Security Report — Secure Microservice (Test)

Summary of risks identified (expected scan categories)

- Container image vulnerabilities (OS/packages): HIGH/CRITICAL should block push.
- Hardcoded secrets: Semgrep checks for them; any findings must be removed.
- Insecure IAM/IaC patterns: tfsec will find broad permissions or public S3/ECR.
- Runtime misconfigurations: containers running as root, writable rootfs, elevated capabilities.

What was implemented

- Docker:
  - Multi-stage build to reduce final image surface.
  - Runtime user is non-root (appuser), NODE_ENV=production set.
  - .dockerignore created to avoid leaking files.
- CI/CD:
  - Semgrep static code analysis fails build on findings.
  - Trivy scans the container and fails the job for HIGH/CRITICAL.
  - tfsec scans Terraform for insecure IaC patterns.
  - Push to container registry only occurs if all security gates pass.
- Secrets:
  - No secrets in source. GitHub Actions Secrets used for registry and DB string.
  - Example k8s secret template provided; instructions to create k8s secret via kubectl from GitHub Actions secrets.
- Runtime security:
  - Kubernetes pod `securityContext` enforces `runAsNonRoot`, `runAsUser`, `readOnlyRootFilesystem`, `allowPrivilegeEscalation: false`, drop all capabilities.
  - Recommendation added to enable Seccomp/AppArmor and runtime detection (Falco) in production.

Suggestions for production-grade hardening

- Replace Alpine with distroless Node runtime and use minimal OS packages.
- Add reproducible SBOM generation (Syft) and store artifacts.
- Add a policy engine (OPA/Gatekeeper) to enforce cluster policies.
- Use short-lived credentials / OIDC for cloud provider access in CI rather than static secrets.
- Enable image signing (cosign) and enforce signed images only in admission controllers.
- Run runtime monitoring (Falco) and a SOC pipeline to ingest alerts.
- Add E2E tests and a proper vulnerability triage workflow (owner assignment, timelines).