Deploying [**efx-mw-gava-connect**](https://github.com/Empire-FXInc/efx-mw-gava-connect/tree/code-cleanup) Journey.

You started with a Java application on your computer and successfully packaged it, pushed it to a container registry, and deployed it to a running Kubernetes cluster. You fixed four distinct problems along the way: a local Docker setup issue, a permissions error when pushing the image, a Kubernetes connectivity error, and finally, a configuration error in your deployment file. You have now reached the final step, which is an application-level issue, not an infrastructure one.

**Phase 1: Building the Docker Image**

Your goal was to package your Java application into a Docker container image.

* **Your Command:** docker build -t empirefx-gava-connect:0.0.1 .
* **The Problem:** You received an error: ERROR: error during connect... The system cannot find the file specified.
* **The Cause:** This meant the Docker command-line tool could not find the Docker Desktop service because the application wasn't running on your Windows machine.
* **The Fix:** You started Docker Desktop, which allowed the next commands to work.

**Phase 2: Pushing the Image to Docker Hub**

After building the image, you needed to upload it to a central registry (Docker Hub) so Kubernetes could download it.

* **Your First Command:** docker push empirefx-gava-connect:0.0.1
* **The Problem:** You received a permissions error: push access denied, repository does not exist or may require authorization.
* **The Cause:** Images must be pushed to a personal namespace. By not including your username, Docker tried to push it to the official "library" namespace, which is not allowed.
* **The Fix:** You correctly logged in, re-tagged the image with your Docker Hub username (jeffare9x), and pushed the correctly tagged image.
  + **Tag Command:** docker tag empirefx-gava-connect:0.0.1 jeffare9x/empirefx-gava-connect:0.0.1
  + **Push Command:** docker push jeffare9x/empirefx-gava-connect:0.0.1

**Phase 3: Deploying to Kubernetes**

With the image on Docker Hub, you started deploying it to your local Kubernetes cluster.

* **Your First Command:** kubectl apply -f deployment.yaml
* **The Problem:** You received a connection error: dial tcp 172.30.199.178:8443: connectex: A connection attempt failed...
* **The Cause:** This meant kubectl could not communicate with the Kubernetes API server, likely because the Kubernetes cluster inside Docker Desktop was not running or ready.
* **The Fix:** You started or enabled the Kubernetes cluster within Docker Desktop, which allowed kubectl to connect. You verified this with kubectl get nodes.

**Phase 4: Fixing the Running Pods**

You successfully created the deployment and service, but the application wouldn't start.

* **Your Command:** kubectl get pods
* **The Problem:** The pods had a status of ImagePullBackOff and ErrImagePull.
* **The Cause:** Your kubectl describe pod output revealed the specific error: Kubernetes was trying to pull a placeholder image named "your-registry/your-image-name:tag". This was because you had not replaced the placeholder text in the deployment.yaml file with your actual image name.
* **The Fix:** You edited the deployment.yaml file, replaced the placeholder with your actual image path (jeffare9x/empirefx-gava-connect:0.0.1), and re-applied the file with kubectl apply -f deployment.yaml. This allowed Kubernetes to pull the correct image, and the pods started successfully.

**Final Step: Accessing the Application**

You have now successfully deployed your application. The final step was to access it.

* **Your Command:** curl http://localhost
* **The Problem:** You received a 404 Not Found error in the form of a JSON message and then a "Whitelabel Error Page".
* **The Cause:** **This is a success!** The error is coming *from your Java application*, not Kubernetes. It means the networking is working perfectly, but your application has no code to handle requests for the root path (/).
* **The Final Task:** You need to check your Java source code (@RestController files) to find the correct application path (e.g., /status, /api/connect) and access that specific URL