PRACTICAL EXERCISE

EX-40O

USAREUR-AF

EX_40O — Platform Engineer

Practical Exercise — SL 4O Proficiency

HEADQUARTERS
UNITED STATES ARMY EUROPE AND AFRICA
(USAREUR-AF)
Wiesbaden, Germany

26 MARCH 2026

DRAFT — UNOFFICIAL — NOT FOR OPERATIONAL USE

EX_40O — PLATFORM ENGINEER

PRACTICAL EXERCISE — SL 4O PROFICIENCY

Field	Value
Version	1.0 — March 2026
Prerequisite	SL 3 REQUIRED; SL 4O — Platform Engineer Technical Manual (and SL 1 through SL 2)
Duration	3–4 hours
Environment	Kubernetes training cluster, container registry, CI/CD tooling, Git — see ENVIRONMENT_SETUP.md

COMPANION RESOURCES

Resource	Reference
Technical Manual	SL 4O — Platform Engineer
Syllabus	SYLLABUS_TM40O
Pre-Exercise Exam	EXAM_TM40O_PRE
Post-Exercise Exam	EXAM_TM40O_POST
Continuation Track	SL 5O — Advanced Platform Engineer

WFF AWARENESS

The platform environment built in this exercise provides the infrastructure layer that all MSS applications depend on. Evaluators should verify that security controls are properly implemented (not bypassed for convenience), deployments are declarative and reproducible, and monitoring covers the health indicators that WFF-facing applications depend on.

SCENARIO

The OPDATA team needs a new platform environment provisioned for an MSS application team. The environment must include namespace isolation, a CI/CD pipeline with security gates, container hardening, GitOps-based deployment, monitoring, and an air-gapped deployment package. The trainee will build this environment from the ground up, applying SL 4O principles.

Training cluster: A shared Kubernetes training cluster with namespace-admin privileges per trainee.

TASK LIST

Task 1 — Namespace and Resource Governance (30 min)

- ☐ Create a dedicated namespace for the application team
- ☐ Apply resource quotas: CPU and memory limits appropriate for a small application
- ☐ Apply LimitRanges: default container resource requests and limits
- ☐ Implement RBAC: create a ServiceAccount with deploy permissions; verify it cannot access other namespaces

Standard	Criteria
Go	Namespace created; resource quotas enforced; LimitRange applied; RBAC verified (positive and negative test)
No-Go	No resource quotas; RBAC not tested; ServiceAccount has cluster-wide permissions

Task 2 — Workload Deployment (30 min)

- ☐ Deploy a sample application using a Deployment manifest (provided in training package)
- ☐ Configure resource requests and limits on all containers
- ☐ Add liveness and readiness probes with appropriate thresholds
- ☐ Expose the application via a Service
- ☐ Verify the application is running and accessible

Standard	Criteria
Go	Deployment is declarative (YAML, not imperative kubectl commands); resource limits set; probes configured; application accessible
No-Go	Imperative deployment; no resource limits; no health probes

Task 3 — Container Hardening (30 min)

- ☐ Build a container image from a provided Dockerfile using multi-stage build
- ☐ Configure non-root execution (runAsNonRoot, drop all capabilities)
- ☐ Scan the image for vulnerabilities; document findings
- ☐ Pin the image by SHA256 digest in the deployment manifest (not by tag)

Standard	Criteria
Go	Multi-stage build used; non-root verified; scan completed with findings documented; digest pinning in manifest
No-Go	Running as root; no vulnerability scan; image referenced by mutable tag

Task 4 — GitOps and CI/CD Pipeline (45 min)

- ☐ Store all manifests in a Git repository
- ☐ Configure a GitOps controller to reconcile the namespace from Git
- ☐ Build a CI/CD pipeline with at minimum: build, test, scan, deploy stages
- ☐ Add at least one security gate that blocks deployment on failure (e.g., image vulnerability scan)
- ☐ Demonstrate: commit a change → pipeline runs → application updates automatically

Standard	Criteria
Go	Git is the source of truth; GitOps reconciliation demonstrated; pipeline has security gate that blocks on failure; end-to-end flow demonstrated
No-Go	Manual deployment bypassing Git; no security gates; pipeline does not block on scan failure

Task 5 — Network Policy and Monitoring (30 min)

- ☐ Implement a default-deny network policy for the namespace
- ☐ Add explicit allow rules for required traffic (application ← service, application → required external endpoints)
- ☐ Configure monitoring: resource utilization dashboard and at least one actionable alert (e.g., pod restart count > threshold)
- ☐ Verify the alert fires by simulating a failure condition

Standard	Criteria
Go	Default-deny network policy applied; allow rules tested (positive and negative); monitoring dashboard shows resource metrics; alert fires on simulated failure

Standard	Criteria
No-Go	No network policy; allow-all policy; monitoring not configured; alert not tested

Task 6 — Air-Gapped Deployment Package (30 min)

- ☐ Bundle the application container image(s) for offline transfer
- ☐ Bundle all Kubernetes manifests and configuration
- ☐ Document the deployment procedure for an air-gapped environment (step-by-step, assuming no external network)
- ☐ Verify: deploy the application to a second namespace using only the bundled artifacts (simulating air gap)

Standard	Criteria
Go	Bundle contains all required artifacts; deployment procedure documented; application deploys successfully from bundle alone (no external network pulls)
No-Go	Bundle incomplete (missing images or config); deployment pulls from external registry; procedure not documented

EVALUATOR NOTES

- Evaluate security posture, not just functionality. An application that works but runs as root with no network policies = No-Go.
- Verify that deployments are truly declarative — all state in Git, reconciled by GitOps controller. `kubectl apply` from a laptop is not GitOps.
- Test RBAC with both positive (authorized action succeeds) and negative (unauthorized action is denied) tests.
- The air-gapped deployment test in Task 6 should genuinely simulate no external network — disable external registry access for the target namespace.