

# **DevSecOps and Cloud Security**

**"Deliver resilient software capability at the speed of relevance"**



--DevSecOps and Cloud security automation--

# Application Security Analyst and DevSecOps expert

- Web and mobile Developer
- +20 Application Security Evaluation
- SANS 540 DevSecOps and Cloud Security automation
- CTF Player/ Organizer
- I love microservices
- Cloud Native Application security advocate

# What you will learn

All what you need to secure your CI/CD pipeline !

# Agenda

## Day 1:

# Attacking and Hardening the DevOps Toolchain

# **Day 1 :**

# **Attacking and Hardening the DevOps Toolchain**

# Lab environment overview

Role	Systems(s)
Version Control	GitLab, CodeCommit
CI/CD	Jenkins, CodePipeline
Configuration manager	Ansible, CloudFormation
Container Execution	Docker, K8s
Secret Storage	Vault
Cloud Infrastructure	AWS

# Lab setup summary

- Port conflict (GITLAB/JENKINS)
- HTTPS issue in GITLAB/JENKINS integration
- Authorizing local IP addresses in GitLab for WebHooks

# Day 1 :Attacking and Hardening the DevOps Toolchain

- DevOps security challenges
- DevOps toolchain



# DevOps success factors

CAMS (or CALMS) is a common lens for understanding DevOps and for driving DevOps change. Your organization succeeds when it reaches "CALMS":

- Culture: people come first
- Automation: rely on tools for efficiency and repeatability
- Lean: apply Lean engineering practices to continuously learn and improve
- Measurement: use data to drive decisions and improvements
- Sharing: share ideas, information, and goals across silos

# DevOps Unicorns

The DNA of DevOps and DevSecOps comes from a few early leaders or "unicorns":

- Netflix: Undifferentiated heavy lifting in the cloud, Chaos Engineering
- Amazon: You Build It... You Run It, two-pizza teams dogfooding AWS
- Google: Site Reliability Engineering, fearless shared postmortems
- Twitter: Self-service security for developers, immediate feedback
- Etsy: Security in Continuous Deployment, "a Just Culture"

## CALMS: Culture

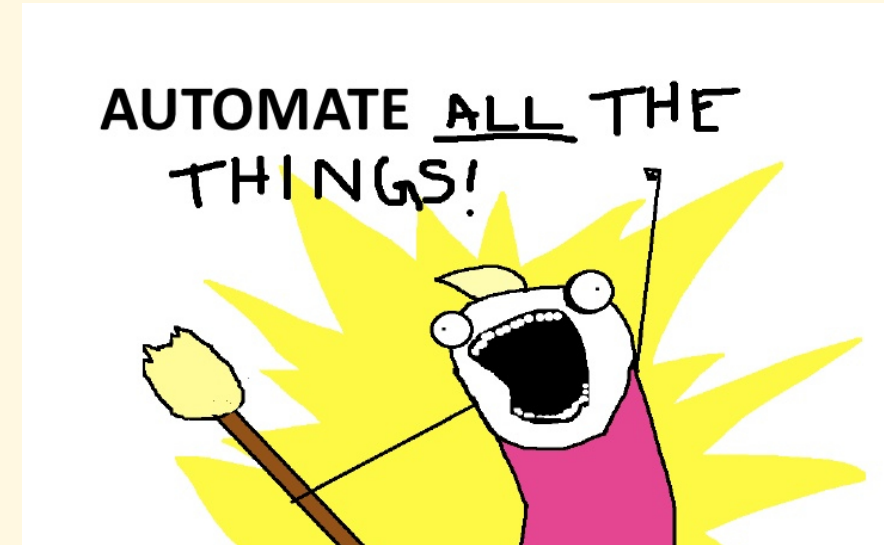
Signs of a culture that supports DevOps:

- Failures are accepted and used as learning opportunities.
  - Accept that failures can and will happen.
  - Utilize blameless postmortems to learn and improve from failures.
- Find ways to minimize impact or "blast radius."
- --Collaboration-- occurs widely across functional lines.
- Problem-solving: the problem is the enemy.
- Empower teams to take action on their own.
- Staff have a shared sense of ownership and accountability.
- --High levels of trust-- allow people to feel safe with each other.

## CALMS: Automation

DevOps teams use programmable tools and cloud service APIs to:

- Take care of "undifferentiated heavy lifting" and repetitive work
- Continuously iterate and experiment in production
- Increase transparency
- Ensure consistency
- Prevent common mistakes
- Enable measurement and sharing
- Control operations at scale



## **CALMS: Automation "Everything as Code"**

Get everything into code and check it into version control:

- Application source code/package dependencies
- Operations scripts
- Application configuration
- Tests and test scripts
- Build, deployment scripts, and runbooks
- Infrastructure provisioning and configuration
- Documentation
- Database definitions and change scripts
- Security/compliance policies
- Security tool configuration

## **CALMS: Lean engineering**

Lean engineering improves efficiency, reduces friction, eliminates hand-offs and delays:

- Confront bureaucracy.
- Use Value Stream Mapping to identify waste, bottlenecks, and delays.
- Leverage Automation for routine work.
- Measurement: Use data to drive continuous improvements.
- Prioritize just in time-always work on what's important.

## **CALMS: Measurements**

Measurements can help you to understand-and improve-how your teams work and where to automate:

1. Change frequency
2. Change failure/success rate
3. Correlation between change frequency and failure rate
4. MTTR recovery window instead of MTTF (time to failure)
5. Change lead time to deliver a change or fix, from check-in to deployment

## CALMS: Measurements for DevSecOps

Collect security metrics along your delivery pipelines to assess how healthy your code is and how healthy your security program is.

- Measure automated test coverage for high-risk code
- Track # of vulnerabilities found... and where they were found in the pipeline
- Track # of vulnerabilities fixed
- How long vulnerabilities remain open (window of exposure)
- Type of vulnerability (OWASP Top 10) for Root Cause Analysis
- Elapsed time for security testing-make feedback loops as short as possible
- False positives versus true positives-improve quality of feedback
- Vulnerability escape rate to production

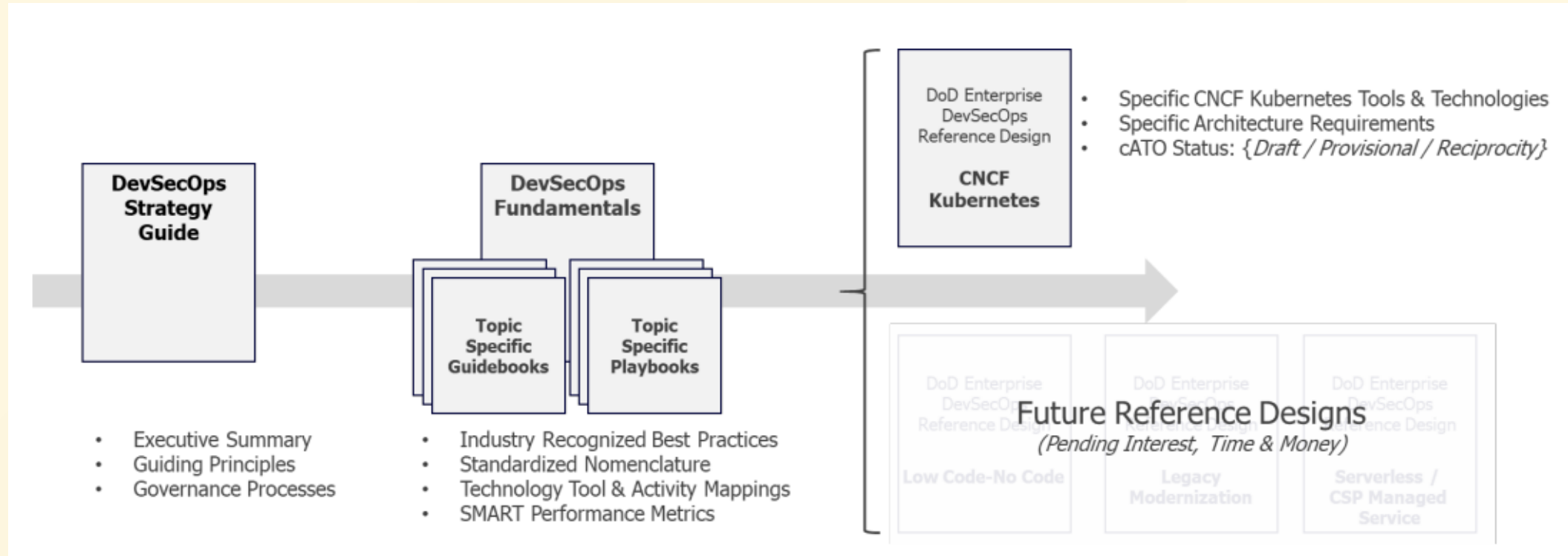


## CALMS: Sharing

Information should be shared freely across the organization.

- Everyone has access to all information that they need to do their job.
- Eliminate hand-offs where information/knowledge is often lost.  
*Open up communications channels across teams/functional lines.*
- Chat and ChatOps, shared code repos, backlog tools, dashboards, etc.  
*Share goals and accountability across Dev, Ops, and Security.*
- Everyone has a stake in success and preventing failures.  
*Evangelism through conferences, blogs, articles, podcasts, etc.*
- DevOps has an active, open community (DevOpsDays, open-source tools...).
- Vendors have latched on to this hyperactivity.

# CALMS: US DOD case study



# Security culture vs DevOps

DevOps culture conflicts with traditional security culture:

- Top-down risk management instead of team-based decision-making
- Need to know restrictions versus extended information sharing
- Zero failure versus fail fast and fail forward
- Limiting change: Security is always ready to say "No!"

# Security Challenges in DevOps

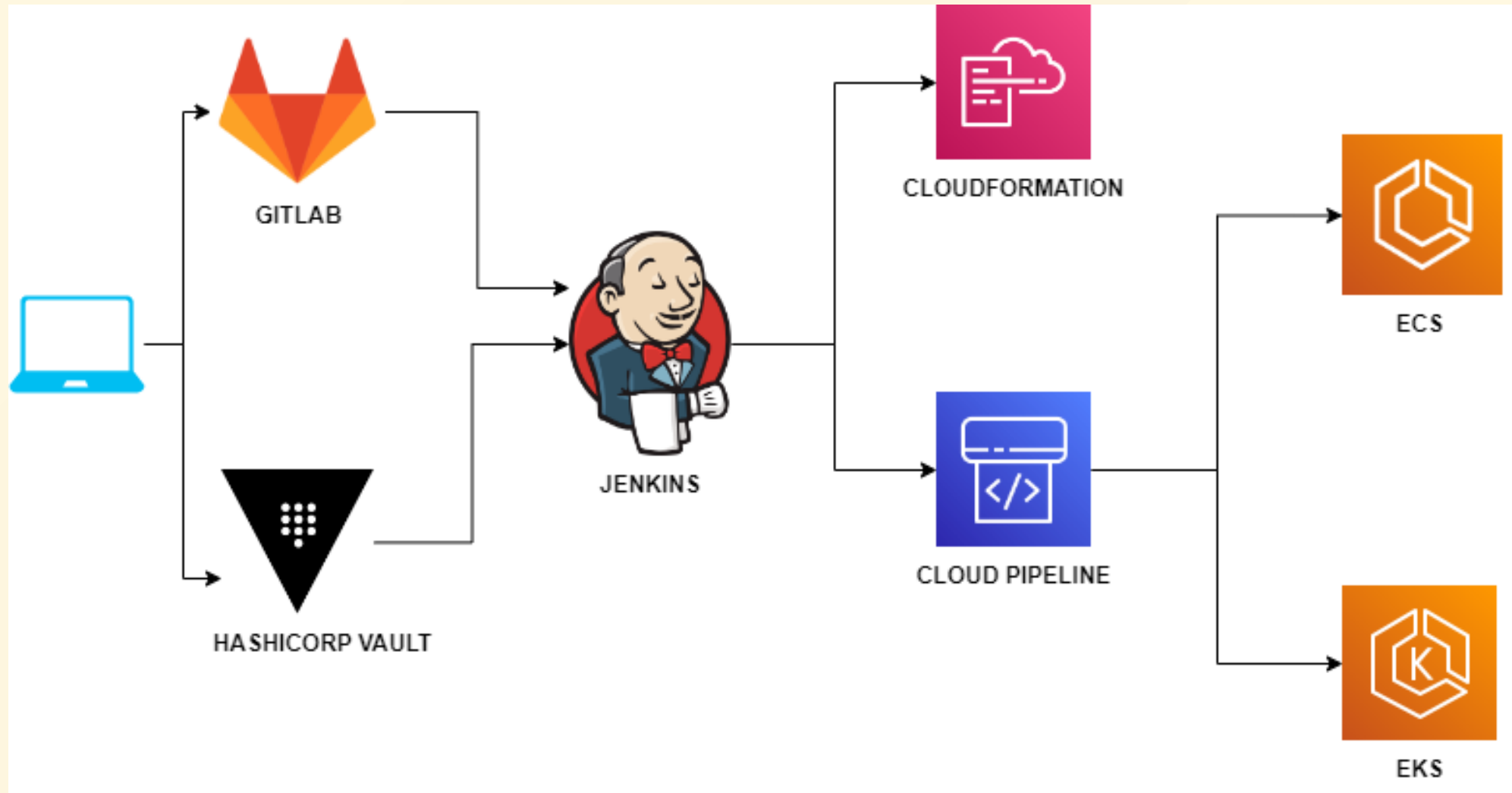
Powerful new technologies create opportunities for attackers-and new risks for organizations:

- Weaknesses in the DevOps toolchain can compromise the entire stack.
- Cloud platform misconfigurations can easily allow unauthorized access to data.
- Containers and orchestrators introduce a new attack surface, often not supported by traditional security scanners.
- Microservice-based architectures, new languages, and frameworks compound security guidelines.
- Delivery at the speed of DevOps requires enhanced detection and automated remediation.

# **Day 1 :Attacking and Hardening the DevOps Toolchain**

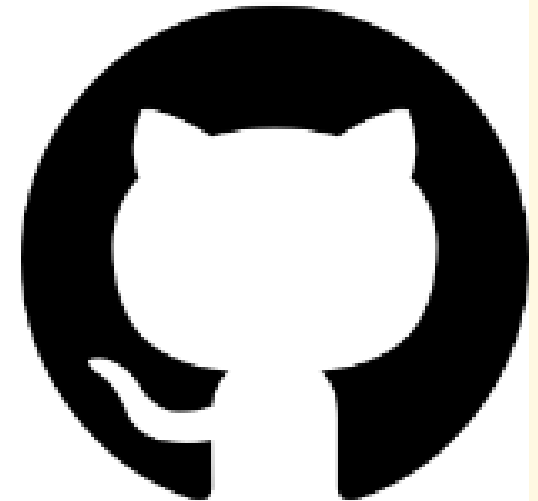
## **DevOps toolchain**

# DevOps toolchain



# Version control

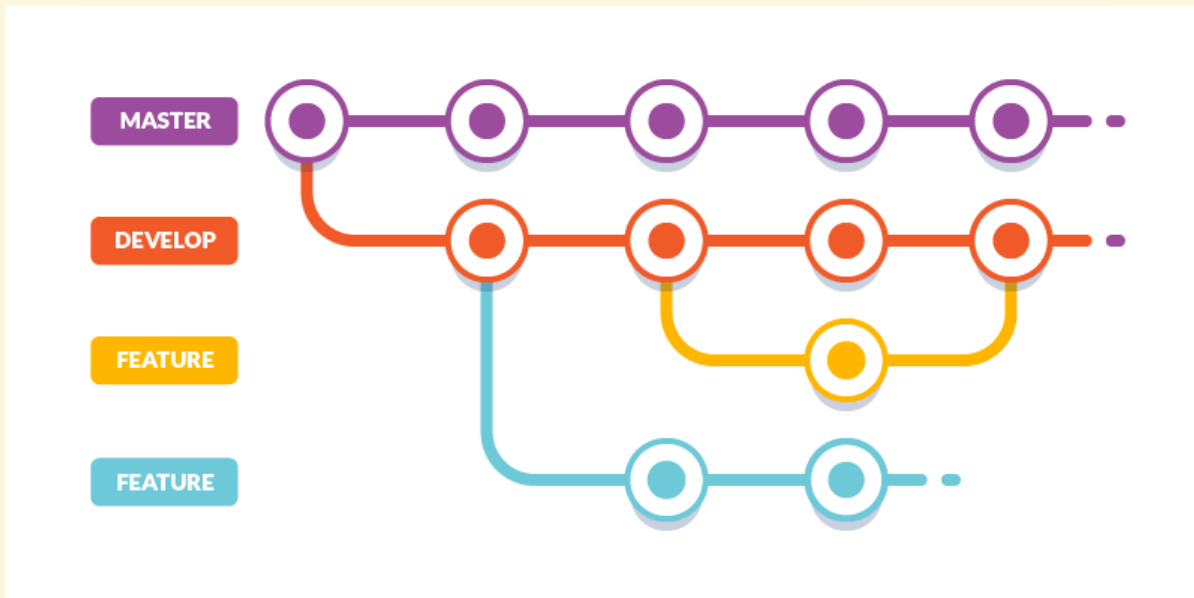
- The DevOps toolchain starts with pushing code to a version control system.
- The most common version control systems include:



# GitFlow

Understanding GitFlow is important for security teams to:

- Contribute to development and operations workflows
- Prevent attackers from compromising the workflow





# Continuous Integration

Approving and completing a merge request creates a commit on the "main" branch and automatically triggers a pipeline:

- Ensure that changes integrate successfully with the rest of the codebase
- Executes automated unit tests and other automated checks
  - Red: no other check-ins are allowed until the build is fixed
  - Green: creates build artifacts for next steps in CD pipeline
- Fast feedback - build and test steps need to run in a few minutes to encourage iterative development (small, frequent changes)

# Continuous Delivery

Pipeline model and control framework extending Continuous Integration:

- It uses the latest good build from CI, packages for deployment, and release.
- Changes are automatically pushed to test/staging environments to conduct more realistic/comprehensive tests.
- It can insert manual reviews/testing/approvals between pipeline stages.
- Log steps and results to provide audit trail from check-in to deploy.
- Any failures will "stop the line."
- No additional changes can be accepted until the failure is corrected.
- This ensures that code is always ready to be deployed.
- Changes may be batched up before production release.

# Continuous Delivery vs Continuous Deployment

## CONTINUOUS DELIVERY



## CONTINUOUS DEPLOYMENT



# Continuous Deployment

Continuous Deployment is how organizations like Netflix, Etsy, and Amazon push out changes n times per day/hour/minute:

- Changes are deployed directly and automatically to production using the CD pipeline once all tests/checks pass.
- Self-service - Changes are pushed to production by developers.
- Blue/Green Deployment - Deploy changes and switch between production environments using load balancing
- Canaries - Incremental deployment that stops and rolls back if errors occur
- A/B testing - Measuring the effect/acceptance of a change or new feature in production
- Dark Launching - Protect changes behind "feature switches"

# CI/CD Systems

Version control push events on the develop/main branches trigger workflow pipelines for building, testing, and deploying the changes.



Actions



# Jenkins pipeline as code declarative syntax

Jenkins pipeline building a docker image

```
pipeline {  
  agent { dockerfile true }  
  stages {  
    stage('Test') {  
      steps {  
        sh 'node --version'  
        sh 'svn --version'  
      }  
    }  
  }  
}
```

# Jenkins credential manager

**Global credentials (unrestricted)** [+ Add Credentials](#)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
e8d2b605-12fe-4a63		Username with password	
03f7a950-6000-4...		SSH Username with private key	
135a3390-e563-4...		SSH Username with private key	
FOO	test	Secret text	test

Icon: S M L

REST API Jenkins 2.375.2

```
AWS_ACCESS_ID = ${env.AWS_ACCESS_ID}
AWS_SECRET = ${env.AWS_SECRET}
```

# **CI/CD Attacks**

**SOLARWINDS**

**CODECOV**



# CI/CD Security Risks

## Top 10 CI/CD Security Risks

- CICD-SEC-1 **Insufficient Flow Control Mechanisms**
- CICD-SEC-2 **Inadequate Identity and Access Management**
- CICD-SEC-3 **Dependency Chain Abuse**
- CICD-SEC-4 **Poisoned Pipeline Execution (PPE)**
- CICD-SEC-5 **Insufficient PBAC (Pipeline-Based Access Controls)**
- CICD-SEC-6 **Insufficient Credential Hygiene**
- CICD-SEC-7 **Insecure System Configuration**
- CICD-SEC-8 **Ungoverned Usage of 3rd Party Services**
- CICD-SEC-9 **Improper Artifact Integrity Validation**
- CICD-SEC-10 **Insufficient Logging and Visibility**



## **DevOps toolchain summary:**

Security teams must understand how the CI/CD pipeline works, the tools involved, and how the teams use them before they start looking how and where to add security checks

# **Day 1 :Attacking and Hardening the DevOps Toolchain**

## **Securing DevOps toolchain**

## DevOps / DevSecOps

Incremental, iterative development and rapid delivery in DevOps can significantly reduce this window of exposure:

- DevOps teams follow "just-in-time prioritization"
- Automated testing cheap and safe to push out changes quickly.
- DevOps teams follow Lean techniques.
- Automated security checks in the CI/CD pipeline will catch many vulnerabilities.
- Security fixes are "standard changes".
- DevOps encourages developers and ops and security specialists to work together.

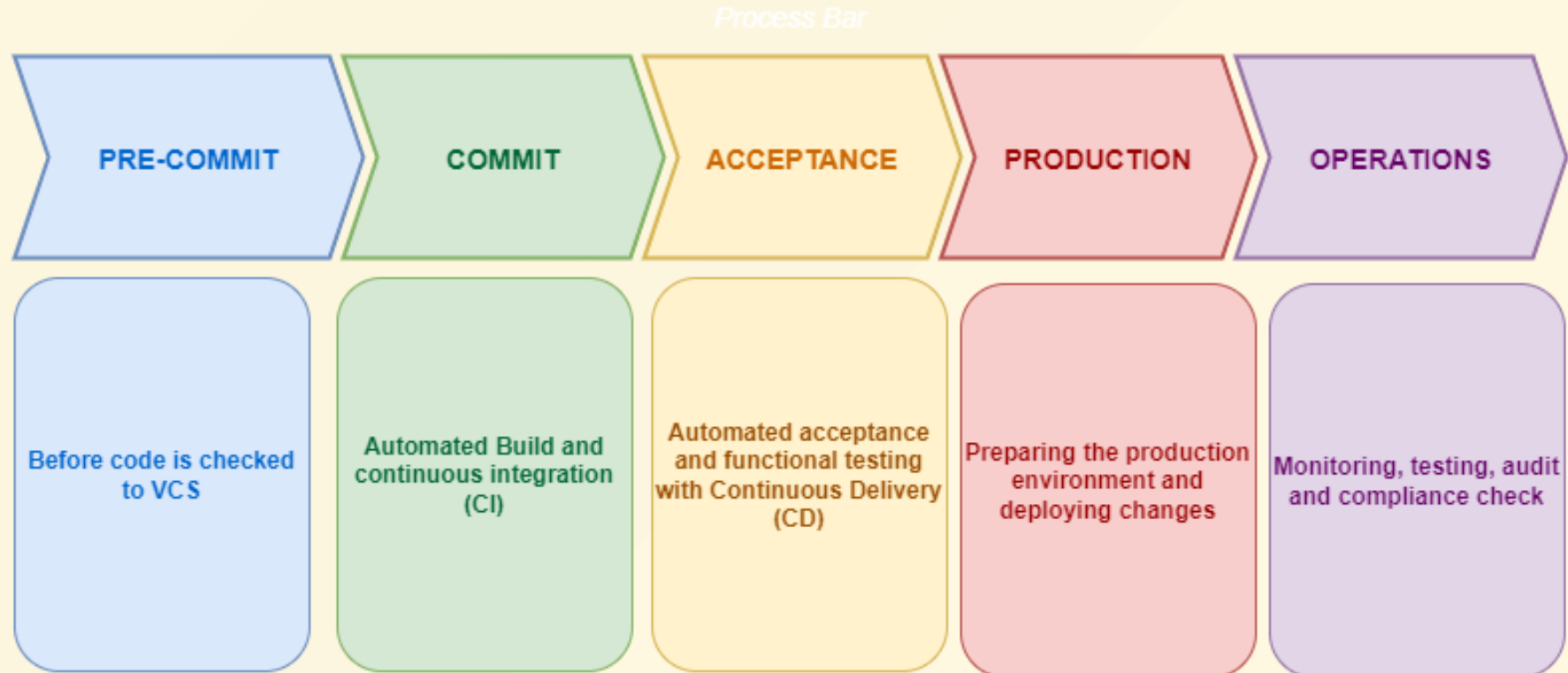
## Security Champions: Breaking Down DevOps Silos

- Security experts embedded into the development organization
- Break down barriers between the central Security and DevOps teams
- Requires funding and support from upper management
- Provides several ancillary benefits:
  - The reputation of the company
  - Teaches valuable leadership skills
  - Increased employee retention

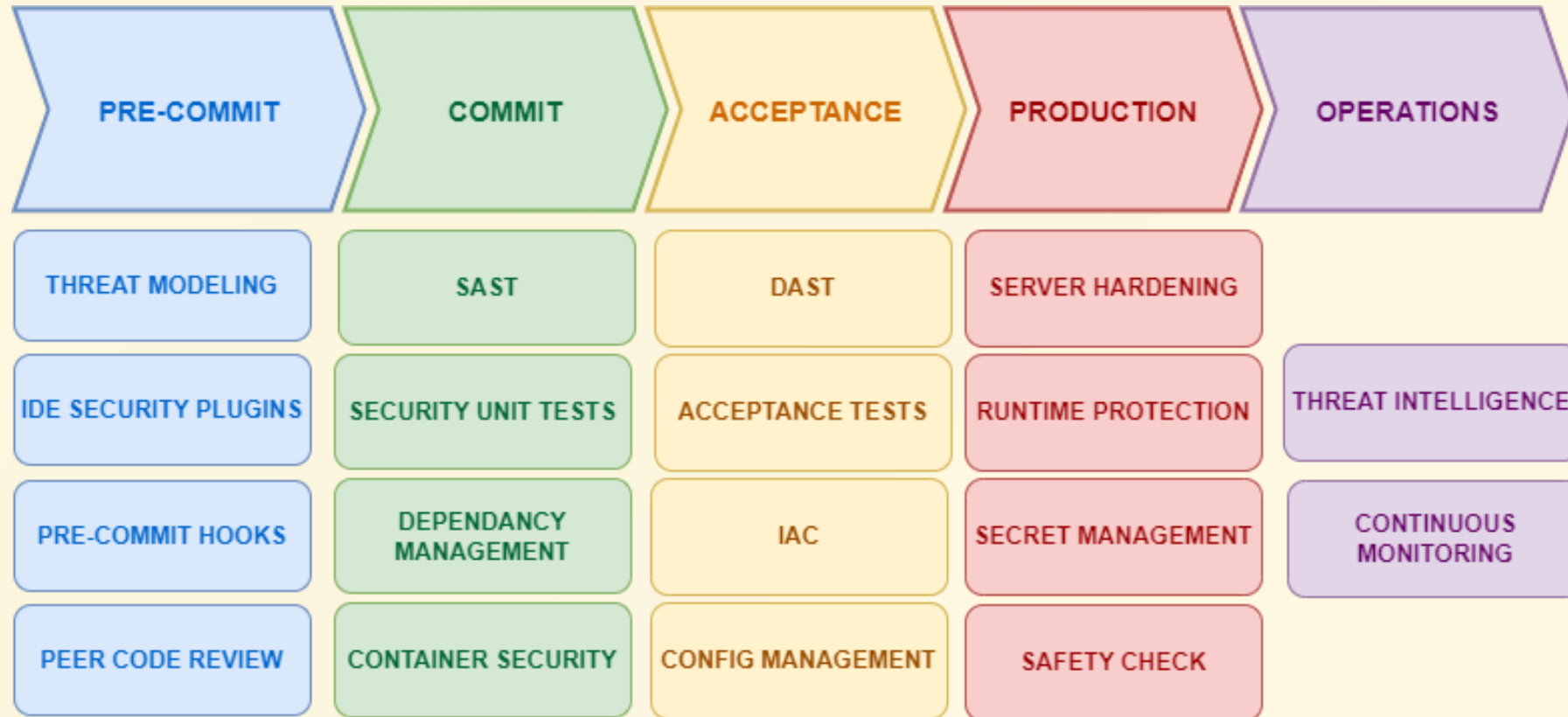
# DevOps workflow

The DevOps workflow is based on five key phases:

- Manual work done before merging code into a delivery branch
- Automated Continuous Integration (CI) & Continuous Delivery (CD) process



# DevOps workflow



# DevOps workflow





# RRA

- **Quick!** The RRA takes 30 to 60 minutes maximum.
- **Very high-level.** Details are for complete threat models. The RRA can become a complete threat model over time though!
- **Concise, readable.** Short and with clear risk levels.
- **Easy to update.** Can be run during any phase of the project development and continuously updated.
- **Informative.** Collects risk impact and a data dictionary. Also, collections information about how the service functions.
- **Let you know what to do.** The RRA includes the list of recommendations from the security team with a priority for each item.

# Threat Modeling

Ask a few basic questions:

1. Are you changing the attack surface of the system in a material way? Are you adding a new API or opening a new port, adding a new user type/role, adding or changing a data store/sink, calling out to a new service or system?
2. Are you changing the technology stack? Are you adding or significantly upgrading a framework or important library or language or runtime component?
3. Are you changing security controls: authentication or session management code, cryptologic, access control code or rules, auditing logic...?
4. Are you adding confidential/sensitive data or changing data classifications?
5. Are you modifying high-risk code?

# **Store Markdown RRA and Threat Modeling in the repository /docs directory**

# Code Inventory

```
$ scc path/to/your/code
```

Language	Files	Lines	Code	Comments	Blanks
Go	112	50145	38876	5449	5838
Markdown	8	2228	1710	163	355
Shell	11	799	598	97	104
YAML	6	515	408	10	97
Total	137	53687	41670	5719	6394

# Code Analysis: IDE Security Extensions

The VS Code Marketplace contains thousands of security-focused plugins:

- Semgrep
- Checkov/CFN Nag
- SonarLint/ ESLint
- Hadolint
- InSpec

# Version Control Security

Version control systems and git provide workflow capabilities for locking down the remainder of the pre-commit security controls:

- Git Hooks
- Mandatory peer reviews
- Mandatory security reviews
- High risk code approvals

# Git Commit Hooks

Run script automatically, checking for embedded secrets, code correctness at different point in the workflow :

- Local repository : pre-commit, commit, post-commit ...
- Remote repository: pre-receive, post-receive
- Implement team-wide workflow policies before CI

# Manual Code Reviews

Take advantage of code review workflows for security reviews

- Security team sets policies and train developers on how to do security code reviews. Create a checklist for them.
- Look for problems that static analysis tools don't find.
- Tag High-risk code



# Branch Protection

All version control providers provides a branch protection capability:

- Require merge requests to commit to the branch
- It defines the role that can complete a merge request.
- It prevents users to directly push to the main branch
- it enables the CodeOwners approval workflow

# Code Owners

Inventory high risk code and identify groups and individuals responsible for approvals :

- Create CODEOWNERS file in the repository root directory
- Define directories or individual files in the CODEOWNERS file
- Require one or many approvers during the pull request to review changes
- The security team can participate in the GitFlow workflow

# Code Owners

```
# This is a comment.
# Each line is a file pattern followed by one or more owners.
# Specify the owners for the entire repository
* @all-developers
# Specify the owners for security-related files
/security/ @security
# Specify the owners for user experience-related files
/frontend/ @user-experience
# Example for specific files
README.md @all-developers
# Example for a specific directory
docs/ @all-developers
# Example for specific file types
*.js @all-developers
CODEOWNERS @security @admin
```

## Detecting High Risk Code

Using unit tests to check if a high-risk code file changed.


```
describe('HighRiskCode Checksum Test', () => {
  const testCases = [
    { file: 'path/to/Jenkinsfile', checksum: '2bf33b66ddb07616f882ceed0718826af298a7' },
    { file: 'path/to/Dockerfile', checksum: 'fe83bf6f453698c5f78cab167bca14c72daf32c0' },
  ];
  testCases.forEach(({ file, checksum }) => {
    it(`should verify the checksum of ${file}`, () => {
      const alertSpy = sinon.spy(requestCodeReview);
      const actualChecksum = getChecksum(file);
      const match = checksum === actualChecksum;
      if (!match) {
        requestCodeReview(file);
      }
    });
  });
});
```


# Security Checklist: Pull Request Templates


Markdown based templates for starting a pull request can include checklists, including security tasks

## New Pull Request

Select the branch to merge into and the branch to pull from.

 merge into: devsecops-training-lab:main

 pull from: devsecops-training-lab:testing-pull-request

 Update Jenkinsfile

Start the title with **WIP:** to prevent the pull request from being merged accidentally.

Write

Preview

### Description

### Checklist

#### General

- ☐ I have performed a self-review of my own code.
- ☐ I have commented my code, particularly in hard-to-understand areas.
- ☐ I have made corresponding changes to the documentation.
- ☐ My changes generate no new warnings.

#### Testing

- ☐ New and existing unit tests pass locally with my changes.
- ☐ I have checked my code and corrected any misspellings.

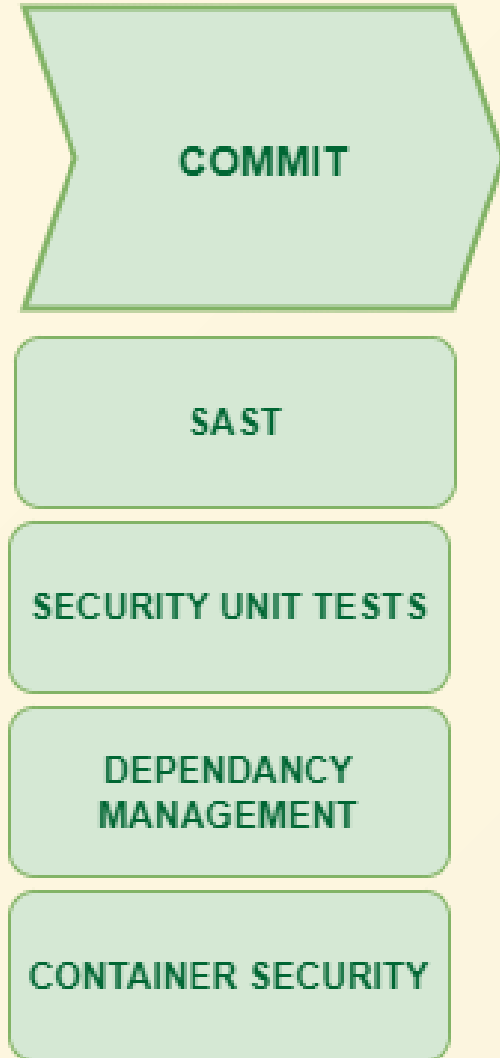
#### Security

- ☐ No sensitive information (like credentials) are included in this PR.

## Summary

# Secure your Version Control System

# DevOps workflow - Security Controls



## Static Analysis :

Provide fast and clear feedback on code commit:

- Run code scans in parallel with unit testing for speed.
- Run high-risk checks early in the pipeline to "fail fast."
- Do incremental scanning, if possible, as deep scanning takes too long for CI/CD, especially on large codebases,
- Set a max time limit with the dev team, alert them if the limit is breached and fix it.
- Return results directly to developers using their IDE or backlog list/tool.
- Run full/long-running scans out-of-band (nightly) or in the scanning



## Minimize False Positives

Developers will tune out results if you don't tune out false positives.

- Carefully review rules to identify which checkers provide high-confidence results.
- Fail the pipeline if these checks fail.
- Turn off noisy, low-confidence checks in the pipeline-run them separately and review/qualify them manually. Feed real positive findings back to the development team's backlog.
- Periodically review and re-tune rules and configurations.  
Track and document tool configurations and decisions on which checkers or rules you disable (for compliance/governance).

# SAST : Configuration management

IaC Technology	Static Code Analysis Tools
Ansible	KICS
CloudFormation	cfn_nag, Cfripper, Checkov, KICS
Terraform	Terrascan, Checkov, TFSec, KICS, ShiftLeft

## SAST : Language Support

Type	Java	Node.js	Python	PHP
Code Quality	SonarQube, PMD	ESLint	Pylint, SonarQube	SonarQube, PHPStan, Psalm, Phan
Security Checker	Fortify, Checkmarx,	npm-audit, NSP	Bandit	RIPS
Bug Corrector	FindBugs	Flow	Pyflakes	Phan

# SAST : Semgrep

Semgrep provides a lightweight, multi-language, extensible static analysis solution.

- Open-source
- Community driven rules
- Language support : Go, Java, JavaScript, Python ...
- Support automation !!!!!!!

# SAST : Semgrep

```
semgrep --config "p/expressjs" --config custom_rules.yml /path/to/your/express/app  
  
/path/to/your/express/app/index.js  
  5:14  error  jwt-hardcoded-secret  Avoid hardcoding JWT secrets  
 10:5   error  security-audit-rule   Some other security issue  
  
2 issues
```

# Semgrep : Custom rules

Custom rules can be added using the Semgrep playground to test the rule syntax :

```
1 rules:
2 - id: public-s3-policy-statement
3   pattern: |
4     {
5       "Effect": "Allow",
6       "Principal": "*",
7       "Resource": [
8         ..., "=~/arn:aws:s3:*/
9         ", ...
10      ],
11    }
12   message: Detected public S3
13   severity: WARNING
14   languages:
15   - json
```

```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Sid": "PublicReadGetObject",
6       "Effect": "Allow",
7       "Principal": "*",
8       "Action": [
9         "s3:GetObject"
10      ],
11       "Resource": [
12         "arn:aws:s3:::example.com/*"
13      ]
14     },
15     {
16       "Sid": "CloudFront Origin Access Identity",
17       "Effect": "Allow",
18       "Principal": {
19         "AWS": "arn:aws:iam::cloudfront:user/CloudFrontOriginAccessIdentityECLCLOUDFRONTOAI"
```

# Dependencies : Component Analysis

Dependency Checker	Supported Languages
OWASP Dependency-Check	Java, JavaScript, .NET, Python, Ruby, PHP, Node.js
retire.js	JavaScript, Node.js
safety	Python
PHP Dependency Checker	PHP

# Machine Readable report

Running security tools in CI/CD requires a supported machine-readable output format.

- xUnit/JUnit
  - Standard XML schema for reporting pass/fail unit test results
- Checkstyle
  - Standard XML schema for reporting static analysis results
- SARIF (Static Analysis Results Interchange Format)
  - JSON based schema primarily used for displaying results in GitHub
- JSON
  - Custom schemas are machine-readable, but you have work to do!