Pipeline & Monitoring Report

Date: December 9, 2025

Purpose: concise (≤10 pages) report describing the CI/CD pipeline architecture, each phase and tools used, resource calculations based on current manifests, and instructions / placeholders to capture screenshots of pipeline runs and monitoring dashboards.

---

1) Architecture Diagram

Mermaid diagram (renderable in many Markdown viewers / VS Code with Mermaid preview):

```
flowchart LR
  subgraph SCM[Source Control]
    A[GitHub repo]
  end

  subgraph CI[CI/CD - GitHub Actions]
    A --> B[Test]
    B --> C[Quality]
    C --> D[Build & Push Docker]
    D --> E[Deploy]
    E --> F[Release]
  end

  subgraph Registry[Registry]
    D --> G[Docker Hub]
  end

  subgraph K8s[Kubernetes Cluster]
    E --> H[Helm / kubectl]
    H --> I[Deployment (attendance-app)]
    H --> J[MySQL StatefulSet]
    I --> K[HPA]
    I --> L[Prometheus / Grafana]
    L --> M[Alertmanager]
    L --> N[Loki]
  end

  subgraph Observability[Observability]
    L --> O[Grafana Dashboards]
    M --> P[Slack / Webhook]
  end

  style SCM fill:#f9f,stroke:#333,stroke-width:1px
  style CI fill:#eef,stroke:#333
  style K8s fill:#efe,stroke:#333
  style Observability fill:#ffe,stroke:#333
```

---

2) Pipeline Phases & Tools

- Source Control: GitHub (repo: `attendance-event`). Triggers on push / PR.
- Test: Jest + Supertest. Runs unit & integration tests. Produces JUnit XML for CI.
- Quality: ESLint, OWASP Dependency Check. Produces reports and fails on critical issues.
- Build & Push: Docker builds image, tags `latest` + commit SHA, pushes to Docker Hub `davidniyonkuru15/attendance-event`.
- Deploy: `helm` (preferred) or `kubectl`. Helm chart at `helm/attendance-event/`.
- Release: GitHub Releases triggered on tag push (v..*).

Observability & Ops tools: - Prometheus / kube-prometheus-stack (Prometheus, Alertmanager, Grafana) - Loki (logs) + Promtail - Alertmanager configured to notify Slack (example config in `monitoring/alertmanager-config.yaml`). - HPA (k8s/hpa.yaml) for automatic scaling.

---

3) Screenshots (placeholders)

Include these screenshots in the final PDF or repo `docs/screenshots/` folder. Capture high-resolution (1200×800) images.

1. GitHub Actions run summary (Tests → Quality → Build → Deploy).
   - How to capture: open your run (https://github.com//attendance-event/actions), take screenshot of the workflow run summary.
2. Docker Hub image page showing pushed tags.
   - How to capture: https://hub.docker.com/r/davidniyonkuru15/attendance-event
3. Helm/Deployment successful in `kubectl get pods -n attendance`.
   - How to capture: `kubectl get pods -n attendance` and screenshot terminal or copy output into report.
4. Grafana dashboard showing CPU/requests and HPA scaling events.
   - How to capture: port-forward Grafana and open dashboard; screenshot panel.
5. Prometheus alert firing (HighErrorRate) and Alertmanager notification to Slack (or webhook).

Place screenshots under a `docs/screenshots/` folder and reference them here.

Screenshot placeholders

Please add the following images to `docs/screenshots/` (these files are referenced below):

- `01_github_actions_run.png` — GitHub Actions workflow run summary
- `02_dockerhub_tags.png` — Docker Hub image/tags page

- `03_kubectl_pods.png` — `kubectl get pods -n attendance` output or screenshot
- `04_grafana_hpa.png` — Grafana dashboard showing CPU and HPA events
- `05_alertmanager_slack.png` — Alertmanager alert and Slack notification screenshot

Insert images below the corresponding sections or place them all in a `Screenshots` section at the end.

Example image insertion (Markdown):

```
![GitHub Actions run](docs/screenshots/01_github_actions_run.png)
```

---

4) Resource Calculation Table

Source: `k8s/deployment.yaml` (app) and `k8s/mysql-statefulset.yaml` (DB).

Summary per pod:

Component	Replicas	CPU request	CPU limit	Memory request	Memory limit	Tota CPI request
attendance-app	3	100m	500m	128Mi	512Mi	300n
mysql (stateful)	1	250m	1000m	256Mi	1Gi	250n
Prometheus (kube-prom-stack) *	1 (varies by chart)	500m (est)	1500m (est)	1Gi (est)	2Gi (est)	500n
Grafana *	1	100m (est)	300m (est)	256Mi (est)	512Mi (est)	100n

*Prometheus/Grafana resource values depend on chart and cluster size — these are conservative estimates for small clusters.

Cluster sizing — minimal cluster capacity to run components comfortably (sum of requests + buffer):

- Sum CPU requests = 300m (app) + 250m (mysql) + 500m (prom) + 100m (grafana) = 1.15 CPU
- Add 30% buffer => ~1.5 CPU
- Sum memory requests = 384Mi + 256Mi + 1Gi + 256Mi = 1.9 Gi
- Add 30% buffer => ~2.5 Gi

Recommendation: a 2-node cluster with 2 vCPU and 4 Gi RAM each (or a single node with 4 CPU / 8 Gi RAM) for testing + monitoring.

---

5) Pipeline Execution & Monitoring Dashboard (how to reproduce)

View GitHub Actions run

Open: https://github.com/davidniyonkuru15/attendance-event/actions and select the latest run. Copy the run URL for your screenshot.

Port-forward Grafana & Prometheus locally

```
        # Grafana
        kubectl -n monitoring port-forward svc/kube-prom-stack-grafana
3000:80
        # Prometheus
        kubectl -n monitoring port-forward svc/kube-prom-stack-prometheus
9090:9090
        # Loki (optional)
        kubectl -n monitoring port-forward svc/loki 3100:3100
```

Open Grafana at `http://localhost:3000` (admin/admin) — import a dashboard (CPU, memory, HPA events). Screenshot the dashboard panels showing scaling.

Watch HPA and pods while generating load

```
        # Apply HPA
        kubectl apply -f k8s/hpa.yaml

        # Start load generator (runs hey for 60s)
        chmod +x scripts/trigger_load.sh
        ./scripts/trigger_load.sh attendance

        # In another terminal:
        kubectl get hpa -n attendance --watch
        kubectl get pods -n attendance --watch
        kubectl top pods -n attendance
```

Expected outcome: HPA will increase `attendance-app` replicas based on CPU usage; Grafana will show increased CPU and new pods. Alertmanager may fire if error thresholds are exceeded.

---

6) Alerting & Feedback Loop

1. Alerting rule `HighErrorRate` is defined in `monitoring/alert-rules.yaml`. When 5xx rate > 5% over 5m, Alertmanager notifies Slack.
2. To implement an automated feedback loop (pipeline trigger):
   - Create a secure HTTP webhook receiver that accepts Alertmanager webhooks, validates them, and calls GitHub REST API `POST /repos/{owner}/{repo}/actions/workflows/{workflow_id}/dispatches` with a token to trigger a remediation workflow.
   - Alternatively, configure Alertmanager to call a small Cloud Function

that triggers the workflow.

Security: Use secrets (Kubernetes Secret, GitHub Secrets) for tokens and do not store them in source code.

---

7) Appendix: Commands & Quick Checks

```bash
# Validate manifests (no cluster required)
kubectl apply -f k8s/ --dry-run=client --validate=false

# Lint helm chart
helm lint helm/attendance-event/

# Install monitoring (requires cluster)
chmod +x scripts/install-monitoring.sh
./scripts/install-monitoring.sh

# Apply alert rules
kubectl apply -f monitoring/alert-rules.yaml -n monitoring
kubectl apply -f monitoring/alertmanager-config.yaml -n monitoring

# Deploy app
kubectl apply -f k8s/

# Trigger load and watch HPA
./scripts/trigger_load.sh attendance
kubectl get hpa -n attendance --watch
kubectl get pods -n attendance --watch
```

If you want, I can now:

1. Attempt to run the monitoring installer in your local cluster (minikube/kind) and then run the load generator to demonstrate HPA scaling and capture screenshots; or
2. Produce a PDF export of this Markdown and add screenshot placeholders so you can drop images in.

Which would you prefer? (I can proceed with 1 if your local cluster is running.)