(Senior Academy - IT training center)

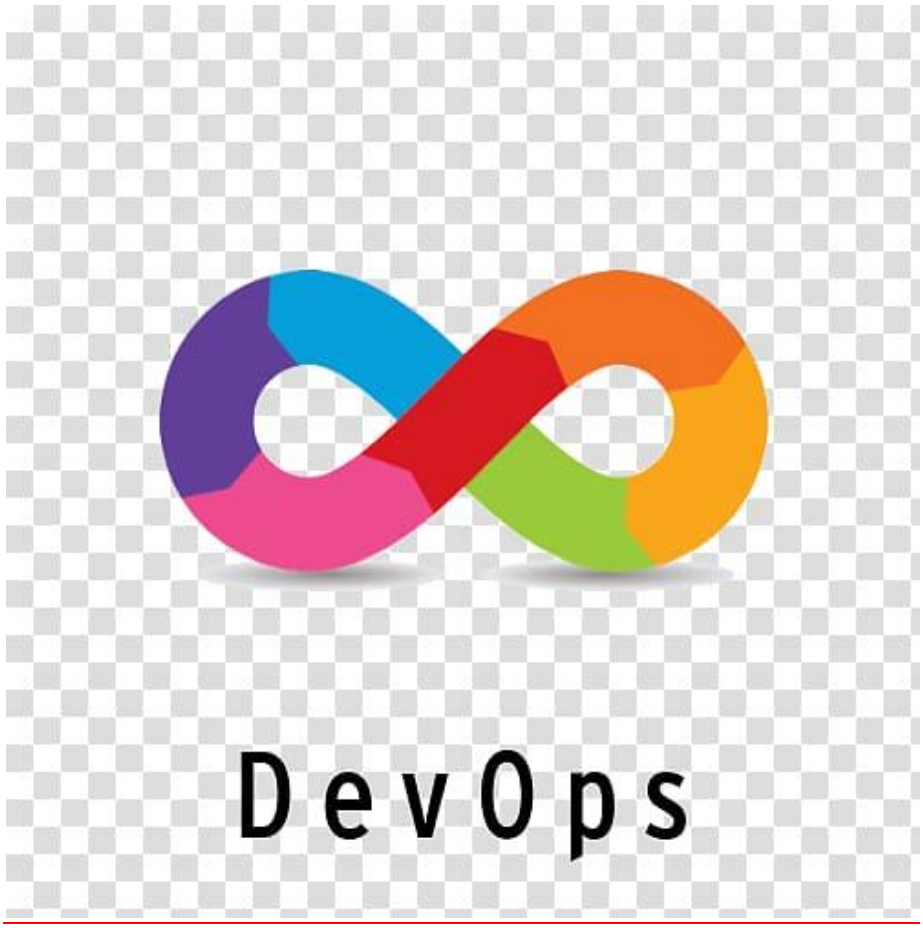
The Place You Can Be A Senior

www.seniorsteps.net
https://www.facebook.com/seniorsteps.it
contact us: 0224153419 - 01090873748

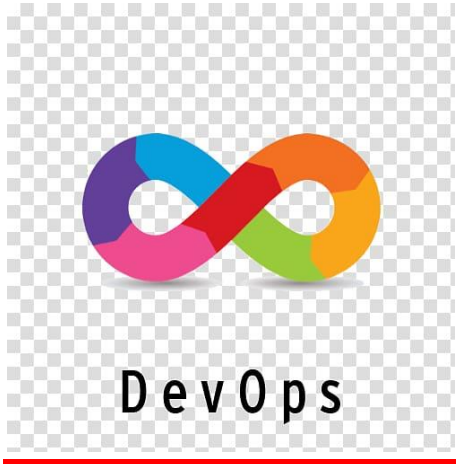
فرع مدينة نصر 1 : عمارة 4 – شارع محمد توفيق دياب – عباس العقاد - مدينة نصر – الدوران 1

Senior Academy - IT training center
www.seniorsteps.net
contact us: 0224153419 - 01090873748
عمارة 4 – شارع محمد توفيق دياب – عباس العقاد - مدينة نصر – الدورال 1

DevOps Engineer Diploma



DevOps Engineer Diploma



OpenShift Labs

Lab 16

Deploying a Three-Tier Application on OpenShift

Lab Objectives

- Understanding OpenShift Project and Deployment Architecture
- Deploying Multi-Tier Applications (Frontend, Backend, Database)
- Configuring Services for Internal and External Communication
- Using ConfigMaps and Secrets for Application Configuration and Credentials
- Implementing Persistent Storage with PVCs for Databases
- Exposing Applications Securely via OpenShift Routes

Objective

Design, deploy, and manage a three-tier web application on OpenShift—**Frontend (Nginx)**, **Backend (Go)**, and **Database (MySQL)**—using best practices for configuration, routing, service discovery, and persistence.

Architecture Components

1) Frontend (Web Tier)

- Technology: Nginx serving static assets (HTML/CSS/JS).
- Exposure: Public via an OpenShift **Route**.
- Internal access to backend through a **ClusterIP Service**.
- Configuration via **ConfigMaps** (e.g., backend API URL, static content).

2) Backend (Application Tier)

- Technology: Golang (Go) implementing business logic and REST/API.
- Communication: Internal **ClusterIP Service** for frontend → backend traffic.
- Configuration:
  - Non-secret parameters via **ConfigMaps** (e.g., DB host, port).
  - Secrets (credentials, tokens) via **Secrets**.
- Reliability: Health checks (readiness/liveness), resource requests/limits.

3) Database (Data Tier)

- Technology: MySQL with persistent storage.
- Persistence: **PersistentVolumeClaim (PVC)** attached to the DB Deployment/StatefulSet.
- Security: Credentials stored in **Secrets**; scoped access only from backend Service.
- Internal exposure only via **ClusterIP Service**.

Deployment on OpenShift (Conceptual Flow)

1. **Project Setup**
   - Create a dedicated project/namespace to isolate resources, quotas, and RBAC.
2. **Configuration & Secrets**
   - Prepare **ConfigMaps** for non-sensitive settings (frontend base URL, backend config).
   - Prepare **Secrets** for sensitive info (DB user, password, database name).
3. **Database Tier**
   - Deploy MySQL with a **PVC** sized appropriately for expected data.
   - Bind configuration from ConfigMaps/Secrets to the DB container environment.
   - Expose internally via a **ClusterIP Service**.
4. **Backend Tier**
   - Deploy the Go application as a separate Deployment.
   - Inject DB connection settings from ConfigMaps/Secrets.
   - Expose internally via a **ClusterIP Service** for the frontend to consume.

5. **Frontend Tier**
    - Deploy Nginx to serve static content and point to the backend Service URL.
    - Publicly expose via an **OpenShift Route**.
6. **Networking & Discovery**
    - Ensure Services are discoverable via internal DNS.
    - Confirm network policies (if used) allow only the required east-west traffic.
7. **Security & Governance**
    - Apply least-privilege **RBAC** for team roles.
    - Use **SCCs** only if needed; prefer default restricted where possible.
    - Separate build/runtime credentials and rotate Secrets periodically.
8. **Scalability & Resilience**
    - Independently scale each tier based on load characteristics.
    - Use readiness/liveness probes for automated recovery.
    - Optionally configure autoscaling policies where appropriate.
9. **Observability & Operations**
    - Collect application logs and platform logs centrally.
    - Track Pod/Deployment health, route availability, and DB metrics.
    - Establish basic alerts for pod restarts, PVC usage, and route health.

Verification Checklist (No Commands)

- Project exists and contains three Deployments (frontend, backend, database).
- Internal Services are present and resolvable; frontend can reach backend; backend can reach DB.
- Route is available and serves the frontend publicly.
- ConfigMaps and Secrets are mounted/injected as intended.
- PVC is bound and used by the database; data persists across restarts.
- Readiness/liveness checks pass; replicas reach “Ready” state.
- RBAC and access boundaries align with least privilege.

Deliverables (Suggested Evidence)

- Short summary describing the project structure and each tier.
- Screenshots of:
    - Project resources overview (Pods/Deployments/Services/Route/PVC).
    - Frontend reachable from the public URL.
    - Backend health/metrics dashboard view (if available).
    - PVC status in the storage view.
    - Config/Secrets references shown in workload details.

Expected Outcome

You’ll have a production-style, modular three-tier app on OpenShift with:

- Clear separation of concerns per tier.
- Secure configuration management (ConfigMaps & Secrets).
- Reliable internal service discovery and public exposure via Routes.
- Durable data with PVC-backed storage.

- Operational readiness through health checks, scaling, and observability.

Cleanup (Conceptual)

- Remove application workloads, Services, Routes, PVC/PV (if ephemeral), ConfigMaps, and Secrets when no longer needed.
- Optionally archive screenshots/evidence before teardown.

You are Welcome