Introduction:

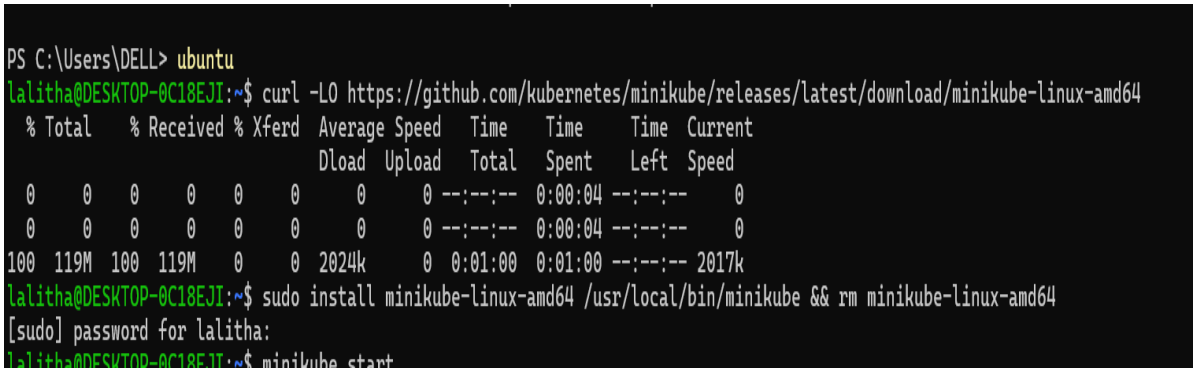
Kubernetes is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications. Originally developed by Google and now maintained by the Cloud Native Computing Foundation (CNCF), Kubernetes helps businesses manage complex applications efficiently across clusters of machines.

It enables developers to deploy applications using Pods, which are the smallest deployable units, and scale them based on demand. Kubernetes ensures high availability by distributing workloads and automatically recovering failed containers. Key features include service discovery, load balancing, storage orchestration, self-healing, and automated rollouts/rollbacks.

Kubernetes supports multiple container runtimes like Docker, containerd, and CRI-O, and integrates with cloud providers such as AWS, Azure, and Google Cloud. It allows organizations to adopt microservices architectures and DevOps practices seamlessly. With its declarative approach, Kubernetes simplifies infrastructure management and improves application reliability.

Installing Minikube

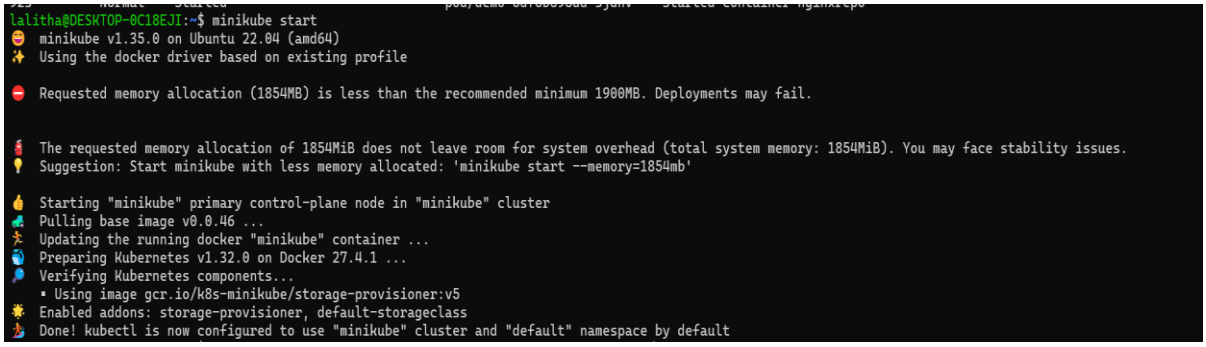
The command curl -LO https://github.com/kubernetes/minikube/releases/latest/download/minikube-linux-amd64 downloads the latest Minikube binary for Linux (AMD64) from GitHub, following any redirects and saving the file with its original name. This allows you to set up and manage local Kubernetes clusters.



Starting Minikube

The minikube start command initializes a local Kubernetes cluster on your machine using Minikube. It downloads the necessary Kubernetes components and starts a virtual machine or container, depending on the environment, to run the cluster.

Once executed, it sets up the cluster, allowing you to deploy and manage Kubernetes workloads locally for development and testing purposes.

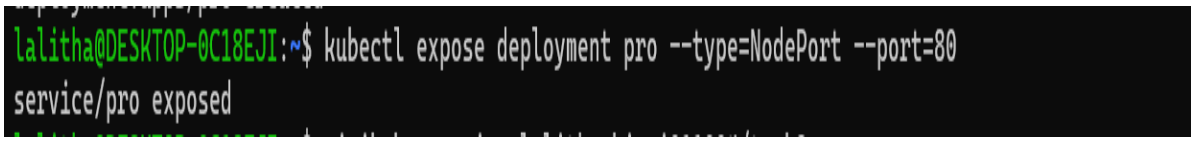


Deploying the image

The command kubectl create deployment pro --image=lalithambigai011004/task2 --port=80 creates a new Kubernetes deployment named "pro". It uses the Docker image lalithambigai011004/task2 from a container registry (such as Docker Hub) to create the deployment. The --port=80 option exposes port 80 within the deployment, allowing the application to listen on that port. Once executed, this command deploys the specified containerized application within your Kubernetes cluster.

Exposing the deployed image

The command kubectl expose deployment pro --type=NodePort --port=80 exposes the "pro" deployment as a Kubernetes service. It creates a service of type NodePort, which makes the application accessible from outside the cluster on a specific port (usually in the range 30000-32767). The --port=80 option specifies that the service will forward traffic on port 80 to the deployment, allowing external access to the application via the NodePort. Once executed, this command enables external users to access the application through the IP of any node in the cluster and the allocated port.



Getting Services

The command kubectl get services (or kubectl get svc) lists all the services running in the current Kubernetes cluster. It displays the services' names, types (e.g., ClusterIP, NodePort, LoadBalancer), cluster-internal IP addresses, external IP addresses (if applicable), and the ports the services are exposed on. This command helps you monitor and manage the services available in your Kubernetes cluster.

```
lalitha@DESKTOP-0C18EJI:~$ kubectl get services
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
demo         NodePort    10.96.191.216   <none>        80:32011/TCP   7m11s
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        38m
pro          NodePort    10.98.210.185   <none>        80:31467/TCP   3m35s
task2        NodePort    10.98.221.141   <none>        80:31643/TCP   10s
```

Listing the Services

The command minikube service list lists all the services that are currently exposed in your Minikube cluster. It shows the name of each service, its URL, and the corresponding port for accessing the service externally. This command helps you quickly find the accessible endpoints for your services running in the Minikube cluster.

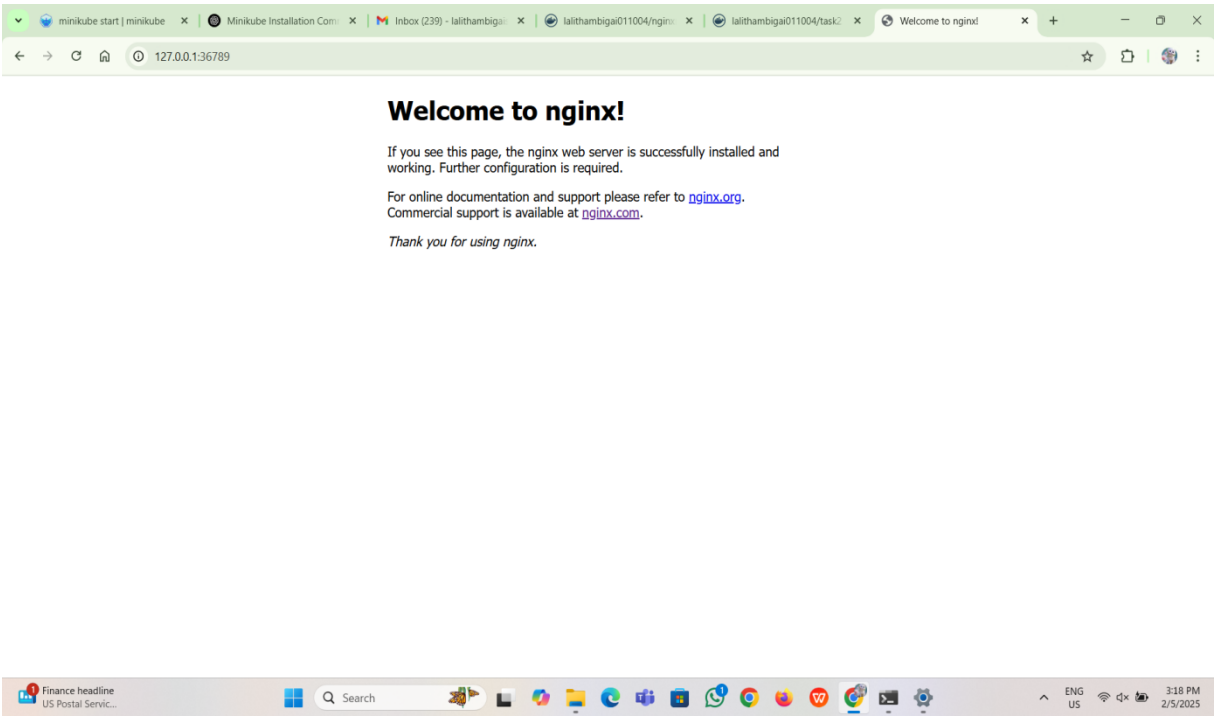
```
lalitha@DESKTOP-0C18EJI:~$ minikube service list
❗  Executing "docker container inspect minikube --format={{.State.Status}}" took an unusually long time: 2.661398014s
💡  Restarting the docker service may improve performance.
|-------------|------------|--------------|-----|
|  NAMESPACE  |    NAME    | TARGET PORT  | URL |
|-------------|------------|--------------|-----|
| default     | demo       |           80 |     |
| default     | kubernetes | No node port |     |
| default     | pro        |           80 |     |
| default     | task2      |           80 |     |
| kube-system | kube-dns   | No node port |     |
|-------------|------------|--------------|-----|
```

Getting URL:

The command minikube service task2 --url retrieves the external URL for accessing the task2 service in your Minikube cluster. This URL includes the IP address and port where the service is exposed, allowing you to access it from your browser or other clients outside the cluster. This is useful when you want to quickly access a service running in Minikube without manually looking up the URL or port.

```
lalitha@DESKTOP-0C18EJI:~$ minikube service task2 --url
http://127.0.0.1:36789
❗  Because you are using a Docker driver on linux, the terminal needs to be open to run it.
```

Output:



Conclusion

In conclusion, the minikube service task2 --url command provides the external URL to access the task2 service in your Minikube cluster. It simplifies the process of finding the service's endpoint by automatically retrieving the URL, making it easier to access the service externally. This functionality is particularly useful when developing and testing Kubernetes applications locally, offering seamless access to deployed services without needing to manually identify the exposed IP and port.