02 DevSecOps

Overview - DevSecOps

DevSecOps is the practice of integrating **security** practices into the **DevOps** process. It is a cultural change that aims to bake security into every stage of the software development lifecycle (SDLC)—from initial code commitment to deployment and monitoring—rather than treating it as an afterthought.

The core principle is that **security is everyone's responsibility**, not just the security team's.

Attribute	Traditional Security (Waterfall/Later Stage)	DevSecOps (Shift-Left)
When Security Happens	Late in the process (Staging/Pre-Production).	Throughout the entire SDLC.
Who Owns Security	Dedicated Security Team (Gatekeepers).	Developers, Operations, and Security (Shared Ownership).
Nature of Security	Manual, periodic audits and vulnerability scans.	Automated, continuous testing, and real-time feedback.
Vulnerability Cost	Expensive to fix (requires full regression/rework).	Cheap to fix (caught early by the developer).

Security Shift-Left Approach

The **"Shift-Left"** concept is the foundational philosophy of DevSecOps. It means moving security activities and testing from the later stages of the SDLC (like QA or production) **earlier** into the development process.

Key Aspects of Shifting Left:

1. **Early Feedback:** Developers get immediate feedback on security flaws as they write code, making remediation faster and simpler.
2. **Proactive vs. Reactive:** Moves the team from reactively fixing discovered breaches to proactively preventing vulnerabilities from being merged.
3. **Tool Integration:** Embedding security tools (like vulnerability scanners, static analysis) directly into the developer's Integrated Development Environment (**IDE**) and the automated build process.
4. **Security Education:** Training developers to write secure code from the start, following secure coding standards.

Security in CI/CD Pipelines

Integrating security tools directly into the **Continuous Integration (CI)** and **Continuous Delivery/Deployment (CD)** pipeline is the *technical* manifestation of the Shift-Left approach. The

pipeline becomes the enforcement point for security policy.

Key Security Gates in the Pipeline:

1. **Commit/Build Phase (CI):**
   - **Secrets Scanning:** Checking code commits for hardcoded credentials, API keys, or private keys.
   - **Linting/Static Analysis (SAST):** Checking code quality and identifying common security flaws and coding errors.
   - **Dependency Scanning:** Checking the application's dependencies (libraries, frameworks) for known Common Vulnerabilities and Exposures (**CVEs**).
2. **Testing/Staging Phase (CD):**
   - **Dynamic Analysis (DAST):** Running automated black-box tests against the running application to simulate attacks and find runtime vulnerabilities (e.g., SQL injection, XSS).
   - **Infrastructure as Code (IaC) Scanning:** Scanning configuration files (e.g., Terraform, CloudFormation) for security misconfigurations before deployment.
   - **Container Security Scanning:** Checking Docker images for vulnerabilities and adherence to security best practices.
3. **Deployment/Release Phase:**
   - **Runtime Monitoring:** Ensuring adequate logging and monitoring is in place (e.g., using logging tools or Security Information and Event Management (SIEM) systems).
   - **Policy Enforcement:** Automated rollbacks or blocking deployments if critical security thresholds (e.g., high-severity CVE count) are exceeded.

Static Code Analysis Tools (SAST)

Static Application Security Testing (SAST) tools analyze application source code, byte code, or binary code *without* executing the program. They are fundamental to the CI phase of DevSecOps.

Feature	Description	Benefits for DevSecOps
Function	Scans non-running code to find vulnerabilities, logic flaws, and coding standard violations.	**Shifts Furthest Left:** Finds issues immediately upon code commit, often integrating directly into the IDE.
Vulnerability Types	Input validation errors, buffer overflows, race conditions, weak cryptographic practices.	Identifies deep-seated architectural or logical flaws that dynamic tests might miss.
Execution Context	Done during the **Build Phase** before deployment.	Provides fast feedback, allowing developers to fix issues quickly before they merge code.
Key Tools	SonarQube, Checkmarx, Fortify, Semgrep.	Reduces the security debt passed down the pipeline.

> **Note:** SAST has a higher rate of **False Positives** compared to DAST, requiring proper tuning and review by developers.

Secrets Management Strategies

A **secret** is any piece of sensitive data needed by an application or user to access resources, such as API keys, database credentials, cryptographic keys, and OAuth tokens. Hardcoding secrets directly into source code is one of the most common and dangerous security flaws.

Goal of Secrets Management:

To **centralize**, **secure**, and **control access** to secrets across all environments.

Core Strategies:

1. **Never Hardcode Secrets:** Use configuration files that are checked into source control. Instead, rely on environment variables or, ideally, a dedicated secrets manager.
2. **Use Dedicated Secrets Management Vaults:** Implement a secure, centralized system designed for storing and retrieving secrets.
   - **Examples:** HashiCorp Vault, AWS Secrets Manager, Azure Key Vault, Google Secret Manager.
   - **Process:** Applications fetch the secret at runtime from the vault using an authenticated identity (e.g., an IAM role or a Kubernetes Service Account).
3. **Principle of Least Privilege (PoLP):** Ensure applications can only access the specific secrets they need, and nothing more. Access to the secrets vault should be strictly controlled by identity and role.
4. **Automatic Rotation:** Configure secrets in the vault to automatically change on a schedule (e.g., every 90 days) to minimize the impact of a compromised secret.
5. **Secrets Scanning in CI:** Use tools like **Gitleaks** or **TruffleHog** to scan every new code commit to ensure no secrets have accidentally been pushed to the code repository.

https://youtu.be/DWww_RXYIzQ

https://medium.com/hellocode/all-in-one-deploy-python-flask-app-to-aws-ec2-b918fc6abf6e