Senior DevOps Engineer – Technical Assignment

Duration: 4-6 hours (actual working time)
Deadline: 3 days from receiving this assignment
Objective: Demonstrate hands-on expertise in Docker Swarm orchestration, stack management, containerization, CI/CD automation, and infrastructure migration strategies.

How to Read This Document

This assignment is comprehensive and designed to assess multiple areas of DevOps expertise. Here's how to approach it:

Time Management

- **Total Duration:** 4-6 hours (actual working time)
- **Deadline:** 3 days from the date you receive this assignment
- Each part includes estimated time allocation to help you pace yourself
- Time estimates are guidelines—adjust based on your experience level
- You don't need to complete it in one sitting; spread it over several days if needed

Deliverables Structure

Each task specifies what to submit: - **Configuration files:** YAML, scripts, Dockerfiles with inline comments - **Documentation:** Markdown files explaining your decisions and procedures - **Working code:** Scripts and automation tools that can be tested - **Analyses:** Written troubleshooting procedures and migration strategies

Flexibility & Choices

- **Application Stack:** Use our suggested repositories OR choose your own (Next.js/React/Vue + Node/Go/Python)
- **Tools:** Choose between Traefik or Nginx for ingress; GitHub Actions or GitLab CI for pipelines
- **Environment:** PlayWithDocker is recommended for quick setup, but you can use your own infrastructure

What We're Looking For

- **Practical knowledge:** Working configurations over theoretical descriptions
- **Best practices:** Production-ready, secure, and maintainable solutions
- **Problem-solving:** How you approach troubleshooting and operational challenges

- **Documentation:** Clear explanations of your architectural decisions and trade-offs

Optional vs. Required

- **Required tasks:** Parts 1-8 form the core assessment
- **Bonus challenges:** Optional extras to showcase additional skills
- **Multi-architecture builds:** Bonus, not required (mentioned in Part 7.2)

Submission Format

- Git repository with organized directory structure (see Submission Guidelines)
- Comprehensive README explaining how to deploy and test your solution
- Document any assumptions or questions you have

Tip: Read through the entire assignment first to understand the full scope, then tackle it section by section.

Assignment Overview

You will design and implement a production-ready Docker Swarm deployment for a microservices application stack. This assignment tests your deep understanding of Swarm mode, service orchestration, troubleshooting skills, and your ability to plan migration paths from Swarm to Kubernetes. You'll also demonstrate best practices for security, observability, and automation in Swarm environments.

Tools you can use

- PlayWithDocker: https://labs.play-with-docker.com/ (for quick Swarm cluster setup)
- Portainer: https://www.portainer.io/ (for Swarm management and UI-based stack deployment)
- Docker CLI: https://docs.docker.com/engine/swarm/ (for command-line Swarm management)
- Traefik: https://doc.traefik.io/traefik/ (for dynamic reverse proxy and load balancing)

Application Stack

You can either: - Use sample repositories we provide (recommended): - **Frontend:** https://github.com/vercel/next.js/tree/canary/examples/blog-starter - **Backend:** https://github.com/hagopj13/node-express-boilerplate

- Or choose your own frontend/backend stack (Next.js, React, Node.js, Go, Python, etc.)

Part 1: Docker Swarm Stack Deployment (90 minutes)

Task 1.1: Docker Compose Stack File

Create a production-ready `docker-compose.yml` for Docker Swarm mode containing: - **Frontend service** (Next.js/React) - multiple replicas with rolling updates - **Backend API** (Node.js/Go/Python) - multiple replicas with health checks - **PostgreSQL database** - single replica with volume persistence - **Redis cache** - single replica for session/caching - **Nginx/Traefik** - reverse proxy with SSL termination

Requirements: - Proper service placement constraints and resource limits - Health checks with appropriate intervals and retries - Secrets management for database credentials and API keys - Named volumes for data persistence - Overlay network configuration with proper isolation - Update and rollback configurations - Deploy labels for Traefik routing (if using Traefik)

Deliverable: `docker-compose.yml` with detailed inline comments

Task 1.2: Swarm Service Configuration Deep Dive

Configure the backend API service with: - **Deployment mode:** Replicated with multiple replicas (adjust based on PlayWithDocker node availability) - **Update config:** Parallelism of 2, delay between updates, failure action - **Rollback config:** Automatic rollback on failure - **Placement preferences:** Spread across available nodes - **Resource reservations and limits:** CPU and memory constraints - **Logging driver:** JSON-file or syslog with proper rotation - **Networks:** Attach to both frontend and backend networks

Deliverable: Annotated service definition explaining each configuration choice inside `docker-compose.yml`

Task 1.3: Docker Swarm Secrets & Configs

Demonstrate proper use of Swarm secrets and configs: - Create secrets for database credentials (username, password, connection string) - Create configs for application configuration files - Mount secrets securely in services (not as environment variables) - Show how to rotate secrets without downtime

Deliverable: Commands and YAML snippets for secrets/configs management

---

Part 2: Docker Swarm Networking & Service Discovery (60 minutes)

Task 2.1: Overlay Network Design

Design a multi-tier network architecture: - **Public network:** For ingress/load balancer services - **Frontend network:** For frontend backend communication - **Backend network:** For backend database communication - **Monitoring network:** For Prometheus/Grafana stack

Requirements: - Demonstrate network isolation between tiers - Configure encrypted overlay networks - Implement proper service discovery using DNS - Show how services communicate across networks

Deliverable: Network topology diagram and `docker network create` commands

Task 2.2: Ingress and Load Balancing

Configure ingress routing using either: - **Traefik v2/v3** as a Swarm service with labels-based routing - **Nginx** with manual upstream configuration

Requirements: - SSL/TLS termination with automatic certificate management - Path-based routing: `/` → frontend, `/api` → backend - Host-based routing for multiple domains - Health checks and automatic service discovery - Sticky sessions for the backend API - Rate limiting and security headers

Deliverable: Complete Traefik/Nginx configuration in stack file with detailed explanations

---

Part 3: Portainer Stack Management & API Automation (45 minutes)

Task 3.1: Portainer Stack Deployment

Install and configure a Portainer instance managing a Docker Swarm cluster.

Follow the instructions here Portainer CE Setup

Create documentation for: - Deploying your stack via Portainer UI vs. Portainer API - Using Portainer templates for repeatable deployments - Managing environment variables in Portainer stacks - Best practices for organizing stacks in Portainer (naming, tagging)

Deliverable: Step-by-step guide with screenshots or API curl commands

Task 3.2: Portainer API Automation

Write a Bash or Python script that: - Authenticates with Portainer API - Retrieves list of running stacks - Deploys/updates a stack from your `docker-compose.yml` file (created in Part 1) and environment variables - Validates deployment status and reports errors - Supports multiple Portainer endpoints

Deliverable: Working script with usage documentation

Part 4: Observability & Monitoring in Swarm (60 minutes)

Task 4.1: Prometheus Stack in Swarm

Deploy a monitoring stack as Swarm services: - **Prometheus** - with service discovery for Swarm services - **Grafana** - with persistent dashboard storage - **Node Exporter** - deployed globally on all nodes - **cAdvisor** - for container metrics

Requirements: - Configure Prometheus to scrape metrics from all services - Use Swarm service labels for dynamic discovery - Create a custom Grafana dashboard showing: - Service replica count and health - Node CPU/memory/disk usage - Container restart count - Network traffic per service

Deliverable: Prometheus config, Grafana dashboard JSON, and stack file

Task 4.2: Logging with Syslog/Fluentd

Configure centralized logging: - Deploy a logging driver configuration - Collect logs from all services - Parse and structure logs (JSON format preferred) - Implement log rotation and retention policies

Deliverable: Logging configuration and architecture explanation

Task 4.3: Alerting Configuration

Set up alerting rules for: - Service replica count drops below desired state - Node goes down or becomes unreachable - High memory usage (>90%) on any service - Backend API response time exceeds threshold - Database connection failures

Deliverable: Prometheus alerting rules and AlertManager configuration

---

Part 5: Swarm Operations & Troubleshooting (75 minutes)

Task 5.1: Rolling Updates & Rollbacks

Demonstrate your understanding of Swarm update strategies:

1. Perform a zero-downtime rolling update of the backend service
2. Simulate a failed deployment and trigger automatic rollback
3. Explain the difference between `update-parallelism`, `update-delay`, and `update-failure-action`
4. Show how to manually rollback a service to a previous version

Deliverable: Commands with explanations and configuration examples

Task 5.2: Service Failure Scenarios

Scenario 1: A service is showing replicas as running (e.g., 3/3 or 5/5), but requests are failing intermittently.

- What commands would you use to investigate?
- How do you check which replicas are actually healthy?
- How do you inspect task logs for a specific replica?
- How would you force-restart a problematic replica?

Scenario 2: After deploying a stack update, services are stuck in "starting" state.

- How do you check why tasks are failing to start?
- What are common reasons for services failing to deploy?
- How do you inspect task history and failure reasons?

Deliverable: Detailed troubleshooting procedures with commands

Task 5.3: Swarm Cluster Management

Demonstrate knowledge of: - Adding and removing nodes from a Swarm cluster - Promoting/demoting manager nodes - Draining nodes for maintenance - Recovering from quorum loss - Backing up and restoring Swarm state - Node labeling and constraint-based scheduling

Deliverable: Operational runbook with commands and best practices

---

Part 6: Migration Strategy: Swarm to Kubernetes (60 minutes)

Task 6.1: Migration Planning Document

Create a comprehensive migration plan for moving the application stack from Docker Swarm to Kubernetes.

Include: - **Assessment:** What Swarm features are being used and their K8s equivalents - Swarm services → K8s Deployments/StatefulSets - Swarm secrets/configs → K8s Secrets/ConfigMaps - Overlay networks → K8s Network Policies - Placement constraints → Node selectors/affinity - Volume mounts → PVs/PVCs

- **Migration approach:** Big-bang vs. phased migration
- **Tooling:** Kompose, custom scripts, or manual conversion
- **Testing strategy:** How to validate parity between environments
- **Rollback plan:** If migration fails
- **Timeline and risk assessment**

Deliverable: Migration strategy document (1000-1500 words)

Task 6.2: Helm Chart Conversion

Convert your Docker Compose stack file to a Kubernetes Helm chart: - Create equivalent Deployments, Services, and Ingress resources - Implement proper ConfigMaps and Secrets - Add resource limits and health checks - Include HPA for auto-scaling (bonus) - Document differences and trade-offs

Deliverable: Basic Helm chart structure with key templates

Part 7: CI/CD & Automation (45 minutes)

Task 7.1: GitHub Actions / GitLab CI Pipeline

Design a CI/CD workflow that: - Builds Docker images on push to `main` - Runs security scanning (optional but preferred) - Pushes images to GHCR or private registry - Deploys to Swarm via Portainer API or SSH - Includes manual approval for production deployments - Supports rollback capability

Deliverable: `.github/workflows/deploy.yml` or `.gitlab-ci.yml`

Task 7.2: Dockerfile Optimization

Create optimized Dockerfiles for frontend and backend: - Multi-stage builds to minimize image size - Proper layer caching for dependencies - Non-root user execution - Health check instructions - Security best practices

Bonus: Multi-architecture build support (amd64/arm64)

Deliverable: Dockerfiles with inline comments

Part 8: Security & Best Practices (30 minutes)

Task 8.1: Swarm Security Hardening

Create a security hardening checklist for Docker Swarm deployments: - **Secrets management:** Using Swarm secrets vs. environment variables - **Network security:** Encrypted overlay networks, network segmentation - **Image security:** Using trusted registries, image scanning, content trust - **Access control:** TLS

for Swarm API, certificate-based authentication - **Resource isolation:** Preventing noisy neighbor problems - **Audit logging:** Tracking who deployed what and when - **Node security:** OS hardening, Docker daemon security

Deliverable: Security checklist with implementation examples

Task 8.2: Production Readiness Checklist

Create a comprehensive checklist for production Swarm deployments: - High availability considerations (manager quorum, node distribution) - Backup and disaster recovery procedures - Update and maintenance windows - Resource capacity planning - Health check best practices - Logging and monitoring requirements

Deliverable: Production readiness checklist (Markdown format)

Submission Guidelines

Required Deliverables

1. **Docker Compose stack file** with all services configured
2. **Portainer automation scripts** (Bash/Python)
3. **Monitoring stack configuration** (Prometheus, Grafana)
4. **Migration strategy document** (Swarm to K8s)
5. **CI/CD pipeline configuration** (GitHub Actions/GitLab CI)
6. **Troubleshooting documentation** with detailed procedures
7. **Security and operations checklists**
8. A comprehensive `README.md` explaining:
   - Architecture overview and design decisions
   - How to deploy and test the Swarm stack
   - Swarm-specific considerations and trade-offs
   - Migration strategy rationale
   - Any assumptions or choices made

Format

- Submit as a Git repository (GitHub/GitLab)
- Organize files in clear directory structure:

```
docker-compose.yml
README.md
dockerfiles/
scripts/
monitoring/
ci-cd/
migration/
```

```
docs/
```

- Include inline comments in all configuration files
- Provide example commands for testing and deployment

Evaluation Criteria

- **Swarm Expertise (40%):** Deep understanding of Swarm services, networking, and operations
- **Problem-Solving (20%):** Thorough troubleshooting analyses and operational procedures
- **Migration Planning (15%):** Comprehensive and realistic Swarm-to-K8s strategy
- **Automation (10%):** Working CI/CD pipelines and Portainer API integration
- **Security (10%):** Production-ready security configurations
- **Documentation (5%):** Clear, maintainable documentation

Bonus Challenges (Optional)

1. **Multi-Architecture Builds:** Implement `buildx` for amd64/arm64 support in CI/CD
2. **High Availability:** Design 5-manager node cluster with proper quorum configuration
3. **Blue-Green Deployments:** Implement blue-green deployment strategy in Swarm
4. **Custom Metrics:** Export custom application metrics to Prometheus
5. **Chaos Engineering:** Simulate node failures and demonstrate self-healing
6. **Cost Optimization:** Document strategies for resource efficiency in Swarm vs. K8s
7. **Automated Backups:** Implement automated backup solution for Swarm state and volumes

---

Sample Repositories (Optional)

You're free to choose your own stack, but here are some suggested starting points:

Frontend Options: - Next.js Blog: https://github.com/vercel/next.js/tree/canary/examples/blog-starter - React Dashboard: https://github.com/creativetimofficial/material-dashboard-react - Vue.js Example: https://github.com/vuejs/examples

Backend Options: - Node.js/Express: https://github.com/hagopj13/node-express-boilerplate - Go API: https://github.com/gothinkster/golang-gin-realworld-example-app - Python/FastAPI: https://github.com/tiangolo/full-stack-fastapi-postgresql

Or use any application you're comfortable with!

Questions?

Document any assumptions or questions in your `README.md`. We value seeing your thought process and how you handle ambiguous requirements in complex infrastructure scenarios.

Focus Areas: We're particularly interested in your Swarm expertise, troubleshooting methodology, and practical migration experience from Swarm to Kubernetes.

Good luck! We look forward to reviewing your submission.