Setup Guide: Centralized Event Review POC on Minikube

This guide provides step-by-step instructions to set up the Proof of Concept (POC) environment for the Centralized Event Review Platform using Minikube on your local machine. This setup includes the Event Distribution Service, Event Ingestor Service, PostgreSQL database, and an Ingress controller for external access.

1. Prerequisites

Before you begin, ensure you have the following tools installed and configured on your system:

1. **Git:** For cloning the repositories.
2. **Node.js & npm:** For building the services and running scripts (LTS version recommended, e.g., v18 or later).
3. **Docker:** For building container images (Docker Desktop or Docker Engine). Ensure the Docker daemon is running.
4. **Minikube:** For running a local Kubernetes cluster.
5. **kubectl:** The Kubernetes command-line tool. This is often installed with Docker Desktop or can be installed separately. Ensure it's configured to communicate with your Minikube cluster.
6. **(Optional) `curl` or Postman:** For testing API endpoints.

2. GitHub Repository Structure

This guide assumes you have structured your code and configurations as follows:

- **GitHub Organization:** PAI-Amit-Tank
- **Service Repositories:**
    - *event_distribution_service_v1*: Contains source code, Dockerfile, entrypoint.sh, migrations for this service.
    - *event_ingestor_service_v1*: Contains source code, Dockerfile for this service.

- **Kubernetes Manifests Repository:**
  k8s/
  ├── platform/

```
│   ├── namespace.yaml
│   └── postgresql/
│       ├── pvc.yaml
│       ├── statefulset.yaml
│       └── service.yaml
└── services/
├── event-distribution-service/
│   ├── configmap.yaml
│   ├── secret.yaml
│   ├── deployment.yaml
│   ├── service.yaml
│   └── cronjob.yaml
|── event-ingestor-service/
|    ├── configmap.yaml
|    ├── deployment.yaml
|    └── service.yaml
```

3. Clone Repositories

Clone the source code for the services and the Kubernetes manifests if you haven't already.

1. **Clone Application Service Repositories:**
   Open your terminal and navigate to your desired workspace.

   ```
   # Clone the event distribution service
   git clone https://github.com/PAI-Amit-Tank/event_distribution_service_v1.git
   cd event_distribution_service_v1
   npm install # Install dependencies

   # Clone the event ingestor service
   git clone https://github.com/PAI-Amit-Tank/event_ingestor_service_v1.git
   cd event_ingestor_service_v1.git
   npm install # Install dependencies
   ```

2. **Clone the Kubernetes Manifests Repository:**

   ```
   git clone https://github.com/PAI-Amit-Tank/k8s.git
   ```

4. Start Minikube

Start your local Kubernetes cluster with sufficient resources.

```
minikube start --memory 4g --cpus 2
```

- **Verification:** Run minikube status. Ensure host, kubelet, and apiserver show Running.

5. Build and Load Docker Images

Build the container images for both microservices and load them into Minikube's internal Docker registry so Kubernetes can find them.

1. **Event Distribution Service:**
   ```
   cd ../event_distribution_service_v1
   docker build -t event-distribution-service:latest .
   minikube image load event-distribution-service:latest
   ```
   - **Verification:** minikube image ls | grep event-distribution-service (should list the image).

2. **Event Ingestor Service:**
   ```
   cd ../event_ingestor_service_v1 # Navigate to the service's source code directory
   docker build -t event-ingestor-service:latest .
   minikube image load event-ingestor-service:latest
   ```
   - **Verification:** minikube image ls | grep event-ingestor-service (should list the image).

3. **Return to K8s Directory:**
   ```
   cd ../k8s
   ```

7. Deploy Kubernetes Resources

Apply the manifest files to create the necessary resources in Minikube. Ensure you are in the root of the k8s-manifests directory.

1. **Apply Namespace:**
   ```
   kubectl apply -f platform/namespace.yaml
   ```
   - **Verification:** kubectl get namespace event-review-platform (Status should be Active).

2. **Apply PostgreSQL PVC (Persistent Volume Claim):**
   ```
   kubectl apply -f platform/postgresql/pvc.yaml
   ```

- **Verification:** kubectl get pvc -n event-review-platform (Status should eventually be Bound).

3. **Apply Secrets & ConfigMaps:**
   kubectl apply -f services/event-distribution-service/secret.yaml
   kubectl apply -f services/event-distribution-service/configmap.yaml
   kubectl apply -f services/event-ingestor-service/ingestor.yaml
   Verification:
      - kubectl get secret -n event-review-platform (Should list event-distribution-secrets and event-ingestor-secrets).
      - kubectl get configmap -n event-review-platform (Should list event-distribution-config and event-ingestor-config).

4. **Apply PostgreSQL (StatefulSet & Service):**
   kubectl apply -f platform/postgresql/statefulset.yaml
   kubectl apply -f platform/postgresql/service.yaml
   - **Verification:** Wait for the PostgreSQL pod to become ready (this can take a minute or two):
     echo "Waiting for PostgreSQL pod to initialize..."
     kubectl wait --for=condition=ready pod -l app=postgresql-db -n event-review-platform --timeout=180s
     kubectl get pods -l app=postgresql-db -n event-review-platform # Should show 1/1 Running

5. **Apply Event Distribution Service (Deployment, Service, CronJob):**
   - The entrypoint.sh script in this service's Docker image will automatically run database migrations (npm run migrate:up) when the container starts.
     kubectl apply -f services/event-distribution-service/deployment.yaml
     kubectl apply -f services/event-distribution-service/service.yaml
     kubectl apply -f services/event-distribution-service/cronjob.yaml

   - **Verification:** Wait for the deployment to become available:
     echo "Waiting for Event Distribution Service deployment..."
     kubectl wait --for=condition=available deployment/event-distribution-service -n event-review-platform --timeout=120s
     kubectl get deployment,service,cronjob -l app=event-distribution-service -n event-review-platform
     kubectl get pods -l app=event-distribution-service -n event-review-platform # Should show 1/1 Running (or your replica count)

   - Check migration logs (look for "[Entrypoint] Migrations complete."):
     # Get a pod name first
     EVENT_DIST_POD=$(kubectl get pods -l app=event-distribution-service -n

event-review-platform -o jsonpath='{.items[0].metadata.name}')
echo "Checking logs for $EVENT_DIST_POD"
kubectl logs $EVENT_DIST_POD -n event-review-platform --tail=50

6. **Apply Event Ingestor Service (Deployment & Service):**
kubectl apply -f services/event-ingestor-service/deployment-ingestor.yaml
kubectl apply -f services/event-ingestor-service/service-ingestor.yaml

   ○ **Verification:** Wait for the deployment to become available:
   echo "Waiting for Event Ingestor Service deployment..." kubectl wait --for=condition=available deployment/event-ingestor-service -n event-review-platform --timeout=120s kubectl get deployment,service -l app=event-ingestor-service -n event-review-platform kubectl get pods -l app=event-ingestor-service -n event-review-platform # Should show 1/1 Running (or your replica count)

8. Load Mock Data (Seed the Database)

This step populates your database (running in Minikube) with sample teams, users and events using the seed script from the `event-distribution-service` project.

1. **Port-Forward PostgreSQL:** Open a **new, dedicated terminal window** and keep this command running:
kubectl port-forward service/postgresql-db -n event-review-platform 5432:5432
2. **Configure Local `.env` for Seed Script:**
   - Navigate to your local event-distribution-service project directory.
   - Ensure the .env file in this directory has the correct database connection details to reach the port-forwarded PostgreSQL:
     - DB_HOST=localhost
     - DB_PORT=5432 (or the local port you used in the port-forward command, e.g., 5433)
     - DB_USERNAME, DB_PASSWORD, DB_DATABASE should match the values you base64 encoded and put in your Kubernetes secret.yaml.
3. **Run Seed Script:**

   npm run db:seed

4. **Keep the printed data:**

Keep the printed data from the script in hand, that will be used while testing the service.