# OCR Application/Environment Setup, Deployment Guide Doc

## ☐ Local Setup and Testing

1. Install python and required dependencies

    a. sudo apt update
    b. sudo apt install software-properties-common -y
    c. sudo add-apt-repository ppa:deadsnakes/ppa -y
    d. sudo apt update
    e. sudo apt install python3.12 python3.12-venv python3.12-dev -y
    f. python3.12 --version
    g. sudo apt update
    h. sudo apt install tesseract-ocr -y
    i. tesseract --version
    j. curl -sSL https://install.python-poetry.org | python3 -
    k. echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.bashrc
    l. source ~/.bashrc
    m. poetry --version
    n. poetry env use python3.12

```
kavindu@Kavindus-MacBook-Air DevOpsEngineer-Assignment % poetry env use python3
Creating virtualenv ocr-model-project-eGjzIEIv-py3.12 in /Users/kavindu/Library/Caches/pypoetry/virtualenvs
Using virtualenv: /Users/kavindu/Library/Caches/pypoetry/virtualenvs/ocr-model-project-eGjzIEIv-py3.12
kavindu@Kavindus-MacBook-Air DevOpsEngineer-Assignment %
```

2. Try to install poetry with the command below and get an error as below.

```
  - Installing uvicorn (0.30.6)
  - Installing kserve (0.14.1)
  - Installing pytesseract (0.3.13)
  - Installing python-multipart (0.0.20)

Writing lock file

Installing the current project: ocr-model-project (0.1.0)
Error: The current project could not be installed: Readme path `/Users/kavindu/ifs/DevOpsEngineer-Assignment/README.md` does not exist.
If you do not want to install the current project use --no-root.
If you want to use Poetry only for dependency management but not for packaging, you can disable package mode by setting package-mode = false in your pyproject.toml file.
If you did intend to install the current project, you may need to set `packages` in your pyproject.toml file.

kavindu@Kavindus-MacBook-Air DevOpsEngineer-Assignment % ls
api-gateway.py                DevOps Engineer Assignment.docx model.py                    pyproject.toml
commands.sh                   DevOps Engineer Assignment.pdf  poetry.lock
```

3. Then I created a sample readme file and retry.
    a. echo "# OCR Model Project" > README.md
    b. poetry install
4. Then I faced another issue as below.

```
kavindu@Kavindus-MacBook-Air DevOpsEngineer-Assignment % poetry install
Installing dependencies from lock file

No dependencies to install or update

Installing the current project: ocr-model-project (0.1.0)
Error: The current project could not be installed: No file/folder found for package ocr-model-project
If you do not want to install the current project use --no-root.
If you want to use Poetry only for dependency management but not for packaging, you can disable package mode by setting package-mode = false in your pyproject.toml file.
If you did intend to install the current project, you may need to set `packages` in your pyproject.toml file.

kavindu@Kavindus-MacBook-Air DevOpsEngineer-Assignment %
```

5. Then I run with --no-root param and it works.
    a. poetry install --no-root.

```
kavindu@Kavindus-MacBook-Air DevOpsEngineer-Assignment %
kavindu@Kavindus-MacBook-Air DevOpsEngineer-Assignment %
kavindu@Kavindus-MacBook-Air DevOpsEngineer-Assignment % poetry install --no-root
Installing dependencies from lock file

No dependencies to install or update
kavindu@Kavindus-MacBook-Air DevOpsEngineer-Assignment %
```

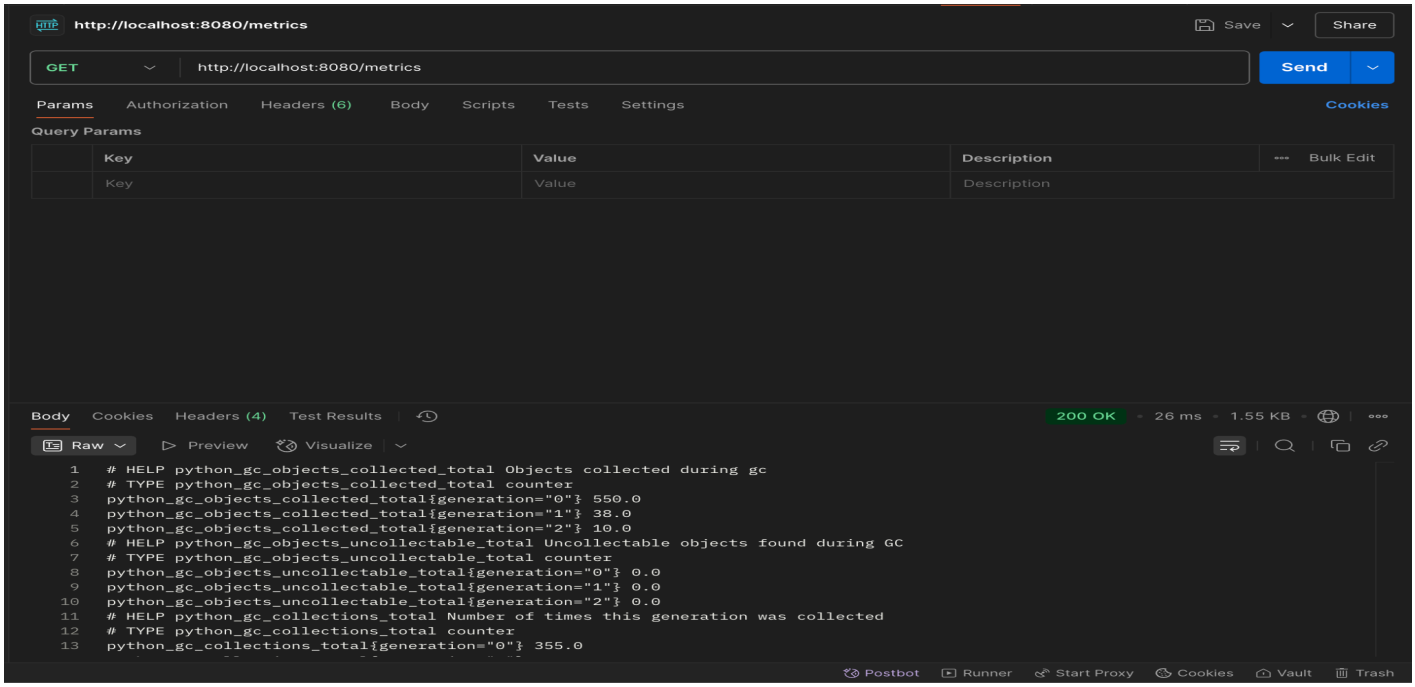
6. Then I started the KServe Model Service.
    a. poetry run python model.py

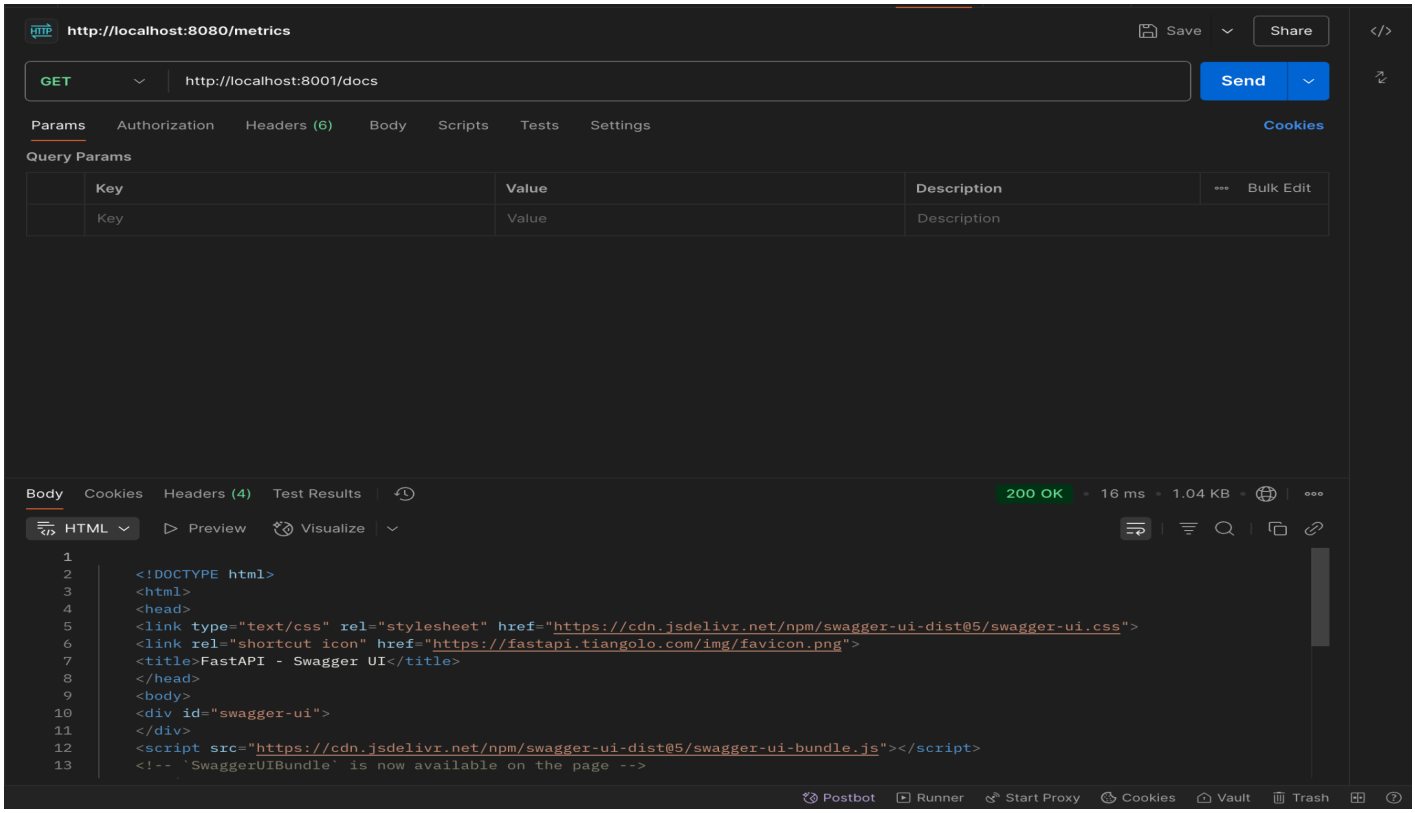
```
kavindu@Kavindus-MacBook-Air DevOpsEngineer-Assignment % poetry run python model.py
2025-06-06 08:31:01.358 82243 kserve INFO [model_server.py:register_model():346] Registering model: ocr-model
2025-06-06 08:31:01.359 82243 kserve INFO [model_server.py:setup_event_loop():264] Setting max asyncio worker threads as 12
2025-06-06 08:31:01.372 82243 kserve INFO [server.py:start():142] Starting uvicorn with 1 workers
2025-06-06 08:31:02.043 uvicorn.error INFO:     Started server process [82243]
2025-06-06 08:31:02.043 uvicorn.error INFO:     Waiting for application startup.
2025-06-06 08:31:02.047 82243 kserve INFO [server.py:start():68] Starting gRPC server with 4 workers
2025-06-06 08:31:02.047 82243 kserve INFO [server.py:start():69] Starting gRPC server on [::]:8081
2025-06-06 08:31:02.047 uvicorn.error INFO:     Application startup complete.
2025-06-06 08:31:02.047 uvicorn.error INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

7. Then I open another terminal and start Gateway Service as well.
    a. poetry run python api-gateway.py

```
DevOpsEngineer-Assignment
kavindu@Kavindus-MacBook-Air ifs % cd DevOpsEngineer-Assignment
kavindu@Kavindus-MacBook-Air DevOpsEngineer-Assignment % ls
api-gateway.py                  DevOps Engineer Assignment.docx model.py                        pyproject.toml
commands.sh                     DevOps Engineer Assignment.pdf  poetry.lock                     README.md
kavindu@Kavindus-MacBook-Air DevOpsEngineer-Assignment % poetry run python api-gateway.py
INFO:     Started server process [82809]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8001 (Press CTRL+C to quit)
```

8. Then i Validate both services are working as expected.

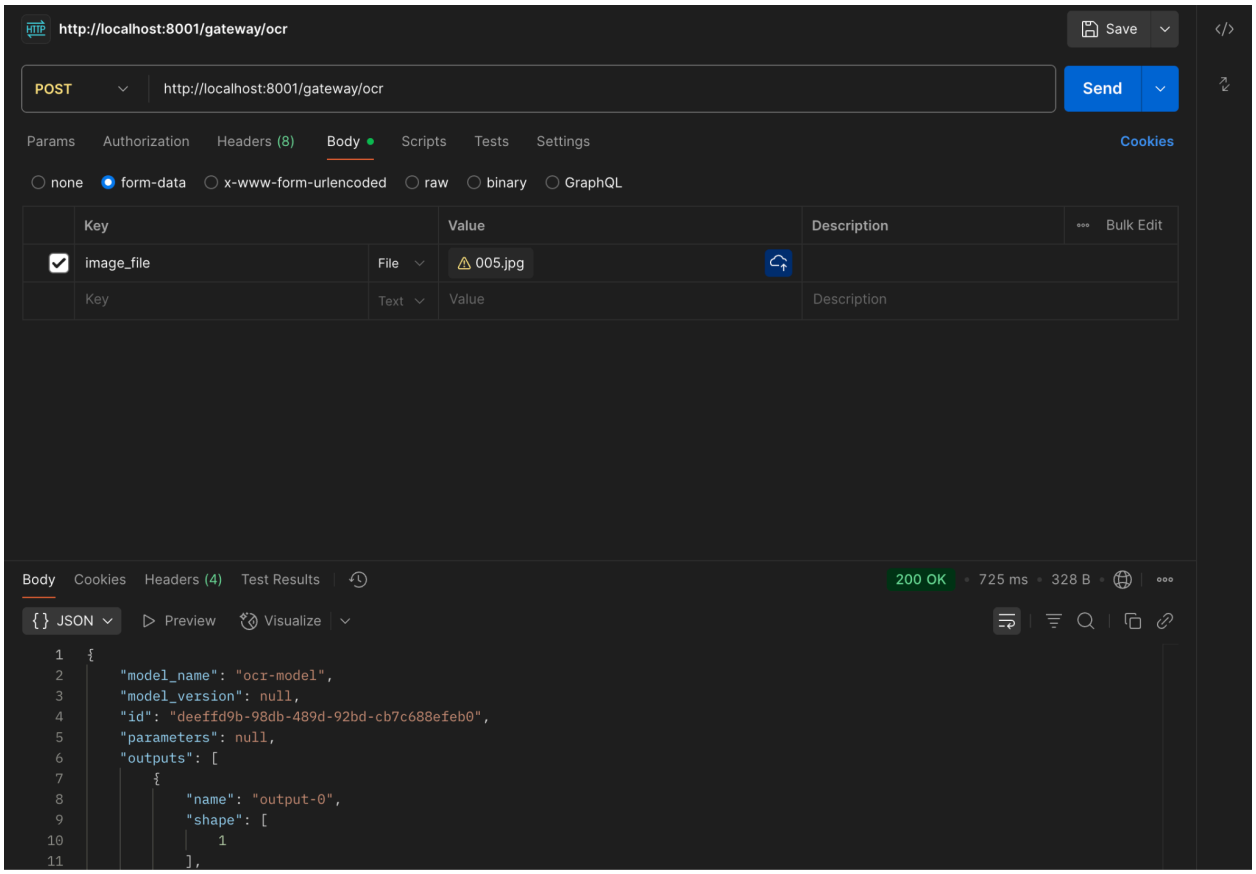


9. But when I was trying to send an image I got this error.
   To fix this we need to modify the model.py file to accept a list, not string.

```
        output = InferOutput(
            name="output-0",
            shape=[1],
            datatype="BYTES",
            data=[extracted_text] # Base64 encode the output string
        )
        infer_response = InferResponse(
            model_name=self.name,
            infer_outputs=[output]
```

10. Then it works.



## ☐ Containerization

1. Create 2 docker files for both Model and Gateway and save them Inside the Docker repo.
2. And create 4 simple shell scripts to build the images , push the images to docker hub, run the containers locally and test them using curl locally. Those all shell scripts are included in scripts repo.
3. We need to change the below line because we deploy both dockers in the same network called ocr-network.

```
app = FastAPI()

KSERVE_URL = "http://ocr-model:8080/v2/models/ocr-model/infer"  # Your KServe model server URL

@app.post("/gateway/ocr")
async def gateway_ocr_request(image_file: UploadFile = File(...)):
    """
```

- Docker Best Practises:
  - **python:3.12-slim**: Official Python image, minimal size (~45MB vs 380MB+ for full).
  - **Multi-stage builds**: Separate build and runtime environments.
  - **Non-root user**: All containers run as `appuser` (not root).
  - **Minimal system packages**: Only install necessary dependencies.
  - **Health checks**: Built-in monitoring for container health.
  - **Minimal runtime dependencies**: Only runtime packages in the final stage.

4. I created a few shell scripts to build docker images, push them to docker hub, run the images.
5. And Gave execute permissions to them.
   a. chmod +x *
6. Execute the start.sh script to start both docker containers.
   a. ./start.sh

## ☐ Infrastructure Setup

1. Remove old Docker installations.
    a. sudo apt remove -y docker docker-engine docker.io containerd runc

2. Add Docker's official GPG key
    a. sudo install -m 0755 -d /etc/apt/keyrings
    b. curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
    c. sudo chmod a+r /etc/apt/keyrings/docker.gpg

3. Add Docker repository
    a. echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

4. Install Docker
    a. sudo apt update
    b. sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

5. Configure Docker for current user
    a. sudo usermod -aG docker $USER
    b. sudo systemctl enable docker
    c. sudo systemctl start docker
    d. newgrp docker

6. Verify Docker installation
    a. docker --version
    b. docker run hello-world

7. Install kubectl
    a. curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

b. curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"
   echo "$(cat kubectl.sha256)  kubectl" | sha256sum --check

c. sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

8. Verify kubectl installation
   a. kubectl version --client

9. Install Helm
   a. curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
      helm version

10. Install Minikube
    a. curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
    b. sudo install minikube-linux-amd64 /usr/local/bin/minikube
    c. rm minikube-linux-amd64
    d. minikube version
    e. minikube start --driver=docker
    f. kubectl get nodes
11. Install minikube addones
    a. minikube addons enable ingress
    b. minikube addons enable metrics-server
    c. minikube addons enable dashboard

12. Create Required Namespaces
    a. kubectl create namespace argocd
    b. kubectl create namespace monitoring
    c. kubectl create namespace ocr-app

13. Install ArgoCD
    a. kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
14. Wait for ArgoCD to be ready
    a. kubectl wait --for=condition=available --timeout=300s deployment/argocd-server -n argocd

15. Verify ArgoCD installation
    a. kubectl get pods -n argocd

16. Get ArgoCD admin password
    a. kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d; echo

17. Port Forward to access ArgoCD UI
    a. nohup kubectl port-forward --address 0.0.0.0 -n argocd service/argocd-server 8080:443 > port-forward.log 2>&1 &

18. Install Prometheus and Grafana
    a. helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
    b. helm repo update
    c. helm install monitoring prometheus-community/kube-prometheus-stack \
       --namespace monitoring \
       --set prometheus.prometheusSpec.storageSpec.volumeClaimTemplate.spec.resources.requests.storage=2Gi \
       --set alertmanager.alertmanagerSpec.storage.volumeClaimTemplate.spec.resources.requests.storage=1Gi \
       --set grafana.persistence.enabled=true \
       --set grafana.persistence.size=1Gi

19. Verify Prometheus installation
    a. kubectl --namespace monitoring get pods -l "release=monitoring"

20. Get Grafana admin password
    a. kubectl --namespace monitoring get secrets monitoring-grafana -o jsonpath="{.data.admin-password}" | base64 -d; echo

21. Port Forward to access grafana UI
    a. nohup bash -c 'export POD_NAME=$(kubectl --namespace monitoring get pod -l "app.kubernetes.io/name=grafana,app.kubernetes.io/instance=monitoring" -o jsonpath="{.items[0].metadata.name}"); kubectl --namespace monitoring port-forward $POD_NAME 3000:3000' > grafana-portforward.log 2>&1 &

22. Deploy OCR Applications with Helm
    a. I created 2 separated helm charts for both ocr model and ocr gateway.

23. Here are the followed steps.
    a. helm create ocr-model
    b. helm create ocr-gateway

24. Then after remove unwanted files and modify templates, values and chart.yml files.
25. Validate Helm charts
    a. helm lint ocr-model
    b. helm lint ocr-gateway
    c. helm template ocr-model ocr-model
    d. helm template ocr-gateway ocr-gateway

26. Deploy applications
    a. helm install ocr-model ./ocr-model --namespace ocr-app
    b. helm install ocr-gateway ./ocr-gateway --namespace ocr-app

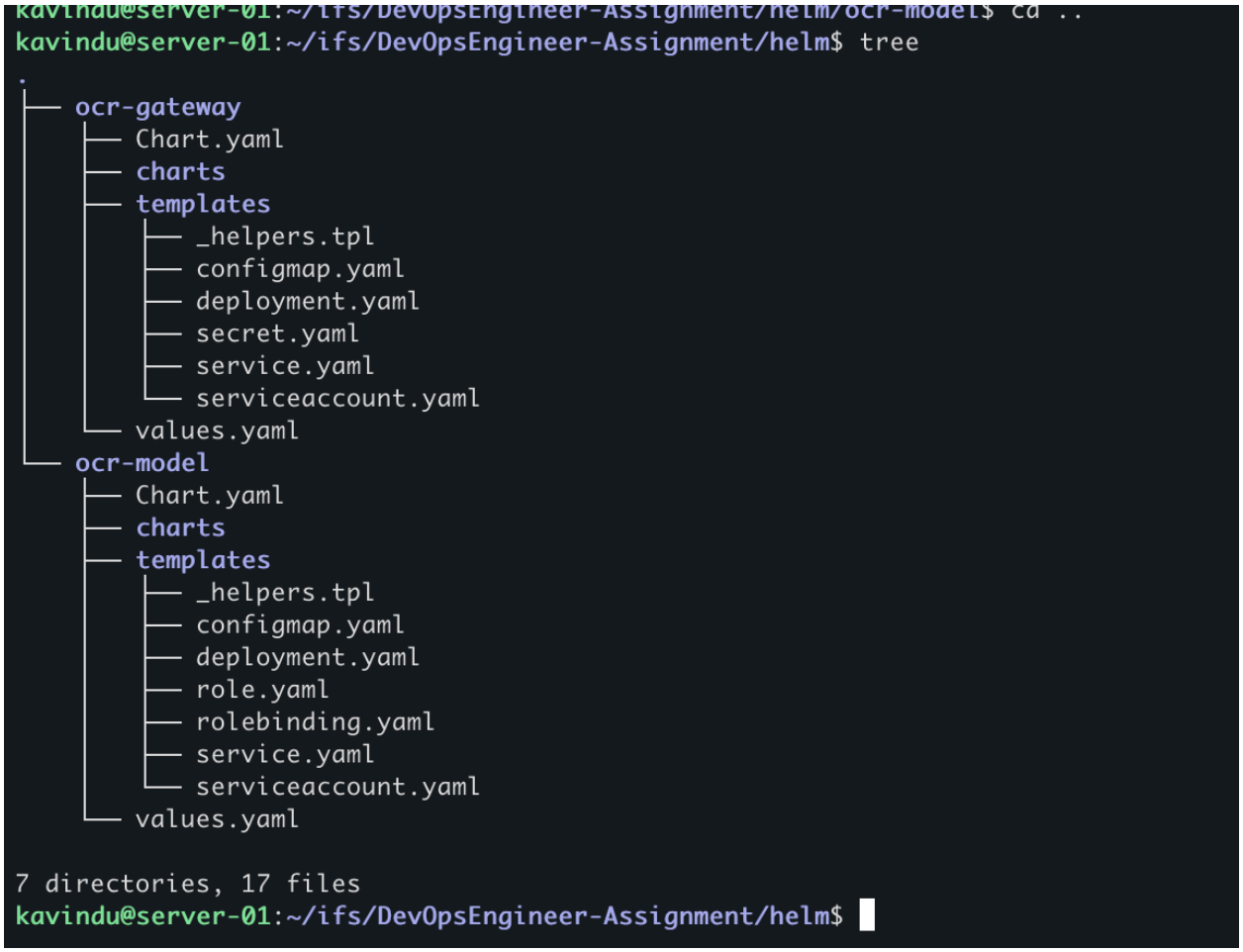
27. Verify deployments
    a. kubectl get pods -n ocr-app
    b. kubectl get services -n ocr-app

28. Forward traffic for external access
    a. kubectl port-forward --address 0.0.0.0 -n ocr-app service/ocr-model 8080:8080 > /dev/null 2>&1 &
    b. kubectl port-forward --address 0.0.0.0 -n ocr-app service/ocr-gateway 8001:8001 > /dev/null 2>&1 &

29. Check all pods across namespaces
    a. kubectl get pods --all-namespaces
    b. kubectl get services --all-namespaces
    c. kubectl get pods -n argocd
    d. kubectl get pods -n monitoring
    e. kubectl get pods -n ocr-app

## ☐ Kubernetes Deployment

Modify Templates folder with chart.yaml and values.yaml




## ☐ GitOps with ArgoCD

1. Now we have all scripts, helm files, and k8s deployment files in the local folder.
2. Now create an argocd folder inside k8s and create below files for applicationset, appproject, ocr-gateway-app and ocr-model-app yaml files.
3. Then apply yaml files.
    a. kubectl apply -f appproject.yaml
    b. kubectl apply -f applicationset.yaml
4. Then inside argocd UI we can see both model and gateway application.
5. If we run

a. kubectl get pod -n orc-app

We can see 2 pods.

## ☐ Monitoring Setup

1. Inside the k8s folder we need to create a folder called monitoring and create servicemonitor.
2. Then apply the created servicemonitor yaml file.
3. Then push the changes to github.
4. Then login to Grafana and create below Dashboards using below queries.
    a. Service Status
        i. up{job="ocr-model"}
    b. Memory Usage
        i. process_resident_memory_bytes{job="ocr-model"} / 1024 / 1024
    c. CPU Usage
        i. rate(process_cpu_seconds_total{job="ocr-model"}[5m]) * 100
    d. Request Rate
        i. rate(python_gc_collections_total{job="ocr-model",generation="0"}[5m])
    e. Python GC Activity
        i. python_gc_objects_collected_total{job="ocr-model"}
    f. Model Inference Latency Histogram
        i. python_gc_collections_total{job="ocr-model"}
    g. Request Success Rate / Error Rate
        i. up{job="ocr-model"} * 100

☐ Simple Architecture Diagram (URL)