anoBytes | TECHNOLOGY SOLUTIONS

DevSecOps cheat sheet

DevSecOps is DevOps with the security work built into the same pipeline that builds and ships the code. What to check at every stage, the open-source tools that do it, and the frameworks to build on.

PIPELINE CONTROLS · OPEN-SOURCE TOOLING · SUPPLY CHAIN · NIST SSDF · OWASP · SLSA · CIS

General educational guidance, not affiliated with or endorsed by any project or vendor named. Try the commands in a test pipeline first. Framework references follow NIST SP 800-218 (SSDF) v1.1, OWASP ASVS 5.0, SLSA v1.2, and the CIS Benchmarks named inline. Published 2026-08-13.

WHERE SECURITY FITS THE PIPELINE

Every stage gets its own checks. The ones in the pipeline run on each change, the way tests do (the "shift left"), so findings reach the developer while the change is still open and cheap to fix.

```
plan -> code -> build -> test -> release -> operate
--------------------------------------------------
1 plan     threat model; requirements from ASVS
2 code     review everything; secrets + SAST pre-push
3 build    SCA + IaC + container scan; SBOM each build
4 test     DAST against a staging deploy
5 release  sign + provenance; verify before deploy
6 operate  monitor; patch what is exploited; feed back
```

SAST static analysis: reads source for vulnerable patterns before it ever runs. Semgrep; CodeQL for open source.

SCA software composition analysis: checks the packages you depend on against known-vulnerability databases. OSV-Scanner, Trivy, Grype.

secrets scanning finds keys, tokens, and passwords in the repo and its history. gitleaks, TruffleHog.

IaC scanning infrastructure as code: reads Terraform, Kubernetes, and Dockerfiles for insecure settings before anything is provisioned. Trivy, Checkov.

DAST dynamic analysis: probes the running application from outside, the way an attacker would. ZAP.

SBOM software bill of materials: the machine-readable parts list of what an artifact contains. Syft.

provenance a signed record of how, where, and from what an artifact was built. Sigstore, SLSA.

1 PLAN: DECIDE WHAT SECURE MEANS FIRST

Threat modeling asks four questions of each new design: what are we building, what can go wrong, what are we doing about it, and did we do enough? Write the answers down; they become test cases and review points downstream.

Take security requirements from OWASP ASVS 5.0 (the Application Security Verification Standard) rather than inventing your own: Level 1 is the starting set, Level 2 covers standard security practice, Level 3 the advanced, high-assurance requirements. NIST's Secure Software Development Framework (SSDF, SP 800-218) names the organizational half: requirements, roles, toolchains, criteria for security checks, and hardened development environments (its PO practices).

2 CODE: CATCH SECRETS BEFORE THE PUSH

Run the fast checks as a pre-commit hook and again in CI: the hook gives the developer a two-second answer, CI catches whoever skipped the hook.

```
# secrets, in the working tree and the whole history
gitleaks git .     # scan the repo, history included
gitleaks dir .     # scan a plain directory
# SAST with the public registry rules
semgrep scan --config auto
# platform side: require review before merge, and turn on
# push protection so a secret is blocked at push time
```

A secret that leaked is revoked and reissued; deleting the commit does not un-leak it. Code from an AI assistant enters through the same review and the same scanners as anyone else's change.

3A BUILD: KNOW WHAT YOU SHIP

```
osv-scanner scan -r .  # lockfiles vs the OSV database
trivy fs .             # deps + secrets across the tree
grype sbom:./sbom.json  # rescan an SBOM, no rebuild
```

Commit lockfiles and pin versions so a build cannot silently pull something new; let a bot (Renovate, Dependabot) open update pull requests so patching stays small and routine. The packages your packages pull in are yours to patch too. Keep each build's SBOM in a store that re-checks it as new advisories land (Dependency-Track).

The ChainDrop worm (August 2026) republished 400+ npm packages as malicious patch releases; a committed lockfile is why a build does not pull those unasked. When the next xz-utils backdoor (CVE-2024-3094) lands, the SBOM archive answers "do we ship it, and where?" in minutes.

3B BUILD: SCAN CONTAINERS AND IAC

```
trivy image <image>   # OS and app packages, secrets
trivy config .        # Terraform, K8s, Dockerfiles
checkov -d .          # IaC checks, policy as code
```

Build images small: a minimal base pinned by digest, multi-stage builds so compilers stay out of the final layer, and a non-root user (CIS Docker Benchmark v1.8.0, 4.1). In Kubernetes, enforce at admission with policy as code (Kyverno or OPA Gatekeeper): only signed images, no privileged pods; audit nodes against the CIS Kubernetes Benchmark with kube-bench.

THE PIPELINE IS PRODUCTION: HARDEN IT

CI runs code from many sources with credentials to your cloud, your registry, and your repos. The OWASP Top 10 CI/CD Security Risks and the CIS GitHub Benchmark v1.2.0 cover pipeline security in depth; start with these:

no stored cloud keys jobs fetch short-lived cloud credentials over OIDC (OpenID Connect) federation; no long-lived key exists to steal.

pin actions to a commit SHA a re-pointed tag is how tj-actions (CVE-2025-30066) exposed CI secrets in build logs.

least-privilege tokens the job token is read-only by default; write scopes are granted per job.

protected branches review plus passing checks to merge; no force-push, no direct pushes to main.

ephemeral runners each job starts from a clean machine; secrets are masked in logs, never echoed.

runtime secrets live in a manager a secrets manager (OpenBao, your cloud's own) injects them at deploy; the repo and CI config hold references, never values.

4 TEST: PROBE THE RUNNING APP

```
docker run -t ghcr.io/zaproxy/zaproxy:stable \
  zap-baseline.py -t https://staging.example.com
```

The baseline scan spiders briefly and reports what passive analysis sees (missing headers, cookie flags, leaky errors); it attacks nothing, so it can run on every staging deploy. The active scan (zap-full-scan.py) does attack: run it with permission and a maintenance window. An unauthenticated scan covers only what an anonymous visitor can reach.

5 RELEASE: SBOM, SIGNATURE, PROVENANCE

Generate the SBOM in the build, sign the artifact by digest with the pipeline's own identity, and publish provenance; at deploy, verify all of it mechanically.

```
syft <image> -o cyclonedx-json > sbom.json
# keyless artifact signing with the job's OIDC identity
cosign sign <image>@<digest>
cosign verify <image>@<digest> \
  --certificate-identity <the signing workflow> \
  --certificate-oidc-issuer <the CI provider>
# identity, on GitHub Actions: the workflow URL + ref
#  https://github.com/acme/shop/.github/workflows/
#  release.yml@refs/tags/v1.2
# issuer: https://token.actions.githubusercontent.com
```

SLSA v1.2 (Supply-chain Levels for Software Artifacts) grades the build: L1 provenance exists, L2 a hosted platform signs it, L3 the build resists tampering; its new Source Track grades the repo the same way. CycloneDX (ECMA-424) and SPDX (ISO/IEC 5962) are the two main SBOM formats; CISA's 2026 Minimum Elements (July 2026) set what a complete one carries.

6 OPERATE: PATCH WHAT IS EXPLOITED FIRST

The SBOM store re-flags shipped builds as advisories land; runtime detection (Falco) watches the workload. Severity scores alone over-queue the work. Fix first what is being exploited (CISA's Known Exploited Vulnerabilities catalog, KEV), then what is likely next (EPSS, the Exploit Prediction Scoring System, gives each CVE a probability), then the rest by severity and reachability. Record rule-outs as VEX statements (vulnerability exploitability exchange: "present, and no risk here, because…") so scanners stop re-opening them.

Publish a security.txt (RFC 9116) and a disclosure policy so reporters reach you; SSDF's RV practices close the loop: confirm, fix, trace the root cause.

THE FRAMEWORKS, AND WHAT EACH ONE IS FOR

FRAMEWORK	USE IT FOR
NIST SP 800-218 (SSDF) v1.1	SDLC practices: prepare, protect, produce, respond
OWASP ASVS 5.0	security requirements, three levels
OWASP Top 10:2025	top web app risks; supply chain enters at A03
OWASP Top 10 CI/CD Security Risks	the ten CICD-SEC pipeline attack classes
OWASP Cheat Sheet Series	secure coding how-tos, per topic
SLSA v1.2	build and source integrity levels
CIS Controls v8.1, Control 16	prioritized app security safeguards
CIS Benchmarks	hardening baselines: GitHub, Docker, Kubernetes
NIST SP 800-204D	supply chain security in CI/CD
OWASP SAMM / DSOMM	measure maturity, plan the roadmap
OWASP GenAI LLM Top 10 2026	risks when the app embeds an LLM

START WHERE THE RISK IS

```
a  turn on the platform's features: push protection,
   dependency alerts and updates, branch protection
b  secrets scanning + SCA in CI, report-only;
   fix the criticals they surface
c  make both blocking for NEW findings; baseline the
   backlog and burn it down on a cadence
d  add SAST, IaC, and image scanning; tune rules
   until findings stay believable
e  SBOM generation, signing, provenance in the
   release job; verify at deploy
f  DAST on staging; then measure: time to remediate
   by severity, share of builds with SBOM + provenance
```

A gate that fails builds on findings nobody triages gets disabled within weeks. Block on new, confirmed findings; schedule the rest.

Regulatory dates: the EU Cyber Resilience Act (CRA, covering products with digital elements placed on the EU market, paid or free) starts its reporting duty 2026-09-11: actively exploited vulnerabilities and severe incidents go to the designated CSIRT (incident response team) and ENISA; obligations apply in full from 2027-12-11. In the US, OMB M-26-05 (January 2026) ended the government-wide attestation mandate; agencies may still require SSDF-based attestations by contract.