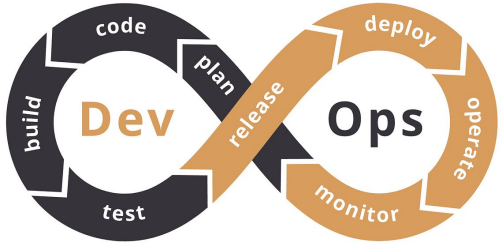
DevOps I - Introducción y aplicación en tecnología de la información

¿Qué es DevOps?

DevOps es una metodología de desarrollo que integra y automatiza el trabajo del **desarrollo de software** (dev) y **operaciones de tecnología de la información** (ops), con el objetivo de mejorar y acortar el ciclo de vida de un sistema.



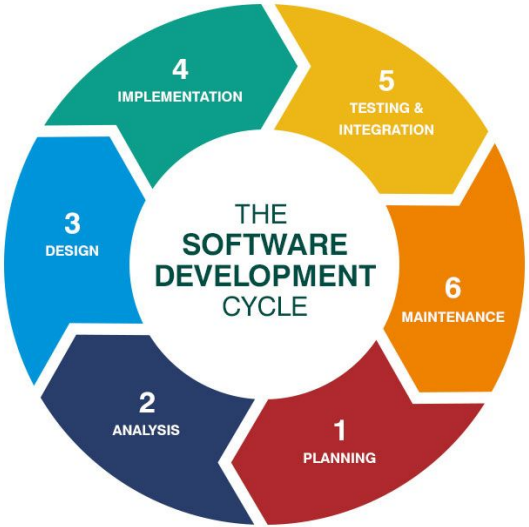
Origen

Ya en los 80s y 90s, existían propuestas para combinar metodologías de desarrollo de software con conceptos de operaciones.

Entre 2007 y 2008, surgieron preocupaciones dentro de las comunidades de desarrollo de software y TI de que la separación entre las dos industrias, donde una escribía y creaba software completamente separada de aquellos que lo implementan y apoyan, estaba generando un nivel fatal de **disfunción** dentro de la industria.

En la conferencia Agile 2008 Toronto, Yhens Wasna y Patrick Debois introdujeron el término en su charla sobre "Infraestructura Ágil". A partir de 2009, el término DevOps se ha promocionado constantemente y se ha incorporado a un uso más general a través de una serie de "devopsdays", que comenzaron en Bélgica y ahora también se han extendido a otros países.

Desarrollo de software



Recordemos que desarrollo de software incluye no solamente **programar** y realizar **mantenimiento** sobre un código dado, como paso central para el proceso, sino que también incluye el **análisis** de requerimientos del sistema, **diseño**, **testeo**, y **lanzamiento**.

Tecnología de la información

Por otra parte, las operaciones de la tecnología de la información tienen más relación con **otras** actividades que son esenciales para el funcionamiento y mantenimiento de sistemas de software en producción, incluyendo:

- **Gestión de infraestructura**, como configuración y mantenimiento de servidores, redes y otros componentes de hardware.
- **Monitoreo y rendimiento**, fijarse bien de que la disponibilidad, rendimiento, y salud general de los sistemas esté asegurado a través de una supervisión constante.
- **Seguridad**, que incluye la protección contra amenazas y vulnerabilidades externas o internas...

Tecnología de la información

- **Despliegue**, es decir la entrega de software (con DevOps se busca automatizarla).
- **Backups**, muchas estrategias para garantizar que los datos se respalden y puedan recuperarse en caso de fallos.
- **Escalabilidad**, muchas veces los sistemas no pueden manejar un aumento en la carga y el tráfico.
- **Configuración y gestión de versiones**, muchos controles relacionadas a las configuraciones y versiones de software para que los entornos de desarrollo y producción mantengan la coherencia; muchas veces quedan desalineados.

Hay muchísimas herramientas de DevOps para cada uno de estos ítems, y los vamos a ir viendo en esta presentación:

Herramientas para gestión de infraestructura

Terraform: definir, aprovisionar y gestionar infraestructuras de manera eficiente utilizando un lenguaje declarativo llamado **HashiCorp Configuration Language (HCL)**, o incluso JSON.

- Se define todo el código en archivos de configuración
- Se describe directamente el estado final que queremos tener en nuestra infraestructura, y Terraform va a automáticamente intentar llegar a ese estado (*declarativo*)
- Compatible con los mejores proveedores de nube y servicios.
- Permite planificación de cambios, gestión del ciclo de vida de la infraestructura y estado de la infraestructura.
- Crea módulos reutilizables, fomentando la **modularización** y **reutilización** del código.
- Gestiona dependencias de forma automática.

¿Cómo funciona Terraform internamente?

Funciona con JSON o un lenguaje **declarativo**, llamado **HashiCorp Configuration Language (HCL)**.

Ejemplo concreto - instancia de una máquina virtual en la nube de Amazon (**EC2 en AWS)**:

```
provider “aws” {

      region = “sa-1”

}

#Acá definimos el proveedor de Amazon Web Service (AWS), y qué región nos conviene a nosotros.

resource “aws_instance” “mi_instancia_con_terraform” {

      ami = "ami-0c55b159cbfafe1f0" #Imagen del Sist. Operativo para la máquina virtual. En este ejemplo, usamos Linux.

      instance_type = "t2.micro" #La instancia que vamos a usar, esta es gratis. Le podemos poner un nombre que queramos nosotros a la instancia con la siguiente línea de código:

      tags = {

            Name = “InstanciaTerraformADS2024”

      }

}
```

¿Cómo funciona Terraform internamente?

Después tenemos comandos básicos para lo que vayamos necesitando. Por ejemplo:

```
terraform init #Iniciar un proyecto

terraform plan #Planificar los cambios

terraform apply #Aplicar los cambios

terraform destroy #Destruir recursos innecesarios
```

Para hacer funcionar el ejemplo del proyecto anterior, nos faltarían los siguientes pasos:


```
export AWS_ACCESS_KEY_ID = “clave_de_acceso”

export AWS_SECRET_ACCESS_KEY = “clave_secreta” #Estas claves las encontramos en el archivo de configuración de AWS con la clave única para nuestro proyecto.

terraform init #Inicializamos terraform

terraform plan #Esto nos va a mostrar un plan de acción en base a lo existente.

terraform apply #Si nos gustó el plan que hizo terraform, creamos la instancia.

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Listo, tenemos nuestra infraestructura hecha :)

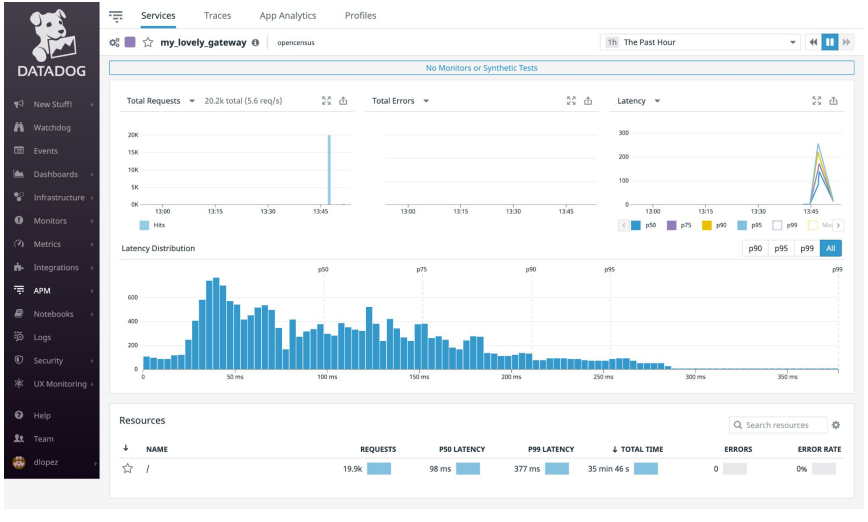
Herramientas para monitoreo y rendimiento

Hay bastante software de monitoreo, ninguno que sea concretamente superior. Algunos ejemplos son:

Grafana: es de código abierto, podemos consultar datos en bases de datos. Se conecta a bases de datos como Oracle, SQL, o la que usemos, recibe los datos, y los transforma en gráficos, tableros e informes. Tiene sistemas de alertas, por si se llega a algún número umbral que no queramos de cualquier información que sea, aparte de muchos plugins e integraciones.

Herramientas para monitoreo y rendimiento

Datadog: muy similar a Grafana, pero mayormente ofrece visibilidad de infraestructura, aplicaciones y logs. Monitorea todo el **stack** tecnológico en servidores, bases de datos, servicios en la nube (como el que creamos con Terraform!!), entre otros. También tiene alertas como Grafana, y se integra con muchas otras herramientas DevOps que están en esta presentación.



Herramientas para seguridad

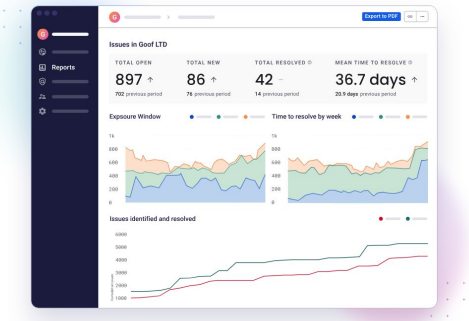
En términos de seguridad, hay dos vertientes: una que se refiere al **scanning de vulnerabilidades**, es decir, problemas que pueda haber en nuestro código y las dependencias, y otra que se refiere a la **autenticación y autorización seguras**, para que no entren hackers a nuestro sistema, etc..

Algunas herramientas pueden ser...

Herramientas para escaneo de vulnerabilidades

Snyk: Identifica, y corrige vulnerabilidades en dependencias de código, contenedores y configuraciones de infraestructura como código.

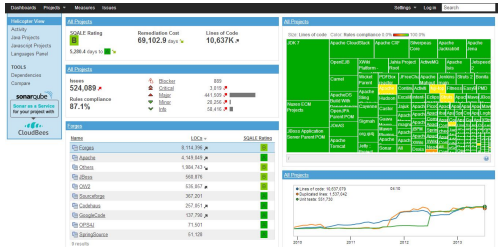
- Primero, Snyk se fija que los archivos de dependencias de los proyectos (package.json), (node-modules), etc.. no tengan vulnerabilidades .
- Después, escanea archivos de configuración (tipo **terraform**) para ver que no haya puertos abiertos, permisos muy amplios, o cualquier vulnerabilidad de ese estilo.
- Una vez que las detecta, Snyk sugiere correcciones automáticas o parches para problemas en las dependencias.



Herramientas para escaneo de vulnerabilidades

SonarQube: analiza código estático, mejorando así la calidad del código y detectando vulnerabilidades de seguridad en el código fuente. Se encarga especialmente de mantener algún patrón de diseño que tenga el código, y de que no se duplique para que esté bien modularizado.

- Detecta bugs y se revisa el código sin ejecutarlo, así que antes de que el software esté en ambiente de explotación, ya podemos ver algún problema que haya.
- Internamente SonarQube tiene miles de reglas de calidad que hay en los distintos lenguajes de programación (como Java, Python, JavaScript, C#, PHP, etc.), y evalúan la estructura y calidad del código.
- No detecta tantas vulnerabilidades como Snyk!!
- Genera informes sobre la calidad del código y las vulnerabilidades detectadas.



Dashboards | Projects | Measures | Issues | Settings | Log in | Search

- Helicopter View
- Activity
- Java Projects
- Javascript Projects
- Languages Panel

TOOLS

- Dependencies
- Compare

sonarqube
Sonar as a Service for your project with
CloudBees

All Projects

SQALE Rating: B
5,280.4 days to A

Remediation Cost: 69,102.9 days

Lines of Code: 10,637K

All Projects

Issues: 524,089

Rules compliance: 87.1%

Blocker	889
Critical	3,019
Major	441,509
Minor	20,256
Info	58,416

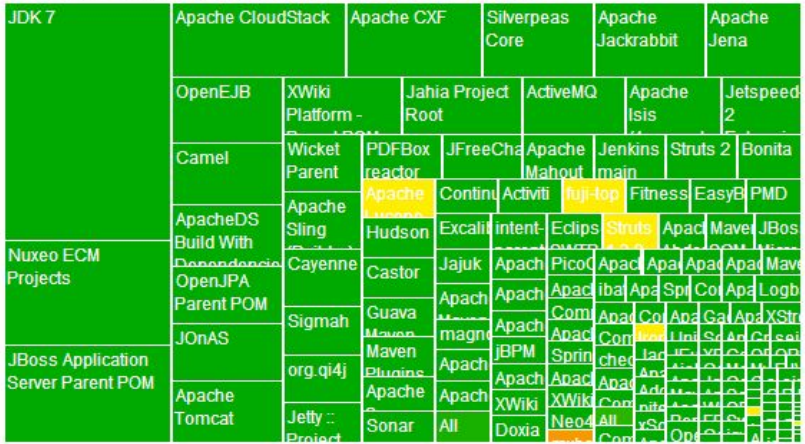
Forges

Name	LOCs	SQALE Rating
Forges	8,114,396	B
Apache	4,149,049	A
Others	1,984,743	B
JBoss	560,876	B
OW2	535,057	B
Sourceforge	367,201	A
Codehaus	257,051	A
GoogleCode	137,790	A
OPS4J	71,501	A
SpringSource	51,128	A

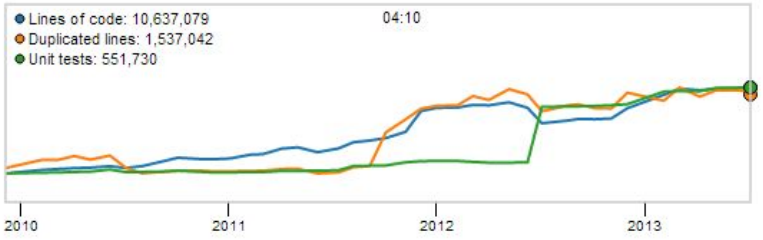
9 results

All Projects

Size: Lines of code Color: Rules compliance 0.0% 100.0%



All Projects



Helicopter View

ALL PROJECTS

Calificación SQALE

A

Ratio De Deuda Técnica

5.2%

Deuda Técnica 39,904d

Líneas De Código 12,515K

ALL PROJECTS

Debt 39,904d

Evidencias 1,825,913

Bloqueante	18,646
Crítica	56,384
Mayor	818,158
Menor	846,298
Info	86,427

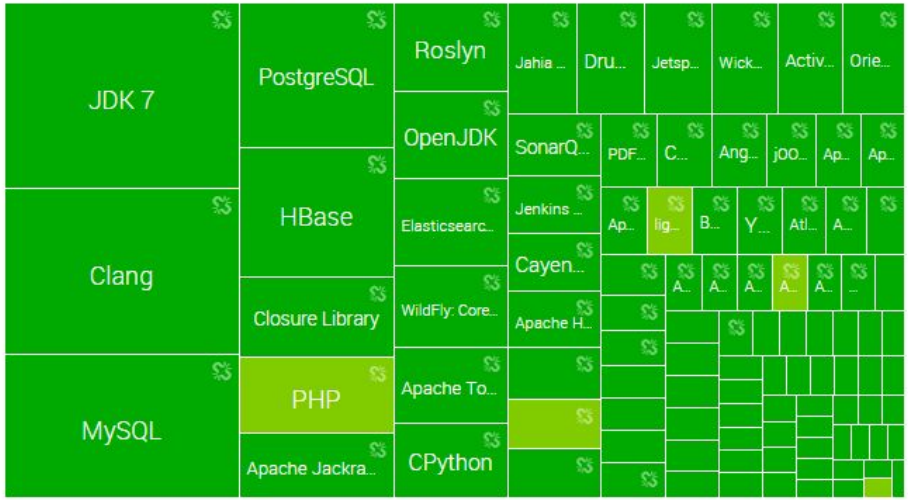
Global Security Issue Tags

error-handling	52379	multi-threading	5856	sans-top25	1049
owasp-a6	939	owasp-a2	807	owasp-a1	802
sans-top25-risky	745	owasp-top10	314	denial-of-service	214
injection	14				

FORGES

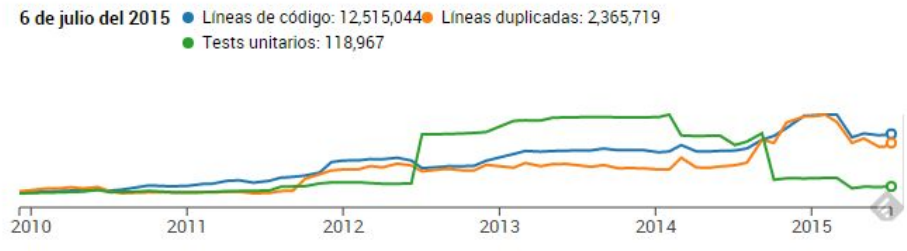
ALL PROJECTS

Size: Líneas de código Color: Calificación SQALE



Solo se muestran los primeros 100 componentes

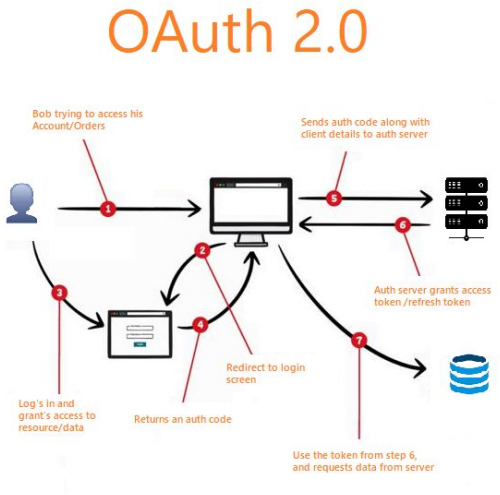
ALL PROJECTS



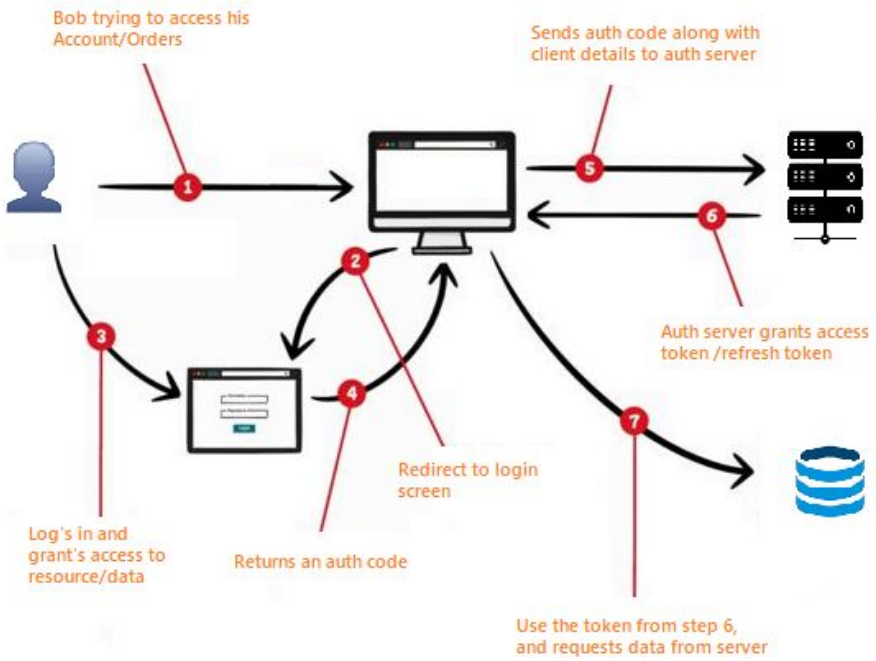
Herramientas para autenticación y autorización seguras

OAuth: protocolo de autorización que permite a un cliente acceder a recursos de un usuario en otro servicio sin compartir las credenciales. Por ejemplo, se usa para cuando uno apreta el botón de "Iniciar sesión con Google" en una aplicación, en ese caso, en lugar de ingresar tus credenciales en la app, OAuth permite que la app acceda a tu perfil en Google de manera segura usando tokens de acceso.

OAuth utiliza internamente algo llamado **OpenID Connect (OIDC)**, una capa de identidad construida sobre OAuth 2.0 que agrega autenticación a las capacidades de autorización de OAuth, lo que nos permite OIDC es verificar la identidad del usuario.



OAuth 2.0


Bob trying to access his Account/Orders
Sends auth code along with client details to auth server
1
2
3
4
5
6
7
Auth server grants access token /refresh token
Redirect to login screen
Log's in and grant's access to resource/data
Returns an auth code
Use the token from step 6, and requests data from server

Herramientas para autenticación y autorización seguras

HashiCorp Vault: Permite gestionar secretos y proteger el acceso a datos sensibles en aplicaciones, como contraseñas, tokens de API, entre otros.

- Los ‘secretos’ se almacenan de manera cifrada.
- Tiene políticas de acceso para controlar quién puede acceder a qué *secreto*, definiendo qué usuario o aplicación tiene permiso para leer, escribir o generar secretos.

Ejemplo: Una aplicación que necesita acceder a una base de datos obtiene sus credenciales de forma dinámica usando esta herramienta, en lugar de almacenarlas en un archivo de configuración existente en la memoria del usuario o de los servidores (muy vulnerable), mejorando así la seguridad de las credenciales.

Herramientas para despliegue

Jenkins: Automatiza código abierto, permitiendo principalmente integración continua (CI) y despliegue continuo (CD).

- **Integración continua**: Cualquier desarrollador que esté metido en el mismo proyecto Jenkins puede integrar su cambio de código de manera frecuente en un repositorio central, ejecutando automáticamente pruebas unitarias y de integración, asegurando que el código no se rompa.
- **Despliegue continuo**: Después de que el código previamente integrado pasa las pruebas, se puede automatizar su entrega a entornos de desarrollo, pruebas o producción.

Ejemplo: Cuando un desarrollador hace un commit de código en Git, Jenkins automáticamente detecta el cambio, ejecuta pruebas automáticas, compila el código, y si anduvo todo bien, lo despliega en un servidor de prueba, notificando al equipo si hay problemas.

Herramientas para despliegue

GitLab: extremadamente similar a Jenkins en muchos aspectos. Permite integración continua y entrega continua de la misma manera que Jenkins, pero fomentando más aún la colaboración en proyectos porque ofrece un repositorio de código basado en Git y un entorno unificado para los desarrolladores. Dentro del repositorio, se pueden crear muy fácilmente ramas, hacer pull requests, realizar revisiones de código, y ver versiones viejas y tomar partes de versiones viejas (cherry pick) del proyecto.

Herramientas para despliegue

AWS CodePipeline: Amazon Web Service Code Pipeline permite compilar, probar y desplegar aplicaciones de manera rápida.

- Define flujos de trabajo o **pipeline**: Define el flujo de trabajo completo y despliegue de una aplicación.
- Automatiza el flujo de desarrollo, detectando un cambio y ejecutando el pipeline definido para el proyecto.
- Tiene excelente control de versiones, facilitando hacer rollback (volver a una versión vieja), si hubo algún tipo de problema en producción.

AWS CodePipeline

Herramientas para backups

Las distintas herramientas utilizadas en DevOps también utilizan automatización de backups.

Veeam: Realiza copias de seguridad a nivel de imagen, lo que significa que puede capturar el estado completo de una máquina virtual (VM), servidor físico o una instancia en la nube, permitiendo restaurar todo el sistema operativo, aplicaciones y datos si es necesario. También tiene la capacidad de restaurar archivos individuales, algún correo electrónico e incluso registros dentro de bases de datos.

Además de los backups, Veeam puede replicar máquinas virtuales o servidores a un sitio alternativo (DR - **disaster recovery**). Esto permite que en caso de un fallo o desastre en el centro de datos principal, se pueda hacer un **failover** a la réplica.

Herramientas para backups

AWS Backup: Amazon Web Service backup permite, al igual que Veeam, la automatización de copias de seguridad y recuperación de recursos almacenados en la nube.

- Tiene centralización de backups - realiza copias de seguridad múltiples de otros servicios AWS.
- Distintas políticas de backup para respaldar los recursos.
- Copias cruzadas entre distintas regiones de AWS.
- **Automatización**; se configuran las políticas para que se ejecuten en intervalos específicos de tiempo.

(Entre nosotros... Veeam es mucho más completo y mejor)

Herramientas para escalabilidad

Microsoft Azure: Azure tiene un montón de soluciones para almacenamiento, cómputo, redes, entre otros. En términos de escalabilidad, Azure tiene muchas herramientas:

- **Virtual Machines**: Se pueden escalar máquinas virtuales bajo demanda, permitiendo cambiar el tamaño de las VMs o aumentar el número de instancias.
- Azure contiene una plataforma para desplegar aplicaciones web y APIs (PaaS - Platform as a Service). Esta plataforma, llamada **Azure App Service** tiene escalabilidad automática basada en el tráfico; si hay mucho uso entonces se agregan instancias de aplicaciones de forma automática.
- Con respecto a CPU, memoria o cantidad de solicitudes, Azure también permite configurar reglas de **escalado automático**.
- El Azure **Load Balancer** nos deja distribuir el tráfico entre varios recursos para que la performance sea la mejor posible. Estos load balancer son particularmente importantes en Azure porque permiten gestionar grandísimos volúmenes de tráfico eficientemente, sin perder performance.

Herramientas para configuración y gestión de versiones

Git: Es posiblemente una de las herramientas más utilizadas, sino la más utilizada en contextos de DevOps, y es elemental entenderla al ingresar a cualquier tipo de proyecto que aplique metodologías ágiles. El funcionamiento de Git es simple y bastante directo:

- **Repositorio**: Lugar local o en una nube (GitHub, GitLab) donde se almacenan los archivos y su historial de cambios.
- **Commit**: Registro de los cambios realizados en el código. Lo puede realizar cualquier desarrollador (esto no significa que vaya a aprobarse), describiendo los cambios realizados.
- **Branching**: Ramas para trabajar en diferentes características o correcciones de manera independiente. Permite desarrollar nuevas funcionalidades sin afectar el código principal (muy importante para trabajar con metodologías como SCRUM o XP que tienen **historias de usuario**).

Herramientas para configuración y gestión de versiones

- **Merge**: Si los cambios en una rama son completados y probados, y funcionan correctamente, se aprueba un **commit** para ser combinados (mergeados) en otra rama superior, como la principal.
- **Clonación y pushing**: Es posible clonar un repositorio para hacer cambios locales en el mismo. Si realizamos cambios y los aprobamos internamente, hacemos *push* para enviarlos de vuelta al repositorio remoto.

Ejemplos de comandos básicos en Git:


```
git clone <url_repositorio> #Esto permite clonar un repositorio a nuestra computadora

git checkout -b rama_nueva_funcionalidad #Acá creamos una nueva rama que se llama 'rama_nueva_funcionalidad'

git add . #Hace los cambios internamente.

git commit -m "Descripción del commit" #Realiza un commit, confirmando los cambios agregados previamente, con un mensaje que describa qué cambios se realizaron
```

Herramientas para configuración y gestión de versiones

git push origin rama_nueva_funcionalidad #Esto sube los cambios del commit al repositorio remoto, agregándolos a nuestra nueva rama.

git merge rama_nueva_funcionalidad #Finalmente, unimos los cambios a la rama principal en el caso en el que el código realizado en la nueva rama haya sido aprobado para que podamos mandarlo al sistema completo.

Herramientas para configuración y gestión de versiones

Ansible: Es una herramienta para **automatización de configuración** y gestión de infraestructuras. Automatiza aprovisionamiento de servidores, instalación de software, entre otras.

Ansible tiene algo llamado **playbooks**, que contienen "*plays*" que describe tareas a realizar; qué hacer y dónde hacerlo. Para saber dónde hacerlo, se utilizan los **inventarios**, especificando en qué servidores deben ejecutarse las tareas.

Aparte, Ansible tiene **módulos**, pequeñas unidades (a veces atómicas) de código que realizan tareas específicas, y **roles**, que permiten estructurar las tareas en módulos reutilizables.

Ansible utiliza archivos de tipo **YAML** (es una onda JSON).