

Kubernetes Deployment of Flask Backend & Express Frontend using Minikube

1. Introduction

This project demonstrates a complete end-to-end deployment of a full-stack application (Flask backend + Express frontend) on a local Kubernetes cluster using Minikube.

Both components are containerized using Docker, the images are built inside Minikube's Docker engine, and Kubernetes deployments & services manage the application lifecycle. The frontend communicates reliably with the backend via Kubernetes internal networking.

2. Tools & Technologies Used

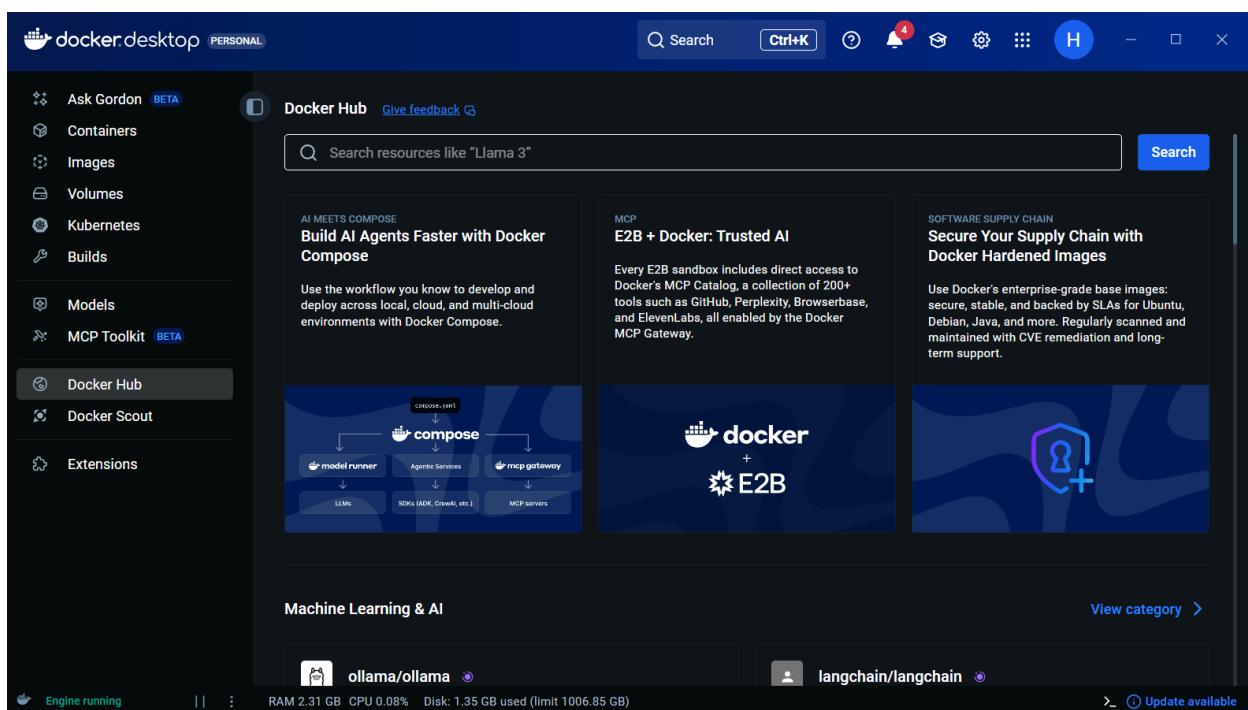
- Kubernetes (Minikube)
 - Docker Desktop
 - kubectl
 - Flask (Python backend)
 - Express.js (Node frontend)
 - YAML manifests (Deployments & Services)
-

3. Environment Setup

3.1 Docker Desktop Running

Description:

This screenshot confirms that **Docker Desktop is running**, which is required because Minikube uses Docker as its container runtime. With Docker active, Minikube can create containers, store images, and execute deployments locally.



3.2 Starting Minikube Cluster

Description:

This screenshot shows Minikube downloading and preparing Kubernetes components such as:

- kube-apiserver
- kube-scheduler
- kube-controller-manager
- Container Networking Interface (CNI)

The final message states:

"kubectl is now configured to use 'minikube' cluster and 'default' namespace."

This confirms that Kubernetes is fully installed and ready to run workloads.

```
PS C:\Users\rites> kubectl config get-clusters
  Downloading Kubernetes v1.34.0 preload ...
    > gcr.io/k8s-minikube/kicbase...: 1.81 MiB / 488.52 MiB 0.37% 131.84 KiB      > preloaded-images-k8s-v18...
    ...: 15.83 MiB / 337.07 MiB 4.70% 1.01      > gcr.io/k8s-minikube/kicbase...: 1.85 MiB / 488.52 MiB 0.38% 1...
    > preloaded-images-k8s-v18-v1...: 15.89 MiB / 337.07 MiB 4.71% 1      > gcr.io/k8s-minikube/kicbase...: 1...
    MiB / 488.52 MiB 0.38% 131.8      > preloaded-images-k8s-v18-v1...: 15.92 MiB / 337.07 MiB 4.72% 986.      > g...
    io/k8s-minikube/kicbase...: 1.85 MiB / 488.52 MiB 0.38% 126.7      > p      > gcr.io/k8s-minikube/kicbase...: 1...
    5 MiB / 488.52 MiB 0.38% 126.77      > pr      > gcr.io/k8s      > preloaded-images-k8s-v18-v      > gcr.io/k8s-mi...
    e/kicbase...: 1.89 MiB / 488.52      > pr      > gc      > pr      > gcr.io/k8s-minikub      > preloaded-images-k8s-v...
    > gc      > gcr.io/k8s      > preloaded-images-k8s-v18-v1...: 1      > gcr.io/k8s-minikube/kicbase...: 2      >...
    eloaded-      > gcr.io/k8s      > gc      > gc      > pr      > gc      > preloaded-      > gcr.io/k8s-minikube/kicba...
    loaded-images-k8s-v18-v1...: 20      > gcr.io/k8s-minikube/kicbase...: 47.72 MiB      > preloaded-images-k8s-v...
    v1...:      > preloa      > gcr.io/k8s-minikube/kicbase...: 50.49 MiB / 488.52 Mi      > preloaded-images-k...
    r.io      > preloa      > gcr.io      > preloa      > gcr.io      > preloa      > preloa      > g...
    > preloaded-images-k8s-v18-v1...: 337.07 MiB / 337.07 MiB 100.00% 2.26 Mi
    > gcr.io/k8s-minikube/kicbase...: 488.52 MiB / 488.52 MiB 100.00% 1.02 Mi
    🔥 Creating docker container (CPUs=2, Memory=3900MB) ...
    ⚠️ Failing to connect to https://registry.k8s.io/ from inside the minikube container
    💡 To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/referer...
    networking/proxy/
    🎉 Preparing Kubernetes v1.34.0 on Docker 28.4.0 ...
    🏛 Configuring bridge CNI (Container Networking Interface) ...
    📑 Verifying Kubernetes components...
      - Using image gcr.io/k8s-minikube/storage-provisioner:v5
    🌟 Enabled addons: default-storageclass, storage-provisioner
    🎉 Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
PS C:\Users\rites>
```

```
PS C:\Users\rites> & minikube -p minikube docker-env | Invoke-Expression
PS C:\Users\rites> docker images
REPOSITORY          TAG      IMAGE ID   CREATED     SIZE
registry.k8s.io/kube-apiserver   v1.34.0   9e00... 3 months ago  88MB
registry.k8s.io/kube-proxy       v1.34.0   d508693... 3 months ago  71.9MB
registry.k8s.io/kube-scheduler   v1.34.0   461c... 3 months ago  52.8MB
registry.k8s.io/kube-controller-manager   v1.34.0   a0af7... 3 months ago  74.9MB
registry.k8s.io/etcfd            3.6.4-0   5f1f... 4 months ago  195MB
registry.k8s.io/pause             3.10.1    cd0f... 5 months ago  736kB
registry.k8s.io/coredns/coredns  v1.12.1   5269... 8 months ago  75MB
gcr.io/k8s-minikube/storage-provisioner  v5       6e38r... 4 years ago  31.5MB
PS C:\Users\rites>
```

3.3 Viewing Internal Minikube Docker Images

Description:

This section shows multiple screenshots:

1. Minikube startup logs

These confirm that cluster provisioning completed successfully.

2. docker images

Displays Kubernetes system images already present inside the Minikube Docker engine (e.g., kube-apiserver, etcd, scheduler).

No custom images are present yet, confirming a clean environment.

3. `kubectl get nodes`

Shows the Minikube node in **Ready** state, meaning it can now schedule pods.

```
| gcr.io/k8s-minikube/storage-provisioner    v5      6e
| PS C:\Users\rites> kubectl get nodes
|   NAME      STATUS    ROLES      AGE     VERSION
|   minikube   Ready     control-plane   5m40s   v1.34.0
| PS C:\Users\rites>
```

4. Building Docker Images inside Minikube

Minikube uses an internal Docker engine.

We must switch Docker to use Minikube's engine:

```
& minikube -p minikube docker-env | Invoke-Expression
```

4.1 Backend Docker Image Build (Flask)

Description:

The screenshot confirms the successful build of the **Flask backend** Docker image (`flask-backend:1.0`).

Key validations visible in the screenshot:

- Requirements installed via `requirements.txt`
- Build layers executed cleanly
- Image exported and tagged correctly

This ensures the backend image is now ready for Kubernetes deployment.

```

PS C:\Users\rites\OneDrive\Desktop\ALL Project\AWS\flask-express-docker-with-AWS-EC2> docker build -t flask-backend:1.0 ./backend
[+] Building 64.2s (12/12) FINISHED
  => [internal] load build definition from Dockerfile
  => transferring dockerfile: 514B
  => [internal] load metadata for docker.io/library/python:3.11-slim
  => [auth] library/python:pull token for registry-1.docker.io
  => [internal] load .dockerignore
  => transferring context: 2B
  => [1/6] FROM docker.io/library/python:3.11-slim@sha256:193fdd0bbcb3d2ae612bd6cc3548d2f7c78d65b549fc当地
  => resolve docker.io/library/python:3.11-slim@sha256:193fdd0bbcb3d2ae612bd6cc3548d2f7c78d65b549fc当地
  => sha256:b3dd773c329649f22e467ae63d1c612a039a0559dec99ffb9ada904ab5c60c55 23.5s
  => sha256:193fdd0bbcb3d2ae612bd6cc3548d2f7c78d65b549fc当地 0.1s
  => sha256:c4116d4d7ea9320db352f6516001262753529edf1e20b2c6609a6b9a49cc6be4 4.0s
  => sha256:040af88f5bce9e9239b7e739e86304d26964d1d55ada56b9297a3d891e91634d 0.0s
  => sha256:0e4bc2bd6656e6e004e3c749af70e5650bac2258243eb0949dea51cb8b7863db 0.0s
  => sha256:22b63e76fde1e200371ed9f3cee91161d192063bc当地65c9ab6bf63819810a974 6.3s
  => sha256:1771569cc1299abc9cc762fc4419523e721b11a3927ef968ae63ba0a488f2da 1.6s
  => sha256:0e4bc2bd6656e6e004e3c749af70e5650bac2258243eb0949dea51cb8b7863db 2.2s
  => sha256:22b63e76fde1e200371ed9f3cee91161d192063bc当地65c9ab6bf63819810a974 8.8s
  => sha256:b3dd773c329649f22e467ae63d1c612a039a0559dec99ffb9ada904ab5c60c55 0.9s
  => sha256:1771569cc1299abc9cc762fc4419523e721b11a3927ef968ae63ba0a488f2da 5.7s
  => sha256:1771569cc1299abc9cc762fc4419523e721b11a3927ef968ae63ba0a488f2da 0.0s
  => [internal] load build context
  => transferring context: 7.36kB
  => [2/6] RUN apt-get update && apt-get install -y wget && rm -rf /var/lib/apt/lists/*
  => [3/6] WORKDIR /app
  => [4/6] COPY requirements.txt .
  => [5/6] RUN pip install --no-cache-dir -r requirements.txt
  => [6/6] COPY . .
  => exporting to image
  => exporting layers
  => writing image sha256:ee0057cd35ea554d8aac05fe98d01b955542f45bae609da0c9844db23ef02a6f
  => naming to docker.io/library/flask-backend:1.0
PS C:\Users\rites\OneDrive\Desktop\ALL Project\AWS\flask-express-docker-with-AWS-EC2> |

```

4.2 Frontend Docker Image Build (Express)

Description:

This screenshot confirms that the **Express frontend image** (`express-frontend:1.0`) was built successfully.

Node dependencies installed correctly, application files were copied, and the final image is stored inside the Minikube Docker environment.

```

  => [internal] load build context
  => transferring context: 5.58MB
  => [2/5] WORKDIR /app
  => [3/5] COPY package*.json .
  => [4/5] RUN npm install --production
  => [5/5] COPY . .
  => exporting to image
  => exporting layers
  => writing image sha256:3759313ad08d273478400fc2f12eda012cacb41e7c0b0f347a97dc4290298563
  => naming to docker.io/library/express-frontend:1.0
PS C:\Users\rites\OneDrive\Desktop\ALL Project\AWS\flask-express-docker-with-AWS-EC2>

```

4.3 Verify Images Inside Minikube

Description:

Both newly built images:

- `flask-backend:1.0`
- `express-frontend:1.0`

now appear alongside Kubernetes system images.

This proves that Minikube's Docker environment is being used correctly and images are available for deployments without needing a remote registry.

```
L Project\AWS\flask-express-docker-with-AWS-EC2"
PS C:\Users\rites\OneDrive\Desktop\ALL Project\AWS\flask-express-docker-with-AWS-EC2> kubectl apply -f k8s-manifests/flask-b
ackend-deployment.yaml
deployment.apps/flask-backend-deployment created
PS C:\Users\rites\OneDrive\Desktop\ALL Project\AWS\flask-express-docker-with-AWS-EC2> kubectl apply -f k8s-manifests/flask-b
ackend-service.yaml
service/flask-backend-service created
PS C:\Users\rites\OneDrive\Desktop\ALL Project\AWS\flask-express-docker-with-AWS-EC2>
```

5. Kubernetes Deployments

5.1 Backend Deployment + Service

Description:

The screenshot confirms:

- `flask-backend-deployment` created
- `flask-backend-service` (ClusterIP) created

The ClusterIP exposes port **5000** internally so the frontend can communicate with the backend through Kubernetes networking.

5.2 Frontend Deployment + Service

Description:

The screenshot shows successful creation of:

- `express-frontend-deployment`
- `express-frontend-service` (NodePort)

The NodePort service exposes the frontend externally so it can be accessed through Minikube.

```
PS C:\Users\rites\OneDrive\Desktop\ALL Project\AWS\flask-express-docker-with-AWS-EC2> kubectl apply -f k8s-manifests/flask-backend-deployment.yaml
deployment.apps/flask-backend-deployment created
PS C:\Users\rites\OneDrive\Desktop\ALL Project\AWS\flask-express-docker-with-AWS-EC2> kubectl apply -f k8s-manifests/flask-backend-service.yaml
service/flask-backend-service created
PS C:\Users\rites\OneDrive\Desktop\ALL Project\AWS\flask-express-docker-with-AWS-EC2> kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
flask-backend-deployment-6859bc67d9-rpt6p  1/1     Running   0          53s
PS C:\Users\rites\OneDrive\Desktop\ALL Project\AWS\flask-express-docker-with-AWS-EC2> kubectl apply -f k8s-manifests/express-frontend-deployment.yaml
deployment.apps/express-frontend-deployment created
PS C:\Users\rites\OneDrive\Desktop\ALL Project\AWS\flask-express-docker-with-AWS-EC2> kubectl apply -f k8s-manifests/express-frontend-service.yaml
service/express-frontend-service created
PS C:\Users\rites\OneDrive\Desktop\ALL Project\AWS\flask-express-docker-with-AWS-EC2>
```

6. Verify Kubernetes Workloads

6.1 Pods Running

Description:

This screenshot verifies that both deployed pods are in **Running** state:

- Express frontend pod
- Flask backend pod

This confirms that Kubernetes successfully pulled the images from Minikube's Docker engine and launched containers without errors.

```
-frontend-service.yaml
service/express-frontend-service created
PS C:\Users\rites\OneDrive\Desktop\ALL Project\AWS\flask-express-docker-with-AWS-EC2> kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
express-frontend-deployment-59cc4d8b57-9ww5t  1/1     Running   0          52s
flask-backend-deployment-6859bc67d9-rpt6p      1/1     Running   0          3m7s
PS C:\Users\rites\OneDrive\Desktop\ALL Project\AWS\flask-express-docker-with-AWS-EC2>
```

7. Accessing the Application

7.1 Opening the Frontend in Browser

Description:

When running:

```
minikube service express-frontend-service
```

Minikube creates a **tunnel** and exposes the service via a local URL such as:

```
http://127.0.0.1:60564
```

This proves that external access via NodePort is functioning.

```
NAME                               READY   STATUS    RESTARTS   AGE
express-frontend-deployment-59cc4d8b57-9ww5t   1/1     Running   0          52s
flask-backend-deployment-6859bc67d9-rpt6p      1/1     Running   0          3m7s
PS C:\Users\rites\OneDrive\Desktop\ALL Project\AWS\flask-express-docker-with-AWS-EC2> kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
express-frontend-service   NodePort   10.99.119.85  <none>        3000:30080/TCP  104s
flask-backend-service     ClusterIP  10.103.72.27   <none>        5000/TCP      3m59s
kubernetes         ClusterIP  10.96.0.1    <none>        443/TCP       30m
PS C:\Users\rites\OneDrive\Desktop\ALL Project\AWS\flask-express-docker-with-AWS-EC2> minikube service express-frontend-service
NAME          NAMESPACE   NAME          TARGET PORT   URL
default       default     express-frontend-service   3000          http://192.168.49.2:30080
star Starting tunnel for service express-frontend-service./_
NAME          NAMESPACE   NAME          TARGET PORT   URL
default       default     express-frontend-service   3000          http://127.0.0.1:60564
star Starting tunnel for service express-frontend-service.
star Opening service default/express-frontend-service in default browser...
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```

7.2 Application Working with Backend API

Description:

This browser screenshot confirms full functionality:

- Grade Checker sends user score → Backend returns JSON
- Student update form sends name + grade → Backend stores and responds

- “Show All Students” successfully retrieves data

This validates:

- ✓ Frontend ↔ Backend communication works
- ✓ Backend processes API requests
- ✓ Application operates entirely inside the Kubernetes cluster

This demonstrates complete end-to-end functionality.

Assignment 2 — Node (Frontend) → Flask (Backend)

Grade Checker

54

Get Grade

```
{
  "result": "Grade: F"
}
```

Student Grades (add/update)

bajaj

B

Add / Update Student

Show All Students

```
{
  "result": "bajaj -> B saved successfully."
}
```

8. Conclusion

This assignment successfully demonstrates:

- Containerization of multi-service applications
- Building Docker images inside Minikube
- Kubernetes deployments for backend and frontend
- Service communication using ClusterIP and NodePort

- Application availability via Minikube tunnels
- Working full-stack application inside Kubernetes

All expected outputs, screenshots, and configurations have been captured and documented for evaluation.