CONTAINER ORCHESTRATION SYSTEM UTILIZING KUBERNETES

K8s Wizards (Team 29)

TEAM MEMBERS

VENKATA NAGENDRA

RAVNDRA REDDY

SHIVA

MANOJ

OVERVIEW

1. INTRODUCTION TO KUBERNETES AND DOCKER

2. CREATING IMAGES IN DOCKER AND PUSHING THEM INTO DOCKER HUB

3. INSTALLATION OF MINIKUBE

4. INSTALLATION OF KUBECTL

5. COMMANDS FOR MINIKUBE

6. INSTALLATION OF CHOCOLATEY AND HELM

7. INSTALLATION OF PROMETHEUS

8. INSTALLATION OF GRAFANA

9. MONITORING THE CONTAINERS

10. CONCLUSION

## 1. INTRODUCTION TO KUBERNETES AND DOCKER

### What is Kubernetes?

Kubernetes (also known as k8s or “kube”) is an open-source container orchestration platform that automates many of the manual processes involved in deploying, managing, and scaling containerized applications.

What is Docker?

Docker is an open-source containerized platform that Is used for developing, deploying, and managing applications in lightweight virtualized environments called containers.

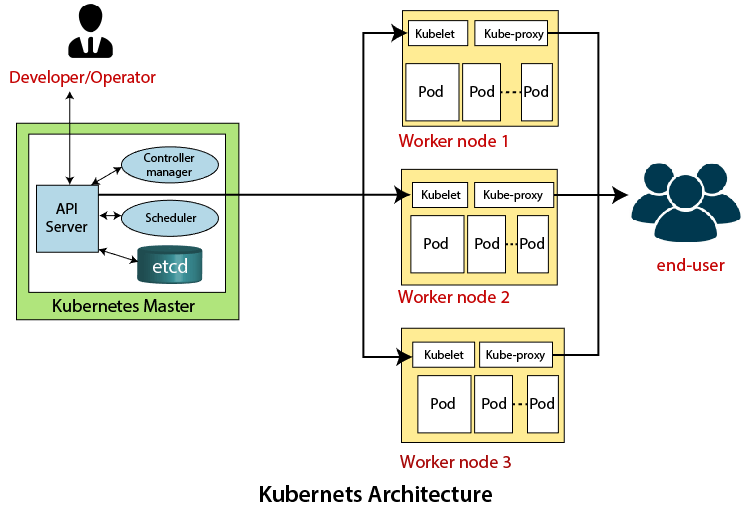
Key Objects of Kubernetes :

* **Pod:** A pod is a collection of one or more containers and is the smallest unit of a Kubernetes application. Any given pod can be composed of multiple, tightly coupled containers or just a single container.
* **Node:** A **Node** is nothing but a single host used to run virtual or physical machines. A node in the Kubernetes cluster is also known as a minion.
* **Service:** A **service** in Kubernetes is a logical set of pods, which works together. With the help of services, users can easily manage load-balancing configurations.
* **Replica Set:** A **Replica Set** is a Kubernetes object used to maintain a stable set of replicated pods running within a cluster at any given time.
* **Namespace:** Kubernetes supports various virtual clusters, which are known as **namespaces**. It is a way of dividing the cluster resources between two or more users.

What are the features of Kubernetes?

* Automatic Bin packing.
* Service Discovery & Load balancing.
* Storage Orchestration.
* Self-Healing.
* Secret & Configuration Management.
* Batch Execution.
* Horizontal Scaling.
* Automatic Rollbacks & Rollouts.

**Architecture Of K8s:**



The architecture of Kubernetes follows the client-server architecture. It consists of the following two main components:

1. Master Node
2. Slave/worker node
3. **Master Node:** The master node in a Kubernetes architecture is used to manage the states of a cluster. It is an entry point for all types of administrative tasks.

Following are the four different components that exist in the Master node or Kubernetes Control plane:

* API Server
* Scheduler
* Controller Manager
* ETCD

**API Server:** The Kubernetes API server receives the REST commands which are sent by the user. After receiving them, it validates the REST requests, processes, and then executes them. After the execution of REST commands, the resulting state of a cluster is saved in '**etcd**' as a distributed key-value store.

**Scheduler:** The scheduler in a master node schedules the tasks to the worker nodes. And, for every worker node, it is used to store the resource usage information. In other words, it is a process that is responsible for assigning pods to the available worker nodes.

**Controller Manager:** The Controller manager is also known as a controller. It is a daemon that executes in the non-terminating control loops. The controllers in a master node perform a task and manage the state of the cluster. In the Kubernetes, the controller manager executes the various types of controllers for handling the nodes, endpoints, etc.

**ETCD:** It is an open-source, simple, distributed key-value storage which is used to store the cluster data. It is a part of a master node which is written in a GO programming language.

1. **Slave/worker node:** The Worker node in a Kubernetes is also known as minions. A worker node is a physical machine that executes the applications using pods. It contains all the essential services which allow a user to assign the resources to the scheduled containers.

Following are the different components which are presents in the Worker or slave node:

* **Kubelet**
* **Kube-proxy**
* **Pods**

**Kubelet:** The Kubelet is responsible for **managing the deployment of pods to Kubernetes nodes**. It receives commands from the API server and instructs the container runtime to start or stop containers as needed.

**Kube-Proxy:** The main aim of this component is requesting forwarding. Each node interacts with the Kubernetes services through **Kube-proxy**.

**Pods:** A **pod** is a combination of one or more containers which logically execute together on nodes. One worker node can easily execute multiple pods.

Docker Architecture:

Docker follows Client-Server architecture, which includes the three main components that are Docker Client, Docker Host, and Docker Registry

Docker Client:

The Docker client gives commands along with REST API to communicate with the docker daemon. When the client gives any docker command in the client terminal. The docker client sends this to the docker daemon. So, the docker daemon will give the result to the client terminal which is given in the form of commands and Rest API requests. Docker Client can communicate with multiple daemons

Docker Host:

The device on which we install the docker software will be the docker host. This docker host will provide an environment to deploy and manage applications. It contains a docker daemon, images, containers, storage, and, networking

Docker Registry:

A Docker registry is used to store Docker images.

They are two types of registries:

1. Public Registry: This can be used by any client and at any time Ex: Docker Hub
2. Private Registry: This can be used to share the images within the enterprise

Docker Commands:

1. Pull: docker “pull” command is used to pull any image specified from the docker hub
2. Run: docker “run” command is used to run the image specified. If the specified image is not available in the local registry, it simply downloads through the docker hub and runs it
3. Create: docker “create” command is used to get an image from the docker hub and get into the system
4. Rm: docker “rm” command followed by container id is used to delete container with specified container id
5. Rmi: docker “rmi” command followed by image id is used to delete the image specified. To remove the image, we must first delete all the container which is created by using this image. The command “rmi -r” is used to delete the image forcefully
6. Build: Docker “build” command is used to build a image with the specified file we selected as software
7. Version: Docker “version” command is used to display the client and server-side full details
8. Images: Docker “images” command is used to see the images that are downloaded in our local repository
9. Stop: Docker “stop” command is used to stop the specified container
10. Start: Docker “start” command is used to start the specified container

# 2. CREATING IMAGES IN DOCKER AND PUSHING THEM INTO DOCKER HUB

Commands:

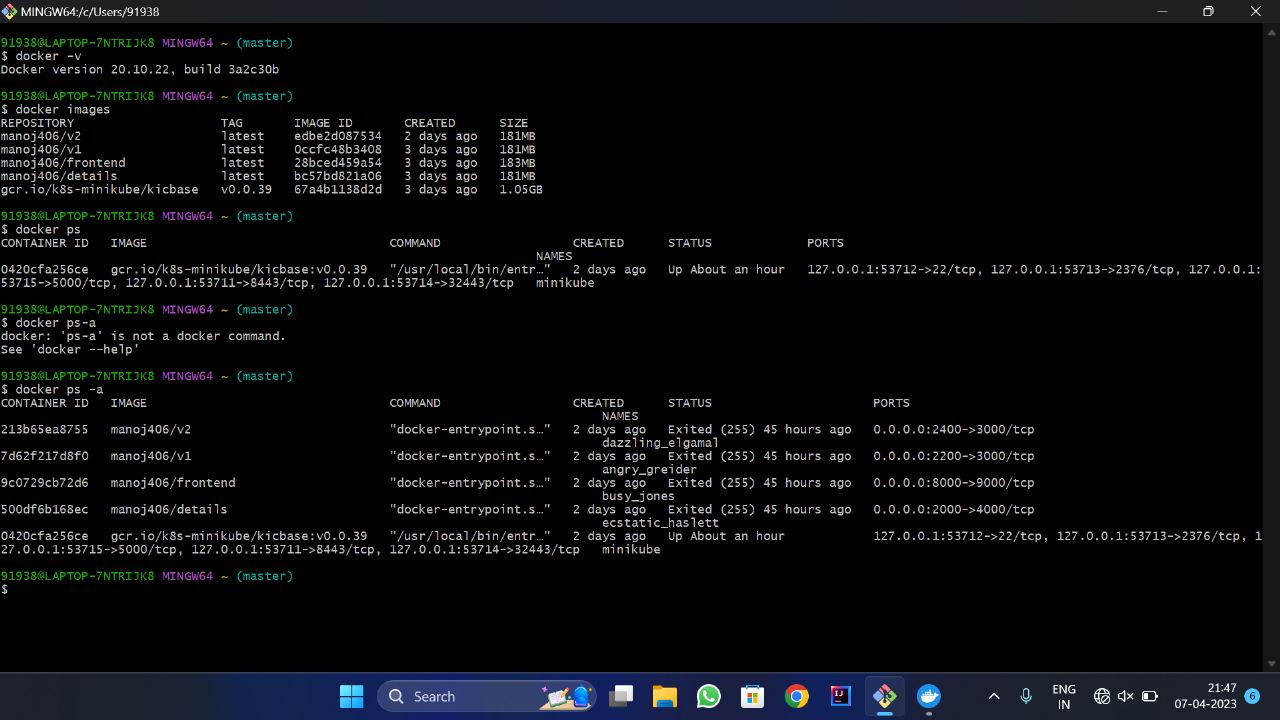
$ docker build -t "username/imagename" 🡪to create image with the files we created

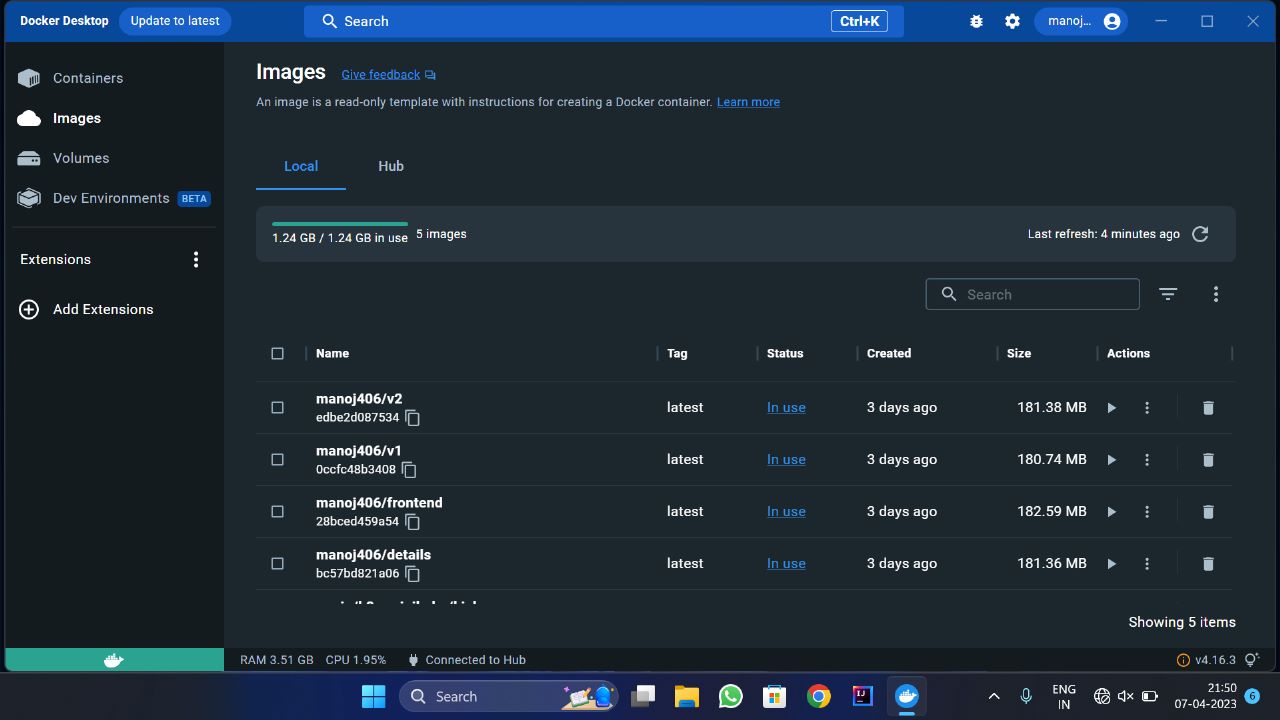
$ "docker images"🡪 to see if that image is created or not.

$ docker run -d -p leftport:rightport(fixed) "image name"🡪 to pull(if not there) and run the container

$ docker push "image name" 🡪to push the image to docker hub

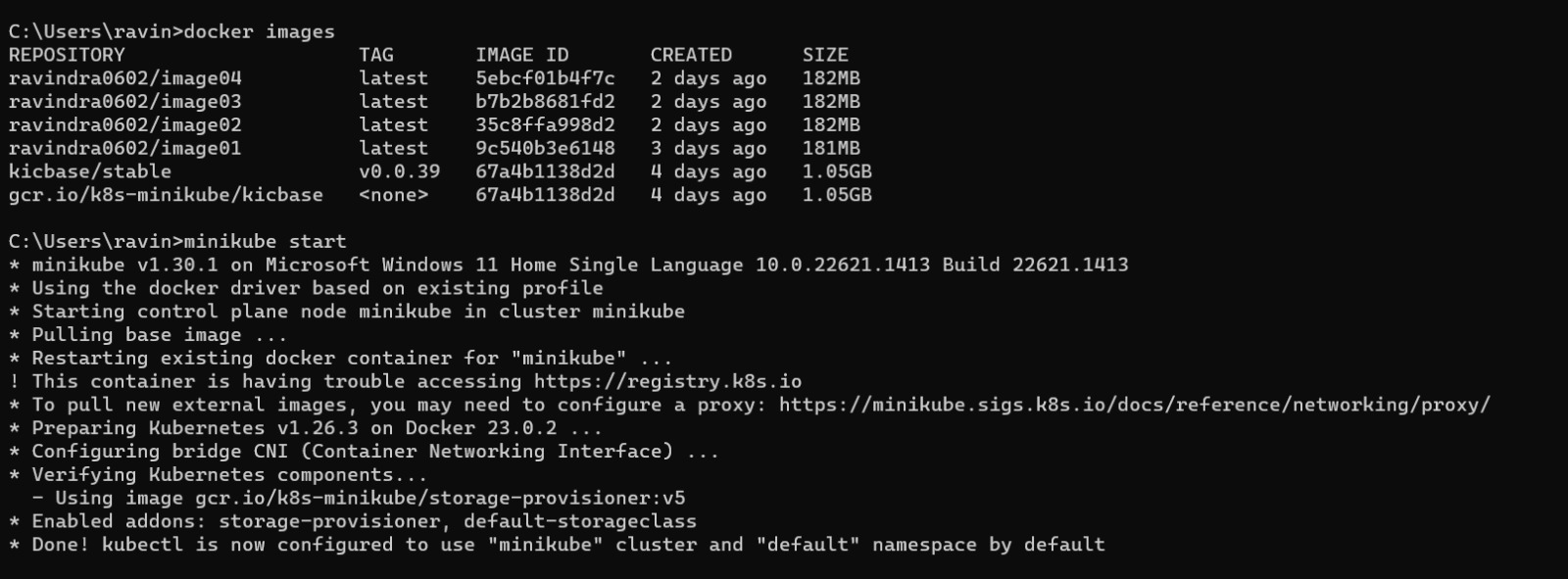
the image name must be like "username/imagename" in order push in our repository





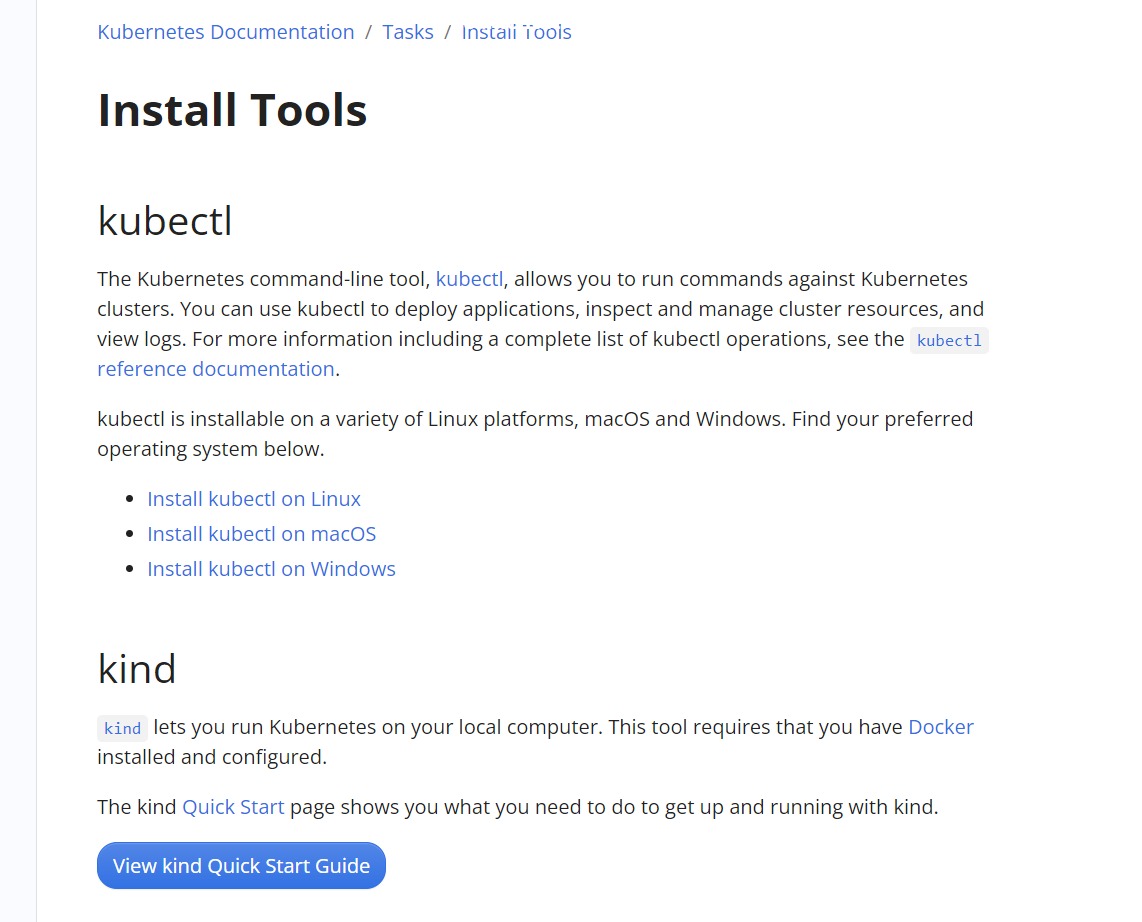
3. INSTALLATION OF MINIKUBE

* Install minikube by using this link [minikube start | minikube (k8s.io)](https://minikube.sigs.k8s.io/docs/start/)
* $ minikube start 🡪this command is used start the minikube while doing this command you must firstly open docker desktop



4. INSTALLATION OF KUBECTL

$winget install -e --id Kubernetes.kubectl🡪this command is used to install kubectl.



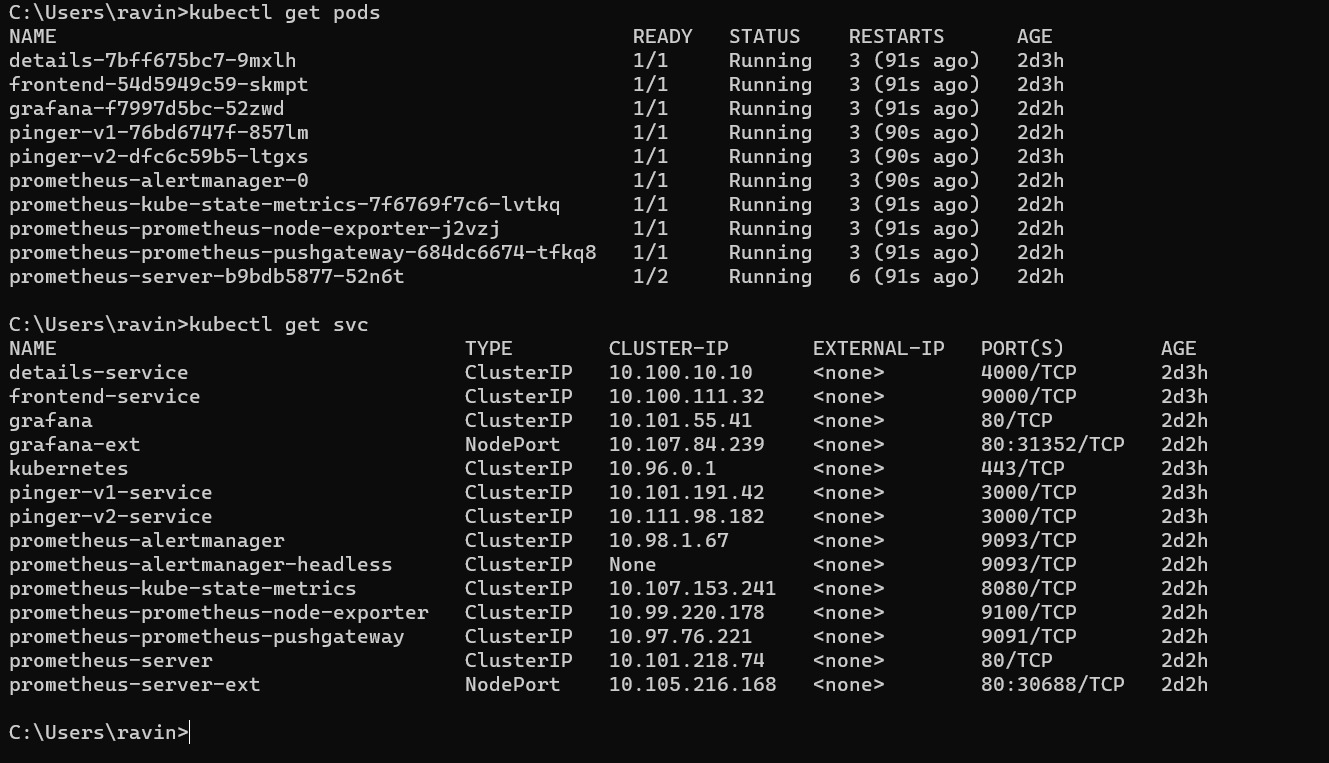
5. COMMANDS FOR MINIKUBE

$ kubectl apply -f details.yaml

$ kubectl get pods

$ kubectl get deploy

$ kubectl get service (COMTAINER CREATING ---> RUNNING)



# 6. INSTALLATION OF CHOCOLETY AND HELM

CHOCOLETY:

Chocolatey is most commonly used to get software and software updates installed on many machines at once. Like many DevOps tools, Chocolatey is used to automate repetitive tasks.

HELM:

Helm is a tool that automates the creation, packaging, configuration, and deployment of Kubernetes applications by combining your configuration files into a single reusable package

Commands:

INSTALL choco and helm in powershell

$ Get-ExecutionPolicy (output--> Restricted)

$ Set-ExecutionPolicy AllSigned or Set-ExecutionPolicy Bypass -Scope Process

$ Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))

$ choco

$choco install kubernetes-helm 🡪helm is installed.

$helm version 🡪this command is used to get present working version of helm on your desktop

7. INSTALLATION OF PROMETHEUS

PROMETHEUS:

Prometheus collects rich metrics and provides a powerful querying language; Grafana transforms metrics into meaningful visualizations

Commands:

$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

$ helm search repo prometheus

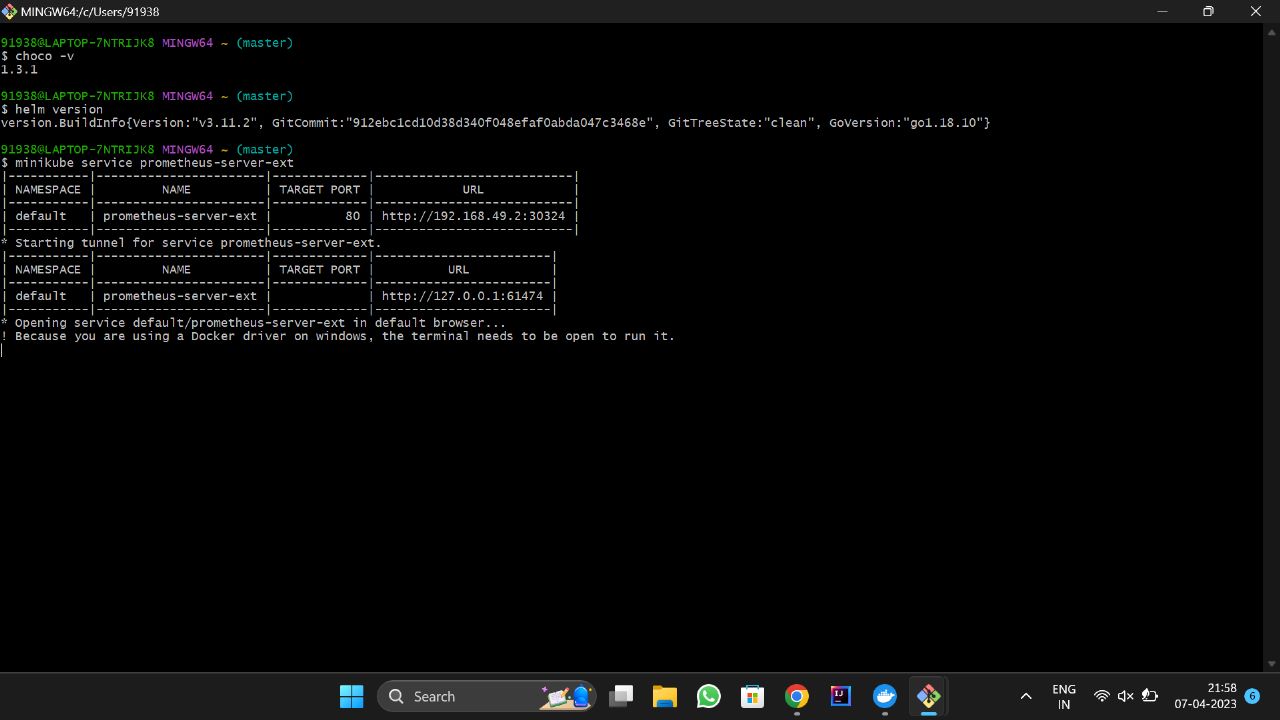
$ helm repo update

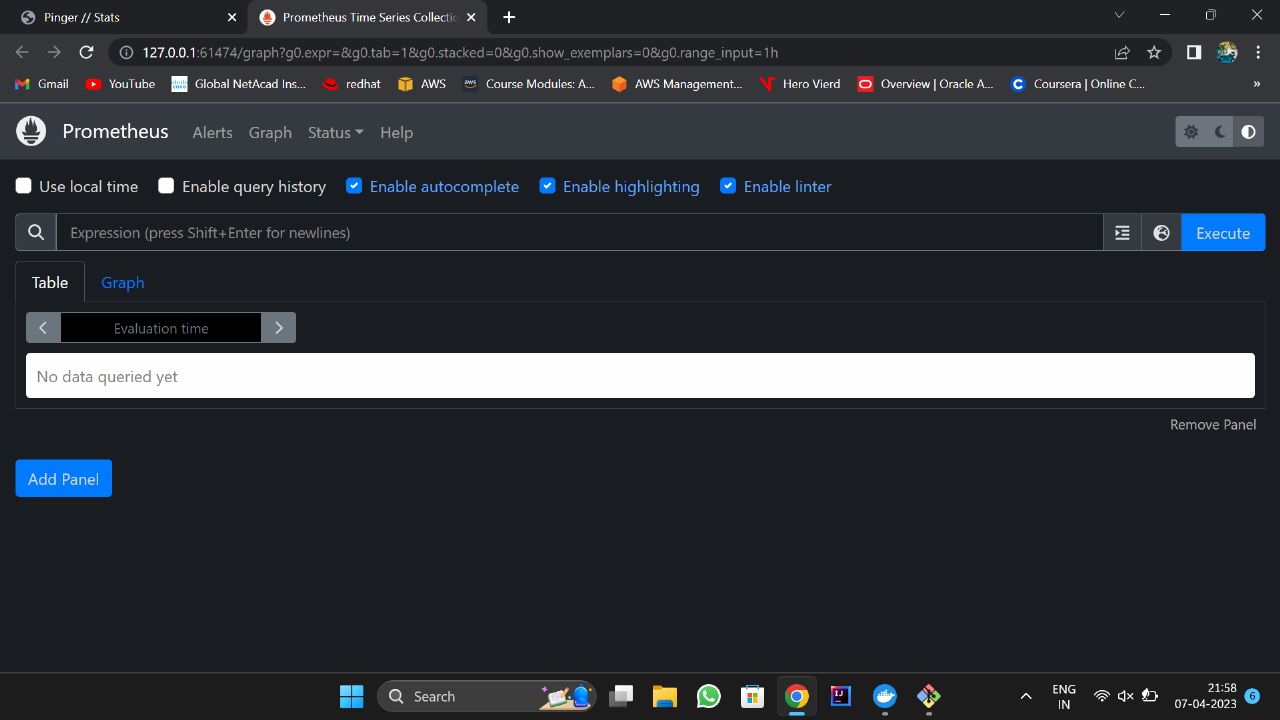
$ helm install prometheus prometheus-community/prometheus

$ kubectl get all

$ kubectl get svc

$ kubectl expose service prometheus-server --type=NodePort --target-port=9090 --name=prometheus-server-ext

$minikube service prometheus-server-ext



# 8 . INSTALLATION OF GRAFANA

GRAFANA:

Grafana is an open-source data visualization and monitoring platform that allows users to query, visualize, and alert on metrics and logs from a variety of data sources

Commands:

$ helm repo add grafana https://grafana.github.io/helm-charts

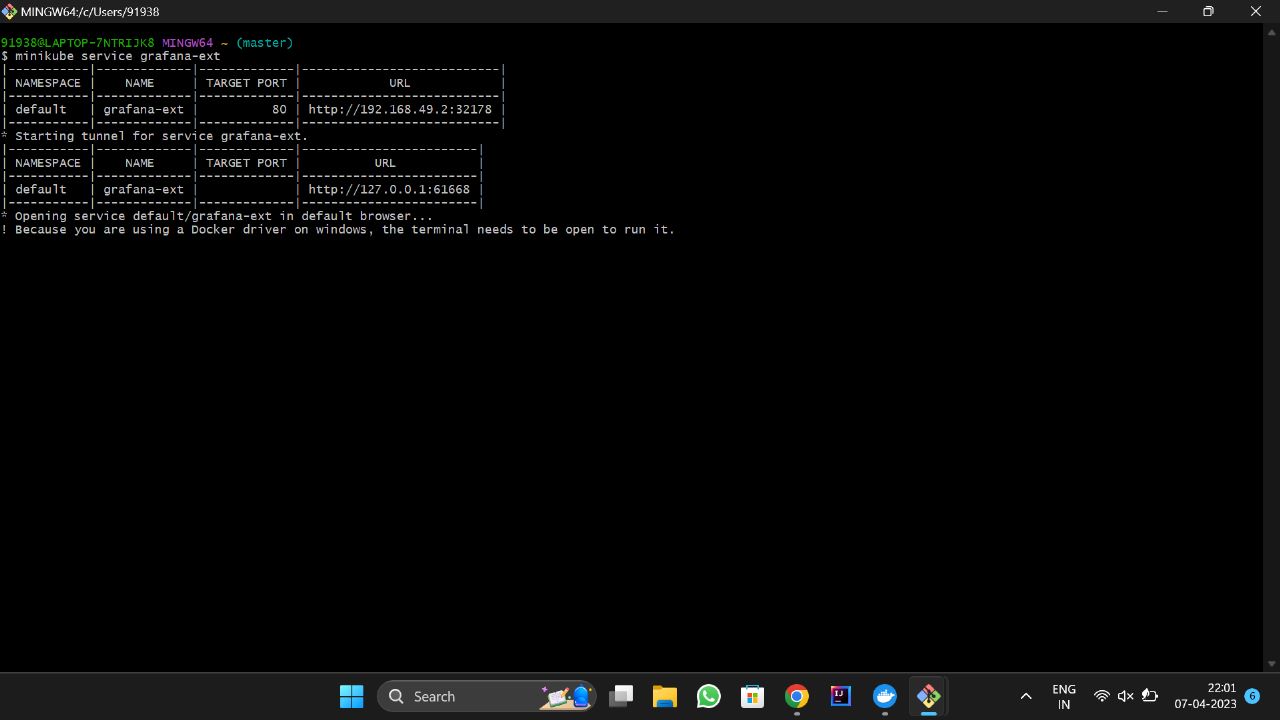
$ helm repo update

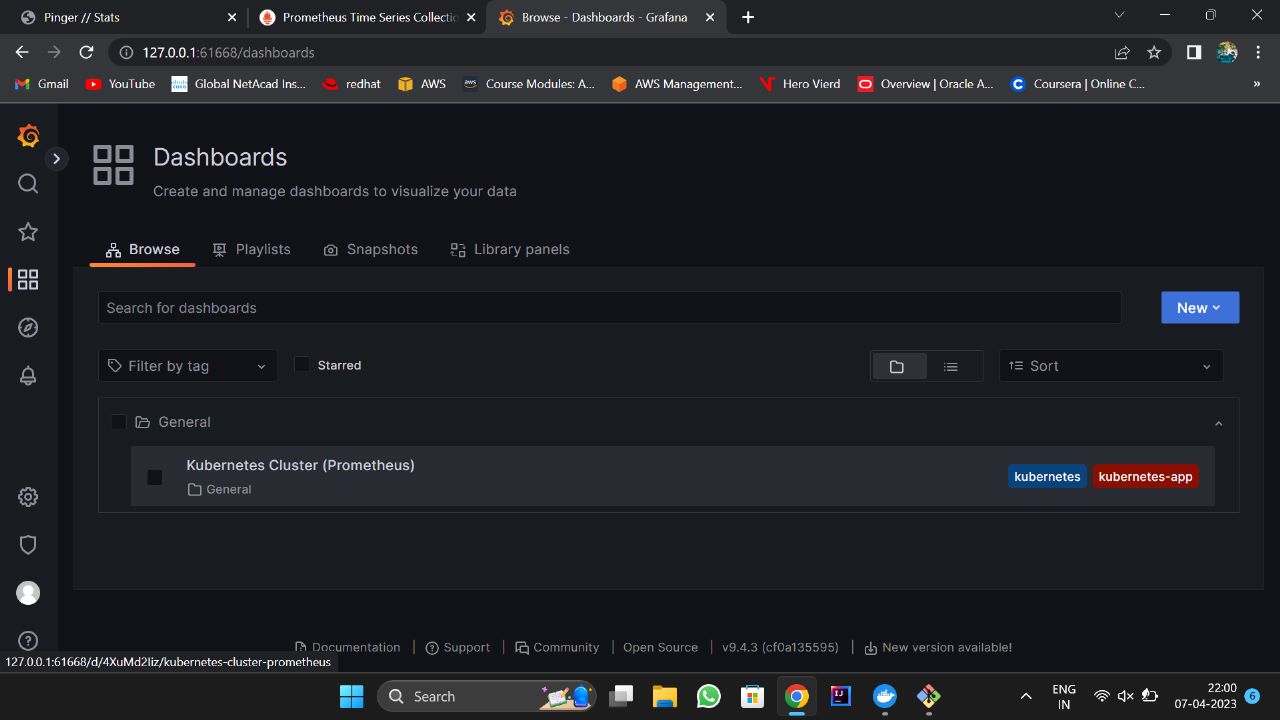
$ helm install grafana grafana/grafana

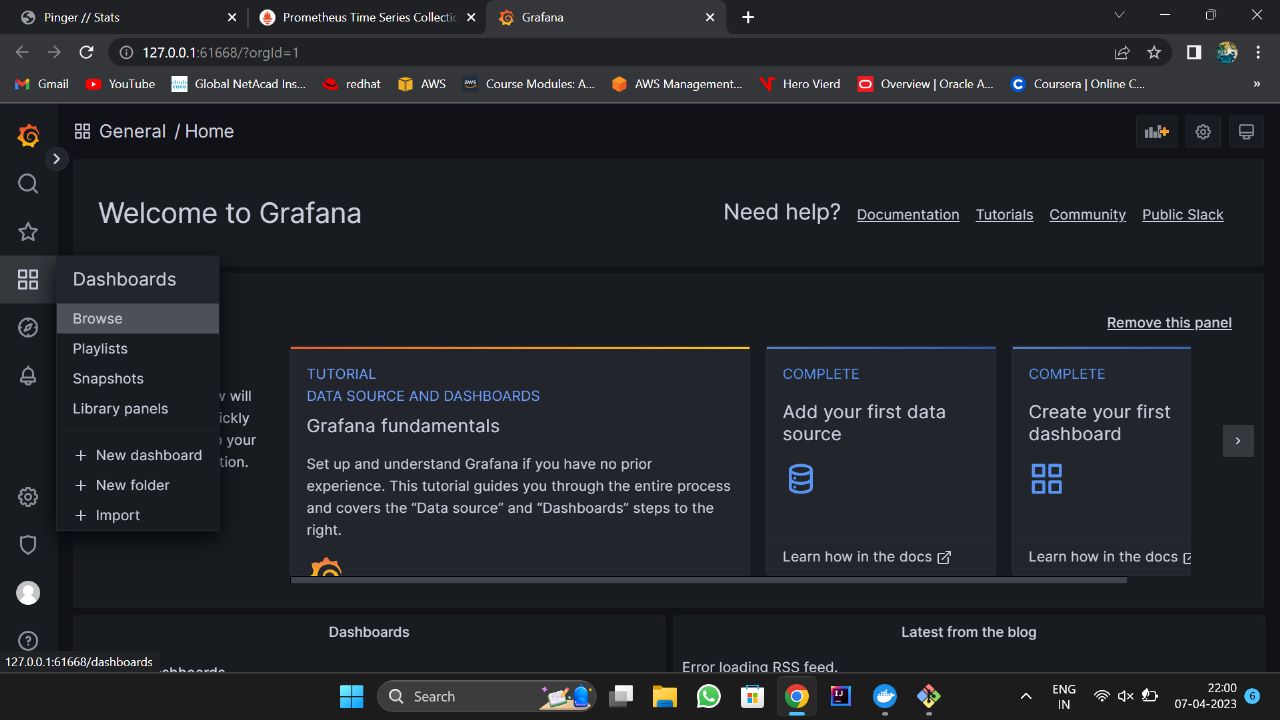
$ helm list

$ kubectl expose service grafana --type=NodePort --target-port=3000 --name=grafana-ext

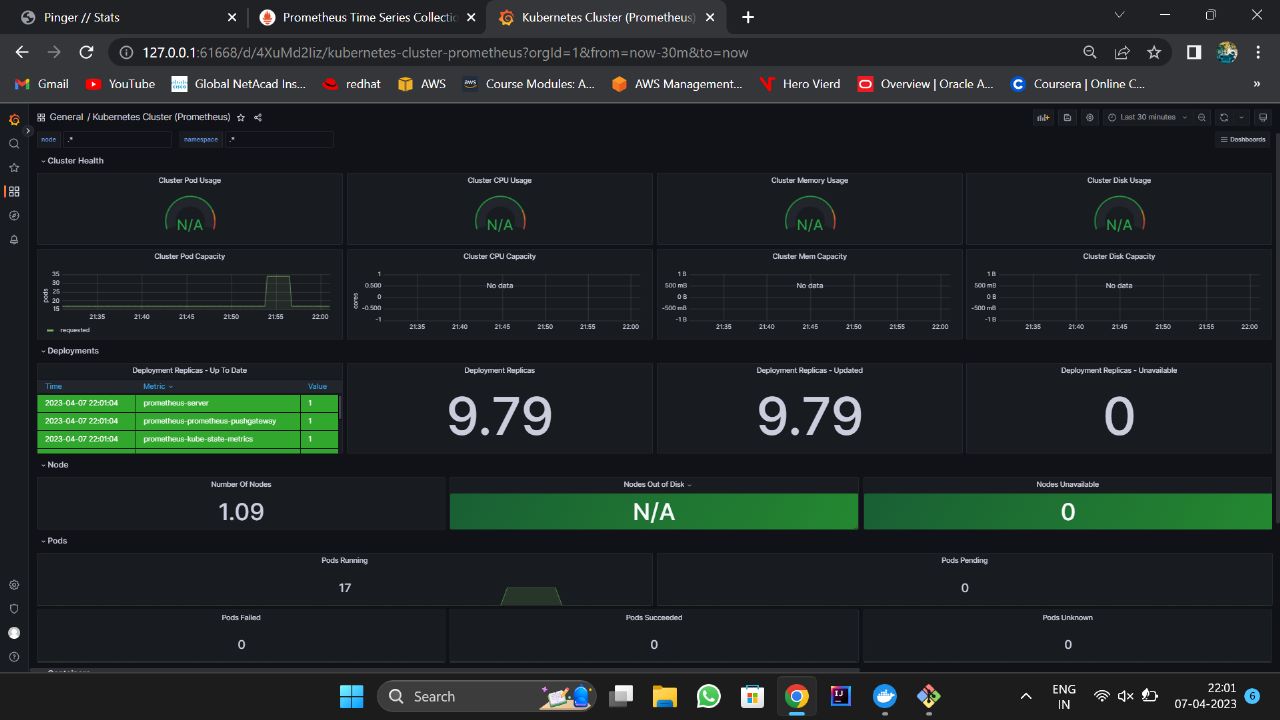
$ minikube service grafana-ext







9. MONITORING THE CONTAINERS



## 10. Outcome and Conclusion

CONCLUSION:

Overall, the successful completion of this project demonstrates a solid understanding of Kubernetes, containerization, monitoring, and visualization. The use of Minikube allowed for the creation of a local environment, which is useful for testing and development purposes. The deployment of multiple pods with a web application container in each pod allowed for scalability and redundancy. The deployment of Prometheus and Grafana helped to monitor the web application containers and provided a visualization of metrics to help identify any issues. By observing metrics and graphs in Grafana, it is possible to optimize the performance of the web application containers and ensure they are running efficiently. Overall, this project highlights the importance of monitoring and visualization in ensuring the reliability and scalability of Kubernetes environments.

OUTCOMES:

* Successfully created a node in Minikube, which allows for local development and testing of Kubernetes environments.
* Deployed four pods, each containing a web application container, which enables scaling and load balancing of the web application.
* Deployed Prometheus and Grafana, which provide monitoring and visualization of metrics for the web application containers.
* Observed metrics and graphs in Grafana, which helps identify issues and optimize the web application's performance.