{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

EMPLEAR METODOS DE DISEÑO DE PLATAFORMAS INTERNAS EN DEVELOPERS, DE ACUERDO A LAS PRACTICAS AVANZADAS DE GITOPS, DEVSECOPS, KUBERNETES, OBSERVABILIDAD, IAC, FINOPS Y AIOPS.

Contenidos

# Módulo 8: PLATFORM ENGINEERING & INTERNAL DEVELOPER PLATFORMS (IDP).

# Capítulo 1: Backstage

Backstage es una plataforma de desarrollo interno (IDP: Internal Developer Platform) de código abierto creada por Spotify y actualmente mantenida por la Cloud Native Computing Foundation (CNCF). Su propósito principal es centralizar herramientas, documentación, servicios y flujos DevOps en una única interfaz autogestionada y extensible, permitiendo a los desarrolladores concentrarse en escribir código en lugar de buscar recursos, entender integraciones o lidiar con procesos manuales.

Backstage se ha convertido en una herramienta clave para organizaciones que buscan estandarizar la experiencia de desarrollo, acelerar el onboarding, mejorar la trazabilidad de servicios y escalar sus prácticas DevOps.

## ¿Qué es Backstage?

Backstage es una plataforma modular que funciona como un portal central de autoservicio para equipos de desarrollo. Actúa como una interfaz unificada para:

- Crear servicios, paquetes, librerías, pipelines, infraestructuras y bases de datos desde plantillas prediseñadas.
- Documentar y descubrir servicios registrados.
- Integrar métricas, estado de CI/CD, seguridad y cumplimiento.
- Explorar dependencias, ownership, despliegues y cambios.

La plataforma se instala generalmente como una aplicación web Node.js/React que los equipos de plataforma despliegan internamente y personalizan según sus necesidades.

## Componentes clave de Backstage

Backstage se construye sobre una arquitectura extensible basada en plugins, permitiendo que la plataforma crezca y se adapte a distintos ecosistemas.

Los principales componentes son:

- **Software Catalog:** núcleo del sistema. Un catálogo de servicios, librerías, herramientas, pipelines y recursos que permite registrar, descubrir y documentar los activos de la organización.
- **Scaffolder:** sistema de plantillas que permite crear recursos de forma automática y guiada (infraestructura, servicios, paquetes, etc.).
- **TechDocs:** sistema de documentación técnica basado en Markdown, integrado directamente en la plataforma.
- **Plugins de terceros:** integraciones con Kubernetes, GitHub, GitLab, Jenkins, ArgoCD, Prometheus, PagerDuty, Sentry, y más.
- **Backstage App:** el frontend y backend que orquesta la experiencia unificada de usuario.

### 3. Software Catalog: trazabilidad de servicios

El Software Catalog es el corazón operativo de Backstage. Cada servicio o componente se representa como una entidad declarada en YAML, con metadata estandarizada:

```yaml
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: backend-orders
  description: Servicio de órdenes del e-commer
spec:
  type: service
  lifecycle: production
  owner: equipo-backend
```

**Estas entidades permiten:**

- Descubrir todos los servicios activos
- Ver qué equipo es responsable
- Navegar dependencias
- Acceder al código fuente, pipelines, documentación y métricas desde un único lugar

Este catálogo mejora la visibilidad organizacional y reduce la fricción al trabajar en entornos complejos.

## Scaffolder: creación guiada de recursos

El componente Scaffolder permite generar nuevos recursos usando plantillas definidas por el equipo de plataforma. Por ejemplo, crear un nuevo microservicio basado en Node.js o una nueva infraestructura en Terraform.

**Este sistema reduce errores humanos y acelera el time-to-market, mediante flujos como:**

**Selección de plantilla**

- Ingreso de parámetros (nombre, equipo, lenguaje)
- Generación de código y estructura
- Registro automático en el catálogo

Además, el Scaffolder puede integrarse con GitOps y CI/CD, generando automáticamente repositorios, pipelines, y configuración de despliegue.

## TechDocs: documentación integrada

Backstage incluye un sistema de documentación basado en Markdown, compatible con MkDocs y alojado directamente en los repositorios.

Cada componente del catálogo puede incluir su propia documentación (docs/), la cual es renderizada y visible directamente desde la interfaz de Backstage, permitiendo:

- Consolidar documentación por servicio o componente
- Establecer estándares documentales transversales
- Facilitar el onboarding de nuevos desarrolladores
- Integrar documentación técnica con herramientas como Swagger, OpenAPI, Postman, etc.

TechDocs elimina la necesidad de plataformas externas fragmentadas para documentación interna.

## Integración con herramientas DevOps

Backstage no reemplaza herramientas existentes, sino que las unifica en una única experiencia. Se integra de forma nativa o mediante plugins con:

- **Repositorios:** GitHub, GitLab, Bitbucket
- **CI/CD:** Jenkins, CircleCI, GitHub Actions, ArgoCD
- **Observabilidad:** Prometheus, Grafana, Sentry, Datadog
- **Orquestadores:** Kubernetes, OpenShift, ECS
- **Monitoreo y alertas:** PagerDuty, Opsgenie

Esto permite que los desarrolladores consulten el estado de sus builds, despliegues, métricas y alertas sin salir de Backstage.

## Beneficios organizacionales

Implementar Backstage proporciona beneficios técnicos y organizacionales clave:

- Estandariza la creación de servicios y recursos
- Centraliza documentación, ownership y métricas
- Acelera el onboarding de nuevos equipos
- Promueve la cultura de autoservicio DevOps
- Reduce la dependencia de operadores o equipos SRE
- Mejora la gobernanza y trazabilidad de cambios

En empresas con múltiples squads, microservicios y entornos cloud, Backstage se convierte en el portal de acceso único a toda la operación técnica.

# Consideraciones para producción

Para usar Backstage en producción, se debe considerar:

- Control de autenticación y autorización (OIDC, GitHub OAuth, SAML)
- Integración con DNS y certificados TLS
- Mantenimiento del catálogo mediante pipelines o flujos de revisión
- Escalabilidad de backend y base de datos (SQLite, PostgreSQL)
- Mantenimiento de plugins y actualizaciones del core
- Uso de scaffolder con GitOps y creación de recursos bajo revisión

Estas prácticas permiten mantener una instancia segura, confiable y adaptada a la cultura organizacional.

Backstage es más que una herramienta de visualización: es una plataforma estratégica de aceleración del desarrollo interno. Al combinar catalogación, documentación, generación automatizada y visibilidad de flujos DevOps, transforma la experiencia del desarrollador en grandes organizaciones.

Cuando se implementa correctamente, Backstage se convierte en el centro neurálgico de la ingeniería moderna, estandarizando procesos, aumentando la productividad, reduciendo la complejidad operativa y fomentando la autonomía de los equipos.

# Capítulo 2: GitOps Stacks.

En el contexto DevOps moderno, el enfoque GitOps ha transformado la forma en que se gestiona el ciclo de vida de aplicaciones e infraestructura. Cuando se amplía este enfoque a nivel organizacional, se hace necesario estructurar el despliegue y gestión de recursos complejos en stacks GitOps, permitiendo controlar múltiples entornos, clústeres, microservicios, módulos de infraestructura y pipelines desde repositorios Git.

Los GitOps Stacks representan una evolución del patrón GitOps, donde los distintos dominios del sistema —infraestructura, aplicaciones, configuración, seguridad y observabilidad— son controlados de forma declarativa, desacoplada y orquestada.

## ¿Qué son los GitOps Stacks?

Un GitOps Stack es una agrupación lógica y técnica de recursos que se gestiona enteramente desde Git como fuente de verdad, mediante archivos declarativos, pipelines automatizados y agentes de sincronización.

**Cada stack puede representar:**

- Un entorno completo (ej: dev, staging, prod)
- Un clúster Kubernetes
- Un conjunto de microservicios relacionados
- Una capa funcional del sistema (infraestructura base, CI/CD, monitoreo, seguridad)
- Una región geográfica o segmento de negocio

La clave está en que cada stack es autosuficiente, declarativo, versionado y desplegable automáticamente mediante herramientas GitOps como ArgoCD o Flux.

## Arquitectura y composición de un GitOps Stack

**Un stack GitOps se define por:**

- Un repositorio o carpeta que contiene todos los manifiestos YAML, Helm charts o configuraciones Terraform necesarias.
- Variables, secretos y configuraciones específicas por stack.
- Un sistema de sincronización continua desde Git hacia el clúster o entorno de destino.
- Control de versiones explícito, revisiones con pull requests y auditoría Git.

**Por ejemplo, el stack infra-prod puede contener:**

- Recursos de red (VPC, subredes)
- Clústeres EKS/GKE/AKS
- Almacenamiento (buckets, bases de datos)
- Controladores de ingreso (Ingress, gateways)
- Configuración de monitoring/logging
- Agentes de seguridad o tracing

Todo esto gestionado desde Git, con flujos automatizados de plan, apply, deploy y validaciones previas.

## Separación por capas y dominios

Una práctica recomendada es dividir los stacks por capas o dominios funcionales, permitiendo que diferentes equipos se encarguen de diferentes partes del sistema sin interferencia directa.

**Ejemplos de capas de stacks:**

- bootstrap/ → stack base del clúster (cert-manager, ingress controller, csi-driver)
- infrastructure/ → red, seguridad, almacenamiento, cloud resources
- platform/ → herramientas compartidas: ArgoCD, Prometheus, Vault, Grafana, Tekton
- applications/ → servicios de negocio, microservicios, APIs
- observability/ → trazabilidad, métricas, dashboards
- compliance/ → escáneres, validadores, políticas

Cada capa se controla mediante su propio flujo GitOps, permitiendo trazabilidad, rollback y despliegues desacoplados.

## Herramientas clave en GitOps Stacks

La implementación de stacks GitOps requiere un conjunto de herramientas bien orquestadas:

- **ArgoCD / Flux:** sincronización declarativa de manifiestos y Helm charts desde Git a Kubernetes.
- **Helm / Kustomize:** plantillas reutilizables por entorno.
- **Terraform + GitOps Bridge (ArgoCD Terraform Plugin, Atlantis, Spacelift):** para stacks de infraestructura fuera de Kubernetes.
- **Sealed Secrets / SOPS / Vault:** gestión segura de credenciales y variables sensibles.
- Pre-commit hooks / linters: validación antes del push (kubeval, conftest, terraform validate).
- **CI pipelines:** para ejecutar validaciones, planes y PRs antes de aplicar cambios.
- **Argo Workflows / Tekton:** para orquestar pipelines de despliegue complejos declarados como código.

Estas herramientas trabajan en conjunto para permitir que cualquier modificación al repositorio active un cambio planificado, trazado y aprobado sobre la infraestructura o las aplicaciones.

## Gestión multientorno con GitOps Stacks

En organizaciones que operan múltiples entornos (dev, qa, prod, sandbox), es fundamental estructurar los stacks para evitar colisiones, garantizar seguridad y mantener consistencia.

**Cada entorno puede tener:**

- Su propia rama (main, staging, dev)
- Su propio subdirectorio (environments/prod/, environments/dev/)
- Su propio repositorio (git@org/stack-prod.git)

Además, cada stack puede ser observado por un agente de sincronización (ej. ArgoCD AppSet) que actualiza el entorno según la versión declarada, respetando el ciclo GitOps de validación, revisión y aplicación automática.

## GitOps para infraestructura y aplicaciones

Los GitOps Stacks se aplican tanto para infraestructura como para aplicaciones:

- **Infraestructura declarativa:** VPCs, bases de datos, clústeres, almacenamiento, servicios en la nube. Se puede usar Terraform, Pulumi o Crossplane con sincronización GitOps.

- **Aplicaciones y microservicios:** Desplegados mediante manifiestos YAML, Helm charts, Kustomize, y sincronizados con herramientas GitOps nativas de Kubernetes.

Este enfoque permite orquestar la infraestructura base y las aplicaciones desde un único plano de control: Git como fuente de verdad, ArgoCD o Flux como ejecutores, y el equipo de plataforma como facilitador.

## Beneficios de usar GitOps Stacks

La adopción de GitOps Stacks trae beneficios tanto operativos como organizacionales:

- Permite escalar el modelo GitOps a toda la organización.

- Establece una arquitectura clara por capas, equipos y entornos.

- Mejora la gobernanza de infraestructura y servicios.

- Facilita el rollback y la comparación de estados históricos.

- Disminuye la dependencia de accesos directos a clústeres o cloud providers.

- Permite aplicar control de cambios, revisiones y validaciones por equipo.

- Promueve la colaboración en infraestructura como código, siguiendo el mismo modelo que se usa para desarrollo de software.

## Consideraciones prácticas

Para implementar GitOps Stacks de manera efectiva es importante:

- Diseñar una convención de estructuras y naming común a todos los repositorios.
- Integrar control de políticas y validaciones antes de aplicar.
- Mantener una gestión robusta de secretos.
- Versionar los componentes desplegados (charts, manifests, módulos).
- Tener dashboards de observabilidad sobre el estado de los stacks (salud, sincronización, diferencias).
- Definir claramente los ownerships por stack, capa y entorno.

Además, es recomendable realizar pruebas de drift (deriva de configuración) para detectar discrepancias entre lo definido en Git y lo que realmente está desplegado.

GitOps Stacks representan una evolución natural y escalable del modelo GitOps, permitiendo que organizaciones complejas gestionen infraestructura, servicios y operaciones desde Git, con control, automatización, seguridad y trazabilidad. Este patrón no solo optimiza la entrega continua de software e infraestructura, sino que redefine la colaboración entre equipos de desarrollo, operaciones y plataforma, alineándolos bajo una misma fuente de verdad y un mismo flujo de trabajo.

# Capítulo 3: Integración con CI/CD y observabilidad.

Una plataforma DevOps madura no se limita a gestionar despliegues; debe además garantizar la trazabilidad, el monitoreo y el control del ciclo de vida completo de software, desde el commit hasta el tiempo de ejecución en producción. Para ello, la integración entre CI/CD (Integración y Entrega Continua) y los sistemas de observabilidad es fundamental.

Esta integración permite que cada cambio en el sistema —sea código, infraestructura o configuración— se valide automáticamente, se despliegue bajo criterios controlados, y quede completamente trazado, medido y monitoreado.

## Integración CI/CD como parte de la plataforma

Los sistemas CI/CD ya no son pipelines aislados por proyecto. En plataformas modernas, el CI/CD se convierte en una extensión programable y estandarizada de los flujos DevOps, integrada al ecosistema GitOps, al catálogo de servicios y a los procesos de observabilidad.

**La integración de CI/CD en una plataforma unificada busca:**

- Automatizar pruebas, builds y validaciones de seguridad en cada PR.
- Generar artefactos y publicarlos en registries de forma controlada.
- Desencadenar despliegues declarativos vía GitOps (ej. push a ramas de despliegue).
- Aplicar gates de aprobación automática o manual en función de criterios definidos (status, métricas, políticas).
- Asociar cada build y deploy a un conjunto de métricas y trazas visibles.

**El pipeline ya no solo construye:** es el primer validador de calidad, cumplimiento y trazabilidad.

## CI/CD desacoplado y programable

Las plataformas modernas utilizan CI/CD programable, desacoplado del repositorio y controlado desde Git. Este patrón permite que:

- Las definiciones de pipeline estén versionadas y revisadas.
- Cada tipo de servicio use plantillas reutilizables de CI/CD.
- El pipeline esté alineado al catálogo (ej. Backstage), usando metadatos del componente.
- El sistema aplique políticas automáticas de pruebas, escaneo y validación.

Este enfoque permite a los equipos incorporar nuevos servicios sin necesidad de crear pipelines desde cero. El equipo de plataforma define un conjunto de flujos genéricos adaptables (por tipo de aplicación, lenguaje, entorno o criticidad).

## Desencadenamiento GitOps desde CI

En modelos GitOps, el CI no aplica directamente los cambios, sino que:

1. Valida y construye los artefactos (por ejemplo, una imagen Docker).
2. Publica el artefacto en un registry versionado.
3. Realiza un pull request o push sobre un repositorio GitOps con manifiestos actualizados.
4. El sistema GitOps (ej. ArgoCD, Flux) detecta el cambio y aplica el despliegue.

Este modelo garantiza que todo lo que se ejecuta esté definido en Git y pueda auditarse, revertirse o replicarse en otros entornos.

## Integración con observabilidad en el ciclo completo

El CI/CD moderno se conecta con los sistemas de observabilidad desde el inicio del proceso, permitiendo:

- Inyectar etiquetas y metadatos en logs y trazas para cada build.
- Asociar un despliegue con sus métricas clave (latencia, errores, throughput).
- Activar dashboards automáticos post-deploy.
- Comparar comportamientos antes y después de cada cambio (análisis comparativo).
- Automatizar decisiones de rollback si los umbrales de observabilidad se ven comprometidos.

**Por ejemplo, al desplegar una nueva versión de un servicio, el pipeline puede:**

- Validar que no haya incremento de errores 5xx por sobre el umbral aceptable.
- Analizar los valores p95 de latencia en los 5 minutos post-deploy.
- Confirmar que el servicio fue registrado correctamente en el catálogo y monitoreado.

## Instrumentación automatizada desde CI/CD

Una práctica avanzada es automatizar la instrumentación de servicios y pipelines desde el mismo flujo de CI/CD. Esto incluye:

- Registrar los nuevos servicios en el catálogo (Backstage u otra plataforma).
- Inyectar anotaciones en el despliegue para activar la recolección de métricas y trazas (ej. prometheus.io/scrape: true).
- Configurar alertas iniciales (ej. Uptime, CPU, errores) asociadas al servicio desplegado.
- Enviar eventos a sistemas de auditoría o control de cambios (ej. slack, teams, audit logs).
- Activar dashboards temporales de observabilidad para evaluar el impacto de cambios.

Estas acciones permiten que cada pipeline deje una huella observable, facilitando la correlación entre cambios de código y eventos operacionales.

## Alertas y políticas conectadas al ciclo de entrega

En entornos avanzados, el sistema de observabilidad no solo mide, sino que también condiciona la ejecución del pipeline. Por ejemplo:

- Si un servicio tiene incidentes activos, se bloquean nuevos despliegues.
- Si el servicio no emite trazas, se invalida el paso a producción.
- Si no se cumple una política de seguridad (scan fallido), se detiene el proceso.
- Si el error rate supera el umbral definido, se activa un rollback automático.

Estos flujos se implementan mediante gates condicionales, hooks y validadores embebidos en la definición del pipeline, que se comunican con sistemas de observabilidad como Prometheus, Grafana, Datadog, Sentry, Jaeger, etc.

## Beneficios de la integración CI/CD + observabilidad

La conexión entre entrega continua y observabilidad genera una plataforma altamente confiable, donde:

- Cada cambio puede rastrearse desde el código hasta su impacto en producción.
- Las decisiones de despliegue se basan en datos, no en percepciones.
- Se detectan y mitigan problemas antes de que escalen.
- Se refuerzan las buenas prácticas de ingeniería desde el inicio del proceso.
- Se fortalece la seguridad, gobernanza y rendimiento general del sistema.

Esto se traduce en mayor velocidad de entrega con menor riesgo, una de las promesas clave de DevOps moderno.

La integración entre CI/CD y observabilidad no es opcional en una plataforma DevOps madura. Representa el puente entre el cambio y el impacto, entre la intención y el comportamiento real. Al conectar cada build, deploy y rollback con sistemas de trazabilidad, métricas y eventos, se construye una plataforma donde los equipos no solo despliegan más rápido, sino que también entienden y controlan lo que despliegan.

Esta conexión es el núcleo operativo de organizaciones modernas, donde cada commit puede tener un efecto productivo inmediato, medido, validado y reversible en cualquier momento.