{desafío} latam_

CURSO

DEVOPS SENIOR

Objetivo General del Curso

DISEÑAR ENTORNOS CLOUD NATIVE INTEGRANDO PRÁCTICAS DE KUBERNETES Y GITOPS, DE ACUERDO CON ESTÁNDARES DE SEGURIDAD Y OBSERVABILIDAD.

Objetivo específico del Módulo

DEFINIR CONCEPTOS DE SEGURIDAD AVANZADA Y DEVSECOPS, SEGUN LAS PRACTICAS AVANZADAS DE GITOPS, DEVSECOPS, KUBERNETES, OBSERVABILIDAD, IAC, FINOPS Y AIOPS.

Contenidos

Módulo 3: SEGURIDAD AVANZADA Y DEVSECOPS.

Capítulo 1: DevSecOps.

DevSecOps (Development, Security, Operations) es un enfoque que busca integrar la seguridad como una responsabilidad compartida a lo largo de todo el ciclo de vida del desarrollo y operaciones, desde la planificación hasta el despliegue y monitoreo. Su objetivo es que la seguridad deje de ser una etapa posterior ("bolted-on") y pase a ser un componente transversal y automatizado dentro del flujo DevOps.

Este modelo es esencial en entornos de alta criticidad, donde los despliegues son frecuentes, la infraestructura es efímera y la superficie de ataque cambia dinámicamente.

Principios clave de DevSecOps

- **Shift Left:** aplicar controles de seguridad desde las primeras etapas del desarrollo.
- **Automatización de controles:** integrar validaciones de seguridad en cada etapa del pipeline CI/CD.
- **Cultura colaborativa:** romper silos entre desarrollo, operaciones y seguridad.
- **Respuesta rápida ante vulnerabilidades:** mecanismos proactivos de detección, alerta y parcheo.
- **Observabilidad de riesgos:** monitoreo continuo de seguridad desde infraestructura hasta capa de aplicación.

Herramientas y técnicas en DevSecOps

1. Análisis de código y dependencias

- **SAST (Static Application Security Testing):** detección de vulnerabilidades en el código fuente (Ej: SonarQube, CodeQL).
- **DAST (Dynamic Application Security Testing):** análisis de seguridad de aplicaciones en ejecución (Ej: OWASP ZAP).
- **SCA (Software Composition Analysis):** detección de vulnerabilidades en librerías de terceros (Ej: Snyk, WhiteSource).

2. Hardening de infraestructura y contenedores

- Escaneo de imágenes con herramientas como Trivy, Grype, Dockle.
- Validación de Dockerfiles y manifiestos Kubernetes contra benchmark CIS.
- Restricción de privilegios: ejecución de contenedores sin root, control de capabilities.

3. Gestión de secretos y variables sensibles

- Uso de HashiCorp Vault, Sealed Secrets, External Secrets Operator.
- Prevención de exposición accidental mediante escáneres como Gitleaks.

4. Controles de admisión y políticas de seguridad

- Implementación de OPA/Gatekeeper o Kyverno para validar configuraciones antes del despliegue.
- **Ejemplos:** impedir ejecución de contenedores sin readOnlyRootFilesystem, evitar uso de imágenes sin firma, bloquear puertos no permitidos.

5. Firmas, integridad y SBOM

- Uso de cosign y sigstore para firmar imágenes.
- Generación de SBOM (Software Bill of Materials) para trazabilidad de componentes.
- Integración de verificación en pipelines y despliegues GitOps.

DevSecOps en el pipeline CI/CD

- **Commit:** escaneo de secretos y análisis estático del código.
- **Build:** análisis de dependencias, escaneo de imagen, validación de configuración.
- **Deploy:** revisión de políticas, validación por OPA/Kyverno.
- **Run:** monitoreo de seguridad, detección de anomalías, control de tráfico y alertas.

Métricas relevantes en DevSecOps

- Tiempo medio de detección y corrección de vulnerabilidades.
- Porcentaje de builds fallidas por errores de seguridad.
- Tasa de cumplimiento de políticas (e.g., imágenes escaneadas, secretos gestionados correctamente).
- Nº de violaciones bloqueadas por políticas preventivas.

Rol del DevOps senior

El DevOps senior debe:

- Integrar herramientas de seguridad en todo el flujo de trabajo sin sacrificar agilidad.
- Diseñar pipelines seguros y auditables.
- Coordinar con equipos de seguridad para establecer estándares y reglas dinámicas.
- Promover la madurez de seguridad como parte de la cultura DevOps, impulsando la responsabilidad compartida y la mejora continua.

Capítulo 2: Snyk, Trivy, Checkov, Vault.

En una arquitectura DevOps moderna, la automatización de seguridad requiere herramientas especializadas que puedan actuar de forma autónoma dentro del ciclo CI/CD. Este contenido aborda cuatro de las herramientas más relevantes para una estrategia DevSecOps integral:

- Snyk (análisis de vulnerabilidades en dependencias y contenedores),
- Trivy (escaneo de imágenes, infraestructura y código),
- Checkov (validación de IaC según políticas de seguridad),
- Vault (gestión segura de secretos y credenciales).

Cada una cubre aspectos específicos del ciclo DevOps y puede integrarse de forma nativa en pipelines CI/CD y flujos GitOps.

1. Snyk

Snyk es una plataforma SaaS especializada en el análisis de seguridad de código abierto, contenedores, IaC y configuraciones de nube. Su enfoque está en facilitar la detección y corrección rápida de vulnerabilidades, con integración directa a repositorios y entornos de desarrollo.

Funcionalidades:

- Análisis de dependencias en tiempo real (Java, Node.js, Python, etc.).
- Escaneo de imágenes de contenedores en CI/CD.
- Recomendaciones de actualización automática.
- Soporte para Kubernetes, Terraform, CloudFormation.

Integraciones:

- GitHub, GitLab, Bitbucket.
- Jenkins, CircleCI, GitHub Actions.
- IDEs como Visual Studio Code.

2. Trivy

Trivy, desarrollado por Aqua Security, es una herramienta de código abierto que permite el escaneo de seguridad en múltiples capas del stack DevOps, destacando por su rapidez, portabilidad y facilidad de integración.

Funcionalidades:

- Escaneo de vulnerabilidades en imágenes Docker.
- Revisión de configuraciones de Kubernetes e IaC (Terraform, Helm).
- Análisis de licencias de paquetes.
- Soporte para SBOM (Software Bill of Materials).

Uso típico:

- trivy image myapp:latest para detectar CVEs en contenedores.
- trivy config . para auditar políticas en manifiestos y plantillas.

Ventajas:

- Bajo tiempo de escaneo.
- Integración simple en pipelines.
- Compatibilidad con GitOps y repositorios Git.

3. Checkov

Checkov, mantenido por Bridgecrew (parte de Palo Alto Networks), es una herramienta para auditar la seguridad de Infraestructura como Código (IaC). Analiza archivos Terraform, CloudFormation, Kubernetes y ARM, buscando configuraciones que no cumplen con buenas prácticas o estándares.

Funcionalidades:

- Escaneo de IaC en repositorios y pipelines.
- Más de 1000 políticas predefinidas (por ejemplo: evitar recursos públicos, detectar secretos duros).
- Soporte para políticas personalizadas (YAML o Python).
- Compatibilidad con plataformas multi-cloud (AWS, Azure, GCP).

Integraciones:

- GitHub, GitLab, Jenkins, CircleCI.
- **GitOps workflows:** ejecución automática en PRs o merges.

4. Vault

Vault, desarrollado por HashiCorp, es un sistema de gestión de secretos de nivel empresarial. Su función es almacenar, acceder y controlar dinámicamente credenciales, tokens, API keys, certificados y otros datos sensibles, bajo políticas de acceso altamente definidas.

Funcionalidades:

- Almacenamiento seguro de secretos con control RBAC.
- Rotación automática de claves y credenciales (AWS, PostgreSQL, MongoDB, etc.).
- Emisión dinámica de certificados y tokens JWT.
- Autenticación vía Kubernetes, GitHub, LDAP, tokens.

Casos de uso:

- Montar secretos como variables de entorno en aplicaciones.
- Integrar con pipelines para inyectar claves durante el build.
- Eliminar la necesidad de almacenar secretos en el repositorio.

Integraciones:

- Kubernetes (via Sidecar Injector, CSI driver).
- Terraform, Jenkins, ArgoCD.
- CLI, API, y SDKs en múltiples lenguajes.

Comparativa funcional estratégica

Herramientas	Propósito principal	Fase DevSecOps
Snyk	Análisis de dependencias y contenedores	CI / Desarrollo
Trivy	Escaneo de imágenes e infraestructura	CI / CD
Checkov	Validación de IaC contra políticas	Planificación / Build
Vault	Gestión de secretos y credenciales	Build / Run

Enfoque del DevOps senior

El profesional senior debe:

- Orquestar estas herramientas dentro del pipeline CI/CD y flujos GitOps.
- Definir reglas de seguridad alineadas a políticas organizacionales.
- Asegurar la trazabilidad de escaneos, alertas y respuestas ante riesgos.
- Automatizar alertas, correcciones y actualizaciones seguras.
- Implementar validación bloqueante en PRs o builds inseguros.

Capítulo 3: Seguridad en Kubernetes.

Kubernetes es hoy el sistema de orquestación de contenedores más utilizado en entornos productivos. Su adopción masiva ha traído importantes desafíos de seguridad, ya que al tratarse de un sistema distribuido con múltiples componentes y capas, su superficie de ataque es amplia. La seguridad en Kubernetes implica proteger el clúster, los contenedores, los datos, las comunicaciones internas, las credenciales y las aplicaciones que se ejecutan en él, bajo un enfoque de defensa en profundidad.

Principios de seguridad en Kubernetes

1. **Mínimos privilegios:** todo debe ejecutarse con los menores permisos posibles (principio de least privilege).

2. **Inmutabilidad y declaratividad:** los recursos deben ser inmutables y auditables mediante código (IaC).

3. **Aislamiento de cargas de trabajo:** separar entornos, namespaces y responsabilidades.

4. **Validación de configuración:** prevenir errores de configuración que expongan vulnerabilidades.

5. **Observabilidad continua:** auditar, monitorear y generar alertas ante comportamientos sospechosos.

Áreas críticas de protección

1. Control de acceso y autenticación

- Integrar RBAC (Role-Based Access Control) para limitar accesos a APIs y recursos.
- Usar mecanismos seguros de autenticación (OIDC, certificados, tokens rotativos).
- Auditar solicitudes a la API Server mediante logs de acceso y eventos.

2. Aislamiento y namespaces

- Separar entornos y servicios mediante namespaces.
- Limitar el uso de recursos por pod a través de ResourceQuota y LimitRange.
- Usar Network Policies para segmentar tráfico pod a pod, evitando conexiones innecesarias.

3. Seguridad en pods y contenedores

- Prohibir ejecución como root (runAsNonRoot: true).
- Forzar sistemas de archivos de solo lectura (readOnlyRootFilesystem: true).
- Usar imágenes firmadas y escaneadas por herramientas como Trivy o Snyk.
- Aplicar PodSecurity Standards (baseline, restricted) o migrar a PodSecurityAdmission.

4. Validación y políticas

- Integrar OPA/Gatekeeper o Kyverno para rechazar manifiestos inseguros.
- **Ejemplo:** denegar pods que usen hostNetwork, hostPID, hostPath, o capacidades elevadas.
- Validar configuraciones mediante escáneres como Checkov o kubescape.

5. Protección del plano de control

- Restringir acceso al API server desde redes públicas.
- Aislar nodos maestros del tráfico externo y no autorizado.
- Monitorear las métricas del API server y eventos en tiempo real.

6. Protección del tráfico y comunicaciones

- Usar TLS para todas las conexiones entre componentes (etcd, kubelet, controller).
- Activar Mutual TLS (mTLS) entre servicios mediante service mesh como Istio o Linkerd.
- Auditar tráfico saliente para detectar conexiones maliciosas o exfiltración de datos.

7. Gestión de secretos

- No codificar secretos en manifiestos o variables de entorno.
- Usar Kubernetes Secrets cifrados con KMS o External Secrets desde Vault o AWS/GCP.
- Aplicar políticas para rotación de secretos y expiración automática.

Herramientas complementarias

- **Kube-bench:** auditoría de cumplimiento con CIS Benchmark.
- **Kube-hunter:** detección de vulnerabilidades en el clúster.
- **Falco:** monitoreo en tiempo real de eventos sospechosos en tiempo de ejecución.
- **Audit logs + Prometheus/Grafana:** análisis de comportamiento anómalo y telemetría.

Métricas clave en seguridad de Kubernetes

- % de pods ejecutándose con privilegios restringidos.
- Nº de cargas de trabajo que cumplen políticas de seguridad.
- Tiempo medio de detección de imágenes vulnerables.
- Tasa de cumplimiento de políticas OPA/Kyverno en PRs o despliegues.

Rol del DevOps senior

El DevOps senior debe ser capaz de:

- Diseñar entornos Kubernetes seguros desde su base (infraestructura, clúster, workloads).
- Establecer políticas de seguridad como código, integradas en el pipeline.
- Asegurar que todos los manifiestos estén validados antes del despliegue.
- Supervisar auditorías y adaptar la arquitectura según el nivel de exposición o criticidad del sistema.
- Promover una cultura de "seguridad continua" entre desarrolladores, operadores y arquitectos.