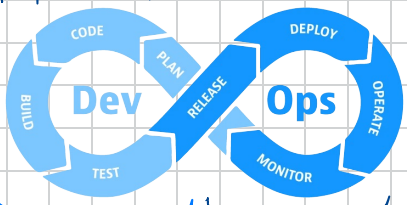
[CONTAINER SECURITY & STATIC APPLICATION TESTING]

DevOps Pipeline

automated processes & tools to allow collab btwn devs + ops. pros to build & deploy code to production environment

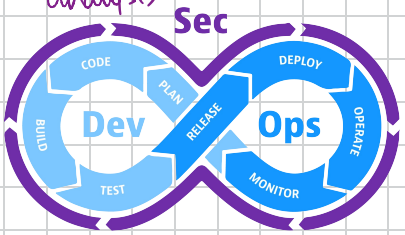


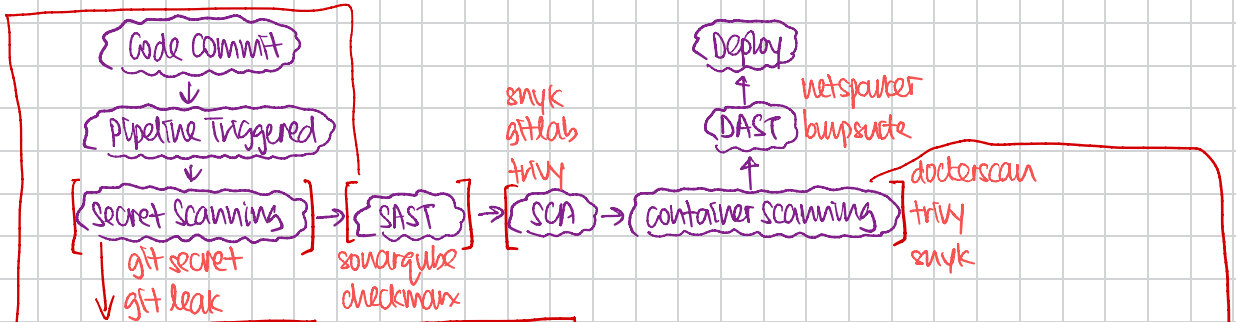
Continuous Integration (CI) | Continuous Development/Delivery (CD)

DevSecOps

DevOps + security practices & tooling at diff. stages

* allows for security scanning, policy enforcement & static analysis





Secret Scanning

scanning code repos & other data srcs for sensitive info aka. secrets

ex., pswds, access keys, certs, tokens

* tool: Gitleaks

Challenges

* ∄ 100% reliable; secret scanning tools try to identify secrets by "grepping" thru src according to rules (regex)
  - ⇒ high false (+)ve rate
  - ↳ ∄ rule to detect specific pattern of secret
    - ⇒ high false (−)ve rate
* disclosed secret must be revoked & replaced
  - ⇒ operational disruption; time consuming

Best Practices: Secrets

* avoid hard-coding creds. in app src code
  - ↳ parse/generate secrets at runtime (ex., w Docker Secret/config files mounted to Docker container)
* use 3rd party key/pswd mgmt solutions to securely store secret

Static Application Security Testing (SAST)

examine app component wout executing them by only analyzing src code

* identify potential security vulns.

ex., SonarQube, Semgrep, mobfscan

(+) scales well; can run on various software & repeatedly (for nightly builds, continuous integration)

(+) identify certain well-known vulns. & misconfigs (ex., network, cryptography)

(+) helps w highlighting problematic code & affected code snippet (by filename, location, line #)

(−) difficult to automate searches for many types of security vulns. (ex., auth. problems, access ctrl issues)

(−) high # false (+)ves

(−) difficult to prove that identified security issue is actual vuln.

(−) manual customization & ongoing maintenance

Software Composition Analysis (SCA)

process of tracking & analyzing 3rd party dependencies used by a software

* impt process; identify & categorize vulns. that might exist in software (CVE, CWE)
* in context of Docker container, software is pkged together w client-side libs (ex., Node.js) & sys. pkgs that support Docker container op.
* container scan = SCA scan + container-specific feats. (ex., scanning each layer separately, secrets detection, misconfig. detection)

Fixing vulns. in SCA

* upgrade lib. to "nearest" ver. free of vulns.
  - ↳ major version upgrades may have several changes in API usage; code refactoring necessary
  - ↳ do even if lib. ∄ called; ∄ false (+)ves in SCA!
* remove lib. if ∄ needed
* change to another lib
* accept risk; security team to track it in their risk register

Linting — VS

continuously inspect code quality to perform static code analysis

* detect bugs or "code smells", ∄ security vulns.

ex., SonarQube, PyLint (Python), Checkstyle (Java)

Terminology

False Positive: occurs when scanning tool incorrectly flags security vuln.

* test case fails w identified vuln, but none exists & finality works correctly

False Negative: occurs when scanning tool ∄ flag out security vuln.

* test case clears wout identifying vuln, but vuln. exists

SonarQube Rules

* bug: coding mistake that can lead to error or unexpected behavior at runtime
* vulnerability/security hotspot: potential open atk pt. in code
* code smell: maintainability issue that makes code confusing & difficult to maintain

SonarQube

linting & SAST tool fm Sonar;
helps in developing clean & secure code
* available as exe. or docker img

Trivy

multifnal security scanner, open-srced by Aqua Security
* can scan: → container img
  → file sys. → vm img
  → Git repo → kubernetes
* can find:
  - ↳ OS pkgs & software dependencies in use (SBOM)
  - ↳ known vulnerabilities
  - ↳ IaC issues & misconfigs
    - ↳ Infrastructure as Code
  - ↳ sensitive info & secrets
  - ↳ software licenses

Software Bill of Materials (SBOM)

provides detailed inventory of components & dependencies used in software project
* also lists:
  - ↳ licenses governing these components
  - ↳ vers. + patch status of components
* helps in identifying any associated security or license risks
* listed info for Docker containers:
  - ↳ Docker img used & associated props.
  - ↳ libs. & its dependencies

Container Security: Best Practices

* never run container as root user!
  - ↳ specify non-root user w/ just enough permission to execute app
* remove OS pkgs & app. libs. if ✗ needed by target app.
  - ↳ achievable by building docker img in stages;
    each stage only consists of required pkgs built fm previous stage
* change ownership of exes. to root & make them ✗ world-writeable
* use trusted base imgs
* use img w/ version that comes w/ latest security patches
  - ↳ minimizes # vulns. found in OS pkgs

Vulnerability Categories

Common Vulnerabilities & Exposures (CVEs)
* list of publicly-known vulns. found in open src & commercial software apps.
* software specific
ex., CVE-2023-25956

Common Weakness Enumerations (CWEs)
* list of publicly known security weaknesses that could lead to security issues
* generic; ✗ restricted to software
ex., CWE-200