DevSecOps and Governance as Code

Joshua Bodmer, Eugene Browder, Jason Ellard, Faith Karanja

Kyle Moore, Bharath Mukka, Tomilola Ogundare

University of Missouri- St. Louis

Table of Contents

DevSecOps Overview

DevSecOps Goals

This DevSecOps development aims to create a framework for a software pipeline that conforms to external regulatory compliance and secures the company's digital assets.

DevSecOps Organization

The culture of the DevSecOps group should embrace these concepts and incorporate them into their daily routine. The team will be trained with new DevSecOps ideas and technologies. So, that collectively, they will all take responsibility for the result. All communication barriers will be lowered from one group to another to promote collaboration throughout the pipeline. QA results should be available to all teams in the pipeline and the software lifecycle phase to make team decisions. The team must feel secure in sharing the result of their work regardless of whether those results or good or bad. The results should be seen as a starting point for improvement. The team should strive for small sprint development times with feedback from the customer than larger ones. A DevSecOps should create new requirements based on user feedback. The team should constantly refactor and clean up the code.

DevSecOps Roles

The DevSecOps team's roles are Management, Security Management, Development, QA, IT, Operations, Application Architects, Cloud Architect, Identity Access Management, Cloud Security Team, Application Security Team, Data Controller, Data Processor, Data Protection Group, and the Customer.

Security Management is the new team added from the DevOps concepts. The management role in DevSecOps is to define the acceptable risk associated with each segment of the pipeline. Security management identifies the security requirements and the need for security compliance to support the business's success. The security manager may define security awareness programs, security assurance programs, security guidelines, and processes or tools for the development, testing, and security monitoring team to achieve this goal. The development team is to build secure software and services with rapid delivery. The principles of security and privacy by design will apply to the whole development cycle, from the security requirements, secure architecture frameworks, hardening compiler options, secure coding, and secure third-party dependencies. We have listed many industry best practices, suggested frameworks, and secure code scanning tools for the development team. The Development team may apply security practices. QA is responsible for quality testing the code after deployment. IT is responsible for the movement, the reading, the writing, and the update of data. Operation is responsible for the execution and feedback of the application. Application Architect plays an essential role in the design and analysis of software projects. They create new applications, improve existing applications, run software tests, develop product prototypes, create technical documents and manuals relating to application development.

A Cloud Architect is responsible for converting the technical requirements of a project into the architecture and design that will guide the final product. Often, Cloud Architects are also responsible for bridging the gaps between complex business problems and solutions in the cloud. Identity management, also known as identity and access management, is the group that manages a framework of policies and technologies for ensuring that the right users have the appropriate access to technology resources. The cloud security team prevents unauthorized data breaches, malware and ransomware attacks, identity theft, and fraud, significantly reducing the operational risks for your company. The application security team is responsible for making apps more secure by finding, fixing, and enhancing the security of apps. A Data Controller is a person, company, or other body that determines the purpose and means of personal data processing (this can be determined alone or jointly with another person, company, or body). A Data Processor is a person or organization who deals with personal data as instructed by a controller for specific purposes and services offered to the controller that involve personal data processing.

DevSecOps Responsibilities of Roles - Planning

Management

- Responsible for meeting with the customer to determine what the requirements are for the application.
- Responsible for Organizational cybersecurity policy is established and communicated.
- Responsible for Cybersecurity roles and responsibilities are coordinated and aligned with internal roles and external partners.
- Responsible for Legal and regulatory requirements regarding cybersecurity, including privacy and civil liberties obligations, are understood and managed.
- Responsible for Governance and risk management processes address cybersecurity risks.
- Risk management processes are established, managed, and agreed to by organizational stakeholders.
- Organizational risk tolerance is determined and clearly expressed.
- The organization's determination of risk tolerance is informed by its role in critical infrastructure and sector specific risk analysis.

Security Management

- Physical devices and systems within the organization are inventoried.
- Software platforms and applications within the organization are inventoried.
- Organizational communication and data flows are mapped.
- External information systems are catalogued.
- Resources (e.g., hardware, devices, data, time, and software) are prioritized based on their classification, criticality, and business value.
- Cybersecurity roles and responsibilities for the entire workforce and third-party stakeholders (e.g., suppliers, customers, partners) are established
- Asset vulnerabilities are identified and documented.
- Cyber threat intelligence and vulnerability information is received from information sharing forums and
- Threats, both internal and external are identified and documented.
- Potential business impacts and likelihoods are identified.
- Threats, vulnerabilities, likelihoods, and impacts are used to determine risk.
- Risk responses are identified and prioritized.

Development

- Responsible for establishing coding standards and conventions.
- Responsible for using safe functions only.
- Responsible for using code analysis tools
- Responsible for handling data safely.
- Responsible for third-party components.

Data Protection Group

- Required by GDPR, a data protection group must be established to satisfy compliance.
- Responsible for monitoring compliance with the UK GDPR and other data protection laws or data protection policies.
- Monitors internal compliance, informs, and advises on data protection obligations.
- Responsible for awareness-raising, training, and audits.
- Provides advice regarding Data Protection Impact Assessments (DPIAs).
- Point of contact for the ICO and for individuals whose data is processed (employees, customers, etc.).

Customer

- Provides details on the necessary functionality, design, and other various needs of the project.
- Provides continuous feedback to make improvements.
- Engages with success metrics established to measure progress and gauge success.
- Validates criteria are being met.
- Provides project deadline and provides information that leads to project timelines.

Quality Assurance

- Integral group that links the development and operations teams by testing applications against expected delivery and performance parameters.
- Should strive for automation over manual testing procedures.
- Should not only focus on finding code and feature issues but should also work to identify pipeline process challenges.
- Should be a flexible, adaptive team that can mold its operation to changing environments and issues.
- Acts as an advocate for quality throughout the CI/CD pipeline such that testing is enabled, enacted, and audited at each phase.

Cloud Architect

- Leads cloud initiatives by developing strategies and processes that are congruent with the organization's overall goals.
- Culture advocate for all cloud initiatives.
- Constructs and maintains a team of cloud engineers and developers with sufficient talent to carry out the cloud initiatives.
- Oversees and manages a parent cloud security plan in coordination with cloud security team.
- Manages all cloud budgets and cost modeling.
- Authors a "best practices" policy for all organization cloud activities.

Data Controller

- Decides what personal information to collect.
- Decides how long the information will be kept.
- Responsible for modifications to the data.
- Decides what the data will be used for.
- Decides if the data will be shared.
- Decides with whom the data will be shared.

Data Processor

- Responsible for collection of personal data.
- Responsible for the systems, methods and tools used to collect personal data.
- Responsible for the security controls used to protect the data.
- Responsible for the storage of the data.
- Responsible for the transfer of the data.

Identity and Access Management Team

- Understand the business, entity, and the IT environment before enforcing the IAM Operations.
- Should be able to understand what exactly the customer needs and what kind of access mechanisms to be enforced in the organization. Should be able to communicate with internal and external people who are involved in the project or organization.
- Collaborate with other teams in the organization while architecting the IAM by having a prior understanding of emerging security threats and staying abreast of new security technologies to integrate them into the current architecture.
- Prepare and maintain a complete documentation of the process to be followed, SOPs, SLA agreements, resiliency plan documentation, and incident response structure internal for IAM.
- Identify the significant accounts, and the processes involved.
- Able to have sound knowledge in the third-party applications that are to be implemented in IAM processes.
- Define, prioritize, and analyze the functional specifications of IAM
- Due diligence of Administrator credentials, Categorization of user management, implementing Multi-Factor Authentication, Enforcing Password hygiene and rotate access tokens, enforce least privilege access management, Risk based access, Management of shared secrets and hard-corded passwords, and implementing end to end visibility so that it becomes easy for organizations to detect misconfigurations and changes, and makes to respond quickly in an efficient manner.
- The IAM manager should be able to server as an SME (Subject Matter Expert)
- Assurance to corporate and Information Security standards, procedures, and policies.
- Should be able to troubleshoot and prepare a root cause analysis for everything that happens in view of security and access management.
- Secure provisioning, Maintenance, and deprovisioning of all user accounts.
- Should be able to achieve Identity Access Management compliance on all the involved requirements, including some Sarbanes Oxley Act, global data privacy requirements, and the state and federal regulations.

Cloud Security Team

- Should be able to understand the technologies, methodologies, and principles which are designed to control and secure the cloud environment.
- Should be able to create cloud-based programs and securely configuring the cloud environments.
- Able to communicate well between the internal and external teams, cloud service providers, and the customers.
- Should be able to understand and making suggestions on the design architecture of the application to be in the cloud in a secure way. Able to suggest on what cloud service model and cloud deployment model can best suit the organization.
- Implementation of cryptography and management of encryption in the cloud.
- Monitoring, Logging, and responding to the incidents that are detected in the cloud environments.
- Design and document strategies being implemented in the cloud, technologies used, documentation on how the API interacts with the cloud, security controls placed, designing, and documenting a disaster recovery plan, and documentation on all the involved processes in the cloud that are related to the organization.
- Should have a sound understanding on how the cloud works, how security is to be implemented, how the cloud service provider maintains and manages the data.
- Knowledge of how cloud networks are designed, however, most of the cloud service providers ensure that they are secure.
- Perform regular and rapid tests like the penetrations testing, DAST, RASP, Vulnerability scans, tests on cloud containers and on all the services deployed in the cloud to ensure the security and by implementing the Zero-trust architecture.
- The cloud security team should also be aware of the existing vulnerabilities, threats, and how the attacks are happening on the cloud environments, and make sure that the solution they are following is up to date.
- Able to design and understand an incident response architecture.
- An understanding of how a third-party application is integrated with the organization's applications in the cloud and how much secure they are.
- Being compliant with the national and international regulatory standards, and frameworks.
- At the end, security of cloud is a shared responsibility between the Cloud Service Provider and the Application Service Provider, or the organization.

Application Security Team

- Responsible for setting security controls and design requirements during the software build and development stage.
- Responsible for code review, testing source code and running code including but not limited to Static AppSec testing (SAST), and Dynamic AppSec testing (DAST)
- Implementing advanced security features through Code obfuscation or runtime application self-protection.
- Provide designs for new software solutions to help mitigate security vulnerabilities.

Application Architects

- Creating high-level product specifications and design documents.
- Providing the development team with architectural blueprints to follow.
- Ensuring that uniform application design standards are maintained.
- Troubleshooting and resolving issues with coding or design.
- Participate in all aspects of the software development process including but not limited to determining business goals, prototype modeling, risk evaluation.

DevSecOps Feedback Loops

The CI/CD pipeline has feedback loops, as seen in Figure 1: CI/CD Pipeline. Each section will describe the purpose of the feedback loops that originate in that section.

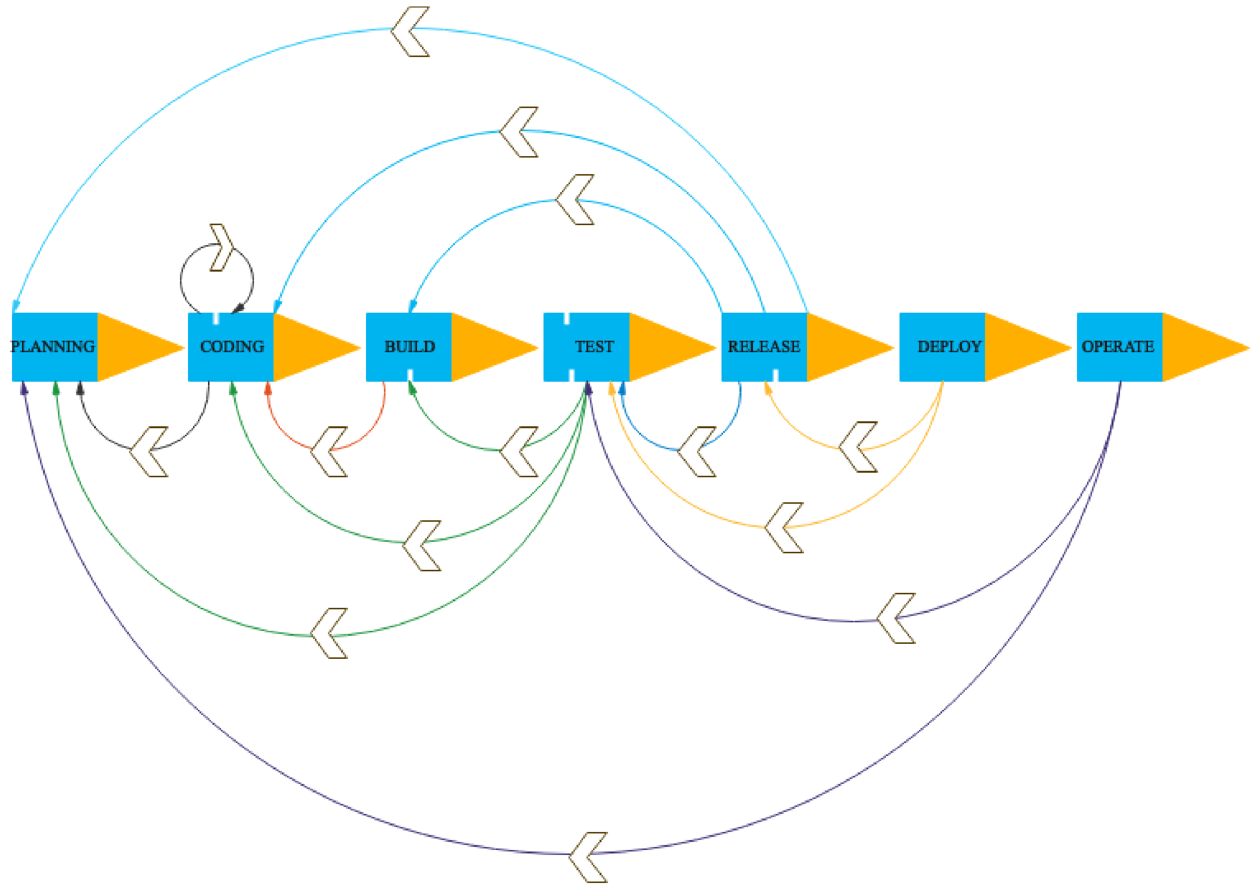

Figure 1: CI/CD Pipeline

DevSecOps Planning

Planning is the first approach to any task at hand, and the core focus of DevSecOps—security—begins from here. The focus should be on security and performance, acceptance test criteria, application interface, and functionality and threat-defense models. The planning phase involves activities that help the project manage time, cost, quality, risk, and issues. These activities include a business-need assessment, project plan creation, feasibility analysis, risk analysis, business requirements gathering, business process creation, system design, DevSecOps design, and ecosystem instantiation, etc. The planning phase repeats when DevSecOps the lifecycle recycles. Teams involved include development teams, operations teams, security teams, and quality assurance teams. Communication channels must be set up so that security specialists and necessary SMEs are available at every step and can provide input wherever it is needed. All people involved must know their role and how it relates to delivering a product that will withstand hostile action. The customer must be engaged throughout the entire process and provide continuous feedback to meet the product needs. A successful product must be both usable and secure and must be created so that one is not sacrificed for the other.

Planning Objective

Understand what the design is going to be, how it will be designed, how it is going to measured (regulatory standards, industry best practices), and the scope of the overall project. Identify the conditions under which development needs to be managed (corporate policies, coding standards, requirements, regulations). Understand the roles of the entire process, who will be needed at each step, and the level of expertise required.

Planning Capabilities and Activities

Plan Phase Capabilities

DevSecOps planning activities are supported by capabilities that facilitate communication, collaboration, project management, change management, automation, and consistent process execution to foster human interaction and increasing team productivity. These capabilities are meant to assist in software development planning, including configuration management planning, change management planning, project management planning, system design, software design, test planning, and security planning. Some capabilities will be used throughout the software lifecycle, such as team collaboration, an issue tracking system, and a project management system. Some capabilities are shared at the enterprise level across programs. Policy and enforcement strategies should be established for access controls on various tools. Below are capabilities that can assist with DevSecOps planning.

- **Team collaboration system**
  - **Features:** Communication channels such as audio or video conferencing, chat/messaging, and group calendars. Information sharing means brainstorming discussion boards, internal information "Wiki" websites, and files shares.
  - **Benefits:** A team collaboration system simplifies communication and boosts team efficiency & collaboration.

    - **Inputs:** Using a team collaboration system/tools requires gathered documentation, information, notes, and designs, as well as team & customer meetings.
    - **Outputs:** Alignment, organized teamwork, version-controlled documents.
- **Issue tracking system**
    - **Features:** An issue tracking system enables support agents, engineers, and managers to track problems until they have been successfully resolved. The issue tracking includes bugs and defect management, assignment management, knowledge base management, feature and change management, prioritization management, and escalation management capabilities.
    - **Benefits:** The purpose of this system is so defects can be tracked, and information & improvements can be attributed to those defects. Defect trends can be identified. Software quality can be improved. The system facilitates organization and improves efficiency while reducing the cost of development.
    - **Inputs:** Issue/bug report, root cause analysis solutions
    - **Outputs:** Issue feature/change tickets. Issue resolution tracking history

- **Asset inventory management**
    - **Features:** Collect information about all IT assets. A real-time inventory of all applications, software licenses, libraries, operating systems, and versioning information.
    - **Benefits:** Increase situation awareness, tracking of assets, improved efficiency by understanding equipment & software utilization.
    - **Inputs:** IT assets (applications, software licenses, libraries, operating systems, and versioning information)
    - **Outputs:** Asset inventory

- **Configuration management database (CMDB*)***
    - **Features:** Acts as a data warehouse for the organization and contains all relevant information about the hardware and software components used in an organization's IT (information technology) services and their relationships.
    - **Benefits:** Centralized database used by many systems (such as asset management, configuration management, incident management, etc.) during development and operations phases.
    - **Inputs:** IT hardware and software components information
    - **Outputs:** Configuration items provides a means of understanding the organization's critical assets and their relationships

- **Project management system**
    - **Features:** To help plan, organize, and manage resource tools and develop resource estimates. Includes task management scheduling, time management, resource management, budget management, risk management.
    - **Benefits:** Assist project progress tracking and optimizes resource allocation.
    - **Inputs:** Tasks, scheduling, resource allocation, etc.

    - **Outputs:** Project plan

- **Software system design**
    - **Features:** Assists requirement gathering, system architecture design, components design, and interface design.
    - **Benefits:** Independent of programming languages, helps visualize the software system design.
    - **Inputs:** User requirements design ideas
    - **Outputs:** System design documents, function design document, test plan, system deployment environment configuration plan.

- **Threat modeling**
    - **Features:** Includes identifying and prioritizing potential threats and security mitigations. Document system security design analyzes the design for potential security issues; review and analysis against common attack patterns; suggest and manage mitigation.
    - **Benefits:** Allows software architects to identify and mitigate potential security issues early.
    - **Inputs:** System design
    - **Outputs:** Potential threats and mitigation plan.

- **Data modeling**
    - **Features:** To create a visual description of a business by analyzing, understanding, and clarifying data requirements and how they underpin business processes**.** Models the interrelationship and flows between different data elements.
    - **Benefits:** Higher quality product, catch errors and oversights early, build software faster, ensures the required data objects by the system are accurately represented
    - **Inputs:** System requirement; Business logic
    - **Outputs:** Data model

Plan Phase Activities

The activities supported by the planning phase are listed below. Many of these activities are continuous throughout the DevSecOps lifecycle.

- **DevSecOps ecosystem design**
    - **Description:** Determine the DevSecOps process workflows specific to the organization's needs and related to the type of work/projects being completed.
    - **Inputs:** Change management process, system design, release plan & schedule.
    - **Outputs:** DevSecOps process flow chart, DevSecOps system tool selection, deployment platform selection
    - **Capability Dependencies:** Team collaboration system
    - **People:** Upper management, project managers, lead developers, lead security experts

- **Project team onboarding planning**
    - **Description:** Plan the project team onboarding process, interface, access control policy, and education of team members. Everyone involved in the project must understand business objectives and requirements, know their role, and the project's expected outcome.
    - **Inputs:** Organization policy
    - **Outputs:** Onboarding plan
    - **Capability Dependencies:** Team collaboration system
    - **People:** Upper management, project managers, lead developers, lead security experts, human resources

- **Change management planning**
    - **Description:** A change management plan serves as the roadmap defining concrete steps an organization will take to execute the change management process. This process must be planned into a DevSecOps system.
    - **Inputs:** Organizational policy, software development best practice.
    - **Outputs:** Change control procedures, review procedures, control review board change management plan
    - **Capability Dependencies:** Team collaboration system, Issue tracking system.
    - **People:** Upper management, project managers

- **Configuration management planning**
    - **Description**: Process for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design, and operational information throughout its life. This process must be planned into a DevSecOps system.
    - **Inputs:** Software development, security, and operations best practice, IT infrastructure assets, software system components.
    - **Outputs:** CM processes and plan, CM tool selection, responsible configuration items; tagging strategy
    - **Capability Dependencies:** Team collaboration system, Issue tracking system
    - **People:** Quality assurance teams, lead developers, project managers

- **Software requirement analysis**
    - **Description:** Gather the requirements from all stakeholders.
    - **Inputs:** Stakeholder inputs or feedback, operation monitoring feedback, test feedback
    - **Outputs:** Feature requirements, performance requirements, privacy requirements, security requirements
    - **Capability Dependencies:** Team collaboration system, Issue tracking system
    - **People:** Project managers, stakeholders

- **System design**
    - **Description:** The process of defining elements of a system like modules, architecture, components, and their interfaces and data for a system based on the specified requirements.
    - **Inputs:** Requirements documents
    - **Outputs:** Documents: System architecture, functional design, data flow diagrams, test plan, infrastructure configuration plan, tool selections, development tool, test tool, Deployment platform
    - **Capability Dependencies:** Team collaboration system, Issue tracking system, software system design tools
    - **People:** Application Architects, Security specialists

- **Project planning**
    - **Description:** Includes project task management, release planning, resource delegation, necessary people identification.
    - **Inputs:** collaboration
    - **Outputs:** Task plan & schedule, release plan & schedule
    - **Capability Dependencies:** Team collaboration system, project management system
    - **People:** Project managers, lead developers, lead security experts

- **Risk management**
    - **Description:** Process of identifying, assessing, and controlling threats to an organization’s capital and earnings. Perform risk assessment. Vulnerability management, a subset of risk management, is also performed.
    - **Inputs:** System architecture, supply chain information, security risks.
    - **Outputs:** Risk management plan
    - **Capability Dependencies:** Team collaboration system
    - **People:** Upper management, project managers, lead developers, lead security experts

- **Configuration identification**
    - **Description:** The process of identifying the attributes that define every aspect of the product, which include details on the product’s performance, functional, and physical attributes, design, and operational information.

- **Inputs:** IT infrastructure asset, software system components, code/document baselines
- **Outputs:** Configuration items
- **Capability Dependencies:** team collaboration system, CMDB, source code repository, artifact repository
- **People:** Upper management, project managers, lead developers, lead security experts, stakeholders

- **Threat modeling**
  - **Description:** Identify potential threats, weaknesses, and vulnerabilities. Define the mitigation plan.
  - **Inputs:** System design
  - **Outputs:** Potential threats and mitigation plan
  - **Capability Dependencies:** A threat modeling tool
  - **People:** Security team, application security SME

- **Database design**
  - **Description:** Determines what data must be stored and how the data elements interrelate. Database design involves classifying data and identifying interrelationships. Includes: Data modeling, database selection, database deployment topology.
  - **Inputs:** System requirement, system design
  - **Outputs:** Database design document
  - **Capability Dependencies:** Data modeling tool, team collaboration system
  - **People:** Database engineers

- **Design review**
  - **Description:** Review and approve plans and documents
  - **Inputs:** Plans and design documents.
  - **Outputs:** Review comments, action items
  - **Capability Dependencies:** Team collaboration system
  - **People:** Project managers, stakeholders

- **Documentation version control**
  - **Description:** The process by which different drafts and versions of a document are recorded and managed. To track design changes.
  - **Inputs:** Plans and design documents
  - **Outputs:** Version controlled documents
  - **Capability Dependencies:** Team collaboration system
  - **People:** Project manager, developers

Planning for Governance as Code:

With governance as code, IT teams can define and automate best practice policies that manage all aspects of services, applications, and infrastructure across cost, availability, security, performance, and usage. As with all major technology changes, it will require modifications in people, process, and technology.

Implementing governance as code:

- **Get stakeholder buy-in**. A good governance as code strategy starts by getting cross-organizational commitment to a revised strategy and agreement on the proposed solution.
- **Map out a strategy.** Establish a governance strategy and be sure to define and adopt policies with cross-departmental best practices in mind.
- **Define & automate policies**. Capture best practices, standards, reference architecture, and internal constraints from the organization and teams and automate these rules in the policy engine of choice. Automating makes life easier and is essential to governing at cloud speed.
- **Track and trend.** Integrate the policies with internal incident and ticketing systems and deliver violations, recommendations, and reports to stakeholders, teams, and departments. Setting metrics provides a great way to benchmark and measure ROI**.**

Planning for Risk Management:

It is recommended to use the Risk Management Framework (RMF) Solution detailed in NIST SP 800-37. This framework offers a structured process to secure, authorize and manage IT systems. RMF defines a process cycle that is used for initially securing the protection of systems through an Authorization to Operate (ATO) and for integrating ongoing risk management (continuous monitoring). The process can be categorized into a six-step procedure that integrates information security and risk management activities into the system development lifecycle. These steps are:

Step 1: Categorize Information Systems
Step 2: Select Security Controls
Step 3: Implement Security Controls
Step 4: Assess Security Controls
Step 5: Authorize Information System
Step 6: Monitor Security Controls

DevSecOps Planning - Licensing

There are five types of licenses:

Public domain
This is the most permissive type of software license. When software is in the public domain, anyone can modify and use the software without any restrictions. But you should always make sure it's secure before adding it to your codebase. Warning: Code that doesn't have an explicit license is NOT automatically in the public domain. This includes code snippets you find on the internet.

Permissive
Permissive licenses are also known as "Apache-style" or "BSD style." They contain minimal requirements about how the software can be modified or redistributed. This type of software license is perhaps the most popular license used with free and open-source software. Aside from the Apache License and the BSD License, another common variant is the MIT License.

LGPL
The GNU Lesser General Public License allows you to link to open-source libraries in your software. If you simply compile or link an LGPL-licensed library with your code, you can release your application under any license you want, even a proprietary license. But if you modify the library or copy parts of it into your code, you'll have to release your application under similar terms as the LGPL.

Copyleft
Copyleft licenses are also known as reciprocal licenses or restrictive licenses. The most well-known example of copyleft or reciprocal license is the GPL. These licenses allow you to modify the licensed code and distribute new works based on it to distribute any new works or adaptations under the same software license. For example, a component's license might say the work is free to use and distribute for personal use only. So, any derivative you create would also be limited to personal use only. (A derivative is any new software you develop that contains the component.)

The catch here is that the users of your software would also have the right to modify the code. Therefore, you'd have to make your source code available. But of course, exposing your source code may not be in your best interests.

Proprietary
Of all types of software licenses, this is the most restrictive. The idea behind it is that all rights are reserved. It's generally used for proprietary software where the work may not be modified or redistributed.

During the development process, you will need to keep track of what part of your code is covered by what license. This is done with an SCA (Software Composition Analysis) tool. This will tell you what you can do with your code.

DevSecOps Planning – Environments

Some possible environments for our development are:

- Local – only on the development machine. This is the only place where the code is edited.
- DEV – the development environment. Only your application is deployed to a server.
- SIT – the system integration and test. Multiplication applications can be visible.
- CAT – the customer acceptance and test. An environment where the customer can sign off on functionality.
- PROD – production. The deployment of code to the customer's site.

DevSecOps Coding

Security in coding involves defining the tools that allow for threat assessment, security requirements, and defining a secure architecture. This section will explain the tools and how the tools should be used to gain security. In our current development environment, the IDE is the center. All coders should have similar IDE's. This prevents the application from working on one machine but not another because they have this or that installed. The IDE has a compiler that will generate syntactical errors. The IDE should allow coders to quickly refractor code continuously. Plugins that will test for specific security requirements. The plugins should give good feedback to trace any incidents found and allow for correction. A version control system will be used that allows for traceability of inputs. Static code analyses should be done when check-in into the branch. This version control system should allow many users to view the branch with the minimum effort quickly. The coder should have secure database interactions that protect application code data and user data. At some point, the code should be peer-reviewed on whether the code meets the company's best practices and security before being merged up into the baseline. The reviewer using the version control system chosen should be able to bring up and discuss the code easily; regardless, of the branch.

People/Roles:

Security Management and the Development team's role in coding is threat assessment, security requirements, and defining a secure architecture. Management roles in coding are to define the risk associated with the Secure Management Team and the Development Team.

Objectives:

The development team is to build secure software and services with rapid delivery. The principles of security and privacy by design will apply to the whole development cycle, from the security requirements, secure architecture frameworks, hardening compiler options, secure coding, and secure third-party dependencies. The development team may apply security practices.

Coding Phase Activities and Tools

Coding Phase Activities

These activities support the coding phase

- **Security code development**
  - **Description:** Security policy enforcement script coding
  - **Inputs:** Developer coding input
  - **Outputs:** Source code
  - **Tool Dependencies:** IDE
- **Testing**
  - **Description:** Develop detailed test procedures, test data, test scripts, test scenario configuration on the specific test tool
  - **Inputs:** Test plan
  - **Outputs:** Test procedure document; Test data file; Test scripts
  - **Tool Dependencies:** IDE; Specific test tool
- **There will be a database to store application data and user data.**

    - **Description:** Implement the data model using data definition language or data structure supported by the database; Implement triggers, views, or applicable scripts; Implement test scripts, test data generation scripts.
    - **Inputs:** Data model
    - **Outputs:** Database artifacts (including data definition, triggers, view definitions, test data, test data generation scripts, test scripts, etc.)
    - **Tool Dependencies:** IDE or tools that come with the database
- **Code commit**
    - **Description:** Commit source code into the version control system
    - **Inputs:** Source code
    - **Outputs:** Version controlled source code
    - **Tool Dependencies:** Source code repository
- **Scanning**
    - **Description:** Check the changes for sensitive information before pushing the changes to the central repository. If it finds suspicious content, it notifies the developer and blocks the commit
    - **Inputs:** Locally committed source code
    - **Outputs:** Security findings and warnings
    - **Tool Dependencies:** Source code repository; security plugin

- **Code review**
    - **Description:** Perform code review to all source code. Note that pair programming counts.
    - **Inputs:** Source code
    - **Outputs:** Review comments
    - **Tool Dependencies:** Code quality review tool

- **Documenting**
    - **Description:** Detailed implementation documentation of code and process
    - **Inputs:** User input; Source code
    - **Outputs:** Documentation; Auto-generated Application Programming Interface (API) documentation
    - **Tool Dependencies:** IDE; Document editor; or build tool

- **Static Analysis**
    - **Description:** Scan and analyze the code as the developer writes it. Notify developers of potential code weaknesses and suggest remediation.
    - **Inputs:** Source code; known weaknesses
    - **Outputs:** source code weakness findings
    - **Tool Dependencies:** IDE security plugin

Coding Phase Tools

- **Integrated development environment (IDE)**
  - **Features:** Source code editor Intelligent code completion Compiler or interpreter Debugger Build automation (integration with a build tool)
  - **Benefits:** Visual representation Increase efficiency Faster coding witless effort Improved bug fixing speed Reproduciblebuilds via scripts
  - **Inputs:** Developer coding input
  - **Outputs:** Source code

- **Specified API security plugins**
  - **Features:** Scan and analyze the code as the developer writes it, notify the developer of potential code weakness, and may suggest remediation
  - **Benefits:** Address source code weaknesses and aid developers to improve secure coding skills
  - **Inputs:** Source code; known weaknesses
  - **Outputs:** source code weakness findings

- **Source code repository**
  - **Features:** Source code version control Branching and merging Collaborative code review
  - **Benefits:** Compare files, identify differences, and merge the changes if needed before committing. Keep track of application builds
  - **Inputs:** Source code Infrastructure as code
  - **Outputs:** Version controlled source code

- **Code quality review tool**
  - **Features:** View code changes, identify defects, reject or approve the changes, and make comments on specific lines. Sets review rules and automatic notifications to ensure that reviews are completed on time.
  - **Benefits:** Automates the review process, which in turn minimizes the task of reviewing the code.
  - **Inputs:** Source code
  - **Outputs:** Review results (reject or accept), code comments

- **Code review, Code scan, and Static code**
  - Code review is done locally. It can be done with a partner, as in pair programming. Code scanning can be done locally or in DEV. Different scans using different tools can check for other things. You might be scanning locally for keywords that would violate company policy. You might scan the code in DEV for OWASP vulnerabilities. Static code scanning is done locally. This is done with third-party plugins.

DevSecOps Coding - Feedback Loop

As seen in Figure 1: CI/CD Pipeline there are two feedback loops in Coding. One feedback loop goes to planning. This feedback purpose is to give feedback that the requirements cannot be accomplished in the time allocated, the status of development, or to clarify what the development requirements are. A second feedback loop is to itself. This is code that needs to be reworked to improve clarity, functionality, or security.

DevSecOps Building

The building phase of DevSecOps entails creating a secure background for helping teams to make changes to a code more frequently and reliably. The build tools perform the tasks of building and packaging applications, services, and microservices into artifacts. A positive aspect of the build is ensuring that monitoring and automation are applied. Therefore, it is possible to identify the specific areas of weakness within the software delivery concept. The insight gives room for the necessary changes and improvements to be adopted in a way that creates the best outcomes (Red Hat, 2021).

People/Roles:

The major roles and positions involved in the build phase are software developer/tester, security engineer, and automation architect. The software developer/tester ensures the code is complete and meets business requirements as well as monitors and tests code. The security engineer works with the software developers to ensure that security comes first during the process of building the code. Lastly, the automation architect designs, implements, and analyzes continuous deployment strategies for the release of the product and continuous updates. These roles all report to their team leads or managers and then to the Chief Information Security Officer (CISO), Chief Technology Officer (CTO), and other executive teams.

Objective:

The main goal of the build phase is to develop the code into a product to be deployed and released. The concept entails adopting both the organizational and technical measures to ensure that information security is attained effectively in the whole process. The code is compiled and built while ensuring there are no errors in the code and incorporate tools like static application security testing (SAST) tools to track down flaws in code before deploying to production. These tools are specific to programming languages. To build languages like C++, the code is compiled and linked. While for interpreted languages like JavaScript or Python, a compilation is not required; however, there will be a need to use lint tools to check for possible syntax errors. For Java Virtual Machine (JVM) based languages, class files are compiled and then compressed to files such as a jar, war, or ear file.

Also, Building includes generating documentation, copying files like libraries or icons to appropriate locations, and creating a distributable file such as a tar or zip file. The build script should also include targets for running automated unit tests.

Build Phase Tools and Activities

Build Phase Tools

Build tools can be incorporated into an integrated development environment (IDE) and a source code repository to build the code both during development and after committing. For those applications that use containers, the build stage also includes a containerization tool. The necessary tools include:

- **Build Tool.**
  - This tool is used for compilation, dependency, management, linking, and Built-in lint stylistic checking Integration with IDE. This is the first tool needed in the build phases by software developers.
  - **Benefits:** Reduces human mistakes and saves time
  - **Inputs:** Source code under version control Artifact
  - **Outputs:** Binary artifacts stored in the Artifact repository
- **Lint Tool**
  - This is the next tool needed in building the code. Linting analyzes source code to flag programming errors, bugs, stylistic errors, and suspicious constructs. This applies to both compiled (i.e., C++) or interpreted Languages (i.e., JavaScript).
  - **Benefits:** Improve code readability; Pre-code review; Finding (syntax) errors before execution for interpreted languages
  - **Inputs:** Source code or scripts
  - **Outputs:** Analyze results
- **Container Builder**
  - The container build is only needed for applications that require a container, and some applications already have a built-in container like node.js. This is used to build a container image based on a build instruction file.
  - **Benefits:** Container image build automation
  - **Inputs:** Container base image; Container build file
  - **Outputs:** OCI compliant container image
- **Artifact Repository**
  - This is an application designed to store, version, and deploy artifacts for builds. The tool provides binary artifact version control and Container registry. This is crucial for software development as it allows for better quality software by avoiding outdated artifacts with known issues.
  - **Benefits:** Separate binary control from source control to avoid external access to the source control system.
    Improved build stability by reducing reliance on external repositories.
  - **Inputs:** Artifacts
  - **Outputs:** Version controlled artifacts
- **Static Application Security Test (SAST) Tool**
  - This tool is essential for quickly detecting security issues more advanced than the code scan in the coding phase, especially in the development phase. Therefore, the proper tuning of these tools is essential in improving their ability to make true positives. SAST analyzes application static codes such as source code, byte code, binary code while they are in a nonrunning state to detect the conditions that indicate code weaknesses. This tool is needed after code is built, compiled, and error-free to ensure the security of the code.

    - **Benefits:** Catch code weaknesses at an early stage and Continuous assessment during development
    - **Inputs:** Source code and known vulnerabilities and weaknesses
    - **Outputs:** Static code scan report and recommended mitigation.
- **Dependency checking /Bill of Materials (BOM) checking Tool.**
    - This tool is used after the SAST tool to compare with publicly disclosed vulnerabilities. The tool Identifies vulnerabilities in the dependent components based on publicly disclosed open-source vulnerabilities. A manager reviews this report.
    - **Benefits:** Secure the overall application and Manage the supply chain risk
    - **Inputs:** Dependency list or BOM list
    - **Outputs:** Vulnerability report

Build Phase Activities

Based on the tools listed above, the following activities occur in the build phase

- **Build**
    - The code is compiled and linked. Also, the code is checked for errors and bugs and prepared for testing.
    - **Inputs:** Source code
    - **Outputs**: Binary artifacts
    - **Tool Dependencies:** Build tool, Lint tool, and Artifact repository.
- **Static application security test and scan**
    - Perform SAST to the software system. This identifies the various areas which are directly linked to system vulnerabilities.
    - **Inputs:** Source code and known vulnerabilities and weaknesses
    - **Outputs:** Static code scan report and recommended mitigation
    - **Tool Dependencies:** SAST tool
- **Dependency vulnerability checking**
    - Identify vulnerabilities in the open-source dependent components. The system is made less vulnerable as the problems which are identified are quickly addressed with the need to ensure that the system works effectively. The building phase is, therefore, carried out effectively to ensure that the code is in right condition and is of the highest quality possible.
    - **Inputs:** Dependency list or BOM list
    - **Outputs:** Vulnerability report
    - **Tool Dependencies:** Dependency checking / BOM checking tool
- **Containerize**
    - This is only needed if an application that uses a container is used for creating the code. This packages all required components OS and developed code into a hardened container
    - **Inputs:** Container base image and Container build file
    - **Outputs:** Container image
    - **Tool Dependencies:** Container builder
- **Release packaging**
    - The product is prepared for deployment and release. All binary artifacts, container or VM images, infrastructure configuration scripts, proper test

scripts, documentation, checksum, digital signatures, and release notes are packaged and used in the release process. Both the software developer and engineer do this phase. They ensure that the application is delivered to the repository.
  - **Inputs:** Binary artifacts, Scripts, Documentation, and Release notes
  - **Outputs:** Released package with checksum and digital signature
  - **Tool Dependencies:** Release packaging tool

- **Store artifacts**
  - This stores artifacts in the artifact repository. The repository serves as a store for all the different software that are developed. It is from the repository that the necessary software can be selected for deployment
  - **Inputs:** Binary artifacts, Database artifacts, Scripts, Documentation and Container images
  - **Outputs:** Versioned controlled artifacts
  - **Tool Dependencies:** Artifact Repository
- **Build Configuration, Control, and Audit**
  - This is the last activity in the build phase. This includes tracking build results, SAST and dependency checking report, generating action items, and making a go/no-go decision to the next phase. The step is important in showing the authenticity of the software. It is vital in safeguarding the identity of the software in a way that would make other people to easily identify with it.
  - **Inputs:** Build results, SAST report, and Dependency checking report
  - **Outputs:** Version controlled build report; Action items; Go/no-go decision
  - **Tool Dependencies:** Team collaboration system; Issue tracking system; CI/CD orchestrator

DevSecOps Testing

Continuous testing is critical to a successful CI/CD pipeline in order to quickly generate quality, secure code. Testing happens across all the phases of the DevSecOps pipeline and can involve any number of in-house (developers, engineers, test engineers, etc.), contractors, cloud providers, applications, etc... Therefore, it is critically important to "plan to be secure" with robust, standardized testing frameworks to include: the major testing milestones, testing responsibilities (by team, individual, owners, etc.) interface requirements (IDE plug-ins, APIs, and hooks for testing), automation requirements (necessary tools and governing compliance standards), access to and creation of the necessary Development and Testing environments, and an overall plan for test data. Additionally, requirements documents should mandate the necessary security test plans, test cases, reporting mechanisms, and an agreed upon formatting for reports, monitoring, and communications with user stories including security and testing components. Most importantly, the requirements documents should require that code is testable and conformable to best practices for reporting and error checking.

Testing Objectives:

Continually test code during all phases of the development pipeline to ensure quality, secure code that is delivered quickly and reliably. Testing should not be a bottleneck or otherwise hinder the DevOps pipeline and instead should facilitate the automated processes of developing applications. Test results should be preserved for compliance and review purposes and test auditing capabilities should be built into the pipeline and orchestration application. Further, defined processes at each testing phase should outline the failure cases and the remediation work flows to follow for failed tests. Lastly, testing does not end at deployment and should be an evergreen part of the software lifecycle to ensure code remains stable and secure.

During the coding/development phase, developers will run tests against their code while it is being written, edited, and accessed in the IDE using Static Code Analysis Tools. Additionally, developers should create a suite of pre-commit tests that run against the code before it is committed to a source code repository.

Pre-Commit Checks: (Locates and fixes common developer coding requirements, bugs, and security issues before they are committed to the source code)

- Establish a pre-determined set of security and code checks that run each time code is prepared for submittal, these checks are automated via the use of pre-commit hooks. The established rule set of pre-commit hooks and tests should be maintained and updated to address the latest threats and code requirements.
- Unit Testing: The developer's coding artifacts (APIs, methods, classes, libraries, etc.) all need to be tested for their individual functionality and security before they are integrated into the parent build product. A good practice here is to self-write a list of generic test cases that have been tested against secure coding standards and verified with their own static and dynamic tests. The test cases suite can include positive/negative indication requirements for:
    - Identity, Authentication and Access Control
    - Input Sanitization, Validation, and Encoding
    - Encryption Requirements

    - User and Session Management
    - Error and Exception Handling
    - Audits and logging events
- Pre-commit checks should trigger:
    - Automatic manual review of large code updates or significant API changes
    - Security and threat model reviews for new/updated APIs
    - Additional security testing for found vulnerabilities
    - Communications to application security teams and/or software security group to notify them about important code changes that are going to be checked in.

When code is committed to the repository, it should be subject to additional layers of automatic tests that build on the earlier tests performed in the IDE. Build-time tests should be fast, incremental checks that provide immediate feedback to developers on any code problems or security issues.

Commit-time Checks: (triggers when there is a check-in to a code repository)

- Create and execute basic, automatic testing to provide fast feedback to developers. Ensures code is compatible and buildable, bringing awareness to critical and high-security issues.
- Additional SAST testing rule sets, top 3 vulnerabilities (SQL Inject, CSS, etc.)
- Break the build and send alerts on critical and high security issues.

When there is a successful commit to the repository, and/or the build process is initiated, more in-depth tests are run against the code.

Build-Time Checks: (automatically triggered by completed commit-time checks)

- Performs automated, advanced testing with a more thorough level of SAST rule sets. For example, using the OWASP top 10 vulnerabilities or industry-specific threat lists.
- Software Composition Analysis (SCA): Set of testing tools for open-source code modules. Since most of the code relies on established banks of source code, it is important that we are continually scanning these modules for new and emerging threats. SCA tools inventory all open-source components, their licensing information, and any applicable security issues, mapping out code dependencies and tracking their vulnerabilities.
- Risk-based security testing: specific risk tests based on the risk profile of the environment.
- Break the build:
    - If the code is unable compile
    - Unit test failures
    - SAST errors
    - Large quantity of issues
    - Vulnerabilities are identified

For the actual testing phase, create a testing environment that is as identical to the production environment as possible. Using this near-peer configuration, QA testers will deploy testing modules and subject them to several tests to verify coding requirements and security compliance. Test-time checks key on well-defined procedures and routines for application and security requirements. These tests allow development teams to resolve critical and high-risk issues as soon as they're introduced into the pipeline. Additionally, these test-time checks allow QA security teams more insight into the applications, allowing for more in-depth analysis.

- Using the latest approved build from the build repository, deploy it to the test environment. Tests include: functional, integration, performance, deep SAST, DAST, IAST, and penetration testing are performed in the testing environment.
- All critical vulnerabilities identified during your SAST, DAST, and IAST testing activities should break the build, generate metrics, and immediately create a defect in your bug tracking system.

Testing does not end after the application is released to deployment. It is critical that Operations security teams continue to test applications for vulnerabilities and code weakness. Best practice is to establish a routine, scheduled set of tests that are kept up to date with the latest vulnerabilities. Additionally, third-party providers may be brought in to run vulnerability and penetration testing of the application.

Post-Deployment Testing:

- Testing post-deployment provides assurance that modifications to the production code environment haven't created new security issues. Best practice is to define a testing strategy that involves periodic security testing.
- Deploy-time checks can identify bugs and vulnerabilities that may have gone undetected through pre-production testing actions. Continuous monitoring allows us visibility into the various types of traffic a given application is receiving. Additionally, collecting application security data helps identify patterns of malicious attacks.
- Deployment checks:
    - Automate collecting application-level security metrics during continuous monitoring
    - Schedule security scanning
    - Perform vulnerability scanning
    - Assist in bug bounty scanning programs
    - Assist with creating an incident response plan
    - Help create a threat intelligence program

Testing Tools and Activities

Various tools and activities aid in the creation and implementation of a testing framework to ensure all code is as secure and error free as possible. Once a suite of testing tools and procedures is established, the testing should be standardized and applied to all layers of the pipeline to ensure consistent adherence to testing best practices.

Testing Tools

- **Test Development Tools**
  - **Features:** Aids in creating testing scenarios, scripts, and input data. Specific tools vary based on the testing activity and language, but all assist in generating the components required in the testing framework.
  - **Benefits:** Increased testing speed and automation.
  - **Inputs:** Parent testing plan.
  - **Outputs:** Various testing scenarios, data, and scripts.

- **Test Data Generator**
  - **Features:** Creates test data that will be used during the actual application testing.
  - **Benefits:** Higher testing precision and consistency.
  - **Inputs:** Test cases, parameters, and scenarios.
  - **Outputs:** Input data for future testing procedures.

- **Testing Tools Suite**
  - **Features:** Toolbox of tests to perform unit testing, interface tests, system and performance tests, integration tests, and user acceptance tests. These tools range from in-house developed toolsets to third-party applications that can be tailored to the language and environment and can be integrated into the pipeline orchestration.
  - **Benefits:** Consistency and increased testing speed and automation.
  - **Inputs:** Test cases, scenarios, and test data.
  - **Outputs:** Reports and test results.

- **SAST (Static Application Security Testing) Tools**
  - **Features:** Static analysis of code at rest in the IDE and code repositories to verify code integrity and security. Additional SAST tools include "linting" tools which are language-specific tools to check for stylistic, syntax, and code errors. SAST tools include: GitLab, Coverity, CodeSight. Linting tools include: StandardJS, Gosec, StyleCop, PHPMD
  - **Benefits:** Find and report on a variety bugs, vulnerabilities, and coding errors.
  - **Inputs:** Code and code repositories
  - **Outputs:** Reports, test results, and suggested remediations

- **SCA (Software Composition Analysis) Tools**
  - **Features:** Set of testing tools for open-source code modules. Since the majority of code relies on established banks of source code, it is important that we are

continually scanning these modules for new and emerging threats. SCA tools inventory all open-source components, their licensing information, and any applicable security issues, mapping out code dependencies and tracking their vulnerabilities. SCA Tools include: GitLab and Black Duck.
    - **Benefits:** Identify and track all open-source and third-party code modules and continuously monitor them for threats and vulnerabilities.
    - **Inputs:** Code repositories
    - **Outputs:** Inventory of code modules, reports on known vulnerabilities and remediation options for found issues

- **DAST (Dynamic Application Security Testing) Tools**
    - **Features:** Tools to examine an application while it is running to identify runtime errors, vulnerabilities, and code environment problems. Unlike SAST testing, DAST tools take a dynamic approach to testing by inputting values and data into the executed application to check for any weakness. Additional DAST tests, known as "fuzz" testing randomly inputs data values into the application to see if it will fail in an unsafe manner. DAST tools include: GitLab, OWASP Zap, HCL AppScan, and Netsparker.
    - **Benefits:** Helps identify coding weaknesses and vulnerabilities in real-time allowing for faster remediation of high-risk and critical security issues.
    - **Inputs:** Run-time application and input data
    - **Outputs:** Reports, test results, and suggested remediations

- **IAST (Interactive Application Security Testing) Tools**
    - **Features:** Advanced, integrated testing suite that includes SAST (static) and DAST (dynamic) functions to provide interactive, holistic application security testing to provide extended visibility into problem code. IAST takes a hybrid approach to application security testing that can be automated or manually executed.
    - **Benefits:** Highly accurate security testing that focuses on the problem code.
    - **Inputs:** Run-time application and input data
    - **Outputs:** Reports, test results, and suggested remediations
- **Database Security and Testing Tools**
    - **Features:** Tests application databases for the most common vulnerabilities and weak security practices. These tools probe database structures and instances for injection attacks, weak passwords, data sanitation, data access, and denial of service capabilities.
    - **Benefits:** Further reduction of security risks and attack surfaces
    - **Inputs:** Testing data and test scenarios
    - **Outputs:** Reports, test results, and suggested remediations

Testing Activities

Testing activities occur throughout the CI/CD pipeline and all are instrumental for rapid, clean, secure delivery of quality applications.

- **Unit Testing**
    - **Description:** During coding and development, the developer's coding artifacts (APIs, methods, classes, libraries, etc.) all need to be tested for their individual functionality and security before they are integrated into the parent build product. A good practice here is to self-write a list of generic test cases that have been tested against secure coding standards and verified with their own static and dynamic tests. The test cases suite can include positive/negative indicator requirements for: Identity, Authentication and Access Control, Input Validation and Encoding, Encryption, User/Session Management, Error and Exception Handling. Unit tests are usually language-specific and include: Junit, NUnit, Emma, ad Test NG
    - **Inputs:** Test scripts and a unit to test (method, function, interface) along with the expected output results.
    - **Outputs:** Testing report that indicates if the unit functions as designed
    - **Tool Dependencies:** Test Tool Suite

- **Integration Testing**
    - **Description:** Unlike unit testing, where the components are tested in isolation, integration testing puts the various tested pieces together to make sure that the code plays well with others and works in concert without interfering or breaking other components. Integration testing typically takes place in the testing phase in a near-peer production testing environment.
    - **Inputs:** Test scripts, the subject units, test data, and expected output results.
    - **Outputs:** Testing report that indicates if the integrated sub-units performed together as expected.
    - **Tool Dependencies:** Test Tool Suite

- **Functional Testing**
    - **Description:** Functional testing verifies that the delivered code satisfies the requirements laid out in the build request. Functional testing can be automated into the CI/CD pipeline using tools like Selenium and TestNG by providing appropriate input values and then comparing the output to predetermined functional requirements. Functional tests can also be a manual process where testers can verify the form and function of the sprint's deliverables. Functional testing typically takes place in the testing phase in a near-peer testing environment.
    - **Inputs:** Test scripts, application, test data, and expected output results.
    - **Outputs:** Testing report that indicates if the application meets the functional requirements
    - **Tool Dependencies:** Test Tool Suite

- **Regression Testing**
    - **Description:** Functional tests are aggregated into regression testing for the entire application, ensuring that all functions work together and that all modules, new and old, operate together as expected. Regression testing typically takes place in a pre-production staging environment prior to application release.
    - **Inputs:** Functional and non-functional regression test cases, application
    - **Outputs:** Testing report that indicates if the application passes the regression testing
    - **Tool Dependencies:** Test Tool Suite

- **Acceptance Testing**
    - **Description:** Ensures the overall acceptance of the application that it meets the requirements and satisfies the functions laid out in the requirements document. Acceptance testing accounts for overall usability, interoperability, compatibility, supportability and any maintenance issues.
    - **Inputs:** Complete test system, test data
    - **Outputs:** Final Test Report
    - **Tool Dependencies:** Test Tool Suite, various tools

- **Performance Testing (Smoke Testing)**
    - **Description:** Ensures the application will perform as expected under real-world loads and conditions. Tests here will mainly focus on latency, reliability, resource use, and scaling operations. Performance testing typically takes place in a pre-production staging environment prior to application release.
        - Load Testing: Load testing subjects the code to heavy usage and monitors the capability to withstand high volumes of input. Load testing ensures the code can operate under real-world usage levels without breaking or operating in an unintended fashion. Load testing also aids in the sizing of required system resources by monitoring application behavior in real time.
        - Stress Testing: Stress testing pushes the code to its limits to determine the breaking point for the module.
    - **Inputs:** Test case, application, test data, and expected performance results.
    - **Outputs:** System performance metrics.
    - **Tool Dependencies:** Test Tool Suite

- **SAST/DAST/IAST Testing**
    - **Description:** Perform the SAST/DAST/IAST testing against the application at various points during the pipeline. SAST testing can occur in the IDE via plugins and hooks and during later commit and build-time checks. DAST and IAST testing typically happen during the defined testing phase in a near-peer production testing environment.
    - **Inputs:** Code repositories, running application, test input values.

    - **Outputs:** Vulnerability and code check report with recommended mitigation strategies.
    - **Tool Dependencies:** SAST/DAST/IAST tools

- **Manual Security Testing**
    - **Description:** Application is subjected to manual tests for security and vulnerability issues. Commonly, penetration testing and vulnerability scans are used to ensure the application does not contain any known, common security weaknesses.
    - **Inputs:** Run-time application
    - **Outputs:** Vulnerability report and suggested remediation actions
    - **Tool Dependencies:** Various tools, scripts, and pen-test libraries

- **System Testing Audit**
    - **Description:** It is important to track which tests were performed, who performed them, when they were performed, and the results of all testing. Testing audits should encompass all code, acceptance, functional, and security tests.
    - **Inputs:** Testing activity, reports, and results
    - **Outputs:** Audit log
    - **Tool Dependencies:** Various

DevSecOps Release

Release phase is one of the main milestones in the DevSecOps pipeline that falls between the TEST and DEPLOYMENT phases. This is the point at which we say that the Build is ready for deployment into the production environment. By this stage, it can be made clear that, the code or software has passed some series of automated and manual tests and the operations team can be made confident on arising breaking issues are unlikely. Most of the organizations call this phase as a pre-release phase that occurs just before the actual deployment.

As we concentrate on the security viewpoint with respect to DevSecOps, we need to make sure that the package (built and tested components) from Testing phase contains the versions or features as expected and nothing else. Here, once we make sure that there is no tampering happened during the process, then we sign the artifacts as verified. Later, these artifacts are pushed to the repository where they need to be stored and can be fetched to the production. The basic security of the product here depends on the security of the repository and containers used. The keys used for signing these artifacts are to be managed properly and these artifacts are stored in a secure registry like Docker Hub, Azure Container, etc. With the Pipeline security at release phase, it helps in mitigating the build integrity tampering.

People/Roles:

In some cases, there will be a Release Manager who will be administering the work in the release phase. But in most cases, in DevSecOps and DevOps, the entire team is responsible for the functioning of this phase and its deliverables. People comprise of Software Developers and Security Team (who perform penetration testing, Vulnerability Assessment, DAST, RASP, Container security, Artifact signing) are involved in managing this work.

Objectives:

The main goal of this phase is to ensure that the software/product has successfully passed all the test cases, met the requirements as planned in the planning phase, managing the version control and configuration management, log each and every activity that is being performed, ensure audit requirements are met, perform penetration testing, vulnerability scanning, DAST, and RASP, and to identify the conditions under which these need to be performed, manage a switch to roll back the version of the software/product that is being released into deployment, ensure container security, and finally to release the deliverable into deployment phase. Some of these important activities and tools to perform these activities are mentioned below.

Release Phase Tools and Activities

As discussed in the above section, the main activities of release phase include maintaining, pre-delivery testing, signing, release decisions, delivering, and replication of artifacts that are stored in a secured container to the deployment/production stage. Below are some of the important and mandatory activities that are to be performed when the product/software reaches the release phase.

<u>Release Phase Tools</u>

To perform the above-mentioned activities, there is a need for tools and these tools are chosen based on the requirements and how they are planned in the planning phase. The entire release

package is considered as a tool that contains everything which involved in the release phase. And other tools are related mostly to the testing of application in its running state.
The choice of tool completely depends on the customer requirements and how it is planned in the planning phase. In case if the customer changes the scope of project/deliverable, then again, the choice of tool is planned based on the new scope, and is used in the release phase. If the security is given utmost importance, then everything mentioned below are performed. As DAST and RASP are generally performed in Testing phase, they are also performed in the release phase. But based on the type of product and customer needs, they may or may not be performed again in the release phase. However, Vulnerability assessment and Penetration testing (Using DAST tools) are always performed in many cases. Features, Benefits, in-particular tools, Inputs, and outputs are explained in detail in the below points.

- **Release Packaging Tool**
    - **Features:** The features of release package include Containers, Virtual Machine images, Package binary artifacts, Proper test and configuration scripts, documentation, This also has features like generating checksum and digital signature for the package. Here the package prepared is for a specific installer or for a self-extracting installer itself.
    - **Benefits:** The entire release package itself is a benefit, it comes with a bundle of artifacts, self-extracting software installer, tar files, etc.
    - **Inputs:** Main inputs include, Base containers, Artifacts, VM images, documentation, script for infrastructure configuration, and some release notes
    - **Outputs:** As checksum and digital signature are to be generated, they are the outputs in this case.

- **Release Phase vulnerability Scan**
    - **Features:**
        - Composition Analysis tools check versions of open-source libraries to assess open-source risk, both with security vulnerabilities and potential licensing issues. Things like Heartbleed bugs, misconfigured databases, and Struts vulnerabilities may not be part of the application testing at all, but they all critical application stack vulnerabilities.
        - Some people equate vulnerability testing with DAST, but there are other ways to identify vulnerabilities. In fact, there are several kinds of vulnerability scans; some look settings like platform configuration, patch levels or application composition to detect known vulnerabilities.
    - **Benefits:** This broadens the scans to include the application, application stack, and the open-source platforms that support it. The vulnerability Assessment automatically identifies known issues in the product/application and these issues are in turn used/supposed for penetration testing.
    - **Tools:** Some important vulnerability scanning tools include Open VAS, Nmap, Nuclei.

    - **Inputs:** Results from the testing phase, Security test and scan reports from both the test and release phase, Design documentation and the artifacts.
    - **Outputs:** After the vulnerability scan is made, it gives a detailed report of the scan and lets the team know if there are any issues found and if there needs something to be changed.

- **Release Phase DAST and RASP**
  Although these are generally performed in the testing phase, most of the organizations also implement them at the Release phase. But in particular, here DAST tools are used for Penetration Testing and Vulnerability Scanning.

  - DAST
    - **Features:**
      - DAST technologies here are basically designed to detect conditions indicative of a security vulnerability in an application in its running state. Most DAST solutions test only the exposed HTTP, HTML, REST, and SOAP interfaces of Web-enabled applications.
      - However, some solutions are designed specifically for non-Web protocol and data malformation such as remote procedure calls and call backs.
    - **Benefits:** DAST techniques can simulate and detect problems associated with authentication, authorization, client-side attacks, inappropriate command execution, SQL injection, erroneous information disclosure, interfaces, and API endpoints.
    - **Tools:** Nmap, Metasploit/Rapid7, BurpSuite, Nikto2, Arachni, the OWASP ZAProxy, SQLMap, Lynis, and Accunetix.
    - **Inputs:** Main inputs include, Base containers, Artifacts, VM images, documentation, script for infrastructure configuration, and some release notes
    - **Outputs:** As checksum and digital signature are to be generated, they are the outputs in this case.

  - RASP
    - **Features:**
      - **RASP** on the other hand can inspect the application behavior as well as the surrounding context. It captures all requests to ensure they are secure and then handles request validation inside the application.
      - While RASP and IAST (Interactive Application Security Testing) have similar methods and use, RASP does not conduct comprehensive scans but instead runs as a part of the application inspecting its traffic and activity. They both report on attacks as they occur, but IAST does so at the time of testing (uses software instrumentation to assess how an application performs and detect vulnerabilities), while RASP does so in pre-release (in a container or VM) and in production.

- **Benefits:**
  - RASP can raise an alarm in diagnostic mode and prevent an attack in protection mode, which is done by either stopping the execution of a certain operation or terminating the session.
  - RASP adds an extra layer of security/protection after the application has been set in motion, usually in the production. But this is sometimes performed as a pre-release test after the actual testing is performed and when the product is imported into a container or Virtual Machine.
- **Tools:** Nmap, Metasploit/Rapid7, BurpSuite, Nikto2, Arachni, the OWASP ZAProxy, SQLMap, Lynis, and Accunetix.
- **Inputs:** Main inputs include, Base containers, Artifacts, VM images, documentation, script for infrastructure configuration, and some release notes
- **Outputs:** As checksum and digital signature are to be generated, they are the outputs in this case.

<u>Release Phase Activities</u>

- **Managing Gates**
  - **Description:**
    - In DevSecOps, the release pipeline enables the team for continuous deployment of application or the software into the deployment stage. As this process is completely made automated, the gates involved in this phase will decide whether to release the application/software into the deployment stage using some jobs and tasks designed for this purpose. This helps in controlling the flow of features rapidly into the deployment in the continuous deployment environment.
    - The gates here are generally configured before an application entering the phase and after the phase does it's work. These gates can be configured for micro-services or the whole application depending on the type of application being developed, it's requirements, and how it is planned in the planning phase.
    - Some of the scenarios to consider for gates are: When dealing with incidents and issues management, when seeking approvals from outside the pipeline, when performing security scans on the artifacts, quality validation, change management, and when monitoring infrastructure management. All the gates are to be executed in a particular order. In pre-deployment, or also the release node
  - **Inputs:** Entire Artifacts, or the portion of modified artifacts, Security test and scan reports, designed jobs and tasks for gates, details about change management, and the order in which the gates need to perform.
  - **Outputs:** Output from the gates include the final test reports, outcome of incident, issues management, approvals, security scan results, Quality Validation, and whether the condition set for the gate has passed or not. If it is passed, the application moves into the deployment stage, if not, the DevSecOps team verifies the problem and a shift left happens in the pipeline addressing the issues.

    - **Example:** In the Microsoft Azure environment, these are the default gates present: Invoking an Azure function, Query monitor alerts, Invoking the REST API, Query work items, and for security and compliance. However, different kind of gates can be created as per the project requirements.

- **Release Decision**
    - **Description:** Here the decision is made whether to release the artifacts to the artifact repository for the deployment/production environment stage. This decision mostly depends on the output of the gates used, and this is also a part of configuration audit.
    - **Inputs:** Main Test reports generated from the testing phase, Security test and scan reports from both the test and release phase, Design documentation and the actual artifacts.
    - **Outputs:** If a decision is made to release the artifacts, then they are tagged with the release tag. Else, if the decision made is not to release, they the artifacts will not be tagged to release into the next phase i.e. deployment.

- **Delivering the released artifacts**
    - **Description:** After the release go decision is made, the artifacts are pushed to the artifact repository.
    - **Inputs:** It’s the entire release package
    - **Outputs:** In the artifact repository, a new release is observed or made.

- **Replication of Artifacts**
    - **Description:** In general, If there are regional artifact repositories, then the newly released artifacts need to be replicated in all of them.
    - **Inputs:** The Artifacts.
    - **Outputs:** In all the regional artifact repositories, the newly released artifact is replicated.

DevSecOps Release - Feedback Loop

To Planning:

By the time an application/software reaches the release phase and customer wants to change/add/remove, some or major part of the requirements, then it should be feedbacked in to the planning phase for more discussing on it.

During the release activities, if the application/software doesn't seem to meet all the criteria of the customer, it is feedbacked into the planning phase to discuss on the uncertainty and a new release is planned in the planning phase.

When a patch is to be made to the application/software when it is in the release phase, then details about the patch and other release needs to be planned in the planning phase. This may happen when a new vulnerability is found during the penetration testing.

If there is any possibility of an attack occurrence observed during the penetration testing, then a discussion and plan is to be implemented on how and when a new release should happen.

To Coding:

When some issues are observed in the code while performing a penetration test or the RASP testing, then those changes in the code need to be made by the developers, due to this, it should be feedbacked in to the coding phase.

To add a patch to some newly discovered vulnerability, developers should be made aware of it and changes to the code is to be made.

If any of the planned requirements that did not meet the customer are observed in the release phase, then it should be taken care and changes to the code is made.

To Building:

Build is a stage where an application is compiled, pull requests are verified, and merged. When the application is in the loop and if there is a new update from the customer that says some changes are to be switched back or completely removed, then the merged pull requests are to be deleted in the build phase or can be managed through the version control in the release phase.

When performing the release phase activities like deploying the application in a secure container, proper API and code files are to be used in correct file locations, if there is any directory issue, then it should be adjusted in the build phase.

If any errors in the file locations or if there is any code file that exposes the data which shouldn't be exposed, then some corrections are to be made in the build phase and then they are properly adjusted.

To Testing:

As testing is involved in each phase, it also does in the release phase. When the application is in the release stage, and if the customer wants to check a new type of test on the application/software, then it is feedbacked into the testing phase.

Release Phase also performs the DAST and RASP testing as performed in the Testing phase, but here the these tools are used to perform the penetration testing. If there is any new third-party tool, container, artifact repository that needs to be used in the release phase, then it should first be tested in the Testing phase.

When there are any discrepancies in the application integrity or on the bult and tested components document or security test scan reports that are passed from testing phase to the release phase, then they are to be feedbacked into the testing phase using a shift left strategy.

DevSecOps Deployment

After the code has been written, tested, and a release is ready, it is time for the update to be deployed. Deployment is when the update is moved from an artifact repository to a server. This should be done using automated deployment software such as Jenkins. There are many options, and one should be selected that best suits the business. Once an update reaches the deployment phase, it can be deployed to either the test server or the production server. It should go to the test server first. Once the software has been deployed on the test server, it will go back into the testing phase until it passes all tests. When it reaches the deployment phase for the second time, it will be put onto the production server.

People/Roles:

This phase should be designed to be as automated as possible. Developers will need to create infrastructure as code configurations to be used in this phase. If a continuous delivery method is chosen, the project manager will need to decide when to deploy each build. Continuous delivery will be explained below.

Objectives:

The main objective of this phase is to take artifacts and infrastructure as code configurations and use them to create a working environment on the server. The infrastructure and the application must be secure. There must be a method of deployments should cause zero downtime. This phase should be set up to get deployments to customers as frequently as possible. The tools and activities needed to accomplish these goals are listed below.

Deployment Tools and Activities

Deployment Phase Tools

- **Virtualization Manager**
  - **Features:** Manages VM instances and resources.
  - **Benefits:** Takes care of the VM installation, scaling, and monitoring all in one place.
  - **Inputs:** Configuration and specification for a VM
  - **Outputs:** Usable VM

- **Container Manager**
  - **Features:** Manages container instances and resources.
  - **Benefits:** Takes care of the container installation, scaling, and monitoring all in one place.
  - **Inputs:** Configuration and specification for a container
  - **Outputs:** Usable container

- **Encryption Tool**
  - **Features:** Encrypts data in transit and at rest.

    - **Benefits:** Provides privacy and security
    - **Inputs:** data
    - **Outputs:** encrypted data

- **Infrastructure as Code Automation Tools**
    - **Features:** Uses configuration file to automatically set up the environment. This includes the infrastructure and the security policy.
    - **Benefits:** Provides automation and the ability to easily scale
    - **Inputs:** IaC configurations that were created in the build phase
    - **Output:** Properly configured infrastructure

- **Service Mesh**
    - **Features:** It controls traffic, communication, and authentication between services.
    - **Benefits:** Allows for microservice to communicate with each other.
    - **Inputs:** Policies for communication and authentication
    - **Output:** Communication between microservices

<u>Deployment Phase Activities</u>

- **Deployments should be automated.**
    - **Description:** Automated deployment saves developer's time and allows for more streamlined releases.
    - **Inputs:** - Artifacts from release phase.
    - **Outputs:** Infrastructure and application on server.
    - **Tool Dependencies:** The deployment should be managed by a standalone automated deployment software or by an orchestrator software such as Jenkins.

- **The team needs to choose between a continuous delivery method and a continuous deployment method.**
    - **Description:** A continuous delivery method allows the team to decide when the build is moved to production. Everything up until that point is automated, and the build stays in the staging area until it is released by the project manager. If a continuous delivery method is chosen, a release schedule needs to be maintained.
      A continuous deployment method pushes the build to production automatically when it has passed all tests. Continuous deployment is faster and more efficient, but it also requires a more mature DevSecOps program to keep up.
    - **Inputs:** Deployment method
    - **Outputs:** Release plan
    - **Tool Dependencies:** An orchestrator software such as Jenkins

- **A zero-downtime deployment method should be used.**
    - **Description:** Because of the frequent updates in DevSecOps, it is important to use a zero-downtime deployment method.

Canary deployment uses feature flags in the source code to turn on the update for a small set of users. If everything goes well, they keep adding users until everyone has the update. If something goes wrong, the feature flags are turned back off. The update would then go back to planning to begin to develop a fix for the problem. Blue-Green deployment has two identical servers. One of them gets the update. The load balancer then sends all the traffic to the updated server. If something goes wrong, all traffic can be immediately switched to the server with the last known good configuration. The update would then go back to planning to begin to develop a fix for the problem. These problems could be caused by a faulty test or by a unplanned situation that was not tested for.

  - **Inputs:** - Project requirements, DevSecOps maturity
  - **Outputs:** Zero-deployment plan
  - **Tool Dependencies:** Feature flags, load balancer

- **Infrastructure as code should be automated.**
  - **Description:** Configuration files need to be set up to allow for the creation and configuration of resources automatically as needed.
  - **Inputs:** Configuration files and requirements
  - **Outputs:** Configured resources as needed
  - **Tool Dependencies:** Most major cloud service providers have their own IaC tool built in. There are also stand-alone tools such as Puppet. Depending on if the project uses virtual machines or containers, a virtualization manager or a container manager will be needed for scaling.

- **All releases must first be deployed on the test server.**
  - **Description:** IaC sets up the resources on the test server. This allows for the configurations and hardening of the resources to be tested.
  - **Inputs:** - Artifacts from release phase
  - **Outputs:** Infrastructure and application on test server
  - **Tool Dependencies:** IaC and automated deployment software

- **System and Performances tests must be run on all releases.**
  - **Description:** System and Performance test must be run to ensure that the build is working as intended. These tests must be run on the test server and post-deployment to the production server.
  - **Inputs:** Test requirements
  - **Outputs:** Test results
  - **Tool Dependencies:** Testing suite

- **Security tests must be run on all releases.**
  - **Description:** Security tests must be run to make sure all security requirements are met. The hardening of the resources is tested here. This includes ensuring that data is encrypted in motion and at rest, that proper access control is in place, that all ports that are not needed are not open, that the system is updated, and no known

    vulnerabilities are present. The application itself and all APIs should also be tested here.
    - **Inputs:** Test requirements
    - **Outputs:** Test results
    - **Tool Dependencies:** Testing suite

- **Builds should be deployed to the production server using the method chosen.**
    - **Description:** No build should be deployed unless it has passed all required tests.
    - **Inputs:** - IaC, artifacts, and deployment method.
    - **Outputs:** System set up on deployment server.
    - **Tool Dependencies:** Orchestrator software, IaC tool

- **Logs should be automatically recorded for all actions.**
    - **Description:** Logs are necessary to check if everything is working as intended. Logs are needed for troubleshooting problems. These logs will be monitored in the operations phase.
    - **Inputs:** System actions
    - **Outputs:** Log files
    - **Tool Dependencies:** Logging and logging configurations

DevSecOps Operations

Operations management puts the ‘ops’ in DevSecOps, a way of approaching IT security having in mind that everyone is responsible for security. This phase comes in when an application has passed the production, automated processes such as real-time monitoring, host- and network-intrusion detection, and compliance validation and evidence attestation. These are used to increase efficiency and detect vulnerabilities. When defects or vulnerabilities are discovered, then resolutions are identified, prioritized, and tracked to ensure product reliability and security constantly improve.

Objectives

This is also the final phase which involves maintenance and regular required updates. This is also when end users fine-tune the system, to boost performance, add new or meet the additional user requirements. It also involves injecting security practices into an organization’s DevOps pipeline. The goal, therefore, is to incorporate security into all stages of the application development workflow.

The various phases depend on operations to get desired outcome which are planning, building, coding, deployment, and release.

<u>**Operations and Monitor Phase Activities**</u>

- **Backup and disaster recovery planning,**
    - **Description:** This involves having a backup management system that is efficient and effective. This prevents loss of data if an attack to an application is done. This system must allow access to back up data and it uses database automation tool which makes updates happen automatically and regularly.
    - **Inputs:** Access to backup system
    - **Outputs:** Backup data
    - **Capabilities Dependencies:** Database management tool

- **Scale,**
    - **Description:** In this activity there is management of containers or virtual machines as a group which change dynamically based on demand and policy. This activity depends on having containers and virtual machine as the hosting environment as the tool.
    - **Inputs:** Real time demand and virtual machine performance measures
    - **Outputs:** Optimized resource allocation
    - **Capabilities Dependencies:** VM management capability

- **Load balancing,**
    - **Description:** This activity depends on load balancing which equalizes resource utilization. It highly depends on load balance policy and balanced resource utilization. It also depends on virtual machine and container management on hosting environment.
    - **Inputs:** Load balance policy
    - **Outputs:** Balanced resource utilization

    - **Capabilities Dependencies:** VM management capability on the hosting environment

- **Log analysis and auditing:**
    - **Description:** These are records of audit logs and records of events being analyzed and correlated. The inputs in this activity are logs and outputs are the reports. It depends on log analysis and auditing tools.
    - **Inputs:** Logs
    - **Outputs:** Alerts and remediation report
    - **Capabilities Dependencies:** Log aggregator

- **System configuration monitoring**,
    - **Description**: Which ensures that the systems have been configured, comply, and can report what is expected of them. The inputs in this system are running system configuration and the outputs are warnings or alerts. They depend on operations dashboard, alert notification tool or issue tracking system tool.
    - **Inputs:** Running system configuration, configuration baseline
    - **Outputs:** Compliance report
    - **Capabilities Dependencies:** Issue tracking system

- **System security monitoring,**
    - **Description:** This ensures monitoring system hardware and baselining system performance. The inputs are running system and the outputs are alerts or warnings. This activity depends on operation monitoring as the tool.
    - **Inputs:** Running system
    - **Outputs:** Warnings and alerts
    - **Capabilities Dependencies:** Alerting and notification

- **Asset Inventory,**
    - **Description**: This activity ensures that there is a proper inventory system that tracks the assets. This activity depends on inventory management.
    - **Inputs:** IT assets
    - **Outputs:** Asset inventory
    - **Capabilities Dependencies**: Inventory Management
- **Database monitoring and security auditing**,
    - **Description:** This ensures that the database efficiently monitors the auditing activities. The tools here are database monitoring tool and issue tracking system.
    - **Inputs:** Database traffic, events
    - **Outputs:** Logs, warnings, and alerts
    - **Capabilities Dependencies:** Database monitoring tools, Database security audit tool

<u>**Operations and Monitoring Capabilities**</u>

- **Backup management system**,
    - **Features:** This tool has the data backup system components snapshot as the features which has access to the backup source as the input. The benefit of using this tool is that it improves failure recovery.

- **Benefits:** Improves failure recovery
- **Inputs:** Accesses to backup source
- **Outputs:** Backups data for virtual machine or container snapshot

- **Operations dashboard:**
  - **Features:** This tool's feature is providing operators a visual view of the operating status, alerts, and actions. It seeks to improve operations management and uses alerts and recommended actions which are displayed on the dashboard.
  - **Benefits:** Improvement of operations management
  - **Inputs:** All operational monitoring status, alerts, and actions
  - **Outputs:** Dashboard display

- **Logs:**
  - **Features:** This tool uses logging events for networks, users, applications, and data activities. The benefits of this, is that it assists with trouble shooting issues and detection of threats. The inputs of this tool are network, all user, and data applications.
  - **Benefits:** Aids in troubleshooting issues, detection of advanced threats
  - **Inputs:** All user network application
  - **Outputs:** Event logs

- **Log aggregator:**
  - **Features:** This tool filters log files for events of interest to have efficient security and control measures. The inputs are logs and the outputs are formatted logs.
  - **Benefits:** Assists security of information
  - **Inputs:** Logs
  - **Outputs:** Formatted event log

- **Log analysis and auditing:**
  - **Features:** This tool analyzes reports to detect malicious threats and gives out detailed information of results found.
  - **Benefits:** Filters log files
  - **Inputs:** Logs
  - **Outputs:** Formatted event logs

- **Operations monitoring**
  - **Features**: This tool enables dashboards to display performance and alert on performance issues. The benefits are that there is better user experience and improvement of performance activities.
  - **Benefits:** Operations improvement continuity
  - **Inputs:** Performance KPIS
  - **Outputs:** Performance statistics

- **Information security continuous monitoring:**
    - **Features:** This tool detects unauthorized personnel getting into the system and identifies cybersecurity vulnerability and verification of the effectiveness of the protective measures.
    - **Benefits:** Detects unauthorized personnel and connections
    - **Inputs:** IT asset network personnel activities
    - **Outputs:** Vulnerabilities incompliance findings

- **Database security audit tool,**
    - **Features**: This tool has performance user access and data access audit as it's feature. It aids to enhance data security and running databases.
    - **Benefits:** Improves database operations continuity
    - **Inputs:** Running database
    - **Outputs:** Logs, warnings, and alerts

<u>**Roles and Responsibilities of Operators in this phase**</u>

- **Operations engineer:** he provides technical support to the programs that have been put in place. He also performs scheduled backup and performs maintenance whenever there is downtime to make the system available to users.
- **Program analyst**: he interprets the user requirements, designs and codes for the programs that are running.
- **Project manager:** he develops, documents, and executes different plans and procedures for various activities and tasks for this phase. This provides a platform for problem reporting and customer satisfaction.
- **Configuration control board**, which encompasses a board of individuals who convene to make recommendations for changes and improvements to the product. These recommendations can be approved or denied.
- **Technical support**, this personnel provides technical support to the product being provided. This involves giving access rights to the program and maintenance of the operating system for the workstation and server.
- **Data administrator** who performs tasks that ensure accuracy and validity of data that are entered into the product. He might also create information systems database and maintain the database security and plans for disaster recovery.
- **Network system analyst**: he plans, installs, and maintains networks as needed and ensures that external communications and connectivity are available.

DevSecOps Operations - Feedback Loop

There exists a feedback loop that flows from Operations back to Planning. Data collected by Operations on how the application is being used and how the application is running show's what features are the most useful to the user, what features the user might be interested in, and the reliability of the application. The goal is to give the user the best experience with your application. It can also provide security information. Since we expose the application to the public, this feedback will provide us with the information we need to set up a secure environment. All of this is passed back to the planning stage to be discussed by the stakeholders.

Governance in DevSecOps

Definition

Our definition of Governance combines both governance and compliance from below, as:
Government and industry groups have issued many software standards to make software safe and secure for users. Software compliance refers to how well an application obeys the rules in a standard. If your application complies with software standards, it's less likely to contain bugs, security weaknesses, and design flaws. But compliance doesn't ensure quality because no software standard addresses every aspect of software quality. To achieve software compliance, you might also have to, for example, produce certain types of documentation or add security testing at more points in your software development life cycle.

Governance actively assesses and manages the risks associated with the development lifecycle.

Governance as Code moves your governance, including implementing best security practices, adhering to compliance requirements and standards, and allocating business resources, away from a manual, human-based approach to a more consistent, efficient, and highly repeatable code-based approach.

Implementation

A tool for governance should have a dashboard landing page that is customizable and summarizes data relative to our role. The type of data that should be available to our governance and compliance tool is analytics, metrics, a scheduler, deployments actions, release action, and reports. All the functionalities should be subject to auditing and control measures applied to any part of the pipeline. The analytics page would be an aggregation of all the analytic and metric data related to a pipeline. It should be easy to read and process; so that decisions can be made. It should be able not just to summarize our pipeline but all pipelines in development. A scheduler should be included to help plan, track, and manage the development process. Test information and defect data should be available to give quality of code for the release. This information should be available not just for your release but all releases in development. You should select a release and see a features list developed in that release and when it will be deployed. The release could be a release of different pipelines, which will be tracked together as a group. The software will track releases, pipelines, phases of development, and governance.

The tool will overlay on to your existing development by interacting with your existing tools and software. Governance can be injected at certain checkpoints in the pipeline through phases and gates. In between gates, you can put conditions and an approval process for those conditions. This entire process will be wholly audited. There will be a history of how each release was planned, tracked, and managed. There will be accountability for each step of the development process.

All the data for a release can be quantified, summarized, and presented for acceptance and completion to our framework stakeholder. This stakeholder could be one of the following: data controller, data processor, a member of the data protection group, or an external auditor. This stakeholder will be given the documentation and certify that we have met or failed to meet company compliance or external framework compliance.

Tools

- Plutora
    - https://www.plutora.com/blog/devsecops-guide
    - https://www.youtube.com/watch?v=q3twmbWlIbs&t=17s

- OpsMx
    - https://www.opsmx.com/governance-compliance-devsecops.html
    - https://www.youtube.com/watch?v=xqX7W1gYlx8&t=4s

Purpose

The reason why you're automating, or any team would automate governance as code is to reduce that impedance mismatch. People who are developing code in the DevSecOps world is turning code out in a very fast manner. If security is lagging or if compliance is lagging in terms of matching up with the speed of developing code. Then security will become an afterthought, and there is a chance that you will have to fix problems much later in the life cycle, which will become much more complex. You want to try to have the same amount of speed in the security process, as in the development process.

References

1. "5 Types of Software Licenses You Need to Understand: Synopsys." *Software Integrity Blog*, 13 Apr. 2021, www.synopsys.com/blogs/software-security/5-types-of-software-licenses-you-need-to-understand/.
2. Akehurst-Ryan, Callum, et al. "Continuous Testing – Creating a Testable CI/CD Pipeline." Scott Logic, 10 Feb. 2020, blog.scottlogic.com/2020/02/10/continuous-testing.html.
3. "Application Architect Job Description." *JobHero*, www.jobhero.com/job-description/examples/computer-software/application-architect.
4. Bell, Laura, et al. *Agile Application Security: Enabling Security in a Continuous Delivery Pipeline*. O'Reilly Media, 2017.
5. *Building an Enterprise DevSecOps Program*. Securosis, L.L.C., 26 Nov. 2019, cdn.securosis.com/assets/library/main/Enterprise_DevSecOps_2019_V2.FINAL_.pdf.
6. Buuri, Marko. "A Practical Framework for DevSecOps." *Medium*, Fraktal, 6 Apr. 2020, blog.fraktal.fi/practical-framework-for-devsecops-dd7fd9e63866.
7. Design by Committee 6 Feb. 2021. Wikipedia, Wikimedia Foundation, en.wikipedia.org/wiki/Design_by_committee.
8. *DoD Enterprise DevSecOps Reference Design*. Department of Defense (DoD) Chief Information Officer, 12 Aug. 2019, dodcio.defense.gov/Portals/0/Documents/DoD%20Enterprise%20DevSecOps%20Reference%20Design%20v1.0_Public%20Release.pdf.
9. Freeman, E. (2019). DevOps For Dummies. *John Wiley & Sons, Inc.*
10. Homann, Maria. "Continuous Testing in CI/CD: What, Why and How." Continuous Testing in CI/CD: What, Why and How, 9 Feb. 2021, www.leapwork.com/blog/ci-cd-continuous-testing-what-why-how?hs_amp=true.
11. Hsu, Tony. *Hands-on Security in DevOps: Ensure Continuous Security, Deployment, and Delivery with DevSecOps*. Packt Publishing, 2018.
12. "Identity Management." *Wikipedia*, Wikimedia Foundation, 14 Apr. 2021, en.wikipedia.org/wiki/Identity_management.
13. Jackson, L. (2020). The CI/CD Pipeline. In *The Complete ASP. NET Core 3 API Tutorial* (pp. 305-347). Apress, Berkeley, CA. https://link.springer.com/chapter/10.1007/978-1-4842-6255-9_12
14. JakobTheDev (2019). The Eight Phases of a DevOps Pipeline. https://medium.com/taptuit/the-eight-phases-of-a-devops-pipeline-fda53ec9bba#:~:text=Plan,roadmap%20to%20guide%20future%20development
15. Modigliani, Pete. *Defense Agile Acquisition Guide*. The Mitre Corporation, Mar. 2014, www.mitre.org/sites/default/files/publications/MITRE-Defense-Agile-Acquisition-Guide.pdf.
16. Nogueira, A. F., Ribeiro, J. C., Zenha-Rela, M. A., & Craske, A. (2018, September). Improving la redoute's ci/cd pipeline and devops processes by applying machine learning techniques. In *2018 11th international conference on the quality of information and communications technology (QUATIC)* (pp. 282-286). IEEE. https://ieeexplore.ieee.org/abstract/document/8590203/
17. *OWASP Application Security Verification Standard*, owasp.org/www-project-application-security-verification-standard/.

18. Patel, C. (2019) Secure and Scalable CI/CD Pipeline With AWS. In *Dzone* https://dzone.com/articles/secure-and-scalable-cicd-pipeline-with-aws
19. Positive Technologies. “SAST, DAST, IAST, and RASP: How to Choose?” *DAST vs SAST, IAST, and RASP: Application Security Testing Methods Guide*, Positive Technologies, 2 Dec. 2020, www.ptsecurity.com/ww-en/analytics/knowledge-base/sast-dast-iast-and-rasp-how-to-choose/.
20. Reed, M. (2020) DevOps: The Ultimate Beginners Guide to Learn DevOps Step-By-Step. *DevOps: The Ultimate Beginners Guide to Learn DevOps Step-by-Step Kindle Edition*, www.amazon.com/DevOps-Ultimate-Beginners-Step-Step-ebook/dp/B08GGF4YB3/ref=sr_1_1?crid=3BBB0CZRBX1U5&dchild=1&keywords=devops+the+ultimate+beginners+guide+to+learn+devops+step-by-step&qid=1613792980&sprefix=devsops+the+ultimate%2Caps%2C151&sr=8-1.
21. Red Hat. (2021). *DEVOPS: What is a CI/CD pipeline?* https://www.redhat.com/en/topics/devops/what-cicd-pipeline
22. “Release: Release Management Tool.” *Digital.ai*, digital.ai/release.
23. “Release Management in DevOps.” *Software Testing Help*, 18 Feb. 2021, www.softwaretestinghelp.com/release-management-in-devops.
24. Sacolick, Isaac. “DevSecOps: How to Bring Security into Agile Development and CI/CD.” *InfoWorld*, InfoWorld, 10 Feb. 2020, www.infoworld.com/article/3520969/how-to-bring-security-into-agile-development-and-cicd.html.
25. Sacolick, Isaac. “What Is CI/CD? Continuous Integration and Continuous Delivery Explained.” InfoWorld, 17 Jan. 2020, www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html. Akehurst-Ryan, Callum, et al. “Continuous Testing – Creating a Testable CI/CD Pipeline.” Scott Logic, 10 Feb. 2020, blog.scottlogic.com/2020/02/10/continuous-testing.html.
26. Sheehy, Michael. “What Exactly Is a Cloud Architect and How Do You Become One?” *Cloud Academy*, Cloud Academy, 21 Sept. 2020, cloudacademy.com/blog/what-exactly-is-a-cloud-architect-and-how-do-you-become-one/#:~:text=A%20Cloud%20Architect%20is%20responsible,and%20solutions%20in%20the%20cloud.
27. Strom, David. “What Is Application Security? A Process and Tools for Securing Software.” *CSO Online*, CSO, 2 Sept. 2020, www.csoonline.com/article/3315700/what-is-application-security-a-process-and-tools-for-securing-software.html.
28. VeritisAdmin. “CI/CD Pipeline: 15 Best Practices for Successful Test Automation.” Veritis Group Inc, 18 Sept. 2020, www.veritis.com/blog/ci-cd-pipeline-15-best-practices-for-successful-test-automation.
29. “Video: IT Release Management Using Plutora.” *Plutora*, 8 Jan. 2019, www.plutora.com/resources/videos/plutora-enterprise-release-management.
30. What is DevOps Lifecycle? | How to manage yours in *Cuelogic* https://www.cuelogic.com/blog/devops-lifecycle
31. What is DevOps? In *NetApp* https://www.netapp.com/devops-solutions/what-is-devops/
32. “What Is Release Management? (All That You Need To Know).” *Plutora*, 5 Feb. 2020, www.plutora.com/software-release-management/what-is-release-management.