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

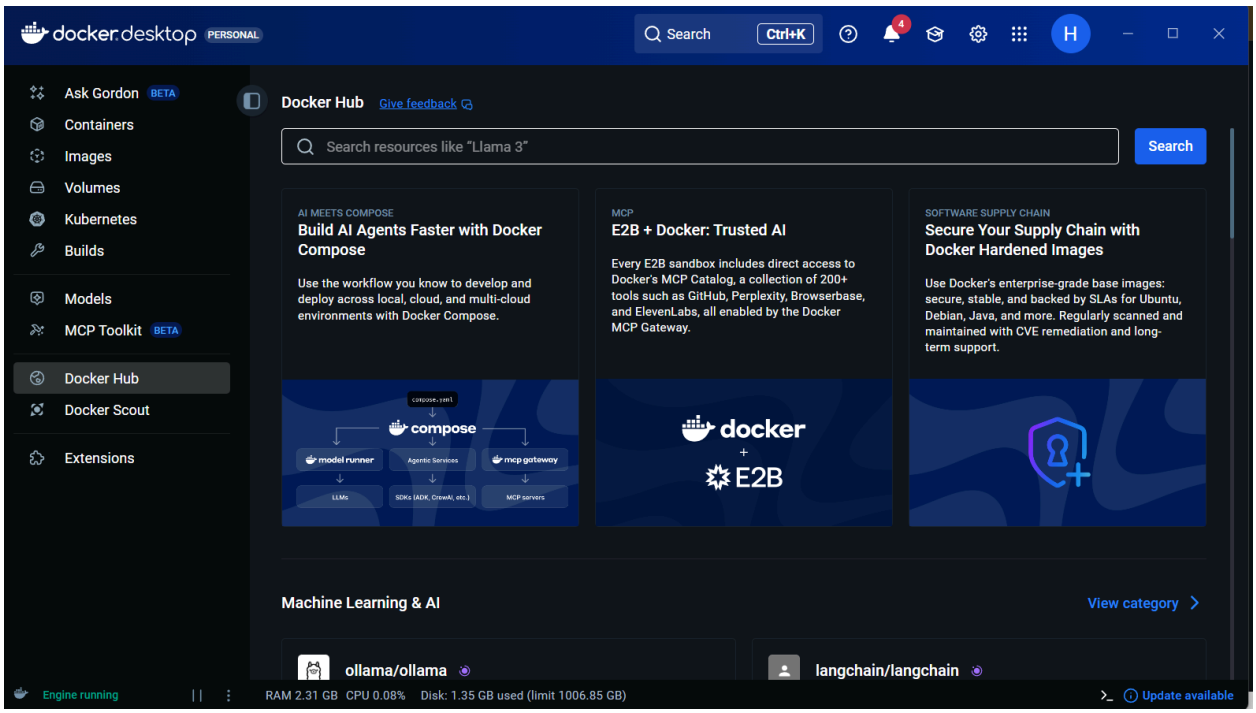
- Kubernetes (Minikube)
- Docker Desktop
- kubectl
- Flask (Python backend)
- Express.js (Node frontend)
- YAML manifests (Deployments & Services)

---

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

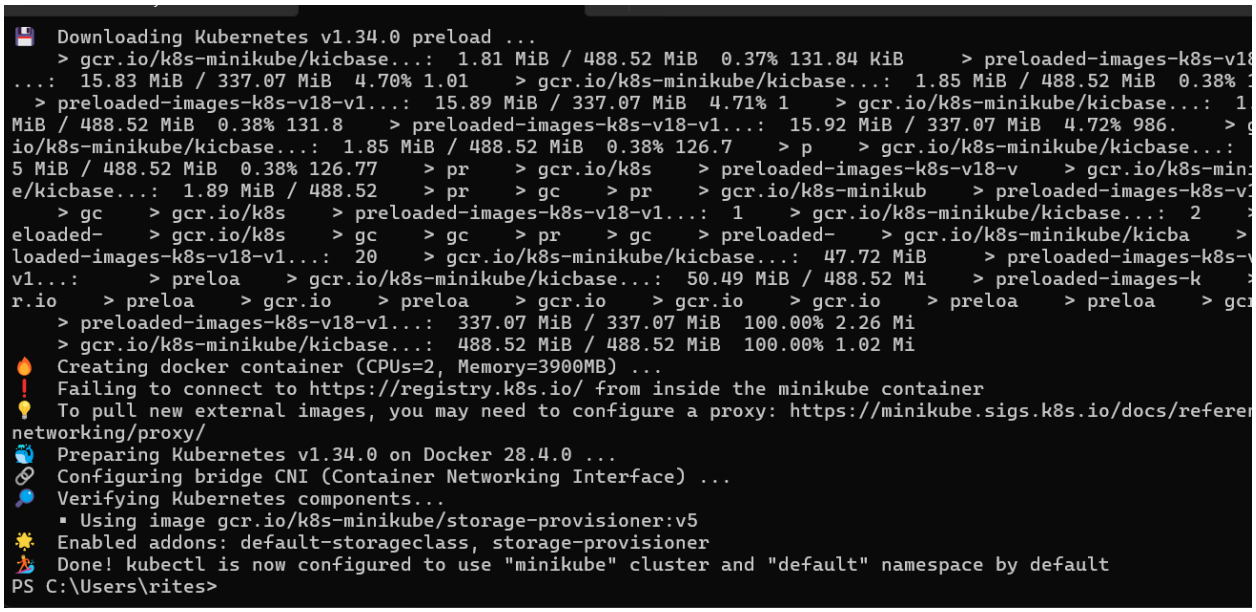
Description:
This screenshot shows Minikube downloading and preparing Kubernetes components such as:

- 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.



3.3 Viewing Internal Minikube Docker Images

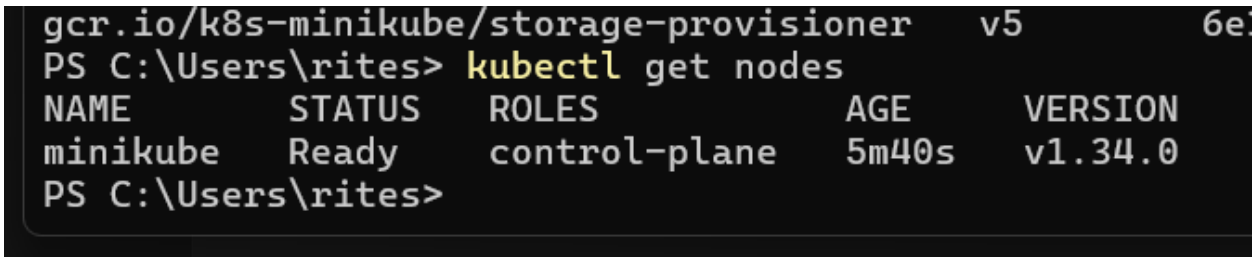
Description:
This section shows multiple screenshots:

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.



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-back
end:1.0 ./backend
[+] Building 64.2s (12/12) FINISHED                                                                 docker:default
 => [internal] load build definition from Dockerfile                                                         0.2s
 => => transferring dockerfile: 514B                                                                         0.1s
 => [internal] load metadata for docker.io/library/python:3.11-slim                                          4.0s
 => [auth] library/python:pull token for registry-1.docker.io                                                0.0s
 => [internal] load .dockerignore                                                                            0.1s
 => => transferring context: 2B                                                                              0.0s
 => [1/6] FROM docker.io/library/python:3.11-slim@sha256:193fdd0bbcb3d2ae612bd6cc3548d2f7c78d65b549fcaa8   23.5s
 => => resolve docker.io/library/python:3.11-slim@sha256:193fdd0bbcb3d2ae612bd6cc3548d2f7c78d65b549fcaa8a   0.1s
 => => sha256:b3dd773c329649f22e467ae63d1c612a039a0559dec99ffb9ada904ab5c60c55 14.36MB / 14.36MB             4.0s
 => => sha256:193fdd0bbcb3d2ae612bd6cc3548d2f7c78d65b549fcaa8af75624c47474444d 10.37kB / 10.37kB             0.0s
 => => sha256:c4116d4d7ea9320db352f6516001262753529edf1e20b2c6609a6b9a49cc6be4 1.75kB / 1.75kB               0.0s
 => => sha256:040af88f5bce9e9239b7e739e86304d26964d1d55ada56b9297a3d891e91634d 5.48kB / 5.48kB               0.0s
 => => sha256:0e4bc2bd6656e6e004e3c749af70e5650bac2258243eb0949dea51cb8b7863db 29.78MB / 29.78MB             6.3s
 => => sha256:22b63e76fde1e200371ed9f3cee91161d192063bcff65c9ab6bf63819810a974 1.29MB / 1.29MB               1.6s
 => => sha256:1771569cc1299abc9cc762fc4419523e721b11a3927ef968ae63ba0a4a88f2da 251B / 251B                   2.2s
 => => extracting sha256:0e4bc2bd6656e6e004e3c749af70e5650bac2258243eb0949dea51cb8b7863db                    8.8s
 => => extracting sha256:22b63e76fde1e200371ed9f3cee91161d192063bcff65c9ab6bf63819810a974                    0.9s
 => => extracting sha256:b3dd773c329649f22e467ae63d1c612a039a0559dec99ffb9ada904ab5c60c55                    5.7s
 => => extracting sha256:1771569cc1299abc9cc762fc4419523e721b11a3927ef968ae63ba0a4a88f2da                    0.0s
 => [internal] load build context                                                                            0.2s
 => => transferring context: 7.36kB                                                                          0.1s
 => [2/6] RUN apt-get update && apt-get install -y wget && rm -rf /var/lib/apt/lists/*                      23.4s
 => [3/6] WORKDIR /app                                                                                       0.2s
 => [4/6] COPY requirements.txt .                                                                            0.2s
 => [5/6] RUN pip install --no-cache-dir -r requirements.txt                                                11.4s
 => [6/6] COPY . .                                                                                           0.2s
 => exporting to image                                                                                       0.7s
 => => exporting layers                                                                                      0.6s
 => => writing image sha256:ee0057cd35ea554d8aac05fe98d01b955542f45bae609da0c9844db23ef02a6f                 0.0s
 => => naming to docker.io/library/flask-backend:1.0                                                         0.0s
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                                                                           13.8s
 => => transferring context: 5.58MB                                                                         13.7s
 => [2/5] WORKDIR /app                                                                                       1.7s
 => [3/5] COPY package*.json ./                                                                              0.7s
 => [4/5] RUN npm install --production                                                                       7.6s
 => [5/5] COPY . .                                                                                           1.0s
 => exporting to image                                                                                       1.2s
 => => exporting layers                                                                                      1.1s
 => => writing image sha256:3759313ad08d273478400fc2f12eda012cacb41e7c0b0f347a97dc4290298563                 0.0s
 => => naming to docker.io/library/express-frontend:1.0                                                      0.0s
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.

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-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> 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

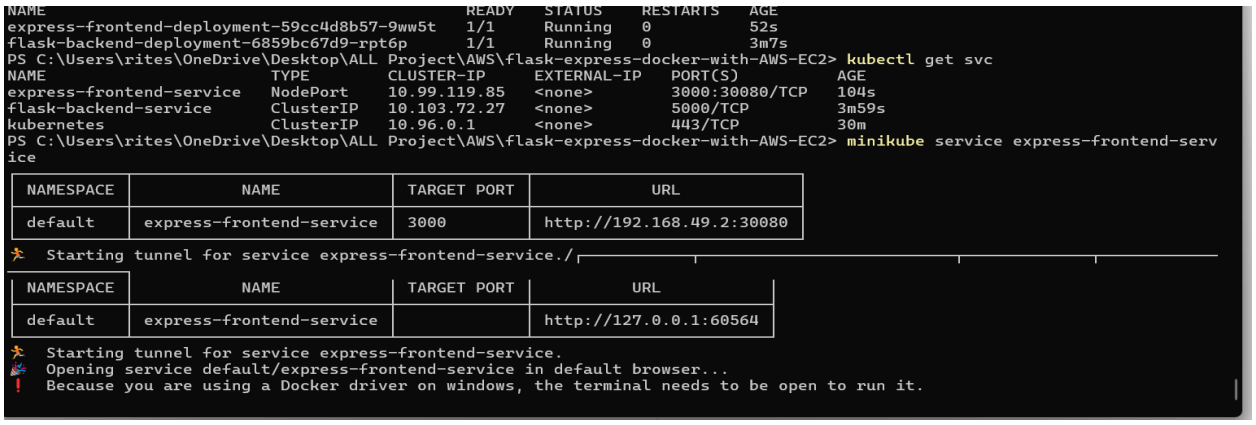
Description:
When running:

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

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

```
http://127.0.0.1:60654
```

This proves that external access via NodePort is functioning.



7.2 Application Working with Backend API

Description:
This browser screenshot confirms full functionality:

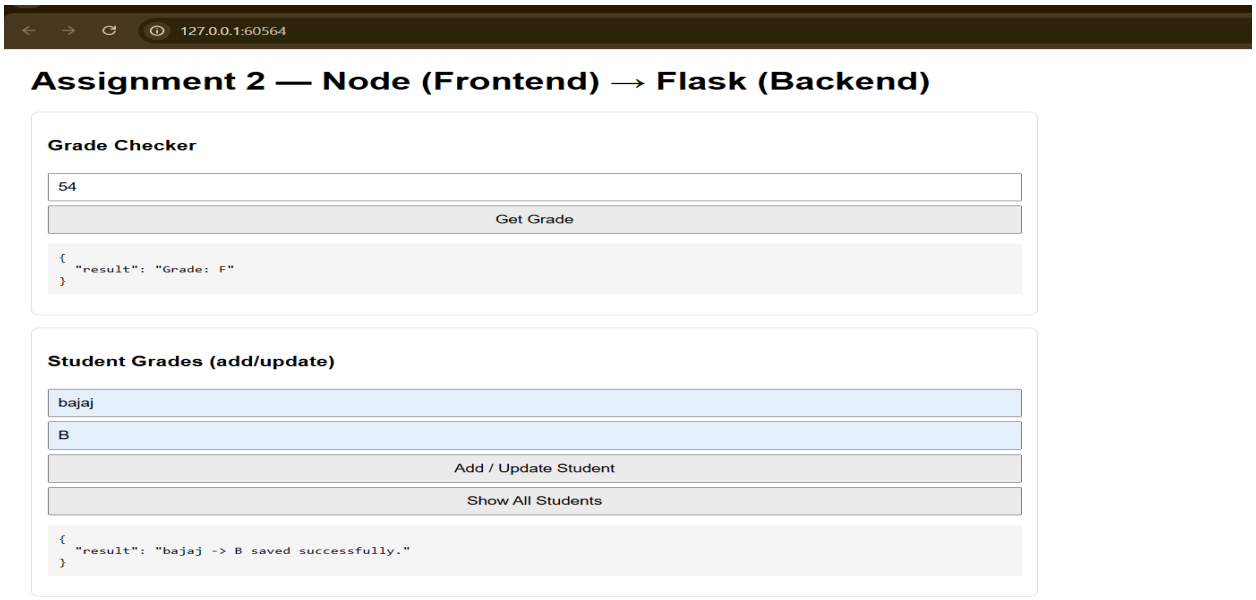
- Grade Checker sends user score → Backend returns JSON
- Student update form sends name + grade → Backend stores and responds

- “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.



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.