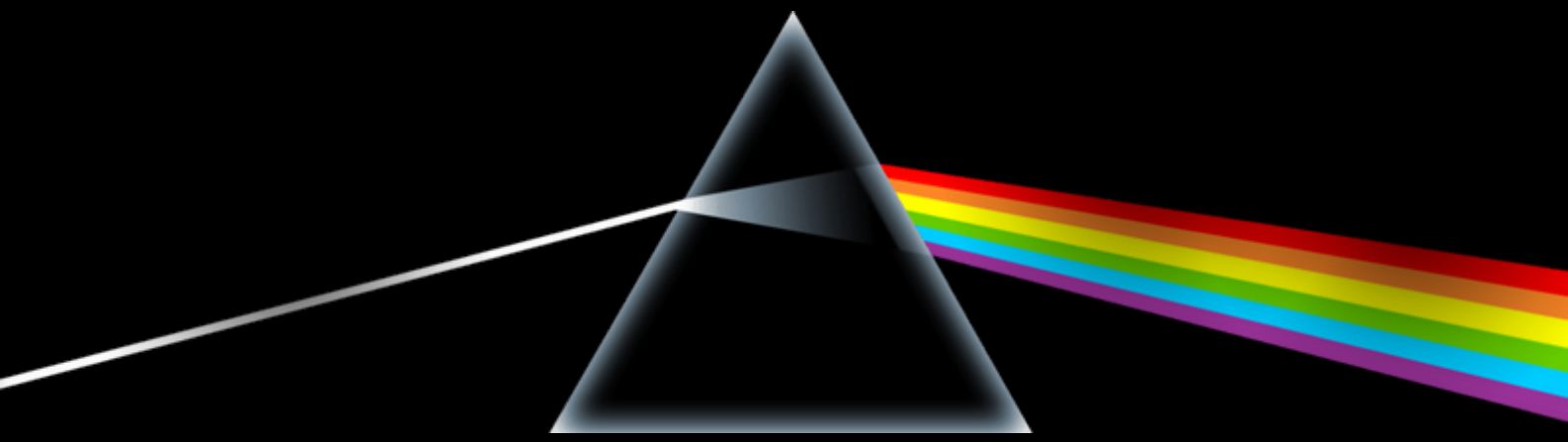
2023

DEVSECOPS MATURITY MODEL

A roadmap for organizations to evolve their practices.



WWW.DEVSECOPSGUIDES.COM

INTRODUCTION

The Evolving World of Software Development In the dynamic realm of software development, integrating security into the development process has become a paramount concern. This introductory section sets the stage for the journey through the DevSecOps Maturity Model, a framework that blends development, security, and operations into a cohesive and efficient workflow. The paragraph emphasizes the changing landscape of software development and the rising need for security to be a foundational element rather than an afterthought.

Understanding DevSecOps DevSecOps represents a cultural and technical shift in how organizations approach software development and security. This section defines DevSecOps, detailing how it extends beyond traditional development practices by embedding security at every stage of the software development life cycle. It underscores the philosophy of 'security as code' and its significance in modern software development.

The Need for a Maturity Model As organizations adopt DevSecOps practices, the need for a structured approach to gauge and enhance their progress becomes clear. This paragraph introduces the concept of a maturity model in the context of DevSecOps, explaining its role in helping organizations systematically improve their integration of security into development processes.

Overview of the DevSecOps Maturity Model The DevSecOps Maturity Model provides a roadmap for organizations to evolve their practices. This section offers an overview of the model, outlining its various stages from initial implementation to advanced stages where DevSecOps principles are deeply ingrained in the organizational culture and workflows.

Key Components of the Maturity Model Delving into the specifics, this part enumerates the key components of the DevSecOps Maturity Model, such as automation, policy as code, security integration, monitoring, and feedback loops. It briefly describes each component's role in enhancing the security and efficiency of the development process.

Challenges and Strategies Implementing DevSecOps is not without its challenges. This section discusses common obstacles organizations face, such as cultural resistance, tool integration issues, and skill gaps. It also suggests strategies to overcome these challenges, emphasizing the importance of leadership, training, and clear communication.

Case Studies and Real-World Examples To illustrate the practical application of the DevSecOps Maturity Model, this paragraph introduces a series of case studies and real-world examples. These narratives showcase how different organizations have successfully navigated their DevSecOps journey, providing insights and lessons learned.

Navigating Forward Concluding the introduction, this section outlines the book's structure, guiding the reader through the detailed exploration of each stage of the DevSecOps Maturity Model. It sets expectations for the comprehensive coverage of best practices, tools, and methodologies that will be discussed in subsequent chapters.

HADESS

To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.

At Hadess, our mission is twofold: to unleash the power of white hat hacking in punishing black hat hackers and to fortify the digital defenses of our clients. We are committed to employing our elite team of expert cybersecurity professionals to identify, neutralize, and bring to justice those who seek to exploit vulnerabilities. Simultaneously, we provide comprehensive solutions and services to protect our client's digital assets, ensuring their resilience against cyber attacks. With an unwavering focus on integrity, innovation, and client satisfaction, we strive to be the guardian of trust and security in the digital realm.

ACKNOWLEDGMENT

- Dan Williams
- Tanya Janca
- Aristide Bouix
- Okan Yıldız
- Carol Valencia
- Ahmed AlaaEldin
- Alessio dl
- Ayoub Najim
- Barnavo Chowdhury
- Carol Valencia
- Charles Chibueze
- David das Neves
- Devyani Vij
- Eslam Samy Hosney
- Ibrahim Akkulak
- Josh
- Mahesh M
- Mohammad Hadi Arghani
- Muthukumar Selvarasu
- Nditah Samweld
- Ahmed Saafan
- SANKALP SANDEEP PARANJPE
- Williams A. Danso
- David das Neves

DEVSECOPS LAYERS IN SECURITY CONTROLS, TOOLS, AND PRACTICES THROUGHOUT THE DEVOPS LIFECYCLE

DevSecOps represents the integration of security practices within the DevOps framework. It emphasizes the need for a collaborative approach between development, operations, and security teams. This integration is crucial for accelerating delivery, enhancing software quality, and ensuring reliable services in production. The DevSecOps Maturity Model outlines four stages of maturity across six major competency areas.

The Stages of DevSecOps Maturity

Beginner

- **Focus**: Cultural shift emphasizing collaboration across technical disciplines.
- **Characteristics**:
  - Beginning of the DevSecOps journey.
  - Emphasis on team performance improvement.
  - Initial integration of security practices.

Intermediate

- **Focus**: Consistent software release with an evolving security integration.
- **Characteristics**:
  - Regular software releases with some bottlenecks and performance issues.
  - Security controls start shifting earlier in the development process.
  - Some friction among teams, with security work still concentrated towards the end of the process.

Advanced

- **Focus**: Efficient and productive release of high-quality, secure software.
- **Characteristics**:
  - High efficiency in software development and release.
  - Security checkpoints embedded throughout the software development lifecycle (SDLC).
  - Regular release of secure and reliable software.

Expert

- **Focus**: Cutting-edge practices with deep integration of security across the SDLC.
- **Characteristics**:
  - Frequent release of high-quality code, potentially multiple times per day.
  - Deeply embedded security controls throughout the SDLC.
  - High level of automation across Development, Operations, and Security.
  - Security is no longer a siloed domain but an integral part of the entire process.

Key Aspects of DevSecOps Integration

- **Collaboration and Sharing**: Breaking down silos between security and DevOps teams.
- **Shift-Left Security**: Integrating security early in the development process.
- **Continuous Feedback and Improvement**: Leveraging feedback loops for continuous enhancement of security practices.
- **Automation**: Automating security tasks to increase efficiency and reduce human error.
- **Regular and Reliable Releases**: Ensuring that each release is secure and stable.
- **Holistic Security Approach**: Considering security at every stage of the SDLC.

DevSecOps is not just a set of practices but a cultural shift that requires commitment from all stakeholders involved in the software development and deployment process. The journey from Beginner to Expert stage in DevSecOps maturity involves a gradual and consistent integration of security practices, tools, and mindsets into the existing DevOps framework.

Competency

Competency: People & Culture

Level	Description
Beginner	- Functional teams siloed - High inter-team friction - Nascent onboarding processes - Burnout common
Intermediate	- Silos breaking down - Embracing experimentation & transparency - Onboarding process exists - Burnout openly discussed
Advanced	- Continuous collaboration across teams - Blameless culture - Comprehensive onboarding process - Burnout quickly addressed
Expert	- Cross-functional teams aligned to products and services - High trust, experimentation, learning culture - Burnout rare

Competency: Plan & Develop

Level	Description
Beginner	- Risk and security not considered - High technical debt - Excessive bug fix work - Code not validated
Intermediate	- Limited risk assessment - Moderate technical debt - Moderate bug fix work - Some code validation
Advanced	- Threat modeling and risk assessments - Low technical debt - Low bug fix work - All code validated
Expert	- Extensive threat modeling/risk assessment - Minimal technical debt - New feature focus - All code validated automatically

Competency: Build & Test

Level	Description
Beginner	- Manual testing - No code scanning - No build/signature validation - Limited core functionality testing
Intermediate	- Partial test automation - Partial code scanning - Partial build/signature validation - Partial core functionality testing
Advanced	- High test automation - Dynamic code scanning - Significant build/signature validation - Significant core functionality testing
Expert	- Complete test automation - Comprehensive dynamic code scanning - Comprehensive build/signature validation - Comprehensive core functionality testing

Competency: Release & Deploy

Level	Description
Beginner	- Manual deployments - Large, infrequent releases - No deployment security posture criteria - Difficult to remediate failed deployment
Intermediate	- Partial deployment automation - Medium-sized, monthly releases - Basic deployment security posture criteria - Acceptable failed deployment remediation times
Advanced	- High deployment automation - Small, weekly releases - Detailed deployment security posture criteria - Fast failed deployment remediation times
Expert	- Full deployment automation - Numerous daily releases - Automated deployment failing - Bias to fast forward fixes

Competency: Operate

Level	Description
Beginner	- Manual provisioning/configuration - Long capacity planning cycles - Manual scaling - Single availability zone
Intermediate	- Partial configuration/provisioning automation - OpEx-based capacity planning - Partial autoscaling - Multi-availability zone/region
Advanced	- Extensive configuration/provisioning automation - Capacity planning based on seasonality/growth - Significant autoscaling - Multiple cloud providers / high availability
Expert	- All infrastructure configurations and instructions instantiated as code - Capacity planning based on granular usage trends/predictions - Comprehensive auto-scaling - Multiple cloud providers / very high availability

Competency: Observe & Respond

Level	Description
Beginner	- No chaos testing or red teaming - Poor patching hygiene - No disaster recovery strategy - No SLOs formed
Intermediate	- Basic chaos testing or red teaming - Basic patching hygiene - Basic DR strategy - Basic SLOs formed
Advanced	- Significant chaos testing & red teaming - Fast patching - Comprehensive DR strategy - SLOs & error budgets favored
Expert	- Continuous chaos testing & red teaming - Patching SLA - DR plans tested often - SLOs & error budgets drive decisions

DevSecOps Maturity Model Measurement

Step 1: Assess Where Your Organization Is

Maturity by Competency

Competency	Beginner	Intermediate	Advanced	Expert
Culture				
Plan & Develop				
Build & Test				
Operate				
Release & Deploy				
Observe & Respond				

Step 2: Define Where Your Organization Needs to Go

Target Maturity by Competency

Competency	Current Level	Target Level (12-18 months)
Culture		
Plan & Develop		
Build & Test		
Operate		
Release & Deploy		
Observe & Respond		

Step 3: Determine How You'll Get There by Prioritizing Initiatives

Initiatives and Ownership

Quarter	Competency	Initiative Owner
Q1	Culture	Q1 Owner
Q1	Plan & Develop	Q1 Owner
Q1	Build & Test	Q1 Owner
Q2	Operate	Q2 Owner/Leader
Q2	Release & Deploy	Q2 Owner
Q2	Observe & Respond	Q2 Owner
Q3	[Additional Initiatives]	Q3 Owner

Overall Maturity Progress

Current Level	Target Level	How We're Getting There
Beginner	Intermediate/Advanced/Expert	[Specific Steps and Initiatives]

DevSecOps Metric Costs Customer Revenue

Four Primary Value Drivers

1. Faster, More Agile Delivery and Reduced Time to Market
2. Improved Security Posture and Reduced Risk
3. Reduced Operational and Development Costs
4. Improved Customer Experiences and Satisfaction

Metrics

Value Driver	Metrics	Costs	Customer Revenue

Faster, More Agile Delivery and Reduced Time to Market	Release Frequency, Time to Market	Tech Support Center Costs, Engineer Time to Resolve Incidents, QA Time Required to Identify, Recreate, and Document Defects, Developer Wait Time	Revenue from New Customers, Revenue from Increase in Share of Wallet, Revenue from Accelerated Time to Market, Revenue from New Products, Revenue from Pricing Innovation
Improved Security Posture and Reduced Risk	Issues Identified in Dev & QA Environments, MTTD/MTTR, FTEs Involved per Incident	Financial Losses Due to Performance Degradation or Security Incidents, Tech Support Center Costs, Engineer Time to Resolve Incidents	Lost Revenue Due to Outages, Revenue from Reduced Churn, Revenue from Higher Share of Wallet
Reduced Operational and Development Costs	Incidents/Outages, QA Time Required to Identify, Recreate, and Document Defects, Developer Wait Time, Over-Provisioning Infrastructure	Tech Support Center Costs, Engineer Time to Resolve Incidents, QA Time Required to Identify, Recreate, and Document Defects, Developer Wait Time, Over-Provisioning Infrastructure	Revenue from Reduced Churn, Revenue from Higher Share of Wallet
Improved Customer Experiences and Satisfaction	Customer Complaint Calls, Customer Satisfaction, Customer Share of Wallet, Customer Churn	Tech Support Center Costs, Engineer Time to Resolve Incidents, QA Time Required to Identify, Recreate, and Document Defects, Developer Wait Time	Revenue from New Customers, Revenue from Increase in Share of Wallet, Revenue from Reduced Churn, Revenue from Higher Share of Wallet

Note: This table outlines the key metrics, costs, and customer revenue impacts associated with each value driver in DevSecOps.

Model

ID	Stage	Level 1: Basic understanding of security practices	Level 2: Adoption of basic security practices	Level 3: High adoption of security practices	Level 4: Very high adoption of security practices	Level 5: Advanced deployment of security practices at scale
1	Build and Deployment: Build	Defined build process	Building and testing of artifacts in virtual environments	Pinning of artifacts	SBOM of components	Signing of code
		[none]	[none]	[none]	[none]	[none]
		Signing of artifacts				
		[none]				
2	Build and Deployment: Deployment	Defined deployment process	Defined decommissioning process	Environment depending configuration parameters (secrets)	Evaluation of the trust of used components	Handover of confidential parameters
		[none]	[none]	[none]	[none]	[none]
		Inventory of dependencies	Inventory of running artifacts	Rolling update on deployment	Same artifact for environments	Usage of feature toggles
		[none]	[none]	[none]	[none]	[none]
		Blue/Green Deployment				
		[none]				
3	Build and Deployment: Patch Management	A patch policy is defined	Automated PRs for patches	Nightly build of images (base images)	Reduction of the attack surface	Usage of a maximum lifetime for images
		[none]	[none]	[none]	[none]	[none]
		Usage of a short maximum lifetime for images				

		[none]				
4	Culture and Organization: Design	Conduction of simple threat modeling on technical level	Information security targets are communicated	Conduction of simple threat modeling on business level	Creation of simple abuse stories	Creation of threat modeling processes and standards
		[none]	[none]	[none]	[none]	[none]
		Conduction of advanced threat modeling	Creation of advanced abuse stories			
		[none]	[none]			
5	Culture and Organization: Education and Guidance	Ad-Hoc Security trainings for software developers	Security consulting on request	Each team has a security champion	Regular security training for all	Regular security training of security champions
		[none]	[none]	[none]	[none]	[none]
		Reward of good communication	Security code review	Conduction of build-it, break-it, fix-it contests	Security Coaching	Security-Lessoned-Learned
		[none]	[none]	[none]	[none]	[none]
		Simple mob hacking	Aligning security in teams	Conduction of collaborative team security checks	Conduction of war games	Regular security training for externals
		[none]	[none]	[none]	[none]	[none]
		Conduction of collaborative security checks with developers and system administrators				
		[none]				
6	Culture and Organization: Process	Definition of simple BCDR practices for critical components	Approval by reviewing any new version	Definition of a change management process		
		[none]	[none]	[none]		
7	Implementation: Application Hardening	App. Hardening Level 1 (50%)	Contextualized Encoding	App. Hardening Level 1	App. Hardening Level 2 (75%)	App. Hardening Level 2
		[none]	[none]	[none]	[none]	[none]
		App. Hardening Level 3				
		[none]				
8	Implementation: Development and Source Control	Versioning	Source Control Protection	API design validation	.gitignore	Local development linting & style checks performed
		[none]	[none]	[none]	[none]	[none]
9	Implementation: Infrastructure Hardening	MFA for admins	Simple access control for systems	Usage of edge encryption at transit	Applications are running in virtualized environments	Backup
		[none]	[none]	[none]	[none]	[none]
		Baseline Hardening of the environment	Isolated networks for virtual environments	MFA	Usage of an security account	Usage of encryption at rest

		[none]	[none]	[none]	[none]	[none]
		Usage of test and production environments	Virtual environments are limited	Filter outgoing traffic	Immutable infrastructure	Infrastructure as Code
		[none]	[none]	[none]	[none]	[none]
		Limitation of system events	Role based authentication and authorization	Usage of internal encryption at transit	Usage of security by default for components	WAF baseline
		[none]	[none]	[none]	[none]	[none]
		Hardening of the Environment	Production near environments are used by developers	Usage of a chaos monkey	WAF medium	Microservice-architecture
		[none]	[none]	[none]	[none]	[none]
		WAF Advanced				
		[none]				
10	Information Gathering: Logging	Centralized system logging	Logging of security events	Visualized logging	Centralized application logging	Correlation of security events
		[none]	[none]	[none]	[none]	[none]
		PII logging concept				
		[none]				
11	Information Gathering: Monitoring	Simple application metrics	Simple budget metrics	Simple system metrics	Alerting	Monitoring of costs
		[none]	[none]	[none]	[none]	[none]
		Visualized metrics	Advanced availability and stability metrics	Audit of system events	Deactivation of unused metrics	Grouping of metrics
		[none]	[none]	[none]	[none]	[none]
		Targeted alerting	Advanced app. metrics	Coverage and control metrics	Defense metrics	Screens with metric visualization
		[none]	[none]	[none]	[none]	[none]
		Metrics are combined with tests				
		[none]				
12	Test and Verification: Application tests	Security unit tests for important components	Security integration tests for important components	Smoke Test	High coverage of security related module and integration tests	
		[none]	[none]	[none]	[none]	
13	Test and Verification: Consolidation	Simple false positive treatment	Treatment of defects with severity high or higher	Simple visualization of defects	Generation of Patch Management Statistics	Generation of Response Statistics
		[none]	[none]	[none]	[none]	[none]
		Integration of vulnerability issues into the development process	Treatment of defects with severity middle	Usage of a vulnerability management system	Advanced visualization of defects	Reproducible defect tickets
		[none]	[none]	[none]	[none]	[none]
		Treatment of all defects				

		[none]				
14	Test and Verification: Dynamic depth for applications	Coverage of client side dynamic components	Simple Scan	Usage of different roles	Coverage of hidden endpoints	Coverage of more input vectors
		[none]	[none]	[none]	[none]	[none]
		Coverage of sequential operations	Usage of multiple scanners	Coverage analysis	Coverage of service to service communication	
		[none]	[none]	[none]	[none]	
15	Test and Verification: Dynamic depth for infrastructure	Test for exposed services	Test network segmentation	Test of the configuration of cloud environments	Test for unauthorized installation	Weak password test
		[none]	[none]	[none]	[none]	[none]
		Load tests	Test for unused Resources			
		[none]	[none]			
16	Test and Verification: Static depth for applications	Software Composition Analysis (server side)	API design validation	Static analysis for important server side components	Local development security checks performed	Software Composition Analysis (client side)
		[none]	[none]	[none]	[none]	[none]
		Static analysis for important client side components	Static analysis for all self written components	Usage of multiple analyzers	Dead code elimination	Exclusion of source code duplicates
		[none]	[none]	[none]	[none]	[none]
		Static analysis for all components/libraries	Stylistic analysis			
		[none]	[none]			
17	Test and Verification: Static depth for infrastructure	Test for stored secrets	Test cluster deployment resources	Test for image lifetime	Test of virtualized environments	Test the cloud configuration
		[none]	[none]	[none]	[none]	[none]
		Test the definition of virtualized environments	Analyze logs	Test for malware	Test for new image version	Correlate known vulnerabilities in infrastructure with new image versions
		[none]	[none]	[none]	[none]	[none]
		Test for known vulnerabilities	Test of infrastructure components for known vulnerabilities			
		[none]	[none]			
18	Test and Verification: Test-Intensity	Default settings for intensity	High test intensity	Regular tests	Deactivating of unneeded tests	Creation and application of a testing concept
		[none]	[none]	[none]	[none]	[none]

Tools

Git Secrets Management Tools

git-secrets

- **Description:** AWS labs tool preventing you from committing secrets to a git repository.
- **Uses Method:** Prevents accidental commit of sensitive information to git repositories.
- **Suggested Maturity Model:** Early to Mature stages of DevSecOps implementation.

git-hound

- **Description:** Searches secrets in git.
- **Uses Method:** Scans git repositories for secrets and sensitive information.
- **Suggested Maturity Model:** Mature DevSecOps practices.

Threat Modeling Tools

goSDL

- **Description:** Security Development Lifecycle checklist.
- **Uses Method:** Provides a checklist for integrating security into the development lifecycle.
- **Suggested Maturity Model:** Early stages of DevSecOps implementation.

ThreatPlaybook

- **Description:** Threat modeling as code.
- **Uses Method:** Enables codified threat modeling for better integration into DevOps.
- **Suggested Maturity Model:** Intermediate to Mature stages.

Threat Dragon

- **Description:** OWASP Threat modeling tool.
- **Uses Method:** Offers a visual tool for threat modeling in software development.
- **Suggested Maturity Model:** Any stage of DevSecOps maturity.

Secrets Management

GitLeaks

- **Description:** Gitleaks is a scanning tool for detecting hardcoded secrets.
- **Uses Method:** Scans repositories for secrets and sensitive information.
- **Suggested Maturity Model:** Intermediate to Mature stages.

ggshield

- **Description:** GitGuardian shield (ggshield) is a CLI application that runs in your local environment or in a CI environment and helps you detect more than 350+ types of secrets and sensitive files.
- **Uses Method:** Local and CI environment scanning for secrets.
- **Suggested Maturity Model:** Mature DevSecOps practices.

OSS and Dependency Management

CycloneDX

- **Description:** CycloneDX format for SBOM.
- **Uses Method:** Provides a standard format for software bill of materials.
- **Suggested Maturity Model:** Intermediate to Mature stages.

Snyk

- **Description:** Scans and monitors your projects for security vulnerabilities.
- **Uses Method:** Continuous scanning and monitoring of project dependencies.
- **Suggested Maturity Model:** Early to Mature stages of DevSecOps implementation.

SAST (Static Application Security Testing)

Brakeman

- **Description:** Brakeman is a static analysis tool which checks Ruby on Rails applications for security vulnerabilities.
- **Uses Method:** Static code analysis for Ruby on Rails applications.
- **Suggested Maturity Model:** Intermediate to Mature stages.

Semgrep

- **Description:** Hi-Quality Open source, works on 17+ languages.
- **Uses Method:** Static code analysis across multiple programming languages.
- **Suggested Maturity Model:** Any stage of DevSecOps maturity.

DAST (Dynamic Application Security Testing)

Zap proxy

- **Description:** Zap proxy providing various docker containers for CI/CD pipeline.
- **Uses Method:** Offers dynamic application security testing, particularly for web applications.
- **Suggested Maturity Model:** Intermediate to Mature stages.

Wapiti

- **Description:** Light pipeline ready scanning tool.
- **Uses Method:** Dynamic security scanning of web applications.
- **Suggested Maturity Model:** Intermediate to Mature stages.

Continuous Deployment Security

1. **SecureCodeBox**
   - **URL:** SecureCodeBox
   - **Description:** Toolchain for continuous scanning of applications and infrastructure.
   - **Uses Method:** SCB
   - **Suggested Maturity Model:** Advanced Continuous Scanning Integration
2. **OpenSCAP**
   - **URL:** OpenSCAP
   - **Description:** Open Source Security Compliance Solution.
   - **Uses Method:** oscap
   - **Suggested Maturity Model:** Compliance and Security Standards Enforcement
3. **ThreatMapper**

- **Description:** Hunts for vulnerabilities in production platforms, ranking them based on risk-of-exploit.
- **Uses Method:** kube-hunter
- **Suggested Maturity Model:** Real-time Vulnerability Assessment

Kubernetes

1. **KubiScan**
   - **URL:** KubiScan
   - **Description:** A tool for scanning Kubernetes cluster for risky permissions.
   - **Uses Method:** Kubiscan
   - **Suggested Maturity Model:** Permission Management and Risk Assessment
2. **Kubeaudit**
   - **URL:** Kubeaudit
   - **Description:** Audit Kubernetes clusters for various security concerns.
   - **Uses Method:** kube-audit
   - **Suggested Maturity Model:** Comprehensive Cluster Auditing
3. **Kubescape**
   - **URL:** Kubescape
   - **Description:** First open-source tool for testing Kubernetes deployment according to NSA-CISA and MITRE ATT&CK®.
   - **Uses Method:** kubescape
   - **Suggested Maturity Model:** NSA-CISA Compliance and Security Testing

Continuous Deployment Security

1. **SecureCodeBox**
   - **URL:** SecureCodeBox
   - **Description:** Toolchain for continuous scanning of applications and infrastructure.
   - **Uses Method:** SCB
   - **Suggested Maturity Model:** Advanced Continuous Scanning Integration
2. **OpenSCAP**
   - **URL:** OpenSCAP
   - **Description:** Open Source Security Compliance Solution.
   - **Uses Method:** oscap
   - **Suggested Maturity Model:** Compliance and Security Standards Enforcement
3. **ThreatMapper**
   - **URL:** ThreatMapper
   - **Description:** Hunts for vulnerabilities in production platforms, ranking them based on risk-of-exploit.
   - **Uses Method:** kube-hunter
   - **Suggested Maturity Model:** Real-time Vulnerability Assessment

Kubernetes

1. **KubiScan**
   - **URL:** KubiScan
   - **Description:** A tool for scanning Kubernetes cluster for risky permissions.
   - **Uses Method:** Kubiscan
   - **Suggested Maturity Model:** Permission Management and Risk Assessment
2. **Kubeaudit**
   - **URL:** Kubeaudit
   - **Description:** Audit Kubernetes clusters for various security concerns.
   - **Uses Method:** kube-audit
   - **Suggested Maturity Model:** Comprehensive Cluster Auditing
3. **Kubescape**
   - **URL:** Kubescape
   - **Description:** First open-source tool for testing Kubernetes deployment according to NSA-CISA and MITRE ATT&CK®.
   - **Uses Method:** kubescape
   - **Suggested Maturity Model:** NSA-CISA Compliance and Security Testing
4. **kubesec**
   - **URL:** kubesec
   - **Description:** Security risk analysis for Kubernetes resources.
   - **Uses Method:** kubesec
   - **Suggested Maturity Model:** Kubernetes Resource Security Analysis
5. **kube-bench**
   - **URL:** kube-bench
   - **Description:** Kubernetes benchmarking tool.
   - **Uses Method:** Kubiscan