DEVSECOPS COURSE

DEVSECOPS IN PRACTICAL

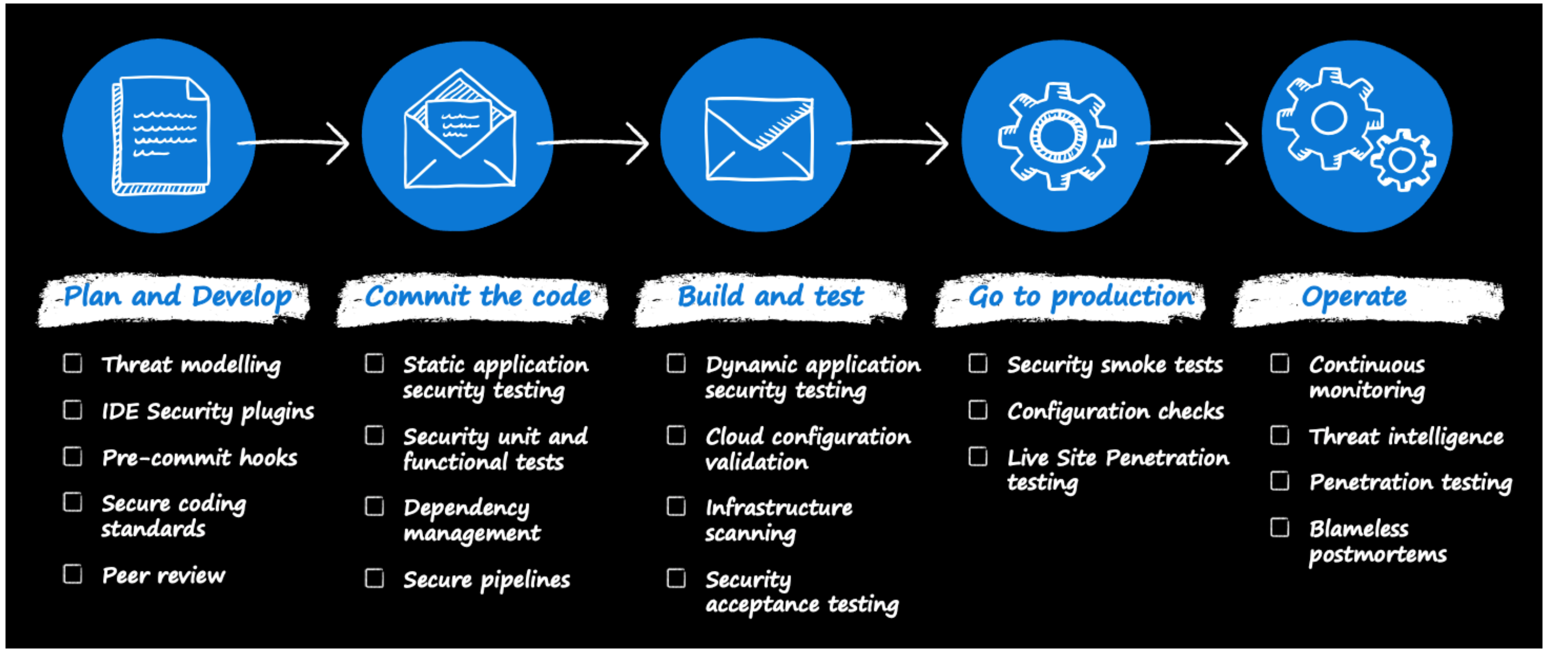
TRAINER: TRAN HUU HOA

AGENDA

- Introduction
- DevSecOps life-cycle
- SAST Tool
- DAST Tool
- SCA Tool

INTRODUCTION

DevSecOps is an enhancement to DevOps that integrates security into every aspect of the software development process. The primary goal is to address security issues right from the project's inception. In this framework, the entire team takes responsibility not only for quality assurance and code integration but also for security.



DEVSECOPS LIFECYCLE

DevSecOps lifecycle serves as the backbone of security enhancement within the software development continuum. It integrates security into every phase of the software development lifecycle, from initial design through integration, testing, delivery, and deployment. By automating the integration of security practices, DevSecOps reduces the risk of deploying software with misconfigurations and other vulnerabilities that bad actors can exploit

DEVSECOPS LIFECYCLE

- Cultural Shift: Foster a security-first mindset across teams. Encourage collaboration between development, security, and operations personnel. Emphasize shared responsibility for security.
- Automation: Automate security checks and testing throughout the development pipeline. Use tools like static code analysis, vulnerability scanning, and penetration testing.
- Secure Coding Practices: Train developers on secure coding practices. Address common vulnerabilities (e.g., OWASP Top Ten) during code reviews.
- Infrastructure as Code (IaC): Apply security controls to IaC templates. Use tools like Terraform or CloudFormation to define infrastructure securely.
- Continuous Monitoring: Monitor applications and infrastructure for security threats. Implement logging, intrusion detection, and real-time alerts.
- Security Testing: Integrate security testing (SAST, DAST, and SCA) into CI/CD pipelines. Fail builds with critical vulnerabilities.
- Secret Management: Securely manage secrets (API keys, passwords) using tools like HashiCorp Vault or AWS Secrets Manager.
- Compliance and Auditing: Ensure compliance with industry standards (e.g., PCI DSS, GDPR). Regularly audit security controls.

SAST TOOLS

- Purpose: SAST tools scan source code, binary, or byte code to find vulnerabilities.
- White-Box Testing: They operate as white-box testing tools, revealing the root cause of security issues.
- Vulnerability Detection: SAST tools help detect issues like buffer overflows, SQL injection flaws, and more.
- Strengths: Scalability that can be run on large software projects and repeatedly. Highlights problematic code with precise details (filename, location, line number).
- Weaknesses:
  - Limited automation for certain security vulnerabilities (e.g., authentication, access control).
  - High false positive rates.
  - Difficulty analyzing code that can’t be compiled.

SAST TOOLS

Selection Criteria:

- Support for your programming language.
- Ability to detect vulnerabilities (e.g., OWASP Top Ten).
- Accuracy (false positive/negative rates).
- Integration with developer IDEs and CI/CD tools.
- License cost and interoperability

Some popular SAST tools:

- Klocwork: Works with C, C#, C++, and Java codebases. It detects issues like division by zero, null pointer problems, and array out-of-bounds errors without running the code. Klocwork adheres to various coding and security standards.
- Semgrep: An open-source SAST tool supporting over 30 languages and more than 2750 rules for detecting security vulnerabilities. It integrates with CI/CD tools like GitHub and GitLab.
- Aikido Security, Checkmarx, and Contrast Security are also notable SAST tools.

SAST TOOLS

SonarQube is a code quality assurance tool that collects and analyzes source code, providing reports on the code quality of your project

- **Purpose**: SonarQube combines static and dynamic analysis tools to measure quality continually over time. It helps you maintain clean code by identifying issues and vulnerabilities.
- Features:
  - **Language Support**: SonarQube works with **30+ languages**, frameworks, and Infrastructure as Code (IaC) platforms.
  - **Integration**: Easily onboard projects via integration with platforms like GitHub, GitLab, Azure, and Bitbucket.
  - **Quality Gates**: Define quality requirements; fail pipelines if code quality doesn’t meet them.
  - **Operability**: Run SonarQube your way—on-premises, in the cloud, or using Docker/Kubernetes.
  - **Fast Analysis**: Get actionable metrics in minutes instead of hours.
  - **Security Analysis**: Detect security vulnerabilities (SAST) and secrets in your code.
  - **Shared Configurations**: Align teams on code health and quality goals.
  - **IDE Integration**: Use SonarLint to find code issues directly in your favorite IDE.
- Use Cases:
  - **Continuous Inspection**: Perform automatic reviews with static analysis to detect bugs and code smells.
  - **Security and Compliance**: Achieve robust application security by eliminating vulnerabilities before building and testing.
  - **Secrets Detection**: Prevent secrets from leaking out and becoming security breaches.

DAST TOOLS

Dynamic Application Security Testing (DAST) is a crucial process for identifying security vulnerabilities in web applications during runtime. Unlike Static Application Security Testing (SAST), which analyzes the source code, DAST tools focus on the application as it runs. Here’s how DAST works:

- Runtime Simulation: DAST tools simulate attacks on your application, interacting with the web interface just like an attacker would. They probe for vulnerabilities by sending requests, analyzing responses, and identifying potential weaknesses.
- Black-Box Testing: DAST treats the application as a “black box.” It doesn’t require access to the source code or internal details. Instead, it examines the exposed endpoints, input fields, and APIs.
- Scanning for Vulnerabilities: During testing, DAST tools look for common security issues such as SQL injection, cross-site scripting (XSS), insecure configurations, and more. They provide detailed reports on discovered vulnerabilities.
- Continuous Testing: DAST can be integrated into your CI/CD pipeline, allowing continuous security testing throughout the development lifecycle.

DAST TOOLS

Dynamic Application Security Testing (DAST) limitations:

- Black-Box Approach: DAST treats the application as a “black box,” meaning it lacks visibility into the internal code and logic. As a result, it may miss vulnerabilities that require deeper analysis.
- False Positives/Negatives: DAST tools can produce false positives (flagging non-existent vulnerabilities) or false negatives (missing actual vulnerabilities). Fine-tuning is necessary to reduce these inaccuracies.
- Limited Coverage: DAST focuses on exposed endpoints and user inputs. It may not cover all parts of the application, especially hidden APIs or backend services.
- Stateful Interactions: DAST doesn’t always handle complex stateful interactions well (e.g., multi-step workflows). It may miss issues related to session management or authentication.
- Performance Impact: Running DAST scans can impact application performance, especially during peak usage. Careful scheduling is essential.
- Manual Validation Needed: After DAST identifies vulnerabilities, manual validation is required to confirm their severity and relevance.

DAST TOOLS

Mitigate the limitations of Dynamic Application Security Testing (DAST):

- Hybrid Testing Approach: Combine DAST with other testing methods like Static Application Security Testing (SAST) and manual penetration testing. SAST provides insights into the code, while DAST covers runtime vulnerabilities.
- Customization and Tuning: Fine-tune DAST tools to reduce false positives and negatives. Customize scan settings based on your application’s architecture and business logic.
- Contextual Scanning: Focus DAST scans on critical areas of your application. Prioritize endpoints with sensitive data, authentication, and authorization mechanisms.
- Session Management Testing: Manually validate session management and authentication-related vulnerabilities that DAST might miss. Test session timeouts, token handling, and user roles.
- Stateful Interaction Testing: Create test scenarios that involve multi-step workflows. Ensure DAST handles complex stateful interactions correctly.
- Regular Scanning: Schedule regular DAST scans as part of your CI/CD pipeline. Detect vulnerabilities early and address them promptly.

DAST TOOLS

Some popular Dynamic Application Security Testing (DAST) tools

- Intruder: An excellent vulnerability scanner available in three editions—external scanning, internal scanning, continuous testing, and DAST for web applications.
- SOOS: A cloud-based application testing system suitable for continuous testing in CI/CD pipelines and domain scanning. It's also useful for operations technicians.
- Invicti: A robust DAST system that helps monitor vulnerabilities. It's particularly desirable for businesses needing compliance with HIPAA or PCI DSS.
- Acunetix: Offers automated DAST with a dashboard, ideal for medium to large enterprises.
- Appknox: A cloud-based vulnerability and penetration testing service specifically designed for testing mobile environments.
- Veracode Dynamic Analysis: Integrates well into the DevOps cycle, providing easy-to-use test automation. It's a cloud-based service with strong person-to-person involvement with service engineers.
- Detectify EASM Platform: Supported by ethical hackers, this tool allows small business owners to run their own DAST exercises from the cloud.
- Rapid7 InsightAppSec: A cloud-based DAST solution provided by a highly experienced cybersecurity consultancy.
- Checkmarx: A cloud-based application testing platform that offers DAST.

SCA TOOLS

Software Composition Analysis (SCA) tools play a crucial role in identifying and managing open-source components within an application’s codebase

- Purpose of SCA Tools: SCA tools analyze applications and related artifacts (such as containers and registries) to detect open-source and third-party software components. They identify known vulnerabilities, outdated components, and licensing risks associated with these packages.
- Benefits of SCA Tools:
  - Security: SCA helps ensure your software supply chain includes only secure components, supporting safe application development.
  - Compliance: It assists in managing licensing obligations and attributions.
  - Risk Mitigation: By identifying vulnerabilities, SCA tools allow proactive fixes before exploitation.
- Popular SCA Tools:
  - Mend.io: Offers effective vulnerability scanning and recommended solutions for healthcare applications.
  - Veracode: Integrates well, provides detailed vulnerability analysis, and offers prompt customer support.
  - Black Duck (by Synopsys): Provides valuable information on licensing and security risks.
  - Timesys Vigiles: Ideal for managing vulnerabilities in embedded Linux devices

COMPLIANCE AS CODE

Compliance as Code is the practice of defining and enforcing compliance policies through automated, programmable configurations integrated into the software development lifecycle. Instead of manual checks or paper-based processes, compliance requirements—like security standards, regulatory rules (e.g., GDPR, HIPAA), or internal governance—are written as code, typically in formats like JSON, YAML, or domain-specific languages

COMPLIANCE AS CODE

Benefits:

- **Speed**: Automates slow, error-prone manual audits.
- **Consistency**: Applies uniform rules across environments.
- **Scalability**: Handles large, complex infrastructures.
- **Auditability**: Provides clear evidence for regulators or internal audits.

Challenges:

- Translating complex regulations into code requires expertise.
- Initial setup can be time-intensive.
- Tools vary in language and scope, requiring careful selection

COMPLIANCE AS CODE

Open Policy Agent (OPA): Used to enforce Kubernetes policies. an open-source, general-purpose policy engine for enforcing fine-grained, declarative policies across software systems. It decouples policy logic from application code, allowing centralized, consistent compliance and authorization decisions. OPA is widely used in cloud-native environments, particularly with Kubernetes, but also supports APIs, CI/CD pipelines, and infrastructure management.

COMPLIANCE AS CODE

Key Features:

- **Declarative Policies**: Write what the policy should achieve, not how to enforce it.
- **Context-Aware**: Evaluates policies based on input data and external context (e.g., user roles, resource metadata).
- **Extensibility**: Supports custom data sources (e.g., REST APIs, databases) for dynamic policies.
- **Testing**: Built-in tools for unit testing policies with opa test.