# Entregable 2 — CI/CD Testing Pipeline (FastTicket)

## A. Diseño de Entornos (Dev / Staging / Prod)

* Dev: iteración rápida y pruebas locales. Uso de imágenes :latest solo en dev, port-forward para endpoints, y costos mínimos. Seguridad base: secrets en Kubernetes (ConfigMap/Secret), RBAC por namespace (dev), y limitación de recursos para evitar abuso.
* Staging (AWS us-east-1, EKS fastticket-eks, ns=staging): entorno casi idéntico a producción. Despliegue vía Kustomize con imágenes docker.io/fercanap:staging (tags inmutables), 1 réplica por servicio (cost-aware), liveness/readiness probes, gp3 para PostgreSQL (PVC). Autenticación GitHub→AWS por OIDC (sin llaves largas). Superficie expuesta solo cuando se valida (Service type LoadBalancer o port-forward). Secrets cifrados por Kubernetes y mínimo de permisos (IRSA para add-ons, principio de least privilege).
* Prod: mismas políticas endurecidas que staging, escalamiento horizontal, rotación de secretos, y límites de recursos más estrictos. Ingress/TLS, monitoreo y políticas de imagen (solo registries aprobados) antes del go-live.

## B. Selección y Justificación de Herramientas

* CI/CD: GitHub Actions con OIDC a AWS (sin secretos estáticos). Rápido, integrado y low‑cost.
* SAST: CodeQL para JavaScript y Python. Alta cobertura, análisis incremental, sin costo en repos públicos/educativos.
* SCA: pip-audit (backend) y npm audit (frontend) para detectar vulnerabilidades de dependencias. Alternativa: Snyk si se habilita.
* Container Security: Trivy (HIGH/CRITICAL con ignore‑unfixed) + Hadolint para buenas prácticas en Dockerfile. Balancea velocidad y señal útil.
* DAST: OWASP ZAP Baseline contra el backend en staging mediante port-forward (127.0.0.1), sin tráfico público innecesario.
* IaC: tfsec y Checkov sobre Terraform (infra/). Visibilidad temprana de riesgos en cloud.

## C. Etapas y Gates del Pipeline

1) Code (Shift‑Left): lint/tests unitarios, revisión PR. Gate: tests verdes y sin fallas críticas de lint.

2) Build & SCA: build de imágenes backend/frontend; pip-audit y npm audit paralelos. Gate: 0 CRITICAL y HIGH ≤ umbral.

3) SAST: CodeQL para JS/Python. Gate: sin alertas “security severe/critical” nuevas.

4) Deploy to Staging: push de imágenes :staging y kustomize apply. Gate: rollout completo y smoke test /health=200.

5) Test & DAST: pruebas funcionales básicas y ZAP Baseline vía port‑forward. Gate: sin hallazgos de riesgo alto en ZAP.

6) Hardening Final: políticas de imagen (tags inmutables), RBAC/NS boundaries, secrets y variables mínimas, recursos y probes, escaneo Trivy de imágenes desplegadas. Gate: 0 CRITICAL y HIGH ≤ umbral global.

7) Deploy to Prod: promoción de tag aprobado. Gate: firma/aprobación manual y verificación post‑deploy.

Racional de orden: SAST antecede a DAST para eliminar fallas de código y dependencia antes de exponer endpoints; reduce falsos positivos, acelera el ciclo y evita gastar tiempo en pruebas dinámicas sobre builds ya rechazadas por fuentes estáticas.