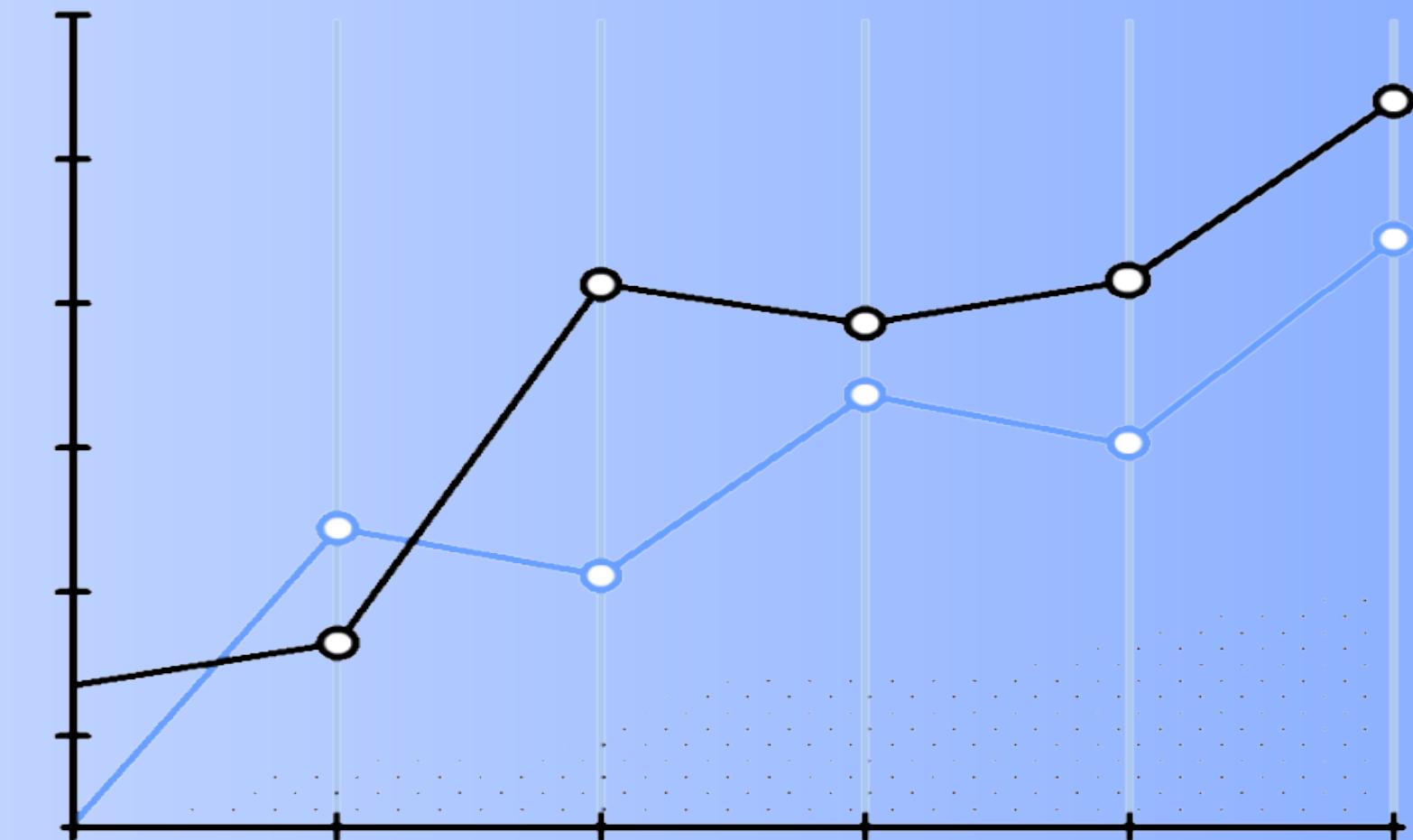




Liceria Tech

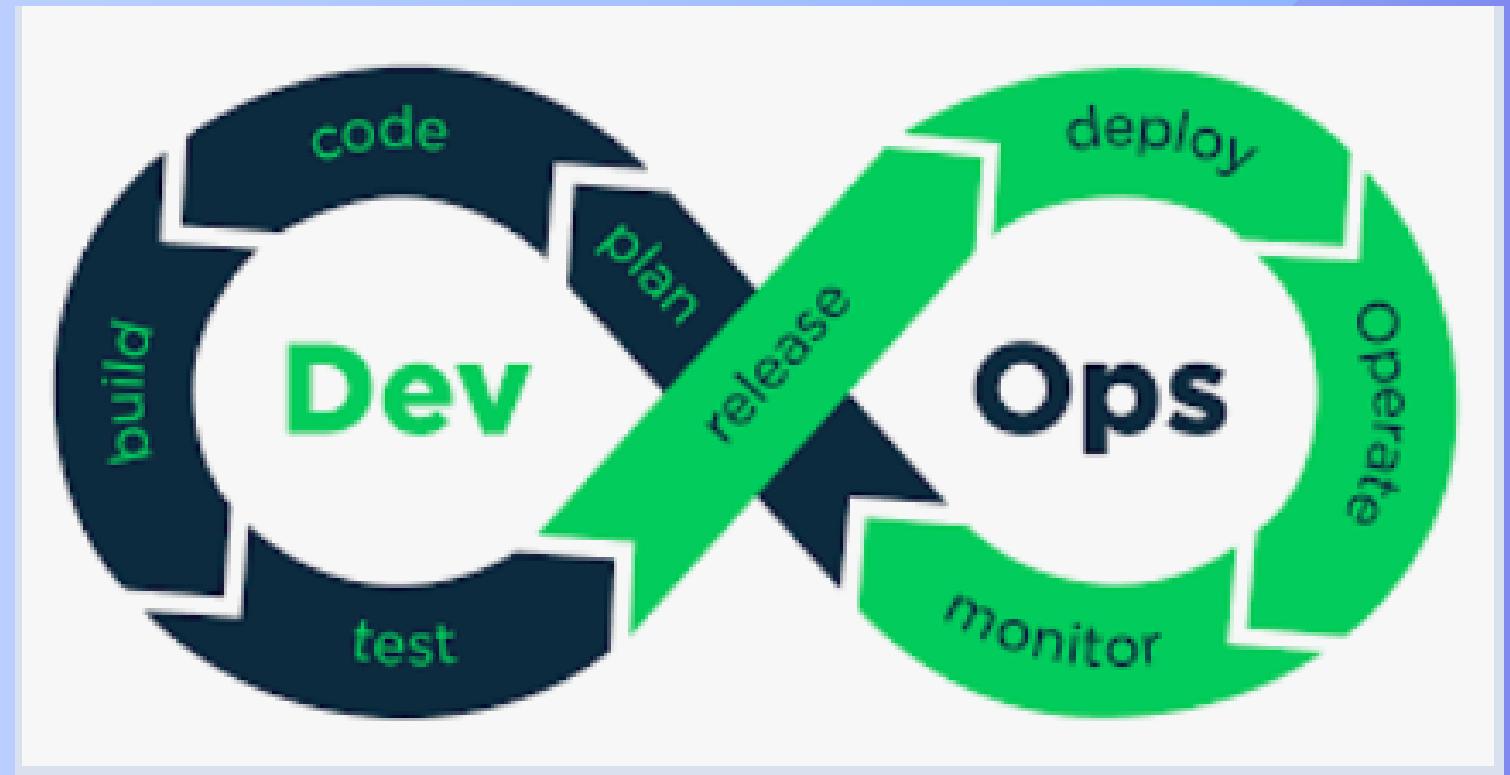
ACTIVIDAD 1

Bianca Merchan Torres
Jesus Diego Santa Cruz Basilio
Daren Herrera Romo



¿QUÉ ES DEVOPS?

DevOps es una metodología que busca unir los equipos de desarrollo, aseguramiento de calidad (QA) y operaciones en un proceso colaborativo continuo, para reducir el tiempo de los ciclos de entrega, mejorar la calidad y responder de manera más ágil a los cambios en el entorno y necesidades del negocio.



HISTORIA Y ANTECEDENTES DE DEVOPS

DevOps apareció como una respuesta a los modelos recurrentes de desarrollo de software, donde los equipos de desarrollo y operaciones trabajaban de manera individual, lo que generaba ciclos largos de desarrollo y problemas de comunicación. Con DevOps se dio mas importancia a la colaboración, acortando el tiempo entre la concepción de una idea y su implementación en producción.



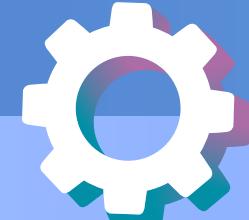
Diferencias entre los equipos de desarrollo y operaciones en el pasado.



Liceria Tech



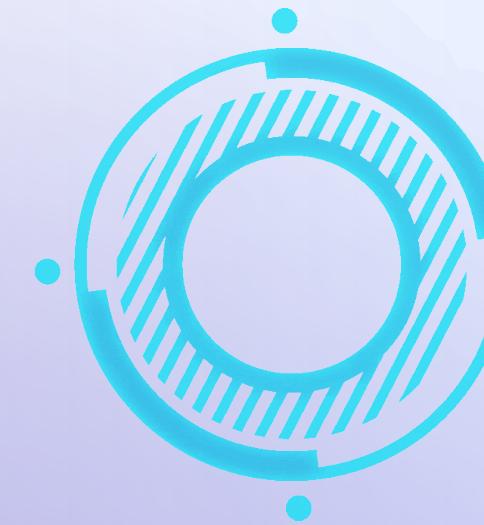
En los modelos tradicionales, los equipos de desarrollo, aseguramiento de calidad (QA) y operaciones trabajaban de forma autosuficiente. El equipo de desarrollo diseñaba y creaba el software, el equipo de QA realizaba pruebas exhaustivas, y el equipo de operaciones se encargaba del despliegue y mantenimiento. Este enfoque generaba una división entre roles y generaba problemas de comunicación, lo que alargaba los ciclos de desarrollo y complicaba la generación de respuesta ante cambios y errores.



PRINCIPIOS FUNDAMENTALES DE DEVOPS

- **Enfoque en el cliente:** La mejora en la entrega de software, con el cliente como el centro del proceso.
- **Equipos autónomos y multifuncionales:** Equipos que combinan disciplinas (desarrollo, QA, operaciones) para permitir un flujo de trabajo más ágil y flexible.
- **Mejora continua:** Un enfoque en la evaluación constante de procesos y resultados, tanto en el ámbito técnico como cultural, para realizar ajustes y optimizaciones.
- **Automatización:** Implementación de herramientas y procesos que mecanizan tareas frecuentes (como integración y entrega continua), disminuyendo la intervención manual y aumentando la eficiencia.



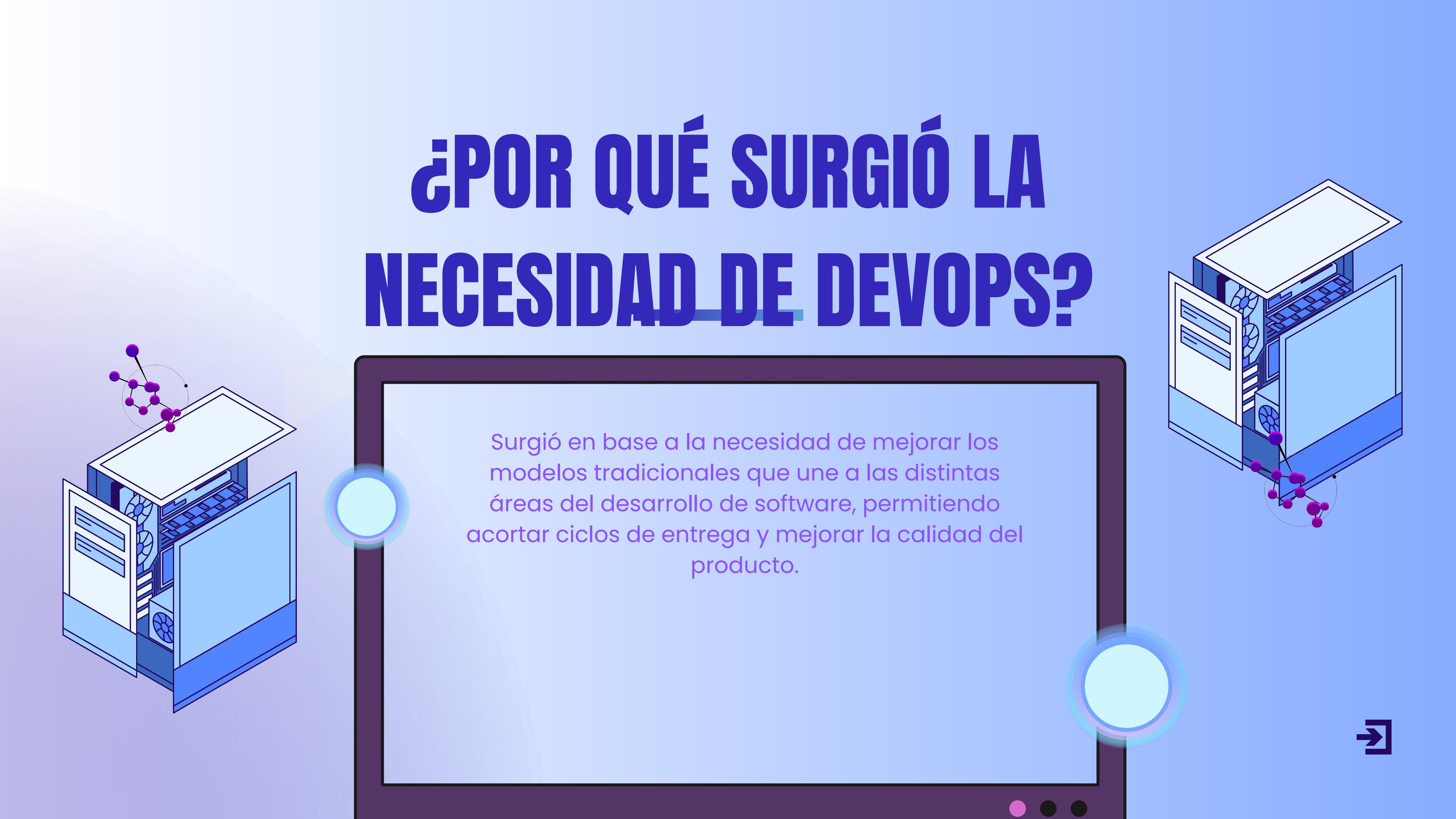


QUE NO ES DEV OPS

DevOps no es solo un conjunto de herramientas, individuos o procesos aislados, sino de una transformación cultural que involucra a toda la organización. DevOps no es una solución mágica ni un proceso automático; implica un cambio profundo en la forma de trabajar, comunicar y colaborar entre los equipos.



¿POR QUÉ SURGIÓ LA NECESIDAD DE DEVOPS?



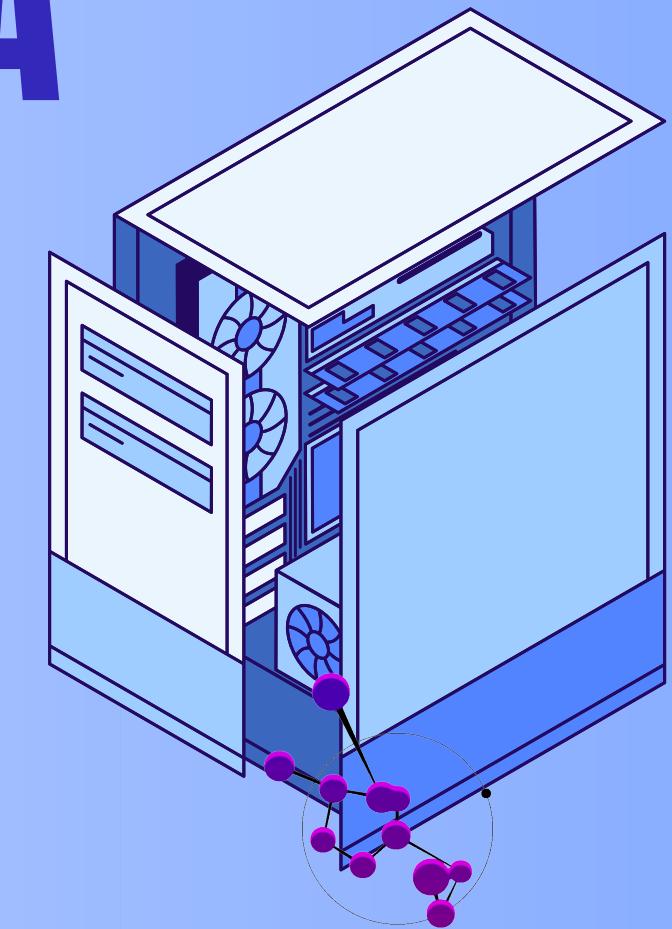
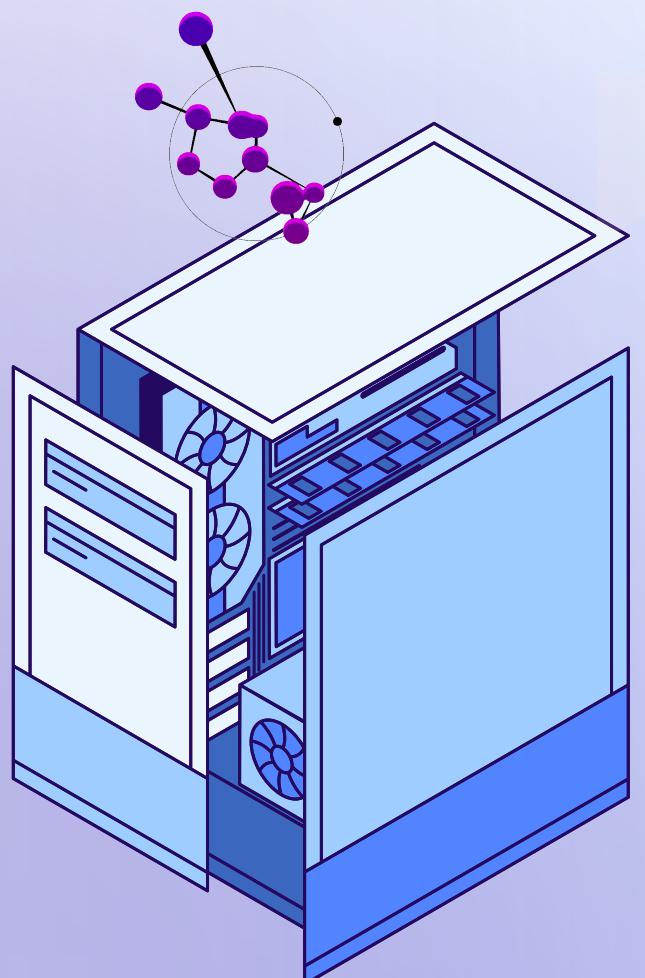
Surgió en base a la necesidad de mejorar los modelos tradicionales que une a las distintas áreas del desarrollo de software, permitiendo acortar ciclos de entrega y mejorar la calidad del producto.



¿CÓMO LA FALTA DE COMUNICACIÓN Y COORDINACIÓN LLEVÓ A LA CREACIÓN DE DEVOPS?

Porque se alarga considerablemente el ciclo de actualización debido a que son procesos secuenciales:

- Diseño y desarrollo
- Pruebas del equipo de QA
- Despliegue y mantenimiento.



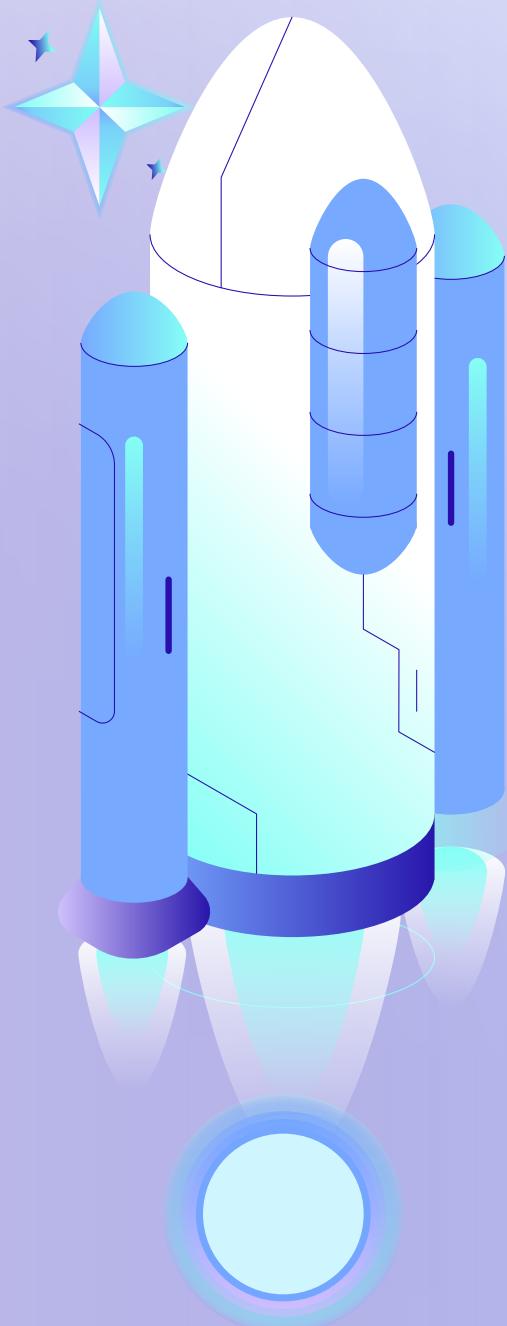
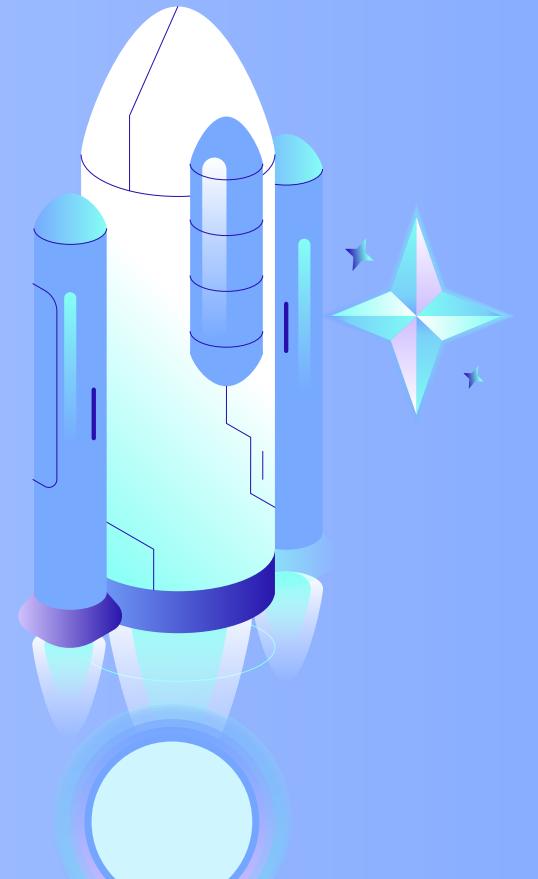


¿CÓMO EL PRINCIPIO DE MEJORA CONTINUA IMPACTA TANTO EN LOS ASPECTOS TÉCNICOS COMO EN LOS CULTURALES DE UNA ORGANIZACIÓN?

A nivel técnico, implica la optimización constante de pipelines de integración y entrega continua, la automatización de tareas y la resolución de problemas rápidamente. A nivel cultural, significa fomentar una mentalidad de aprendizaje y adaptación, donde cada miembro del equipo se siente responsable de la calidad y mejora del proceso general.

¿QUÉ SIGNIFICA QUE DEVOPS NO SE TRATA SOLO DE HERRAMIENTAS, INDIVIDUOS O PROCESOS?

Esto significa que DevOps no debe verse únicamente como la adopción de tecnologías específicas, ni como la implementación de ciertos procesos. DevOps es una transformación organizacional que busca cambiar la forma en que los equipos interactúan y colaboran. Implica un cambio cultural en cómo se gestionan las responsabilidades, se comunican los equipos y se toman decisiones de forma conjunta. No se trata solo de usar las herramientas adecuadas, sino de trabajar juntos y con un enfoque en la mejora continua.





¿CÓMO CONTRIBUYEN LOS EQUIPOS AUTÓNOMOS Y MULTIFUNCIONALES A UNA IMPLEMENTACIÓN EXITOSA DE DEVOPS?

INTRODUCCIÓN A LA IAC

LA IAC ES UN ENFOQUE QUE PERMITE GESTIONAR LA INFRAESTRUCTURA MEDIANTE ARCHIVOS DE CONFIGURACIÓN EN LUGAR DE REALIZAR CONFIGURACIONES MANUALES A TRAVÉS DE INTERFACES GRÁFICAS O HERRAMIENTAS DE ADMINISTRACIÓN TRADICIONALES. LA DIFERENCIA PRINCIPAL ENTRE IAC Y EL MANUAL DE CONFIGURACIÓN ES QUE EN IAC LA INFRAESTRUCTURA SE GESTiona COMO CÓDIGO, PERMITIENDO LA AUTOMATIZACIÓN COMPLETA DEL CICLO DE VIDA DE LOS RECURSOS. ESTO NO SOLO MEJORA LA EFICIENCIA, SINO QUE REDUCE SIGNIFICATIVAMENTE LOS ERRORES HUMANOS,

ESCRITURA DE IAC

Algunas herramientas son Terraform, Ansible, Pulumi y AWS CloudFormation

NOMBRES CLAROS DE RECURSOS : UTILIZAR NOMBRES DESCRIPTIVOS Y CONSISTENTES PARA LOS RECURSOS AYUDA A IDENTIFICAR RÁPIDAMENTE SU PROPÓSITO

USO DE VARIABLES : LAS VARIABLES PERMITEN REUTILIZAR CONFIGURACIONES DE MANERA EFICIENTE Y HACER QUE LOS RECURSOS SEAN MÁS DINÁMICOS Y PERSONALIZABLES SEGÚN EL ENTORNO.

MODULARIZACIÓN DEL CÓDIGO :. POR EJEMPLO, UN MÓDULO PARA REDES, OTRO PARA BASES DE DATOS Y OTRO PARA APLICACIONES.

USO DE REPOSITORIOS DE CONTROL DE VERSIONES (GIT) : MANTENER EL CÓDIGO DE LA INFRAESTRUCTURA EN UN REPOSITORIO GIT PERMITE UN CONTROL COMPLETO SOBRE LOS CAMBIOS, ADEMÁS DE COLABORAR Y HACER AUDITORÍA SOBRE LAS CONFIGURACIONES.

PATRONES PARA MÓDULOS

Modularización

Separar los recursos en módulos lógicos facilita la reutilización y la gestión. Por ejemplo, un módulo para configurar redes, otro para definir bases de datos y otro para gestionar servidores de aplicaciones.



PATRONES PARA DEPENDENCIAS

GESTIÓN DE DEPENDENCIAS :PARA ENLAZAR MÓDULOS QUE DEPENDEN ENTRE SÍ

SALIDAS Y ENTRADAS :

```
output "dB password" {
    value = aws_secretsmanager_secret.db_password.secret_string
}

module "application" {
    source = "./modules/application"
    dB password = module.database.db_password
}
```

TAREA TEÓRICA

TERRAFORM ES UNA HERRAMIENTA DE INFRAESTRUCTURA COMO CÓDIGO QUE PERMITE A LOS USUARIOS DEFINIR Y GESTIONAR INFRAESTRUCTURA MEDIANTE ARCHIVOS DE CONFIGURACIÓN. PARA ORGANIZAR MEJOR LOS PROYECTOS, TERRAFORM UTILIZA MÓDULOS, LOS CUALES AGRUPAN ARCHIVOS DE CONFIGURACIÓN RELACIONADOS PARA FACILITAR SU REUTILIZACIÓN Y MANTENIMIENTO. LOS MÓDULOS EN TERRAFORM PERMITEN ESTRUCTURAR PROYECTOS COMPLEJOS DE MANERA MÁS ORDENADA Y EFICIENTE. UN MÓDULO PUEDE ESTAR COMPUESTO POR UN SOLO ARCHIVO .TF O POR UN CONJUNTO DE CARPETAS Y ARCHIVOS QUE DIVIDEN EL PROYECTO EN SEGMENTOS LÓGICOS.

EXISTEN DOS TIPOS PRINCIPALES DE MÓDULOS EN TERRAFORM:

MÓDULOS RAÍZ: SON LOS ARCHIVOS .TF UBICADOS EN EL DIRECTORIO PRINCIPAL DEL PROYECTO. SE EJECUTAN AUTOMÁTICAMENTE Y REPRESENTAN LA INFRAESTRUCTURA PRINCIPAL.

MÓDULOS REUTILIZABLES: SE ENCUENTRAN EN CARPETAS SEPARADAS Y PUEDEN SER UTILIZADOS EN DISTINTOS PUNTOS DE LA INFRAESTRUCTURA, PERMITIENDO UNA MAYOR FLEXIBILIDAD Y ESCALABILIDAD.

40

JUSTIFICACIÓN DE LA JERARQUÍA ELEGIDA

MAIN.TF: ORGANIZA LOS MÓDULOS Y PUEDES LLAMAR A LOS MÓDULOS NETWORK, DATABASEY APPLICATION. ESTE ARCHIVO ACTÚA COMO EL PUNTO DE ENTRADA PRINCIPAL PARA LA CONFIGURACIÓN DE LA INFRAESTRUCTURA.

VARIABLES.TF: DEFINA LAS VARIABLES GLOBALES QUE SE PUEDEN UTILIZAR EN TODO EL PROYECTO. AL CENTRALIZAR LAS VARIABLES, SE FACILITA LA REUTILIZACIÓN Y LA FLEXIBILIDAD EN LA INFRAESTRUCTURA.

OUTPUTS.TF: DEFINA LAS SALIDAS GLOBALES DEL PROYECTO, LO QUE PERMITE ACCEDER FÁCILMENTE A LOS RECURSOS CREADOS, COMO LA DIRECCIÓN IP PÚBLICA DE LA APLICACIÓN.

MODULES/: CADA MÓDULO ES INDEPENDIENTE, LO QUE FACILITA SU REUTILIZACIÓN Y MANTENIMIENTO. LOS MÓDULOS ESTÁN ORGANIZADOS PARA SU FUNCIÓN, DE FORMA QUE CADA UNO CONTIENE LOS ARCHIVOS NECESARIOS PARA GESTIONAR UN RECURSO ESPECÍFICO.

¿CÓMO CONTRIBUYEN LOS EQUIPOS AUTÓNOMOS Y MULTIFUNCIONALES A UNA IMPLEMENTACIÓN EXITOSA DE DEVOPS?

¿QUÉ SON LOS CONTENEDORES?

LOS CONTENEDORES SON UNIDADES DE SOFTWARE QUE EMPAQUETAN UNA APLICACIÓN Y SUS DEPENDENCIAS, COMO BIBLIOTECAS, ARCHIVOS DE CONFIGURACIÓN Y HERRAMIENTAS NECESARIAS PARA EJECUTAR EL SOFTWARE. A DIFERENCIA DE LAS MÁQUINAS VIRTUALES (VM), QUE REQUIEREN UN SISTEMA OPERATIVO COMPLETO PARA FUNCIONAR, LOS CONTENEDORES COMPARTEN EL MISMO NÚCLEO DEL SISTEMA OPERATIVO SUBYACENTE, LO QUE LOS HACE MUCHO MÁS LIGEROS Y RÁPIDOS DE ARRANCAR.

DIFERENCIA CON MÁQUINAS VIRTUALES (VM):

MÁQUINAS VIRTUALES (VMS): VIRTUALIZAN UN SISTEMA OPERATIVO COMPLETO Y REQUIEREN UNA CANTIDAD SIGNIFICATIVA DE RECURSOS (MEMORIA Y CPU).

CONTENEDORES: SOLO VIRTUALIZAN EL ESPACIO DE USUARIO Y COMPARTEN EL NÚCLEO DEL SISTEMA OPERATIVO SUBYACENTE, LO QUE LOS HACE MÁS LIGEROS Y EFICIENTES.

BENEFICIOS DE LOS CONTENEDORES:

AISLAMIENTO DE PROCESOS: LOS CONTENEDORES PUEDEN EJECUTARSE DE FORMA INDEPENDIENTE SIN INTERFERIR ENTRE SÍ.

LIGEREZA: NO NECESA UN SISTEMA OPERATIVO COMPLETO Y COMPARTIR RECURSOS CON EL SISTEMA HOST, OPTIMIZANDO EL USO DE RECURSOS.

IMAGEN VS CONTENEDOR

IMAGEN: ES UNA PLANTILLA INMUTABLE QUE DEFINE LO QUE CONTIENE UN CONTENEDOR. SE USA PARA CREAR CONTENEDORES.

CONTENEDOR: ES UNA INSTANCIA EN EJECUCIÓN DE UNA IMAGEN.

ESTRUCTURR DE DOCKERFILE

1. Especificar la imagen base
FROM ubuntu:20.04
2. Instalar dependencias necesarias
RUN apt-get update && apt-get install -y python3 python3-pip
3. Copiar archivos locales al contenedor
COPY . /app
4. Establecer el directorio de trabajo
WORKDIR /app
5. Instalar dependencias de la aplicación
RUN pip3 install -r requirements.txt
- | 6. Comando para ejecutar la aplicación
CMD ["python3", "app.py"]

KUBERNETES

KUBERNETES ES UNA PLATAFORMA DE ORQUESTACIÓN DE CONTENEDORES QUE AUTOMATIZA EL DESPLIEGUE, LA ESCALABILIDAD Y LA GESTIÓN DE APLICACIONES EN CONTENEDORES. TIENE PODS, SERVICIOS, IMPLEMENTACIONES Y CONJUNTOS DE RÉPLICAS.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-app-container
          image: my-app-image:v1
          ports:
            - containerPort: 80
```

ESTRATEGIAS DE IMPLEMENTACIÓN EN KUBERNETES:

ACTUALIZACIONES CONTINUAS: SIN TIEMPO DE INACTIVIDAD.

LANZAMIENTOS CANARY: SE PRUEBA UNA NUEVA VERSIÓN CON UN SUBCONJUNTO DE USUARIOS.

IMPLEMENTACIONES AZUL-VERDE: DOS VERSIONES COEXISTEN Y EL TRÁFICO SE REDIRIGE A LA NUEVA VERSIÓN UNA VEZ LISTA.

1. DESARROLLADOR REALIZA UN CAMBIO EN EL CÓDIGO

EL DESARROLLADOR MODIFICA EL CÓDIGO EN SU ENTORNO LOCAL Y LO SUBE A UN REPOSITORIO GIT (POR EJEMPLO, GITHUB, GITLAB O BITBUCKET). SE CREA UN PULL REQUEST (PR) Y SE REVISAN LOS CAMBIOS ANTES DE FUSIONARLOS A LA RAMA PRINCIPAL.

2. CONSTRUCCIÓN DE UNA NUEVA IMAGEN DOCKER

UNA CANALIZACIÓN DE CI/CD (POR EJEMPLO, GITHUB ACTIONS, GITLAB CI/CD, JENKINS, ARGOCD) DETECTA LOS CAMBIOS Y EJECUTA LOS SIGUIENTES PASOS: DESCARGA EL CÓDIGO ACTUALIZADO.

CONSTRUYE UNA NUEVA IMAGEN DOCKER CON EL COMANDO:

DOCKER BUILD -T MYAPP:V2 .

ETIQUETA LA IMAGEN Y LA SUBE A UN REGISTRO DE CONTENEDORES (DOCKER HUB, AMAZON ECR, GOOGLE CONTAINER REGISTRY , ETC.):

DOCKER PUSH MYREGISTRY/MYAPP:V2

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
        - name: myapp
          image: myregistry/myapp:v2
          ports:
            - containerPort: 80
```

VERIFICACIÓN Y MONITOREO

SE SUPERVISA EL DESPLIEGUE PARA ASEGURARSE DE QUE LOS NUEVOS PODS ESTÁN EN FUNCIONAMIENTO:

KUBECTL GET PODS

SE REVISAN LOS REGISTROS PARA DETECTAR POSIBLES ERRORES:

KUBECTL LOGS -F MYAPP-POD-NAME

SI ALGO FALLA, SE PUEDE HACER UN ROLLBACK A LA VERSIÓN ANTERIOR:

KUBECTL ROLLOUT UNDO DEPLOYMENT MYAPP

VENTAJAS DE KUBERNETES PARA ESCALAR EN UN EVENTO DE ALTO TRÁFICO

- 1. AUTOESCALADO BASADO EN DEMANDA**
- 2. BALANCE DE CARGA**
- 3. ORQUESTACIÓN EFICIENTE DE CONTENEDORES**
- 4. ESCALABILIDAD VERTICAL Y HORIZONTAL**
- 5. RECUPERACIÓN AUTOMÁTICA**

INTEGRACIÓN DE PROMETHEUS Y GRAFANA CON KUBERNETES PARA MONITOREO

PROMETHEUS Y GRAFANA SON HERRAMIENTAS ESENCIALES PARA EL MONITOREO DE INFRAESTRUCTURA Y APLICACIONES EN KUBERNETES, PROPORCIONANDO MÉTRICAS DETALLADAS Y VISUALIZACIÓN EN TIEMPO REAL.

PROMETHEUS

PROMETHEUS ES UN SISTEMA DE MONITOREO QUE RECOLLECTA MÉTRICAS DE APLICACIONES Y SISTEMAS, CONFIGURADO PARA HACER SCRAPING DE DATOS EN KUBERNETES MEDIANTE DESCUBRIMIENTO DINÁMICO DE SERVICIOS Y NODOS.

RECOLECCIÓN DE MÉTRICAS: USA EXPORTADORES COMO CADVISOR (CONTENEDORES), NODE EXPORTER (NODOS) Y KUBE-STATE-METRICS (ESTADO DE KUBERNETES).

ALERTAS: SE DEFINEN CON PROMQL Y SE GESTIONAN CON ALERTMANAGER PARA NOTIFICACIONES.

GRAFANA

GRAFANA ES UNA PLATAFORMA DE VISUALIZACIÓN QUE SE CONECTA CON PROMETHEUS PARA GENERAR DASHBOARDS INTERACTIVOS. PERMITE CONFIGURAR ALERTAS Y ENVIARLAS A CANALES COMO SLACK O CORREO ELECTRÓNICO.

CONEXIÓN CON PROMETHEUS

GRAFANA SE CONFIGURA PARA USAR PROMETHEUS COMO FUENTE DE DATOS. UNA VEZ CONFIGURADO, GRAFANA PUEDE CONSULTAR PROMETHEUS UTILIZANDO PROMQL Y MOSTRAR LOS RESULTADOS EN GRÁFICOS Y TABLAS.

PANELES DE CONTROL

GRAFANA PERMITE CREAR PANELES PERSONALIZADOS QUE VISUALIZAN DIFERENTES MÉTRICAS Y ALERTAS EN TIEMPO REAL. EXISTEN PANELES PREDEFINIDOS DISPONIBLES EN EL GRAFANA DASHBOARD REPOSITORY , O PUEDES CREAR LOS TUYOS PROPIOS.

ALERTAS EN GRAFANA

GRAFANA TAMBIÉN PUEDE GESTIONAR ALERTAS BASADAS EN LAS MÉTRICAS DE PROMETHEUS Y ENVIAR NOTIFICACIONES A TRAVÉS DE CANALES COMO SLACK O CORREO ELECTRÓNICO.

PROPIUESTA DE MÉTRICAS Y ALERTAS MÍNIMAS PARA UNA APLICACIÓN WEB

PARA UNA APLICACIÓN WEB, EL MONITOREO DEBE CENTRARSE EN CLAVES MÉTRICAS QUE PERMITAN DETECTAR PROBLEMAS DE RENDIMIENTO, DISPONIBILIDAD Y ESTABILIDAD.

MÉTRICAS MÍNIMAS PARA MONITOREAR

LATENCIA DE PETICIONES

MÉTRICA : LATENCIA PROMEDIO DE LAS SOLICITUDES HTTP (EN MILISEGUNDOS).

PROMQL :

`HTTP_REQUEST_DURATION_SECONDS_SUM / HTTP_REQUEST_DURATION_SECONDS_COUNT`

TASA DE ERRORES

MÉTRICA : TASA DE RESPUESTAS CON CÓDIGO DE ERROR (4XX, 5XX).

PROMQL :

`SUM(RATE(HTTP_REQUESTS_TOTAL{STATUS=~"4..5.."}[1M])) / SUM(RATE(HTTP_REQUESTS_TOTAL[1M]))`

USO DE CPU

MÉTRICA : PORCENTAJE DE USO DE CPU DE LOS CONTENEDORES DE LA APLICACIÓN.

PROMQL :

`AVG(RATE(CONTAINER_CPU_USAGE_SECONDS_TOTAL{CONTAINER_NAME!="", POD_NAME=~"APP-*"}[5M])) BY (POD_NAME)`

USO DE MEMORIA

MÉTRICA : MEMORIA UTILIZADA POR LOS CONTENEDORES DE LA APLICACIÓN.

PROMQL :

`AVG(CONTAINER_MEMORY_USAGE_BYTES{CONTAINER_NAME!="", POD_NAME=~"APP-*"}) BY (POD_NAME)`

TASA DE SOLICITUDES

MÉTRICA : NÚMERO TOTAL DE SOLICITUDES HTTP RECIBIDAS POR LA APLICACIÓN.

PROMQL :

`SUM(RATE(HTTP_REQUESTS_TOTAL{METHOD="GET", STATUS="200"}[5M])) BY (POD_NAME)`

DISPONIBILIDAD DEL SERVICIO

MÉTRICA : ESTADO DEL POD (SI ESTÁ EN ESTADO "RUNNING" O NO).

PROMQL :

`KUBE_POD_STATUS_PHASE{PHASE="RUNNING"}`

DIFERENCIA ENTRE ENTREGA CONTINUA (CONTINUOUS DELIVERY) Y DESPLIEGUE CONTINUO (CONTINUOUS DEPLOYMENT)

1. ENTREGA CONTINUA (ENTREGA CONTINUA)

LA ENTREGA CONTINUA ES UNA PRÁCTICA DE DESARROLLO EN LA QUE LOS CAMBIOS EN EL CÓDIGO SON AUTOMÁTICAMENTE PROBADOS Y PREPARADOS PARA SU LANZAMIENTO A PRODUCCIÓN. SIN EMBARGO, EL DESPLIEGUE EN PRODUCCIÓN NO ES AUTOMÁTICO ; REQUIERE UNA APROBACIÓN MANUAL ANTES DE QUE LA NUEVA VERSIÓN DEL SOFTWARE SEA PUBLICADA.

CARACTERÍSTICAS PRINCIPALES:

SE GARANTIZA QUE EL CÓDIGO SIEMPRE ESTÉ EN UN ESTADO DESPLEGABLE.

SE AUTOMATIZAN LAS PRUEBAS Y LA GENERACIÓN DE ARTEFACTOS LISTOS PARA PRODUCCIÓN.

EL EQUIPO PUEDE DECIDIR CUÁNDO Y CÓMO HACER EL DESPLIEGUE EN PRODUCCIÓN.

💡 EJEMPLO: UN EQUIPO DE DESARROLLO CONFIGURA UN PIPELINE CI/CD QUE EJECUTA PRUEBAS Y CONSTRUYE UNA NUEVA VERSIÓN DEL SOFTWARE, PERO UN INGENIERO DE DEVOPS DEBE APROBAR MANUALMENTE EL DESPLIEGUE EN PRODUCCIÓN.

2. DESPLIEGUE CONTINUO

EL DESPLIEGUE CONTINUO LLEVA LA ENTREGA CONTINUA UN PASO MÁS ALLÁ AL AUTOMATIZAR COMPLETAMENTE EL PROCESO DE DESPLIEGUE. EN ESTE CASO, CADA CAMBIO EN EL CÓDIGO QUE PASA LAS PRUEBAS SE DESPLIEGA AUTOMÁTICAMENTE EN PRODUCCIÓN , SIN INTERVENCIÓN MANUAL.

CARACTERÍSTICAS PRINCIPALES:

REQUIERE UNA ESTRATEGIA ROBUSTA DE PRUEBAS AUTOMÁTICAS PARA MINIMIZAR RIESGOS.

REDUZCA EL TIEMPO DE ENTREGA DE NUEVAS FUNCIONALIDADES A LOS USUARIOS FINALES.

AUMENTE LA FRECUENCIA DE DESPLIEGUES EN PRODUCCIÓN.

💡 EJEMPLO: CADA VEZ QUE UN DESARROLLADOR FUSIONA (MERGE) CAMBIOS EN LA RAMA PRINCIPAL, UN PIPELINE AUTOMATIZADO EJECUTA PRUEBAS, GENERA UNA NUEVA VERSIÓN Y LA DESPLIEGA DIRECTAMENTE EN PRODUCCIÓN SIN MANUAL DE INTERVENCIÓN.

IMPORTANCIA DE LAS PRUEBAS AUTOMÁTICAS EN EL PIPELINE

IMPLEMENTAR PRUEBAS AUTOMÁTICAS DENTRO DEL PIPELINE DE CI/CD ES FUNDAMENTAL PARA GARANTIZAR LA CALIDAD DEL SOFTWARE Y EVITAR ERRORES EN PRODUCCIÓN. EXISTEN VARIOS TIPOS DE PRUEBAS QUE DEBEN INCLUIRSE:

1. PRUEBAS UNITARIAS

OBJETIVO : VALIDAR QUE CADA UNIDAD DE CÓDIGO (FUNCIÓN, MÓDULO O CLASE) FUNCIONA CORRECTAMENTE DE MANERA AISLADA.

BENEFICIO : DETECTAR ERRORES EN ETAPAS TEMPRANAS Y GARANTIZAR QUE LAS FUNCIONES INDIVIDUALES OPEREN SEGÚN LO ESPERADO.

EJEMPLO : PROBAR QUE UNA FUNCIÓN QUE SUMA DOS NÚMEROS DEVUELVA EL RESULTADO CORRECTO.

2. PRUEBAS DE INTEGRACIÓN

OBJETIVO : EVALUAR CÓMO INTERACTÚAN ENTRE SÍ LOS DIFERENTES MÓDULOS O SERVICIOS DE LA APLICACIÓN.

BENEFICIO : DETECTAR FALLOS EN LA COMUNICACIÓN ENTRE COMPONENTES Y EVITAR PROBLEMAS DE INTEGRACIÓN ANTES DEL DESPLIEGUE.

EJEMPLO : VERIFICAR QUE EL BACKEND PUEDA COMUNICARSE CORRECTAMENTE CON LA BASE DE DATOS.

3. PRUEBAS DE SEGURIDAD

OBJETIVO : IDENTIFICAR VULNERABILIDADES EN EL CÓDIGO ANTES DE QUE LLEGUEN A PRODUCCIÓN.

BENEFICIO : PREVENIR ATAQUES COMO INYECCIÓN SQL, XSS (CROSS-SITE SCRIPTING), FUGAS DE DATOS O ACCESOS NO AUTORIZADOS.

EJEMPLO : EJECUTAR ESCANEOS DE SEGURIDAD AUTOMATIZADOS CON HERRAMIENTAS COMO OWASP ZAP O SONARQUBE.

Computación en la nube

Problemas o Limitaciones Antes de la Computación en la Nube y Cómo los Solucionó la Centralización de Servidores en Data Centers

Antes del surgimiento de la computación en la nube, las organizaciones enfrentaban una serie de limitaciones y problemas al gestionar su infraestructura tecnológica:

- **Alta inversión en hardware**
- **Escalabilidad limitada**
- **Gestión y mantenimiento costosos**

"The Power Wall" y la Influencia de los Procesadores Multi-Core en la Nube

"The Power Wall" hace referencia a una limitación física en la computación. A medida que los procesadores aumentaban en velocidad, también lo hacía su consumo de energía y la generación de calor. Este fenómeno llevó a un punto donde no era posible seguir aumentando la frecuencia de los procesadores sin superar los límites de energía y disipación térmica.

La aparición de los procesadores multinúcleo fue clave en la evolución hacia la nube. En lugar de intentar hacer que un solo procesador fuera más rápido, los procesadores multi-core permitieron dividir el trabajo en Múltiples núcleos de procesamiento, lo que mejoró el rendimiento sin aumentar significativamente el consumo de energía. Este enfoque permitió que los servidores en la nube pudieran manejar cargas de trabajo más grandes y complejas de manera más eficiente, favoreciendo la adopción de centros de datos masivos y escalables que forman la base de la computación en la nube.

Clústeres y equilibrio de carga

(a) Necesidad de Atender Grandes Volúmenes de Tráfico y la Adopción de Clústeres y Balanceadores de Carga

Cuando los sitios web o aplicaciones web alcanzan una gran cantidad de usuarios o solicitudes simultáneas, un solo servidor puede no ser capaz de manejar todo el tráfico, lo que resulta en tiempos de respuesta lentos o incluso caídas del servicio. Para abordar este problema, se implementan clústeres de servidores y balanceadores de carga :

Ejemplo Práctico del Uso de Load Balancers para una Aplicación Web

Imaginemos un desarrollador que está trabajando en una aplicación de comercio electrónico que experimenta picos de tráfico durante eventos especiales (como descuentos o lanzamientos de productos). Si solo hubiera un servidor manejando todas las peticiones, el sitio podría caerse o volverse muy lento durante esos picos.

Con un balanceador de carga , el desarrollador podría distribuir el tráfico entre varios servidores, de modo que cada uno reciba solo una parte de las solicitudes. Esto permite que el sistema mantenga un buen rendimiento incluso durante momentos de alto tráfico. Además, si uno de los servidores falla, el balanceador de carga puede redirigir automáticamente el tráfico a los servidores restantes, asegurando que la aplicación siga funcionando sin interrupciones.

Computación elástica

(a) Definición de Computación Elástica

Elastic Computing se refiere a la capacidad de una infraestructura de ajustarse automáticamente en función de la demanda. Esto significa que los recursos (como CPU, memoria, almacenamiento) pueden aumentar o disminuir según las necesidades de la aplicación, de forma dinámica y en tiempo real. La elasticidad permite que los sistemas se escalen hacia arriba (aumentando recursos) o hacia abajo (disminuyendo recursos) de manera eficiente.

(b) La Virtualización y Su Papel en la Elasticidad en la Nube

La virtualización es una tecnología que permite crear instancias virtuales de servidores físicos, permitiendo ejecutar múltiples máquinas virtuales (VMs) en una sola máquina física. Esta es una pieza clave para la elasticidad en la nube porque:

Permite crear o destruir instancias virtuales rápidamente.

Hace posible la asignación dinámica de recursos (como CPU y memoria) a las máquinas virtuales, adaptándolas a las necesidades del momento.

Facilita la migración de cargas de trabajo entre diferentes servidores físicos sin interrupciones, lo que permite escalabilidad flexible.

Escenario de Escalabilidad Sin un Entorno Elástico

Imagina que un desarrollador está creando una aplicación web para una campaña de marketing que tendrá picos de tráfico significativos durante un corto período de tiempo (por ejemplo, durante la duración de una oferta especial). Sin un entorno elástico, tendría que predecir y aprovisionar manualmente los recursos para manejar este tráfico adicional, lo que puede ser costoso y poco eficiente.

Modelos de Servicio en la Nube: IaaS, PaaS, SaaS y DaaS

Los modelos de servicio en la nube ofrecen distintos niveles de control y gestión sobre la infraestructura y las aplicaciones.

IaaS (Infraestructura como Servicio): Brinda recursos básicos como servidores, almacenamiento y redes, permitiendo a los usuarios gestionar el sistema operativo y las aplicaciones. Ejemplos: Amazon EC2, Google Compute Engine.

PaaS (Plataforma como Servicio): Facilita el desarrollo y gestión de aplicaciones sin preocuparse por la infraestructura subyacente. Ejemplos: Google App Engine, Heroku.

SaaS (Software como Servicio): Proporciona aplicaciones listas para usar a través de internet sin necesidad de instalación ni mantenimiento. Ejemplos: Gmail, Salesforce.

DaaS (Escritorio como Servicio): Permite el acceso a escritorios virtuales desde cualquier ubicación, mejorando la flexibilidad y movilidad. Ejemplos: Amazon WorkSpaces, Microsoft Windows Virtual Desktop.

¿Cuándo Elegir PaaS en Lugar de IaaS?

PaaS es ideal cuando el enfoque está en el desarrollo y despliegue rápido de aplicaciones sin preocuparse por la gestión de servidores, bases de datos o infraestructura. En cambio, IaaS es más adecuado cuando se requiere mayor control sobre la configuración de servidores y redes.

Ejemplos de Proveedores por Modelo

IaaS: Amazon EC2, Google Compute Engine, Microsoft Azure VMs.

PaaS: Google App Engine, Heroku, Red Hat OpenShift.

SaaS: Gmail, Dropbox, Salesforce.

DaaS: Amazon WorkSpaces, Citrix Virtual Apps, VMware Horizon Cloud.

Tipos de Nubes (Pública, Privada, Híbrida, Multi-Cloud)

Ventajas de Implementar una Nube Privada para una Organización Grande

Una nube privada permite a una organización tener un control total sobre su infraestructura y datos. Esto es ventajoso para organizaciones grandes que requieren:

Seguridad mejorada : Al estar bajo control exclusivo de la organización, la nube privada puede ofrecer mejores políticas de seguridad personalizadas.

Cumplimiento regulatorio : Algunas industrias (finanzas, salud) necesitan cumplir con estrictas normativas de privacidad y seguridad que se gestionan mejor en una nube privada.

Rendimiento optimizado : Al no compartir recursos con otras organizaciones, se puede garantizar un rendimiento constante.

Afectaciones del "Bloqueo de Proveedor"

El "proveedor lock-in" se refiere a la dependencia de una empresa de un solo proveedor de nube. Esto puede afectar a las empresas porque:

Limitaciones en la flexibilidad : Cambiar de proveedor puede ser costoso y complicado.

Dependencia de precios y servicios : Las empresas pueden estar atadas a las tarifas y políticas de servicio del proveedor.

(c) Rol de los "Hyperscalers" en el Ecosistema de la Nube

Los "hyperscalers" son grandes proveedores de servicios en la nube como Amazon Web Services (AWS) , Microsoft Azure y Google Cloud . Juegan un papel crucial en el ecosistema de la nube, proporcionando infraestructuras escalables, confiables y globalmente distribuidas, que permiten a las empresas escalar rápidamente sus servicios sin tener que gestionar hardware.

Estudio de Casos: Empresas que Migraron a la Nube

Caso 1: Netflix (Gran Organización)

Motivaciones:

Escalabilidad: Necesitaba manejar grandes volúmenes de datos y tráfico global.

Flexibilidad y disponibilidad: Garantizar un servicio continuo en todo el mundo.

Beneficios:

Reducción de costos: Eliminó infraestructura física costosa.

Escalabilidad: Uso de AWS para adaptarse dinámicamente a la demanda.

Disponibilidad global: Experiencia fluida para usuarios en diferentes regiones.

Desafíos:

Seguridad: Implementó medidas adicionales para cumplir con sus estándares.

Cumplimiento normativo: Adaptación a regulaciones como GDPR.

Caso 2: Airbnb (Startup)

Motivaciones:

Agilidad en el despliegue: Aceleración en la entrega de productos.

Optimización de costos: Reducción de gastos en infraestructura.

Beneficios:

Reducción de costos: Pago solo por los recursos utilizados.

Escalabilidad y flexibilidad: Capacidad de crecimiento según la demanda.

Desafíos:

Complejidad de migración: Reestructuración de su arquitectura.

Seguridad: Refuerzo en la protección de datos.

Comparativa de Modelos de Servicio (IaaS, PaaS, SaaS)

Aspecto	IaaS (Infraestructura como Servicio)	PaaS (Plataforma como servicio)	SaaS (Software como servicio)
Responsabilidad del desarrollador	Control total sobre el sistema operativo, redes y almacenamiento.	Despliegue de aplicaciones, pero sin acceso directo a la infraestructura.	Uso del software, sin acceso a la infraestructura o plataforma subyacente.
Responsabilidad del proveedor	Proporciona la infraestructura subyacente (máquinas virtuales, almacenamiento).	Proporciona la plataforma, herramientas y servicios para desarrollo de aplicaciones.	Y proporciona y mantiene el software listo para usar.
Responsabilidad del equipo de operaciones	Gestionar servidores, redes, almacenamiento y garantizar la disponibilidad.	Gestionar la plataforma y asegurar el funcionamiento de las aplicaciones.	No es necesario, ya que el proveedor maneja el software y la infraestructura.
Instalación de SO	El equipo de operaciones instala y gestiona el SO.	El proveedor ya gestiona el SO, el desarrollador usa la plataforma.	El software ya está disponible para usar, sin necesidad de instalación.
Despacho de aplicaciones	El equipo de desarrollo gestiona el despliegue en máquinas virtuales.	El desarrollador se despliega en la plataforma proporcionada sin .	El software se usa tal cual, sin necesidad de desplegarlo.

Estrategia Multi-Cloud o Híbrida

Escenario: Empresa mediana con infraestructura distribuida entre un centro de datos propio y un proveedor de nube pública (por ejemplo, AWS).

Estrategia para Migrar el 50% de las Cargas de Trabajo a un Segundo Proveedor de Nube:

Objetivo : Diversificar la infraestructura para evitar depender de un solo proveedor de nube (evitar el "proveedor lock-in") y aumentar la resiliencia.

- Base de Datos : La base de datos debe permanecer en un entorno que asegure la consistencia y disponibilidad. Para mantener la alta disponibilidad, la base de datos podría distribuirse entre ambos proveedores. Esto implicaría:
 - Sincronización entre bases de datos : Usar herramientas de replicación entre los dos proveedores, como AWS RDS replicado a Azure SQL Database o un servicio de base de datos autónomo multirregión.
- Red y Conectividad :
 - Configurar VPNs o Conexiones Directas entre el centro de datos privado y ambos proveedores de nube.
 - Utilizar redes privadas virtuales (VPC) para gestionar la conectividad entre los recursos en la nube y los locales.
- Plan de Contingencia :
 - Comutación por error : si un proveedor de nube falla, se puede configurar un plan de recuperación ante desastres (DR). Las aplicaciones críticas podrían tener réplicas en el segundo proveedor, y las cargas de trabajo podrían ser migradas dinámicamente usando herramientas de automatización.
 - Monitoreo : Utilizar herramientas como Prometheus y Grafana para monitorear la infraestructura en ambos proveedores y activar alertas en caso de fallos.

Debate sobre Costos: Comparativa de Nubes

Pros y contras de cada tipo de nube

Tipo de nube	Ventajas	Contras
Nube Pública	- Bajo costo inicial (pago por uso) - Escalabilidad y flexibilidad - Sin mantenimiento de hardware	- Dependencia de un proveedor - Preocupaciones de seguridad y privacidad
Nube privada	- Control total sobre los recursos y la seguridad - Cumple con regulaciones estrictas (ej. HIPAA) - Mejor rendimiento	- Altos costos iniciales (CAPEX) - Mantenimiento constante de hardware y personal especializado
Nube Híbrida	- Flexibilidad de combinar lo mejor de nubes públicas y privadas - Redundancia y resiliencia	- Complejidad en la gestión de recursos - Costos adicionales por gestión e integración
Multi-nube	- No depende de un solo proveedor - Optimización de costos entre diferentes proveedores	- Complejidad en la gestión de recursos y redes - Desafíos de integración y monitoreo

Costos Iniciales (CAPEX vs OPEX)

CAPEX (Capital Expenditure) : Costos asociados con la compra de hardware y recursos que se amortizan con el tiempo.

OPEX (Operational Expenditure) : Costos operativos recurrentes, como pago por uso en la nube pública, que son más flexibles y escalables.