2023-29DEC

DEVSECOPS SCENARIOS

WWW.DEVSECOPSGUIDES.COM

DevSecOps Scenarios

Scenario: Full Lifecycle DevSecOps Implementation

1. Planning & Code Development

- **Tools**: Jira, GitHub Copilot
- **Security Practice**: Secure coding guidelines
- **Description**: Developers use Jira for task tracking, ensuring security requirements are included from the start. GitHub Copilot assists in writing secure code by suggesting best practices.

2. Source Code Repository

- **Tools**: GitHub, GitGuardian
- **Security Practice**: Secret scanning and code security
- **Command/Code**:

```
# GitGuardian pre-receive hook
gitguardian pre-receive
```

- **Description**: Code is committed to GitHub, where GitGuardian scans for secrets and sensitive data.

3. Continuous Integration

- **Tools**: Jenkins, SonarQube, Snyk
- **Security Practice**: SAST and dependency scanning
- **Command/Code**:

```
// Jenkinsfile for SAST with SonarQube and Snyk
pipeline {
  agent any
  stages {
    stage('SAST') {
      steps {
        script {
          sh 'sonar-scanner ...'
          sh 'snyk test'
        }
      }
    }
  }
}
```

- **Description**: Jenkins integrates with SonarQube for static analysis and Snyk for vulnerability scanning in dependencies.

4. Continuous Deployment

- **Tools**: Ansible, HashiCorp Vault
- **Security Practice**: Secure deployment and secrets management
- **Command/Code**:

```
# Ansible playbook using HashiCorp Vault for secrets
- hosts: all
  vars:
```

```
    db_password: "{{ lookup('hashi_vault', 'secret=myapp/data/db password=my_vault_password') }}"
  tasks:
    - name: Deploy application
      ansible.builtin.git:
        repo: 'https://github.com/myrepo/myapp.git'
        dest: '/var/www/myapp'
```

- **Description**: Ansible automates deployment, with HashiCorp Vault managing secrets securely.

5. Infrastructure Management

- **Tools**: Terraform, Checkov
- **Security Practice**: Infrastructure as Code security
- **Command/Code**:

```
# Terraform script for secure AWS setup
resource "aws_instance" "example" {
  ami = "ami-123456"
  instance_type = "t2.micro"
  # Security group, roles, etc.
}
```

```
# Checkov for Terraform security
checkov -d .
```

- **Description**: Terraform scripts manage cloud infrastructure, with Checkov ensuring security best practices.

6. Continuous Monitoring & Logging

- **Tools**: Datadog, Splunk
- **Security Practice**: Real-time security monitoring and log analysis
- **Command/Code**:

```
# Datadog agent configuration for monitoring
logs_enabled: true
logs:
  - type: file
    path: /var/log/myapp.log
    service: myapp
    source: python
```

- **Description**: Datadog and Splunk are used for monitoring application performance and analyzing logs for security insights.

7. Incident Response & Feedback

- **Tools**: PagerDuty, TheHive
- **Security Practice**: Automated incident response and case management
- **Command/Code**:

```
# Triggering an incident in PagerDuty
curl -X POST --header 'Content-Type: application/json' \
--header 'Authorization: Token token=my_pagerduty_token' \
-d '{"incident": {"type": "incident", "title": "Security Alert", "service": {"id": "service_id",
"type": "service_reference"}}}' \
'https://api.pagerduty.com/incidents'
```

- **Description**: PagerDuty manages incident alerts, while TheHive is used for case management and investigation.

Textual System Diagram

```
[Developers] --plan & code--> [Jira & GitHub Copilot]
    |
    | (1) Secure Coding
    v
[GitHub Copilot] --commit code--> [GitHub Repository]
    |
    | (2) Secret Scanning
    v
[GitHub] --trigger CI--> [Jenkins Server]
    |
    | (3) SAST & Dependency Scanning
    v
[Jenkins] --deploy code--> [Ansible & HashiCorp Vault]
    |
    | (4) Secure Deployment & Secrets Management
    v
[Ansible] --manage infrastructure--> [Terraform & Checkov]
    |
    | (5) Infrastructure as Code Security
    v
[Terraform] --log data--> [Datadog & Splunk]
    |
    | (6) Monitoring & Logging
    v
[Datadog] --incident detection--> [PagerDuty & TheHive]
    |
    | (7) Incident Response & Case Management
    v
[PagerDuty & TheHive] --alert & manage--> [Developers/Security Team]
```

CIA Triangle for Compliance

- **Confidentiality (Rating: 9/10)**
  - **Proof**: Use of HashiCorp Vault for secrets management and GitGuardian for secret scanning ensures high confidentiality of sensitive data.
- **Integrity (Rating: 8/10)**
  - **Proof**: SonarQube and Snyk maintain code and dependency integrity. Terraform and Checkov ensure infrastructure integrity.
- **Availability (Rating: 7/10)**
  - **Proof**: Datadog and Splunk provide continuous monitoring to maintain high availability, but there's always room for improvement in disaster recovery and redundancy strategies.

Scenario: Advanced DevSecOps Workflow in a Cloud-Native Environment

1. Planning & Code Development

- **Tools**: Azure DevOps Boards, GitHub
- **Security Practice**: Agile planning with security user stories
- **Description**: Azure DevOps Boards is used for sprint planning, ensuring security tasks and user stories are included. Code is developed and managed in GitHub repositories.

2. Source Code Repository

- **Tools**: GitHub, GitLab
- **Security Practice**: Code scanning and review
- **Command/Code**:

```
# GitHub Actions for code scanning
```

```yaml
name: "Code Scanning - Action"

on:
  push:
  pull_request:
  schedule:
    - cron: '0 1 * * 0'

jobs:
  build:
    name: Build
    runs-on: ubuntu-latest

    steps:
    - name: Checkout repository
      uses: actions/checkout@v2

    - name: Run CodeQL
      uses: github/codeql-action/analyze@v1
```

- **Description**: GitHub Actions is used for automated code scanning using CodeQL to detect vulnerabilities and errors.

3. Continuous Integration

- **Tools**: Jenkins, SonarCloud
- **Security Practice**: SAST (Static Application Security Testing)
- **Command/Code**:

```
// Jenkinsfile for SAST with SonarCloud
pipeline {
  agent any
  stages {
    stage('SonarCloud Analysis') {
      steps {
        withSonarQubeEnv('SonarCloud') {
          sh 'mvn clean verify sonar:sonar'
        }
      }
    }
  }
}
```

- **Description**: Jenkins integrates with SonarCloud for static application security testing, ensuring code quality and security.

4. Continuous Deployment

- **Tools**: Azure DevOps Pipelines, Terraform Cloud
- **Security Practice**: Secure and automated deployment
- **Command/Code**:

```
# Azure DevOps pipeline for deployment
trigger:
- main

pool:
  vmImage: 'ubuntu-latest'

steps:
- script: echo Deploying Application!
```

- **Description**: Azure DevOps Pipelines automate the deployment process, with Terraform Cloud managing the infrastructure provisioning securely.

5. Infrastructure Management

- **Tools**: Terraform, Pulumi
- **Security Practice**: Infrastructure as Code (IaC) security
- **Command/Code**:

```
// Pulumi script for AWS infrastructure
import * as pulumi from "@pulumi/pulumi";
import * as aws from "@pulumi/aws";

const bucket = new aws.s3.Bucket("myBucket", {
    acl: "private",
});
```

- **Description**: Pulumi and Terraform are used for infrastructure management, ensuring best practices and security in cloud infrastructure provisioning.

6. Continuous Monitoring & Logging

- **Tools**: Datadog, New Relic
- **Security Practice**: Real-time monitoring and logging
- **Command/Code**:

```
# Datadog agent configuration for log collection
logs:
  - type: file
    path: /var/log/myapp/*.log
    service: myapp
    source: python
```

- **Description**: Datadog and New Relic provide comprehensive monitoring and logging capabilities, offering insights into application performance and security.

7. Incident Response & Feedback

- **Tools**: Opsgenie, Jira Service Management
- **Security Practice**: Incident management and feedback loop
- **Command/Code**:

```
# Opsgenie alert creation
curl -X POST https://api.opsgenie.com/v2/alerts \
  -H "Content-Type: application/json" \
  -H "Authorization: GenieKey YOUR_API_KEY" \
  -d '{
        "message": "High CPU Usage detected",
        "alias": "High CPU",
        "description": "CPU usage exceeded 90%",
        "responders": [{"type": "team", "name": "Operations"}],
        "visibleTo": [{"type": "team", "name": "Development"}],
        "actions": ["Restart", "Scale"],
        "tags": ["cpu", "high_usage"],
        "details": {"monitoring_tool": "Datadog"}
      }'
```

- **Description**: Opsgenie for alerting and incident management, integrated with Jira Service Management for

Textual System Diagram

```
[Developers] --plan & code--> [Azure DevOps Boards & GitHub]
    |
    | (1) Agile Planning & Code Management
    v
[GitHub] --code scanning--> [GitHub Actions]
    |
    | (2) Automated Code Scanning
    v
[GitHub] --trigger CI--> [Jenkins & SonarCloud]
    |
    | (3) SAST & Code Quality Check
    v
[Jenkins] --deploy code--> [Azure DevOps Pipelines & Terraform Cloud]
    |
    | (4) Deployment Automation
    v
[Azure DevOps Pipelines] --manage infrastructure--> [Terraform & Pulumi]
    |
    | (5) IaC Security & Management
    v
[Terraform & Pulumi] --monitoring--> [Datadog & New Relic]
    |
    | (6) Continuous Monitoring & Logging
    v
[Datadog & New Relic] --incident detection--> [Opsgenie & Jira Service Management]
    |
    | (7) Incident Response & Management
    v
[Opsgenie & Jira] --feedback loop--> [Developers/Security Team]
```

CIA Triangle for Compliance

- **Confidentiality (Rating: 8/10)**
  - **Proof**: Use of GitHub for secure code management and HashiCorp Vault for secrets management ensures data confidentiality.
- **Integrity (Rating: 9/10)**
  - **Proof**: SonarCloud and Jenkins ensure code integrity. Terraform and Pulumi maintain infrastructure integrity.
- **Availability (Rating: 7/10)**
  - **Proof**: Continuous monitoring with Datadog and New Relic ensures high availability, but disaster recovery strategies could be further enhanced.

Scenario: Modern DevSecOps in a Microservices Architecture

1. Planning & Code Development

- **Tools**: Atlassian Confluence, JetBrains IntelliJ IDEA
- **Security Practice**: Secure design documentation and IDE-based security linting
- **Description**: Confluence is used for documenting security requirements and architecture. IntelliJ IDEA, with its security plugins, assists developers in writing secure code.

2. Source Code Repository

- **Tools**: Bitbucket, GitLeaks
- **Security Practice**: Secure source code management and secret scanning
- **Command/Code**:

```
# GitLeaks pre-push hook
gitleaks protect --staged
```

- **Description**: Bitbucket hosts the repositories, and GitLeaks scans for accidental secret commits.

3. Continuous Integration

- **Tools**: CircleCI, Veracode
- **Security Practice**: SAST and container scanning
- **Command/Code**:

```
# CircleCI config for SAST with Veracode
version: 2.1
orbs:
  veracode: veracode/veracode-scan@1.0.0
workflows:
  version: 2
  build-and-scan:
    jobs:
      - veracode/scan:
          file_path: "./target/*.jar"
```

- **Description**: CircleCI integrates with Veracode for static application security testing and container vulnerability scanning.

4. Continuous Deployment

- **Tools**: Spinnaker, HashiCorp Vault
- **Security Practice**: Secure deployment automation and secret management
- **Command/Code**:

```
// Spinnaker pipeline JSON snippet
{
  "keepWaitingPipelines": false,
  "limitConcurrent": true,
  "stages": [
    {
      "name": "Deploy to Kubernetes",
      "type": "deployManifest"
      // Additional deployment details
    }
  ]
}
```

- **Description**: Spinnaker manages deployments across environments, with HashiCorp Vault handling secrets.

5. Infrastructure Management

- **Tools**: AWS CloudFormation, Bridgecrew
- **Security Practice**: Infrastructure as Code security and compliance
- **Command/Code**:

```
# AWS CloudFormation snippet for secure infrastructure
Resources:
  MyEC2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: "ami-0abcdef1234567890"
```

```
# Bridgecrew scan
bridgecrew -d /path/to/cloudformation
```

- **Description**: AWS CloudFormation scripts define the infrastructure, and Bridgecrew checks them for security and compliance.

6. Continuous Monitoring & Logging

- **Tools**: Grafana Loki, Prometheus
- **Security Practice**: Log aggregation and performance monitoring
- **Command/Code**:

```
# Prometheus configuration for monitoring
global:
  scrape_interval: 15s
scrape_configs:
  - job_name: 'microservices'
    static_configs:
      - targets: ['service1:9090', 'service2:9090']
```

- **Description**: Grafana Loki aggregates logs, and Prometheus monitors microservices performance and health.

7. Incident Response & Feedback

- **Tools**: PagerDuty, Tenable.io
- **Security Practice**: Incident alerting and vulnerability management
- **Command/Code**:

```
# Tenable.io scan initiation
curl -X POST -H 'Content-Type: application/json' -H 'X-ApiKeys:
accessKey=ACCESS_KEY;secretKey=SECRET_KEY' \
-d '{"uuid": "TEMPLATE_UUID"}' \
'https://cloud.tenable.com/scans'
```

- **Description**: PagerDuty handles incident alerts, and Tenable.io is used for continuous vulnerability management.

Textual System Diagram

```
[Developers] --plan & code--> [Confluence & IntelliJ IDEA]
    |
    | (1) Secure Design & Development
    v
[IntelliJ IDEA] --commit code--> [Bitbucket Repository]
    |
    | (2) Source Code Management & Secret Scanning
    v
[Bitbucket] --trigger CI--> [CircleCI & Veracode]
    |
    | (3) SAST & Container Scanning
    v
[CircleCI] --deploy code--> [Spinnaker & HashiCorp Vault]
    |
    | (4) Deployment Automation & Secret Management
    v
[Spinnaker] --manage infrastructure--> [AWS CloudFormation & Bridgecrew]
    |
    | (5) IaC Security & Compliance
    v
[AWS CloudFormation] --monitoring--> [Grafana Loki & Prometheus]
```

```
    |
    | (6) Log Aggregation & Performance Monitoring
    v
[Grafana Loki & Prometheus] --incident detection--> [PagerDuty & Tenable.io]
    |
    | (7) Incident Alerting & Vulnerability Management
    v
[PagerDuty & Tenable.io] --feedback loop--> [Developers/Security Team]
```

CIA Triangle for Compliance

- **Confidentiality (Rating: 8.5/10)**
  - **Proof**: HashiCorp Vault for secrets management and Bitbucket with GitLeaks for source code confidentiality.
- **Integrity (Rating: 9/10)**
  - **Proof**: Veracode and Bridgecrew ensure the integrity of code and infrastructure. CircleCI maintains the integrity of the CI/CD process.
- **Availability (Rating: 8/10)**
  - **Proof**: Prometheus and Grafana Loki ensure high availability through monitoring, but additional disaster recovery measures could improve the score.

Scenario: Comprehensive DevSecOps in a Hybrid Cloud Environment

1. Project Management & Planning

- **Tools**: Microsoft Azure DevOps, Miro
- **Security Practice**: Integrating security into Agile workflows
- **Description**: Azure DevOps for sprint planning with security tasks integrated. Miro is used for collaborative threat modeling.

2. Code Development & Review

- **Tools**: Visual Studio Code, Codacy
- **Security Practice**: Secure coding and automated code reviews
- **Command/Code**:

```
# Codacy analysis CLI command
codacy-analysis-cli analyze --directory /path/to/project --format json
```

- **Description**: Developers use Visual Studio Code for development, with Codacy integrated for automated code reviews and security checks.

3. Continuous Integration

- **Tools**: GitLab CI/CD, Whitesource
- **Security Practice**: SAST and open-source vulnerability scanning
- **Command/Code**:

```
# GitLab CI/CD pipeline configuration for Whitesource
whitesource-scan:
  image: whitesource/agent
  script:
    - ws unify -apiKey $WHITESOURCE_API_KEY -projectName "MyProject"
```

- **Description**: GitLab CI/CD pipelines are configured to use Whitesource for scanning dependencies and ensuring open-source security compliance.

4. Continuous Deployment

- **Tools**: Jenkins, Terraform
- **Security Practice**: Secure deployment automation and infrastructure as code
- **Command/Code**:

```
// Jenkins pipeline for Terraform deployment
pipeline {
  agent any
  stages {
    stage('Deploy') {
      steps {
        sh 'terraform apply -auto-approve'
      }
    }
  }
}
```

- **Description**: Jenkins automates the deployment process, with Terraform scripts managing the provisioning of cloud infrastructure securely.

5. Infrastructure & Configuration Management

- **Tools**: Ansible, Cloud Custodian
- **Security Practice**: Configuration management and cloud security enforcement
- **Command/Code**:

```
# Ansible playbook for configuration management
- hosts: all
  tasks:
    - name: Ensure latest security patches are applied
      apt:
        upgrade: 'dist'
```

```
# Cloud Custodian policy
policies:
  - name: ec2-compliance
    resource: ec2
    filters:
      - type: value
        key: InstanceType
        value: t2.micro
```

- **Description**: Ansible ensures configurations are secure and consistent. Cloud Custodian enforces security policies in the cloud environment.

6. Monitoring & Incident Response

- **Tools**: Splunk, Grafana
- **Security Practice**: Security information and event management (SIEM) and incident response
- **Command/Code**:

```
# Splunk search query for monitoring
index=main sourcetype=myapp | stats count by host
```

- **Description**: Splunk for SIEM, aggregating logs and providing insights for security monitoring. Grafana is used for visualizing metrics and setting up alerts.

7. Compliance & Vulnerability Management

- **Tools**: Qualys, Prisma Cloud

- **Security Practice**: Continuous compliance checks and vulnerability assessment
- **Command/Code**:

```
# Qualys API call for vulnerability scan
curl -X POST -H "X-Requested-With: Curl" -u "USERNAME:PASSWORD" \
"https://qualysapi.qualys.com/api/2.0/fo/scan/"
```

- **Description**: Qualys for regular vulnerability scanning and assessment. Prisma Cloud ensures continuous compliance in the cloud environment.

Textual System Diagram

```
[Project Management Team] --plan--> [Azure DevOps & Miro]
    |
    | (1) Agile Planning & Threat Modeling
    v
[Developers] --code--> [Visual Studio Code & Codacy]
    |
    | (2) Code Development & Automated Review
    v
[Code Repository] --CI/CD pipeline--> [GitLab CI/CD & Whitesource]
    |
    | (3) Continuous Integration & Security Scanning
    v
[GitLab CI/CD] --deploy--> [Jenkins & Terraform]
    |
    | (4) Deployment Automation & Infrastructure Provisioning
    v
[Infrastructure] --manage & enforce--> [Ansible & Cloud Custodian]
    |
    | (5) Configuration Management & Cloud Security
    v
[Operational Environment] --monitor & respond--> [Splunk & Grafana]
    |
    | (6) SIEM & Incident Response
    v
[Compliance Team] --compliance & vulnerability management--> [Qualys & Prisma Cloud]
    |
    | (7) Compliance Checks & Vulnerability Assessment
    v
[Feedback] --back to planning--> [Project Management Team]
```

CIA Triangle for Compliance

- **Confidentiality (Rating: 8.5/10)**
  - **Proof**: Codacy and Whitesource ensure code confidentiality. Cloud Custodian enforces data privacy policies.
- **Integrity (Rating: 9/10)**
  - **Proof**: GitLab CI/CD and Jenkins maintain the integrity of the deployment process. Qualys ensures the integrity of the operational environment.
- **Availability (Rating: 8/10)**
  - **Proof**: Terraform and Ansible ensure the availability of infrastructure. Splunk and Grafana provide operational monitoring, but disaster recovery strategies could be further optimized.

Scenario: Streamlined DevSecOps in a Serverless Architecture

1. Project Initiation & Planning

- **Tools**: Trello, Lucidchart
- **Security Practice**: Agile planning with integrated security tasks

- **Description**: Trello for task management with security tasks integrated into the sprint. Lucidchart for creating secure architecture diagrams.

2. Development & Code Review

- **Tools**: VS Code with Security Extensions, CodeClimate
- **Security Practice**: Secure coding and automated code quality checks
- **Command/Code**:

```
# CodeClimate CLI analysis
codeclimate analyze -f json > codeclimate_report.json
```

- **Description**: Developers use VS Code with security extensions for development. CodeClimate is integrated for automated code quality and security checks.

3. Continuous Integration

- **Tools**: GitHub Actions, SonarCloud
- **Security Practice**: SAST and code quality analysis
- **Command/Code**:

```
# GitHub Actions workflow for SonarCloud
name: SonarCloud
on: [push, pull_request]
jobs:
  build:
    name: SonarCloud Scan
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: SonarCloud Scan
        uses: sonarsource/sonarcloud-github-action@master
```

- **Description**: GitHub Actions triggers SonarCloud for static application security testing and code quality analysis.

4. Deployment & Configuration Management

- **Tools**: AWS SAM (Serverless Application Model), AWS Secrets Manager
- **Security Practice**: Secure serverless deployment and secret management
- **Command/Code**:

```
# AWS SAM template for serverless deployment
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs14.x
      Environment:
        Variables:
          SECRET_NAME: mySecret
```

- **Description**: AWS SAM for deploying serverless applications. AWS Secrets Manager securely manages and retrieves secrets.

5. Infrastructure as Code (IaC)

- **Tools**: Terraform, Checkov
- **Security Practice**: IaC security and compliance
- **Command/Code**:

```
# Terraform script for AWS infrastructure
resource "aws_s3_bucket" "my_bucket" {
  bucket = "my-secure-bucket"
  acl    = "private"
}
```

```
# Checkov for Terraform security
checkov -d /path/to/terraform
```

- **Description**: Terraform scripts define cloud infrastructure, with Checkov ensuring IaC security and compliance.

6. Monitoring & Incident Management

- **Tools**: Datadog, PagerDuty
- **Security Practice**: Real-time monitoring and incident alerting
- **Command/Code**:

```
# Datadog configuration for AWS Lambda monitoring
logs:
  - type: lambda
    source: my_lambda_function
```

- **Description**: Datadog for monitoring serverless applications and infrastructure. PagerDuty for incident alerting and management.

7. Compliance & Vulnerability Assessment

- **Tools**: Prisma Cloud, Snyk
- **Security Practice**: Cloud compliance and vulnerability scanning
- **Command/Code**:

```
# Snyk command for vulnerability scanning
snyk test --all-projects
```

- **Description**: Prisma Cloud for continuous cloud compliance checks. Snyk scans for vulnerabilities in dependencies and serverless functions.

Textual System Diagram

```
[Team] --plan & design--> [Trello & Lucidchart]
    |
    | (1) Agile Planning & Secure Design
    v
[Developers] --code & review--> [VS Code & CodeClimate]
    |
    | (2) Development & Automated Review
    v
[Code Repository] --CI pipeline--> [GitHub Actions & SonarCloud]
    |
    | (3) CI & Security Analysis
    v
[GitHub Actions] --deploy--> [AWS SAM & Secrets Manager]
    |
    | (4) Serverless Deployment & Secret Management
```

```
[Deployment] --IaC--> [Terraform & Checkov]
    |
    | (5) Infrastructure as Code Security
    v
[Terraform] --monitor & manage--> [Datadog & PagerDuty]
    |
    | (6) Monitoring & Incident Management
    v
[Operational Environment] --compliance & assessment--> [Prisma Cloud & Snyk]
    |
    | (7) Compliance Checks & Vulnerability Scanning
    v
[Feedback] --back to planning--> [Team]
```

CIA Triangle for Compliance

- **Confidentiality (Rating: 8.5/10)**
  - **Proof**: AWS Secrets Manager for secret confidentiality. CodeClimate and SonarCloud ensure secure code.
- **Integrity (Rating: 9/10)**
  - **Proof**: GitHub Actions and Terraform maintain deployment integrity. Checkov ensures IaC integrity.
- **Availability (Rating: 8/10)**
  - **Proof**: Datadog ensures high availability through monitoring. AWS SAM optimizes serverless deployment availability.

Scenario: Advanced DevSecOps in a Containerized Environment

1. Project Management & Requirement Analysis

- **Tools**: Jira, Draw.io
- **Security Practice**: Agile security task integration and architectural design
- **Description**: Using Jira for sprint planning with integrated security tasks. Draw.io for designing secure system architectures.

2. Development & Static Code Analysis

- **Tools**: Visual Studio Code, SonarLint
- **Security Practice**: Secure coding and real-time static analysis
- **Command/Code**:

```
# SonarLint integration in VS Code for real-time analysis
# Extension installed directly in VS Code
```

- **Description**: Developers use Visual Studio Code with SonarLint extension for real-time static code analysis and secure coding practices.

3. Version Control & Secret Scanning

- **Tools**: Git, GitGuardian
- **Security Practice**: Source code management and secret leak prevention
- **Command/Code**:

```
# GitGuardian pre-commit hook for secret scanning
ggshield secret scan pre-commit
```

- **Description**: Git for version control with GitGuardian integrated to scan and prevent secrets from being committed.

4. Continuous Integration & Security Scanning

- **Tools**: Jenkins, Anchore Engine
- **Security Practice**: Automated CI pipeline with container image scanning
- **Command/Code**:

```
// Jenkins pipeline for building and scanning Docker images
pipeline {
  agent any
  stages {
    stage('Build & Scan Image') {
      steps {
        sh 'docker build -t myapp:latest .'
        sh 'anchore-cli image add myapp:latest'
        sh 'anchore-cli image wait myapp:latest'
        sh 'anchore-cli evaluate check myapp:latest'
      }
    }
  }
}
```

- **Description**: Jenkins automates the CI process, including building Docker images and scanning them with Anchore Engine for vulnerabilities.

5. Container Orchestration & Configuration Management

- **Tools**: Kubernetes, Helm, HashiCorp Vault
- **Security Practice**: Secure container orchestration and secret management
- **Command/Code**:

```
# Helm chart snippet for Kubernetes deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 3
  template:
    spec:
      containers:
      - name: myapp
        image: myapp:latest
```

- **Description**: Kubernetes for container orchestration with Helm charts for deployment management. HashiCorp Vault for handling secrets and sensitive configuration.

6. Continuous Monitoring & Incident Response

- **Tools**: Prometheus, Grafana, Alertmanager
- **Security Practice**: Real-time monitoring and alerting
- **Command/Code**:

```
# Prometheus configuration for monitoring Kubernetes pods
global:
  scrape_interval: 15s
scrape_configs:
  - job_name: 'kubernetes-pods'
    kubernetes_sd_configs:
      - role: pod
```

- **Description**: Prometheus and Grafana for monitoring containerized applications and infrastructure. Alertmanager for handling alerts and incidents.

7. Compliance & Vulnerability Management

- **Tools**: Aqua Security, Prisma Cloud
- **Security Practice**: Compliance enforcement and vulnerability assessment
- **Command/Code**:

```
# Aqua Security CLI command for scanning
aqua scan --image myapp:latest --registry myregistry
```

- **Description**: Aqua Security for runtime protection and scanning container images. Prisma Cloud for ensuring compliance in the cloud environment.

Textual System Diagram

```
[Team] --plan & design--> [Jira & Draw.io]
    |
    | (1) Agile Planning & Architectural Design
    v
[Developers] --code & analyze--> [VS Code & SonarLint]
    |
    | (2) Development & Static Analysis
    v
[Code Repository] --version control & scan--> [Git & GitGuardian]
    |
    | (3) SCM & Secret Scanning
    v
[Git] --CI pipeline--> [Jenkins & Anchore Engine]
    |
    | (4) CI & Container Scanning
    v
[Jenkins] --orchestrate & configure--> [Kubernetes, Helm & Vault]
    |
    | (5) Orchestration & Configuration Management
    v
[Kubernetes] --monitor & alert--> [Prometheus, Grafana & Alertmanager]
    |
    | (6) Monitoring & Incident Response
    v
[Operational Environment] --compliance & vulnerability--> [Aqua Security & Prisma Cloud]
    |
    | (7) Compliance & Vulnerability Management
    v
[Feedback] --back to planning--> [Team]
```

CIA Triangle for Compliance

- **Confidentiality (Rating: 9/10)**
  - **Proof**: HashiCorp Vault for secrets management. GitGuardian to prevent secret leaks.
- **Integrity (Rating: 9/10)**
  - **Proof**: SonarLint and Anchore Engine ensure code and container integrity. Jenkins maintains CI process integrity.
- **Availability (Rating: 8/10)**
  - **Proof**: Kubernetes and Prometheus ensure high availability. Grafana and Alertmanager for real-time alerting, but disaster recovery strategies could be further optimized.

Scenario: Comprehensive DevSecOps in AI/ML Project Development

1. Project Planning & Design 📈

- **Tools**: Monday.com, Lucidchart
- **Security Control**: Security-focused project management, secure architecture design
- **Security Rule**: "Incorporate security considerations in all project phases and architectural designs."
- **Description**: Monday.com is used for project management with a focus on security tasks. Lucidchart for designing a secure AI/ML architecture.

2. Code Development & Review 👨‍💻

- **Tools**: PyCharm, Bandit (Python Security Linter)
- **Security Control**: Secure coding practices, static code analysis
- **Security Rule**: "All Python code must pass security linting with Bandit before merging."
- **Command/Code**:

```
# Bandit security linting command
bandit -r ./my_ml_project
```

- **Description**: Developers use PyCharm for Python development. Bandit is integrated for static security analysis of Python code.

3. Version Control & Secret Management 🔒

- **Tools**: GitLab, Doppler
- **Security Control**: Secure source code management, secret protection
- **Security Rule**: "No hard-coded secrets; use Doppler for secret management."
- **Command/Code**:

```
# Doppler setup command
doppler setup --project my_ml_project --config dev
```

- **Description**: GitLab for version control with integrated CI/CD. Doppler for managing and injecting secrets into the CI/CD pipeline.

4. Continuous Integration & Dependency Scanning 🔄

- **Tools**: Jenkins, OWASP Dependency-Check
- **Security Control**: Automated CI pipeline, open-source vulnerability scanning
- **Security Rule**: "Scan all dependencies for vulnerabilities in each CI build."
- **Command/Code**:

```
// Jenkins pipeline stage for dependency scanning
stage('Dependency Scan') {
  steps {
    sh 'dependency-check.sh --project "My ML Project" --scan ./src'
  }
}
```

- **Description**: Jenkins for CI with automated builds and tests. OWASP Dependency-Check scans project dependencies for vulnerabilities.

5. Containerization & Orchestration 📦

- **Tools**: Docker, Kubernetes
- **Security Control**: Secure containerization, orchestrated deployment
- **Security Rule**: "Use Docker for containerization with secure base images and Kubernetes for orchestration."
- **Command/Code**:

```
# Dockerfile example
FROM python:3.8-slim
```

```
WORKDIR /app
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

- **Description**: Docker for containerizing the AI/ML application. Kubernetes for orchestrating container deployment.

6. Monitoring & Incident Response 🚨

- **Tools**: Prometheus, Grafana, PagerDuty
- **Security Control**: Real-time monitoring, alerting, and incident management
- **Security Rule**: "Monitor application performance and health; alert and respond to incidents promptly."
- **Command/Code**:

```
# Prometheus scrape config
scrape_configs:
  - job_name: 'kubernetes'
    kubernetes_sd_configs:
      - role: pod
```

- **Description**: Prometheus and Grafana for monitoring the Kubernetes environment. PagerDuty for incident alerting and response.

7. Compliance & Vulnerability Management 🔎

- **Tools**: Nessus, AWS Security Hub
- **Security Control**: Regular vulnerability assessments, centralized security insights
- **Security Rule**: "Conduct regular vulnerability scans and review AWS Security Hub for insights."
- **Command/Code**:

```
# Nessus scan initiation command
nessuscli scan new --template "basic" --name "My ML Project Scan" --targets "10.0.0.1,10.0.0.2"
```

- **Description**: Nessus for vulnerability scanning of the network and hosts. AWS Security Hub for centralized security insights and compliance checks.

Textual System Diagram

```
[Team] --plan & design--> [Monday.com & Lucidchart] 📈
    |
    | (1) Project Planning & Secure Design
    v
[Developers] --code & review--> [PyCharm & Bandit] 👨‍💻
    |
    | (2) Secure Coding & Static Analysis
    v
[Code Repository] --version control & secrets--> [GitLab & Doppler] 🔒
    |
    | (3) SCM & Secret Management
    v
[GitLab] --CI pipeline--> [Jenkins & OWASP Dependency-Check] 🔄
    |
    | (4) CI & Dependency Scanning
    v
[Jenkins] --containerize & orchestrate--> [Docker & Kubernetes] 📦
    |
    | (5) Containerization & Orchestration
    v
[Kubernetes] --monitor & respond--> [Prometheus, Grafana & PagerDuty] 🚨
```

```
    | (6) Monitoring & Incident Response
    v
[Operational Environment] --compliance & assessment--> [Nessus & AWS Security Hub] 🔎
    |
    | (7) Compliance & Vulnerability Management
    v
[Feedback] --back to planning--> [Team]
```

CIA Triangle for Compliance

- **Confidentiality (Rating: 9/10)** 🔐
  - **Proof**: Doppler for secret management. GitLab for secure code storage.
- **Integrity (Rating: 8.5/10)** ✅
  - **Proof**: Bandit and OWASP Dependency-Check ensure code and dependency integrity.
- **Availability (Rating: 8/10)** 🔄
  - **Proof**: Kubernetes ensures high availability. Prometheus and Grafana for monitoring, but disaster recovery strategies could be enhanced.

Scenario: DevSecOps in Full-Stack Web Application Development

1. Project Planning & Requirement Analysis 📋

- **Tools**: ClickUp, Miro
- **Security Control**: Security-focused Agile project management, threat modeling
- **Security Rule**: "All project tasks must include security considerations, documented in Miro."
- **Description**: ClickUp for managing project tasks with a focus on security. Miro for collaborative threat modeling and secure design.

2. Development & Static Code Analysis 👨‍💻

- **Tools**: Visual Studio Code, ESLint with Security Plugin
- **Security Control**: Secure coding practices, static code analysis
- **Security Rule**: "All JavaScript code must pass ESLint security linting before commit."
- **Command/Code**:

```
# ESLint security linting command
eslint --ext .js,.jsx src/
```

- **Description**: Developers use Visual Studio Code for coding. ESLint with security plugins for JavaScript linting.

3. Version Control & Secret Scanning 🔐

- **Tools**: GitHub, GitGuardian
- **Security Control**: Secure source code management, secret leak prevention
- **Security Rule**: "Scan for secrets pre-commit; block commits containing secrets."
- **Command/Code**:

```
# GitGuardian pre-commit hook
ggshield secret scan pre-commit
```

- **Description**: GitHub for version control. GitGuardian integrated to scan and prevent secrets from being committed.

4. Continuous Integration & Security Testing 🔄

- **Tools**: Jenkins, SonarQube
- **Security Control**: CI pipeline security, SAST

- **Command/Code**:

```
// Jenkins pipeline for SonarQube integration
pipeline {
  agent any
  stages {
    stage('SonarQube Scan') {
      steps {
        withSonarQubeEnv('SonarQube') {
          sh 'mvn clean verify sonar:sonar'
        }
      }
    }
  }
}
```

- **Description**: Jenkins for CI, integrating SonarQube for static application security testing.

5. Containerization & Orchestration 📦

- **Tools**: Docker, Kubernetes
- **Security Control**: Secure containerization, orchestrated deployment
- **Security Rule**: "Use Docker for containerization with secure base images and Kubernetes for orchestration."
- **Command/Code**:

```
# Dockerfile for web application
FROM node:14-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

- **Description**: Docker for containerizing the web application. Kubernetes for managing container deployment.

6. Monitoring & Incident Management 🚨

- **Tools**: Prometheus, Grafana, Alertmanager
- **Security Control**: Real-time monitoring, alerting, and incident management
- **Security Rule**: "Continuously monitor application and infrastructure; alert on anomalies."
- **Command/Code**:

```
# Prometheus configuration for monitoring
global:
  scrape_interval: 15s
scrape_configs:
  - job_name: 'nodejs-app'
    static_configs:
      - targets: ['nodejs-app:3000']
```

- **Description**: Prometheus for monitoring the application. Grafana for dashboards and visualization. Alertmanager for handling alerts.

7. Compliance & Vulnerability Management 🔍

- **Tools**: Nessus, AWS Security Hub
- **Security Control**: Regular vulnerability assessments, centralized security insights
- **Security Rule**: "Conduct regular vulnerability scans and review AWS Security Hub insights."

```
# Nessus scan initiation command
nessuscli scan new --template "basic" --name "Web App Scan" --targets "webapp.example.com"
```

- **Description**: Nessus for vulnerability scanning. AWS Security Hub for centralized security insights and compliance checks.

Textual System Diagram

```
[Team] --plan & design--> [ClickUp & Miro] 📋
    |
    | (1) Project Planning & Threat Modeling
    v
[Developers] --code & analyze--> [VS Code & ESLint] 👨‍💻
    |
    | (2) Development & Static Analysis
    v
[Code Repository] --version control & scan--> [GitHub & GitGuardian] 🔐
    |
    | (3) SCM & Secret Scanning
    v
[GitHub] --CI pipeline--> [Jenkins & SonarQube] 🔄
    |
    | (4) CI & Security Testing
    v
[Jenkins] --containerize & orchestrate--> [Docker & Kubernetes] 📦
    |
    | (5) Containerization & Orchestration
    v
[Kubernetes] --monitor & alert--> [Prometheus, Grafana & Alertmanager] 🚨
    |
    | (6) Monitoring & Incident Management
    v
[Operational Environment] --compliance & assessment--> [Nessus & AWS Security Hub] 🔍
    |
    | (7) Compliance & Vulnerability Management
    v
[Feedback] --back to planning--> [Team]
```

CIA Triangle for Compliance

- **Confidentiality (Rating: 9/10)** 🔒
  - **Proof**: GitGuardian prevents secret leaks. Docker containers encapsulate application secrets.
- **Integrity (Rating: 8.5/10)** ✅
  - **Proof**: SonarQube ensures code integrity. Jenkins maintains CI process integrity.
- **Availability (Rating: 8/10)** 🔄
  - **Proof**: Kubernetes ensures high availability. Prometheus and Grafana provide operational monitoring, but disaster recovery strategies could be enhanced.

Scenario: DevSecOps in E-commerce Web Application Development

1. Project Planning & Design 📈

- **Tools**: Notion, Adobe XD
- **Security Control**: Security-focused Agile project management, secure UX/UI design
- **Security Rule**: "Incorporate security considerations in all project tasks and UX/UI designs."
- **Description**: Notion for managing project tasks with a focus on security. Adobe XD for designing a secure and user-friendly e-commerce interface.

- **Tools**: Sublime Text, SonarLint
- **Security Control**: Secure coding practices, real-time static analysis
- **Security Rule**: "All code must pass SonarLint security checks before being pushed."
- **Command/Code**:

```
# SonarLint is integrated into Sublime Text for real-time analysis
# No specific command needed; runs automatically
```

- **Description**: Developers use Sublime Text for coding. SonarLint is integrated for real-time static security analysis.

3. Version Control & Secret Management 🔐

- **Tools**: Bitbucket, Doppler
- **Security Control**: Secure source code management, secret protection
- **Security Rule**: "Use Doppler for managing all application secrets and API keys."
- **Command/Code**:

```
# Doppler setup command
doppler setup --project e-commerce --config dev
```

- **Description**: Bitbucket for version control with integrated CI/CD. Doppler for managing and injecting secrets into the CI/CD pipeline.

4. Continuous Integration & Dependency Scanning 🔄

- **Tools**: CircleCI, WhiteSource
- **Security Control**: CI pipeline security, open-source vulnerability scanning
- **Security Rule**: "Scan all dependencies for vulnerabilities in each CI build using WhiteSource."
- **Command/Code**:

```
# CircleCI configuration for WhiteSource scanning
version: 2.1
orbs:
  whitesource: whitesource/wss_agent@1.0.0
workflows:
  version: 2
  build:
    jobs:
      - whitesource/scan
```

- **Description**: CircleCI for continuous integration. WhiteSource scans project dependencies for vulnerabilities.

5. Containerization & Orchestration 📦

- **Tools**: Docker, Amazon ECS
- **Security Control**: Secure containerization, orchestrated deployment
- **Security Rule**: "Use Docker for containerization with secure base images and Amazon ECS for orchestration."
- **Command/Code**:

```
# Dockerfile for e-commerce application
FROM node:14
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 8080
```

- **Description**: Docker for containerizing the e-commerce application. Amazon ECS for managing container deployment.

6. Monitoring & Incident Response 🚨

- **Tools**: New Relic, PagerDuty
- **Security Control**: Real-time monitoring, alerting, and incident management
- **Security Rule**: "Monitor application performance and health; alert and respond to incidents promptly using New Relic and PagerDuty."
- **Command/Code**:

```
# New Relic configuration for monitoring
# Configured via New Relic APM UI;
```

- **Description**: New Relic for monitoring the application. PagerDuty for incident alerting and response.

7. Compliance & Vulnerability Management 🔍

- **Tools**: Qualys, AWS Security Hub
- **Security Control**: Regular vulnerability assessments, centralized security insights
- **Security Rule**: "Conduct regular vulnerability scans with Qualys and review AWS Security Hub for insights."
- **Command/Code**:

```
# Qualys scan initiation command
qualys-api.sh --start-scan --target "e-commerce.example.com"
```

- **Description**: Qualys for vulnerability scanning. AWS Security Hub for centralized security insights and compliance checks.

Textual System Diagram

```
[Team] --plan & design--> [Notion & Adobe XD] 📈
    |
    | (1) Project Planning & Secure Design
    v
[Developers] --code & analyze--> [Sublime Text & SonarLint] 💻
    |
    | (2) Development & Static Analysis
    v
[Code Repository] --version control & secrets--> [Bitbucket & Doppler] 🔐
    |
    | (3) SCM & Secret Management
    v
[Bitbucket] --CI pipeline--> [CircleCI & WhiteSource] 🔄
    |
    | (4) CI & Dependency Scanning
    v
[CircleCI] --containerize & orchestrate--> [Docker & Amazon ECS] 📦
    |
    | (5) Containerization & Orchestration
    v
[Amazon ECS] --monitor & alert--> [New Relic & PagerDuty] 🚨
    |
    | (6) Monitoring & Incident Response
    v
[Operational Environment] --compliance & assessment--> [Qualys & AWS Security Hub] 🔍
    |
    | (7) Compliance & Vulnerability Management
```

```
[Feedback] --back to planning--> [Team]
```

CIA Triangle for Compliance

- **Confidentiality (Rating: 9/10)** 🔒
  - **Proof**: Doppler ensures confidentiality of secrets. Bitbucket provides secure code storage.
- **Integrity (Rating: 8.5/10)** ✅
  - **Proof**: SonarLint and WhiteSource maintain code and dependency integrity. CircleCI ensures CI process integrity.
- **Availability (Rating: 8/10)** 🔄
  - **Proof**: Amazon ECS and Docker ensure high availability. New Relic provides operational monitoring, but disaster recovery strategies could be improved.

Scenario: DevSecOps in a Modern Web Application Development

1. Project Planning & Design 📋

- **Tools**: Trello, Balsamiq
- **Security Control**: Security-integrated Agile project management, secure UI/UX design
- **Security Rule**: "All project tasks must include security considerations, and UI/UX designs should be vetted for security."
- **Description**: Trello is used for Agile project management with security tasks. Balsamiq for wireframing secure UI/UX designs.

2. Development & Static Code Analysis 👨‍💻

- **Tools**: VS Code, ESLint with Security Plugins
- **Security Control**: Secure coding practices, static code analysis
- **Security Rule**: "All JavaScript code must pass ESLint security linting before commit."
- **Command/Code**:

```
# ESLint security linting command
eslint --ext .js,.jsx,.ts,.tsx src/
```

- **Description**: Developers use VS Code for coding. ESLint with security plugins for JavaScript/TypeScript linting.

3. Version Control & Secret Management 🔐

- **Tools**: GitHub, HashiCorp Vault
- **Security Control**: Secure source code management, secret protection
- **Security Rule**: "No hard-coded secrets; use HashiCorp Vault for secret management."
- **Command/Code**:

```
# HashiCorp Vault command to retrieve a secret
vault kv get secret/my-web-app/db-password
```

- **Description**: GitHub for version control. HashiCorp Vault for managing and storing secrets.

4. Continuous Integration & Dependency Scanning 🔄

- **Tools**: Jenkins, OWASP Dependency-Check
- **Security Control**: CI pipeline security, open-source vulnerability scanning
- **Security Rule**: "Scan all dependencies for vulnerabilities in each CI build."
- **Command/Code**:

```
// Jenkins pipeline stage for OWASP Dependency-Check
pipeline {
```

```
  agent any
  stages {
    stage('Dependency Check') {
      steps {
        sh 'dependency-check.sh --project "My Web App" --scan ./src'
      }
    }
  }
}
```

- **Description**: Jenkins for CI, integrating OWASP Dependency-Check for scanning project dependencies.

5. Containerization & Orchestration 📦

- **Tools**: Docker, Kubernetes
- **Security Control**: Secure containerization, orchestrated deployment
- **Security Rule**: "Use Docker for containerization with secure base images and Kubernetes for orchestration."
- **Command/Code**:

```
# Dockerfile for the web application
FROM node:14-alpine
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

- **Description**: Docker for containerizing the web application. Kubernetes for managing container deployment.

6. Monitoring & Incident Response 🚨

- **Tools**: Prometheus, Grafana, Alertmanager
- **Security Control**: Real-time monitoring, alerting, and incident management
- **Security Rule**: "Continuously monitor application and infrastructure; alert on anomalies."
- **Command/Code**:

```
# Prometheus configuration for monitoring
global:
  scrape_interval: 15s
scrape_configs:
  - job_name: 'my-web-app'
    static_configs:
      - targets: ['my-web-app:3000']
```

- **Description**: Prometheus for monitoring the application. Grafana for dashboards and visualization. Alertmanager for handling alerts.

7. Compliance & Vulnerability Management 🔍

- **Tools**: Nessus, AWS Security Hub
- **Security Control**: Regular vulnerability assessments, centralized security insights
- **Security Rule**: "Conduct regular vulnerability scans and review AWS Security Hub insights."
- **Command/Code**:

```
# Nessus scan initiation command
nessuscli scan new --template "basic" --name "My Web App Scan" --targets "webapp.example.com"
```

- **Description**: Nessus for vulnerability scanning. AWS Security Hub for centralized security insights and compliance checks.

Textual System Diagram

```
[Team] --plan & design--> [Trello & Balsamiq] 📋
    |
    | (1) Project Planning & Secure Design
    v
[Developers] --code & analyze--> [VS Code & ESLint] 👨‍💻
    |
    | (2) Development & Static Analysis
    v
[Code Repository] --version control & secrets--> [GitHub & HashiCorp Vault] 🔐
    |
    | (3) SCM & Secret Management
    v
[GitHub] --CI pipeline--> [Jenkins & OWASP Dependency-Check] 🔄
    |
    | (4) CI & Dependency Scanning
    v
[Jenkins] --containerize & orchestrate--> [Docker & Kubernetes] 📦
    |
    | (5) Containerization & Orchestration
    v
[Kubernetes] --monitor & alert--> [Prometheus, Grafana & Alertmanager] 🚨
    |
    | (6) Monitoring & Incident Response
    v
[Operational Environment] --compliance & assessment--> [Nessus & AWS Security Hub] 🔍
    |
    | (7) Compliance & Vulnerability Management
    v
[Feedback] --back to planning--> [Team]
```

CIA Triangle for Compliance

- **Confidentiality (Rating: 9/10)** 🔒
  - **Proof**: HashiCorp Vault ensures the confidentiality of secrets. GitHub provides secure code storage.
- **Integrity (Rating: 8.5/10)** ✅
  - **Proof**: ESLint and OWASP Dependency-Check maintain code and dependency integrity. Jenkins ensures CI process integrity.
- **Availability (Rating: 8/10)** 🔄
  - **Proof**: Kubernetes ensures high availability. Prometheus and Grafana provide operational monitoring, but disaster recovery strategies could be enhanced.

Scenario: DevSecOps in a Financial Services Application

1. Project Planning & Threat Modeling 📈

- **Tools**: Jira, ThreatModeler
- **Security Control**: Security-focused Agile project management, threat modeling
- **Security Rule**: "Incorporate security and threat modeling in all project phases."
- **Solution**: Use Jira for task management with security tasks. ThreatModeler for creating threat models specific to financial services.
- **Attacker Technique**: T1587.001 (Gather Victim Identity Information)

2. Secure Coding & Code Review 👨‍💻

- **Tools**: Eclipse, SonarQube
- **Security Control**: Secure coding practices, static code analysis

- **Solution**: Eclipse IDE with SonarLint plugin for real-time code analysis. SonarQube for comprehensive code review.
- **Attacker Technique**: T1195 (Supply Chain Compromise)
- **Command/Code**:

```
# SonarQube analysis command
mvn sonar:sonar -Dsonar.projectKey=financial_app -Dsonar.host.url=http://localhost:9000
```

3. Version Control & Secret Management 🔐

- **Tools**: GitLab, Azure Key Vault
- **Security Control**: Secure source code management, secret protection
- **Security Rule**: "No hard-coded secrets; use Azure Key Vault for all secrets."
- **Solution**: GitLab for version control with integrated CI/CD. Azure Key Vault for storing and managing secrets.
- **Attacker Technique**: T1552.001 (Unsecured Credentials: Credentials in Files)
- **Command/Code**:

```
# Azure Key Vault secret retrieval
az keyvault secret show --name db-password --vault-name MyVault
```

4. Continuous Integration & Dependency Scanning 🔄

- **Tools**: Jenkins, OWASP Dependency-Check
- **Security Control**: CI pipeline security, dependency vulnerability scanning
- **Security Rule**: "Scan all dependencies for vulnerabilities in each CI build."
- **Solution**: Jenkins for CI, integrating OWASP Dependency-Check for scanning dependencies.
- **Attacker Technique**: T1199 (Trusted Relationship)
- **Command/Code**:

```
// Jenkins pipeline stage for dependency scanning
stage('Dependency Check') {
  steps {
    sh 'dependency-check --project "Financial App" --scan ./src'
  }
}
```

5. Containerization & Orchestration 📦

- **Tools**: Docker, Kubernetes
- **Security Control**: Secure containerization, orchestrated deployment
- **Security Rule**: "Use Docker for containerization with secure base images and Kubernetes for orchestration."
- **Solution**: Docker for containerizing the application. Kubernetes for managing container deployment.
- **Attacker Technique**: T1525 (Server Software Component)
- **Command/Code**:

```
# Dockerfile for the application
FROM openjdk:11-jdk-slim
COPY target/financial-app.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

6. Monitoring & Incident Response 🚨

- **Tools**: Splunk, TheHive
- **Security Control**: Real-time monitoring, alerting, and incident management
- **Security Rule**: "Monitor application and infrastructure; alert and manage incidents."
- **Solution**: Splunk for SIEM and log analysis. TheHive for incident response and management.

- **Command/Code**:

```
# Splunk search query for monitoring
index="financial_logs" sourcetype="financial_app" | stats count by host, severity
```

7. Compliance & Vulnerability Management 🔍

- **Tools**: Nessus, AWS Security Hub
- **Security Control**: Regular vulnerability assessments, centralized security insights
- **Security Rule**: "Conduct regular vulnerability scans and review AWS Security Hub insights."
- **Solution**: Nessus for vulnerability scanning. AWS Security Hub for centralized security insights and compliance checks.
- **Attacker Technique**: T1595 (Active Scanning)
- **Command/Code**:

```
# Nessus scan initiation command
nessuscli scan new --template "advanced" --name "Financial App Scan" --targets "app.example.com"
```

Textual System Diagram

```
[Team] --plan & threat model--> [Jira & ThreatModeler] 📈
    |
    | (1) Project Planning & Threat Modeling
    v
[Developers] --code & review--> [Eclipse & SonarQube] 👩‍💻
    |
    | (2) Secure Coding & Code Review
    v
[Code Repository] --version control & secrets--> [GitLab & Azure Key Vault] 🔐
    |
    | (3) SCM & Secret Management
    v
[GitLab] --CI pipeline--> [Jenkins & OWASP Dependency-Check] 🔄
    |
    | (4) CI & Dependency Scanning
    v
[Jenkins] --containerize & orchestrate--> [Docker & Kubernetes] 📦
    |
    | (5) Containerization & Orchestration
    v
[Kubernetes] --monitor & respond--> [Splunk & TheHive] 🚨
    |
    | (6) Monitoring & Incident Response
    v
[Operational Environment] --compliance & assessment--> [Nessus & AWS Security Hub] 🔍
    |
    | (7) Compliance & Vulnerability Management
    v
[Feedback] --back to planning--> [Team]
```

CIA Triangle for Compliance

- **Confidentiality (Rating: 9/10)** 🔒
  - **Proof**: Azure Key Vault ensures confidentiality of secrets. GitLab provides secure code storage.
- **Integrity (Rating: 8.5/10)** ✅
  - **Proof**: SonarQube and OWASP Dependency-Check maintain code and dependency integrity. Jenkins ensures CI process integrity.
- **Availability (Rating: 8/10)** 🔄

- **Proof**: Kubernetes ensures high availability. Splunk and TheHive provide operational monitoring and incident response, but disaster recovery strategies could be improved.

Scenario: DevSecOps in Advanced Cloud-Based IoT Application Development

1. Project Planning & Threat Modeling 📋

- **Tools**: Azure DevOps, securiCAD
- **Security Control**: Security-integrated Agile project management, advanced threat modeling
- **Security Rule**: "Incorporate detailed security considerations and threat modeling in all project phases."
- **Solution**: Azure DevOps for task management with security tasks. securiCAD for advanced threat modeling specific to IoT.
- **Attacker Technique**: T1584.003 (Exploit Public-Facing Application)

2. Secure Coding & Code Review 👨‍💻

- **Tools**: PyCharm, CodeQL
- **Security Control**: Secure coding practices, advanced static code analysis
- **Security Rule**: "All Python code must pass CodeQL security checks before merging."
- **Solution**: PyCharm for Python development. CodeQL for deep static code analysis and identifying vulnerabilities.
- **Attacker Technique**: T1195 (Supply Chain Compromise)
- **Command/Code**:

```
# CodeQL analysis command
codeql database create mydb --language=python
codeql database analyze mydb python-security-queries.qls --format=sarif-latest --
output=results.sarif
```

3. Version Control & Secret Management 🔐

- **Tools**: Git, HashiCorp Vault
- **Security Control**: Secure source code management, enhanced secret protection
- **Security Rule**: "No hard-coded secrets; use HashiCorp Vault for all secret management."
- **Solution**: Git for version control. HashiCorp Vault for advanced secret management.
- **Attacker Technique**: T1552.003 (Credentials from Password Stores)
- **Command/Code**:

```
# HashiCorp Vault secret retrieval
vault kv get -field=password secret/my-iot-app/db
```

4. Continuous Integration & Dependency Scanning 🔄

- **Tools**: GitHub Actions, Dependabot
- **Security Control**: CI pipeline security, automated dependency updates
- **Security Rule**: "Automatically update dependencies and scan for vulnerabilities."
- **Solution**: GitHub Actions for CI. Dependabot for automated dependency updates and vulnerability scanning.
- **Attacker Technique**: T1199 (Trusted Relationship)
- **Command/Code**:

```
# GitHub Actions workflow for Dependabot
name: Dependabot Auto Merge
on: pull_request
jobs:
  auto-merge:
    runs-on: ubuntu-latest
    steps:
      - name: Merge PRs
```

```
        uses: dependabot/fetch-metadata@v1.1.0
        with:
          github-token: "${{ secrets.GITHUB_TOKEN }}"
```

5. Containerization & Orchestration 📦

- **Tools**: Docker, Amazon EKS
- **Security Control**: Secure containerization, cloud-based orchestration
- **Security Rule**: "Use Docker for containerization and Amazon EKS for orchestration."
- **Solution**: Docker for containerizing IoT applications. Amazon EKS for cloud-based Kubernetes orchestration.
- **Attacker Technique**: T1525 (Server Software Component)
- **Command/Code**:

```
# Dockerfile for IoT application
FROM python:3.8-slim
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

6. Monitoring & Incident Response 🚨

- **Tools**: Datadog, IBM QRadar
- **Security Control**: Real-time monitoring, advanced incident response
- **Security Rule**: "Implement comprehensive monitoring and rapid incident response."
- **Solution**: Datadog for real-time monitoring. IBM QRadar for advanced SIEM and incident response.
- **Attacker Technique**: T1592 (Gather Victim Network Information)
- **Command/Code**:

```
# Datadog configuration for IoT monitoring
logs:
  - type: file
    path: /var/log/iot-app.log
    service: iot-app
    source: python
```

7. Compliance & Vulnerability Management 🔍

- **Tools**: Nessus, AWS Security Hub
- **Security Control**: Regular vulnerability assessments, centralized security insights
- **Security Rule**: "Conduct regular vulnerability scans and review AWS Security Hub insights."
- **Solution**: Nessus for comprehensive vulnerability scanning. AWS Security Hub for centralized security insights and compliance checks.
- **Attacker Technique**: T1595 (Active Scanning)
- **Command/Code**:

```
# Nessus scan initiation command
nessuscli scan new --template "advanced" --name "IoT App Scan" --targets "iot-app.example.com"
```

Textual System Diagram

```
[Team] --plan & threat model--> [Azure DevOps & securiCAD] 📋
    |
    | (1) Project Planning & Threat Modeling
    v
[Developers] --code & review--> [PyCharm & CodeQL] 👨‍💻
```

```
    | (2) Secure Coding & Code Review
    v
[Code Repository] --version control & secrets--> [Git & HashiCorp Vault] 🔐
    |
    | (3) SCM & Secret Management
    v
[Git] --CI pipeline--> [GitHub Actions & Dependabot] 🔄
    |
    | (4) CI & Dependency Scanning
    v
[GitHub Actions] --containerize & orchestrate--> [Docker & Amazon EKS] 📦
    |
    | (5) Containerization & Orchestration
    v
[Amazon EKS] --monitor & respond--> [Datadog & IBM QRadar] 🚨
    |
    | (6) Monitoring & Incident Response
    v
[Operational Environment] --compliance & assessment--> [Nessus & AWS Security Hub] 🔍
    |
    | (7) Compliance & Vulnerability Management
    v
[Feedback] --back to planning--> [Team]
```

CIA Triangle for Compliance

- **Confidentiality (Rating: 9/10)** 🔒
  - **Proof**: HashiCorp Vault ensures the confidentiality of secrets. Git provides secure code storage.
- **Integrity (Rating: 9/10)** ✅
  - **Proof**: CodeQL and Dependabot maintain code and dependency integrity. GitHub Actions ensures CI process integrity.
- **Availability (Rating: 8.5/10)** 🔄
  - **Proof**: Amazon EKS ensures high availability. Datadog and IBM QRadar provide comprehensive monitoring and incident response.

Scenario: DevSecOps in a Cloud-Based Payment Processing System

1. Project Planning & Risk Assessment 📈

- **Tools**: ClickUp, ThreatModeler
- **Security Control**: Agile project management with integrated security tasks, automated threat modeling
- **Security Rule**: "Incorporate security risk assessments in all project phases."
- **Solution**: Use ClickUp for sprint planning with security tasks. ThreatModeler for automated threat modeling.
- **Attack Simulation**: N/A (Planning tools do not involve attack simulations)

2. Secure Development & Code Analysis 👨‍💻

- **Tools**: Visual Studio Code, SonarCloud
- **Security Control**: Secure coding practices, static code analysis
- **Security Rule**: "All code must pass SonarCloud analysis before merging."
- **Solution**: Visual Studio Code for development. SonarCloud for static code analysis.
- **Command/Code**:

```
# SonarCloud analysis command
sonar-scanner \
  -Dsonar.projectKey=payment_system \
  -Dsonar.organization=myorganization \
  -Dsonar.sources=. \
  -Dsonar.host.url=https://sonarcloud.io
```

Attack Simulation: SQL Injection Attack

```
SELECT * FROM users WHERE username = 'admin' --' AND password = 'password';
```

3. Version Control & Secret Management 🔐

- **Tools**: GitHub, AWS Secrets Manager
- **Security Control**: Secure source code management, secret protection
- **Security Rule**: "No hard-coded secrets; use AWS Secrets Manager for all secrets."
- **Solution**: GitHub for version control. AWS Secrets Manager for secret management.
- **Command/Code**:

```
# AWS Secrets Manager secret retrieval
aws secretsmanager get-secret-value --secret-id /payment/apikey
```

Attack Simulation: Git Repository Secrets Leak

```
git log -p -G 'password|apikey'
```

4. Continuous Integration & Dependency Scanning 🔄

- **Tools**: Jenkins, OWASP Dependency-Check
- **Security Control**: CI pipeline security, open-source vulnerability scanning
- **Security Rule**: "Scan all dependencies for vulnerabilities in each CI build."
- **Solution**: Jenkins for CI, integrating OWASP Dependency-Check.
- **Command/Code**:

```
// Jenkins pipeline stage for OWASP Dependency-Check
pipeline {
  agent any
  stages {
    stage('Dependency Check') {
      steps {
        sh 'dependency-check --project "Payment System" --scan ./src --out .'
      }
    }
  }
}
```

Attack Simulation: Dependency Confusion Attack

```
npm install fake-payment-sdk@latest
```

5. Containerization & Orchestration 📦

- **Tools**: Docker, Kubernetes
- **Security Control**: Secure containerization, cloud-based orchestration
- **Security Rule**: "Use Docker for containerization and Kubernetes for orchestration."
- **Solution**: Docker for containerizing the application. Kubernetes for managing container deployment.
- **Command/Code**:

```
# Dockerfile for the payment system
FROM node:14
WORKDIR /app
COPY package.json .
RUN npm install
COPY
```

```
CMD ["node", "server.js"]
```

Attack Simulation: Container Escape Attack

```
docker run -v /:/host --rm -it alpine chroot /host
```

6. Monitoring & Incident Response 🚨

- **Tools**: Prometheus, Grafana, ELK Stack
- **Security Control**: Real-time monitoring, alerting, and incident management
- **Security Rule**: "Implement comprehensive monitoring and incident response."
- **Solution**: Prometheus and Grafana for monitoring. ELK Stack for log management.
- **Command/Code**:

```
# Prometheus configuration for monitoring
global:
  scrape_interval: 15s
scrape_configs:
  - job_name: 'payment-service'
    static_configs:
      - targets: ['localhost:9090']
```

Attack Simulation: DDoS Attack Simulation

```
ab -n 10000 -c 100 http://payment-service.example.com/
```

7. Compliance & Vulnerability Management 🔍

- **Tools**: Nessus, AWS Config
- **Security Control**: Regular vulnerability assessments, compliance monitoring
- **Security Rule**: "Regularly assess vulnerabilities and ensure compliance."
- **Solution**: Nessus for vulnerability scanning. AWS Config for compliance monitoring.
- **Command/Code**:

```
# Nessus scan initiation command
nessuscli scan new --template "advanced" --name "Payment System Scan" --targets
"service.example.com"
```

Attack Simulation: Vulnerability Exploitation

```
searchsploit [vulnerability details from Nessus scan]
```

Textual System Diagram

```
[Team] --plan & model--> [ClickUp & ThreatModeler] 📈
    |
    | (1) Project Planning & Risk Assessment
    v
[Developers] --develop & analyze--> [VS Code & SonarCloud] 👨‍💻
    |
    | (2) Secure Development & Code Analysis
    v
[Code Repository] --version control & secrets--> [GitHub & AWS Secrets Manager] 🔐
    |
    | (3) Version Control & Secret Management
    v
[GitHub] --CI pipeline--> [Jenkins & OWASP Dependency-Check] 🔄
```

```
    |
    | (4) Continuous Integration & Dependency Scanning
    v
[Jenkins] --containerize & orchestrate--> [Docker & Kubernetes] 📦
    |
    | (5) Containerization & Orchestration
    v
[Kubernetes] --monitor & respond--> [Prometheus, Grafana & ELK Stack] 🚨
    |
    | (6) Monitoring & Incident Response
    v
[Operational Environment] --compliance & assessment--> [Nessus & AWS Config] 🔍
    |
    | (7) Compliance & Vulnerability Management
    v
[Feedback] --back to planning--> [Team]
```

CIA Triangle for Compliance

- **Confidentiality (Rating: 9/10)** 🔒
  - **Proof**: AWS Secrets Manager ensures the confidentiality of secrets. GitHub provides secure code storage.
- **Integrity (Rating: 9/10)** ✅
  - **Proof**: SonarCloud and OWASP Dependency-Check maintain code and dependency integrity. Jenkins ensures CI process integrity.
- **Availability (Rating: 8.5/10)** 🔄
  - **Proof**: Kubernetes ensures high availability. Prometheus, Grafana, and ELK Stack provide comprehensive monitoring.