Deploying MW to a Kubernetes Cluster Procedure.

Of course. Here’s a step-by-step guide to deploying that Java application to Kubernetes.

This process involves two main stages:

1. **Packaging the application** into a container image.
2. **Creating Kubernetes configuration files** to deploy the image.

**Step 1: Build and Push the Docker Image 📦**

First, you need to build the Docker image from the project's Dockerfile and push it to a container registry that your Kubernetes cluster can access (like Docker Hub, Google Artifact Registry, or Amazon ECR).

1. **Navigate to the project directory** (the one containing the Dockerfile).
2. **Build the Docker image**. Run the following command, replacing your-registry/your-image-name:tag with your registry path, desired image name, and a version tag (e.g., v1.0).

Bash

docker build -t your-registry/your-image-name:tag .

1. **Push the image** to your container registry.

Bash

docker push your-registry/your-image-name:tag

**Step 2: Create the Kubernetes Deployment File 📝**

Next, create a file named deployment.yaml. This file tells Kubernetes how to run your application, including which container image to use and how many replicas to create.

YAML

apiVersion: apps/v1

kind: Deployment

metadata:

name: efx-mw-gava-connect-deployment

spec:

replicas: 2

selector:

matchLabels:

app: efx-mw-gava-connect

template:

metadata:

labels:

app: efx-mw-gava-connect

spec:

containers:

- name: gava-connect-app

# Replace this with the image you just pushed

image: your-registry/your-image-name:tag

ports:

- containerPort: 8080 # Default port for many Java web apps

**Note**: Check your application's application.properties or similar configuration file to confirm the port it runs on and adjust containerPort if necessary.

**Step 3: Create the Kubernetes Service File 🌐**

Now, create a service.yaml file to expose your application to network traffic. This example uses a LoadBalancer, which creates an external IP address to access your application from the internet.

YAML

apiVersion: v1

kind: Service

metadata:

name: efx-mw-gava-connect-service

spec:

type: LoadBalancer

selector:

# This must match the 'app' label in your deployment.yaml

app: efx-mw-gava-connect

ports:

- protocol: TCP

port: 80

targetPort: 8080

**Step 4: Deploy to Kubernetes 🚀**

Finally, use kubectl to apply these configurations to your cluster. Make sure your kubectl is configured to connect to your Kubernetes cluster.

1. **Apply the deployment:**

Bash

kubectl apply -f deployment.yaml

1. **Apply the service:**

Bash

kubectl apply -f service.yaml

Your Java application is now deployed. You can check the status and find the external IP address with these commands:

Bash

# Check the status of your running pods

kubectl get pods

# Get the external IP address for your service

kubectl get service efx-mw-gava-connect-service