Server 24.04 LTS (HVM)** under **Amazon Machine Image(AMI)**

****

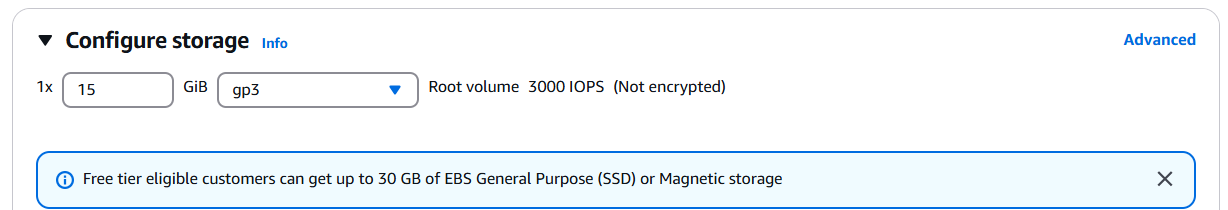
* Choose **t2.medium** under **Instance type**.
* Under **Key pair (login)**, give your key pair name or create a new key pair eg: devops-live-project-k8s is my keypair.



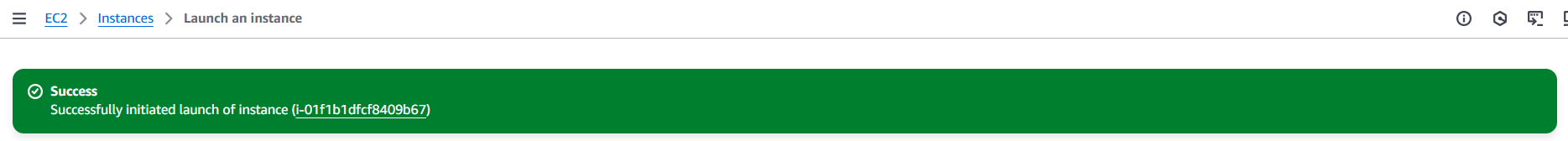
* In the network settings, choose the default VPC and subnet, and enable the option to auto-assign a public IP. For the firewall security groups, create a new security group and configure the following rules: allow **SSH** traffic from anywhere (port 22) to enable secure remote access, allow **HTTP**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

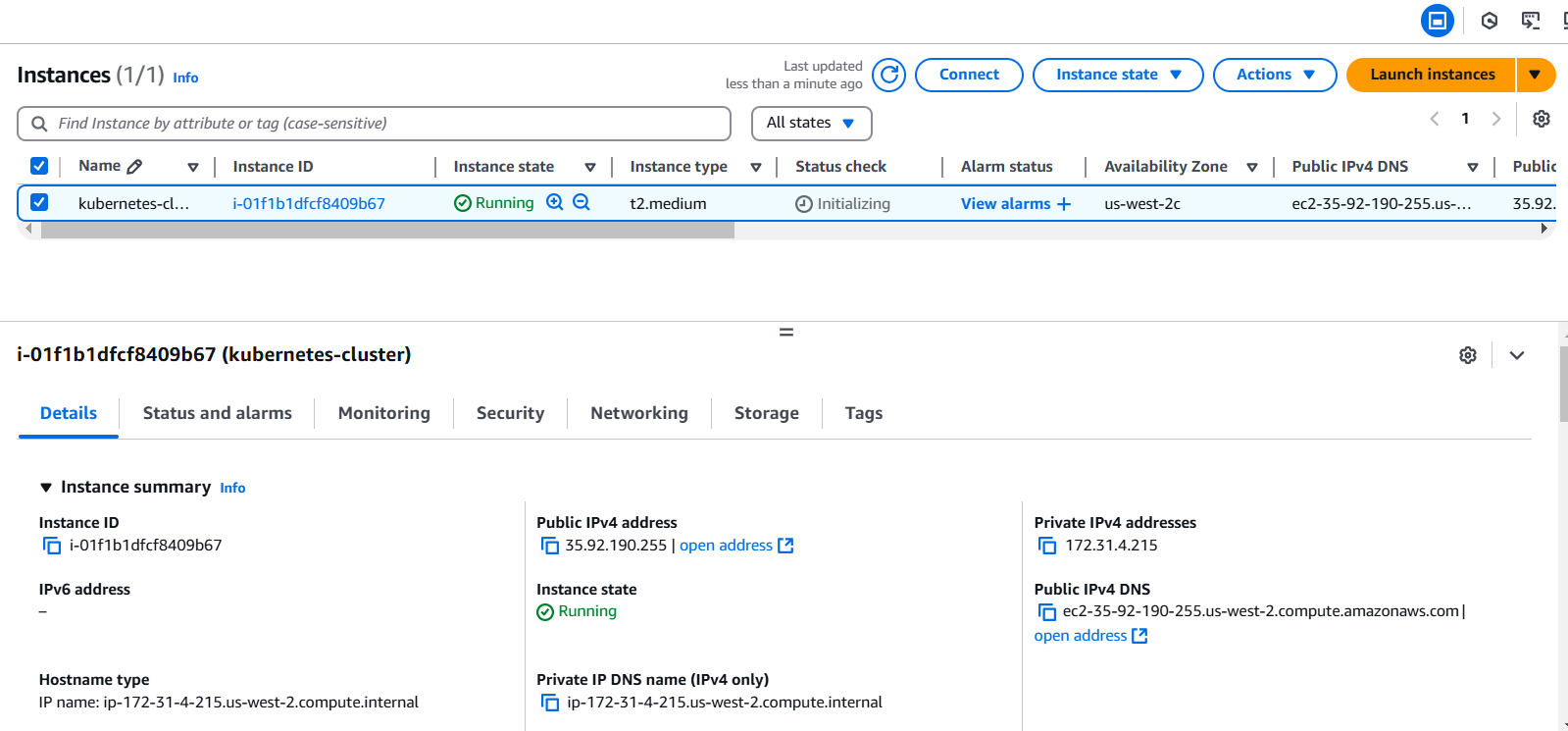


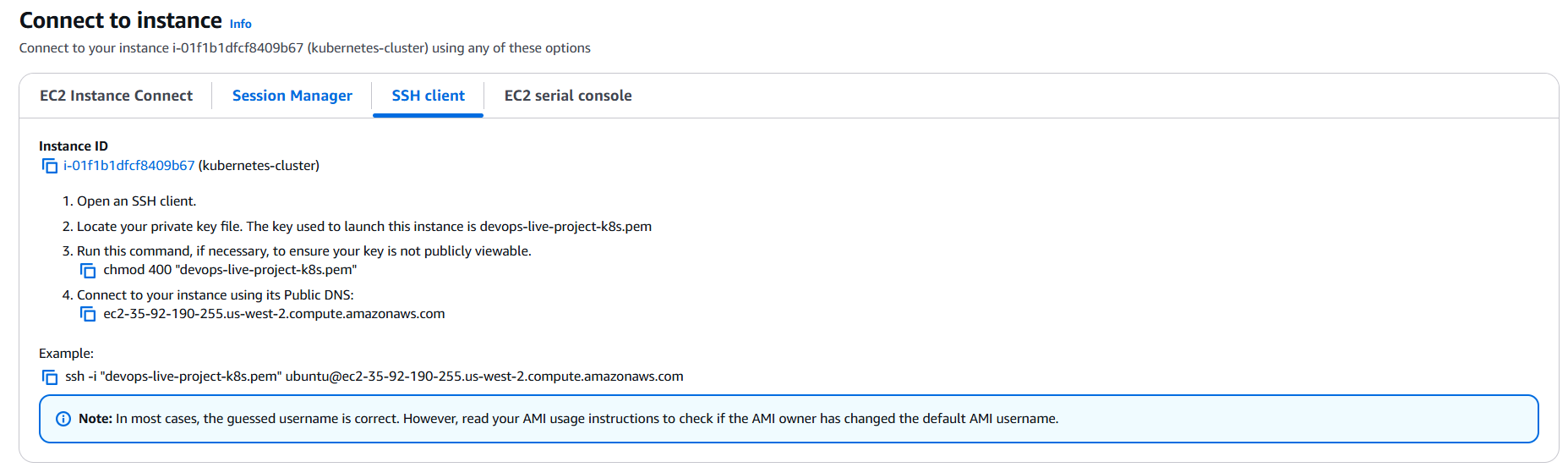
* Configure the storage settings by setting the root volume size to **15 GiB** to ensure sufficient space for the application and related dependencies.



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





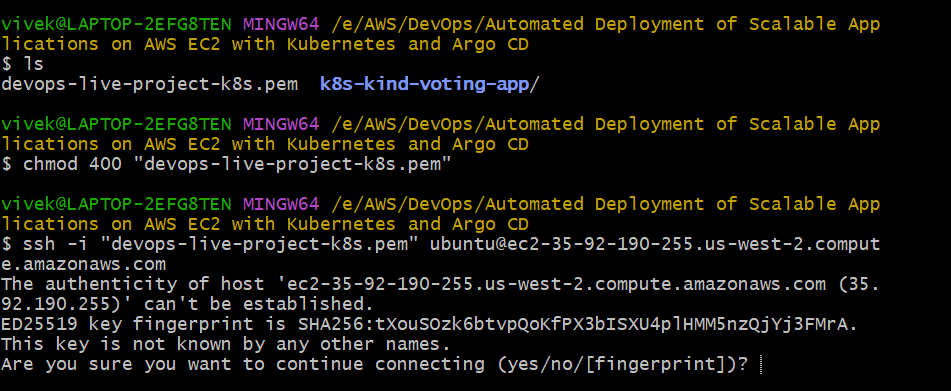


* To connect to the EC2 instance using an SSH client, open the **Git Bash terminal** on your local computer. Since we've already created the key pair named **devops-live-project-k8s** during the instance setup and downloaded it to our local machine, navigate to the directory where the key pair file (devops-live-project-k8s.pem) 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**:

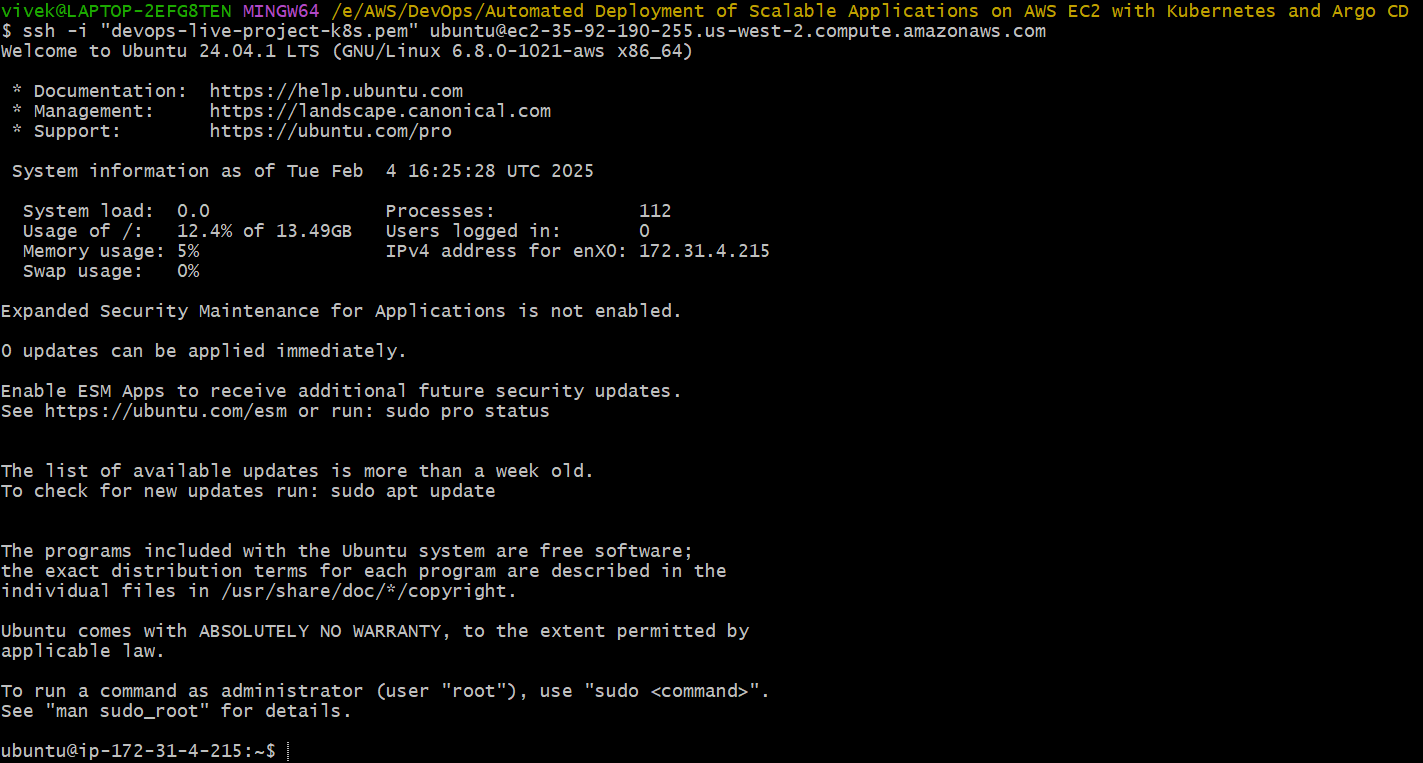
chmod 400 "devops-live-project-k8s.pem"

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;



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

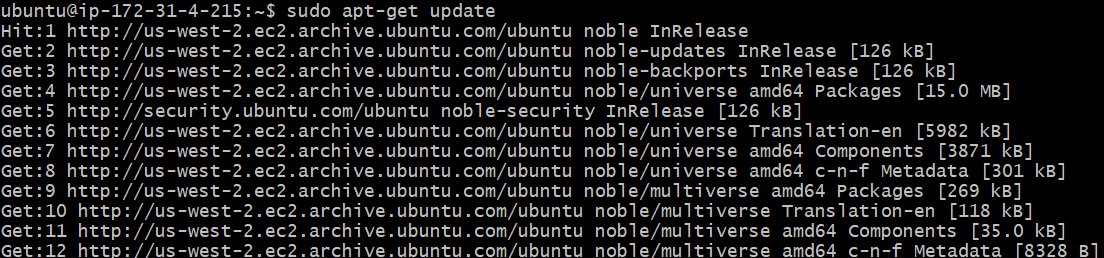


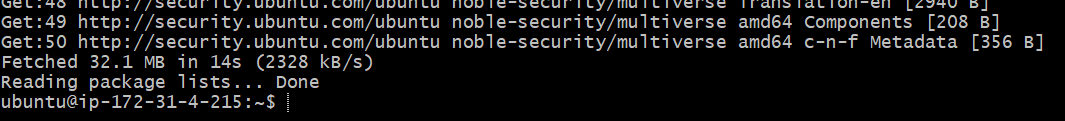
**Install Docker and Kind.**

* 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:

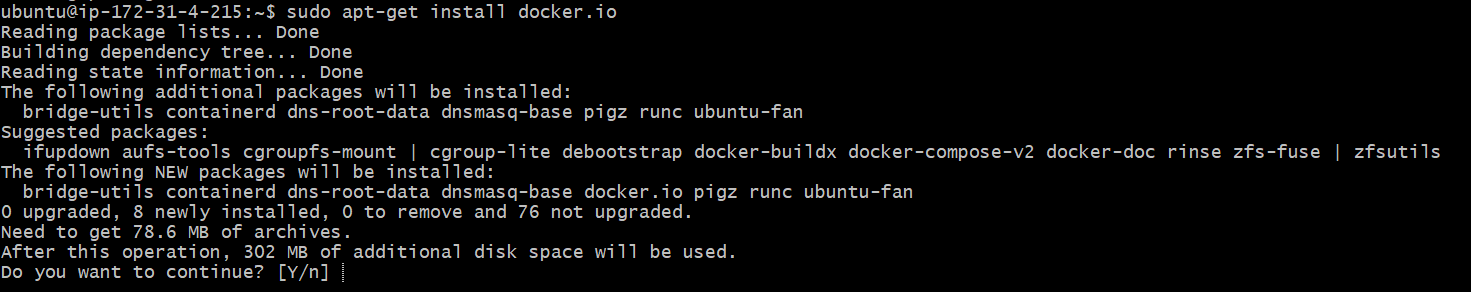
sudo apt-get update



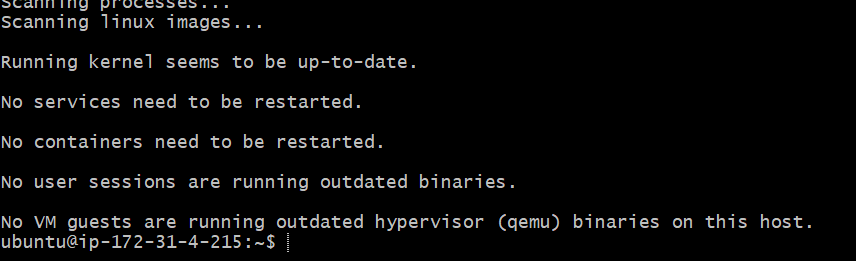


1. **Install Docker** by running the following command:

sudo apt-get install docker.io



**Type Y**

****

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.

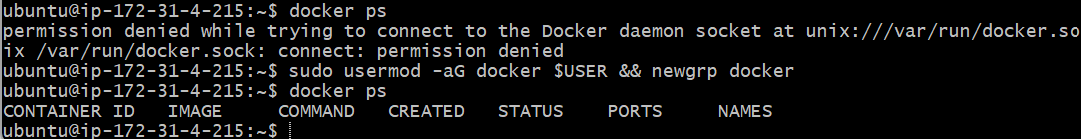
The sudo usermod -aG docker $USER part of the command adds your user ($USER) to the docker group, enabling your user to execute Docker commands.

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.



* 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.



* + 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

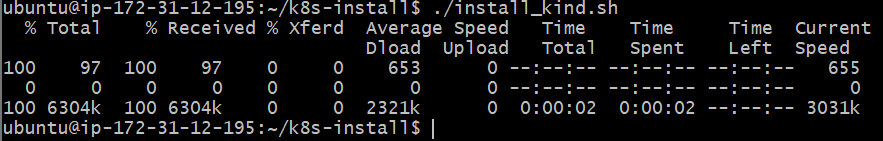
* + Save the script and exit vim.



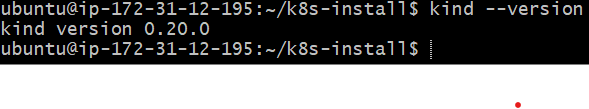
* + Grant executable permissions using chmod +x install\_kind.sh



* + Execute the script with ./install\_kind.sh.



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



**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.

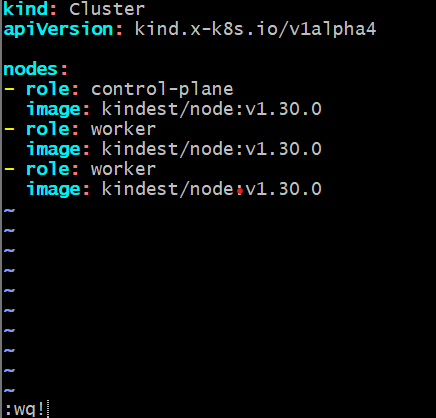
- role: worker     # Defines the third node as an additional worker node for better scalability.

  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.

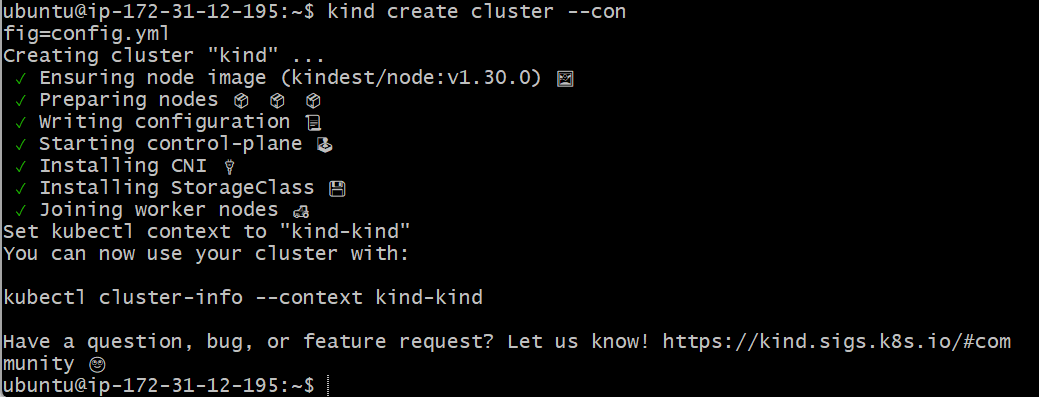




* Now that the config.yml file is set up, you can create a 3-node Kubernetes cluster using Kind by executing the command:

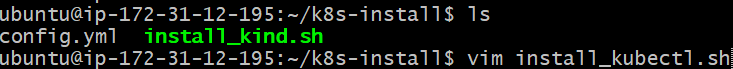
kind create cluster --config=config.yml.

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.



**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.



* + 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/kubectl"

INSTALL\_DIR="/usr/local/bin"

# Download and install kubectl

curl -LO "$URL"

chmod +x kubectl

sudo mv kubectl $INSTALL\_DIR/

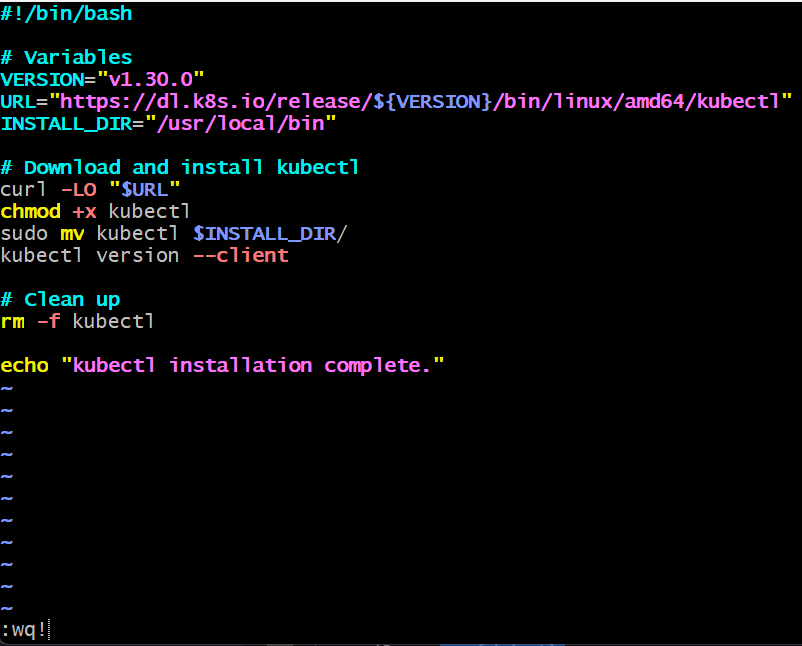
kubectl version --client

# Clean up

rm -f kubectl

echo "kubectl installation complete."

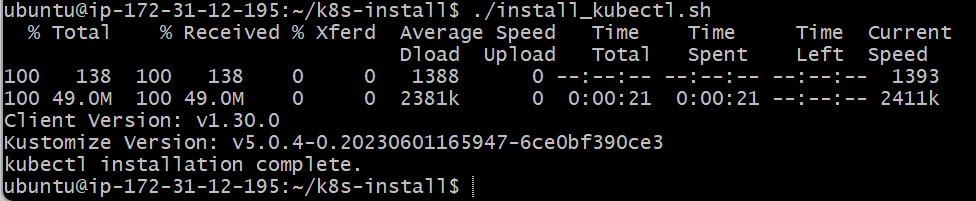
* + Save the script and exit vim.



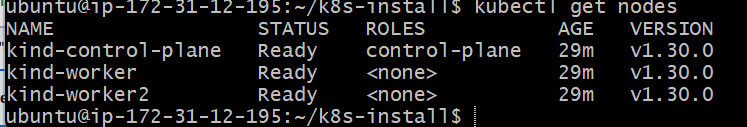
* + Grant executable permissions using chmod +x install\_kubectl.sh



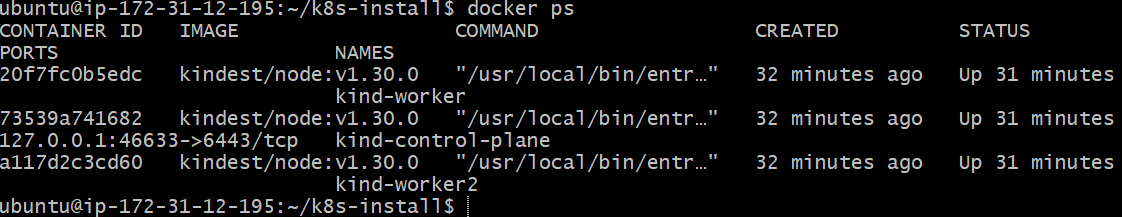
* + Execute the script with ./install\_kubectl.sh.



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



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



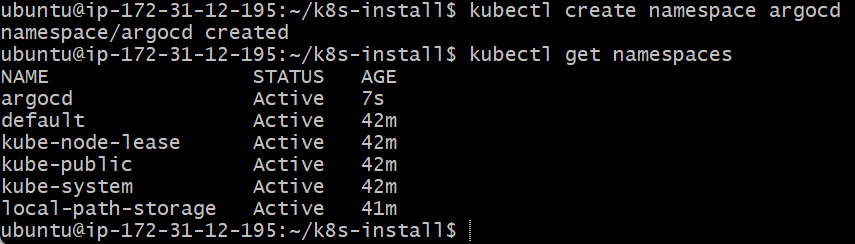
**Install and configure Argo CD.**

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.

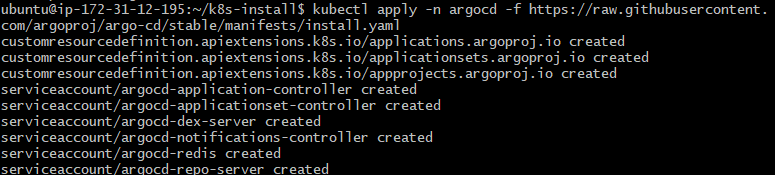
We can install Argo CD by carrying out the following steps:

* **Create a namespace for Argo CD**:  
  First, create a dedicated namespace for Argo CD using the command

kubectl create namespace argocd.

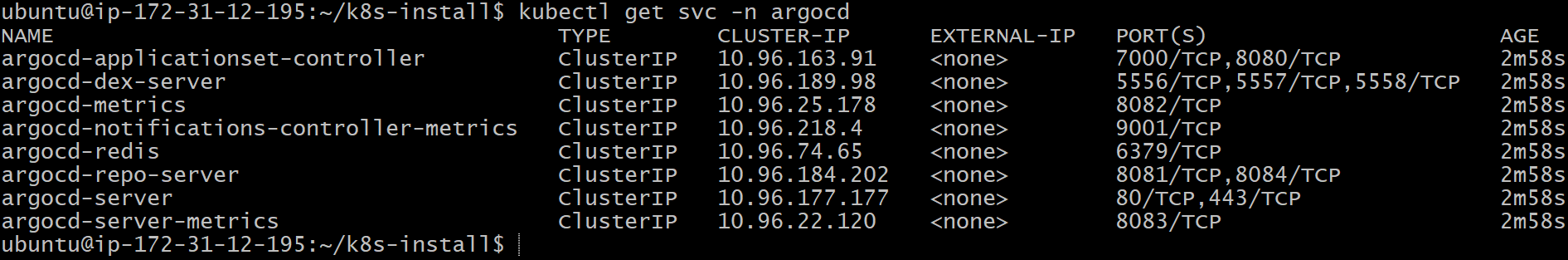


* **Apply the Argo CD manifest**:  
  Deploy Argo CD by applying the official Argo CD manifest with the command kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml.



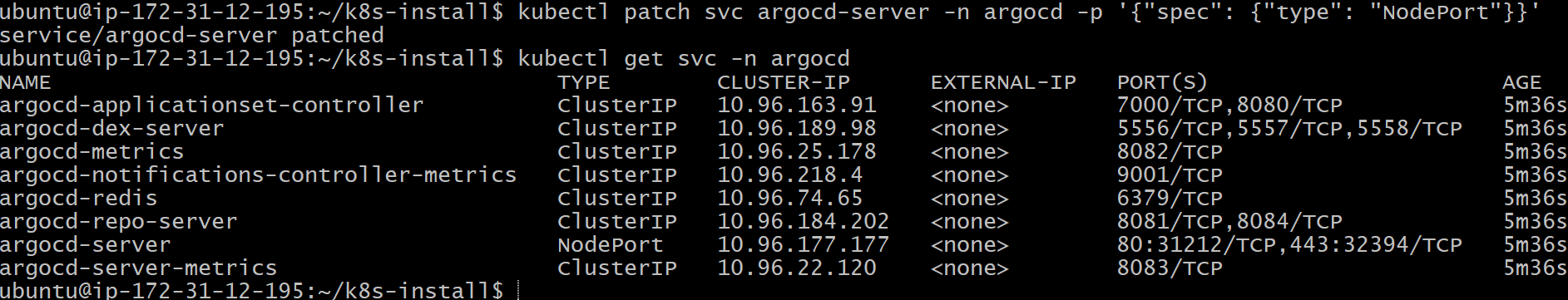
* **Check services in the Argo CD namespace**:

To verify that the services are running in the argocd namespace, execute the command kubectl get svc -n argocd.

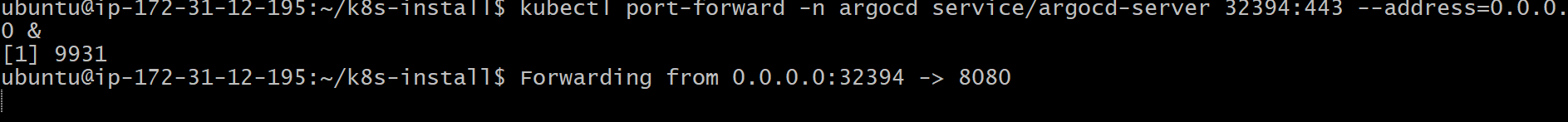


* **Expose the Argo CD server using NodePort**:  
  Expose the Argo CD server service using NodePort to make it accessible from outside the cluster with the command

kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "NodePort"}}'



* **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 &



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:

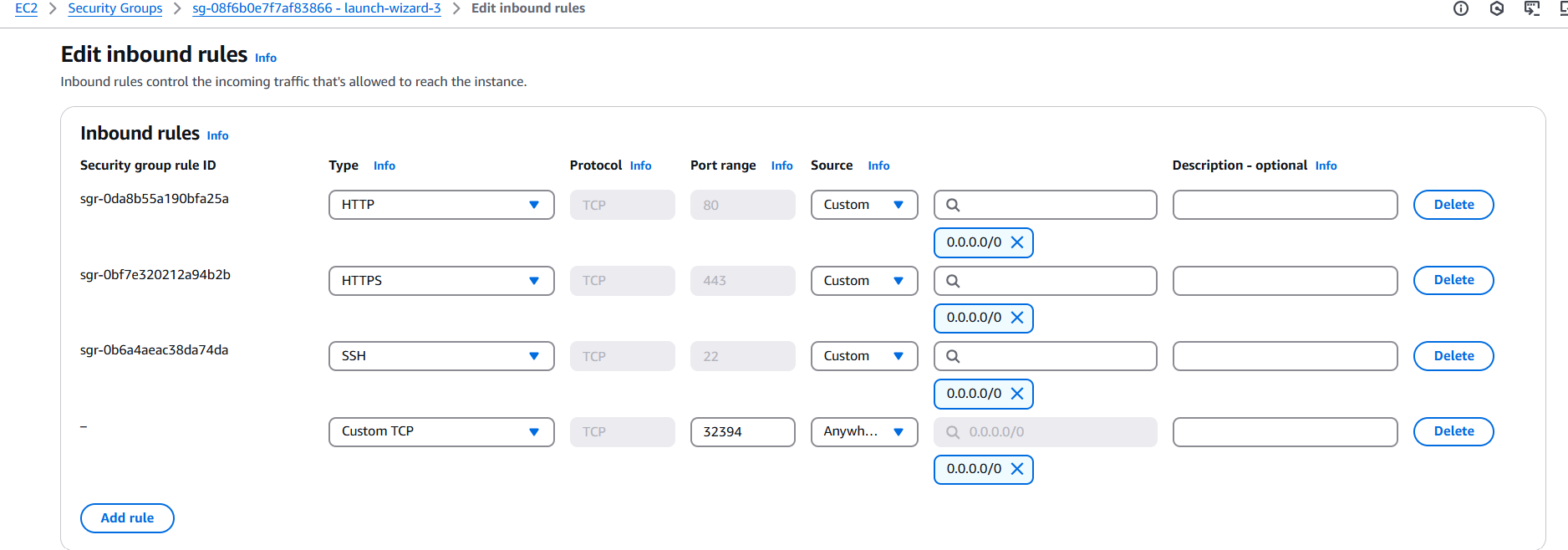
pkill -f "kubectl port-forward"

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.

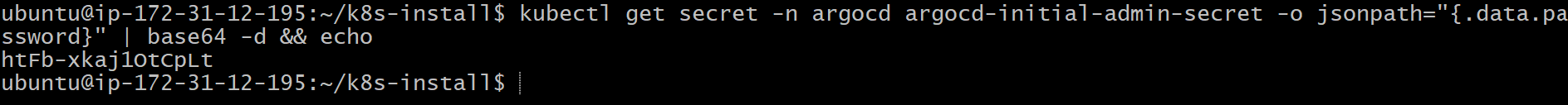




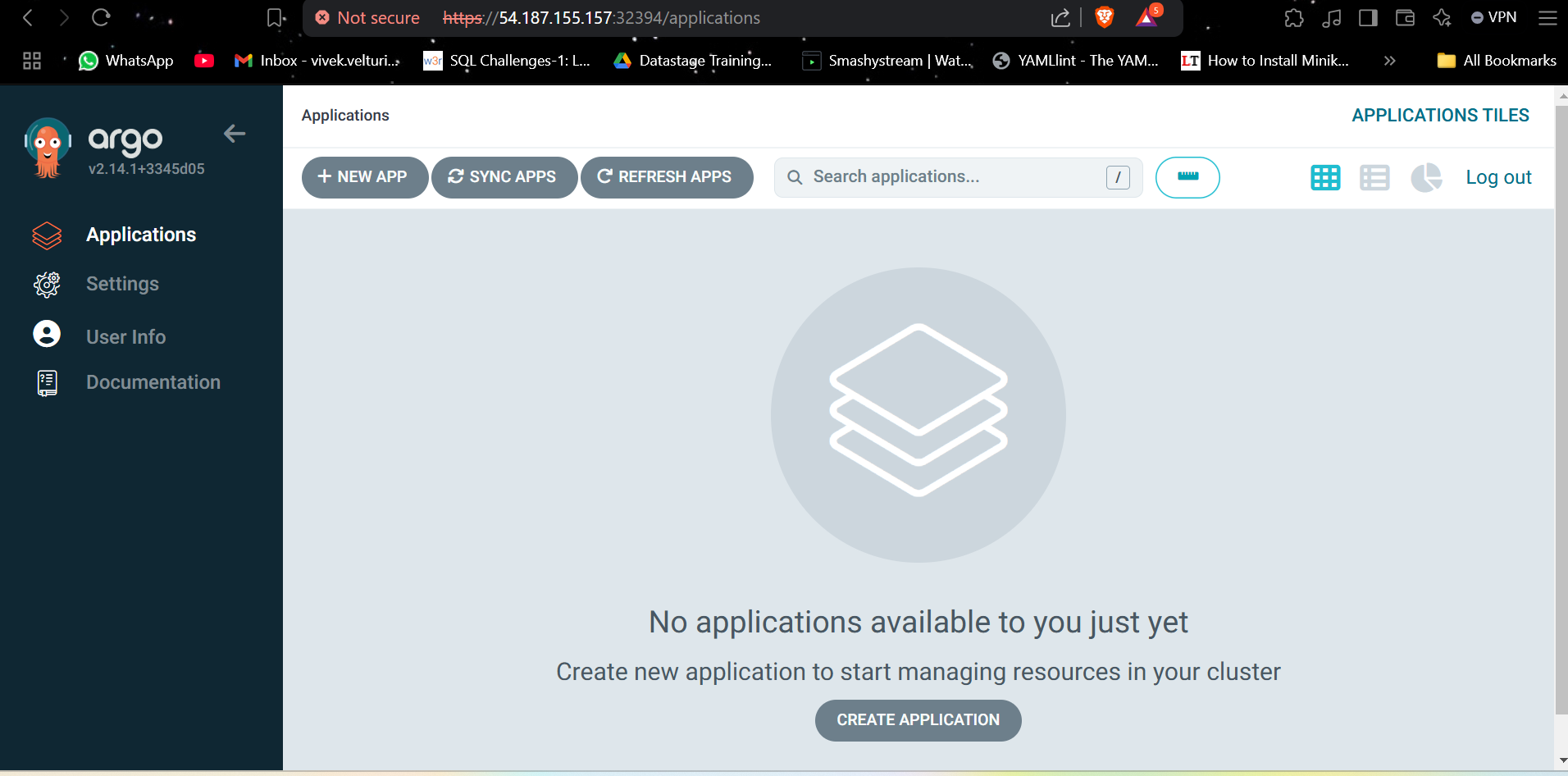
* To log in to the Argo CD web interface, the default **username** is **admin**, and you’ll need to retrieve the **initial admin password** generated during the Argo CD installation. This password is stored as a Kubernetes secret in the argocd namespace.

To obtain it, run the following command:

kubectl get secret -n argocd argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d && echo



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.

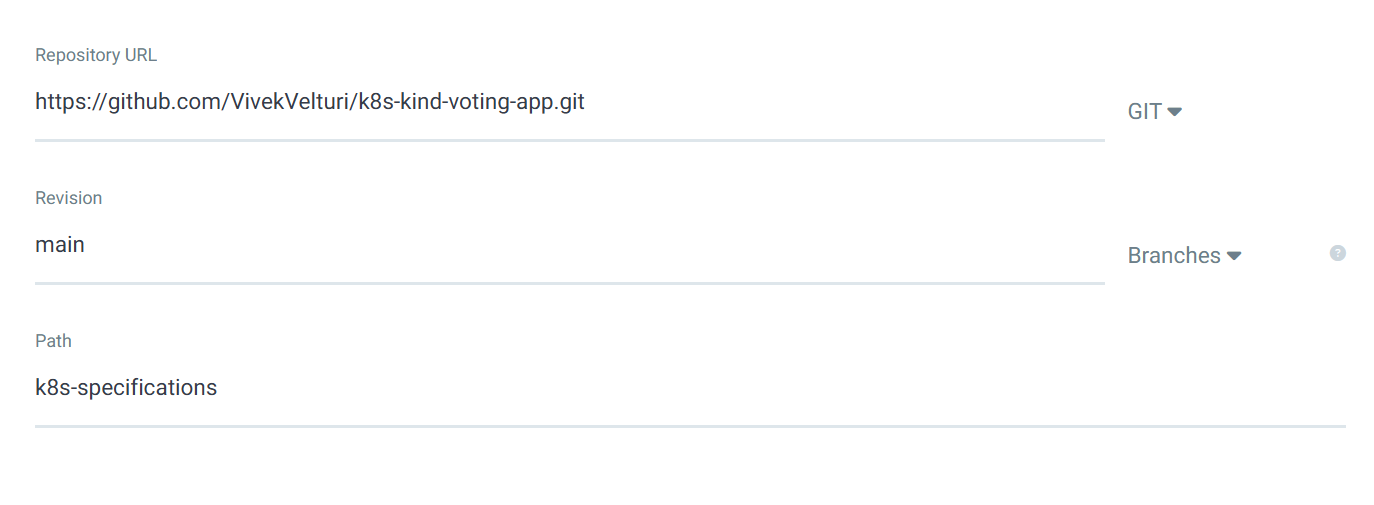


**Connect and manage your Kubernetes cluster with Argo CD.**

* 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.

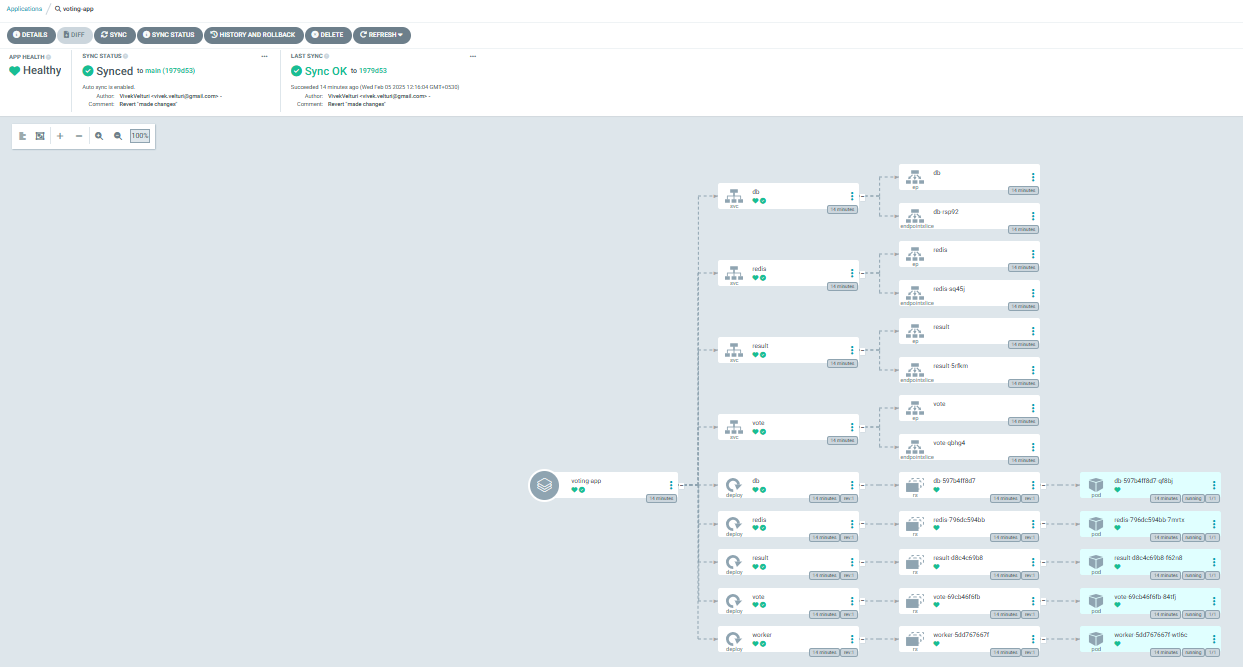




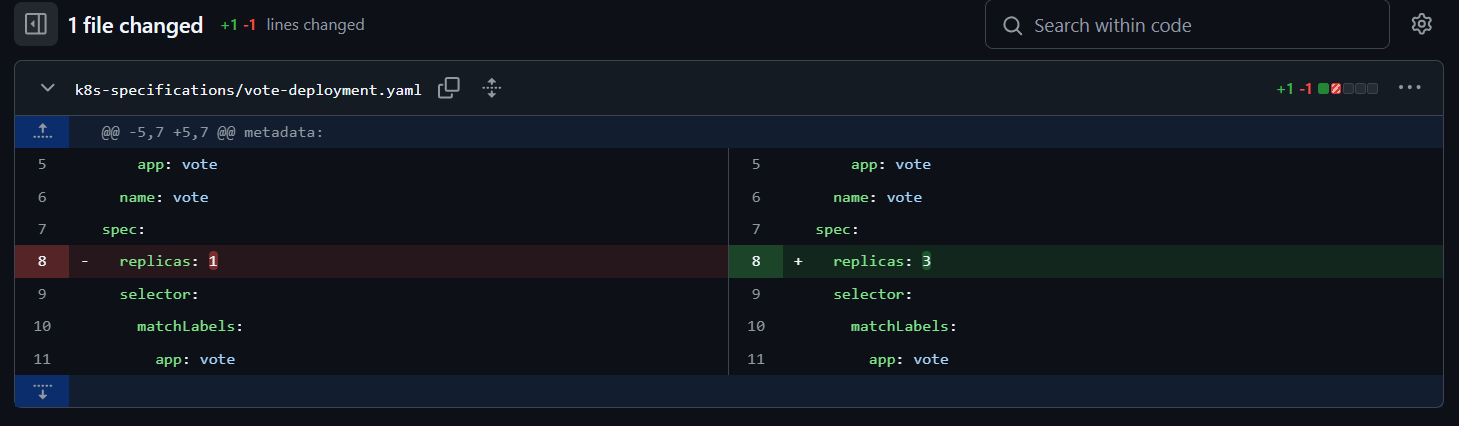


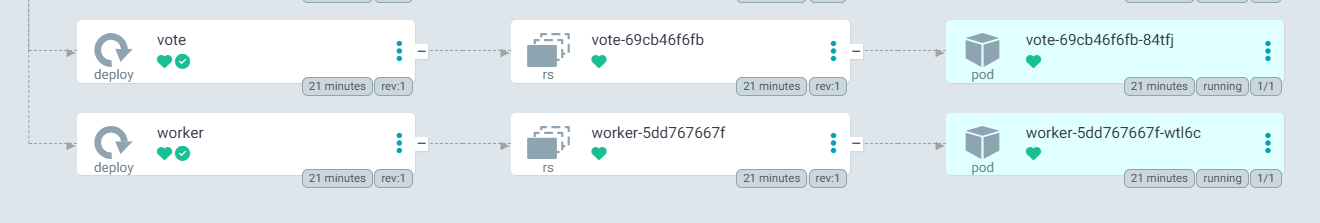


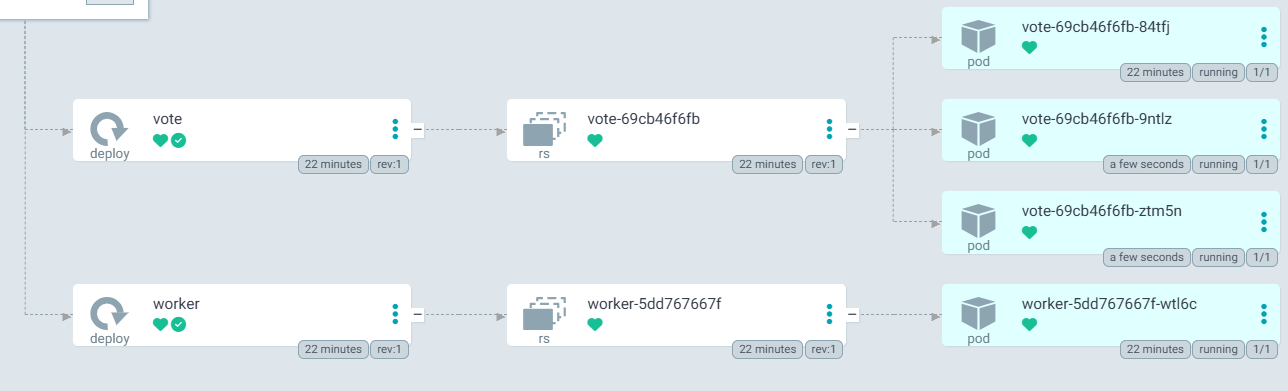
* Click on the **voting-app** application in the Argo CD UI, and you will see that it is periodically syncing from the repository. It will display the status of the resources and create the pods as specified in the k8s-specifications folder of the repository. This process ensures that the desired state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



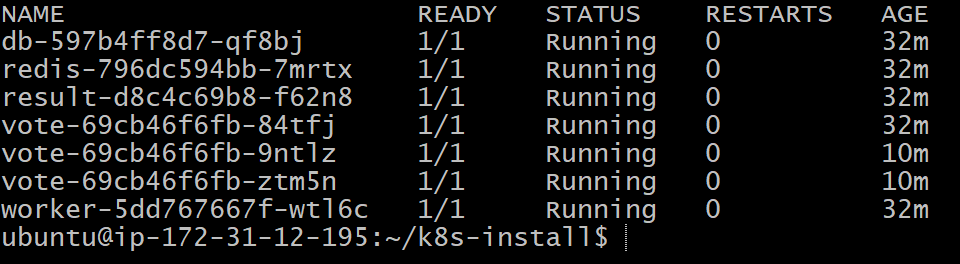
* To observe the sync and changes in Argo CD, we will edit the replica count in the vote.deployment YAML file located inside the k8s-specifications folder in our GitHub repository. Change the number of replicas from 1 to 3 in the spec.replicas section of the file. After saving and committing the changes, 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.



***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.

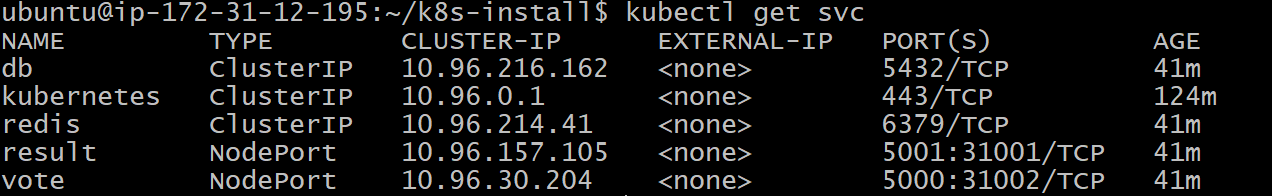


* Now, if you execute the command kubectl get deployments in our instance, This command will show the updated state of the deployment based on the latest changes applied in the repository and synced via Argo CD.

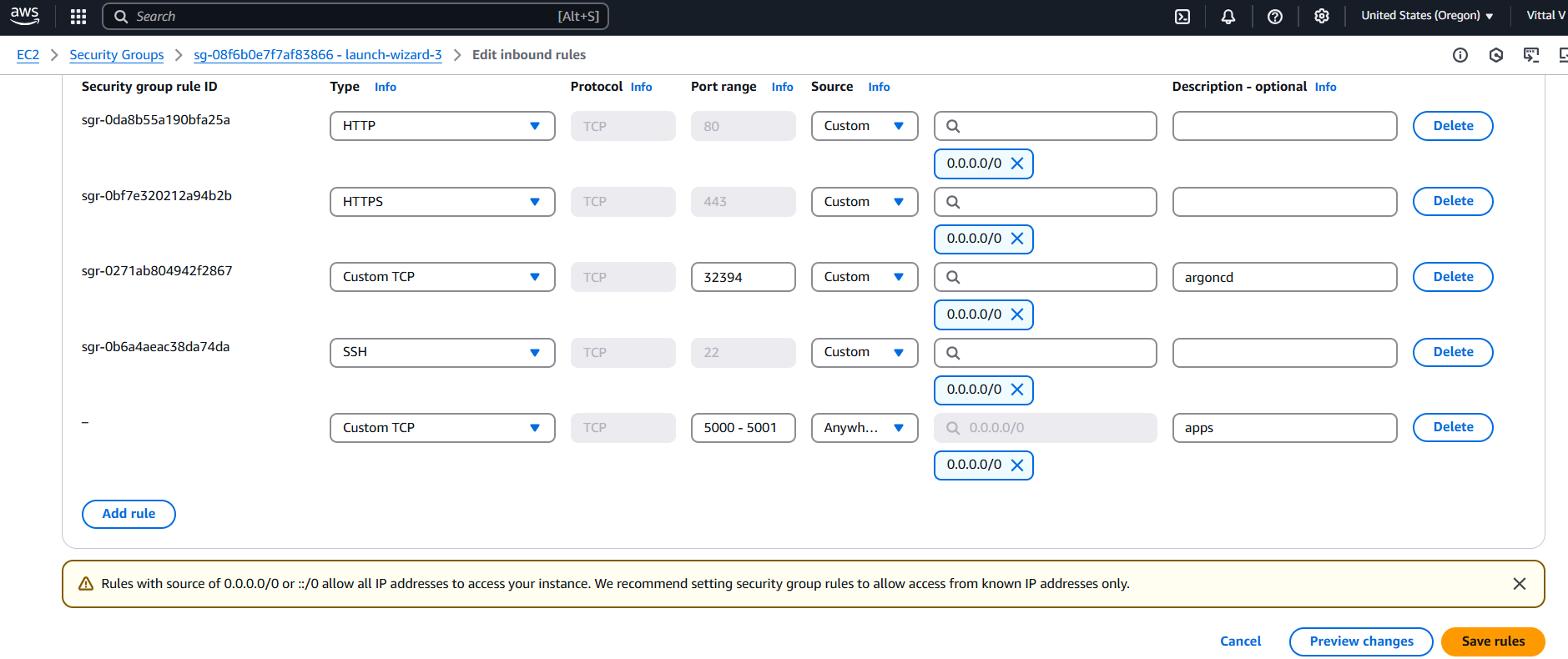


* In order to access the results and votes of our app in the browser, we need to find the ports on which the services are exposed. You can retrieve this information by executing the command kubectl get svc. This will display the services running in the Kubernetes cluster, along with their associated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



* 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.



* 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.

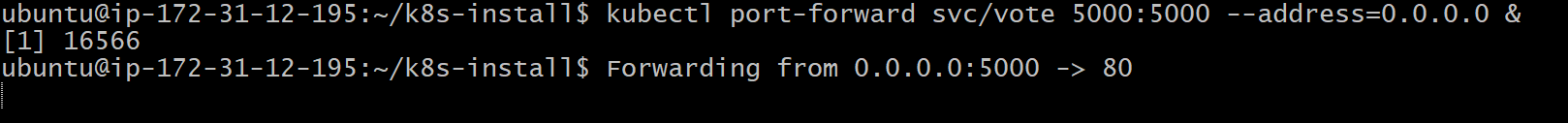
For the voting service, use

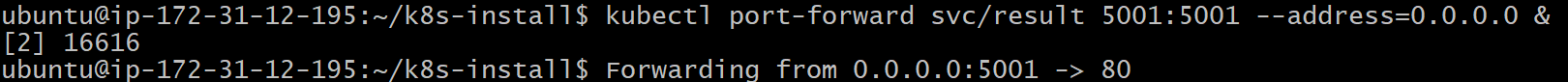
kubectl port-forward svc/vote 5000:5000 --address=0.0.0.0 &

and for the result service, use

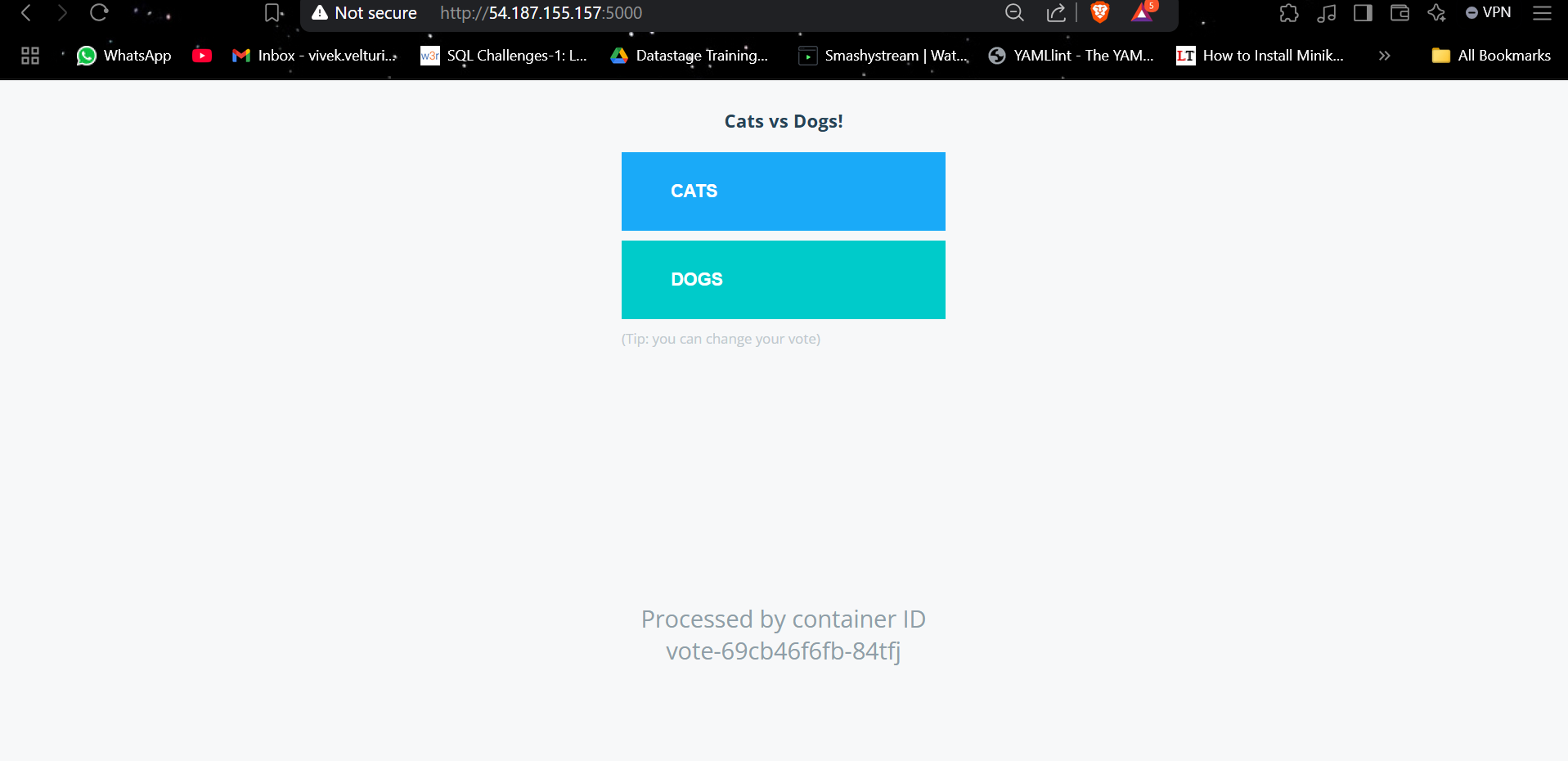
kubectl port-forward svc/result 5001:5001 --address=0.0.0.0 &

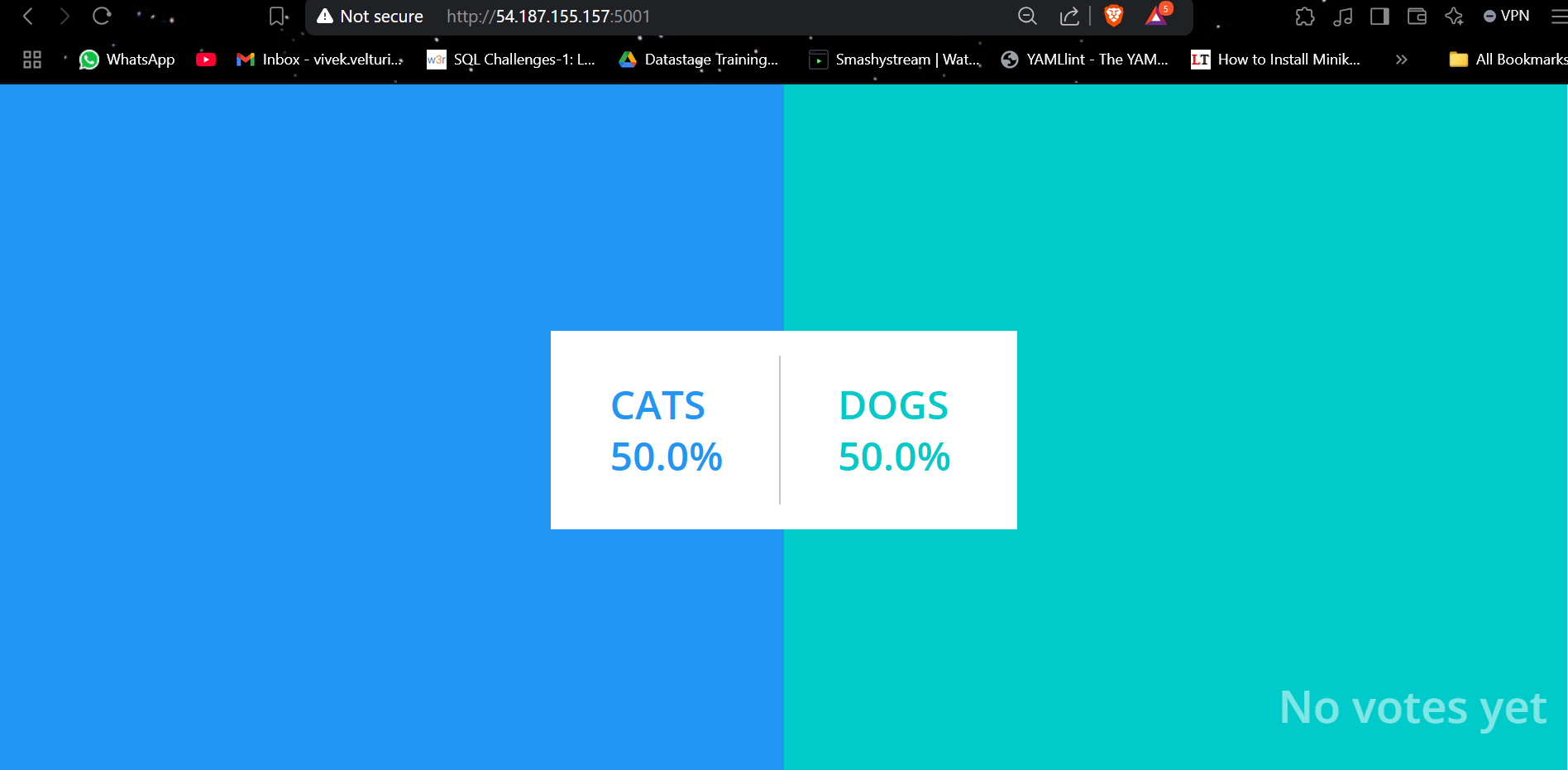
These commands will forward the ports 5000 and 5001 from the services in the cluster to your local machine. The --address=0.0.0.0 flag ensures that the services can be accessed from any address on the local machine.





* 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.



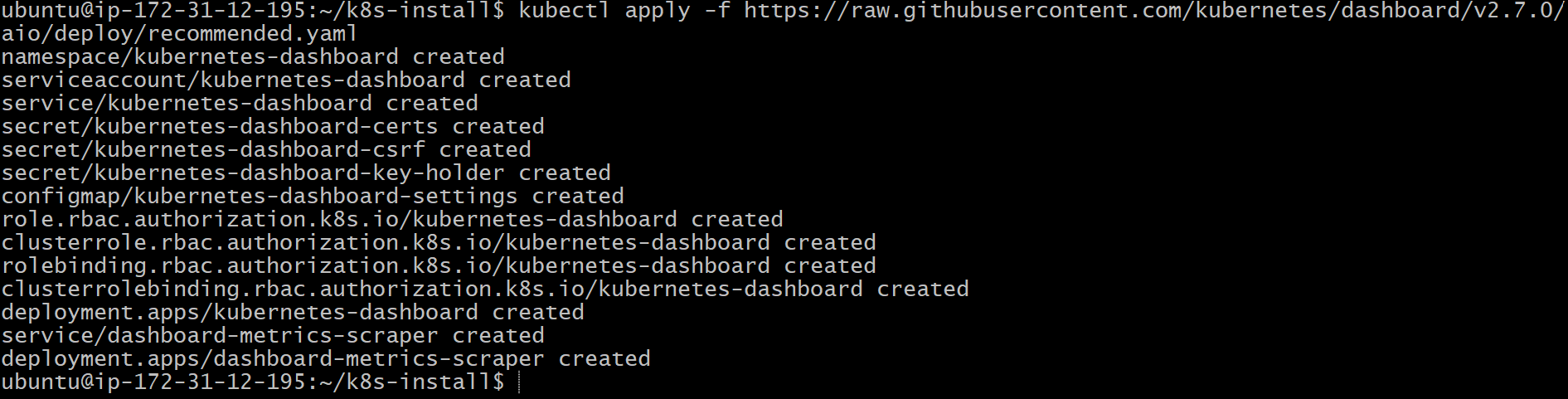


**Set up the Kubernetes Dashboard.**

* 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.



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

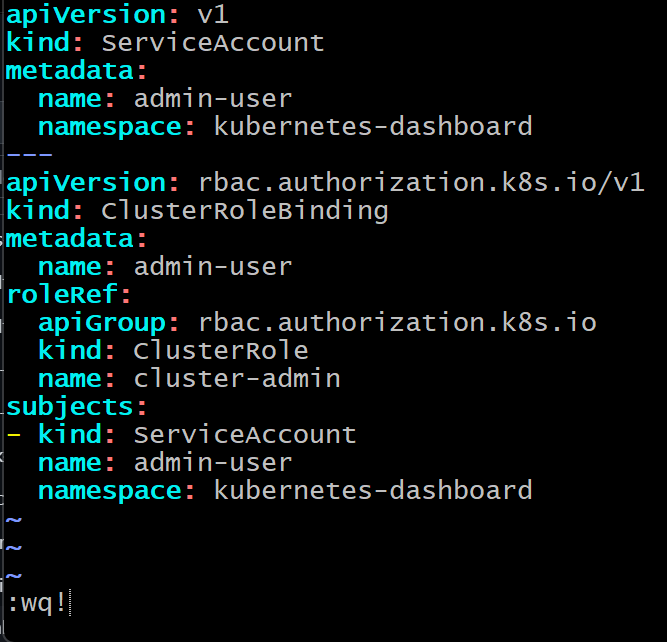
  name: cluster-admin  # The name of the ClusterRole being referenced. 'cluster-admin' grants superuser privileges at the cluster level.

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.

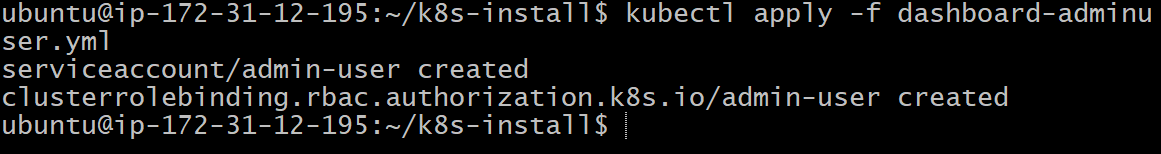
  namespace: kubernetes-dashboard  # The namespace in which the ServiceAccount exists, aligning with the namespace defined earlier.



* 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:

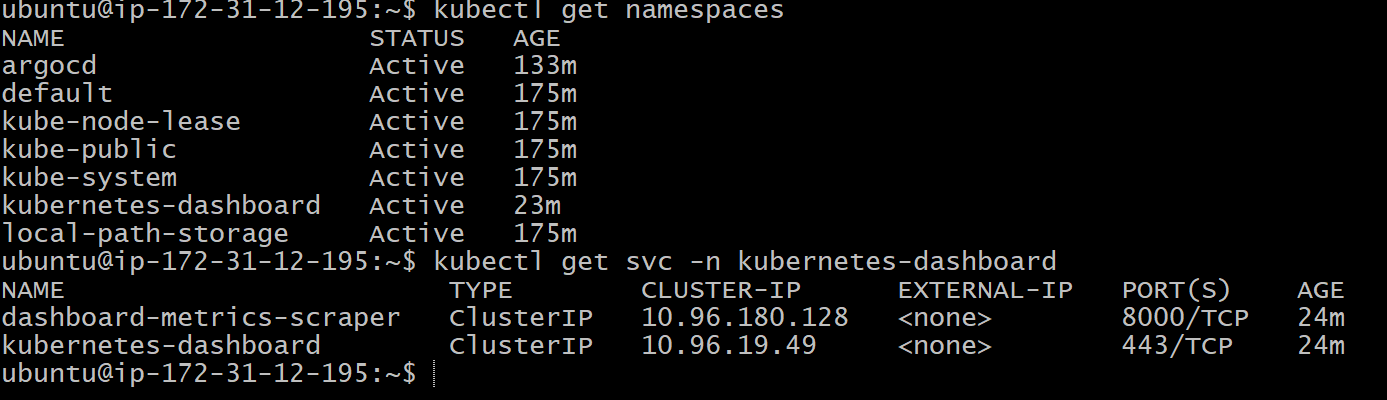
kubectl apply -f dashboard-adminuser.yml

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.



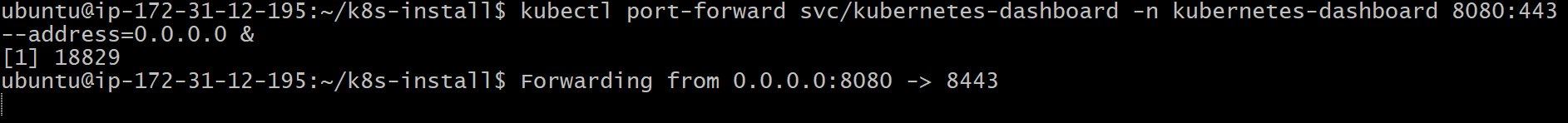
* To find the port where the Kubernetes Dashboard can be accessed, first run the command:

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.

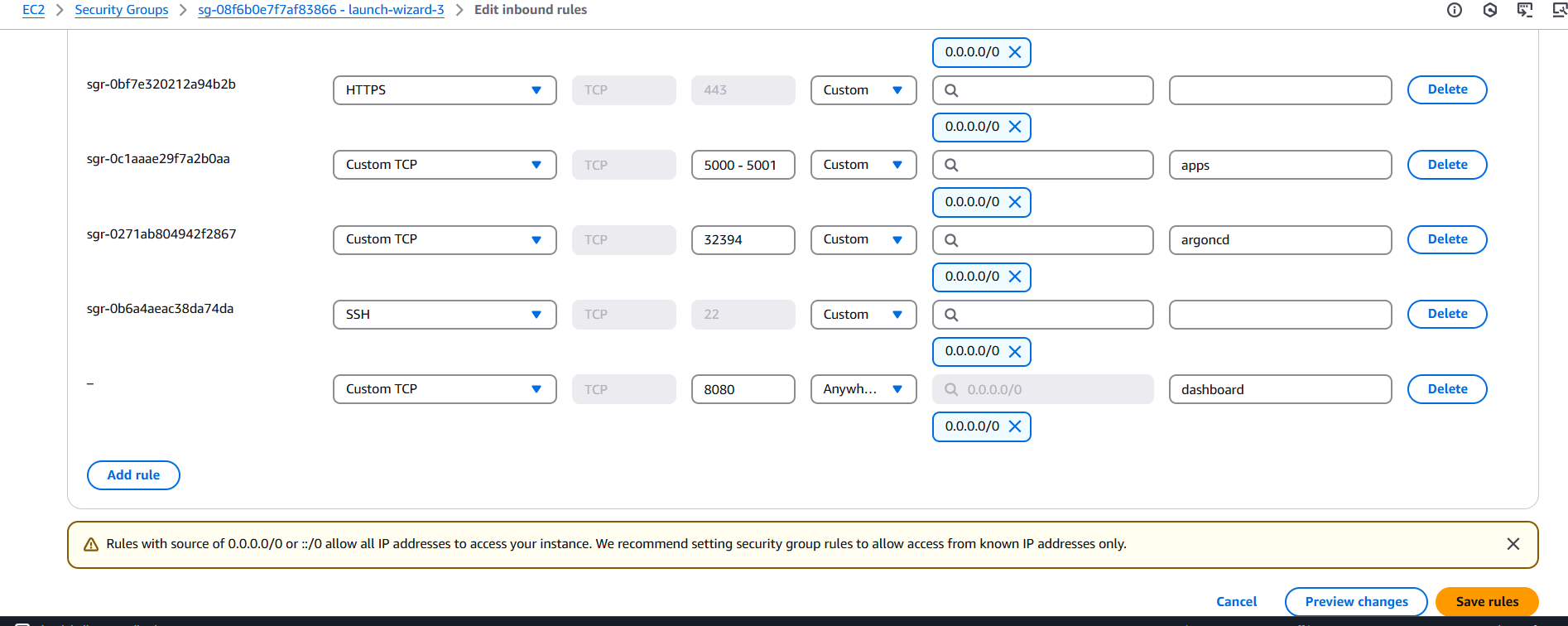


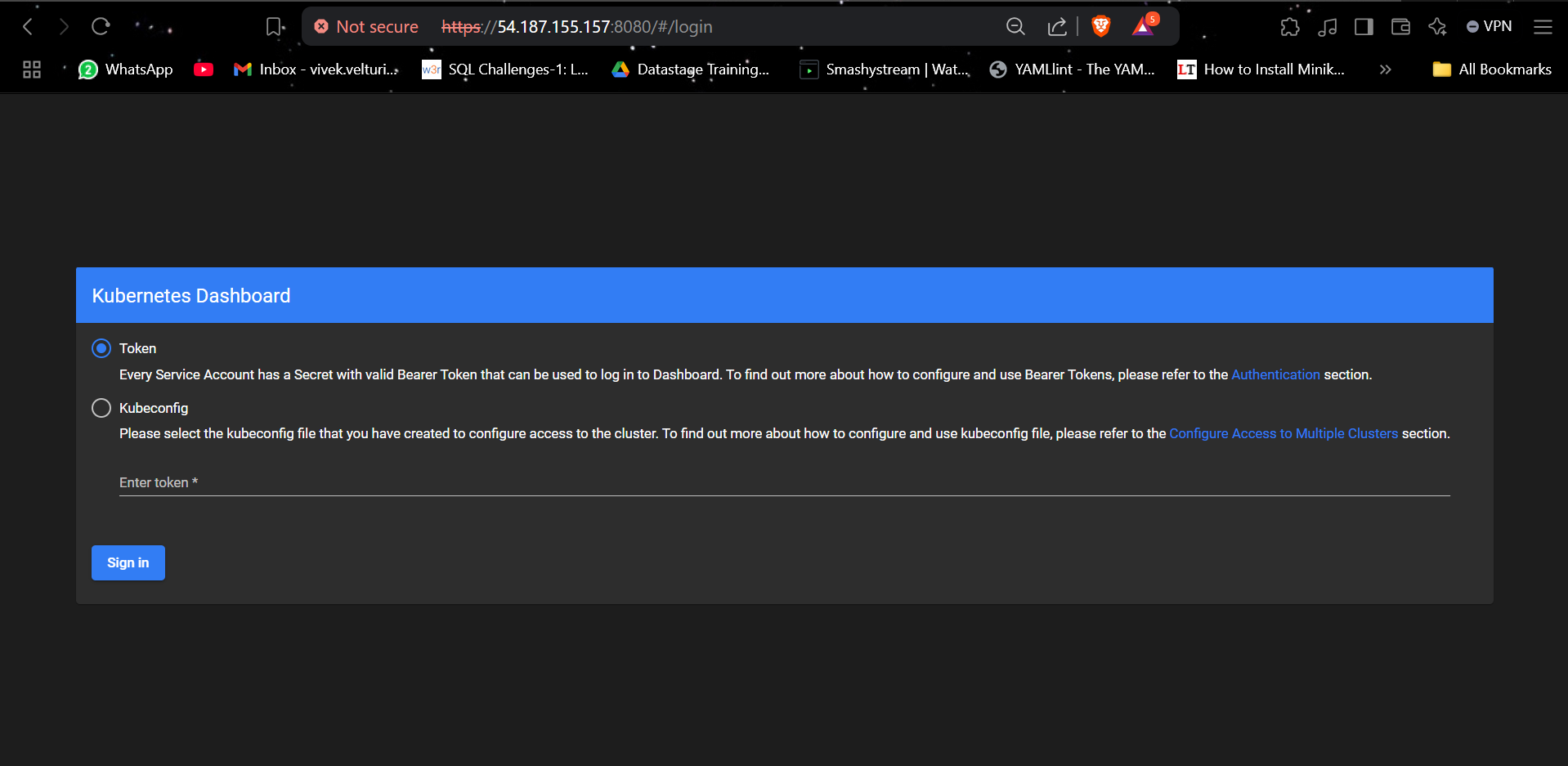
* To access the Kubernetes Dashboard, you first need to forward the necessary port by running the command:

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



* 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.





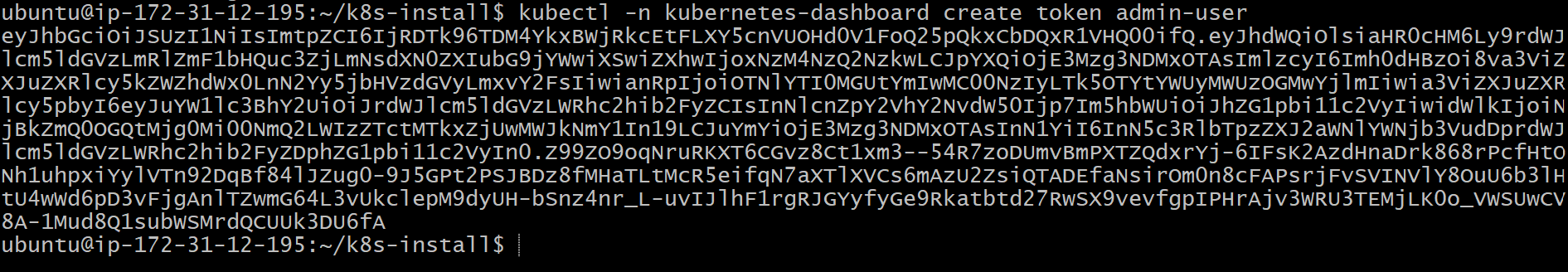
* Now, to create a token for accessing the Kubernetes Dashboard, run the command :

kubectl -n kubernetes-dashboard create token admin-user.

This will generate a token for the admin-user ServiceAccount within the kubernetes-dashboard namespace.

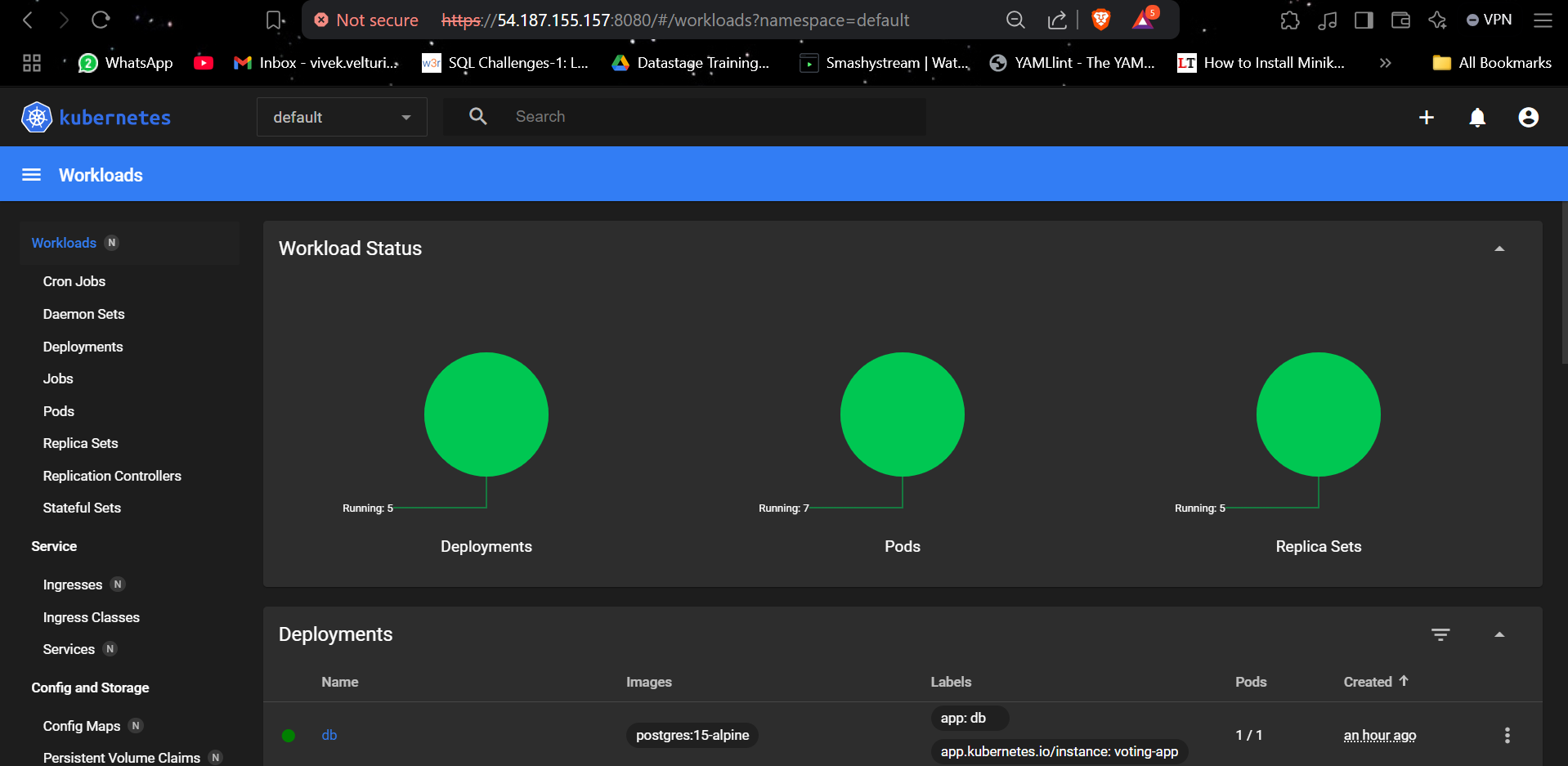
The token grants administrative access to the Kubernetes Dashboard as specified by the cluster-admin role binding.

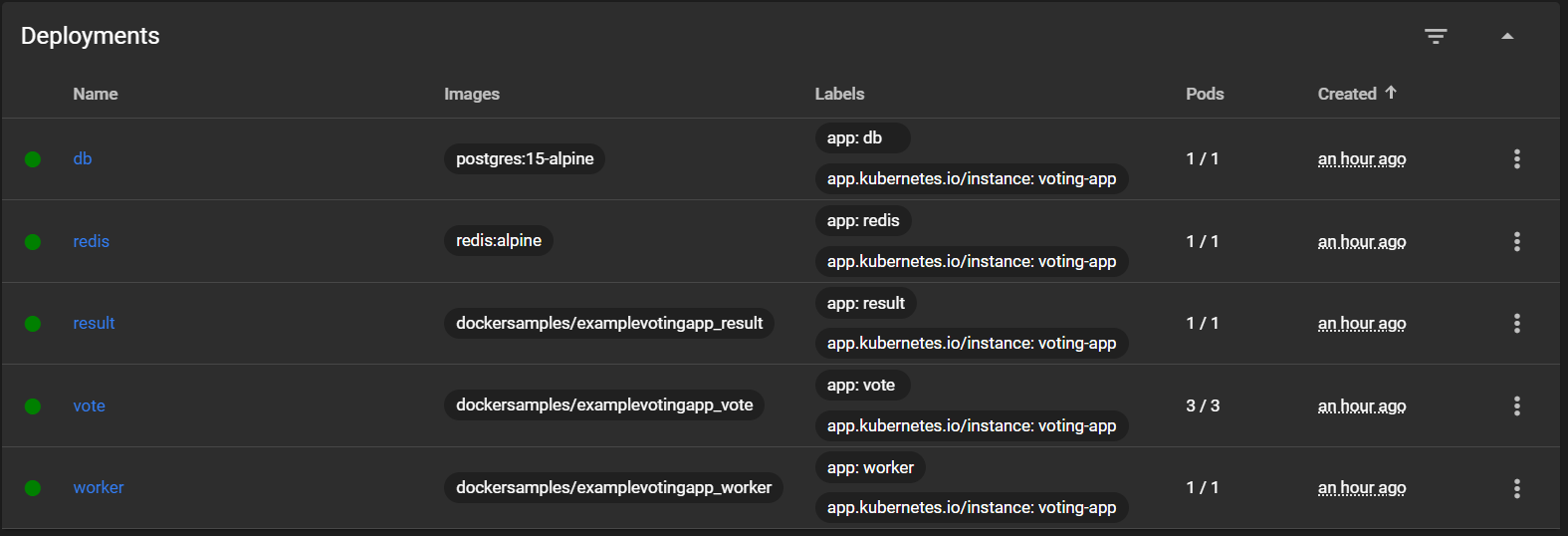
Once executed, the generated token will be displayed in the terminal, and you can use it to log in to the Dashboard with full administrative privileges.

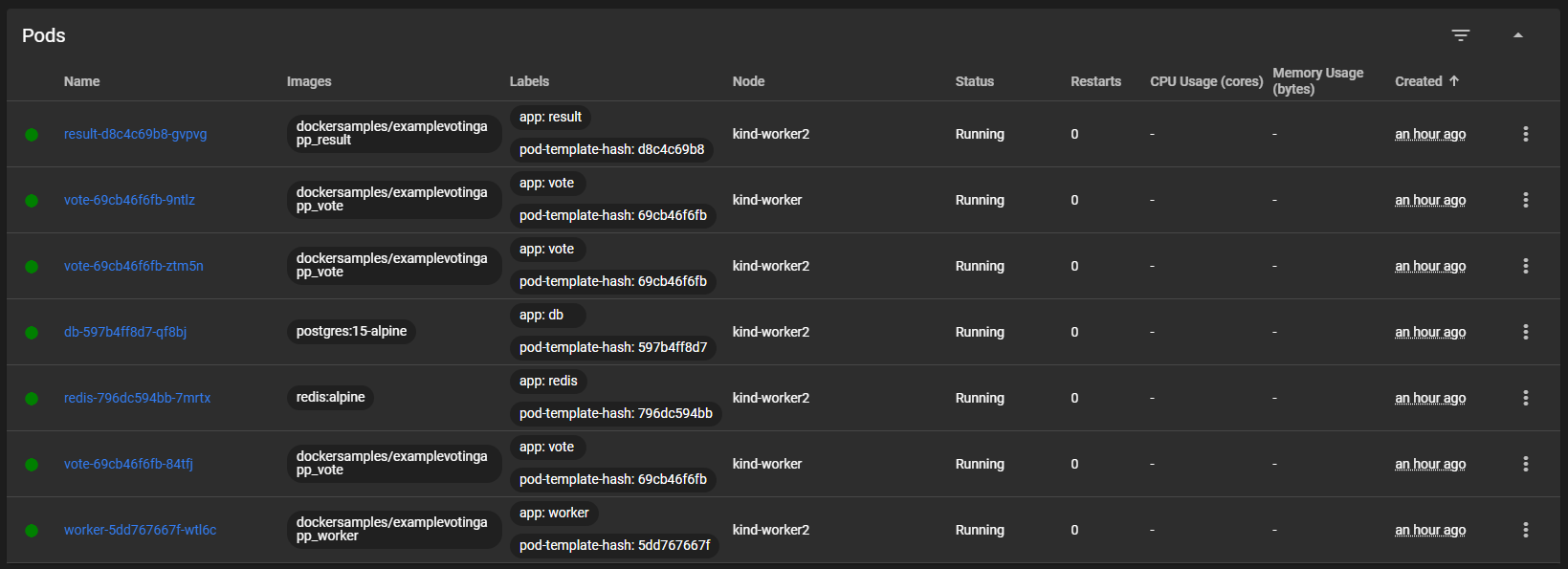


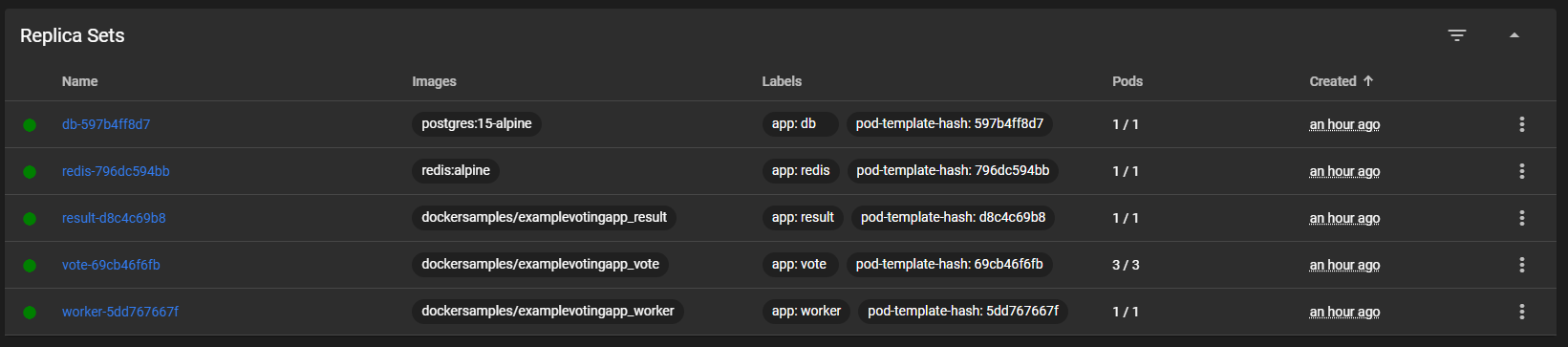
**Kubernetes Dashboard Overview**

* 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.









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.