**Devops**:

* Movimiento organizacional y cultural
* Busca incrementar velocidad de entrega de software
* Busca mejorar fiabilidad del servicio
* Busca construir una responsabilidad compartida

Medidas,

-frecuencia de entrega de software

-el tiempo que toma en poder gestionar un cambio

**OBJETIVO, LOGRAR VELOCIDAD Y ESTABILIDAD**

**MEJORAR AL COMUNICACIÓN Y AUTOMATIZACION**

**Class SRE implements DevOps**

**SRE** es una aplicación practica de los principios de DevOps. SRE complementa DevOps con guías tangibles a lo largo de la cultura de DevOps.

**DevOps** es una cultura y un conjunto de prácticas diseñadas para quitar las diferencias entre los equipos de Desarrollo y Operación.

**Mejores prácticas de SRE**

1. **Shared Ownership** / **Titularidad compartida**: Todos somos responsables de la velocidad entrega y por la estabilidad del software.
2. **Blameless postmortem**/ \*\* *Post Mortem* sin culpa\*\*: Aceptar el fracaso como algo normal, no buscamos a quien culpar sino como podemos mejorar el proceso o código para que no vuelva a ocurrir.
3. **Reducir el costo del error**: Implementar cambios graduales, pequeños y frecuentes; para que el impacto sea acotado. Entregar software de forma continua no solo agrega valor a los clientes sino que también reduce el impacto que un error pueda tener.
4. **Automatizar los casos comunes**: Aprovechar las herramientas y la automatización para evitar errores humanos.
5. **Medir todo**: Medir nos ayuda a mejorar, nos ayuda a identificar problemas de forma proactiva. Medir el esfuerzo y *reliability*.

## Service Level Terminology

#### SLI

Service Level Indicator  
Un SLI es un indicador de nivel de servicio, una medida cuantitativa cuidadosamente definida de algún aspecto del nivel de servicio que se proporciona.

* La mayoría de los servicios consideran la **latencia de la solicitud** (request latency) (cuánto tiempo se tarda en devolver una respuesta a una solicitud) como un SLI clave.
* Otros SLI comunes incluyen la **tasa de error** (error rate), a menudo expresada como una fracción de todas las solicitudes recibidas.
* Por último el **rendimiento del sistema**(system throughput), también es un SLI común, generalmente medido en solicitudes por segundo.
* Otro tipo de SLI importante para las SRE es la **disponibilidad**(availability), o la fracción del tiempo que se puede utilizar un servicio.

#### SLO

Service Level Objective  
Un Objetivo de nivel de servicio es un valor objetivo o rango de valores para un nivel de servicio medido por un SLI. El objetivo al que apuntamos es al SLO, mientras que la métrica o indicador es SLI.

* La elección y publicación de SLO para los usuarios establece expectativas sobre el rendimiento de un servicio. Esta estrategia puede reducir las quejas infundadas a los propietarios del servicio sobre, por ejemplo, la lentitud del servicio.
* Sin un SLO explícito, los usuarios a menudo desarrollan sus propias creencias sobre el desempeño deseado, que pueden no estar relacionadas con las creencias de las personas que diseñan y operan el servicio.

#### SLA

Service level agreements  
Un Acuerdo a nivel de servicio es un contrato explícito o implícito con sus usuarios que incluye las consecuencias de cumplir (o no cumplir) los SLO que contienen.

Las consecuencias se reconocen más fácilmente cuando son financieras (un reembolso o una multa), pero pueden adoptar otras formas.

Una manera fácil de distinguir entre un SLO y un SLA es preguntarse “¿qué sucede si no se cumplen los SLO?”: Si no hay una consecuencia explícita, es casi seguro que esté mirando un SLO.

Por lo general, los y las SRE no se involucra en la construcción de SLA, porque los SLA están estrechamente vinculados a las decisiones comerciales y de productos.

Sin embargo, SRE se involucra para ayudar a evitar desencadenar las consecuencias de los SLO omitidos. También pueden ayudar a definir los SLI: obviamente, debe haber una forma objetiva de medir los SLO en el acuerdo, o surgirán desacuerdos.

## Error Budget

El Porcentaje de error aceptable es la diferencia entre el 100% y nuestro objetivo (SLO) que tenemos como oportunidad para hacer cambios/mejoras/mantenimiento.

El Porcentaje de error aceptable nos permite

* Aumentar la velocidad de desarrollo.
* Incrementar mejoras.
* Innovar en nuestros productos.

#### Beneficios del porcentaje de error aceptable

* **Incentivos comunes para desarrolladores y SREs**: Encuentra el balance adecuado entre innovación y reliability.
* **Los equipos de desarrollo pueden gestionar el riesgo por su cuenta**: Ellos deciden cómo utilizar el porcentaje de error aceptable.
* **Los objetivos de reliability poco realistas no son atractivos**: Estos objetivos disminuyen la velocidad de la innovación.
* **Responsabilidad compartida por uptime del sistema**: Los fallos de la infraestructura utilizan el porcentaje de error aceptable de los desarrolladores.

#### Consecuencias del porcentaje de error aceptable

✅ **Cuando hay** porcentaje de error aceptable: **Priorizar la velocidad**  
● Lanzamiento de nuevas funciones.  
● Cambios previstos en el sistema.  
● Fallos inevitables en el hardware, las redes, etc.  
● Experimentos arriesgados.

❌ **Cuando se agota** el porcentaje de error aceptable: **Priorizar la estabilidad**  
● Ralentizar o detener los lanzamientos de nuevas funciones.  
● Priorizar items del postmortem.  
● Automatizar los procesos de implementación.

#### Preparando los archivos fuente

quickstart.sh  
Para hacer ejecutable el script chmod +x quickstart.sh

echo "Hola mundo!"

Dockerfile

FROM alpine

COPY quickstart.sh /

CMD ["/quickstart.sh"]

#### Crear un repo de Docker en Artifact Registry

gcloud artifacts repositories create quickstart-docker-repo --repository-format=docker \

--location=us-central1 --description="Docker repository"

Para verificar que el repo se creo:

gcloud artifacts repositories list

#### Construyendo la imagen usando Docker

El project-id se puede obtener con el comando gcloud config ger-value project

gcloud builds submit --tag us-central1-docker.pkg.dev/project-id/quickstart-docker-repo/quickstart-image:tag1

#### Construyendo usando un build config file

cloudbuild.yaml  
El $PROJECT\_ID se puede obtener con el comando gcloud config ger-value project

steps:

- name: 'gcr.io/cloud-builders/docker'

args: [ 'build', '-t', 'us-central1-docker.pkg.dev/$PROJECT\_ID/quickstart-docker-repo/quickstart-image:tag1', '.' ]

images:

- 'us-central1-docker.pkg.dev/$PROJECT\_ID/quickstart-docker-repo/quickstart-image:tag1'

Para ejecutar el cloudbuild.yaml usamos:

gcloud builds submit --config cloudbuild.yaml

En **Cloud Build** bajo **Build history** podemos ver los archivos que se han compilado con el comando anterior.

Cloud Build es un servicio que ejecuta tus compilaciones en la infraestructura de Google Cloud Platform. Cloud Build puede importar código fuente de Cloud Storage, Cloud Source Repositories, GitHub o Bitbucket, ejecutar una compilación según tus especificaciones y producir artefactos como contenedores de Docker o archivos de Java. Más información

Cloud Build ejecuta la compilación como una serie de pasos de compilación, en los que cada uno se ejecuta en un contenedor de Docker. Un paso de la compilación puede todo lo que se puede hacer desde un contenedor, sin importar el entorno. Para realizar tus tareas, puedes usar los pasos de la compilación compatibles que proporciona Cloud Build o escribir tus propios pasos de la compilación

****

## Despliegue de Infraestructura como Código

#### Beneficios

* **Administración de código**: Commit, versiona y colabora, como cualquier código fuente.
* **Declarativo**: Especifica el estado deseado de la infraestructura.
* **Auditable**: Valida las diferencias entre la arquitectura deseada y la actual.
* **Portable**: Crea módulos reutilizables en tu organización.

#### Pipeline de automatización

* **Colabora** en un repositorio de código.
* Asegura la **consistencia**.
* **Reduce** esfuerzos manuales y **errores**.
* Aplica **políticas** proactivamente.

#### Deployment Manager

* **Despliegues por capa**  
  En vez de usar un despliegue monolítico, separa cada componente lógico en despliegues separados (red, front, back, data, etc).
* **Archivos de esquema**  
  Define tu plantilla a través de un esquema.
* **Código de verdad**  
  Usa configuraciones, almacenadas en múltiples archivos .py, escribe código de verdad, fácil de reusar y no repetir.
* **Usa funciones adicionales**  
  Crea lógica en funciones de Python y compártelas en distintos despliegues.
* **Usa Python**
* Jinja es muy bueno para cosas básicas, pero con Python se puede lograr capacidades superiores.
* \*\*Previsualiza antes de aplicar \*\*  
  Usa la opción de preview antes de actualizar un ambiente, para evitar problemas y detectar posibles configuraciones erróneas.