Kestra Deployment Guide: From Docker Compose to Amazon EKS

This guide details the **local deployment** of **Kestra** using **Docker Compose** and the **deployment to Amazon EKS** using Kubernetes manifests.

---

1. Deploying Kestra Locally Using Docker Compose

Step 1: Create `docker-compose.yml`

Create a file named `docker-compose.yml` with the following content:

yaml
CopyEdit

```
version: '3.8'

volumes:
  postgres-data:
    driver: local
  kestra-data:
    driver: local

services:
  postgres:
    image: postgres:16
    volumes:
      - postgres-data:/var/lib/postgresql/data
    environment:
      POSTGRES_DB: kestra
      POSTGRES_USER: kestra
      POSTGRES_PASSWORD: k3str4
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -d $${POSTGRES_DB} -U
$${POSTGRES_USER}"]
      interval: 30s
```

```
    timeout: 10s
    retries: 10

kestra:
  build: .
  user: "root"
  volumes:
    - kestra-data:/app/storage
    - /var/run/docker.sock:/var/run/docker.sock
    - /tmp/kestra-wd:/tmp/kestra-wd
  environment:
    KESTRA_CONFIGURATION: |
      datasources:
        postgres:
          url: jdbc:postgresql://postgres:5432/kestra
          driverClassName: org.postgresql.Driver
          username: kestra
          password: k3str4
      kestra:
        server:
          basicAuth:
            enabled: false
            username: "admin@kestra.io"
            password: kestra
        repository:
          type: postgres
        storage:
          type: local
          local:
            basePath: "/app/storage"
        queue:
          type: postgres
        tasks:
          tmpDir:
            path: /tmp/kestra-wd/tmp
        url: http://localhost:8080/
  ports:
    - "8080:8080"
```

```
      - "8081:8081"
    depends_on:
      postgres:
        condition: service_healthy
```

Step 2: Create `Dockerfile`

Create a file named **`Dockerfile`** in the same directory:

dockerfile
CopyEdit

```
FROM kestra/kestra:v0.20.12

USER root

# Install required packages, download Oracle Instant Client, and unzip
it
RUN  apt-get update && \
    apt-get install -y curl unzip && \
    rm -rf /var/lib/apt/lists/* && \
    curl -o /opt/oracle-instantclient.zip
"https://download.oracle.com/otn_software/linux/instantclient/1926000/
instantclient-basic-linux.x64-19.26.0.0.0dbru.zip" && \
    cd /opt && \
    unzip oracle-instantclient.zip

RUN apt-get update && \
    apt-get install -y libaio1 && \
    ldconfig

RUN apt-get update && \
    apt-get install -y libaio-dev

# Set Oracle Instant Client environment variable
ENV LD_LIBRARY_PATH=/opt/instantclient_19_26

# Install required Python dependencies
RUN pip install --upgrade pip
```

```
RUN pip install oracledb==2.5.1
RUN pip install pandas==2.2.3
RUN pip install sqlalchemy==2.0.38
RUN pip install pymysql==1.1.1
RUN pip install pysolr==3.10.0
RUN pip install tqdm==4.67.1
RUN pip install cx_oracle==8.3.0
RUN pip install numpy==1.26.4
RUN pip install boto3==1.37.5

CMD ["server", "standalone"]
```

Step 3: Build and Run Locally

Run the following command to build and start Kestra locally:

sh
CopyEdit
```
docker-compose up -d --build
```

Check if the containers are running:

sh
CopyEdit
```
docker ps
```

Access **Kestra UI** at:

arduino
CopyEdit
```
http://localhost:8080
```

---

2. Deploying Kestra on AWS EKS

Step 1: Push Docker Image to AWS ECR

Authenticate Docker with AWS ECR

sh
CopyEdit
```
aws ecr get-login-password --region us-east-2 | docker login
--username AWS --password-stdin
<AWS_ACCOUNT_ID>.dkr.ecr.us-east-2.amazonaws.com
```

1.

Tag the Docker Image

sh
CopyEdit
```
docker tag kestra-db_kestra
<AWS_ACCOUNT_ID>.dkr.ecr.us-east-2.amazonaws.com/kestra:latest
```

2.

Push the Image to ECR

sh
CopyEdit
```
docker push
<AWS_ACCOUNT_ID>.dkr.ecr.us-east-2.amazonaws.com/kestra:latest
```

3.

Step 2: Create `postgres-deployment.yaml`

Create a file called **`postgres-deployment.yaml`**:

yaml
CopyEdit
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres
spec:
  replicas: 1
  selector:
```

```
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres
          image: postgres:16
          env:
            - name: POSTGRES_DB
              value: "kestra"
            - name: POSTGRES_USER
              value: "kestra"
            - name: POSTGRES_PASSWORD
              value: "k3str4"
          ports:
            - containerPort: 5432
---
apiVersion: v1
kind: Service
metadata:
  name: postgres
spec:
  selector:
    app: postgres
  ports:
    - protocol: TCP
      port: 5432
      targetPort: 5432
  clusterIP: None
```

Step 3: Create `kestra-deployment.yaml`

Create a file called **`kestra-deployment.yaml`**:

yaml
CopyEdit

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kestra
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kestra
  template:
    metadata:
      labels:
        app: kestra
    spec:
      containers:
        - name: kestra
          image:
<AWS_ACCOUNT_ID>.dkr.ecr.us-east-2.amazonaws.com/kestra:latest
          ports:
            - containerPort: 8080
            - containerPort: 8081
          env:
            - name: KESTRA_CONFIGURATION
              value: |
                datasources:
                  postgres:
                    url: jdbc:postgresql://postgres:5432/kestra
                    driverClassName: org.postgresql.Driver
                    username: kestra
                    password: k3str4
                kestra:
                  server:
                    basicAuth:
                      enabled: false
                      username: "admin@kestra.io"
                      password: kestra
                  repository:
```

```
      type: postgres
    storage:
      type: local
      local:
        basePath: "/app/storage"
    queue:
      type: postgres
    tasks:
      tmpDir:
        path: /tmp/kestra-wd/tmp
    url: http://localhost:8080/
---
apiVersion: v1
kind: Service
metadata:
  name: kestra
spec:
  type: LoadBalancer
  selector:
    app: kestra
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
    - protocol: TCP
      port: 8081
      targetPort: 8081
```

Step 4: Deploy on EKS

Apply PostgreSQL Deployment

sh
CopyEdit

```
kubectl apply -f postgres-deployment.yaml
```

1.

Apply Kestra Deployment

sh
CopyEdit
```
kubectl apply -f kestra-deployment.yaml
```

2.

Step 5: Get LoadBalancer External IP

Check if the **Kestra service** has an external IP:

sh
CopyEdit
```
kubectl get svc
```

Expected output:

sh
CopyEdit
```
NAME         TYPE           CLUSTER-IP       EXTERNAL-IP
PORT(S)                         AGE
kestra       LoadBalancer   10.100.67.146
a8774be45b2b44eef8c8b9dd381b432c-468344845.us-east-2.elb.amazonaws.com
8080:31363/TCP,8081:30434/TCP   7s
```

Now, access **Kestra UI** at:

cpp
CopyEdit
```
http://<EXTERNAL-IP>:8080
```

Final Summary

✅ **Deployed Kestra with Oracle Instant Client**
✅ **Pushed Docker image to AWS ECR**
✅ **Deployed Kestra & PostgreSQL on Amazon EKS**
✅ **Accessed Kestra UI via LoadBalancer** 🚀