Automated Deployment and Observability of Scalable Applications on AWS EC2 with Kubernetes, Prometheus, and Grafana

In this project, we focus on the automated deployment of scalable applications on AWS EC2 instances using Kubernetes, along with implementing robust observability practices through Prometheus and Grafana. The documentation outlines the step-by-step process of setting up a Kubernetes cluster on an EC2 instance, deploying containerized applications, and configuring monitoring tools to track the performance and health of the cluster. By integrating Prometheus for metrics collection and Grafana for data visualization, this project ensures effective observability, enabling proactive management and optimization of the deployed applications in a scalable cloud environment.

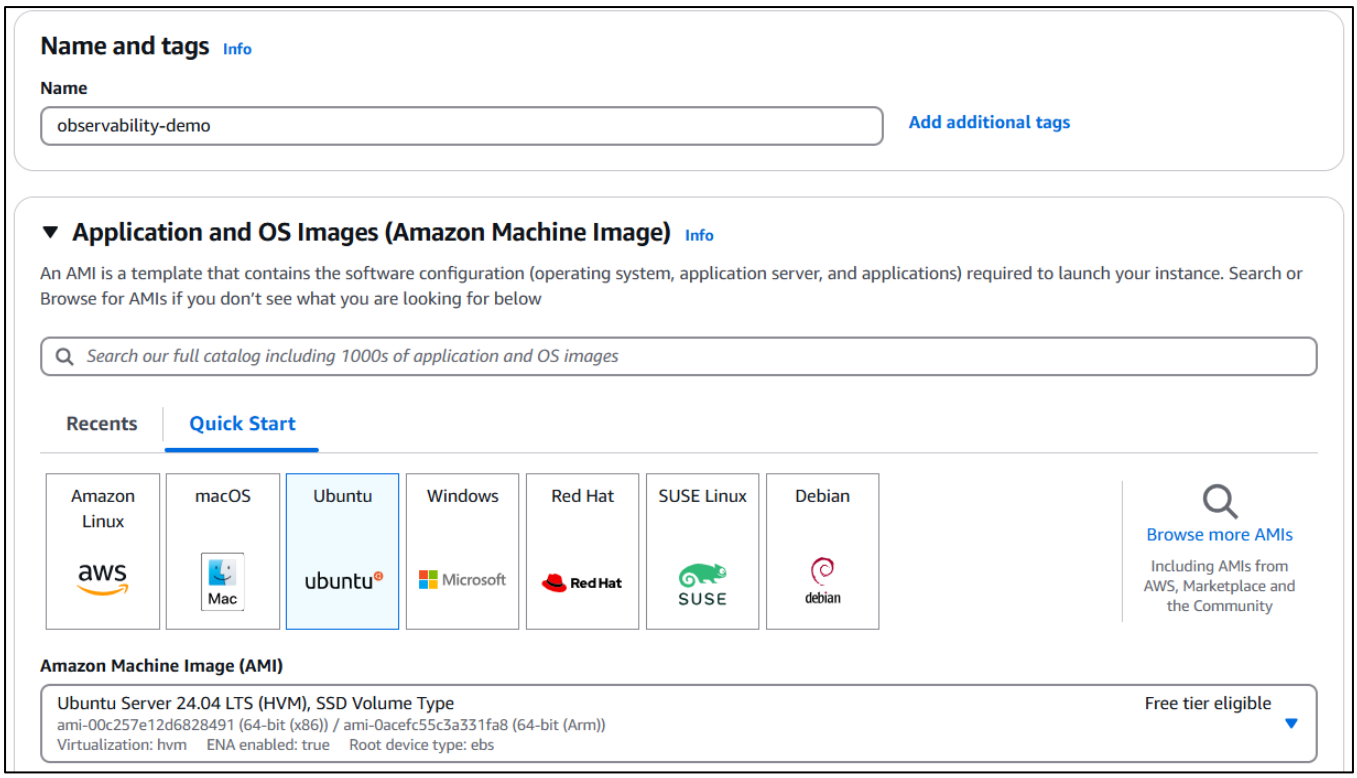
This project is similar to the previous one, where we deployed scalable applications on AWS EC2 using Kubernetes and Argo CD. However, in this project, we will focus on creating a Kubernetes cluster on an AWS EC2 instance and enhancing observability through Prometheus and Grafana. While the core setup of the Kubernetes cluster remains consistent, the key difference lies in implementing robust monitoring and visualization techniques to track the health, performance, and resource utilization of the applications and the cluster. This approach ensures better insights and proactive management of the deployed infrastructure.

Preliminary Steps: Preparing the Project from the External GitHub Repository

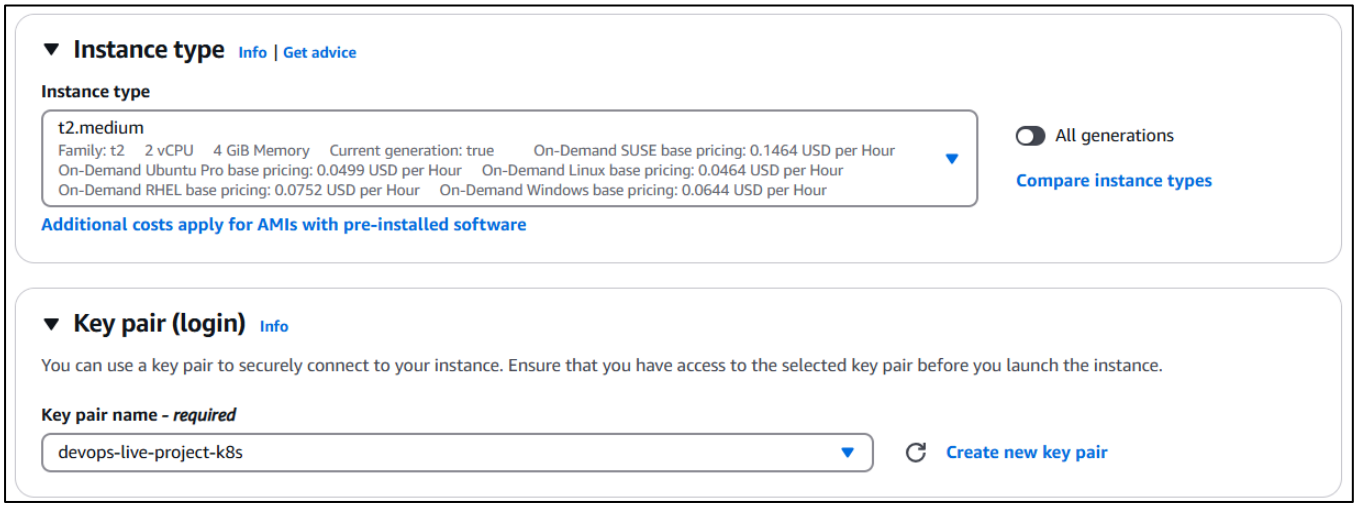
- Before starting with the deployment and other project-related tasks, you need to clone the project from the external GitHub repository available at **https://github.com/LondheShubham153/k8s-kind-voting-app.git**. This repository contains the application that we will deploy. After cloning it to your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

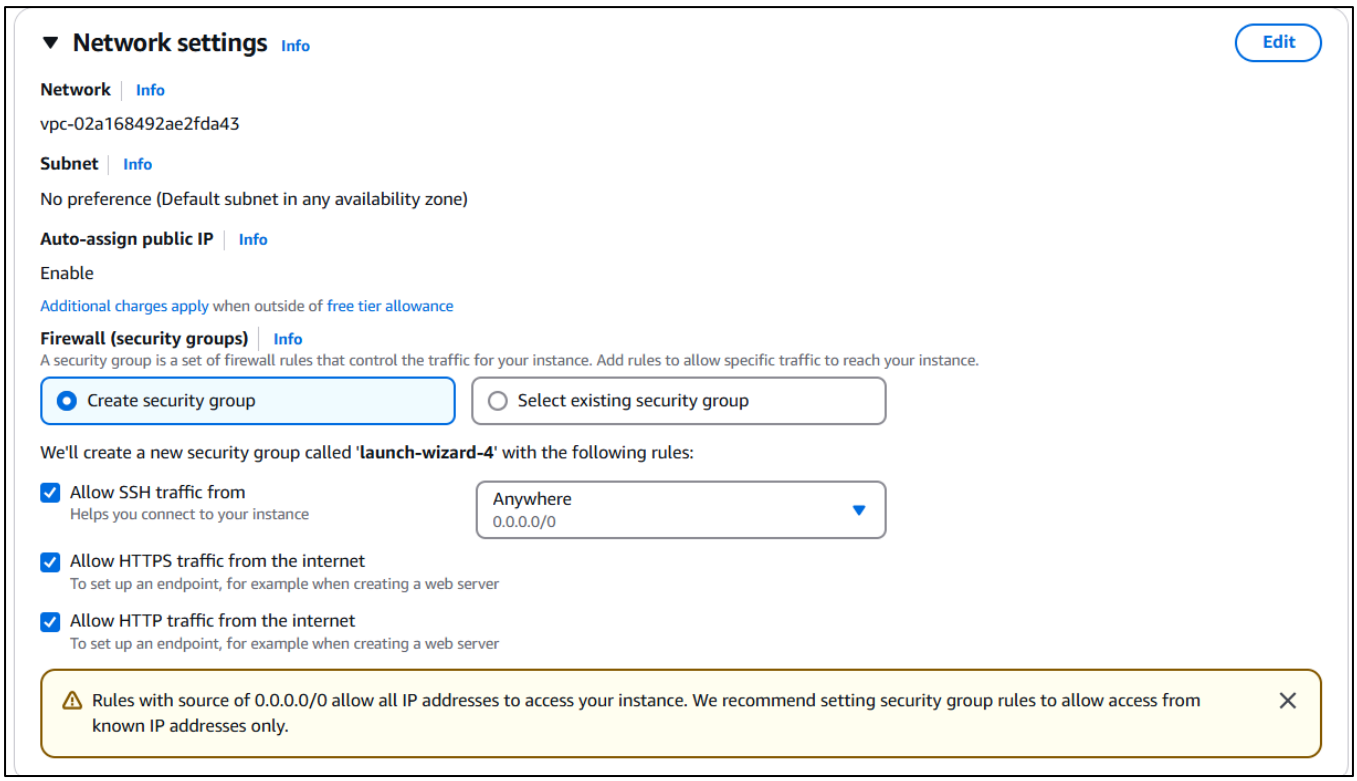
- **Enter the Name of the Instance**, eg: observability-demo
- Choose **Ubuntu Server 24.04 LTS (HVM)** under **Amazon Machine Image(AMI)**



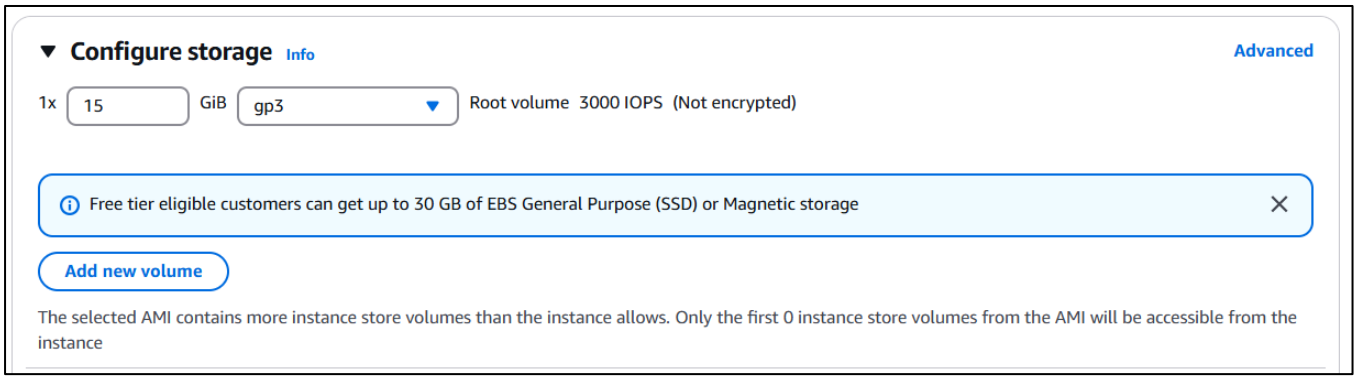
- Choose **t2.medium** under **Instance type**.
- Under **Key pair (login)**, give your key pair name or create a new key pair eg: devops-live-project-k8s is my keypair.



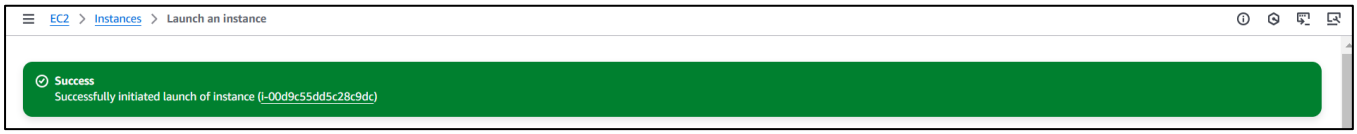
- In the network settings, choose the default VPC and subnet, and enable the option to auto-assign a public IP. For the firewall security groups, create a new security group and configure the following rules: allow **SSH** traffic from anywhere (port 22) to enable secure remote access, allow **HTTP**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

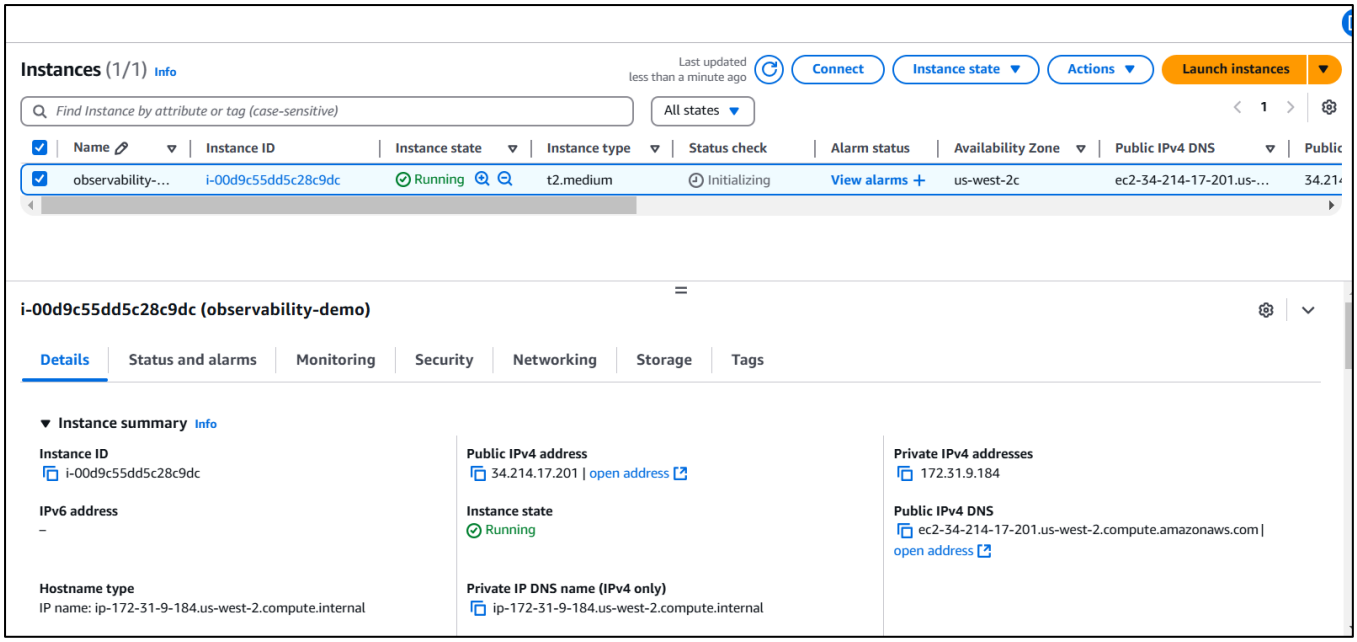


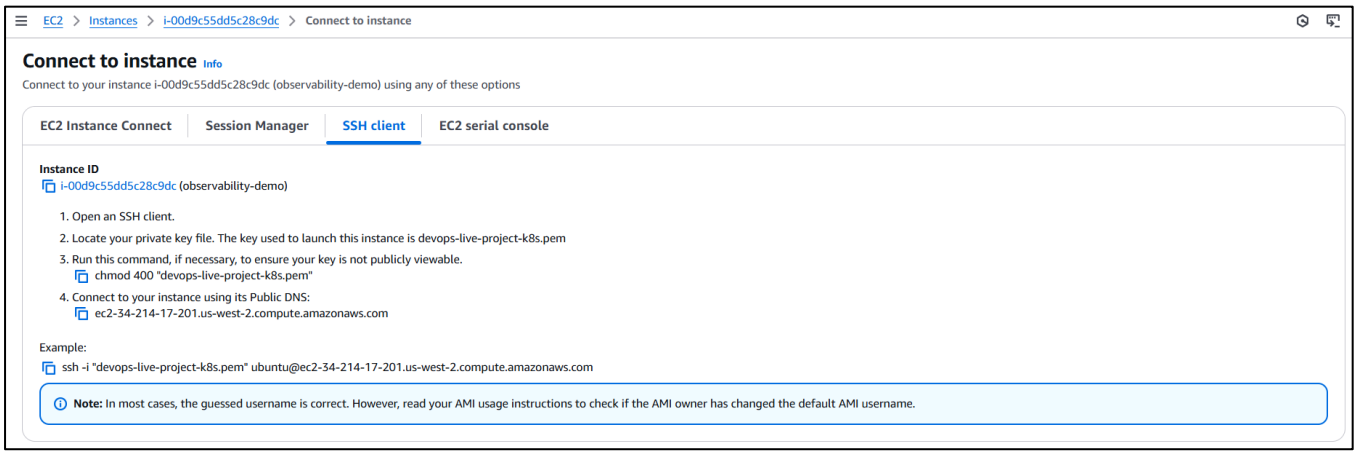
- Configure the storage settings by setting the root volume size to **15 GiB** to ensure sufficient space for the application and related dependencies.



- 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.

```
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
$
```

- 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;

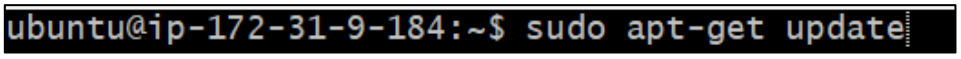
```
vivek@LAPTOP-2EFG8TEN MINGW64 /e/AWS/DevOps/Automated Deployment of Scalable App
lications on AWS EC2 with Kubernetes and Argo CD
$ ssh -i "devops-live-project-k8s.pem" ubuntu@ec2-34-214-17-201.us-west-2.comput
e.amazonaws.com
The authenticity of host 'ec2-34-214-17-201.us-west-2.compute.amazonaws.com (34.
214.17.201)' can't be established.
ED25519 key fingerprint is SHA256:Csxg+nFMWNOmoPqNcIzReYfcLn5qqN9hHo7yxL/BnfU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

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

```
Warning: Permanently added 'ec2-34-214-17-201.us-west-2.compute.amazonaws.com' (
ED25519) to the list of known hosts.
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 Thu Feb  6 06:00:26 UTC 2025

  System load:  0.07              Processes:             127
  Usage of /:   12.4% of 13.49GB  Users logged in:       0
  Memory usage: 5%                IPv4 address for enX0: 172.31.9.184
  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-172-31-9-184:~$
```

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`




```
Get:43 http://security.ubuntu.com/ubuntu noble-security/universe amd64 c-n-f Metadata [13.5 kB]
Get:44 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Packages [620 kB]
Get:45 http://security.ubuntu.com/ubuntu noble-security/restricted Translation-en [119 kB]
Get:46 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Components [212 B]
Get:47 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Packages [12.4 kB]
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 [212 B]
Get:50 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 c-n-f Metadata [356 B]
Fetched 32.1 MB in 12s (2772 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-9-184:~$
```

  2. **Install Docker** by running the following command:
     `sudo apt-get install docker.io`

```
ubuntu@ip-172-31-9-184:~$ 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 89 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]
```

When prompted with "**Do you want to continue? [Y/n]", type Y** and press Enter to proceed with the installation.

```
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-172-31-9-184:~$
```

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

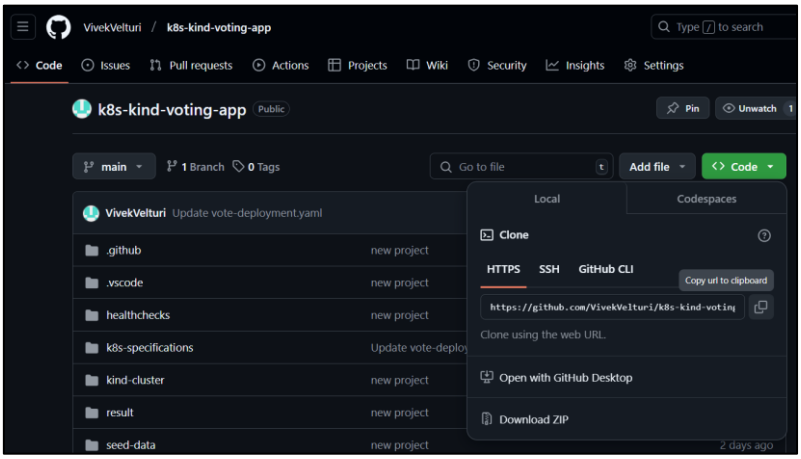
- You can check the containers running in Docker with the command `docker ps`. However, if you encounter the error **"permission denied"** while trying to connect to the Docker daemon at `unix:///var/run/docker.sock`, it means your user does not have the necessary permissions to access the Docker socket.

```
ubuntu@ip-172-31-9-184:~$ docker ps
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get "http://%2Fvar%2Frun%2Fdocker.sock/v1.45/containers/json": dial un
ix /var/run/docker.sock: connect: permission denied
ubuntu@ip-172-31-9-184:~$
```

- 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-172-31-9-184:~$ sudo usermod -aG docker $USER && newgrp docker
ubuntu@ip-172-31-9-184:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu@ip-172-31-9-184:~$
```

- Now that Docker is installed on our EC2 instance, the next step is to clone the GitHub repository into our instance. This will provide the necessary configuration files and deployment manifests required for setting up the Kubernetes cluster and implementing observability with Prometheus and Grafana.
- To proceed, we need to clone the GitHub repository that contains all the necessary configuration files and Kubernetes manifests for setting up our environment. In our case, the repository URL is https://github.com/VivekVelturi/k8s-kind-voting-app.git.



- To clone it into our EC2 instance, use the command: `git clone https://github.com/VivekVelturi/k8s-kind-voting-app.git`. This will download the repository contents, allowing us to access and utilize the required files for deploying and monitoring our Kubernetes cluster.

```
ubuntu@ip-172-31-9-184:~$ git clone https://github.com/VivekVelturi/k8s-kind-voting-app.git
Cloning into 'k8s-kind-voting-app'...
remote: Enumerating objects: 91, done.
remote: Counting objects: 100% (91/91), done.
remote: Compressing objects: 100% (74/74), done.
remote: Total 91 (delta 17), reused 80 (delta 10), pack-reused 0 (from 0)
Receiving objects: 100% (91/91), 1.59 MiB | 5.19 MiB/s, done.
Resolving deltas: 100% (17/17), done.
ubuntu@ip-172-31-9-184:~$
```

- After cloning the repository, we can verify that all the necessary files and scripts have been successfully copied into our EC2 instance. By executing the `ls` command, we can list the contents of the cloned repository and confirm that it includes the required configuration files, YAML manifests, and scripts needed for setting up our Kubernetes cluster and observability stack.

```
ubuntu@ip-172-31-9-184:~$ ls
k8s-kind-voting-app
ubuntu@ip-172-31-9-184:~$ cd k8s-kind-voting-app
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app$ ls
LICENSE      README.md                 docker-compose.images.yml  docker-stack.yml  healthchecks            k8s-specifications  prometheus.png  seed-data  worker
MAINTAINERS  architecture.excalidraw.png  docker-compose.yml         grafana.png       k8s-kind-voting-app.png  kind-cluster        result          vote
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app$
```

- Now, navigate into the `kind-cluster` folder within the cloned repository to access the necessary configuration and installation scripts. Inside this folder, you will find the `config.yml` file, which defines the cluster setup. You can view its contents using the command `cat config.yml`.

```
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app$ cd kind-cluster
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/kind-cluster$ ls
commands.md  config.yml  dashboard-adminuser.yml  install_kind.sh  install_kubectl.sh
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/kind-cluster$
```

```
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/kind-cluster$ cat 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
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/kind-cluster$
```

- However, before executing this file to create the Kubernetes nodes, we first need to install Kind. The executable shell script required for this installation, `install_kind.sh`, is also present in the same folder. We will execute this script to set up Kind before proceeding with the cluster creation.

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

```
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/kind-cluster$ chmod +x install_kind.sh
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/kind-cluster$
```

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

```
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/kind-cluster$ ./install_kind.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    97  100    97    0     0    707      0 --:--:-- --:--:-- --:--:--   708
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 6304k  100 6304k    0     0  2342k      0  0:00:02  0:00:02 --:--:-- 3069k
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/kind-cluster$
```

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

```yaml
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**.

- You can create a 3-node Kubernetes cluster using Kind by executing the command:
  `kind create cluster --config=config.yml -name=my-cluster`
  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-172-31-9-184:~/k8s-kind-voting-app/kind-cluster$ kind create cluster --config=config.yml --name=my-cluster
Creating cluster "my-cluster" ...
 ✓ 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-my-cluster"
You can now use your cluster with:

kubectl cluster-info --context kind-my-cluster

Thanks for using kind!
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/kind-cluster$
```

- 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:
- The repository we cloned also contains the executable shell script **`install_kubectl.sh`**, which is used to install `kubectl`.
- Before executing this script, we need to grant it the necessary execute permissions. This can be done using the command: `chmod +x install_kubectl.sh`.

```
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/kind-cluster$ ls
commands.md  config.yml  dashboard-adminuser.yml  install_kind.sh  install_kubectl.sh
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/kind-cluster$ chmod +x install_kubectl.sh
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/kind-cluster$
```

- Once the permissions are set, we can proceed with the installation by running `./install_kubectl.sh`. This will install `kubectl` on our EC2 instance, allowing us to interact with the Kubernetes cluster once it is set up.

```
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/kind-cluster$ ./install_kubectl.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   138  100   138    0     0   1372      0 --:--:-- --:--:-- --:--:--  1380
100 49.0M  100 49.0M    0     0  2078k      0  0:00:24  0:00:24 --:--:-- 2094k
Client Version: v1.30.0
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
kubectl installation complete.
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/kind-cluster$
```

- 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-172-31-9-184:~/k8s-kind-voting-app/kind-cluster$ kubectl get nodes
NAME                          STATUS   ROLES           AGE     VERSION
my-cluster-control-plane   Ready    control-plane   3m38s   v1.30.0
my-cluster-worker          Ready    <none>          3m18s   v1.30.0
my-cluster-worker2         Ready    <none>          3m19s   v1.30.0
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/kind-cluster$
```

- Now, if we navigate into the `k8s-specifications` folder within the cloned GitHub repository on our EC2 instance, we will find the Kubernetes deployment files for our application.

```
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/kind-cluster$ cd ..
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app$ ls
LICENSE      README.md                docker-compose.images.yml  docker-stack.yml  healthchecks            k8s-specifications  prometheus.png  seed-data  worker
MAINTAINERS  architecture.excalidraw.png  docker-compose.yml         grafana.png       k8s-kind-voting-app.png  kind-cluster        result          vote
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app$ cd  k8s-specifications
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/k8s-specifications$ ls
db-deployment.yaml  redis-deployment.yaml  result-deployment.yaml  vote-deployment.yaml  worker-deployment.yaml
db-service.yaml     redis-service.yaml     result-service.yaml     vote-service.yaml
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/k8s-specifications$
```

- Instead of applying each YAML file individually, we can deploy all the resources at once using a single command: `kubectl apply -f .`
  This command applies all the Kubernetes configurations present in the folder, setting up our application.

```
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/k8s-specifications$ kubectl apply -f .
deployment.apps/db created
service/db created
deployment.apps/redis created
service/redis created
deployment.apps/result created
service/result created
deployment.apps/vote created
service/vote created
deployment.apps/worker created
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/k8s-specifications$
```

- Now, apply the command `kubectl get all`, which will display all the Kubernetes resources currently running in the cluster. This includes:
  - **Pods**: All the running instances of the application components, such as `vote`, `result`, `worker`, `redis`, and `postgres`.
  - **Services**: The internal and external networking endpoints that allow communication between different components of the application.
  - **Deployments**: The controller managing the lifecycle of application pods, ensuring the desired number of replicas are running.
  - **ReplicaSets**: Maintaining the specified number of pod replicas for each deployment.
    This command provides a complete overview of the application's current state, helping to verify that all resources have been deployed correctly.

```
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/k8s-specifications$ kubectl get all
NAME                          READY   STATUS    RESTARTS   AGE
pod/db-597b4ff8d7-ss5t2       1/1     Running   0          41s
pod/redis-796dc594bb-xzt4p    1/1     Running   0          41s
pod/result-d8c4c69b8-4gtxs    1/1     Running   0          41s
pod/vote-69cb46f6fb-r45rw     1/1     Running   0          41s
pod/worker-5dd767667f-kfhtl   1/1     Running   0          41s

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/db           ClusterIP   10.96.240.107   <none>        5432/TCP         41s
service/kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP          6m53s
service/redis        ClusterIP   10.96.147.4     <none>        6379/TCP         41s
service/result       NodePort    10.96.165.98    <none>        5001:31001/TCP   41s
service/vote         NodePort    10.96.57.173    <none>        5000:31002/TCP   41s

NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/db       1/1     1            1           41s
deployment.apps/redis    1/1     1            1           41s
deployment.apps/result   1/1     1            1           41s
deployment.apps/vote     1/1     1            1           41s
deployment.apps/worker   1/1     1            1           41s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/db-597b4ff8d7       1         1         1       41s
replicaset.apps/redis-796dc594bb    1         1         1       41s
replicaset.apps/result-d8c4c69b8    1         1         1       41s
replicaset.apps/vote-69cb46f6fb     1         1         1       41s
replicaset.apps/worker-5dd767667f   1         1         1       41s
ubuntu@ip-172-31-9-184:~/k8s-kind-voting-app/k8s-specifications$
```

Installing and Configuring Helm

- Now, we need to install **Helm** on our EC2 instance. **Helm** is a package manager for Kubernetes that simplifies the deployment and management of applications using **Helm charts**. Instead of manually creating and managing Kubernetes manifests, Helm allows us to deploy complex applications with predefined configurations efficiently.

  To install Helm, we first download the Helm installation script using the following command:

  `curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3`

  This command fetches the **Helm installation script** from the official Helm GitHub repository and saves it as `get_helm.sh`. Next, we will make this script executable and run it to install Helm.

```
ubuntu@ip-172-31-9-184:~$ curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
ubuntu@ip-172-31-9-184:~$
```

- Now that we have downloaded the Helm installation script, we need to grant it execution permissions before running it. This is done using the command:
  `chmod 700 get_helm.sh`,
  it allows only the current user to read, write, and execute the script while restricting access for others. This ensures a secure installation process. Once the necessary permissions are set, we can proceed with executing the script to install Helm on our EC2 instance.

```
ubuntu@ip-172-31-9-184:~$ chmod 700 get_helm.sh
ubuntu@ip-172-31-9-184:~$
```

- we can now proceed by executing the script with `./get_helm.sh`.
  This command runs the script, which will automatically download and install Helm on our EC2 instance. Helm is essential for managing Kubernetes applications, as it simplifies the deployment and management of applications using Helm charts.

```
ubuntu@ip-172-31-9-184:~$ ./get_helm.sh
Downloading https://get.helm.sh/helm-v3.17.0-linux-amd64.tar.gz
Verifying checksum... Done.
Preparing to install helm into /usr/local/bin
helm installed into /usr/local/bin/helm
ubuntu@ip-172-31-9-184:~$
```

- Next, we added the Prometheus Helm chart repository to our Helm setup using the command: `helm repo add prometheus-community https://prometheus-community.github.io/helm-charts`.
  This step is crucial because it allows us to access and install the Prometheus chart from the Prometheus community repository. Helm charts simplify the process of deploying and managing Prometheus in our Kubernetes environment, enabling efficient monitoring and observability of our applications.

```
ubuntu@ip-172-31-9-184:~$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
"prometheus-community" has been added to your repositories
ubuntu@ip-172-31-9-184:~$
```

- we can verify its addition by executing the command `helm repo list`.
  This command will show the list of repositories, and we will see the Prometheus Community repository listed among them.

```
ubuntu@ip-172-31-9-184:~$ helm repo list
NAME                    URL
prometheus-community    https://prometheus-community.github.io/helm-charts
ubuntu@ip-172-31-9-184:~$
```

- Next, we added the stable Helm chart repository using the command: `helm repo add stable https://charts.helm.sh/stable`.
  This command allows us to access a wide range of pre-packaged, stable Helm charts for various applications, including Grafana, Nginx, and more. By adding this repository, we ensure that we can install and manage stable, production-ready applications within our Kubernetes cluster using Helm.

```
ubuntu@ip-172-31-9-184:~$ helm repo add stable https://charts.helm.sh/stable
"stable" has been added to your repositories
ubuntu@ip-172-31-9-184:~$
```

- To verify the addition of the stable repository, we ran the command `helm repo list`, which displays the list of Helm repositories, including the newly added stable repository.

```
ubuntu@ip-172-31-9-184:~$ helm repo list
NAME                    URL
prometheus-community    https://prometheus-community.github.io/helm-charts
stable                  https://charts.helm.sh/stable
ubuntu@ip-172-31-9-184:~$
```

- Next, execute the command `helm repo update`.
  This command updates the Helm repositories, ensuring that the latest charts and applications from the repositories added earlier are available. Running this update ensures that Helm pulls the most recent versions of the applications and their dependencies, keeping the environment up-to-date.

```
ubuntu@ip-172-31-9-184:~$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "stable" chart repository
Update Complete. ⎈Happy Helming!⎈
ubuntu@ip-172-31-9-184:~$
```

- Next, create a new namespace for monitoring using the command:
  `kubectl create namespace monitoring`.
  This namespace will be used to deploy the Prometheus and Grafana components, helping to organize and isolate monitoring-related resources within your Kubernetes cluster.

```
ubuntu@ip-172-31-9-184:~$ kubectl create namespace monitoring
namespace/monitoring created
ubuntu@ip-172-31-9-184:~$
```

- Execute the command:

```
helm install kind-prometheus prometheus-community/kube-prometheus-stack --namespace monitoring --set prometheus.service.nodePort=30000 --set prometheus.service.type=NodePort --set grafana.service.nodePort=31000 --set grafana.service.type=NodePort --set alertmanager.service.nodePort=32000 --set alertmanager.service.type=NodePort --set prometheus-node-exporter.service.nodePort=32001 --set prometheus-node-exporter.service.type=NodePort.
```

  This command installs the Prometheus and Grafana stack using Helm in the `monitoring` namespace. It deploys several monitoring components, including Prometheus, Grafana, Alertmanager, and Prometheus Node Exporter, within your Kubernetes cluster.

Here's a breakdown of the command:

- `kind-prometheus`: Specifies the name of the Helm release.
- `prometheus-community/kube-prometheus-stack`: Refers to the Helm chart from the Prometheus community, which contains all the necessary configurations for installing Prometheus, Grafana, Alertmanager, and other monitoring components.

- `--namespace monitoring`:
  Deploys the stack in the `monitoring` namespace, ensuring that the resources are logically grouped and isolated.
- `--set prometheus.service.nodePort=30000 --set prometheus.service.type=NodePort`:
  Configures Prometheus to be accessible via NodePort on port 30000.
- `--set grafana.service.nodePort=31000 --set grafana.service.type=NodePort`:
  Configures Grafana to be accessible via NodePort on port 31000.
- `--set alertmanager.service.nodePort=32000 --set alertmanager.service.type=NodePort`:
  Configures Alertmanager to be accessible via NodePort on port 32000.
- `--set prometheus-node-exporter.service.nodePort=32001 --set prometheus-node-exporter.service.type=NodePort`:
  Configures the Prometheus Node Exporter to be accessible via NodePort on port 32001.

By setting the `NodePort` for these services, you can access the Prometheus, Grafana, Alertmanager, and Node Exporter dashboards externally using the public IP of your EC2 instance and the corresponding ports.

```
ubuntu@ip-172-31-9-184:~$ helm install kind-prometheus prometheus-community/kube-prometheus-stack --namespace monitoring --set prometheus.service.nodePort=30000 --set prometheus.service.type=NodePort --set grafana.service.nodePort=31000 --set grafana.service.type=NodePort --set alertmanager.service.nodePort=32000 --set alertmanager.service.type=NodePort --set prometheus-node-exporter.service.nodePort=32001 --set prometheus-node-exporter.service.type=NodePort
NAME: kind-prometheus
LAST DEPLOYED: Thu Feb  6 06:29:17 2025
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace monitoring get pods -l "release=kind-prometheus"

Get Grafana 'admin' user password by running:

  kubectl --namespace monitoring get secrets prom-grafana -ojsonpath="{.data.admin-password}" | base64 -d ; echo

Access Grafana local instance:

  export POD_NAME=$(kubectl --namespace monitoring get pod -l "app.kubernetes.io/name=grafana,app.kubernetes.io/instance=kind-prometheus" -oname)
  kubectl --namespace monitoring port-forward $POD_NAME 3000

Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create & configure Alertmanager and Prometheus instances using the Operator.
ubuntu@ip-172-31-9-184:~$
```

- Now,running `kubectl get pods -n monitoring` will display the pods in the `monitoring` namespace, which include the Prometheus, Grafana, Alertmanager, and Prometheus Node Exporter pods. These pods are created as part of the Kube-Prometheus-Stack deployment, and their status will indicate whether they are running successfully.

```
ubuntu@ip-172-31-9-184:~$ kubectl get pods -n monitoring
NAME                                                     READY   STATUS    RESTARTS   AGE
alertmanager-kind-prometheus-kube-prome-alertmanager-0   2/2     Running   0          56s
kind-prometheus-grafana-9bf6d8569-wsb62                  3/3     Running   0          65s
kind-prometheus-kube-prome-operator-746664db74-blgtq     1/1     Running   0          65s
kind-prometheus-kube-state-metrics-5db69c8d6-8hhkl       1/1     Running   0          65s
kind-prometheus-prometheus-node-exporter-6q5rz           1/1     Running   0          65s
kind-prometheus-prometheus-node-exporter-dc4cd           1/1     Running   0          65s
kind-prometheus-prometheus-node-exporter-f87c4           1/1     Running   0          65s
prometheus-kind-prometheus-kube-prome-prometheus-0       2/2     Running   0          55s
ubuntu@ip-172-31-9-184:~$
```

- Following this, executing `kubectl get svc -n monitoring` will show the services for Prometheus, Grafana, Alertmanager, and Prometheus Node Exporter, all of which are exposed via NodePort. The output will display the service names along with their

types (NodePort), Cluster IPs, external IPs (if applicable), and the assigned NodePorts for accessing the services externally.

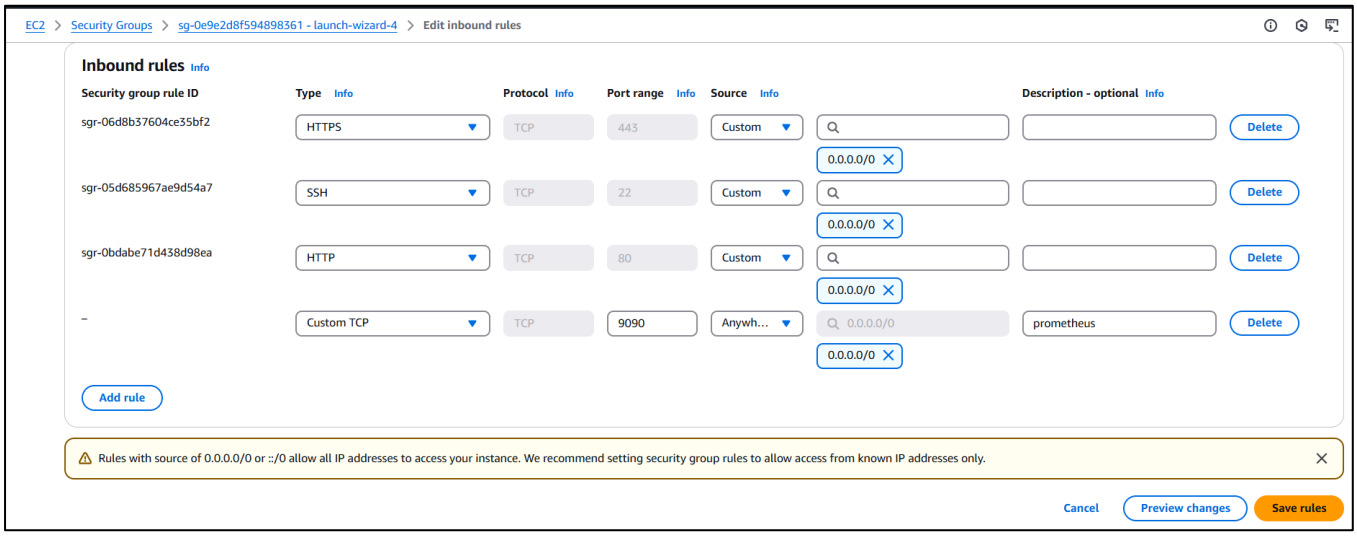
```
ubuntu@ip-172-31-9-184:~$ kubectl get svc -n monitoring
NAME                                      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                        AGE
alertmanager-operated                     ClusterIP   None             <none>        9093/TCP,9094/TCP,9094/UDP     2m4s
kind-prometheus-grafana                   NodePort    10.96.93.251     <none>        80:31000/TCP                   2m13s
kind-prometheus-kube-prome-alertmanager   NodePort    10.96.26.2       <none>        9093:32000/TCP,8080:31489/TCP  2m13s
kind-prometheus-kube-prome-operator       ClusterIP   10.96.23.32      <none>        443/TCP                        2m13s
kind-prometheus-kube-prome-prometheus     NodePort    10.96.83.194     <none>        9090:30000/TCP,8080:30477/TCP  2m13s
kind-prometheus-kube-state-metrics        ClusterIP   10.96.102.164    <none>        8080/TCP                       2m13s
kind-prometheus-prometheus-node-exporter  NodePort    10.96.92.253     <none>        9100:32001/TCP                 2m13s
prometheus-operated                       ClusterIP   None             <none>        9090/TCP                       2m3s
ubuntu@ip-172-31-9-184:~$
```

Deploying Prometheus for Monitoring

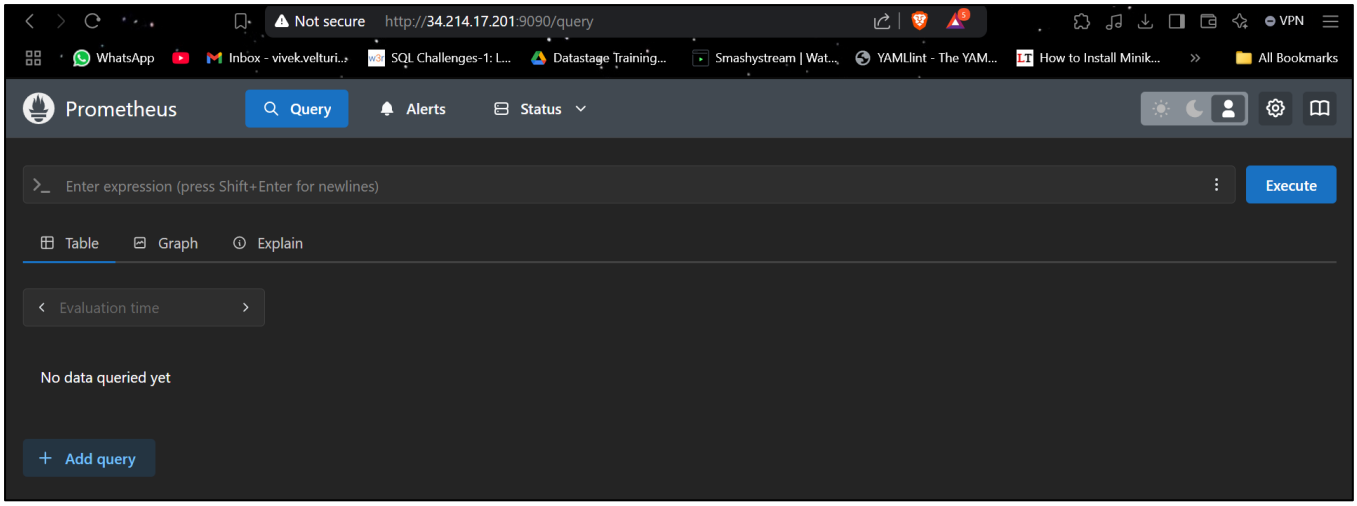
- After obtaining the service details with the `kubectl get svc -n monitoring` command, the next step is to port-forward the Prometheus service to access the Prometheus dashboard. This can be done by executing the following command:
  `kubectl port-forward svc/kind-prometheus-kube-prome-prometheus -n monitoring 9090:9090 --address=0.0.0.0 &`
  This command forwards port 9090 of the Prometheus service to port 9090 on the local machine, allowing external access to Prometheus via the public IP of the EC2 instance.

```
ubuntu@ip-172-31-9-184:~$ kubectl port-forward svc/kind-prometheus-kube-prome-prometheus -n monitoring 9090:9090 --address=0.0.0.0 &
[1] 9615
ubuntu@ip-172-31-9-184:~$ Forwarding from 0.0.0.0:9090 -> 9090
```

- Next, to make Prometheus accessible over the internet, navigate to the security group of your EC2 instance. Modify the inbound rules to include port 9090.

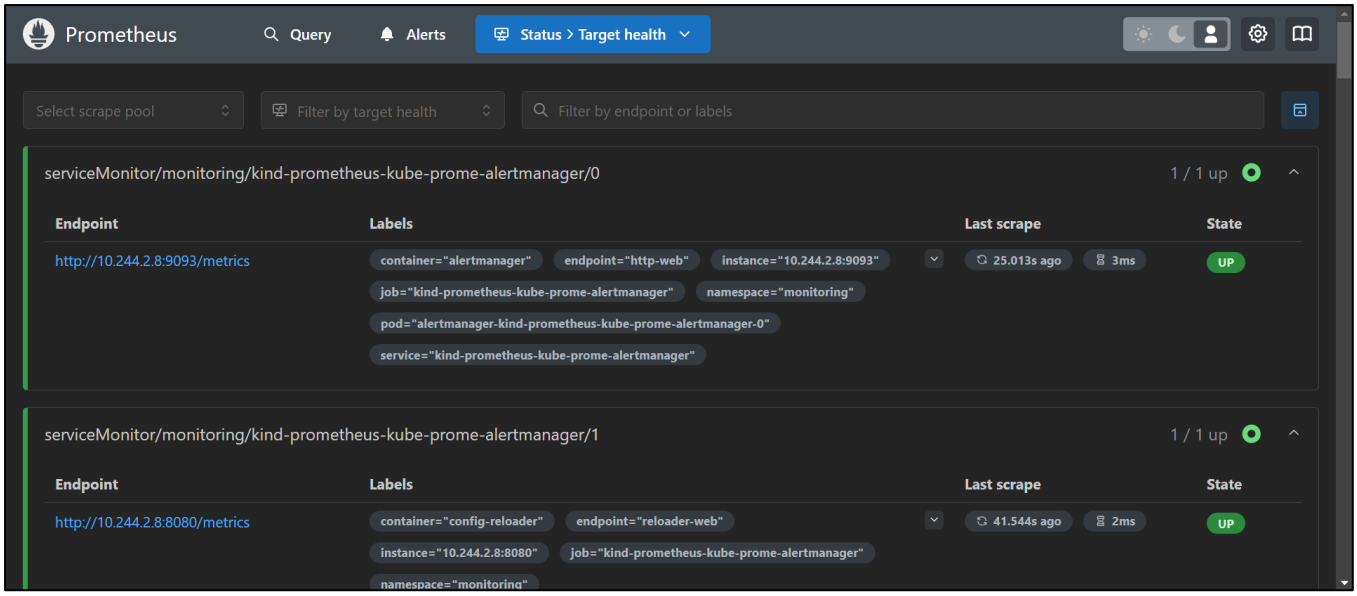


- Once the rule is updated, you can access the Prometheus dashboard by opening your browser and entering the public IP of your instance, followed by the port number `9090`. This provides access to the Prometheus monitoring interface for visualizing metrics and monitoring your Kubernetes cluster.



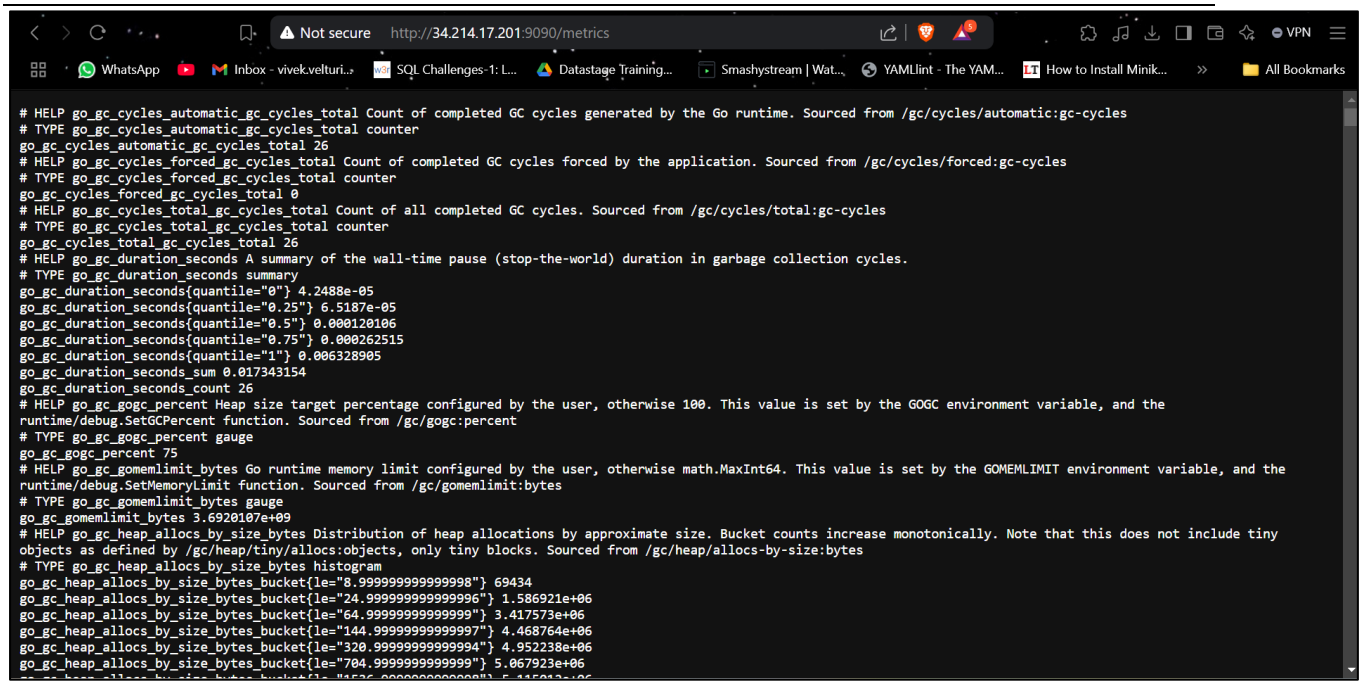
- Once you've accessed the Prometheus dashboard by navigating to `http://<your-public-ip>:9090`, (e.g., `http://34.214.17.201:9090`)in your browser, you can monitor the health of your targets by going to the **Status** and **Targets** pages in Prometheus.
  Here's how to do it:
  1. **Navigate to the "Status" Menu**:
     - On the Prometheus dashboard, you will find a navigation menu at the top of the page. Click on the **"Status"** dropdown menu.
  2. **Select "Targets"**:
     - From the **"Status"** menu, click on **"Targets"**. This page shows a list of all the targets Prometheus is scraping for metrics. These targets represent the services or applications that Prometheus is monitoring.
  3. **View Target Health**:
     - On the **Targets** page, you will see several columns such as **"Target"**, **"Last Scrape"**, **"Health"**, and **"Discovered Labels"**.
       - **Target**: Displays the name or address of the service being monitored.
       - **Last Scrape**: Indicates the last time Prometheus successfully scraped metrics from the target.
       - **Health**: Displays the health status of the target. The health status could be **"up"** (indicating the target is healthy) or **"down"** (indicating that Prometheus failed to scrape data from the target).
       - **Discovered Labels**: Shows additional labels related to the service or target.
     - In the **"Health"** column, if the status is **"up"**, the target is healthy and being actively monitored. If the status is **"down"**, this indicates an issue, and Prometheus is not able to scrape metrics from that target.

By regularly checking the **Targets** page in Prometheus, you can ensure that all your monitored services are healthy and that Prometheus is correctly scraping the metrics needed for observability. If any target is down, you can take necessary actions to troubleshoot and resolve the issue.



- When you append `/metrics` at the end of the IP address along with the port number (e.g., `http://34.214.17.201:9090/metrics`), you access the raw metrics data that Prometheus is collecting from the target services.
  This is the format where Prometheus exposes all the collected metrics in a plain text format. These metrics include data such as the number of requests handled by your application, memory usage, CPU usage, and more. Prometheus scrapes this data from your application or services based on the configuration you have set up.

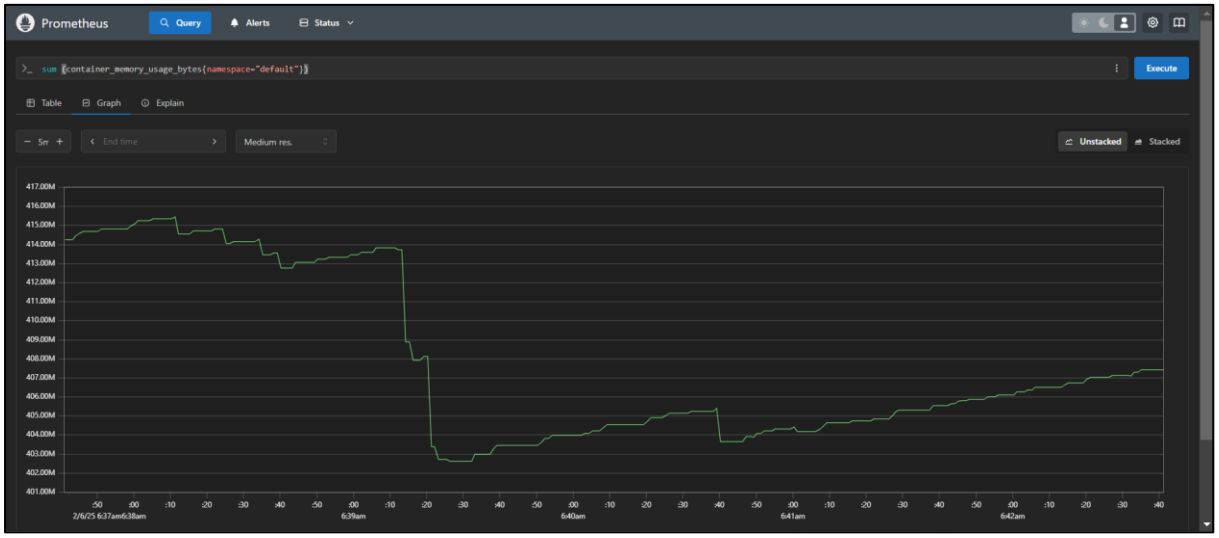
  In simple terms, accessing `http://<your-public-ip>:9090/metrics` allows you to view detailed performance and health statistics about the services being monitored by Prometheus in a machine-readable format. This data can be used for further analysis, alerting, and monitoring purposes.



- In our Prometheus window, we can run the following queries to monitor various resource usages within the Kubernetes cluster:

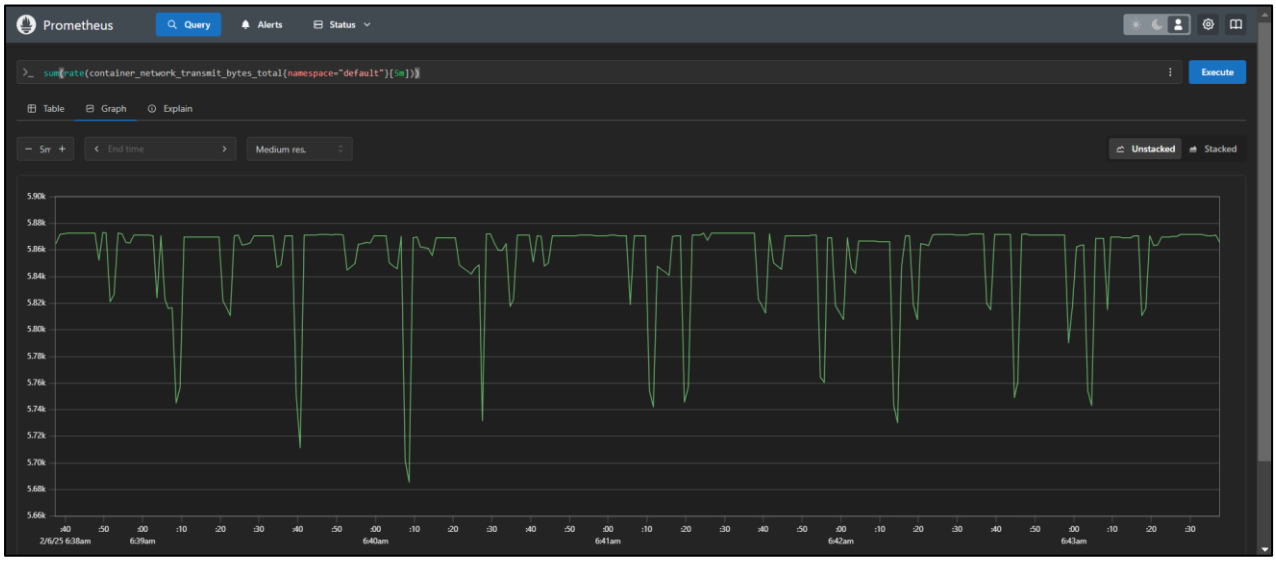
1. **CPU Usage Percentage:**
    - **Query:**
      `sum(rate(container_cpu_usage_seconds_total{namespace="default"}[1m])) / sum(machine_cpu_cores) * 100`
    - **Explanation:**
      This query calculates the CPU usage percentage for containers in the "default" namespace. It measures the rate of CPU usage over a 1-minute interval and compares it to the available CPU cores on the machine, giving an overview of overall CPU utilization.

2. **Memory Usage by Pod:**
    - Query: `sum(container_memory_usage_bytes{namespace="default"}) by (pod)`
    - **Explanation:**
      This query provides the total memory usage for each pod in the "default" namespace. It sums the memory usage across all containers within a pod, allowing us to monitor memory consumption for individual pods.



3. **Incoming Network Traffic by Pod:**
    - Query:
      ```
      sum(rate(container_network_receive_bytes_total{namespace="default"}[5m])) by (pod)
      ```
    - **Explanation:**
      This query calculates the total incoming network traffic (in bytes) for containers within the "default" namespace, grouped by pod. It helps in tracking incoming traffic trends and detecting any pods receiving unusually high levels of network traffic.

4. **Outgoing Network Traffic by Pod:**
    - Query:
      ```
      sum(rate(container_network_transmit_bytes_total{namespace="default"}[5m])) by (pod)
      ```
    - **Explanation:**
      This query calculates the total outgoing network traffic (in bytes) for containers within the "default" namespace, grouped by pod. It gives insights into the network traffic being sent from the pods, helping monitor outbound data usage.



These queries can be executed within the Prometheus window, providing valuable insights into CPU, memory, and network resource utilization, helping to monitor the performance and health of applications running in your Kubernetes cluster.

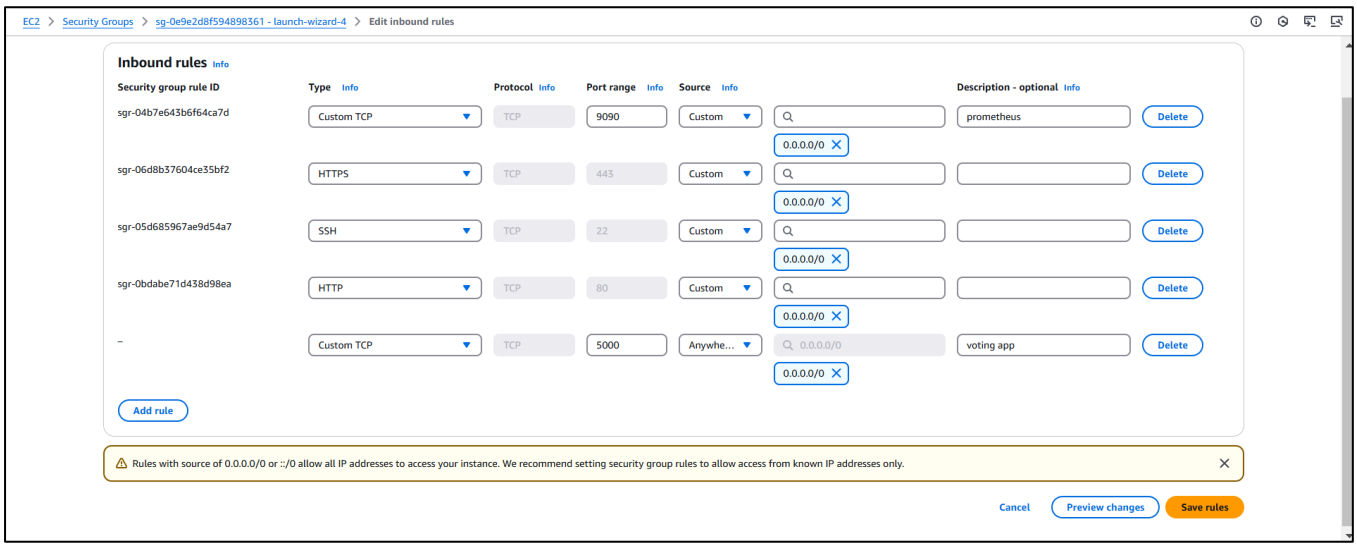
- Now, to monitor the network traffic for our application, we first need to identify the ports on which our application is running. Since we deployed our application in the default namespace, we can use the command `kubectl get svc` to view the services and their associated ports.

```
ubuntu@ip-172-31-9-184:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
db           ClusterIP   10.96.240.107   <none>        5432/TCP         25m
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP          31m
redis        ClusterIP   10.96.147.4     <none>        6379/TCP         25m
result       NodePort    10.96.165.98    <none>        5001:31001/TCP   25m
vote         NodePort    10.96.57.173    <none>        5000:31002/TCP   25m
ubuntu@ip-172-31-9-184:~$
```

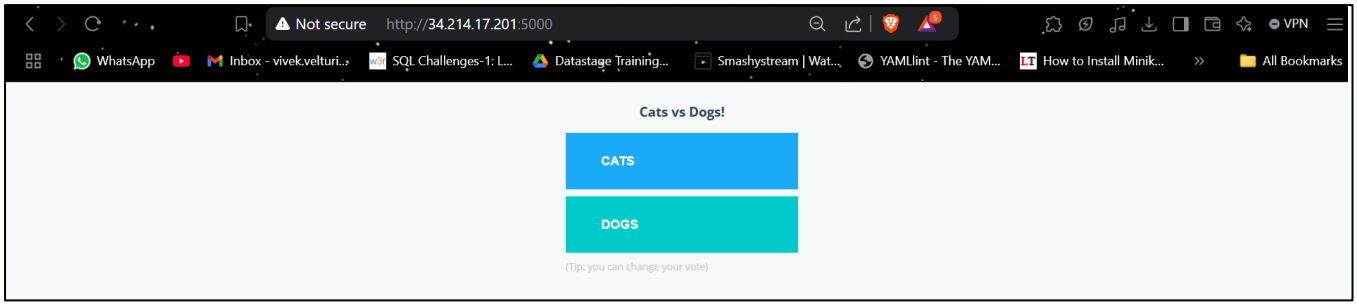
- Next, we need to carry out port forwarding for the vote service so that it can be accessed from outside the Kubernetes cluster. You can use the command:
`kubectl port-forward svc/vote 5000:5000 --address=0.0.0.0 &`

```
ubuntu@ip-172-31-9-184:~$ kubectl port-forward svc/vote 5000:5000 --address=0.0.0.0 &
[2] 10094
ubuntu@ip-172-31-9-184:~$ Forwarding from 0.0.0.0:5000 -> 80
```

- After executing the port forwarding command, go to the security group settings of the EC2 instance and add an inbound rule for port 5000, allowing external access to the application.



- Once the port forwarding is set up and the security group is updated, you can open your browser and access the voting application using the EC2 instance's **public IP** followed by the port number (e.g., `http:// 34.214.17.201:5000`).



- Now, to observe the effect of traffic on the network, run the **Prometheus query for incoming traffic** again:
  `sum(rate(container_network_receive_bytes_total{namespace="default"}[5m])) by (pod)`

Before accessing the application, the network traffic metrics show a baseline level of incoming traffic. No significant activity is observed.

After accessing the application and initiating port forwarding, the network traffic metrics show an increase in incoming traffic, reflecting the interaction with the voting application

Setting Up Grafana for Visualization:

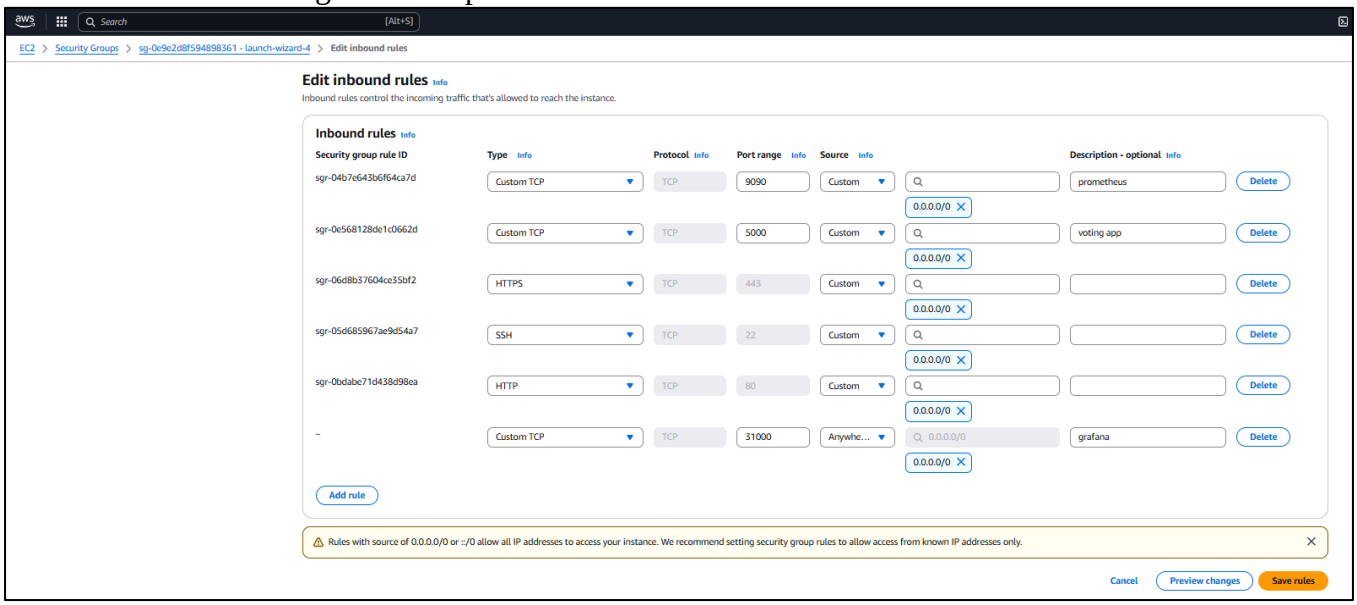
- Now, to set up Grafana, we first need to determine the ports Grafana is allocated to. We can do this by running the command `kubectl get svc -n monitoring`.

```
ubuntu@ip-172-31-9-184:~$ kubectl get svc -n monitoring
NAME                                      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                       AGE
alertmanager-operated                     ClusterIP   None            <none>        9093/TCP,9094/TCP,9094/UDP    22m
kind-prometheus-grafana                   NodePort    10.96.93.251    <none>        80:31000/TCP                  22m
kind-prometheus-kube-prome-alertmanager   NodePort    10.96.26.2      <none>        9093:32000/TCP,8080:31489/TCP 22m
kind-prometheus-kube-prome-operator       ClusterIP   10.96.23.32     <none>        443/TCP                       22m
kind-prometheus-kube-prome-prometheus     NodePort    10.96.83.194    <none>        9090:30000/TCP,8080:30477/TCP 22m
kind-prometheus-kube-state-metrics        ClusterIP   10.96.102.164   <none>        8080/TCP                      22m
kind-prometheus-prometheus-node-exporter  NodePort    10.96.92.253    <none>        9100:32001/TCP                22m
prometheus-operated                       ClusterIP   None            <none>        9090/TCP                      22m
```

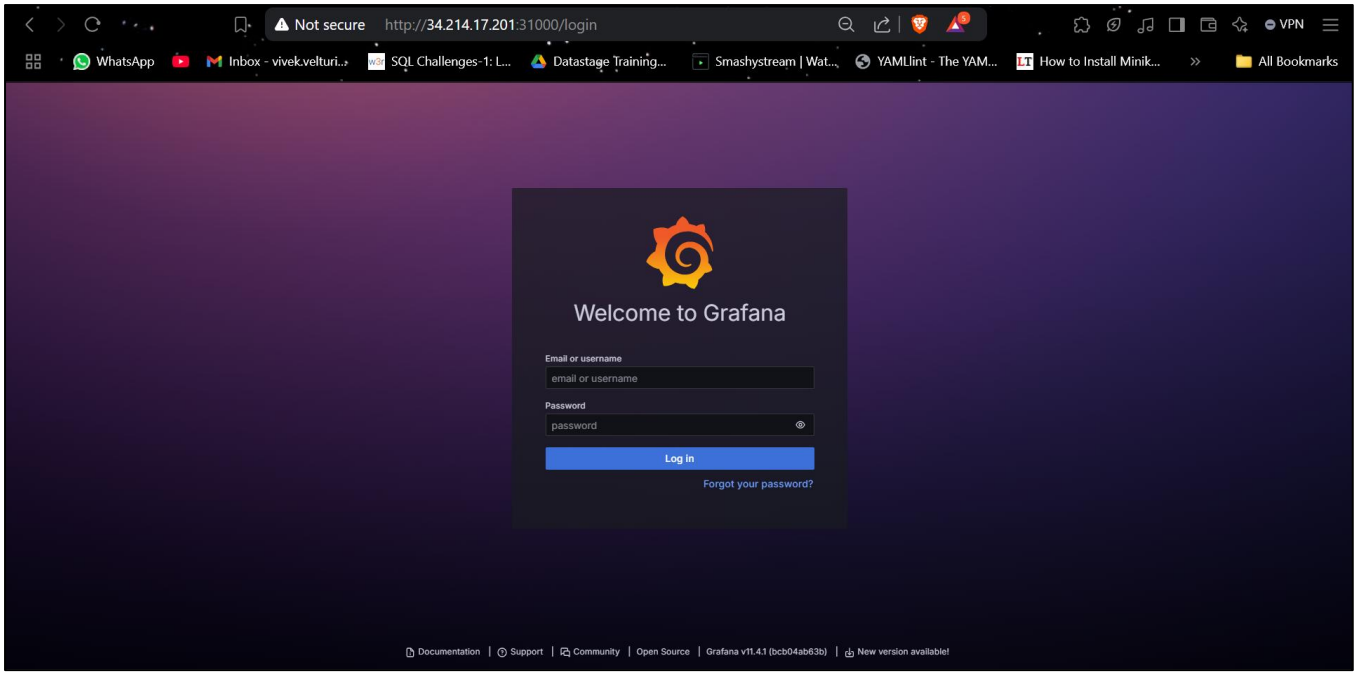
- Once we identify the correct port, we will use the `kubectl port-forward` command to forward the port to the local machine. In this case, we forward Grafana's port with the following command:
  `kubectl port-forward svc/kind-prometheus-grafana -n monitoring 31000:80 --address=0.0.0.0 &`

```
ubuntu@ip-172-31-9-184:~$ kubectl port-forward svc/kind-prometheus-grafana -n monitoring 31000:80 --address=0.0.0.0 &
[3] 10316
ubuntu@ip-172-31-9-184:~$ Forwarding from 0.0.0.0:31000 -> 3000
```

- To access the Grafana dashboard, we need to edit the EC2 instance's security group inbound rules. Go to the EC2 console, select the instance, and modify the inbound rules to allow incoming traffic on port **31000** for **TCP**.

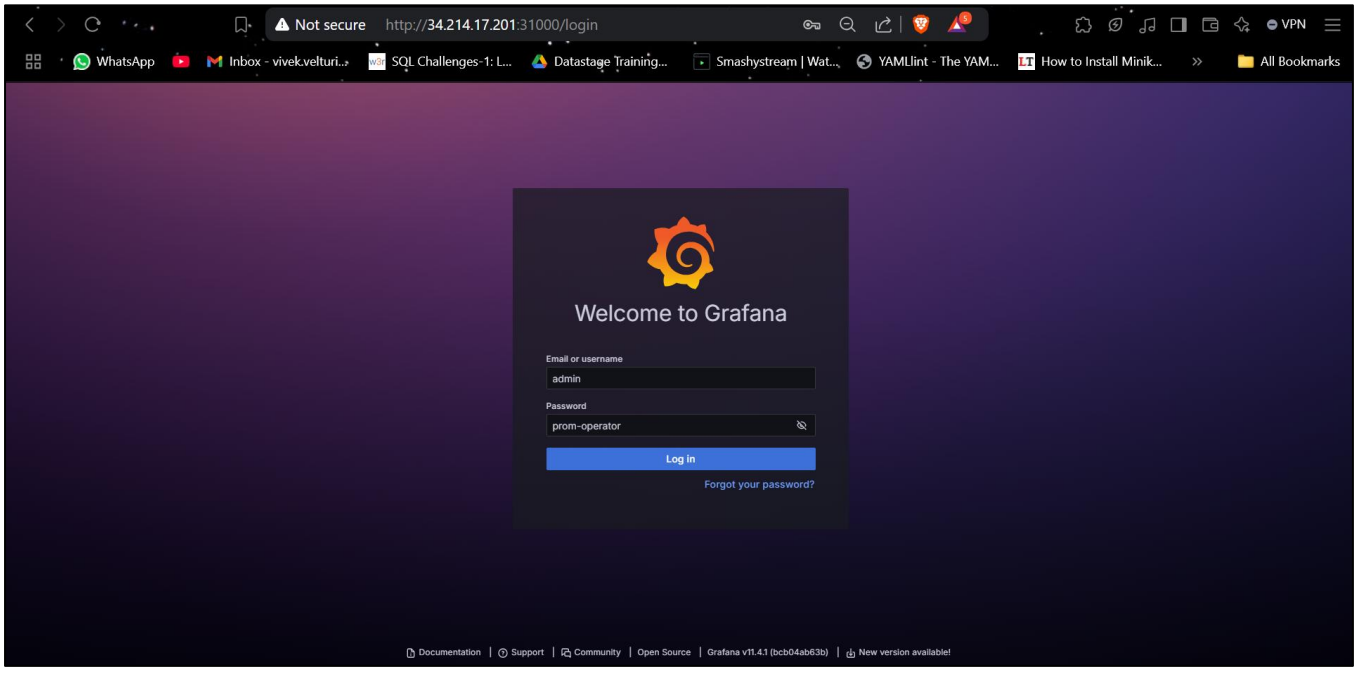


- Once saved, access the Grafana web interface by entering the EC2 public IP followed by `:31000` in the browser, eg. `http:// 34.214.17.201:31000`. This will open the Grafana dashboard for monitoring and visualization.

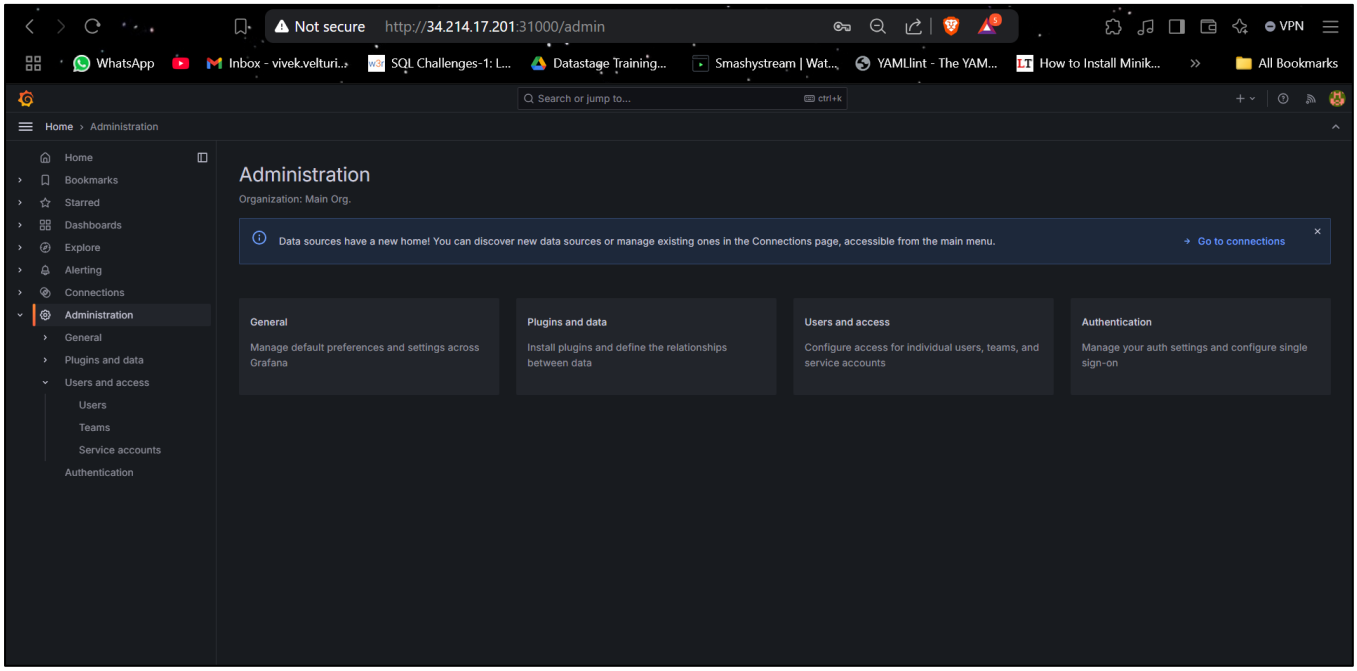


- The default username and password for Grafana, when deployed using the **kube-prometheus-stack** Helm chart, are as follows:
  - **Username**: `admin`
  - **Password**: The password is generated during the Helm installation process, and it can be retrieved from the Kubernetes secret.

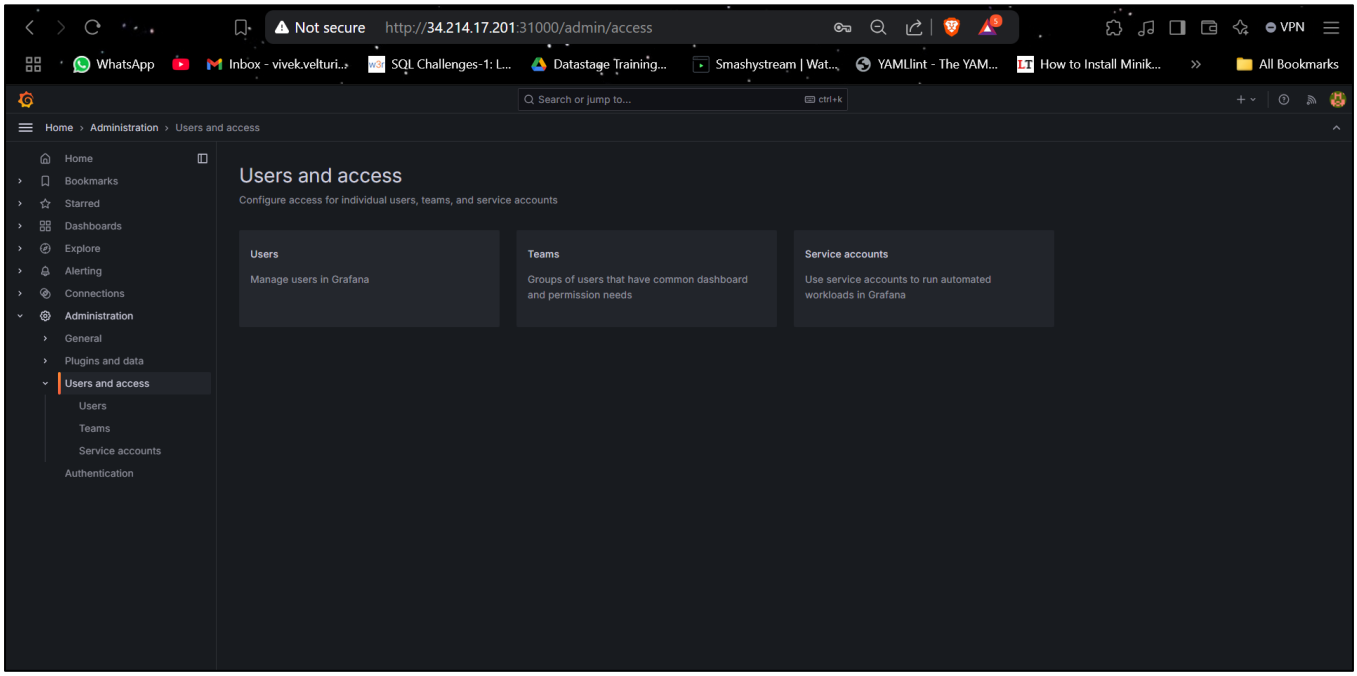
  To get the default password, you can run the following command:
  ```
  kubectl get secret -n monitoring kind-prometheus-grafana -o jsonpath='{.data.admin-password}' | base64 --decode
  ```
  This will retrieve the password from the secret and decode it, allowing you to log in to the Grafana dashboard using the default username `admin` and the decoded password.

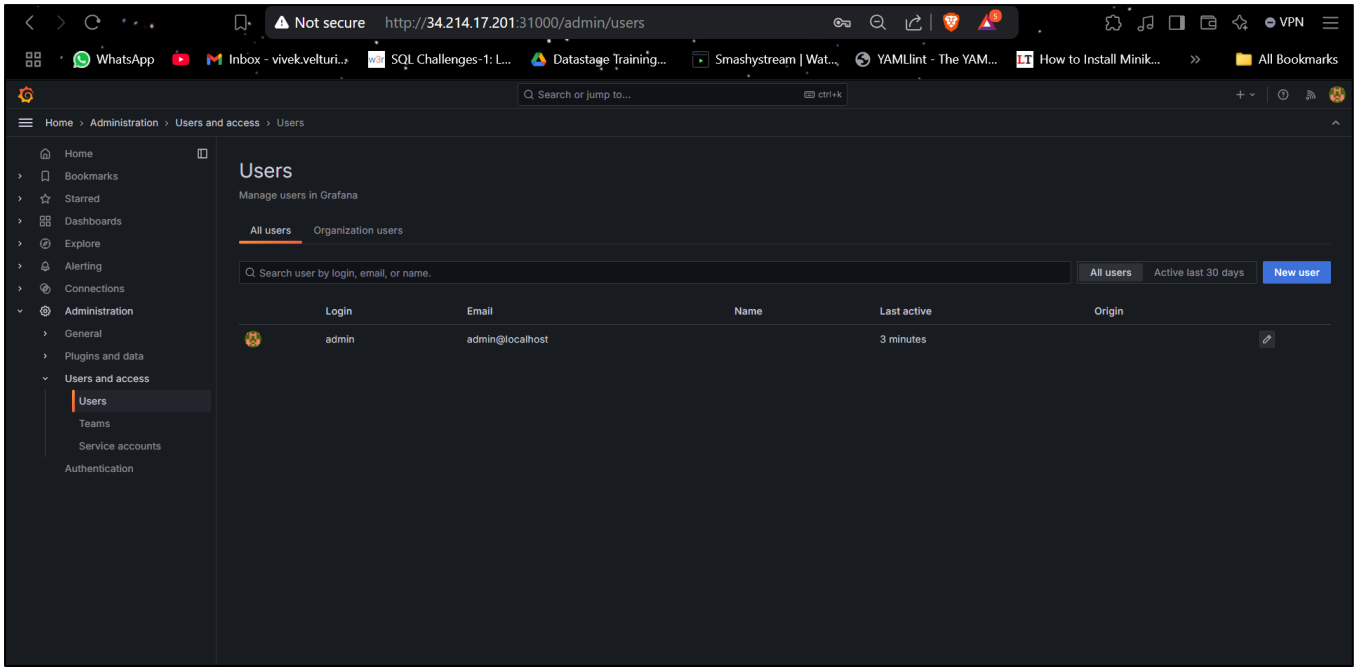


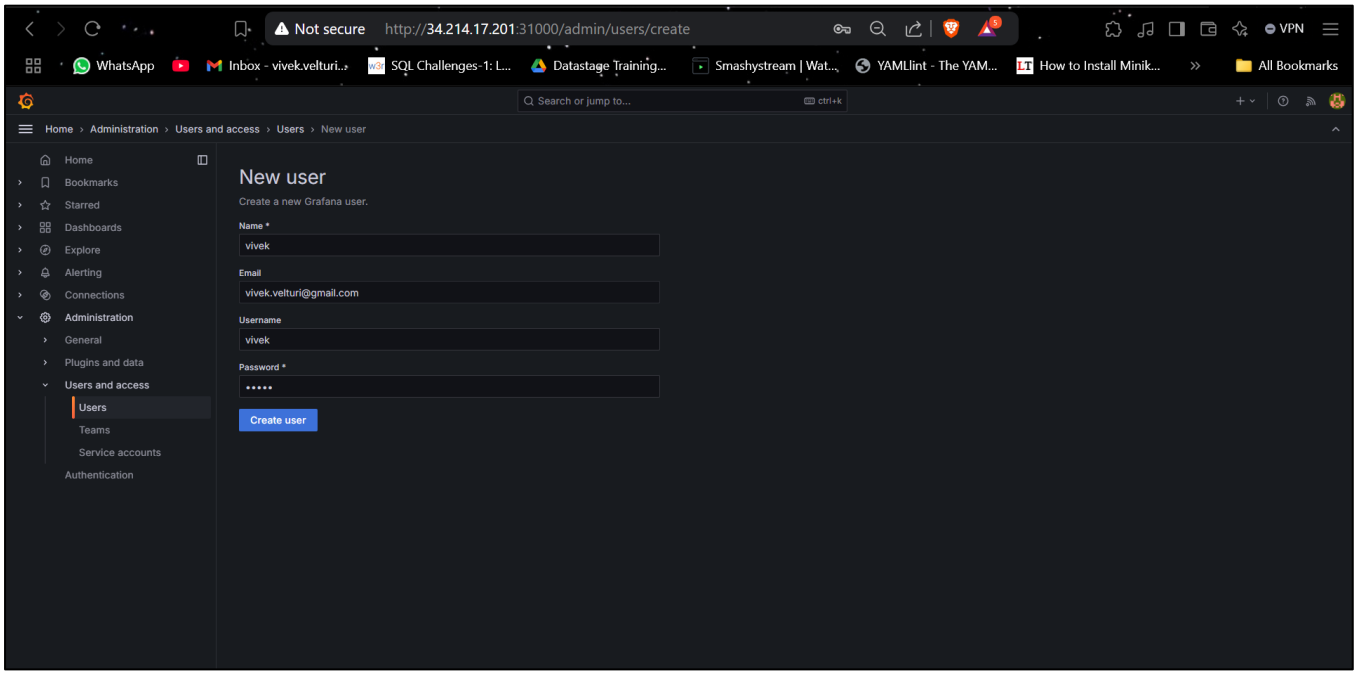
- After logging in to Grafana with the username `admin` and the retrieved password, you will be directed to the Grafana dashboard. The page will display a clean interface with various options like creating and viewing dashboards, configuring data sources, and exploring metrics. You can start monitoring and visualizing the metrics collected from your Kubernetes cluster and other integrated sources.



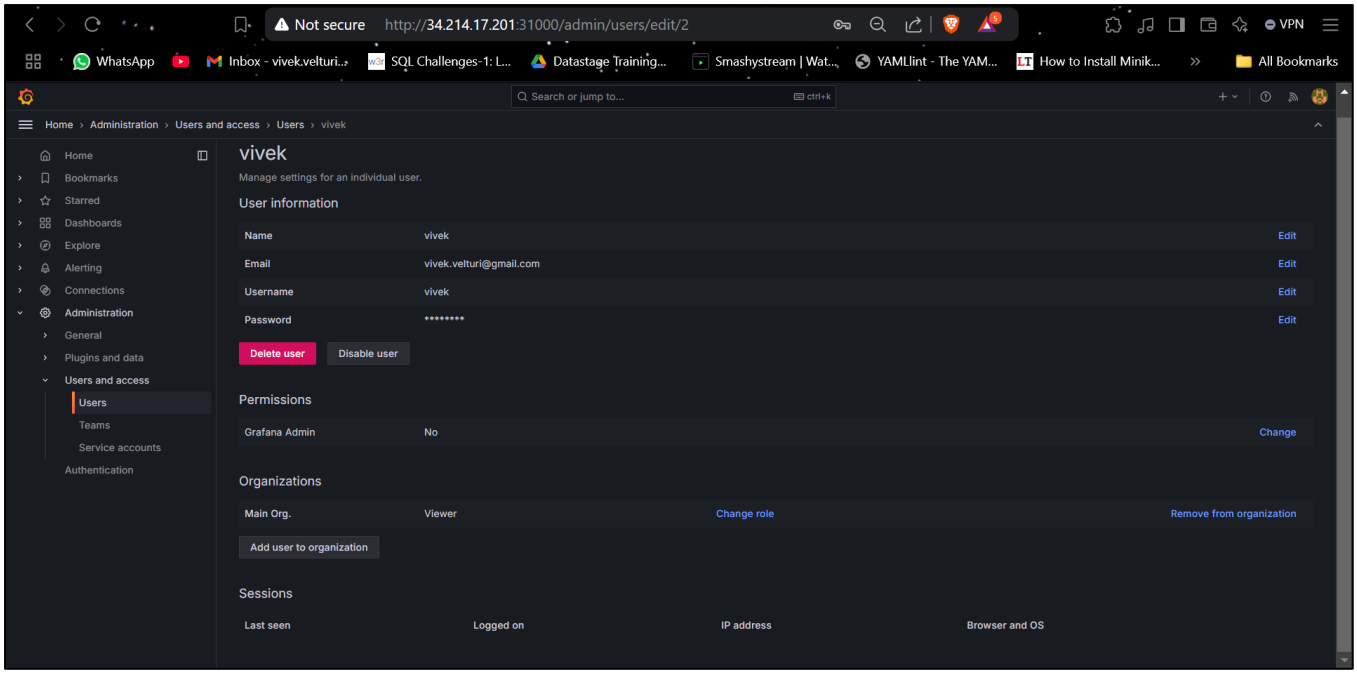
- To create a new user in Grafana, navigate to **"User & Access"** and click on **"Users."** Then, click on **"Add New User"** and **provide the necessary details, including a username and password**.



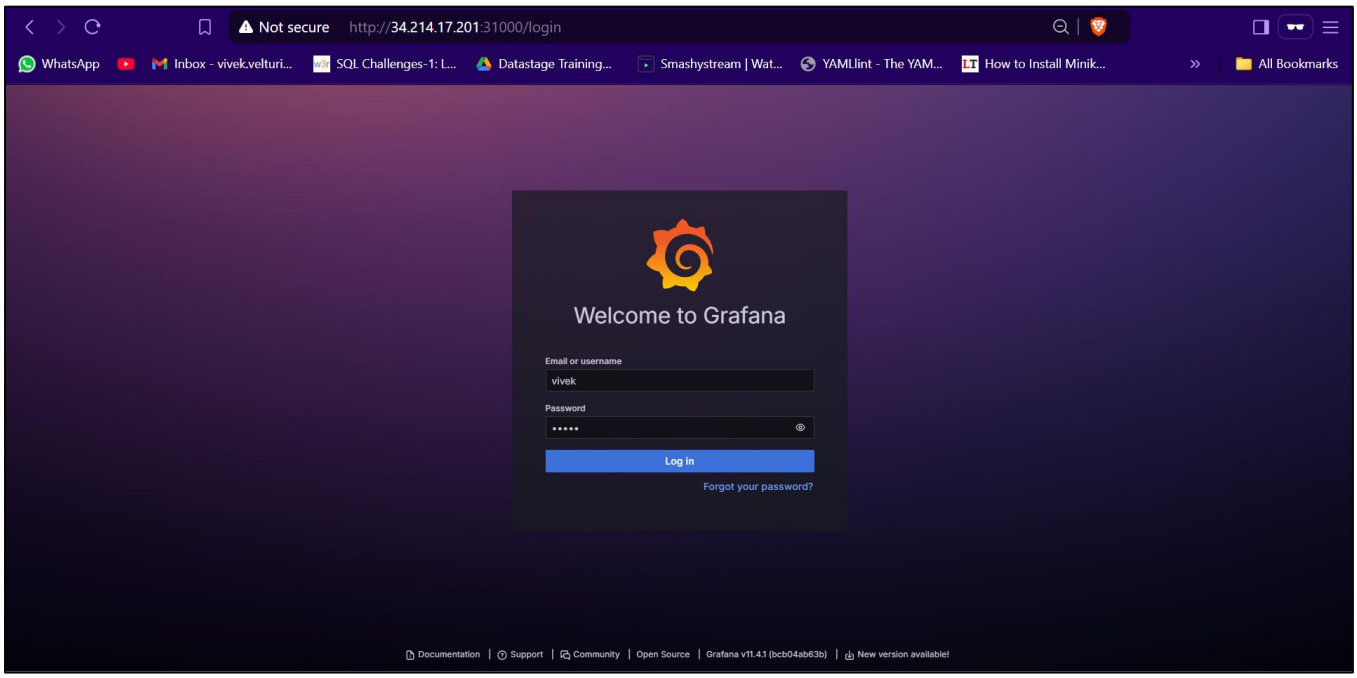


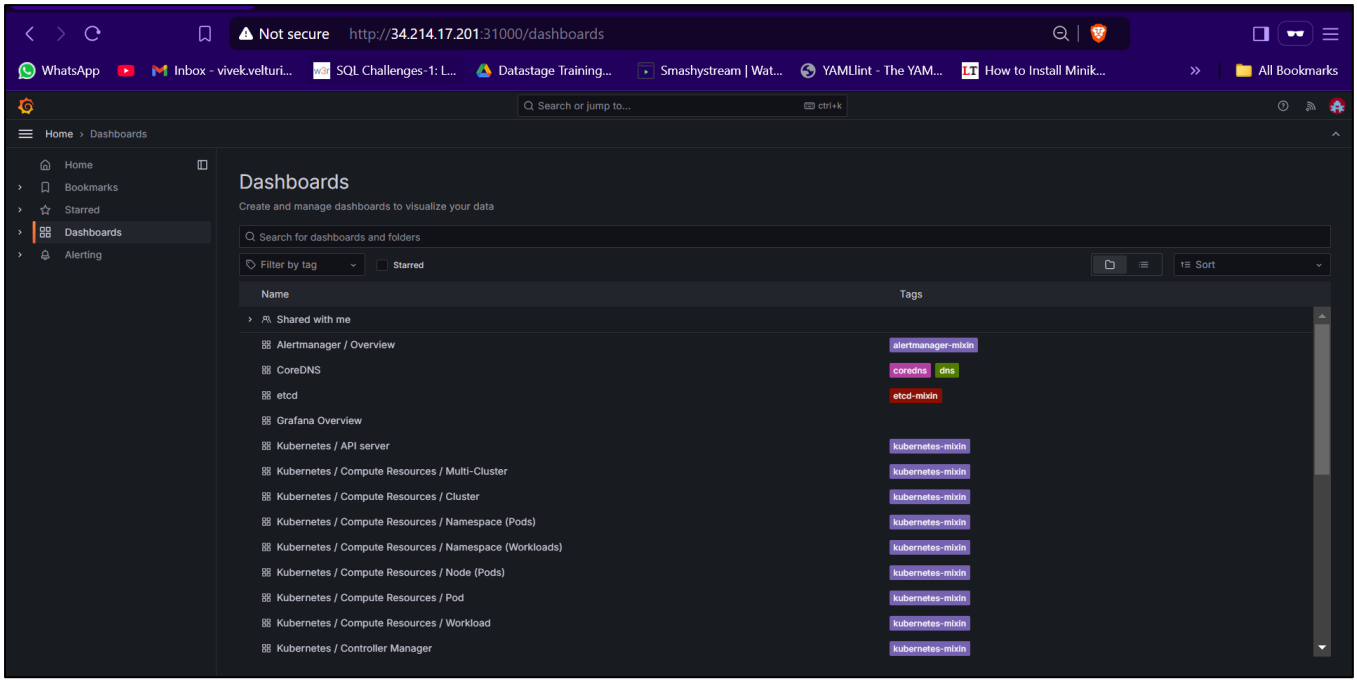


- You can also set the user's role according to your preferences; in this case, a "Viewer" role has been assigned.

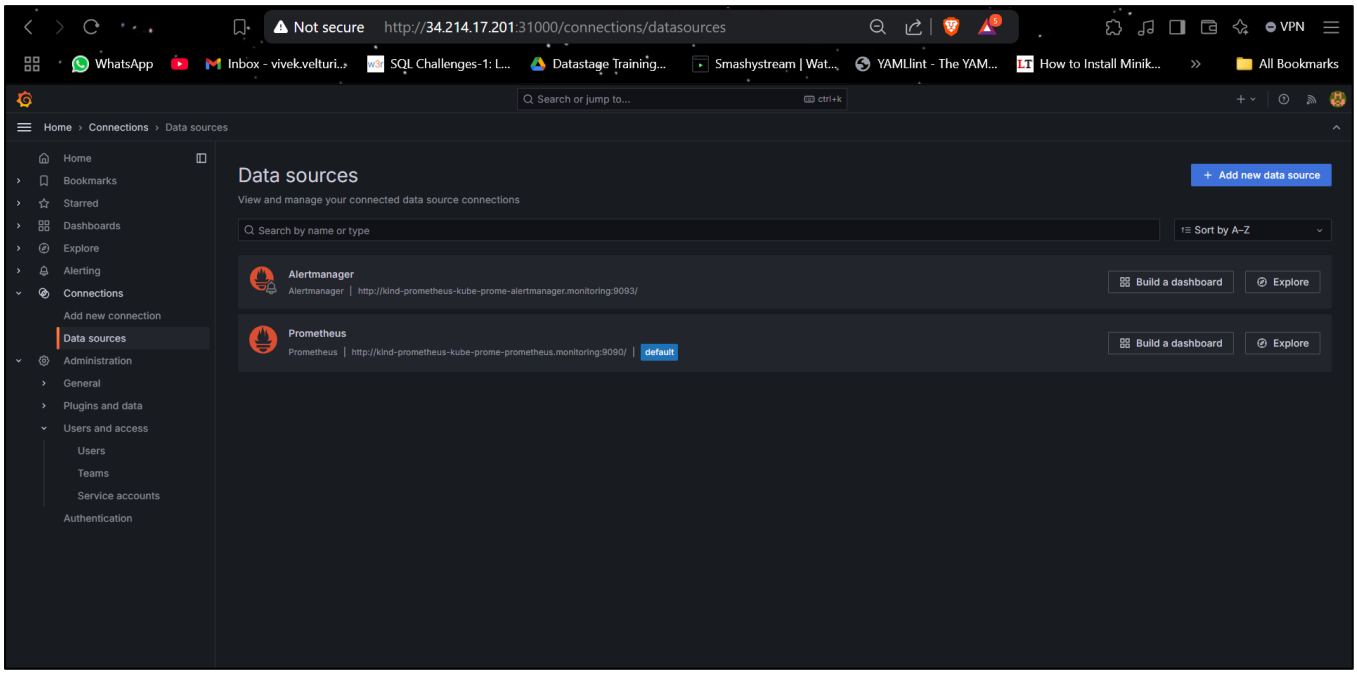


- Once saved, the new user will have access to the Grafana dashboard using their credentials. To provide access to others, simply share the Grafana URL (with the port number) in our case: `http:// 34.214.17.201:31000`, along with the new username and password for them to view the Grafana dashboard.



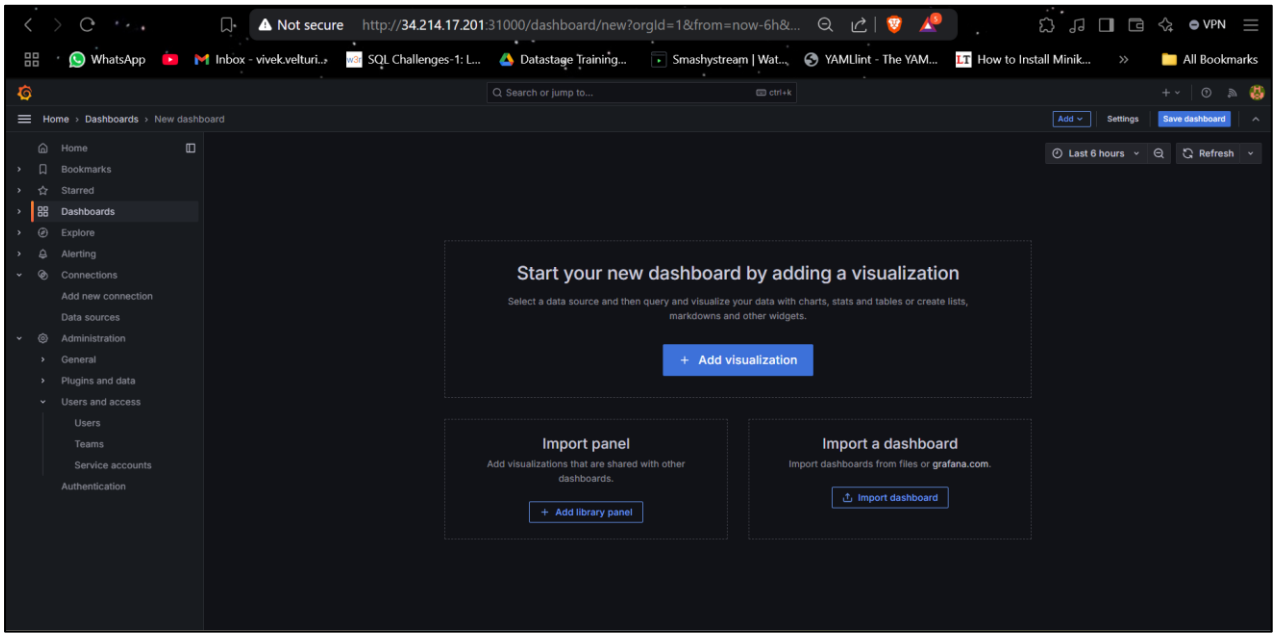


- While setting up Grafana, we can check the data sources in the **"Connections"** section. As expected, **Prometheus** and **Alertmanager** are already listed as data sources. This is because these data sources were automatically configured during the Helm installation when we executed the `helm install` command with the `kube-prometheus-stack` chart. This integration streamlines the monitoring and alerting process within the Kubernetes cluster, allowing us to easily view the metrics collected by Prometheus and the alerts managed by Alertmanager.

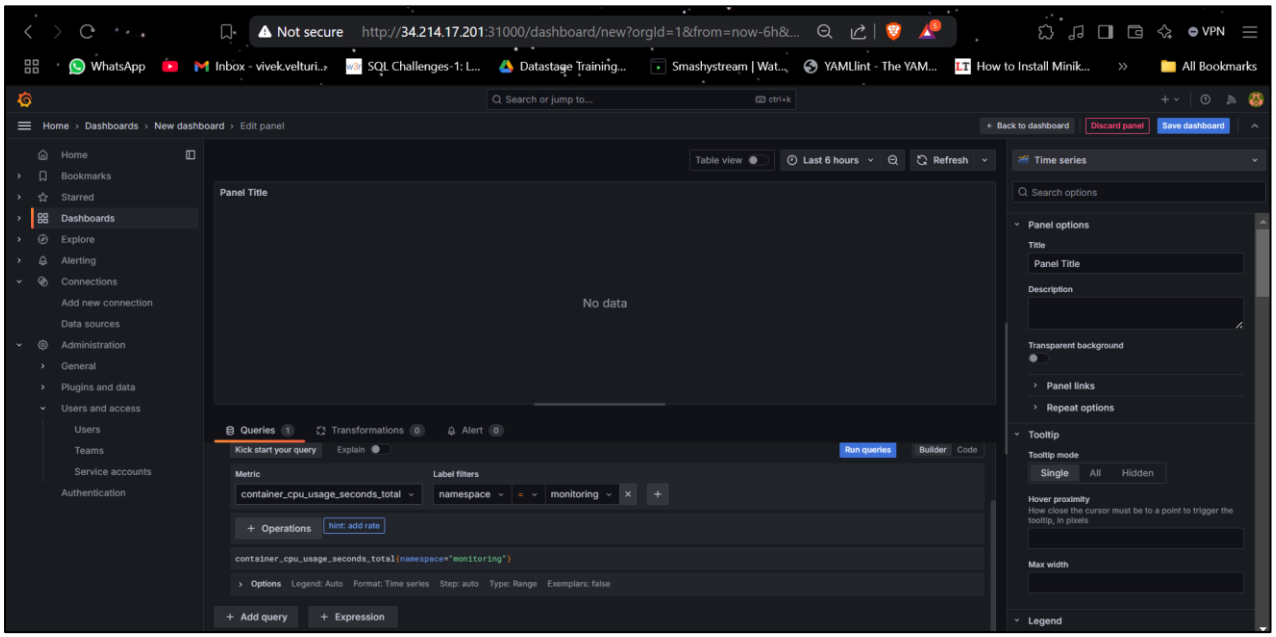


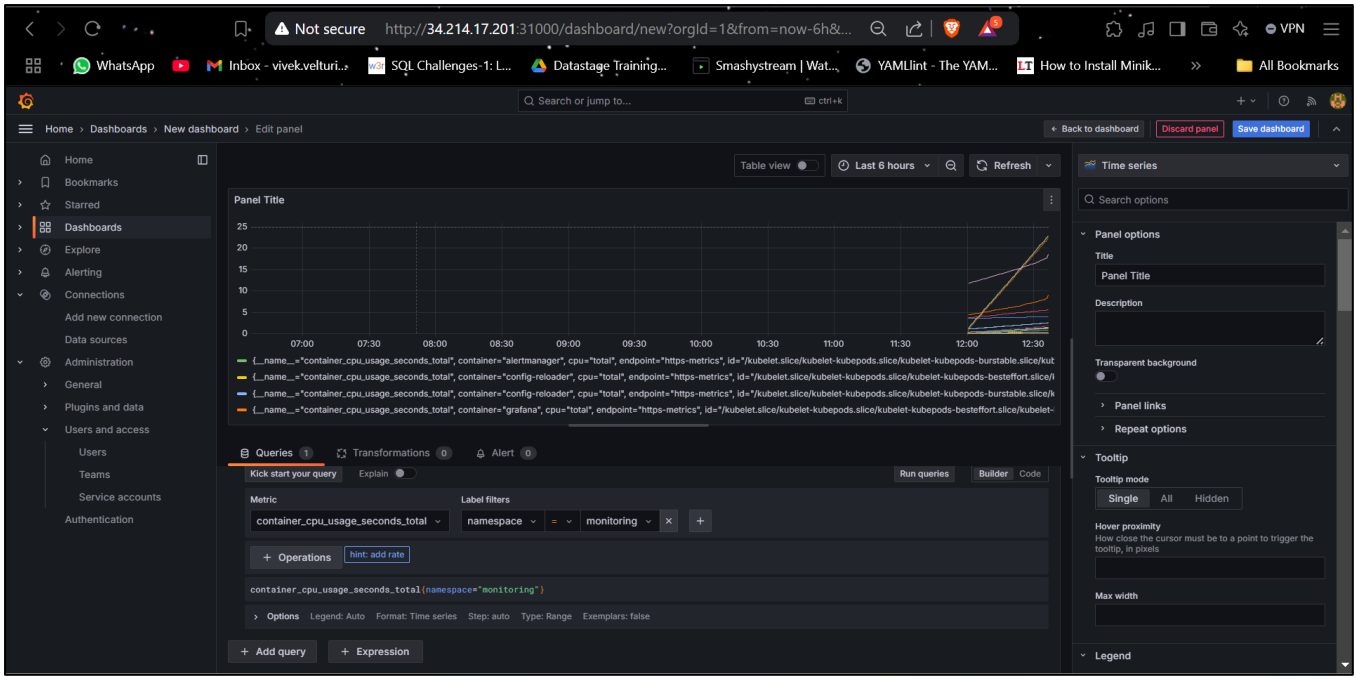
- In the Grafana dashboard, we can create custom dashboards as per our requirements. To do so, navigate to the **"Dashboard"** panel and click on **"Create New Dashboard."** From there, we can add various panels, such as graphs, tables, or heatmaps, to visualize

the metrics collected by Prometheus. These dashboards provide a flexible way to monitor and analyze the performance of applications and infrastructure within the Kubernetes cluster.

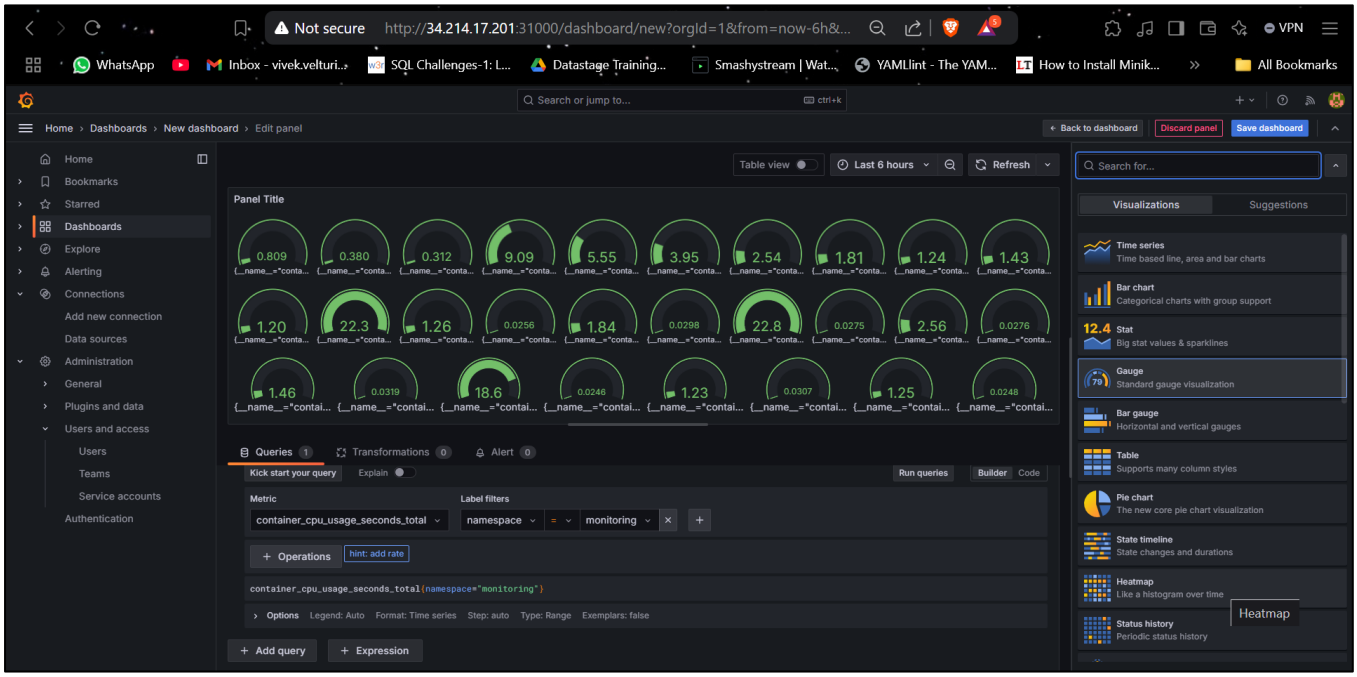


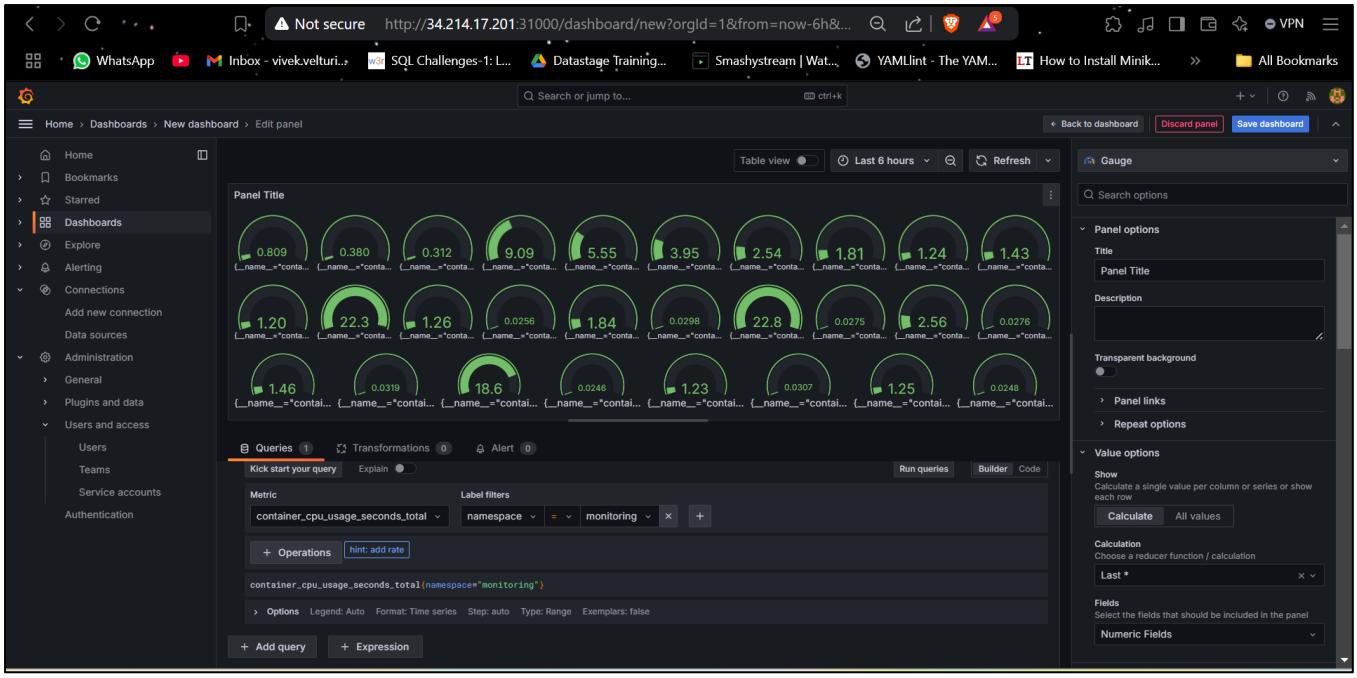
- In the previous step, after clicking on "Add Visualization," you can select **Prometheus** as the data source.
- Once Prometheus is selected, you can enter a query in the metric field to specify the data you want to visualize. For example, you could use a query like `rate(container_cpu_usage_seconds_total[1m])` to visualize CPU usage. Additionally, you can add label filters for **namespace** and set it to **monitoring** to narrow down the data to the relevant metrics within the monitoring namespace. After specifying the query and label filters, you can choose the type of visualization (such as graph or gauge), customize the settings, and save the dashboard. This enables you to track the metrics of your application in real-time on the Grafana dashboard.

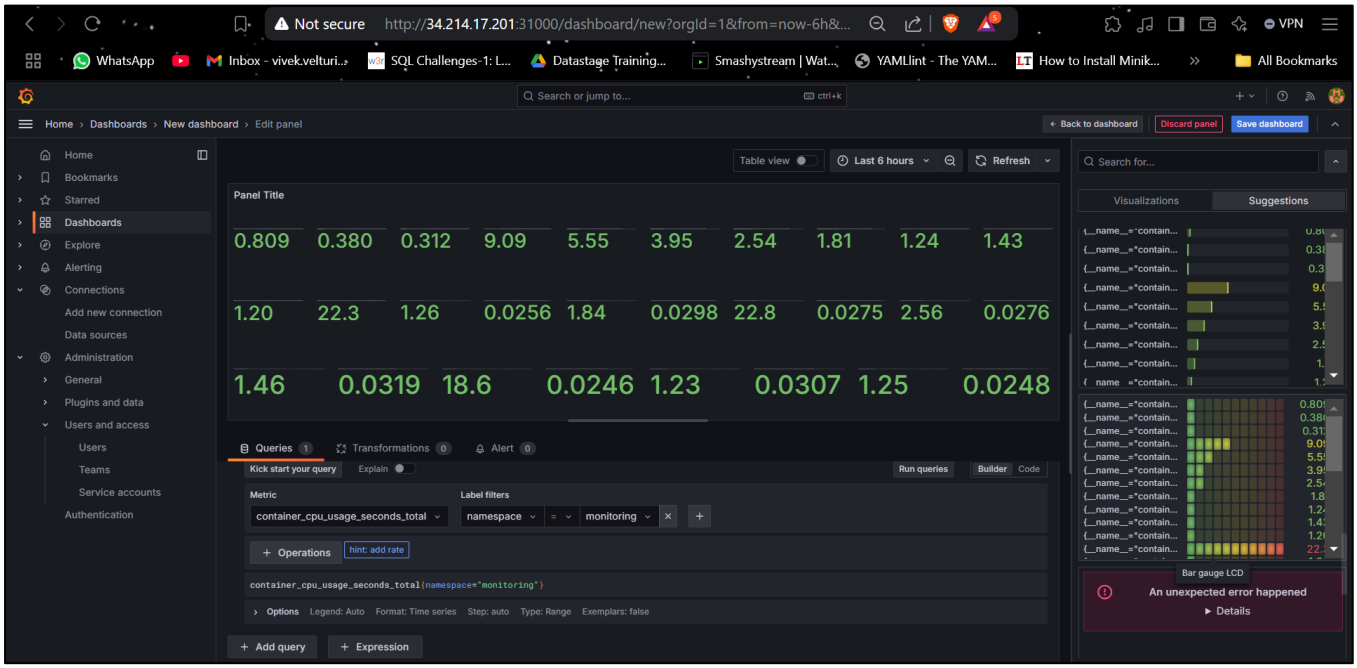




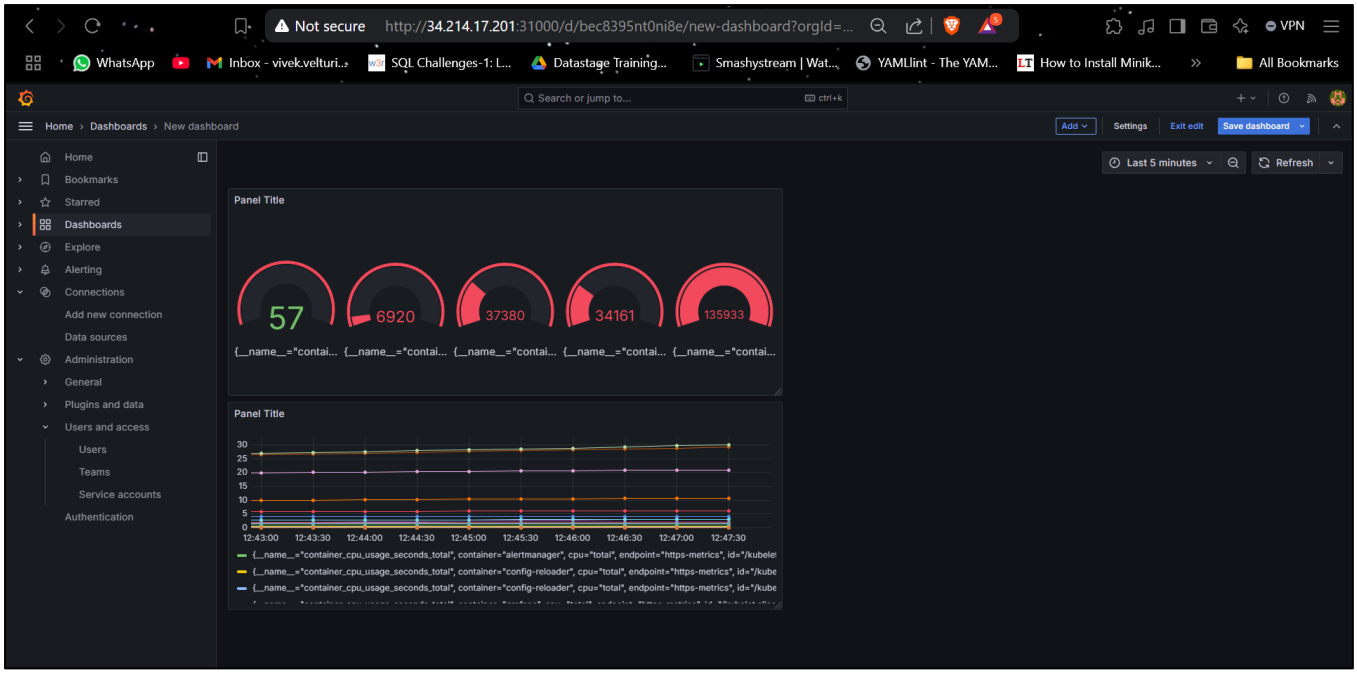
- In **Visualizations**, you will find many types of charts available. In the **Suggestions** tab under **Charts**, you'll find various pre-configured visualizations based on your selected data source, like Prometheus. You can choose from these suggested charts or customize them as needed.



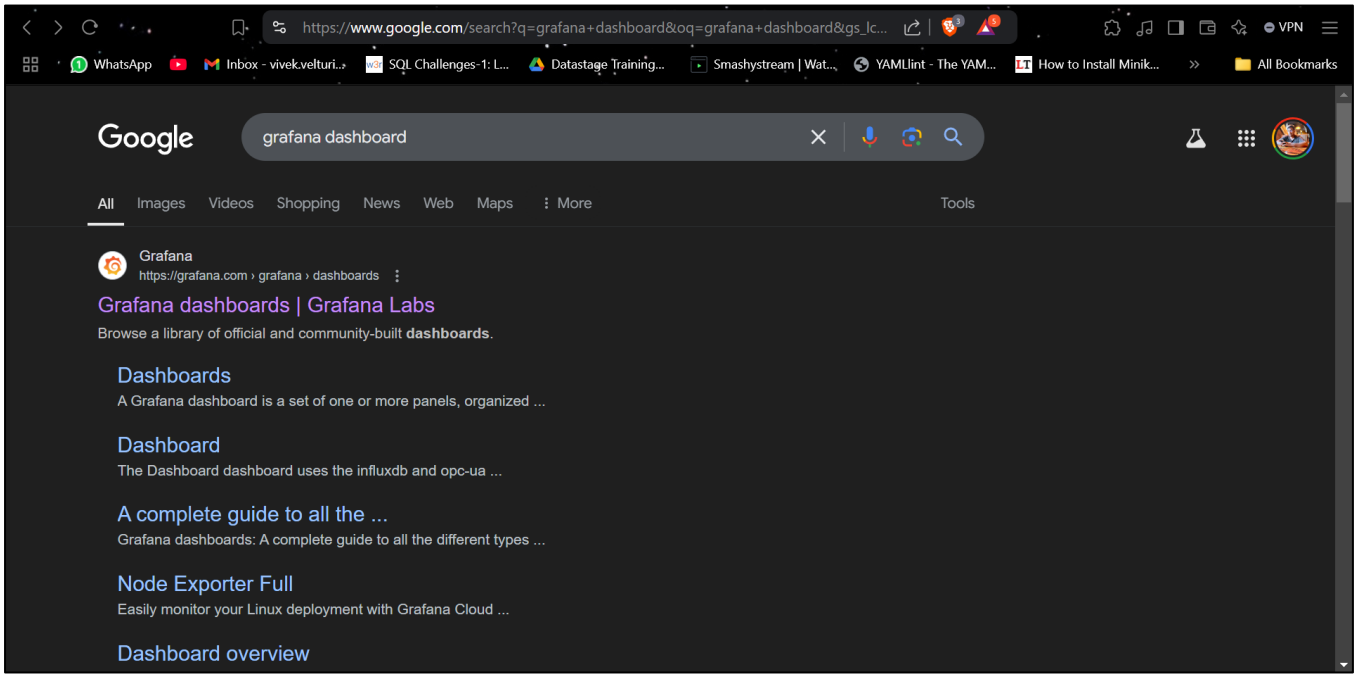


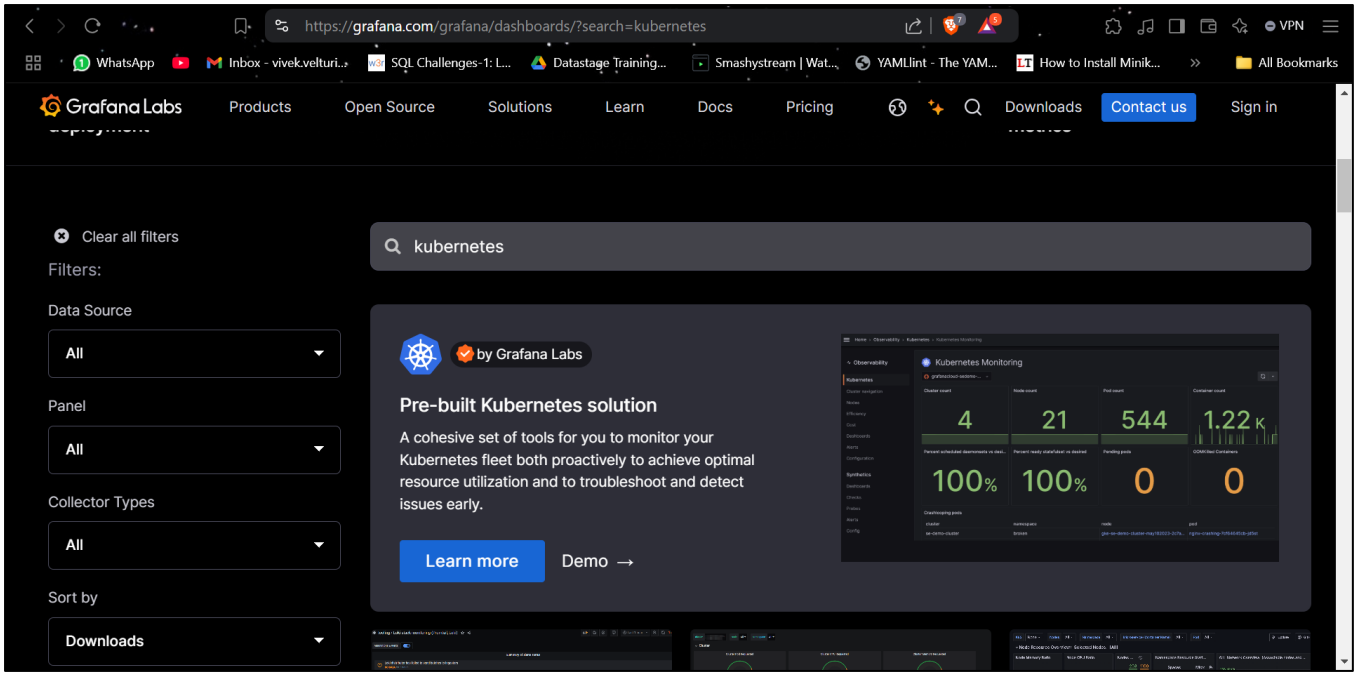


- In the Grafana dashboard, you can combine multiple visualizations into a single view by adding different types of graphs or panels. This allows you to monitor different metrics in one place, such as CPU usage, memory consumption, and network traffic, by placing them together in an organized layout. You can adjust the size, position, and appearance of each graph to fit your preferences. Once you've arranged the visualizations to your liking, you can save the dashboard to preserve the configuration, allowing you to easily access it for ongoing monitoring and analysis.

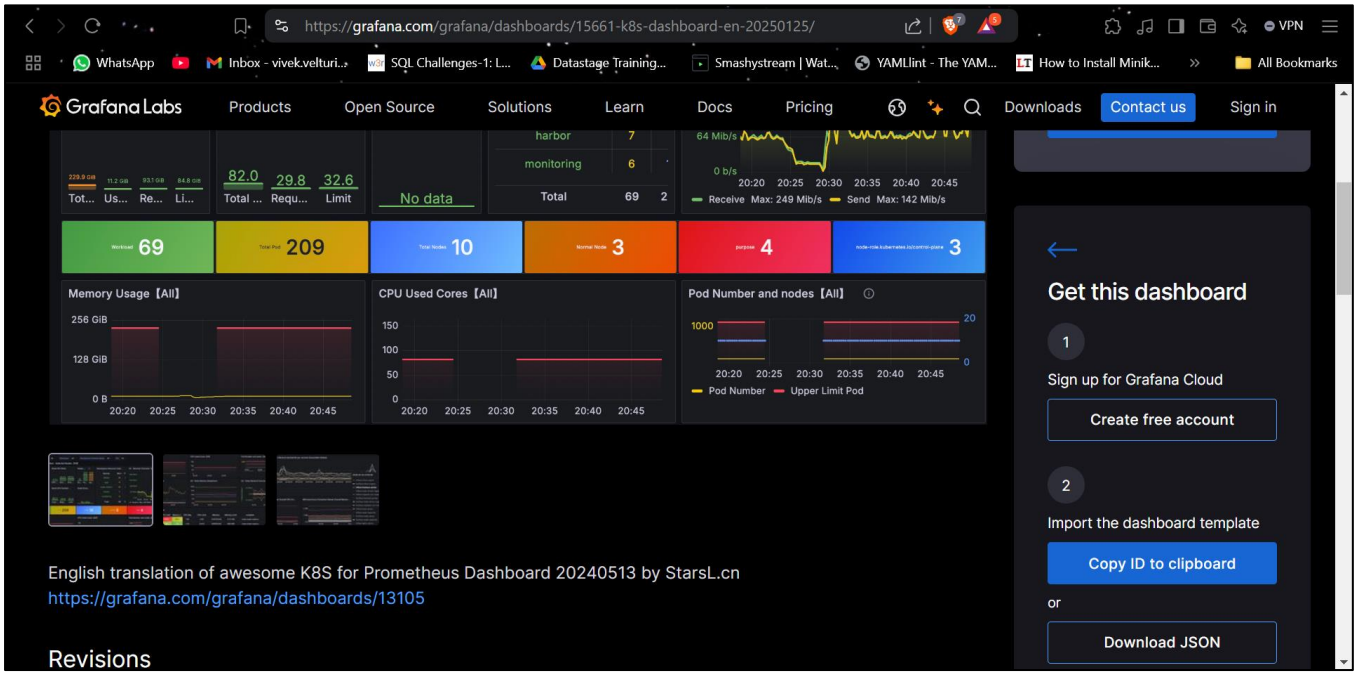


- You can also find pre-made Grafana dashboard templates on the internet. Simply search for "Grafana dashboard" in your browser and visit the official Grafana website or other community resources. You can search for dashboards tailored to specific applications, such as Kubernetes, to get ready-made templates that provide instant insights into your data. These dashboards can be imported directly into your Grafana instance, saving time on setup and offering valuable monitoring capabilities for your application.

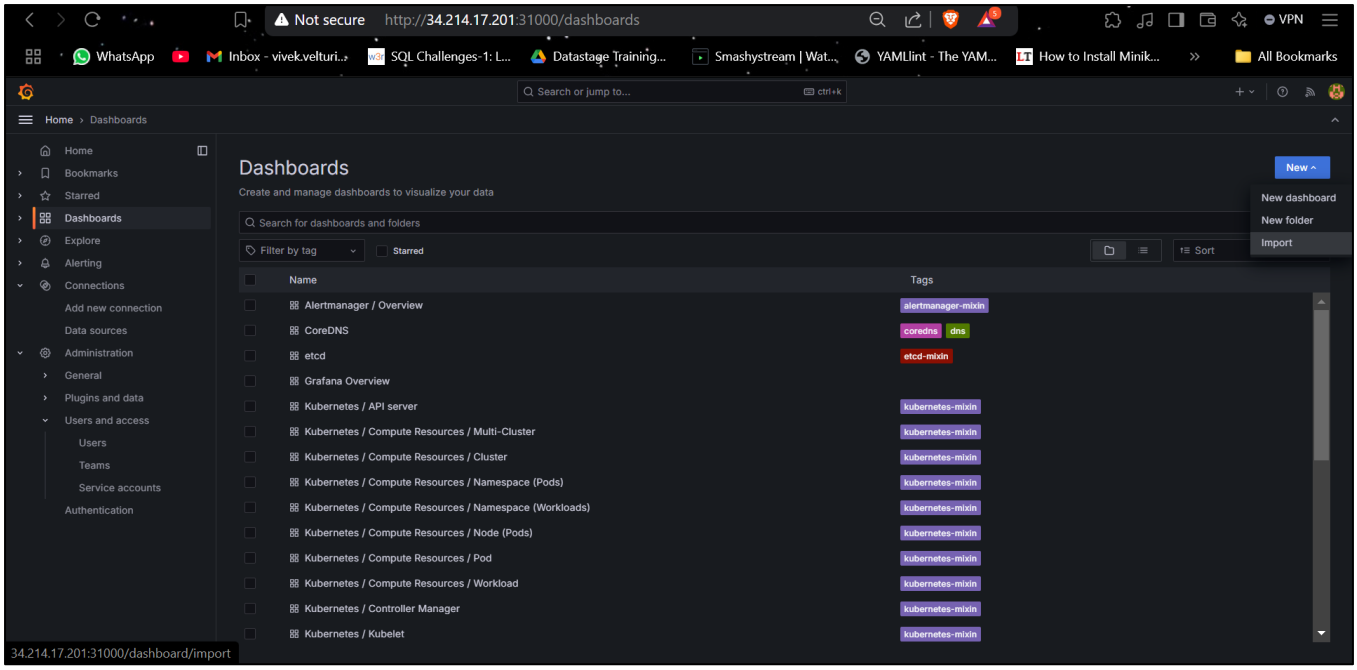




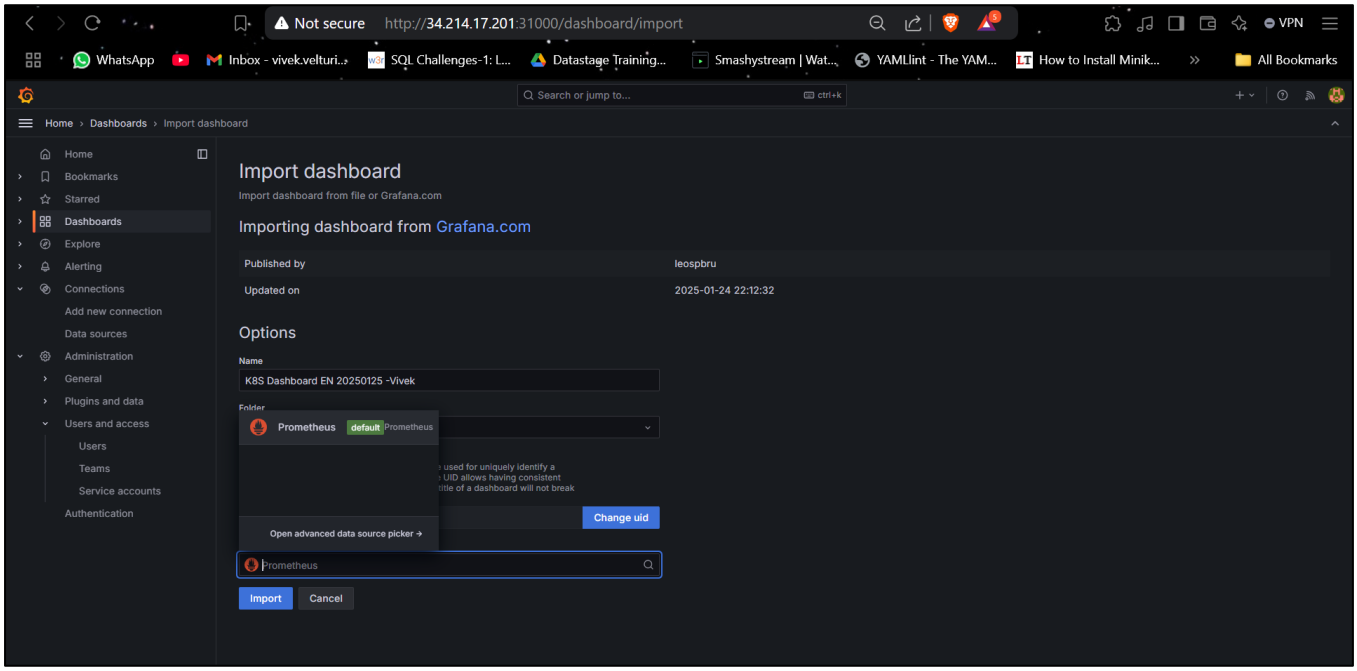
- Select the template that best meets your requirements from the Grafana website or other community sources. Once you've found the desired dashboard, copy its ID. This ID will be used to import the dashboard into your Grafana instance, allowing you to visualize your data with the pre-configured metrics and visualizations that suit your needs.



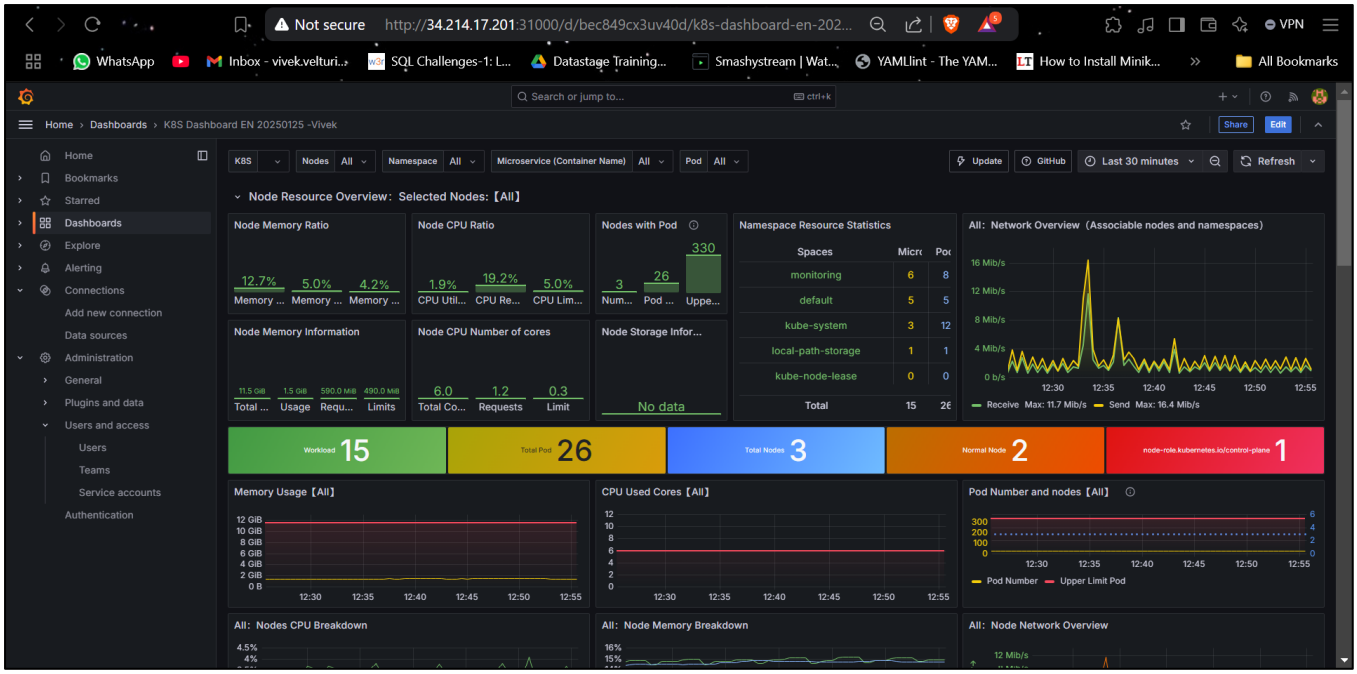
- To import a pre-made Grafana dashboard, click on the "**New**" option in the sidebar, then select "**Import**" from the dropdown.

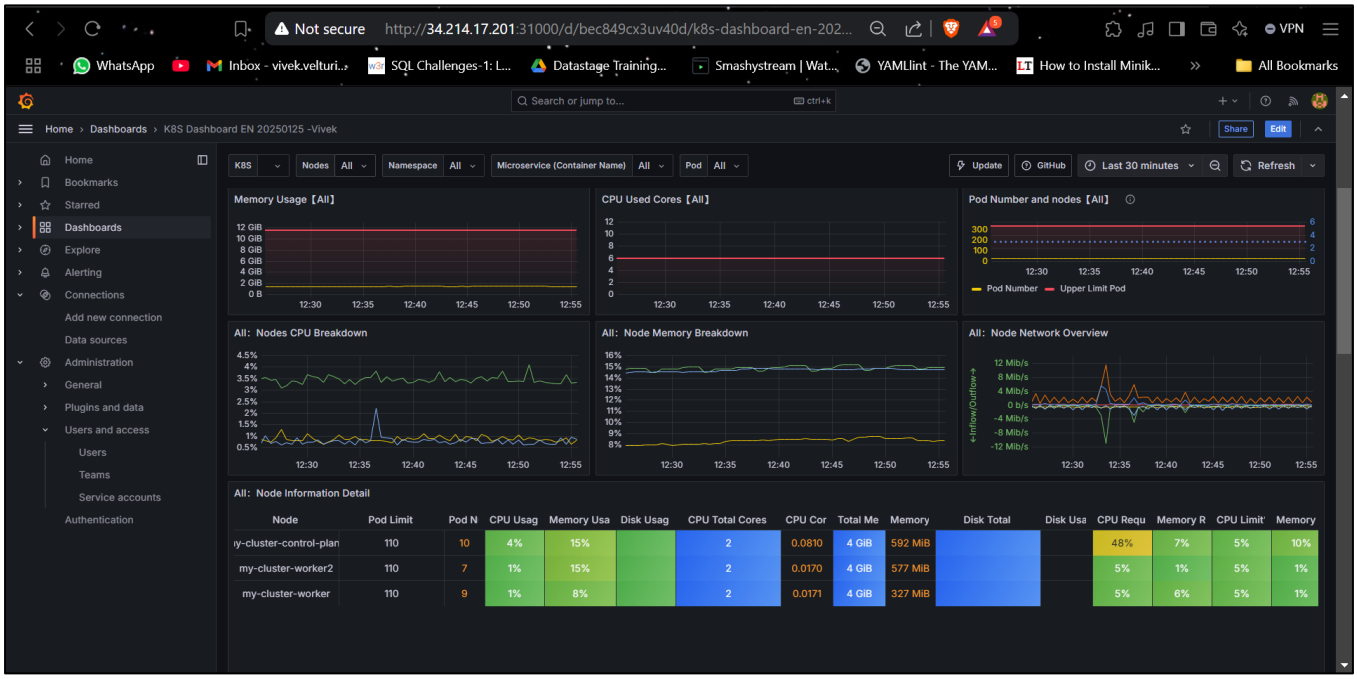


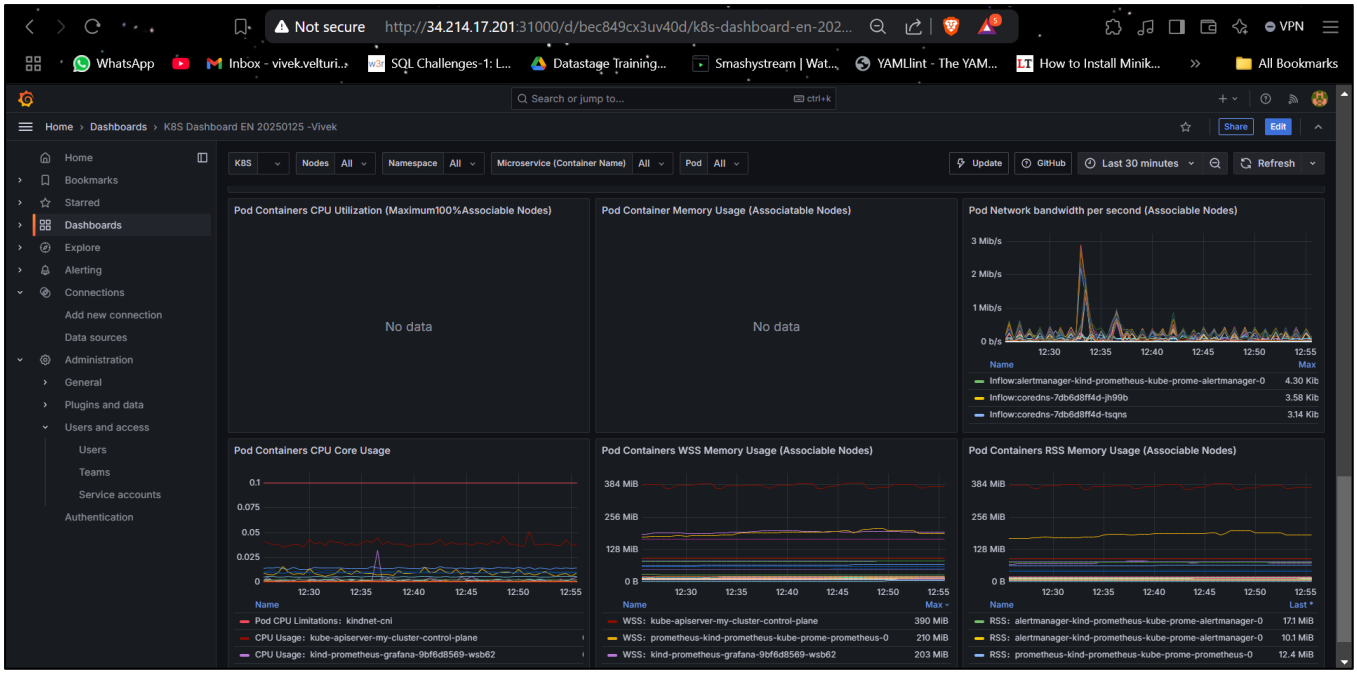
- Once you're on the Import page, paste the dashboard ID you copied from the Grafana website into the provided field. Afterward, click "**Load**" to load the dashboard template. In the next step, ensure **Prometheus** is selected as the data source. This will allow you to quickly integrate a pre-configured dashboard for your specific application, such as Kubernetes, into your Grafana instance, visualizing metrics from Prometheus.



- Now, once the dashboard is successfully imported and linked with the correct data source (Prometheus), you can view the statistics of your Kubernetes cluster. The pre-configured template will display various metrics such as CPU usage, memory usage, network traffic, and more, giving you a clear overview of your cluster's performance. This allows you to monitor and analyze your Kubernetes environment effectively through Grafana's visualizations.


Not secure
http://34.214.17.201:31000/d/bec849cx3uv40d/k8s-dashboard-en-202...
VPN
WhatsApp
Inbox - vivek.velturi...
SQL Challenges-1: L...
Datastage Training...
Smashystream | Wat...
YAMLlint - The YAM...
How to Install Minik...
All Bookmarks
Search or jump to...
Home › Dashboards › K8S Dashboard EN 20250125 -Vivek
Share
Edit
Home
Bookmarks
Starred
Dashboards
Explore
Alerting
Connections
Add new connection
Data sources
Administration
General
Plugins and data
Users and access
Users
Teams
Service accounts
Authentication
K8S
Nodes All
Namespace All
Microservice (Container Name) All
Pod All
Update
GitHub
Last 30 minutes
Refresh
Node Resource Overview: Selected Nodes: [All]
Node Memory Ratio
12.7% Memory ...
5.0% Memory ...
4.2% Memory ...
Node CPU Ratio
1.9% CPU Util...
19.2% CPU Re...
5.0% CPU Lim...
Nodes with Pod
330
3 Num...
26 Pod ...
Uppe...
Namespace Resource Statistics
Spaces
monitoring 6 8
default 5 5
kube-system 3 12
local-path-storage 1 1
kube-node-lease 0 0
Total 15
All: Network Overview (Associable nodes and namespaces)
16 Mib/s
12 Mib/s
8 Mib/s
4 Mib/s
0 b/s
12:30 12:35 12:40 12:45 12:50 12:55
Receive Max: 11.7 Mib/s
Send Max: 16.4 Mib/s
Node Memory Information
11.5 GiB Total ...
1.5 GiB Usage
590.0 MiB Requ...
490.0 MiB Limits
Node CPU Number of cores
6.0 Total Co...
1.2 Requests
0.3 Limit
Node Storage Infor...
No data
Workload 15
Total Pod 26
Total Nodes 3
Normal Node 2
node-role.kubernetes.io/control-plane 1
Memory Usage [All]
12 GiB
10 GiB
8 GiB
6 GiB
4 GiB
2 GiB
0 B
12:30 12:35 12:40 12:45 12:50 12:55
CPU Used Cores [All]
12
10
8
6
4
2
0
Pod Number and nodes [All]
300
200
100
0
Pod Number
Upper Limit Pod
All: Nodes CPU Breakdown
4.5%
4%
All: Node Memory Breakdown
16%
15%
All: Node Network Overview
12 Mib/s


Not secure
http://34.214.17.201:31000/d/bec849cx3uv40d/k8s-dashboard-en-202...
VPN
WhatsApp
Inbox - vivek.velturi...
SQL Challenges-1: L...
Datastage Training...
Smashystream | Wat...
YAMLlint - The YAM...
How to Install Minik...
All Bookmarks
Search or jump to...
Home › Dashboards › K8S Dashboard EN 20250125 -Vivek
Share
Edit
Home
Bookmarks
Starred
Dashboards
Explore
Alerting
Connections
Add new connection
Data sources
Administration
General
Plugins and data
Users and access
Users
Teams
Service accounts
Authentication
K8S
Nodes All
Namespace All
Microservice (Container Name) All
Pod All
Update
GitHub
Last 30 minutes
Refresh
Memory Usage [All]
12 GiB
10 GiB
8 GiB
6 GiB
4 GiB
2 GiB
0 B
12:30 12:35 12:40 12:45 12:50 12:55
CPU Used Cores [All]
12
10
8
6
4
2
0
Pod Number and nodes [All]
300
200
100
0
Pod Number
Upper Limit Pod
All: Nodes CPU Breakdown
4.5%
4%
3.5%
3%
2.5%
2%
1.5%
1%
0.5%
All: Node Memory Breakdown
16%
15%
14%
13%
12%
11%
10%
9%
8%
All: Node Network Overview
12 Mib/s
8 Mib/s
4 Mib/s
0 b/s
-4 Mib/s
-8 Mib/s
-12 Mib/s
←Inflow/Outflow→
All: Node Information Detail
Node
Pod Limit
Pod N
CPU Usag
Memory Usa
Disk Usag
CPU Total Cores
CPU Cor
Total Me
Memory
Disk Total
Disk Usa
CPU Requ
Memory R
CPU Limit
Memory
ıy-cluster-control-plan 110 10 4% 15% 2 0.0810 4 GiB 592 MiB 48% 7% 5% 10%
my-cluster-worker2 110 7 1% 15% 2 0.0170 4 GiB 577 MiB 5% 1% 5% 1%
my-cluster-worker 110 9 1% 8% 2 0.0171 4 GiB 327 MiB 5% 6% 5% 1%



- In conclusion, this project successfully demonstrated how to set up a Kubernetes cluster on AWS EC2, deploy applications, and implement observability with Prometheus and Grafana. By using Helm to simplify the installation of monitoring tools, we were able to visualize critical metrics from the cluster and applications in real time. This setup enhances the ability to monitor performance, troubleshoot issues, and ensure the reliability of the deployed applications, providing a comprehensive observability solution.