Universidad Tecnológica Nacional
Facultad Regional Santa Fe

Cátedra: Proyecto Final
Proyecto Final de Carrera

Análisis y optimización de tiempos en el ciclo de vida de la infraestructura utilizando enfoques de infraestructura como código

Alumno:
Gregoret, Guillermo Andrés - Legajo N° 24687 - ggregoret@frsf.utn.edu.ar

Director
Ing. Filippa, Gabriel

Año
2023

Introducción

El presente proyecto final de carrera tiene como objetivo principal solucionar la problemática relacionada con la falta de automatización en la aplicación de cambios en el proceso de desarrollo de software. Esta problemática ha generado ineficiencias y la posibilidad de errores en el proceso de copia de los cambios en los diferentes ambientes.

En este contexto, los objetivos generales del proyecto son mejorar los tiempos que la empresa posee al implementar cambios para desplegarlos en testing, staging y producción. Para alcanzar este objetivo, se plantean una serie de objetivos específicos que incluyen el análisis de la problemática actual del cliente, la definición del método para automatizar los deploys, la comparación de alternativas en infraestructura como código (IaC) para la implementación, la realización del código IaC, el análisis de la optimización del tiempo al utilizar IaC y la comparación de los proveedores de servicios en la nube para la implementación.

La solución propuesta en este proyecto incluye la implementación de prácticas de CI/CD y automatización en el proceso de desarrollo de software, utilizando herramientas y tecnologías como Terraform para la gestión de la infraestructura como código. De esta manera, se espera mejorar la eficiencia y la calidad del proceso de desarrollo de software, reduciendo la posibilidad de errores y asegurando un seguimiento adecuado de los cambios.

Infraestructura como código

¿Qué es Infraestructura como código?

La idea detrás de *Infrastructure-as-Code* (IaC) es escribir y ejecutar código que define, implementa, modifica, actualiza y destruye infraestructura. Esto representa un cambio importante en la mentalidad en la que trata todos los aspectos de las operaciones como software, incluso aquellos aspectos que representan hardware (por ejemplo, configurar servidores físicos).
Una de las ideas claves del rol de DevOps es administrar, en lo posible, todo como si fuera código, como servidores, bases de datos, documentación, pruebas automáticas, etc.

Existen 5 categorías principales de IaC:

- Scripts ad hoc
- Herramientas de gestión de la configuración
- Herramientas de plantillas de servidor
- Herramientas de orquestación
- Herramientas de aprovisionamiento

Scripts ad hoc

Es la forma más directa de automatizar tareas. Se basa en tomar cualquier secuencia de tareas que se realizaban manualmente, dividirlas en pasos acotados, definir cada uno de esos pasos en cualquier lenguaje de scripting (Por ejemplo: Bash, PowerShell, Python) y finalmente ejecutarlo en el servidor.

```
#Actualizar los paquetes
apt-get update
#Instalar PHP y Apache
apt-get install - y php apache2
#Clonar el sitio web desde un repositorio Git
git clone github.com/guillegregoret/test-website /var/www/html
#Iniciar el servicio de Apache
service apache2 start
```

Ejemplo 1: Script ad hoc en bash para levantar un sitio PHP en Apache

Como se muestra en el ejemplo 1, los scripts ad hoc son muy simples de crear para realizar tareas sencillas, lo cual también puede ser una de sus desventajas, ya que realizar un script para tareas un poco más complejas implica escribir varias líneas de código personalizado. Sumado a esto, los scripts bash no fuerzan una estructura definida en el código, por lo que scripts que hacen lo mismo pueden tener códigos muy diferentes, cada desarrollador los escribirá a su estilo y de diferentes formas.
Mantener un gran repositorio de scripts bash casi siempre involucra código desordenado e inmantenible. Estos scripts son excelentes para tareas pequeñas, simples y puntuales, pero si se busca administrar toda la infraestructura se debe ir a herramientas IaC que están pensadas para ese propósito.

Herramientas de gestión de la configuración

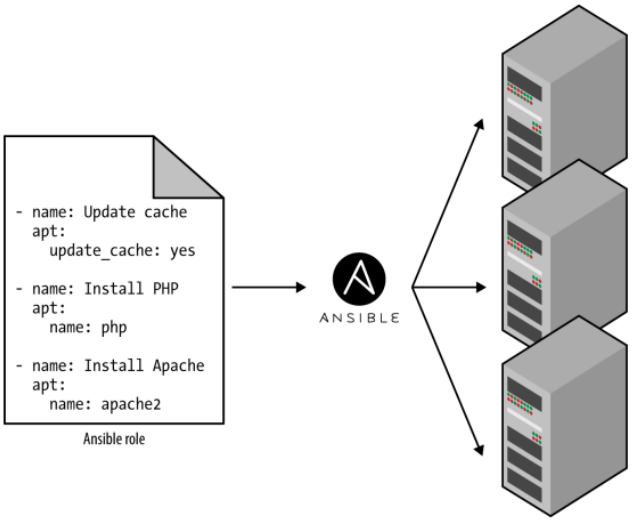
Ansible o Chef son ejemplos de herramientas de gestión de configuración, lo que significa que están diseñados para instalar y administrar software en servidores existentes. Un código en Ansible para configurar el mismo servidor Apache que en el Ejemplo 1:

```
- name: Actualizar la caché de repositorios
  apt: null
  update_cache: 'yes'
- name: Instalar PHP
  apt:
    name: php
- name: Instalar Apache
  apt:
    name: apache2
- name: Clonar el sitio desde el repositorio
        git:     'repo=https://github.com/guillegregoret/test-website
dest=/var/www/html'
- name: Iniciar el servicio de Apache
  service: name=apache2 state=started enabled=yes
```

Ejemplo 2: Ansible Role para levantar un sitio PHP en Apache

El código de este ejemplo se ve bastante similar al ejemplo en Bash, sin embargo herramientas como Ansible ofrecen una serie de ventajas:

- Convenciones de código: Ansible fuerza una sintaxis y estructura puntual, una disposición del código definida y parámetros bien claro, administración de información sensible (secrets management) y demás. Estas convenciones hacen el código más sencillo de comprender, navegar y, de ser necesario, modificar.
- Idempotencia: Realizar un código Ad hoc que funcione una vez no es complicado, realizar un código Ad hoc que funcione cada vez que se lo ejecuta es mucho más complejo, hay que verificar que los paquetes a instalar no se encuentren ya instalados, que los directorios o archivos a crear no existan previamente, que los servicios que se quiere levantar no esten corriendo previamente. Estas verificaciones en un código Bash implicaría varias líneas de condiciones *if*, mientras que la mayoría de las funciones de Ansible son, por defecto, idempotentes, es decir, va a funcionar todas y cada una de las veces que se ejecute el código.
- Distribución: El código Ad hoc está pensado para correr localmente en cada servidor, mientras que las herramientas de gestión de la configuración, como Ansible, están diseñados para administrar grandes cantidades de servidores e instancias remotas.



Por ejemplo, para correr el rol web-server.yml en 4 servidores, primero se crea un archivo hosts que contiene las direcciones IP de los servidores destino:

```
[webservers]
192.168.100.10
192.168.100.20
192.168.100.30
192.168.100.40
```

A continuación se define lo que se conoce como *Ansible Playbook*:

```
- hosts: webservers
  roles:
  - webserver
```

Finalmente corremos la playbook:
`ansible-playbook playbook.yml`
Esto le dice a Ansible que debe configurar los 4 servidores, realizar esta lógica en Bash implicaría un número considerable de líneas de código.

Herramientas de plantillas de servidor

Como alternativa a herramientas de gestión de la configuración, existen herramientas de plantillas de servidor, como Docker, Vagrant y Packer, que en vez de realizar la misma configuración sobre uno o más servidores preexistentes, lanzan servidores ya configurados desde una imagen que ya contiene el sistema operativo, software, los archivos y configuraciones necesarias, todas contenidas en una imagen o “snapshot”.
Existen 2 grandes categorías de imágenes:

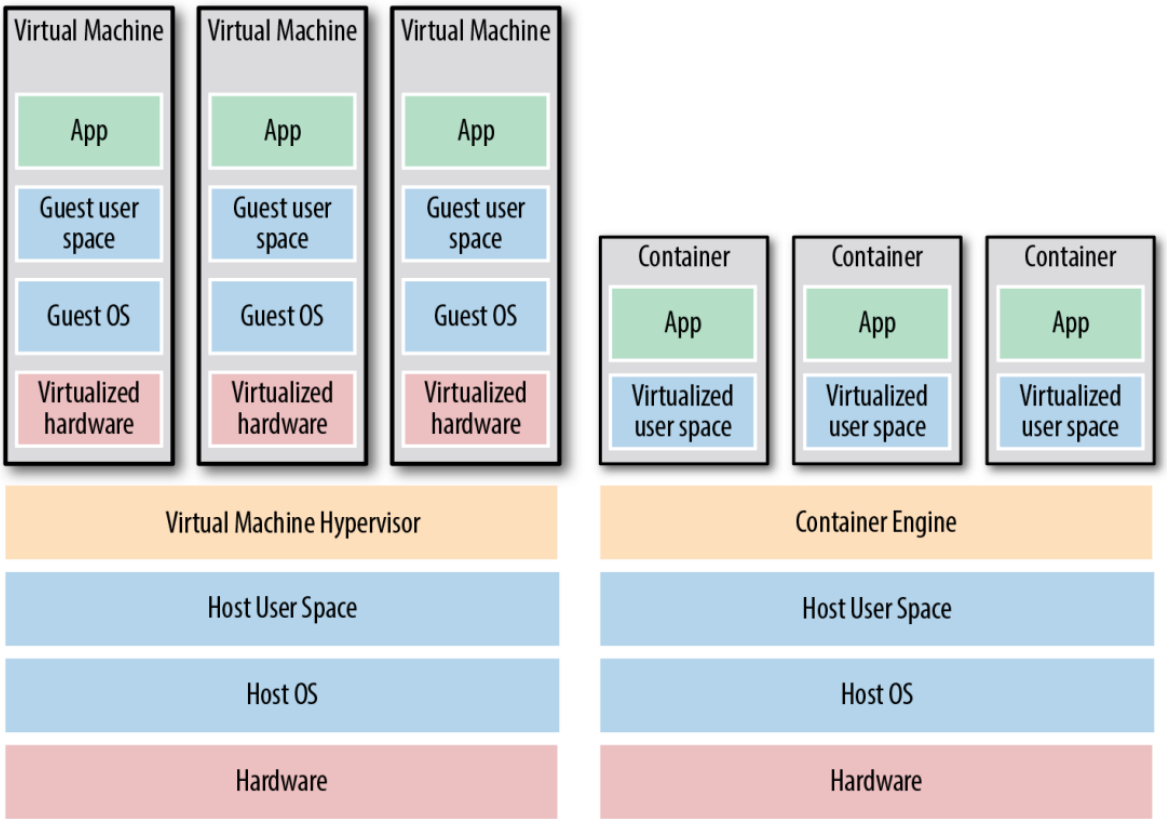
Máquinas Virtuales (VMs):

Las máquinas virtuales ejecutan un sistema operativo completo, incluido su propio kernel, sobre un hardware virtualizado gracias a un hipervisor. La desventaja de una VM es que suele ser bastante más pesada que un container, además es más lenta de levantar y el rendimiento se ve afectado debido a la emulación del hardware sobre el cual corre todo el sistema. Una de las ventajas es que la máquina virtual corre completamente aislada de las

demás VMs que puedan estarse ejecutando en el host. Otra ventaja es que al emular el hardware subyacente, se puede correr la misma VM en diferentes entornos y va a funcionar de la misma manera.

Containers

La filosofía de los contenedores es diferente a la de las VMs. Si bien tratan también de aislar a las aplicaciones y de generar un entorno replicable, en lugar de albergar un sistema operativo completo lo que hacen es compartir los recursos del propio sistema operativo del host sobre el que se ejecutan, lo que les permite iniciar en cuestión de milisegundos.
Docker funciona a partir de imágenes que se pueden reutilizar entre varias aplicaciones. Cada una de esas imágenes funciona como un sistema de capas que se puede superponer unas a otras para formar un sistema de archivos que es la combinación de todas ellas. Por ejemplo, una capa puede llevar las bibliotecas o runtimes que necesitemos utilizar (como Node.js o PHP), otra con unas bibliotecas determinadas de las que hace uso nuestra aplicación, y otra capa final con el código de nuestra aplicación. La combinación resultante de todas estas capas es una nueva imagen, única para nuestra aplicación.



Arquitectura de una Máquina Virtual vs Containers

Herramientas de orquestación

Las herramientas de plantillas de servidor son excelentes para crear máquinas virtuales y contenedores, pero ¿cómo realmente administrarlos?, el número de recursos a monitorear puede crecer rápidamente, algunas tareas comunes suelen ser:

- Crear VMs y containers, haciendo uso eficiente de la infraestructura

- Implementar actualizaciones en una flota existente de máquinas virtuales y contenedores con estrategias como rolling deployment o blue-green deployment.
- Supervisar el estado de sus máquinas virtuales y contenedores y reemplace automáticamente insalubres (Autohealing).
- Escale el número de máquinas virtuales y contenedores hacia arriba o hacia abajo en respuesta a la carga (Autoscaling).
- Distribuir el tráfico (Load Balancing)
- Permita que VMs y containers se puedan encontrar (Service Discovery)

Manejar estas tareas es el campo principal de las herramientas de orquestación, como Kubernetes, Docker Swarm o Amazon Elastic Containers Service (AWS ECS). Por ejemplo, Kubernetes permite definir los containers Docker como código en lo que se llama un “deployment”

```
apiVersion: apps/v1
#Un deployment permite desplegar varias réplicas de containers Docker y
declarativamente lanzar actualizaciones de estos
kind: Deployment
#Nombre del Deployment
metadata:
 name: web-app
#Especificación de la configuración del Deployment
spec:
#El selector le especifica al Deployment cómo encontrar los containers
 selector:
   matchLabels:
     app: web-app
#Se definen 3 replicas del container
replicas: 3
#Define la estrategia de distribución de actualizaciones
strategy:
 rollingUpdate:
   maxSurge: 3
   maxUnavailable: 0
 type: RollingUpdate
#Se define la plantilla para definir que containers desplegar
template:
 metadata:
   labels:
     app: web-app
#Especificación de los containers que se van a correr en el deployment
 spec:
```

```
  containers:
    - name: web-app
      image: 'httpd:2.4.54'
      ports:
        - containerPort: 80
```

Un deployment define lo siguiente:

- Un container o más que corren juntos, esto se llama “Pod”
- Las configuraciones que tendrá cada container dentro del Pod.
- Cómo manejar las actualizaciones, si hay una versión nueva del container, levanta las 3 copias y espera a que empiecen a responder antes de detener las réplicas anteriores.
- Cuantas copias o réplicas del Pod correr en el cluster, en este caso, 3. Kubernetes se encarga automáticamente de distribuir los Pods según la disponibilidad de recursos. También monitorea la salud de cada Pod y los reemplaza ante un crash o falta de respuestas

Los deployments en Kubernetes son una herramienta potente en muy pocas lineas de código. Para correr el archivo solo es necesario ejecutar `kubectl apply -f deployment.yml` y en pocos segundos estará la infraestructura definida corriendo, luego se pueden realizar modificaciones al archivo y aplicarlos corriendo nuevamente `kubectl apply`.

Herramientas de aprovisionamiento

Mientras que las últimas 3 herramientas definen código que corre en cada servidor, herramientas de aprovisionamiento como Terraform o CloudFormation son los encargados de crear los servidores donde el código va a correr, y no solo servidores, sino diferentes elementos de infraestructura como bases de datos, balanceadores de carga, firewalls, monitoreo, redes y subredes, emitir certificados SSL y casi cualquier aspecto de infraestructura. Un simple ejemplo de Terraform para levantar un servidor web Apache sería el siguiente:

```
resource "aws_instance" "app" {
 instance_type = "t2.micro"
 availability_zone = "us-east-2a"
 ami = "ami-0c55b159cbfafe1f0"
 user_data = <<-EOF
             #!/bin/bash
             sudo service apache2 start
             EOF
}
```

Se detalla el tipo de instancia, en que zona o región de disponibilidad se creará, que ami (Amazon Machine Image) y un simple script que corre al iniciar el sistema operativo.

Beneficios de IaC

Utilizar infraestructura definida como código, permite utilizar una gran cantidad de prácticas de ingeniería de software que permite mejorar drásticamente el proceso de entrega, por ejemplo:

Autoservicio

Las implementaciones de código automatizadas son una forma poderosa de reducir el costo de implementar nuevas funciones, así como de aumentar la velocidad y la agilidad del desarrollador. Al proporcionarles a los desarrolladores una manera fácil de implementar su código, pueden tomar posesión de su propio proceso de implementación, lo que reduce las barreras de entrada, aumenta la productividad y permite que la innovación suceda más rápido.
A medida que crezcan sus necesidades, deberá aumentar la cantidad de personas que tienen acceso a su infraestructura de producción o compartirla entre varios equipos. Si no tiene un proceso documentado, esto puede generar confusión e ineficiencia. Asegurarse de que todo su código esté correctamente versionado y compartido con todos le dará la tranquilidad de no repetir pasos y errores en todas las implementaciones.

Documentación

Todo el estado de la infraestructura está definido en código, en vez de en la cabeza del sysadmin que los crea. Esto hace que la IaC sean archivos que cualquiera puede acceder y modificar.

Versionado

Como el código de cualquier software, la infraestructura como código se puede almacenar en repositorios y tener versionado cualquier cambio hecho, facilitando seguir o revertir modificaciones.

Reutilización

Herramientas de IaC como Terraform permiten crear módulos, lo que permite que en vez de desarrollar desde cero, se puede desarrollar desde módulos previamente creados, ya bien conocidos y testeados.

Validación

Como la infraestructura es código, es sencillo hacer pasar cualquier cambio por una herramienta de validación o tests automatizados, prácticas que aportan a reducir las posibilidades de errores o defectos.

Cómo funciona Terraform

Terraform utiliza un lenguaje de configuración propio, llamado HashiCorp Configuration Language (HCL), que permite a los usuarios definir y gestionar recursos de infraestructura de forma sencilla y clara.

Para utilizar Terraform, los usuarios deben crear un archivo de configuración que define los recursos de infraestructura que desean provisionar. Este archivo de configuración es un

archivo de texto plano que utiliza el lenguaje HCL para describir los recursos y sus propiedades. Una vez que se ha creado el archivo de configuración, los usuarios pueden utilizar Terraform para provisionar los recursos de infraestructura especificados en la plataforma de nube o sistema operativo de su elección.

Terraform es capaz de provisionar una amplia variedad de recursos de infraestructura, como máquinas virtuales, bases de datos, contenedores, redes y muchos otros. Además, Terraform es compatible con una amplia variedad de plataformas de nube, como AWS,

Conceptos principales de Terraform

State

El estado de Terraform es un archivo que almacena la configuración actual de la infraestructura gestionada por Terraform. Este archivo se utiliza para sincronizar la configuración actual de la infraestructura con la configuración deseada descrita en el archivo de configuración de Terraform.
Cuando se ejecuta Terraform, se comparan los recursos actualmente provisionados con los recursos descritos en el archivo de configuración y se determina qué cambios deben realizarse para que la infraestructura cumpla con la configuración deseada. Estos cambios se registran en el estado de Terraform y se utilizan para aplicar los cambios de forma segura y controlada.
El estado de Terraform se almacena en un archivo local por defecto, pero también se puede almacenar en una ubicación remota, como un repositorio de Git o un almacenamiento en la nube, para facilitar la colaboración y la gestión del ciclo de vida de la infraestructura.

Providers

Los proveedores o providers de Terraform son componentes de software que permiten a Terraform interactuar con diferentes plataformas de nube y sistemas operativos para aprovisionar y gestionar recursos de infraestructura. Cada proveedor de Terraform tiene un conjunto de recursos y operaciones que pueden ser utilizados por Terraform para interactuar con la plataforma de nube o sistema operativo correspondiente.

Por ejemplo, el proveedor de Terraform para AWS permite a Terraform provisionar y gestionar recursos en Amazon Web Services, como máquinas virtuales, bases de datos y contenedores.

Modulo

Los módulos de Terraform son fragmentos reutilizables de código que se utilizan para encapsular lógica de configuración de infraestructura y hacerla más fácil de comprender y mantener. Los módulos de Terraform permiten a los usuarios definir un conjunto de recursos de infraestructura y sus propiedades de forma independiente y luego reutilizar esa configuración en diferentes archivos de configuración de Terraform.

Variable

Las variables en Terraform son valores que se pueden utilizar en el archivo de configuración de Terraform para personalizar la configuración de infraestructura y hacerla más flexible y

reutilizable. Las variables en Terraform se pueden utilizar para proporcionar valores de entrada a la configuración de Terraform y para obtener valores de salida de la infraestructura provisionada.

Ciclo de vida de Terraform

`terraform init` inicializa el directorio y los archivos de configuración necesarios para terraform

`terraform plan` lee todos los archivos de texto y arma una planificación de cómo crear la infraestructura en el provider.

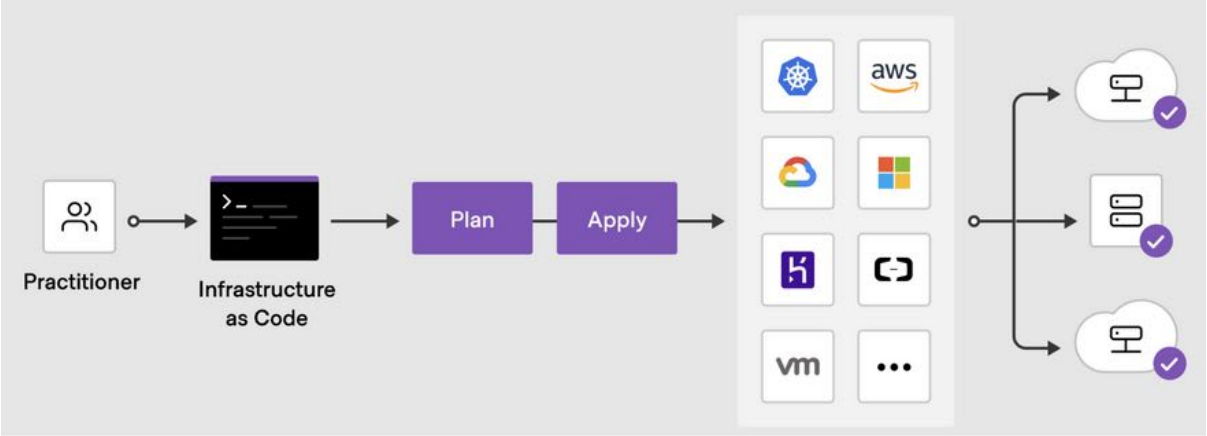
`terraform apply` ejecuta el plan anterior y crea los recursos.

`terraform destroy` elimina todos los recursos relacionados con el state actual.

Table 1-4. A comparison of the most common way to use the most popular IaC tools

	Source	Cloud	Type	Infrastructure	Language	Agent	Master	Community	Maturity
Chef	Open	All	Config Mgmt	Mutable	Procedural	Yes	Yes	Large	High
Puppet	Open	All	Config Mgmt	Mutable	Declarative	Yes	Yes	Large	High
Ansible	Open	All	Config Mgmt	Mutable	Procedural	No	No	Huge	Medium
SaltStack	Open	All	Config Mgmt	Mutable	Declarative	Yes	Yes	Large	Medium
CloudFormation	Closed	AWS	Provisioning	Immutable	Declarative	No	No	Small	Medium
Heat	Open	All	Provisioning	Immutable	Declarative	No	No	Small	Low
Terraform	Open	All	Provisioning	Immutable	Declarative	No	No	Huge	Low

Terraform puede provisionar infraestructura a través de proveedores de nube pública como Amazon Web Services, Azure, Google Cloud y DigitalOcean, así como plataformas de nube privada y virtualización como Open Stack y VMWare.


Practitioner
Infrastructure as Code
Plan
Apply
aws
vm

Integración Continua vs Entrega Continua vs Despliegue Continuo

Un pipeline de entrega continua genera releases del código fuente de forma rápida automatizada y de manera reproducible. El diseño general para como esto se lleva a cabo se llama “Entrega Continua” *(“Continuous Delivery”)*. El proceso que inicia el “proceso de ensamblado” al proveer el código se llama “Integración Continua” *(“Continuous Integration”)*. El proceso que se encarga de hacer que el producto llegue al usuario es “Despliegue continuo” *(“Continuous Deployment”).*

Si se realiza correctamente, un pipeline minimiza las barreras, las transferencias y la confusión entre los equipos de desarrollo y los equipos operativos, objetivos que forman parte del enfoque de DevOps para la entrega de software.

Continuous Integration

Los cambios de un desarrollador se fusionan y validan. El objetivo de CI es validar rápidamente estos cambios de código enviados. El resultado previsto es identificar cualquier problema en el código y notificar automáticamente al desarrollador. Esto ayuda a garantizar que la base del código no se rompa más de lo necesario.

Tests unitarios

La verificación del código se lleva a cabo mediante lo que se conoce como Tests Unitarios y los desarrolladores deben estar encargados de crearlos. Estos tests verifican pequeñas partes (unidades) de código, dándole una entrada y verificando la salida, si la salida es la esperada el test se “aprueba”.

Ejemplos de herramientas CI:

- Jenkins
- JetBrains TeamCity
- GitHub Actions
- Bitbucket Pipelines
- GitLab CI

Prechecks

Luego de hacer un fork de un repositorio Git para contribuir o modificar el código, se puede hacer un push al repositorio original, en ese momento el dueño del repositorio puede ver y revisar las modificaciones que tiene el código de este push comparado con el código original y decidir si acepta hacer un merge a la rama definida, esta función se conoce como Pull Request (PR) o también Merge Request.

Continuous Delivery

Refiere a la cadena de procesos que obtiene automáticamente los cambios del código y los ejecuta a través de las operaciones de compilación, testing y empaquetado para producir una versión implementable. Por lo general, se hace con poca o nula intervención humana.

Esta etapa recibe los cambio recibidos por la etapa de CI y ejecuta el resto del proceso de despliegue
Si bien podemos pensar en el único entregable de un pipeline como un conjunto de código desplegable, en el camino se crean salidas intermedias clave. De hecho, una de las cosas clave que suceden en el despliegue es que los nuevos cambios (validados y fusionados durante la CI) se combinan con otro código con el que necesitan trabajar (o del que pueden depender) para producir “Artifacts”.

Los Artifacts

Un Artifacts es un artículo que es un entregable (algo directamente utilizado por el producto final) o incluido en un entregable. Por ejemplo, un ZIP que contiene una serie de archivos comprimidos.

Versionado

Independientemente del tipo de Artifact, a medida que avanza a lo largo del pipeline, se debe versionar a través de un proceso como el control de versiones semántico.
Aunque la tecnología de implementación ha hecho que el versionado general del producto sea una preocupación menor para los usuarios, para los desarrolladores, evaluadores y algunos procesos automatizados, no siempre es deseable actualizar a las últimas versiones de inmediato. Las operaciones como la prueba y la depuración pueden requerir versiones estables de artefactos que no cambien cada vez que se ejecute un pipeline. Y para rastrear los problemas por completo, la versión del artefacto debe poder rastrearse hasta el conjunto exacto de código fuente que se usó para producirlo.

Continuous Deployment

La implementación continua se refiere a poder tomar una versión de código que salió del pipeline y ponerlo automáticamente a disposición de los usuarios finales. (Un pipeline que incluye esto generalmente se denomina “deployment pipeline”). Según la forma en que los usuarios pretendan "instalar" el código, la implementación puede significar la implementación automática en una nube, hacer que una actualización esté disponible, actualizar un sitio web o simplemente actualizar la lista de versiones disponibles.
Tener que revertir o deshacer una implementación para todos los usuarios puede ser una situación costosa (tanto técnicamente como en la percepción de los usuarios). Por lo tanto, si se implementa o no una versión de una ejecución de un pipeline, puede depender de decisiones humanas. Estos pueden basarse en varios métodos empleados para "probar" una versión antes de implementarla por completo.
Se han desarrollado numerosas técnicas para permitir probar implementaciones de nuevas funciones y deshacerlas fácilmente si se encuentran problemas. Algunas de las cuales se detallan a continuación:

Blue/Green Deployment

Green/Blue Deployment es un modelo de lanzamiento de aplicaciones que transfiere gradualmente el tráfico de usuarios de una versión anterior de una aplicación o microservicio a una nueva versión casi idéntica, las cuales se ejecutan en producción.

La versión anterior puede denominarse entorno “blue”, mientras que la nueva versión puede denominarse entorno “green”. Una vez que el tráfico de producción se transfiere por completo de azul a verde, el azul puede quedar en espera en caso de revertir el despliegue.

Canary Deployment

En ingeniería de software, Canary deployment es la práctica de realizar lanzamientos por etapas. Primero implementamos una actualización de software para una pequeña parte de los usuarios, para que puedan probarla y proporcionar comentarios. Una vez aceptado el cambio, la actualización se extiende al resto de usuarios.

Feature Toggles

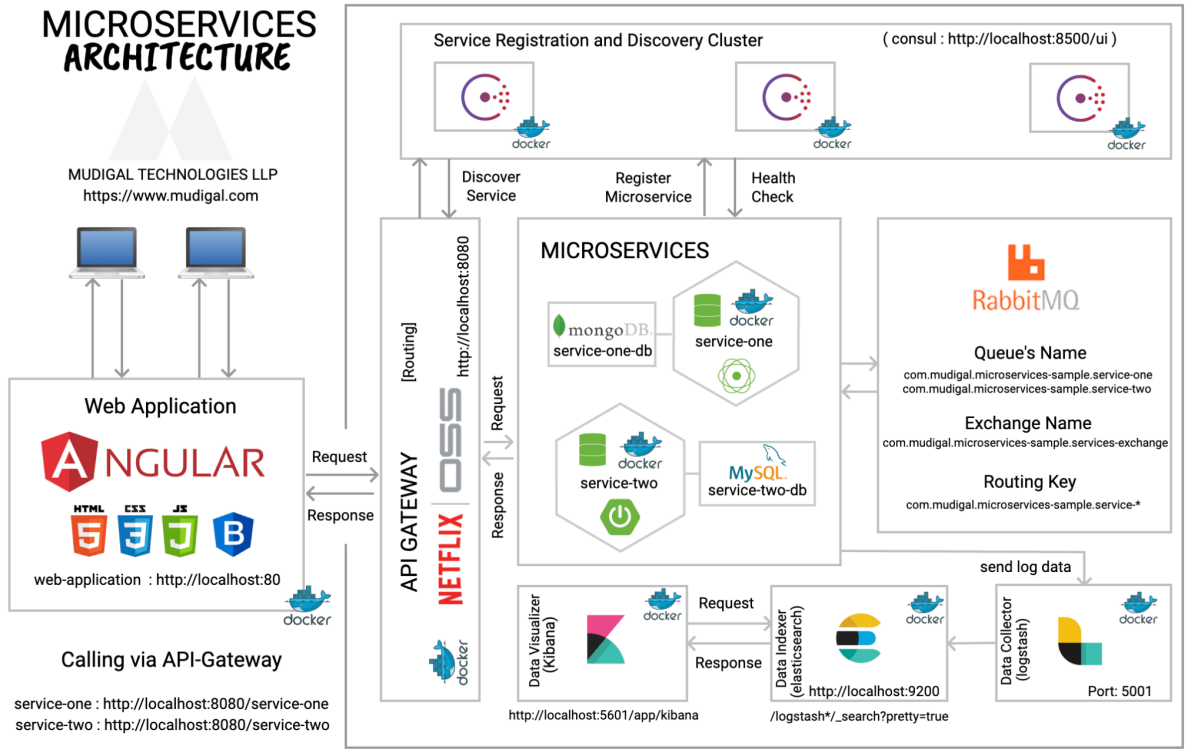
Feature toggles, conocidas también como feature flags, son una técnica muy potente que permiten a los equipos de desarrollo modificar el comportamiento de un sistema sin cambiar el código. Esto permite a los desarrolladores tomar el control del lanzamiento de sus funcionalidades y retirar funcionalidades que sean inestables o no rindan bien.
Las feature toggles son principalmente variables que se usan dentro de declaraciones condicionales. Por lo tanto, los bloques que encontramos dentro de estas declaraciones condicionales se pueden activar o desactivar según el valor de las feature toggles.

Dark Launch

El dark launch o dark release es un término que se refiere a lanzar sus funciones a un subconjunto de usuarios para recopilar sus comentarios y mejorar sus nuevas funciones en consecuencia. Por lo tanto, es una forma de implementar una función pero limitar el acceso a ella para obtener comentarios útiles.
Puede pensar en ello como una forma segura de lanzar sus funciones a un pequeño grupo de usuarios para probar si les gusta esta nueva función.

Caso práctico

Partiendo de la siguiente infraestructura como ejemplo, que consiste en una arquitectura de microservicios con sus propias bases de datos SQL y NoSQL, que se comunican a través de una cola de mensajes RabbitMQ, utilizan un service discovery con Consul, realizan el envío de logs con el stack de Elastic y cuenta con un frontend Angular, plantamos la migración de esta infraestructura a la nube aprovechando todas las ventajas que ofrece este entorno y utilizando infraestructura como código.



Link al repositorio en GitHub: https://github.com/mudigal-technologies/microservices-sample

Cada uno de los microservicios denominados service-one y service-two implementa un mecanismo de generación de identificadores únicos universales (UUID) de manera aleatoria cada 60 segundos. Estos identificadores son almacenados en la base de datos correspondiente a cada servicio y son compartidos con el otro servicio mediante el uso de RabbitMQ, un sistema de mensajería. Además, cada servicio emite un registro de actividad a través de Logstash que se envía a Elasticsearch cada vez que se genera o recibe un nuevo UUID.

El formato de respuesta obtenida de service-one es:

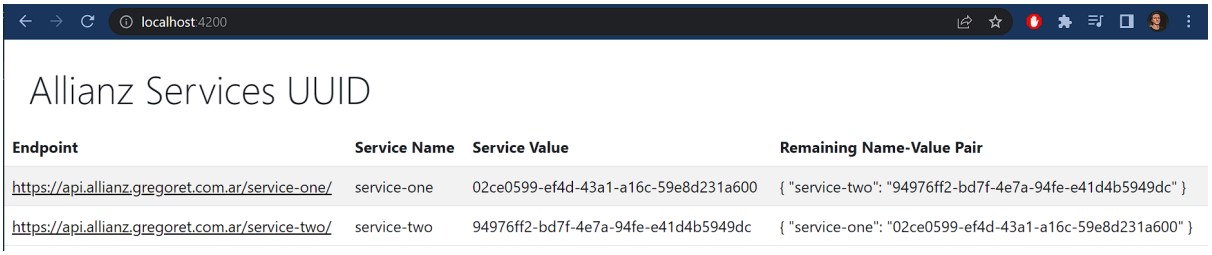
```
{"remainingNameValuePair":{"service-two":"94976ff2-bd7f-4e7a-94fe-e41d4b5949dc"},"originalName":"service-one","originalValue":"02ce0599-ef4d-43a1-a16c-59e8d231a600"}
```

El formato de respuesta obtenida de service-two es:

```
{"originalName":"service-two","originalValue":"94976ff2-bd7f-4e7a-94fe-e41d4b5949dc","remainingNameValuePair":{"service-one":"02ce0599-ef4d-43a1-a16c-59e8d231a600"}}
```

Como se puede apreciar, ambos servicios muestran correctamente en cualquier instante de tiempo tanto su UUID como el del servicio opuesto, demostrando que la comunicación mediante cola de mensajes es correcta.

La interfaz de usuario del proyecto se caracteriza por su simplicidad y tiene como objetivo principal presentar en una única pantalla los pares de valores correspondientes a cada endpoint. Además, la pantalla se actualiza automáticamente cada 5 segundos para mostrar los valores más recientes.



En las siguientes tablas se plantea cuales son las alternativas en cada una de los 3 principales proveedores de servicios en la nube, Google Cloud, Azure y Amazon Web Services, para implementar la infraestructura propuesta.

Google Cloud Platform

Servicio	Alternativa	
Container Service	Docker	GCP Cloud Run
Cola de mensajes	RabbitMQ	GCP Cloud Pub/Sub o Cloud Tasks
Service Discovery	Hashicorp Consul	-
API Gateway	Netflix Zuul	GCP API Gateway
Database	MySQL	GCP Cloud SQL
	MongoDB	GCP Datastore o Cloud Bigtable
Data Collector	Logstash	GCP Cloud Logging
Data Indexer	ElasticSearch	-
Data Visualizer	Kibana	GCP Cloud Monitoring
In-Memory Data Storage	Redis	GCP Memorystore
Load Balancing	Nginx LB	GCP Cloud Load Balancing
Certificate Issue	Let’s Encrypt Certbot	GCP Certificate Authority Service

Firewall	NGINX ModSecurity WAF	Google Cloud Armor
Almacenamiento		GCP Cloud Storage o Persistent Disk o Filestore
Registro de contenedores	DockerHub	GCP Container Registry
IAM	KeyCloak	Firebase Authentication
Servicio DNS	Cloudflare	GCP Cloud DNS
Content Distribution Network	Cloudflare CDN	GCP Cloud CDN

Azure

	Servicio	Azure
Container Service	Docker	Azure Container Instances
Cola de mensajes	RabbitMQ	Azure Service Bus, Azure Storage Queues
Service Discovery	Hashicorp Consul	-
API Gateway	Netflix Zuul	Azure API Management
SQL Database	MySQL	Azure Database for MySQL
NoSQL Database	MongoDB	Azure Cosmos DB
Data Collector	Logstash	Azure Monitor
Data Indexer	ElasticSearch	-
Data Visualizer	Kibana	Azure Monitor
In-Memory Data Storage	Redis	Azure Cache
Load Balancing	Nginx LB	Azure Load Balancer
Certificate Issue	Let's Encrypt Certbot	Azure App Service Managed Certificate
Firewall	NGINX ModSecurity WAF	Azure WAF, Azure Front Door
Almacenamiento		Azure Blob Storage o Disk Storage o Azure Files
Registro de contenedores	DockerHub	Azure Container Registry
IAM	KeyCloak	Azure Active Directory Service
Servicio DNS	Cloudflare	Azure DNS
Content Distribution Network	Cloudflare CDN	Azure CDN

AWS

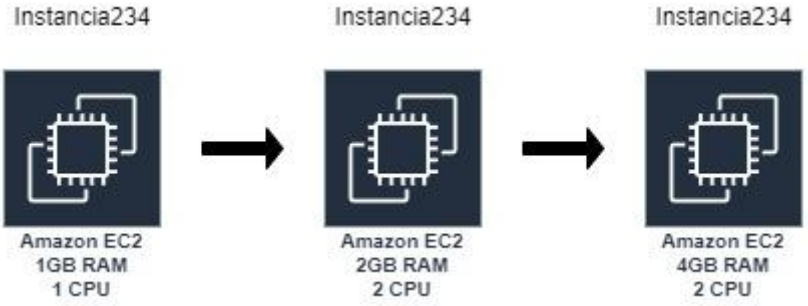
	Servicio	aws
Container Service	Docker	AWS ECS
Cola de mensajes	RabbitMQ	AWS MQ o SQS
Service Discovery	Hashicorp Consul	AWS Cloud Map
API Gateway	Netflix Zuul	AWS API Gateway
SQL Database	MySQL	AWS RDS o AWS Aurora
NoSQL Database	MongoDB	AWS DocumentDB
Data Collector	Logstash	AWS Cloudwatch
Data Indexer	ElasticSearch	AWS OpenSearch
Data Visualizer	Kibana	AWS OpenSearch Dashboards
In-Memory Data Storage	Redis	AWS MemoryDB for Redis
Load Balancing	Nginx LB	AWS Application Load Balancer
Certificate Issue	Let's Encrypt Certbot	AWS Certificate Manager
Firewall	NGINX ModSecurity WAF	AWS WAF on ALB
Almacenamiento		AWS S3 o AWS EFS o AWS EBS
Registro de contenedores	DockerHub	AWS ECR
IAM	KeyCloak	AWS Cognito
Servicio DNS	Cloudflare	AWS Route53
Content Distribution Network	Cloudflare CDN	AWS CloudFront

Se ha optado por utilizar AWS debido a las diversas ventajas que ofrece esta plataforma de servicios en la nube en términos de escalabilidad, flexibilidad y seguridad.

¿Por qué Amazon Web Services?

Escalado Vertical

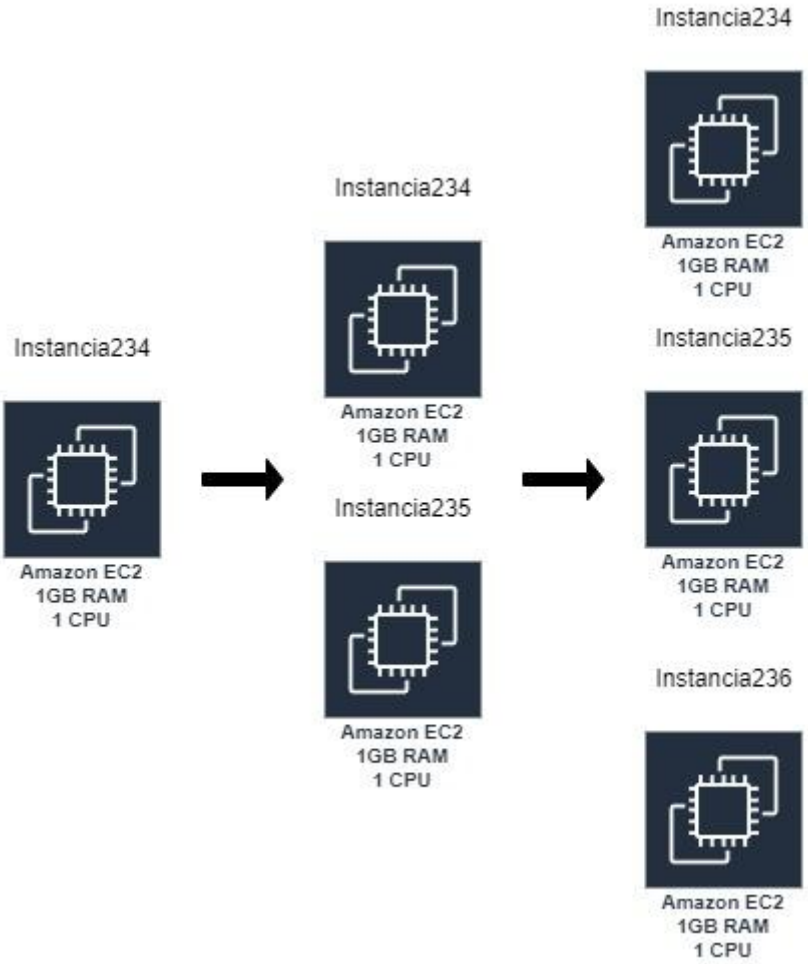
Es la práctica de agregar más capacidad de procesamiento o memoria a un servidor para mejorar la performance o rendimiento final de la aplicación, es una opción sencilla en la teoría pero costosa en la práctica si se trata de un entorno "on-premises". Al migrar a AWS, existen las suscripciones *"pago-por-uso" o "pay-as-you-go"*, que consisten en un modelo de pago dinámico (se paga por lo que se consume). Esto es muy útil ya que en la infraestructura "on-premises" siempre se debe sobreestimar costosos recursos de hardware que no se utilizaran en su totalidad a lo largo de su vida útil, mientras que en los costos dinámicos en la nube se estima un mínimo o piso de recursos necesarios para el funcionamiento de la aplicación y luego se paga incrementalmente por más consumo en los momentos de alto uso de la aplicación.



Ejemplo de escalado vertical en una instancia EC2

Escalado Horizontal

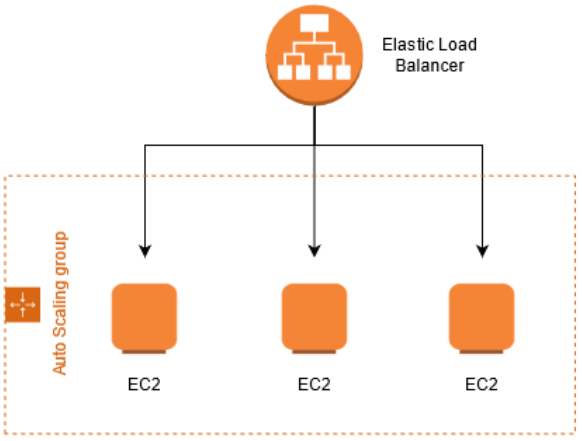
A diferencia del escalado vertical, que agrega más recursos al mismo servidor para hacer frente a la carga, el escalado horizontal agrega más servidores para cumplir con los requerimientos de recursos. Nuevamente, utilizando el esquema de pago-por-uso se pueden lograr grandes reducciones en los costos al usar escalado.
Hay que tener en cuenta un detalle importante para implementar escalado horizontal: La aplicación debe ser serverless, es decir, no hay archivos o configuraciones específicas dentro de cada instancia, permitiendo que estas se creen y destruyan sin efectos adversos al funcionamiento de la aplicación.
En el caso de AWS, se ofrecen 2 servicios elementales para el escalado: Elastic Load Balancing y Auto Scaling.



Ejemplo de escalado horizontal de un grupo de instancias EC2

Elastic Load Balancing

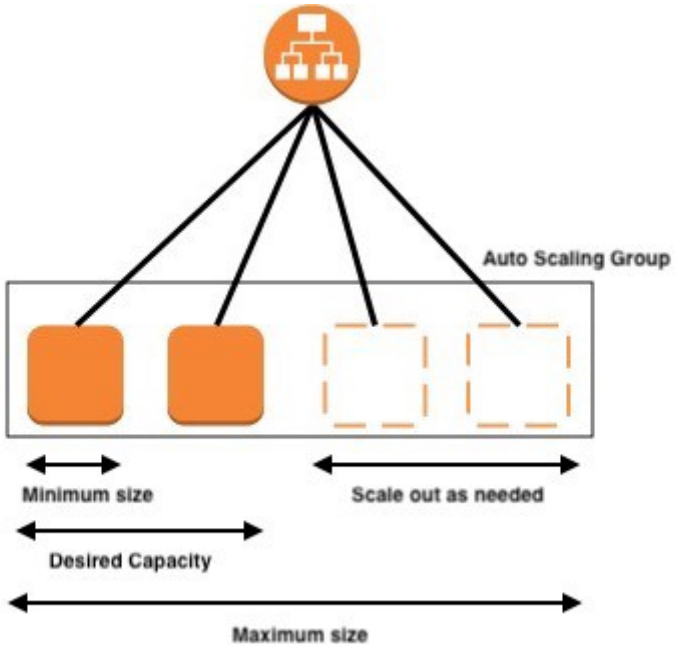
Si se quisiera hacer un escalado horizontal sobre una infraestructura on-premises se requeriría un balanceador de carga, que implica un costo de capital y mantenimiento. En el caso de AWS se simplifica la tarea ya que se basa en tecnología serverless[1], no hay costos por adelantado y solo se paga por lo que se usa, lo que hace el escalado horizontal en la nube más atractivo y accesible.



[1] Un servicio que no requiere que se administre la infraestructura subyacente.

Auto Scaling

Es un servicio que contiene la lógica para controlar la expansión y contracción del grupo de servidores, sin esta función, los costos serían estáticos ya que la asignación de recursos no variaría en el tiempo. Auto Scaling usa varios disparadores para decidir el aumento o disminución de la capacidad en el grupo de servidores, por ejemplo: Uso de CPU, uso de memoria, cantidad de operaciones de lectura/escritura en disco.
Otra alternativa es, si se conoce con anticipación que habrá un aumento en la carga, programar el autoescalado. Esto es utilizado en casos previsibles, por ejemplo, el software de facturación de una empresa hace un gran uso de recursos en los primeros días del més, no así el resto, en estos casos se puede programar el escalado de recursos para cumplir los requerimientos de carga en la ventana definida de tiempo, optimizado el uso de recursos.
Auto Scaling también permite definir una mínima cantidad servidores activos, trabajando como un orquestador de alta disponibilidad, asegurando que un mínimo de capacidad de cómputo definido siempre esté disponible para servir a los clientes de la aplicación.



Además de lo mencionado anteriormente, Amazon provee los siguientes puntos a destacar:

1. Amplia gama de servicios: AWS proporciona una amplia variedad de servicios y herramientas, que van desde el almacenamiento, procesamiento, análisis de datos, aprendizaje automático, seguridad, entre otros.
2. Escalabilidad: AWS ofrece una escalabilidad flexible y rápida, lo que permite a las empresas adaptarse a las necesidades cambiantes de sus usuarios y al volumen de sus negocios.
3. Alta disponibilidad: AWS proporciona un alto nivel de disponibilidad en sus servicios, lo que minimiza el riesgo de interrupciones y garantiza la continuidad del negocio.
4. Seguridad: AWS tiene una gran preocupación por la seguridad, por lo que proporciona una amplia gama de herramientas y servicios de seguridad, como autenticación de usuarios, encriptación de datos, protección contra DDoS, entre otros.

5. Costo-efectividad: AWS ofrece un modelo de pago por uso que permite a las empresas pagar solo por los servicios que utilizan, lo que ayuda a reducir los costos de infraestructura.
6. Integración: AWS permite la integración con una amplia variedad de herramientas y servicios de terceros, lo que permite a las empresas crear soluciones personalizadas y escalables.

A continuación se presenta una descripción detallada del funcionamiento de cada servicio disponible en Amazon Web Services, así como servicios alternativos. Además, se incluyen algunas ventajas y desventajas que se han tenido en cuenta para la selección final de la implementación más adecuada.

Microservicios

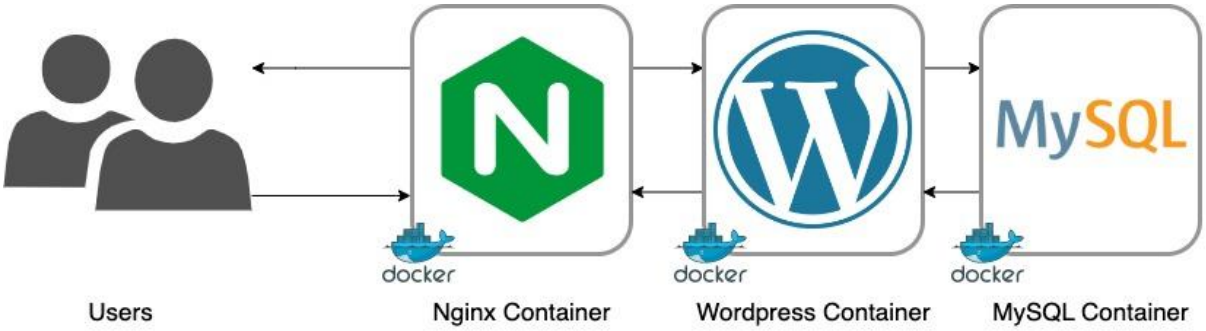
La arquitectura de microservicios es un enfoque de diseño de aplicaciones que se basa en la división de una aplicación en pequeños servicios autónomos que se pueden desarrollar, implementar y escalar de forma independiente. Cada microservicio se encarga de una tarea específica y se comunica con otros microservicios a través de interfaces establecidas, como APIs web.
La arquitectura de microservicios tiene varias ventajas en comparación con otras arquitecturas, como la arquitectura monolítica. Una de las principales ventajas es que permite una mayor flexibilidad y escalabilidad, ya que cada microservicio se puede desarrollar y escalar de forma independiente. Esto también facilita el mantenimiento y la actualización de la aplicación, ya que se pueden realizar cambios en un microservicio sin afectar a los demás. Además, la arquitectura de microservicios permite una mayor reutilización de código y una mayor colaboración entre equipos de desarrollo, ya que cada microservicio puede ser desarrollado por un equipo diferente.

Docker

Docker es la principal plataforma de código abierto para crear, implementar y ejecutar aplicaciones en contenedores. Los contenedores son una forma de empaquetar una aplicación y sus dependencias, de modo que se pueda ejecutar en cualquier entorno de manera predecible y aislada.
Docker utiliza un enfoque de contenerización, que implica empaquetar una aplicación y sus dependencias, como bibliotecas y otros archivos binarios, en una sola imagen. Esta imagen de contenedor se puede ejecutar en cualquier infraestructura que admita Docker, incluidos servidores locales, máquinas virtuales y plataformas en la nube.
Docker brinda varios beneficios, incluida la capacidad de ejecutar aplicaciones de manera liviana y portátil, eficiencia mejorada y utilización de recursos, y mejor colaboración y abstracción entre los equipos de desarrollo y operaciones. Estos beneficios hacen de Docker una opción popular para crear e implementar aplicaciones modernas.

Docker y los microservicios a menudo se usan juntos para crear e implementar aplicaciones modernas. Los microservicios son un enfoque de arquitectura de software en el que una aplicación se compone de componentes pequeños, independientes y modulares que se comunican entre sí a través de API bien definidas. Este enfoque permite una mejor

escalabilidad, flexibilidad y capacidad de mantenimiento de las aplicaciones. Docker y los microservicios funcionan bien juntos porque los contenedores proporcionan un entorno de tiempo de ejecución ligero y portátil para los microservicios, esto permite empaquetar e implementar cada microservicio de forma independiente, sin afectar a los demás componentes de la aplicación.



Ejemplo de arquitectura de un sitio wordpress con containers

AWS Lambda

AWS Lambda es un servicio serverless que ejecuta su código en respuesta a eventos y administra automáticamente los recursos informáticos subyacentes. Estos eventos pueden incluir cambios de estado o una actualización, como que un usuario coloque un artículo en un carrito de compras en un sitio web de comercio electrónico. Puede utilizar AWS Lambda para ampliar otros servicios de AWS con lógica personalizada o crear sus propios servicios de back-end que funcionan con la escala, el rendimiento y la seguridad de AWS. AWS Lambda ejecuta código automáticamente en respuesta a múltiples eventos, como solicitudes HTTP a través de Amazon API Gateway, modificaciones de objetos en depósitos de Amazon Simple Storage Service (Amazon S3), actualizaciones de tablas en Amazon DynamoDB y transiciones de estado en AWS Step Functions.
Lambda ejecuta su código en una infraestructura informática de alta disponibilidad y realiza toda la administración de sus recursos informáticos. Esto incluye el mantenimiento del servidor y del sistema operativo, el aprovisionamiento de capacidad y el escalado automático, la implementación de parches de seguridad y código, y la supervisión y el registro de código. Todo lo que necesita hacer es proporcionar el código.

En general, AWS Lambda es una buena opción para aplicaciones sin servidor basadas en eventos, mientras que Amazon ECS es una mejor opción para aplicaciones que requieren la flexibilidad y el control de los contenedores.

AWS ECS

Amazon Elastic Container Service es un servicio de administración de contenedores muy escalable. Se puede utilizar para ejecutar, detener y administrar contenedores en un clúster. En ECS, los contenedores se definen en una definición de tarea (task definition) que se utiliza para ejecutar tareas individuales o tareas dentro de un servicio. En este contexto, un servicio es una configuración que se puede usar para ejecutar y mantener un número determinado de tareas simultáneamente en un clúster. Un poco más de detalle:

Task Definition

Es una colección de 1 o más configuraciones de contenedores. Algunas tareas pueden necesitar solo un contenedor, mientras que otras tareas pueden necesitar 2 o más contenedores potencialmente vinculados que se ejecutan simultáneamente. La definición de tareas le permite especificar qué imagen de Docker usar, qué puertos exponer, cuánta CPU y memoria asignar, cómo recopilar registros y definir variables de entorno.

Task

Una task o tarea es la instanciación de una task definition dentro de un clúster. Después de crear una definición de tarea para una aplicación dentro de Amazon ECS, se puede especificar la cantidad de tareas que se ejecutarán en el clúster.

Service

Se utiliza para garantizar que siempre tenga una cierta cantidad de Tasks se estén ejecutando en todo momento. Si el contenedor de una tarea se cierra debido a un error, o si la instancia EC2 subyacente falla y se reemplaza, el servicio ECS reemplazará la task fallida. Es por eso que creamos Clusters para que el servicio tenga muchos recursos en términos de puertos de CPU, Memoria y Red para usar. Para nosotros, realmente no importa en qué instancia se ejecutan las tareas, siempre y cuando se ejecuten. La configuración de service hace referencia a una Task Definition. Un Service es responsable de crear Tasks.

Tipos de lanzamiento

Las tareas y los servicios se pueden ejecutar en una infraestructura serverless administrada por AWS Fargate o si se desea más control sobre su infraestructura, se pueden ejecutar en un clúster de instancias de Amazon EC2:

ECS con EC2:

Como servicio administrado para la orquestación de contenedores, hay muchos aspectos de ECS que ayudan a simplificar la administración de contenedores, incluida la creación, configuración y mantenimiento de clústeres. Sin embargo, ECS no automatiza todo lo relacionado con la administración de clústeres. Todavía deja la capa de “compute” visible, lo que requiere que los usuarios aprovisionen, escalen, monitoreen, aseguren y administren las instancias EC2 subyacentes por sí mismos. Los proveedores de capacidad de ECS se pueden usar para administrar la infraestructura de tareas a través de la estrategia predeterminada, o los usuarios pueden definir la suya propia.

- Cuando los contenedores de ECS se implementan en instancias de EC2, depende del usuario determinar qué tipo de instancia usar y cuándo escalarlas.
- Cargas de trabajo que requieren un uso constante de memoria y núcleo de CPU
- Grandes cargas de trabajo que deben optimizarse por precio
- Sus aplicaciones necesitan acceder al almacenamiento persistente
- Debe administrar directamente su infraestructura

ECS con Fargate

Sin embargo, ejecutar ECS con Fargate elimina la necesidad de aprovisionar, escalar y administrar manualmente las instancias informáticas. Los usuarios crean un clúster, le agregan cargas de trabajo y especifican los requisitos de recursos (CPU y memoria), y cuando se implementan los contenedores ECS, Fargate iniciará, ejecutará y administrará servidores preconfigurados que cumplan con los requisitos del contenedor. Estos beneficios de ahorro de tiempo eliminan la carga operativa de administrar la computación, pero la compensación es características limitadas, menos control y posiblemente costos más altos.

- Grandes cargas de trabajo que deben optimizarse para reducir la sobrecarga
- Pequeñas cargas de trabajo que tienen ráfagas ocasionales
- Cargas de trabajo pequeñas
- Cargas de trabajo en lotes

Servicios de Cola de Mensajes

RabbitMQ

RabbitMQ es un software de cola de mensajes de código abierto que implementa el Protocolo avanzado de cola de mensajes (Advanced Message Queuing Protocol - AMQP). Es una opción popular para crear sistemas escalables de alto rendimiento porque permite que las aplicaciones se comuniquen e intercambien mensajes de manera confiable y eficiente.
El software de intermediario de mensajes como RabbitMQ actúa como intermediario entre las aplicaciones. Cuando una aplicación envía un mensaje, el intermediario lo almacena en una cola y luego lo reenvía al destinatario apropiado cuando está listo para ser procesado. Esto desacopla el emisor y el receptor, lo que les permite operar de forma independiente y asíncrona.
RabbitMQ está escrito en el lenguaje de programación Erlang y está diseñado para ser fácil de usar, confiable y escalable. Admite una amplia gama de lenguajes y protocolos, incluidos Java, .NET y Python, lo que lo convierte en una opción versátil para crear sistemas distribuidos.

Beneficios de utilizar RabbitMQ

Además de permitir la integración de diferentes aplicaciones a través de mensajes de forma asíncrona (desacoplamiento en tiempo) y desde diversas ubicaciones (desacoplamiento en espacio), RabbitMQ nos ofrece otros beneficios que lo han hecho muy popular dentro del mundo de los brokers de mensajería:

- Confiabilidad: RabbitMQ incorpora varias características que le permiten garantizar la entrega de los mensajes. Entre ellas, proporciona almacenamiento cuando no hay consumidores disponibles para recibir el mensaje, brinda la posibilidad de que el consumidor acepte la entrega del mensaje para asegurarse de que lo procesó correctamente y, en caso de que haya fallado su procesamiento, permite que el mensaje se pueda reencolar para ser consumido por una instancia diferente del consumidor o que sea procesado de nuevo por el mismo consumidor que inicialmente falló, cuando este se recupere.

RabbitMQ también garantiza el orden de entrega de los mensajes, es decir, estos se van consumiendo en el mismo orden en que han estado llegando a las colas.

- Creación de clusters: Si bien RabbitMQ proporciona gran rendimiento procesando miles de mensajes por segundo, en ocasiones debe ser capaz de procesar una mayor cantidad de mensajes sin impactar el rendimiento de las aplicaciones. Para esto RabbitMQ permite la creación de clústeres para escalar horizontalmente la solución, lo cual es transparente tanto para los productores como para los consumidores.
- Colas altamente disponibles: En RabbitMQ las colas pueden ser replicadas en diversos nodos de un cluster, proporcionando la seguridad de que en caso de una falla o indisposición de un nodo, el broker puede seguir recibiendo mensajes de los productores y entregandolos a los consumidores adecuados.
- Permite la escalabilidad de las aplicaciones: Cuando un consumidor se subscribe a una cola, y existen mensajes para este consumidor, RabbitMQ le va entregando mensajes para su procesamiento . Si la velocidad de producción de mensajes es mayor a la capacidad que el consumidor puede procesar dichos mensajes, se pueden crear nuevas instancias de ese consumidor para hacer frente al mayor flujo de mensajes.
  Cuando RabbitMQ identifica que existen varias instancias de un mismo consumidor suscritas a una misma cola, balancea la entrega de mensajes a cada una de las instancias. Esta característica no solo nos posibilita la distribución de carga en las aplicaciones consumidoras, también nos permite incrementar la disponibilidad de las mismas.
- Enrutamiento flexible: En RabbitMQ se pueden definir reglas de enrutamiento flexible, incluso que cumplan un determinado patrón, para enrutar los mensajes entre los exchanges y las colas, a través de los bindings.
- Soporte a múltiples protocolos: Aparte de soportar el protocolo AMQP, RabbitMQ también soporta STOMP, MQTT y HTTP a través de plugins.
- Mecanismos de autenticación: Incorpora mecanismos de autenticación y control de acceso a cada uno de los componentes del broker.
- Soporte de lenguajes: RabbitMQ soporta una gran cantidad de lenguajes de programación con los que es posible construir productores y consumidores de mensajes. Entre estos tenemos Java, Scala, PHP, Python, Ruby, entre otros.

AWS SQS

Amazon SQS es un sistema de cola de mensajes que le permite escribir aplicaciones distribuidas al exponer una canalización de mensajes que los workers pueden procesar en segundo plano. Si necesita una herramienta para facilitar las aplicaciones distribuidas, la arquitectura basada en eventos o el procesamiento de eventos, entonces SQS puede ser una opción sólida para agregar a su arsenal tecnológico.

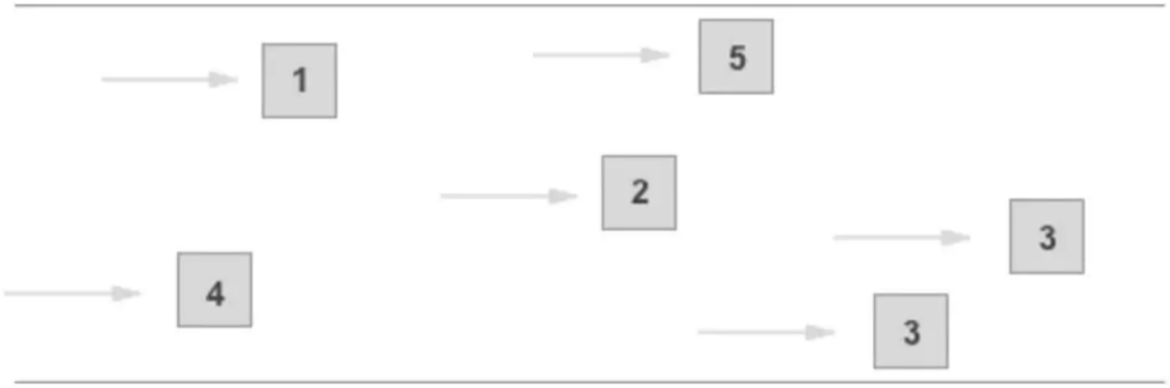
- En comparación con otras colas de mensajes populares, SQS no tiene costos iniciales y, en cambio, se paga por lo que se usa. Esto hace que experimentar con SQS sea un servicio atractivo de valor agregado para su aplicación.
- SQS es un servicio administrado, por lo que no hay que preocuparse de costos operacionales de correr un sistema de mensajería, incluyendo administración, monitoreo y seguridad.

- Es elástico y puede escalar a volúmenes muy altos, quedan atrás los días de tener que expandir las instancias para aceptar más carga.
- Alta disponibilidad avalada por AWS, lo que quita una preocupación sobre la aplicación.
- No se cobra por el almacenamiento usado, pero se tiene un límite de persistencia de 14 días.

Tipos de cola de mensajes

Standard

- At-Least-Once Delivery: Los mensajes se entregan al menos una vez
- Best-Effort Ordering: Ocasionalmente los mensajes pueden llegar en orden distinto al enviado
- Enviar datos entre aplicaciones cuando el rendimiento es importante
- Standard Queue posee un TPS (Transactions per second) casi ilimitado



FIFO

- First-In-First-Out Delivery: El orden de envío y recepción de los mensajes se respeta estrictamente.
- Exactly-Once Processing: Se garantiza que todos los mensajes van a llegar al menos una vez y los duplicados serán eliminados
- Enviar datos entre aplicaciones cuando el orden es importante
- Rendimiento limitado: Máximo 300 TPS

AWS Amazon MQ

Amazon MQ es un servicio de mensajería gestionado y proporcionado por Amazon Web Services. Amazon MQ facilita la implementación y la gestión de sistemas de mensajería basados en los protocolos de colas de mensajes más populares, como Apache ActiveMQ y RabbitMQ. Permite a los usuarios implementar sistemas de mensajería de alta disponibilidad y escalables en la nube sin tener que preocuparse por el mantenimiento y la administración de la infraestructura subyacente y es compatible con varias APIs y lenguajes de programación, lo que facilita la integración con aplicaciones existentes. Además, Amazon

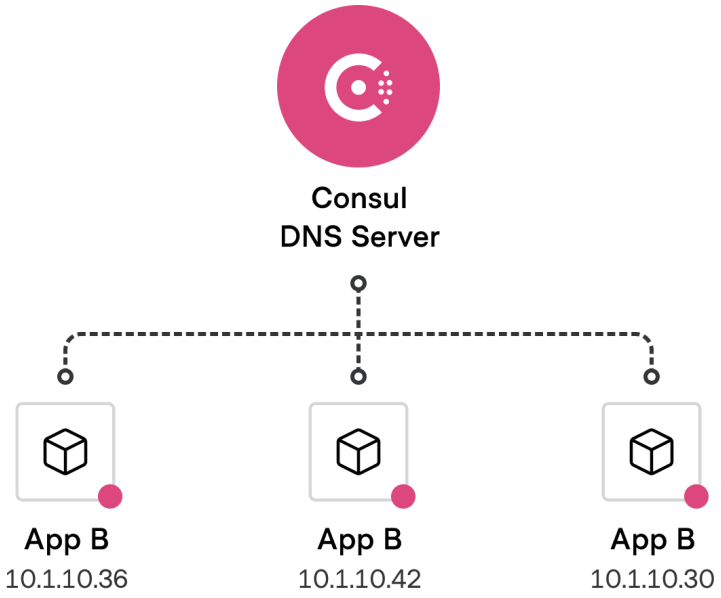
MQ ofrece características como la entrega de mensajes en orden, la tolerancia a fallos, la replicación de datos y la seguridad de los datos en tránsito y en reposo.

Service Discovery

Hashicorp Consul

HashiCorp Consul es una solución de redes de servicios que permite a los equipos administrar una conectividad de red segura entre servicios y entornos de múltiples nubes y tiempos de ejecución. Consul ofrece descubrimiento de servicios, autorización basada en identidad, gestión de tráfico L7 y cifrado de servicio a servicio.

Consul ofrece una solución de service mesh con todas las funciones que resuelve los desafíos de redes y seguridad de operar microservicios e infraestructura en la nube (nube híbrida y multinube).



AWS Cloud Map

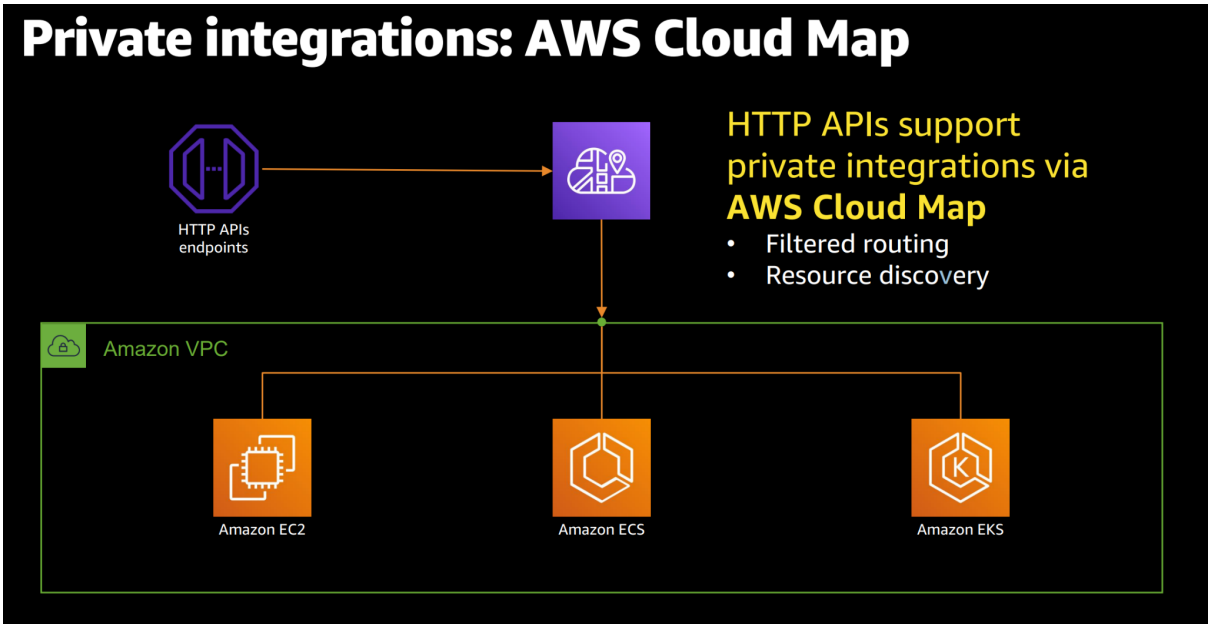
Amazon Cloud Map es un servicio en la nube que facilita el descubrimiento y la conexión a recursos alojados en la nube. Proporciona una forma unificada de descubrir y acceder a servicios en múltiples cuentas y regiones de AWS, para poder crear aplicaciones resistentes, escalables y de alta disponibilidad.

Con Cloud Map, puede definir nombres personalizados para los recursos, como bases de datos, servidores o microservicios. Luego puede usar estos nombres para descubrir y conectarse a los recursos de sus aplicaciones, sin necesidad de obtener y utilizar sus direcciones IP o nombres de host.

Cloud Map admite varios mecanismos de descubrimiento, incluidos DNS, HTTP y AWS PrivateLink, y proporciona API y SDK para diferentes lenguajes de programación. También se integra con otros servicios de AWS, como Amazon ECS, Amazon EKS y AWS App Mesh,

para brindar una manera perfecta de descubrir y conectarse a los recursos dentro de sus aplicaciones.

Cloud Map está completamente administrado, por lo que no tiene que preocuparse por la infraestructura o el mantenimiento. Está diseñado para ser altamente disponible y escalable, por lo que puede confiar en él para descubrir y conectarse a sus recursos en todo momento.



API Gateway

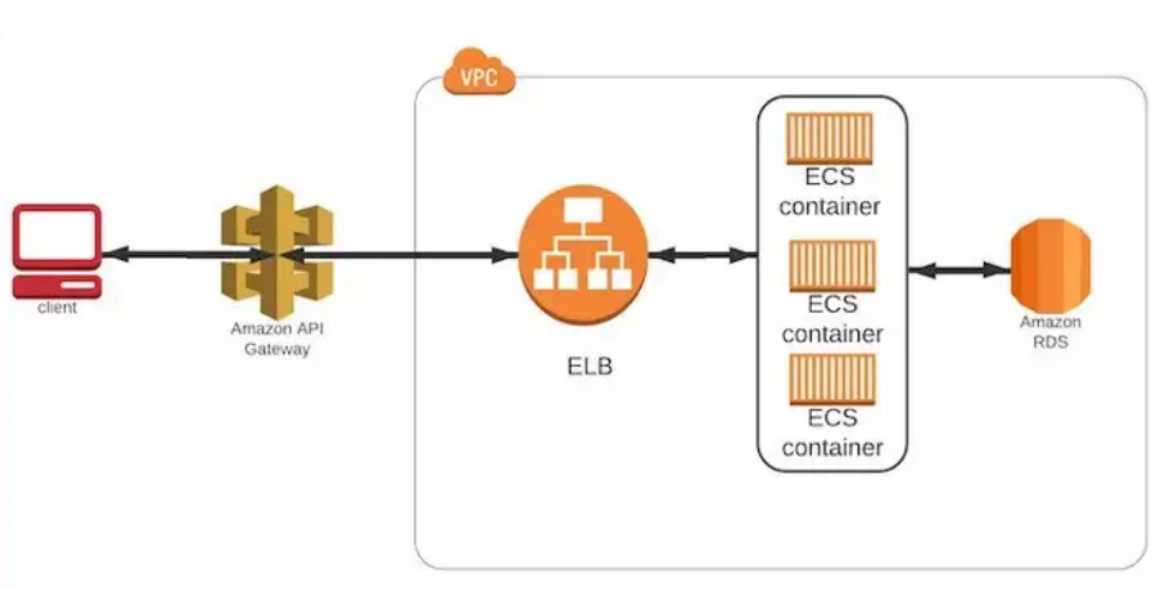
AWS API Gateway

Amazon API Gateway es un servicio completamente administrado que facilita a los desarrolladores la creación, la publicación, el mantenimiento, el monitoreo y la protección de API a cualquier escala. Las API actúan como la "puerta de entrada" para que las aplicaciones accedan a los datos, la lógica empresarial o la funcionalidad de sus servicios de backend. Con API Gateway, puede crear API RESTful y API WebSocket que permiten aplicaciones de comunicación bidireccional en tiempo real:

Característica	API RESTful	API WebSocket
Protocolo de comunicación	HTTP	WebSocket
Transferencia de datos	Unidireccional (cliente a servidor)	Bidireccional (cliente y servidor)
Comunicación en tiempo real	No	Sí
Persistencia de conexión	No	Sí

Tipo de datos	Transferencia de datos en formato JSON o XML	Transferencia de datos en formato de flujo de mensajes
Complejidad de implementación	Fácil	Moderada
Soporte de lenguajes	Compatible con la mayoría de los lenguajes y marcos de desarrollo	Requiere soporte específico para WebSocket en el lado del servidor
Escalabilidad	Escalabilidad limitada	Altamente escalable

En general, un API RESTful es más adecuado para aplicaciones web simples que requieren una comunicación unidireccional y la transferencia de datos en formatos estructurados. Por otro lado, un API WebSocket es más adecuado para aplicaciones en las que se requiere una comunicación en tiempo real y bidireccional, como aplicaciones de chat, juegos en línea, entre otras.



Netflix Zuul

Zuul es un API Gateway de código abierto desarrollado por Netflix. Es utilizado por el servicio de transmisión de Netflix para proporcionar enrutamiento dinámico, monitoreo, resistencia y seguridad a sus aplicaciones.
Zuul se basa en la pila de OSS (Open Source Software) de Netflix, que incluye componentes como Hystrix para la tolerancia a fallas y Ribbon para el balanceo de carga del lado del cliente. También se integra con otros componentes de OSS de Netflix, como Eureka para Service Discovery y Archaius para la Configuration Management.
En general, Netflix Zuul es un API gateway potente y ampliamente utilizado que proporciona muchas funciones útiles para crear arquitecturas de microservicios escalables y resistentes, por ejemplo, puede actuar como la API Gateway para una aplicación basada en microservicios desarrollada en Java Spring Boot o NodeJS, proporcionando enrutamiento dinámico, monitoreo y seguridad a la aplicación.

Bases de datos

Antes de abarcar las diferentes alternativas de persistencia en bases de datos, vale la pena revisar una abstracción más general de estas, SQL vs NoSQL:

Tipos de bases de datos

SQL

Las bases de datos SQL (acrónimo de Structured Query Language), también llamadas bases de datos relacionales. Están constituidas por un conjunto de tablas en las que los datos están clasificados por categorías.
Cada columna de estas tablas corresponde y comprende una cierta cantidad de datos de esta categoría. Estas tablas respetan generalmente el mismo esquema fijo, es decir que la forma de la tabla (cantidad de columnas, títulos, tipos de datos y eventualmente otras características). Por ejemplo:

ID	Nombre	Apellido	Ciudad
213	Juan	Sanz	Buenos Aires
383	Andrés	Pérez	Santa Fe
701	Lionel	Messi	Rosario

Como se puede ver, cada columna corresponde a una categoría específica y las columnas tienen un tipo bien definido (nombre, texto, etc).
A esta tabla, se le asocia otra tabla que contiene otras informaciones.
Por ejemplo:

Ciudad	Código Postal
Rosario	2000
Santa Fe	3000
Buenos Aires	1000

Podemos observar que las dos tablas están relacionadas por la columna “Ciudad”, es lo que llamamos una clave extranjera, clave foránea o Foreign Key. Esta clave permite unir de manera coherente los datos.
Casos de uso para SQL:

- Necesidad una base de datos estructurada y segmentada (la esencia de las bases de datos relacionales)
- Tipo y validez de datos muy importante
- Necesidad recurrente de escritura y modificaciones de datos sobre elementos específicos (SQL permite modificar fácilmente lineas especificas)
- Necesidad de búsquedas complejas

Ejemplos de motores de bases de datos SQL son: MySQL, SQL Server, PostgreSQL.

NoSQL

Las bases de datos NoSQL son, como su nombre lo indica, son no relacionales. Estas bases de datos no necesitan un esquema fijo y son fácilmente modulares. Existen diferentes tipos que permiten adaptarse a múltiples formatos de datos, por ejemplo documentos, gráficos o incluso formatos de claves. El objetivo es recuperar los datos de un mismo lugar sin necesidad de pasar por las relaciones entre tablas.

Tomemos el ejemplo anterior, con un formato de documentos, frecuentemente es un objeto JSON que es utilizado (un tipo de documento). En ese caso, cada atributo de las columnas SQL es un campo y los detalles de registro de una persona son los valores de datos asociados a cada campo:

ID Usuario: “701”, Nombre: “Lionel” , Apellido: ”Messi”, Ciudad: “Rosario”, Codigo postal: “2000”

Cada tipo de bases de datos NoSQL está concebida en función de una situación específica y las especificaciones técnicas respectivamente justifican una estructura en particular.

Las bases de datos NoSQL se desarrollaron a finales de los años 2000 y rápidamente se democratizaron gracias a su capacidad para manejar grandes bases de datos distribuidas de Big Data.

Casos de uso para NoSQL:

- Bases de datos sin esquemas específicos (estructura no fija por ejemplo)
- Necesidad de múltiples búsquedas de lectura, todos los datos necesarios se pueden recuperar efectivamente de una vez sin combinación en particular.
- Grandes conjuntos de datos (Big Data)
- Datos distribuidos (varias fuentes)

Ejemplos de bases de datos no relacionales son: MongoDB, Cassandra, AWS DynamoDB, AWS DocumentDB.

Las bases de datos NoSQL se adaptan perfectamente a muchas aplicaciones modernas, como dispositivos móviles, web y juegos, que requieren bases de datos flexibles, escalables, de alto rendimiento y altamente funcionales para proporcionar excelentes experiencias de usuario.

- Flexibilidad: las bases de datos NoSQL generalmente ofrecen esquemas flexibles que permiten un desarrollo más rápido y más iterativo. El modelo de datos flexible hace que las bases de datos NoSQL sean ideales para datos semiestructurados y no estructurados.
- Escalabilidad: las bases de datos NoSQL generalmente están diseñadas para escalar usando clústeres distribuidos de hardware en lugar de escalar añadiendo servidores caros y sólidos. Algunos proveedores de la nube manejan estas operaciones en segundo plano, como un servicio completamente administrado.
- Alto rendimiento: la base de datos NoSQL está optimizada para modelos de datos específicos y patrones de acceso que permiten un mayor rendimiento que el intento de lograr una funcionalidad similar con bases de datos relacionales.

- Altamente funcional: las bases de datos NoSQL proporcionan API altamente funcionales y tipos de datos que están diseñados específicamente para cada uno de sus respectivos modelos de datos.

MySQL

MySQL es un popular sistema de gestión de bases de datos relacionales de código abierto. Es una poderosa herramienta para almacenar y administrar datos, y es utilizado en una amplia variedad de aplicaciones. Algunas de las ventajas de MySQL incluyen:

- Es fácil de usar y aprender.
- Es rápido y eficiente.
- Es altamente escalable, por lo que puede manejar grandes cantidades de datos.
- Tiene una gran cantidad de características y admite una amplia gama de tipos de datos.
- Es seguro y confiable, con soporte sólido para transacciones e integridad de datos.
- Es ampliamente compatible y tiene una gran comunidad de usuarios y desarrolladores.

En general, MySQL es una herramienta poderosa y versátil que se adapta bien a una amplia gama de aplicaciones. Es una buena opción para cualquier persona que necesite administrar datos y busque una solución confiable y eficiente.

High Availability en MySQL

La alta disponibilidad (HA) se refiere a la capacidad de un sistema para operar continuamente sin fallas durante un período prolongado de tiempo. El marco HA funciona para verificar que un sistema cumpla con los requisitos del usuario sin ninguna interrupción en el procesamiento de datos de extremo a extremo, lo que garantiza la tolerancia a fallas y una alta disponibilidad de datos.
En otros términos, la alta disponibilidad (HA) es una técnica que elimina un punto único o una falla accidental de los sistemas o aplicaciones para garantizar operaciones continuas o tiempo de actividad durante un período prolongado. El porcentaje de tiempo que los servicios de un sistema informático están disponibles durante un período de tiempo determinado se denomina disponibilidad o alta disponibilidad. Se expresa comúnmente como una sucesión de 9's.
El porcentaje de alta disponibilidad variará según el sistema o aplicación. Por ejemplo, cuatro 9, es decir, 99,99 % de disponibilidad, es un estándar de la industria para aplicaciones de misión crítica, como sus sistemas de comercio electrónico y servicios financieros. No puede esperar más de 52,60 minutos de interrupción por año o 8,64 segundos de tiempo de inactividad por día si su sistema está disponible en un 99,99 por ciento disponible.

De manera similar, para aplicaciones y sistemas no críticos, la alta disponibilidad puede ser de dos 9, es decir, 99 %, lo que equivale a 8,77 horas de tiempo de inactividad al año o 1,44 minutos de indisponibilidad al día. De acuerdo con los estándares de la industria, si el sistema o la aplicación no pueden permitirse más de unos pocos minutos de interrupción al año, es evidente que el servicio requiere un 99,999 % de alta disponibilidad.

La siguiente tabla muestra la disponibilidad y el tiempo de inactividad correspondiente a lo largo del año.

Disponibilidad	Tiempo de inactividad al año
90%	36,53 días
99%	3,65 días
99,9%	8,77 horas
99,99%	52,60 minutos
99,999%	5,26 minutos
99,9999%	31,56 segundos

Características del marco de alta disponibilidad

La capacidad de recuperarse instantáneamente de fallas que pueden ocurrir en cualquier componente de un sistema es la base de ser un sistema de alta disponibilidad. Para garantizar la capacidad de recuperación y la alta disponibilidad, cuatro componentes críticos en cualquier sistema HA deben trabajar juntos en un método automatizado.

Infraestructura y redundancia de datos

Para garantizar que un servicio tenga una alta disponibilidad, debemos asegurarnos de que la infraestructura que lo aloja tenga redundancia, así como una copia redundante actualizada de los datos que consume u ofrece el servicio. Esto sirve como un servicio de respaldo, listo para tomar el control si las fallas interrumpen la base de datos primaria o principal.

Mecanismo de detección y corrección de fallas

Es fundamental detectar cualquier falla en cualquier aspecto del sistema central que pueda afectar su disponibilidad lo antes posible. Esto permitirá que el marco realice pasos correctivos en el sistema principal o realice una conmutación por error de los servicios a un sistema en espera.

Mecanismo de conmutación por error

Este componente se encarga de transferir los servicios a su infraestructura de respaldo. Los usuarios deben asegurarse de que si hay numerosos sistemas redundantes disponibles, este componente del mecanismo de conmutación por error debe elegir el mejor y promocionarlo como el servicio principal.

Mecanismo de redirección de aplicaciones/usuarios

Cuando los sistemas en espera asumen el papel principal, la función de alta disponibilidad garantiza que todas las conexiones de aplicaciones y usuarios se transfieran o redirijan a la nueva base de datos principal. En el contexto de las bases de datos MySQL, se descubrirá más en las próximas secciones de alta disponibilidad de MySQL.

Funcionamiento continuo

Cuando un sistema o aplicación se ejecuta de forma continua, incluso una sola o breve interrupción puede causar dificultades considerablemente mayores a medida que los

sistemas conectados y sincronizados fallan en cascada. En tales escenarios, vale la pena considerar invertir en alta disponibilidad, ya que los costos de reparación del sistema o de la aplicación son significativamente más altos.

MongoDB

MongoDB es una popular base de datos NoSQL de código abierto. Algunas de las ventajas de usar MongoDB incluyen:

- Es una base de datos orientada a documentos, lo que significa que almacena datos en documentos similares a JSON a los que se puede acceder y actualizar fácilmente.
- Es altamente escalable, por lo que puede manejar grandes cantidades de datos y admitir una gran cantidad de usuarios.
- Es flexible y le permite cambiar fácilmente la estructura de sus datos sin necesidad de migrarlos o transformarlos.
- Tiene un lenguaje de consulta enriquecido que le permite encontrar y recuperar fácilmente los datos que necesita.
- Tiene un fuerte soporte para fragmentación, lo que le permite distribuir sus datos a través de múltiples servidores para mejorar el rendimiento y la disponibilidad.

Sin embargo, también hay algunas desventajas al usar MongoDB:

- No admite transacciones, por lo que puede no ser adecuado para aplicaciones que requieren garantías de integridad de datos complejas.
- Puede ser difícil migrar datos de otras bases de datos a MongoDB, ya que los formatos de datos son diferentes.
- Puede ser difícil ajustar y optimizar el rendimiento, ya que la base de datos no proporciona tanto control sobre la colocación e indexación de datos como otras bases de datos.

En general, MongoDB tiene algunas ventajas significativas, pero puede que no sea la mejor opción para todas las aplicaciones. Es importante considerar cuidadosamente sus necesidades y requisitos específicos antes de decidir si usar MongoDB.

Bases de datos relacionales en AWS

Aurora[2]

Aurora es un servicio de base de datos de AWS que es compatible con MySQL y PostgreSQL pero utiliza un motor de base de datos innovador en segundo plano. Las aplicaciones que actualmente usan MySQL/PostgreSQL se pueden migrar a Aurora con cambios menores o sin cambios.
Una instancia de base de datos de Aurora puede almacenar entre 10 GB y 128 TB, con datos divididos en bloques de 10 GB y distribuidos en diferentes discos.
Ofrece escalabilidad sencilla: el almacenamiento se escala automáticamente cuando la base de datos crece y, para admitir más solicitudes de lectura, puede crear hasta 15 réplicas de lectura.

[2] https://aws.amazon.com/es/rds/aurora/features/

Configuración serverless: Aurora Serverless es una configuración de escalado automático bajo demanda para Aurora, en la que la base de datos activa, cierra, amplía o reduce su capacidad automáticamente en función de las necesidades de la aplicación.
Escalado automático de almacenamiento: Aurora aumenta automáticamente el tamaño del volumen de la base de datos a medida que aumentan las necesidades de almacenamiento. El volumen aumenta en bloques de 10 GB, hasta un máximo de 128 TB.
Réplicas de lectura de baja latencia: Puede aumentar el rendimiento de lectura para admitir solicitudes de aplicaciones de volumen alto mediante la creación de hasta 15 réplicas de bases de datos de Amazon Aurora. Estas réplicas comparten el mismo almacenamiento subyacente que la instancia de origen, lo que reduce los costos y evita la necesidad de escribir en los nodos de réplica.

RDS

Amazon Relational Database Service es un servicio de base de datos administrado que admite varios motores de base de datos populares, incluidos PostgreSQL, MySQL, SQL Server, MariaDB y Oracle.
RDS ayuda a realizar una amplia gama de tareas de administración de bases de datos, incluidas la copia de seguridad y recuperación de datos, la aplicación de parches y las migraciones.
Realiza automáticamente una copia de seguridad de las instancias de la base de datos, captura instantáneas diarias de los datos, conserva los registros de transacciones y permite la recuperación de un punto en el tiempo.
Además, el servicio parchea automáticamente el software del motor de la base de datos.
Para mejorar la confiabilidad y disponibilidad de las cargas de trabajo, permite la replicación de bases de datos con conmutación por error automatizada en varias zonas de disponibilidad.

Bases de datos no relacionales en AWS

DynamoDB

DynamoDB es una base de datos NoSQL de clave-valor completamente administrada que está diseñada para ejecutar aplicaciones de alto rendimiento a cualquier escala. DynamoDB ofrece seguridad integrada, copias de seguridad continuas, replicación automatizada en varias regiones, almacenamiento de caché en memoria y herramientas de importación y exportación de datos.
DynamoDB es una base de datos NoSQL que admite modelos de datos de documentos y clave valor. Los desarrolladores pueden utilizar DynamoDB para crear aplicaciones modernas y sin servidores que pueden comenzar a pequeña escala y alcanzar una escala global para admitir petabytes de datos y decenas de millones de solicitudes de lectura y escritura por segundo. DynamoDB se ha diseñado para ejecutar aplicaciones de alto rendimiento a escala de Internet que sobrecargarían las bases de datos relacionales tradicionales.
Está optimizada para tener un buen rendimiento y escalabilidad horizontal añadiendo más nodos al clúster. Si el tamaño de los datos crece, no se va a penalizar en el rendimiento de la base de datos de una manera muy significativa.
Además, tiene alta disponibilidad (99.999% de uptime SLA, esto es menos de 5 minutos al año de caídas) y durabilidad de los datos. Para ello, se almacena varias copias de los datos

en diferentes nodos. Si uno de estos nodos falla, otro puede tomar su lugar como nodo primario.
Tradicionalmente, cuando teníamos una base de datos debíamos gestionar todo su hardware, seguridad, actualizaciones y tener en cuenta su mantenimiento. Con DynamoDB todo esto no es necesario ya que es un servicio administrado, y aunque hay servidores y sistemas de almacenamiento por detrás, es totalmente transparente y no es necesario preocuparse por configuraciones complejas o seguridad.

DocumentDB

Amazon DocumentDB es un servicio de bases de datos documentales totalmente administrado y escalable que se ejecuta en la nube de Amazon Web Services (AWS). Algunas de las ventajas de Amazon DocumentDB son:

- Compatibilidad con MongoDB: Amazon DocumentDB es compatible con la API y las herramientas de MongoDB, lo que facilita la migración de aplicaciones existentes a la nube de AWS.
- Escalabilidad: Amazon DocumentDB se escala horizontalmente de forma automática en cuestión de minutos, sin interrupciones ni tiempos de inactividad.
- Alta disponibilidad: Amazon DocumentDB ofrece una alta disponibilidad gracias a su replicación automática y distribución en múltiples zonas de disponibilidad de AWS.
- Seguridad: Amazon DocumentDB proporciona medidas de seguridad integradas, como la encriptación en reposo y en tránsito, y la autenticación y autorización de usuario.
- Rendimiento: Amazon DocumentDB proporciona un rendimiento de baja latencia y alta velocidad de lectura y escritura, lo que permite a las aplicaciones realizar operaciones de bases de datos a alta velocidad.
- Administración simplificada: Amazon DocumentDB es un servicio completamente administrado que se encarga de tareas de mantenimiento, parches y copias de seguridad, lo que permite a los desarrolladores centrarse en el desarrollo de aplicaciones.
- Integración con AWS: Amazon DocumentDB se integra fácilmente con otros servicios de AWS, como Amazon EC2, AWS Lambda, AWS CloudFormation y Amazon CloudWatch, lo que permite a los desarrolladores construir aplicaciones escalables y flexibles.

Análisis y visualización de métricas

Elastic Stack

El Elastic Stack, anteriormente conocido como Stack ELK, es una colección de herramientas de software de código abierto para la ingesta, el enriquecimiento, el almacenamiento, el análisis y la visualización de datos. El Elastic Stack está compuesto por los siguientes componentes:

- Elasticsearch: un motor de búsqueda y análisis distribuido para almacenar y consultar grandes volúmenes de datos.
- Logstash: un pipeline de datos para recopilar, analizar y transformar registros y otros

datos.
- Kibana: una herramienta de visualización y análisis para explorar y analizar datos almacenados en Elasticsearch.
- Beats: una familia de transportadores de datos livianos para enviar datos desde varias fuentes (por ejemplo, archivos de registro, métricas del sistema, paquetes de red) a Elasticsearch o Logstash.

El Elastic Stack se usa ampliamente para aplicaciones como análisis de registros, análisis en tiempo real y monitoreo. Es altamente escalable y puede manejar grandes cantidades de datos, lo que lo convierte en una opción popular para organizaciones de todos los tamaños.

Elastic Search

Elasticsearch es un popular motor de análisis y búsqueda que forma parte de Elastic Stack. Es un motor de análisis y búsqueda distribuido, escalable y de alta disponibilidad que está diseñado para almacenar, buscar y analizar grandes volúmenes de datos. Algunas de las ventajas de Elasticsearch incluyen:
- Es altamente escalable y puede manejar grandes volúmenes de datos.
- Tiene un lenguaje de consulta rico y poderoso que le permite buscar y filtrar fácilmente los datos.
- Tiene un fuerte soporte para la búsqueda de texto completo y otras funciones de búsqueda avanzada.
- Está construido sobre el motor de búsqueda Apache Lucene, que es conocido por su velocidad y eficiencia.
- Tiene alta disponibilidad, con soporte para fragmentación y replicación para garantizar que sus datos estén siempre accesibles.

Sin embargo, también hay algunas desventajas al usar Elasticsearch:
- Puede ser difícil de aprender y usar, especialmente para los principiantes.
- Puede ser difícil ajustar y optimizar el rendimiento, ya que el motor no brinda tanto control sobre la ubicación e indexación de datos como otras bases de datos.
- Puede ser un desafío migrar datos de otras bases de datos a Elasticsearch, ya que los formatos de datos son diferentes.

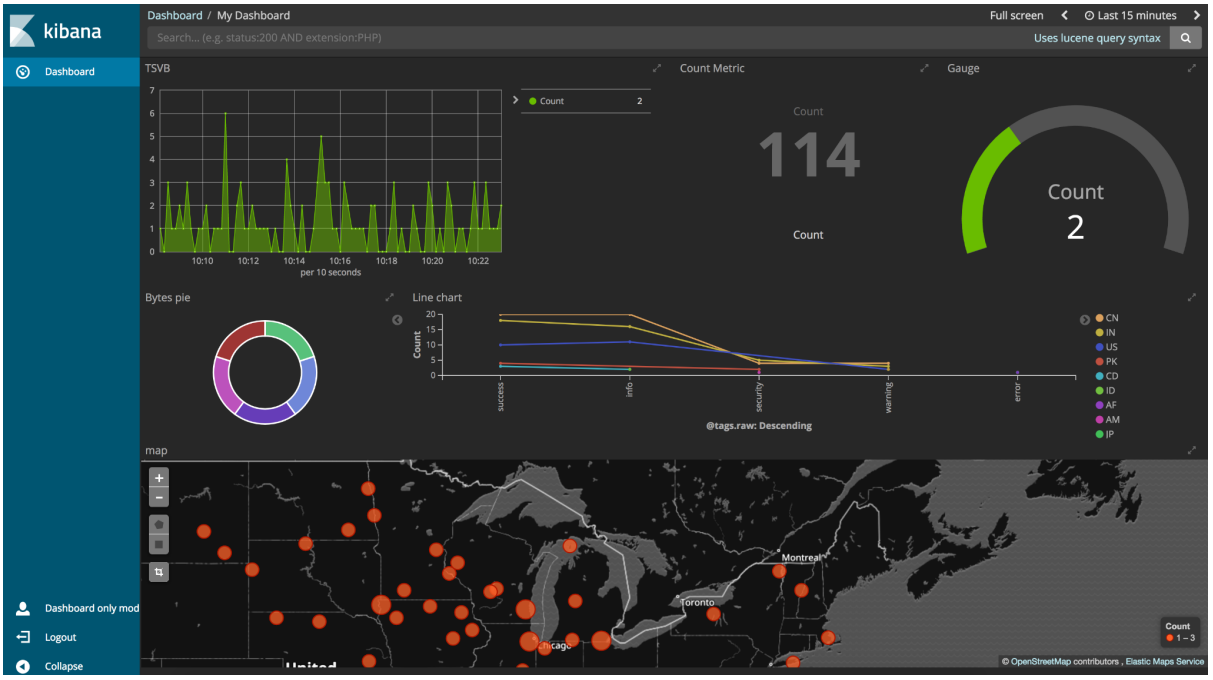
En general, Elasticsearch tiene muchas características poderosas y es una buena opción

para muchas aplicaciones. Sin embargo, puede que no sea la mejor opción para cada caso de uso, y es importante considerar cuidadosamente sus necesidades y requisitos específicos antes de decidir si usar Elasticsearch.

Kibana

Kibana es una popular herramienta de análisis y visualización de datos que forma parte de Elastic Stack. Se utiliza para explorar, analizar y visualizar datos almacenados en Elasticsearch. Kibana proporciona una interfaz web fácil de usar que permite crear y compartir visualizaciones y dashboards enriquecidos basados en sus datos. También proporciona un amplio conjunto de herramientas para filtrar los datos para descubrir perspectivas y tendencias.

Kibana se usa ampliamente para aplicaciones como análisis de registros, seguridad de redes y Business Intelligence (BI). Es altamente personalizable y puede utilizarse para crear una amplia gama de visualizaciones, incluidos gráficos de líneas, gráficos de barras, gráficos circulares y mapas de calor. Kibana es una herramienta poderosa que puede ayudarlo a obtener información de sus datos y tomar decisiones basadas en datos.



Ejemplo de dashboard de Kibana

Logstash

Logstash es un pipeline de procesamiento de datos popular que forma parte de Elastic Stack. Se utiliza para recopilar, analizar y transformar registros y otros datos de una variedad de fuentes y luego enviarlos a Elasticsearch u otros destinos para su almacenamiento y análisis.

Logstash es una poderosa herramienta que permite ingerir y procesar fácilmente grandes volúmenes de datos de una amplia gama de fuentes. Tiene un amplio conjunto de

complementos de entrada, filtrado y salida que le permiten configurar y personalizar fácilmente el pipeline de datos. Por ejemplo, puede usar Logstash para recopilar registros de diferentes sistemas, analizar los registros para extraer campos específicos, enriquecer los datos con información adicional y luego enviarlos a Elasticsearch para su almacenamiento y análisis.

En general, Logstash es una herramienta poderosa y versátil que puede ayudar a recopilar, transformar y analizar sus datos. Es ampliamente utilizado para aplicaciones como análisis de registros, análisis de seguridad y Business Intelligence.

Beats

Filebeat

Filebeat está pensado principalmente para leer ficheros de logs, aunque soporta algunos otros inputs como TCP, etc. El caso de uso más común es la recogida de logs en equipos remotos para su posterior centralización en Logstash o Elasticsearch.
Además de ser capaz de gestionar el último evento enviado y continuar posteriormente desde ese punto en caso de un corte en la comunicación, dispone de un control de carga para Logstash que le permite reducir la ratio de envío en casos de saturación en Logstash.
A lo largo del tiempo, Elastic ha ido ampliando la funcionalidad del agente, dotando a éste de una multitud de módulos especialmente pensados para facilitar el parseo de logs de soluciones ampliamente reconocidas (Apache, MySQL, Nginx, etc). De este modo no sólo se simplifica enormemente el proceso de identificación de path de logs y su posterior parseo, sino que también se facilita la ingesta optimizada del dato en Elasticsearch mediante templates y pipelines de ingesta predefinidos, y la creación de visualizaciones para posteriormente poder explotar la información desde dashboards en Kibana.

Metricbeat

Si lo que queremos es recoger métricas de servicios o sistemas (memoria, cpu, i/o disco, etc), Metricbeat será una solución perfecta para integrar el dato en nuestro clúster de Elasticsearch.
Como con el resto de beats, se trata de un agente ligero, el cual no repercutirá negativamente en el rendimiento del equipo donde se ejecuta.
Tal cómo indicamos anteriormente en el caso de Filebeat, Metricbeat también dispone de multitud de módulos preestablecidos (MongoDB, Apache, PostgreSQL, etc) para facilitar la recogida, ingesta y explotación de nuestras métricas en Elasticsearch mediante visualizaciones predefinidas.
Adicionalmente a las visualizaciones que podamos crear para abordar nuestras propias necesidades, Kibana dispone del módulo Metrics mediante el cual podremos visualizar de forma centralizada la información recogida por Metricbeat.

Packetbeat

Con Packetbeat podremos analizar paquetes de red en tiempo real, pudiendo entender así qué sucede en nuestra red. Nos permitirá recoger e ingestar en nuestro clúster de Elasticsearch datos relativos a latencias, tiempos de respuesta, errores, tendencias, etc sin interferir en nuestra infraestructura.

Packetbeat es compatible con muchos protocolos de capa de aplicación, desde bases de datos hasta protocolos de bajo nivel. En esta url podremos consultar toda la información relativa al beat y la lista de protocolos soportados, así como disponer de toda la información completa aquí.

Winlogbeat

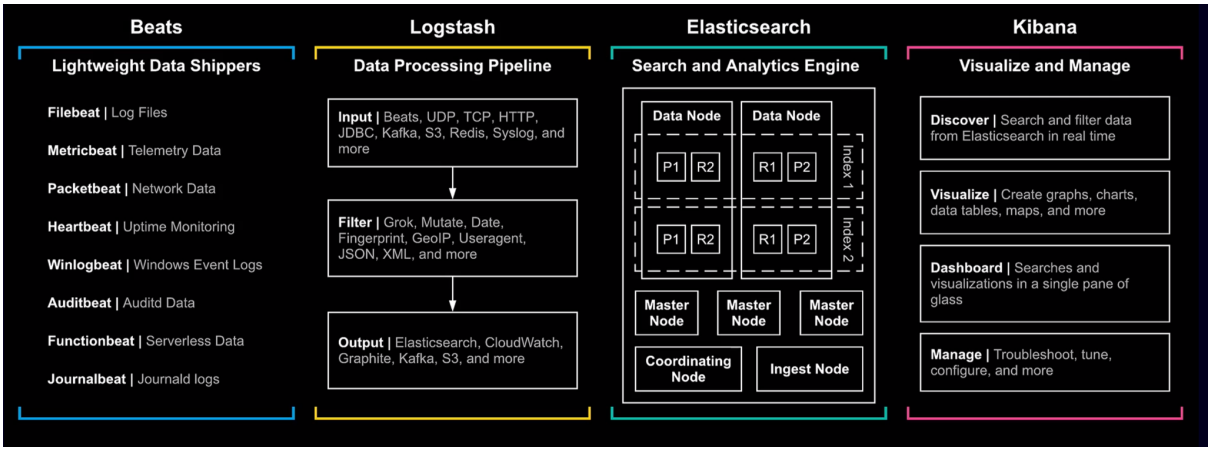
Para poder analizar los registros de eventos de log de nuestro parque de Windows, Elastic pone a nuestra disposición Winlogbeat. Con este beat podremos tanto recoger y enviar a Logstash todos nuestros eventos de Windows para realizar operaciones complejas sobre los datos, como insertarlos directamente en nuestro clúster de Elasticsearch para posteriormente analizar y explotar los datos desde dashboards personalizados.

Auditbeat

Auditbeat nos permite recoger datos de auditoría, monitoreando procesos y la actividad del usuario en tiempo real. En Linux se integra con el framework de audit, con la funcionalidad que esto representa.
Con Auditbeat también podremos monitorizar la integridad de ficheros, reportando información completa (metadata, hashes, …) sobre los cambios detectados en tiempo real.

Heartbeat

Con Heartbeat podremos monitorizar el tiempo de actividad y de respuesta de nuestros endpoints, mediante la configuración centralizada de estos.
De esta manera, si lo que necesitamos es supervisar el ping de hosts o servicios, ya sea mediante ICMP, HTTP o TCP, Heartbeat nos facilitará la recogida de datos y posterior ingesta en Elasticsearch para poder así analizar y visualizar el dato.
Kibana dispone del módulo Uptime, mediante el cual podremos visualizar toda la información recogida por Heartbeat, adicionalmente a las visualizaciones que podamos crear para abordar nuestras propias necesidades.



AWS OpenSearch Service

Amazon OpenSearch es un servicio de búsqueda y análisis gestionado que ofrece la misma funcionalidad que Elasticsearch, un motor de búsqueda y análisis de código abierto. OpenSearch se bifurcó de Elasticsearch y está diseñado para proporcionar a los usuarios

una experiencia familiar al mismo tiempo que ofrece una mayor estabilidad, escalabilidad y seguridad.

Si bien OpenSearch y Elasticsearch ofrecen capacidades de búsqueda y análisis similares, hay algunas diferencias clave entre ellos. Aquí hay algunas de las diferencias importantes:

- Servicio gestionado: OpenSearch es un servicio completamente gestionado proporcionado por Amazon Web Services (AWS), lo que significa que AWS maneja la mayoría de las tareas operativas, incluida la instalación, configuración, mantenimiento y seguridad. Elasticsearch, por otro lado, requiere que los usuarios administren sus propias implementaciones e infraestructura.
- Licencia: OpenSearch tiene licencia Apache License 2.0, lo que permite a los usuarios usar, modificar y distribuir el software de forma gratuita. Elasticsearch, sin embargo, tiene licencia Elastic License, que limita el uso del software en ciertos escenarios.
- Integración: OpenSearch está diseñado para integrarse con los servicios de AWS, como AWS Identity and Access Management (IAM), AWS Key Management Service (KMS) y Amazon CloudWatch. Elasticsearch se puede integrar con los servicios de AWS, pero requiere configuración adicional.
- Soporte: OpenSearch es compatible con AWS, lo que significa que los usuarios pueden recibir soporte técnico de AWS. Elasticsearch, por otro lado, es compatible con Elastic, que ofrece opciones de soporte gratuitas y pagas.

AWS OpenSearch Dashboards

Opensearch dashboards es el fork open source de Kibana. A continuación se detallan algunas de las diferencias entre AWS OpenSearch Dashboards y Kibana:

- Disponibilidad: OpenSearch Dashboards está disponible como un servicio gestionado en AWS, mientras que Kibana es una herramienta de código abierto que se debe instalar y ejecutar.
- Funcionalidades: En cuanto a funcionalidades, ambas herramientas ofrecen una amplia variedad de opciones para la visualización de datos y análisis de datos, incluyendo gráficos, tablas, mapas y paneles personalizados. Sin embargo, OpenSearch Dashboards está diseñado específicamente para trabajar con OpenSearch y se integra de manera más estrecha con otras herramientas de AWS, mientras que Kibana se integra de manera más estrecha con Elasticsearch y sus herramientas asociadas.
- Gestión: OpenSearch Dashboards es un servicio completamente gestionado, lo que significa que AWS se encarga de la mayoría de las tareas operativas, como la instalación, configuración, escalado y mantenimiento. Kibana, por otro lado, requiere que los usuarios administren sus propias implementaciones e infraestructura.
- Costos: OpenSearch Dashboards se cobra en función de la cantidad de datos procesados y almacenados, mientras que Kibana es gratuito y de código abierto. Sin embargo, Kibana requiere más experiencia técnica para configurar y mantener.

In-Memory Cache

Redis

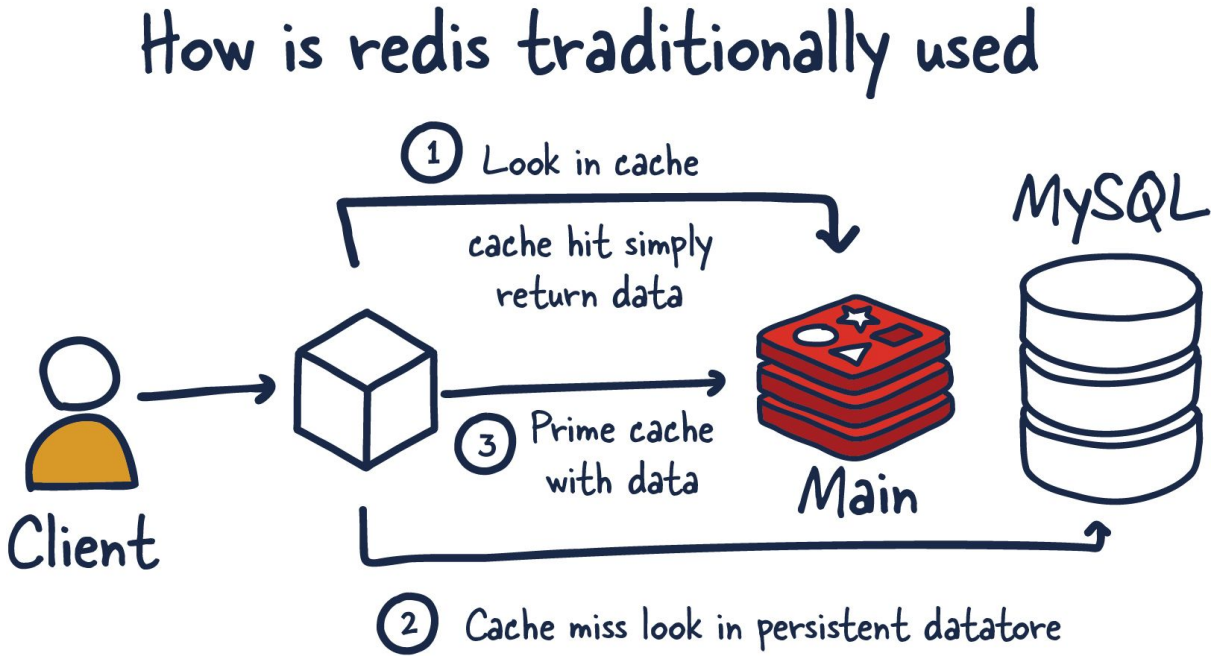
Redis (“**RE**mote **DI**ctionary **S**ervice”) es un almacén de estructura de datos en memoria de código abierto. Se utiliza como base de datos, caché e intermediario de mensajes, y admite una amplia gama de estructuras de datos, como cadenas, hashes, listas, conjuntos y conjuntos ordenados.

Redis es conocido por su alto rendimiento, facilidad de uso y flexibilidad. A menudo se usa como caché para acelerar el rendimiento de las aplicaciones web mediante el almacenamiento temporal de datos en la memoria, y también se usa como intermediario de mensajes para facilitar la comunicación entre los microservicios y otros sistemas distribuidos. Redis también admite transacciones, mensajes de publicación/suscripción y otras funciones avanzadas.

En general, Redis es una herramienta poderosa y versátil que se usa ampliamente para una variedad de aplicaciones. Es especialmente adecuado para aplicaciones que requieren un acceso rápido a grandes cantidades de datos y pueden beneficiarse del almacenamiento de datos en la memoria.

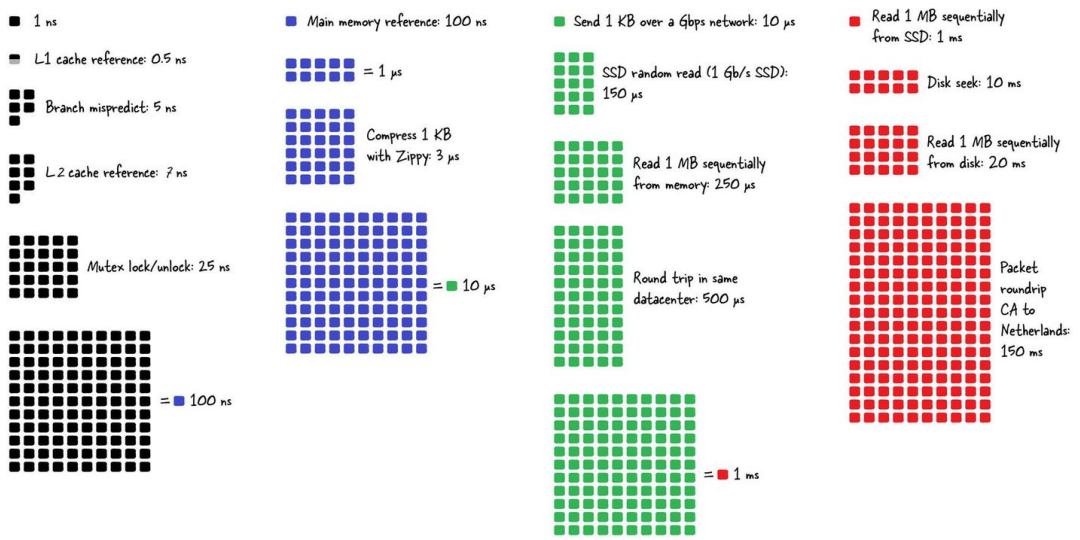
Principalmente, Redis es una base de datos en memoria que se utiliza como caché frente a otra base de datos "real" como MySQL o PostgreSQL para ayudar a mejorar el rendimiento de la aplicación. Aprovecha la velocidad de la memoria y alivia la carga de la base de datos central de la aplicación para:

- Datos que cambian con poca frecuencia y se solicitan con frecuencia
- Datos que son menos críticos para la misión y evolucionan con frecuencia.



Sin embargo, para muchos casos de uso, Redis ofrece suficientes garantías de que se puede usar como una base de datos principal completa. Junto con los complementos de Redis y sus diversas configuraciones de alta disponibilidad (HA), Redis como base de datos se ha vuelto increíblemente útil para ciertos escenarios y cargas de trabajo.

Otro aspecto importante es que Redis difuminó las líneas entre un caché y un almacén de datos. Una nota importante para entender aquí es que leer y manipular datos en la memoria es mucho más rápido que cualquier cosa posible en los almacenes de datos tradicionales que usan SSD o HDD.



Redis vs Memcached

Memcached fue creado por Brad Fitzpatrick en 2003, antecediendo a Redis por seis años. Originalmente comenzó como un proyecto de Perl y luego fue reescrito en C. Era la herramienta de almacenamiento en caché de facto de su época. El principal punto que lo diferencia de Redis es su falta de tipos de datos y su política de desalojo limitada de solo LRU (least recently used).
Otra diferencia es que Redis es del tipo “single-threaded” mientras que Memcached es multithreaded. Memcached puede funcionar en un entorno estrictamente de almacenamiento en caché, pero requiere alguna configuración en un clúster distribuido, mientras que Redis tiene soporte para esto de manera inmediata.

	Memcached	**Redis**
Sub-millisecond latency	Yes	Yes
Developer ease of use	Yes	Yes
Data partitioning	Yes	Yes
Support for a broad set of programming languages	Yes	Yes
Advanced data structures	-	Yes
Multithreaded architecture	Yes	-
Snapshots	-	Yes

Replication	-	Yes
Transactions	-	Yes
Pub/Sub	-	Yes
Lua scripting	-	Yes
Geospatial support	-	Yes

AWS ElastiCache

Amazon Elasticache es un servicio web que facilita la implementación, el funcionamiento y el escalado de una caché en memoria en la nube. Es un servicio completamente administrado que le permite usar los almacenes de datos en memoria de código abierto Redis y Memcached sin la necesidad de configurar y administrar su propia infraestructura de caché.

Con Elasticache, puede configurar, escalar y administrar fácilmente un caché que se puede usar para mejorar el rendimiento de sus aplicaciones web. Elasticache maneja automáticamente tareas como el aprovisionamiento, la configuración, la aplicación de parches y las copias de seguridad, para que pueda concentrarse en sus aplicaciones en lugar de en la infraestructura subyacente. Elasticache también se integra a la perfección con otros servicios de AWS, lo que facilita el uso de su caché con otros servicios de AWS, como Amazon RDS, Amazon EC2 y Amazon S3.

En general, AWS Elasticache es una forma conveniente y rentable de utilizar el almacenamiento en caché en memoria para mejorar el rendimiento de sus aplicaciones web. Es un servicio completamente administrado que facilita la configuración y el funcionamiento de una memoria caché, y se integra a la perfección con otros servicios de AWS.

Certificados SSL/TLS

Los certificados SSL/TLS son utilizados para establecer una conexión segura entre un servidor web y un navegador web. Esto se logra mediante el uso de criptografía para encriptar la información transmitida entre el servidor y el navegador, lo que evita que terceros puedan acceder a la información. Esto es especialmente importante cuando se transmiten datos sensibles, como información bancaria o contraseñas. Los certificados SSL/TLS también pueden ser utilizados para verificar la identidad del servidor web, lo que ayuda a prevenir ataques de phishing. En resumen, los certificados SSL/TLS son una herramienta importante para proteger la privacidad y la seguridad de la información transmitida en línea.

¿Cómo funciona TLS?

TLS es un protocolo de encriptación y autenticación diseñado para asegurar las comunicaciones por Internet. Un protocolo de enlace TLS es el proceso que inicia una sesión de comunicación que utiliza TLS. Durante un protocolo de enlace TLS, los dos lados

que se comunican intercambian mensajes para reconocerse, verificarse, establecer los algoritmos criptográficos que utilizarán y acordar las claves de sesión. Los protocolos de enlace TLS son una parte fundamental del funcionamiento de HTTPS.

¿Qué sucede durante el establecimiento de una sesión TLS?

Durante el transcurso de un handshake TLS, el cliente y el servidor juntos harán lo siguiente:

- Especifique qué versión de TLS (TLS 1.0, 1.2 o 1.3) usarán.
- Decidir qué suites de cifrado usarán.
- Autenticar la identidad del servidor a través de la clave pública del servidor y la firma digital de la autoridad de certificación SSL.
- Genere claves de sesión para usar el cifrado simétrico después de que se complete el protocolo de enlace.

Los beneficios de la seguridad de la capa de transporte (TLS)

Los beneficios de TLS son sencillos cuando se analiza el uso versus no uso de TLS. Como se indicó anteriormente, una sesión cifrada con TLS proporciona un mecanismo de autenticación seguro, cifrado de datos y controles de integridad de datos. Sin embargo, al comparar TLS con otro conjunto de protocolos de cifrado y autenticación segura, como Internet Protocol Security, TLS ofrece beneficios adicionales y es una de las razones por las que IPsec se está reemplazando con TLS en muchas situaciones de implementación empresarial. Estos incluyen beneficios como los siguientes:

- La seguridad está integrada directamente en cada aplicación, a diferencia del software o hardware externo para construir túneles IPsec.
- Existe un verdadero cifrado de extremo a extremo (E2EE - End to end encryption) entre los dispositivos que se comunican.
- Existe un control granular sobre lo que se puede transmitir o recibir en una sesión cifrada.
- Dado que TLS opera dentro de las capas superiores del modelo de interconexión de sistemas abiertos (OSI), no tiene las complicaciones de traducción de direcciones de red (NAT) que son inherentes a IPsec.

CertBot

Let's Encrypt Certbot es una herramienta de línea de comandos gratuita y de código abierto para obtener y administrar certificados de Secure Sockets Layer/Transport Layer Security (SSL/TLS). Certbot es proporcionado por la organización sin fines de lucro Let's Encrypt, que se dedica a facilitar que los propietarios de sitios web protejan sus sitios con certificados.
Certbot automatiza el proceso de obtención y renovación de certificados SSL/TLS de Let's Encrypt. Es una herramienta local que debe instalarse y ejecutarse en el servidor. Una vez instalado, Certbot se puede utilizar para solicitar un nuevo certificado SSL/TLS o para renovar un certificado existente que está a punto de caducar.

Certbot también proporciona una serie de funciones y opciones útiles que facilitan la gestión de certificados SSL/TLS. Por ejemplo, puede configurar automáticamente un sitio web para usar el certificado recién obtenido o renovado, y también puede renovar automáticamente los certificados antes de que caduquen. Esto facilita que los propietarios de sitios web mantengan sus sitios seguros con certificados SSL/TLS válidos.
Posee varios plugins que le permiten integrarse con los principales servidores web, Apache y NGINX, y con diferentes DNS como Cloudflare, AWS Route53, Google DNS, DigitalOcean para automatizar el proceso de verificación de dominio.

AWS Certificate Manager

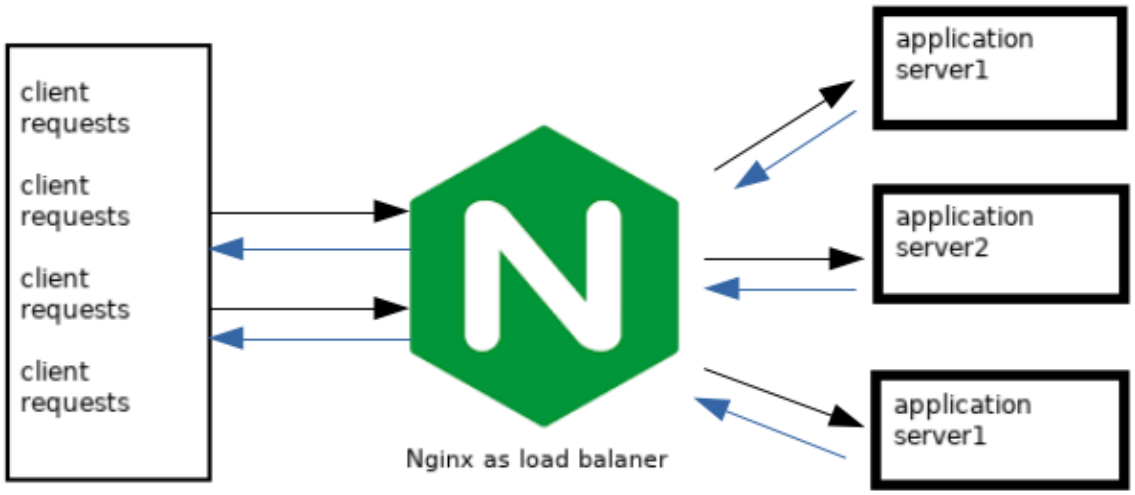
Si usa HTTPS para el frontend, debe implementar un certificado SSL/TLS en el balanceador de carga. El balanceador de carga usa el certificado para establecer y cifrar la conexión y luego descifrar las solicitudes de los clientes antes de enviarlas a las instancias internas.
Los protocolos SSL y TLS utilizan un certificado X.509 (certificado de servidor SSL/TLS) para autenticar tanto al cliente como a la aplicación de backend. Un certificado X.509 es una forma digital de identificación emitida por una autoridad certificadora (CA) y contiene información de identificación, un período de validez, una clave pública, un número de serie y la firma digital del emisor.
Puede crear un certificado con AWS Certificate Manager o una herramienta que admita los protocolos SSL y TLS, como OpenSSL. Deberá especificar este certificado cuando cree o actualice un agente de escucha (listener) HTTPS en el balanceador de carga. Cuando crea un certificado para usarlo con su balanceador de carga, debe especificar un nombre de dominio y pasar un proceso de verificación.
Para emitir un certificado SSL/TLS válido, una autoridad de certificación (CA) debe verificar que el solicitante sea el propietario legítimo del dominio o sitio web para el que se solicita el certificado. Esto garantiza que el certificado sólo se emita a entidades de confianza y ayuda a evitar que actores malintencionados obtengan certificados para dominios que no son de su propiedad.
Para verificar la identidad del solicitante, una CA puede solicitar que el solicitante proporcione una prueba de propiedad del dominio, por ejemplo, respondiendo a un correo electrónico enviado a una dirección asociada con el dominio, o colocando un archivo específico en el servidor del sitio web o agregando un registro puntual en el DNS del dominio.
ACM también integra con Route53, el servicio DNS de AWS, y permite automatizar el proceso de verificación de todos los dominios delegados a esa cuenta de Amazon, ahorrando tiempo y facilitando el proceso.

Balanceador de carga

NGINX

El balanceo de carga es una excelente manera de escalar una aplicación y aumentar su rendimiento y redundancia. NGINX, un popular software de servidor web, se puede configurar como un balanceador de carga simple pero potente para mejorar la disponibilidad y la eficiencia de los recursos de sus servidores.

¿Cómo funciona Nginx? Nginx actúa como un único punto de entrada a una aplicación web distribuida que funciona en varios servidores independientes.



Métodos de balanceo ofrecidos:

- round-robin: Las solicitudes a los servidores de aplicaciones se distribuyen por turnos.
- Menos solicitado: La siguiente solicitud se asigna al servidor con el menor número de conexiones activas.
- ip-hash: se utiliza una función hash para determinar qué servidor debe seleccionarse para la próxima solicitud (según la dirección IP del cliente).

Configuración:

```
http {
    upstream myapp1 {
        server srv1.example.com;
        server srv2.example.com;
        server srv3.example.com;
    }

    server {
        listen 80;

        location / {
```

```
            proxy_pass http://myapp1;
        }
    }
}
```

El archivo de arriba define en pocas líneas cómo configurar un simple balanceador de carga con NGINX, si bien es bastante sencillo puede ganar complejidad bastante rápido, dependiendo de la cantidad de instancias que debe balancear y si sus direcciones de red son estáticas o dinámicas, es poco automatizable, además de que requiere un buen know-how de networking y linux para levantarlo, configurarlo y mantenerlo, por ese motivo se suele evitar esa carga e ir por una solución administrada como lo son los Load Balancers de AWS.

AWS Load Balancing

Los balanceadores de carga son un elemento omnipresente en un entorno de nube. Tan pronto como se necesite alta disponibilidad, es probable que se encuentre con un balanceador de carga frente a al menos dos instancias de su aplicación.

AWS ofrece tres tipos de balanceadores de carga, adaptados para varios escenarios: Elastic Load Balancers, Application Load Balancers, and Network Load Balancers.

Elastic Load Balancers

ELB o también conocido como Classic Load Balancers funciona tanto en la capa 4 (TCP) como en la 7 (HTTP). Además, es el único balanceador que admite cookies de sesión persistentes definidas por la aplicación; por el contrario, ALB usa sus propias cookies y no se tiene control sobre eso.
En la capa 7, ELB puede terminar el tráfico TLS. También puede volver a cifrar el tráfico a los objetivos siempre que proporcionen un certificado SSL (por cierto, un certificado autofirmado sirve). Esto proporciona cifrado de extremo a extremo, que es un requisito habitual en muchos programas de cumplimiento. Opcionalmente, ELB se puede configurar para verificar el certificado TLS proporcionado por el objetivo para mayor seguridad.
ELB tiene bastantes limitaciones. Por ejemplo, no es compatible con los contenedores EKS (Elastic Kubernetes Service) que se ejecutan en Fargate. Además, no puede reenviar tráfico en más de un puerto por instancia y no admite el reenvío a direcciones IP; solo puede reenviar a instancias o contenedores EC2 explícitos en ECS o EKS. Finalmente, ELB no es compatible con websockets; sin embargo, es posible que pueda solucionar esta limitación utilizando la capa 4.

Se puede considerar como una instancia de Nginx o HAProxy si eso lo hace más fácil de entender.

Application Load Balancer

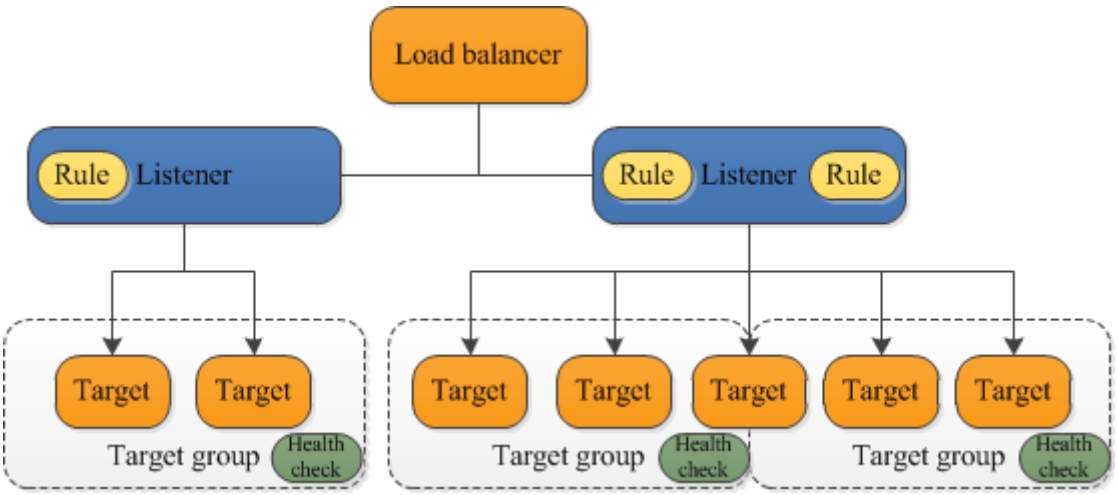
Un Application Load Balancer (ALB) solo funciona en la capa 7 (HTTP). Tiene una amplia gama de reglas de enrutamiento para las solicitudes entrantes según el nombre del host, la ruta, el parámetro de cadena de consulta, el método HTTP, los encabezados HTTP, la IP de

origen o el número de puerto. Por el contrario, ELB solo permite el enrutamiento según el número de puerto. Además, a diferencia de ELB, ALB puede enrutar solicitudes a muchos puertos en un solo destino. ALB también puede enrutar solicitudes a funciones Lambda.
Una característica muy útil de ALB es que se puede configurar para devolver una respuesta fija o una redirección. Por lo tanto, no necesita un servidor para realizar tareas tan básicas porque todo está integrado en el propio ALB.
ALB también es compatible con la indicación de nombre de servidor (SNI), lo que le permite servir muchos nombres de dominio. (Por el contrario, ELB solo puede servir un nombre de dominio). Sin embargo, existe un límite en la cantidad de certificados que puede adjuntar a un ALB, 25 certificados más el certificado predeterminado.
Los ALB se utilizan normalmente para aplicaciones web. Si tiene una arquitectura de microservicios, ALB se puede usar como un balanceador de carga interno frente a instancias EC2 o contenedores Docker que implementan un servicio determinado. También puede usarlos frente a una aplicación que implemente una API REST, aunque AWS API Gateway generalmente sería una mejor opción aquí.

Network Load Balancers

Un Network Load Balancer (NLB) funciona solo en la capa 4 y puede manejar tanto TCP como UDP, así como conexiones TCP cifradas con TLS. Su característica principal es que tiene un rendimiento muy alto. Además, utiliza direcciones IP estáticas y se le pueden asignar direcciones IP elásticas, algo que no es posible con ALB y ELB.
NLB conserva de forma nativa la dirección IP de origen en los paquetes TCP/UDP; por el contrario, ALB y ELB se pueden configurar para agregar encabezados HTTP adicionales con información de reenvío, y su aplicación debe analizarlos correctamente.
Los NLB se usarían para cualquier cosa que los ALB no cubran. Un caso de uso típico sería un servicio de transmisión de datos casi en tiempo real (video, cotizaciones de acciones, etc.). Otro caso típico es que necesitaría usar un NLB si su aplicación usa protocolos distintos de HTTP.

Un dato importante es que tanto ALB como NLB usan el concepto de "target group", que es un nivel adicional de redirección. Se puede conceptualizar de esta manera. Los oyentes reciben solicitudes y deciden (basándose en una amplia gama de reglas) a qué target group reenviarán las solicitudes. Luego, un target group enruta las solicitudes a instancias, contenedores o direcciones IP. Los target group administran los objetivos en términos de decidir cómo dividir el tráfico y realizar “Health Checks” en los objetivos.



Health Checks

Los health checks en balanceadores de carga de Amazon Web Services (AWS) son una herramienta que se utiliza para determinar si una instancia de servidor o aplicación está disponible para recibir tráfico. Los health checks se realizan en intervalos regulares para asegurar que las instancias estén disponibles y sean capaces de procesar solicitudes. Si una instancia falla un health check, el balanceador de carga dejará de enviar tráfico a esa instancia y, en su lugar, lo enviará a otras instancias que estén disponibles.

Los health checks son especialmente útiles en entornos de alta disponibilidad, ya que permiten al balanceador de carga detectar y evitar el envío de tráfico a instancias que no estén disponibles. También pueden utilizarse para garantizar que las instancias estén disponibles y sean capaces de procesar solicitudes de manera rápida y eficiente.

Hay diferentes tipos de health checks disponibles en AWS, incluyendo health checks basados en ping, health checks basados en solicitudes HTTP y health checks basados en TCP.

Web Application Firewall

Web Application Firewall (WAF) es un firewall de aplicaciones, al igual que un firewall de red trabaja filtrando los paquetes en sus políticas o reglas, un WAF está especializado en la capa de aplicación (Capa 7 del modelo OSI), filtrando de manera minuciosa el tráfico a nivel de aplicación, pudiendo detectar los ataques y estableciendo filtro o reglas personalizadas o definidas en el estándar OWASP (Open Web Application Security Project). OWASP es una metodología de seguridad de código abierto y colaborativa que se utiliza como referente para auditorias de seguridad de aplicaciones web

NGINX con ModSecurity

Cuando se utiliza NGINX con ModSecurity, NGINX se encarga de servir el contenido web y ModSecurity se encarga de proteger la aplicación web contra ataques. ModSecurity se integra en el flujo de tráfico de NGINX y analiza las solicitudes entrantes en busca de patrones sospechosos que puedan indicar un intento de ataque. Si se detecta una solicitud sospechosa, ModSecurity puede bloquearla o alterarla para evitar que llegue a la aplicación

web. Esto ayuda a proteger la aplicación web de ataques como inyecciones de SQL, ataques XSS y otros tipos de ataques comunes.
Es una herramienta muy flexible y potente, pero requiere un profundo conocimiento en infraestructura para configurarlo y mantenerlo, ya que de la configuración de ModSec se desprende la robustez de la seguridad de nuestra información.

AWS WAF

Amazon Web Services WAF es un servicio de firewall web que se puede utilizar para proteger aplicaciones web de ataques. AWS WAF permite a los administradores de sistemas definir reglas de firewall que determinan qué solicitudes web son permitidas y cuáles son bloqueadas. Por ejemplo, un administrador de sistemas puede crear reglas para bloquear solicitudes que contengan determinadas cadenas de caracteres maliciosas, como caracteres de inyección de SQL, o que provengan de determinadas direcciones IP maliciosas. También ya trae consigo un grupo de reglas administradas para proteger aplicaciones o sistemas comunes como sitios Wordpress, servidores Windows/Linux o aplicaciones PHP, entre otras.

AWS WAF se integra con otros servicios de AWS, como Amazon CloudFront, Amazon API Gateway o Amazon Application Load Balancer para proteger aplicaciones web distribuidas en la nube. AWS WAF también se puede integrar con herramientas de detección de amenazas externas, como Threat Intelligence Feeds, para obtener información en tiempo real sobre las últimas amenazas y utilizarla para crear reglas de firewall más precisas. AWS WAF ofrece una interfaz de usuario sencilla que permite a los administradores de sistemas crear, gestionar y monitorear fácilmente sus reglas de firewall.

Almacenamiento

AWS S3

Amazon Simple Storage Service (Amazon S3) es un servicio de almacenamiento de objetos. Proporciona una forma sencilla y segura de almacenar y recuperar cualquier cantidad de datos desde cualquier lugar en Internet.
S3 se utiliza a menudo para almacenar y recuperar grandes cantidades de datos, como imágenes, vídeos, audios, documentos y otros tipos de archivos. También puede utilizarse para realizar copias de seguridad y para la distribución de contenidos a través de la red de distribución de contenidos (CDN) de AWS, Amazon CloudFront.
S3 ofrece un alto nivel de disponibilidad y escalabilidad, y puede gestionar grandes cantidades de tráfico y datos sin problemas. Además, ofrece opciones de seguridad y privacidad avanzadas, como el cifrado de datos y la autenticación de usuarios.

AWS EFS

Amazon Elastic File System (Amazon EFS) es un servicio de almacenamiento de archivos. Proporciona un sistema de archivos montable en la nube que puede utilizarse para almacenar y acceder a archivos desde varias instancias EC2 a la vez.

EFS es ideal para aplicaciones que necesitan acceder a archivos de manera concurrente desde varias instancias, como aplicaciones de bases de datos, entornos de desarrollo y pruebas, y entornos de producción de alta escalabilidad.
EFS ofrece un alto nivel de disponibilidad y rendimiento, y puede escalar automáticamente para adaptarse a las necesidades de almacenamiento de la aplicación.

AWS EBS

Amazon Elastic Block Store (Amazon EBS) es un servicio de almacenamiento de bloques. Proporciona una forma de almacenar y recuperar datos de manera persistente y segura para instancias EC2.
EBS se utiliza a menudo para almacenar sistemas operativos, aplicaciones y datos que deben estar disponibles de manera continua. Los volúmenes de EBS se pueden montar como discos duros lógicos en instancias EC2 y se pueden utilizar como cualquier otro disco duro.
EBS ofrece diferentes tipos de volúmenes, cada uno de los cuales está optimizado para diferentes usos. Por ejemplo, los volúmenes SSD (Solid State Drive) proporcionan un rendimiento alto y son adecuados para aplicaciones que requieren acceso aleatorio a los datos, mientras que los volúmenes HDD (Hard Disk Drive) ofrecen una mayor capacidad de almacenamiento a un precio más bajo y son adecuados para aplicaciones que requieren acceso secuencial a los datos.

Content Distribution Network

Los CDN son redes de distribución de contenidos o Content Delivery Network. Su función principal es acelerar la entrega de contenido estático, como imágenes, videos y archivos JavaScript y CSS, a los usuarios finales.

Los CDN se basan en una red de servidores distribuidos geográficamente que almacenan copias de los contenidos de un sitio web. Cuando un usuario solicita acceder a una página web, el CDN redirige la solicitud a uno de sus servidores más cercanos al usuario, lo que reduce la latencia y mejora la velocidad de carga de la página.

Los CDN son especialmente útiles para sitios web con un alto tráfico y una amplia distribución geográfica de usuarios, ya que permiten a los usuarios acceder a los contenidos de forma más rápida y eficiente. También son útiles para proteger contra ataques de denegación de servicio (DoS) y mejorar la disponibilidad de un sitio web.

Algunos de los CDN más conocidos y utilizados a nivel mundial son:

- Akamai Technologies: Uno de los CDN más grandes y con mayor presencia en el mercado, con más de 216.000 servidores en más de 130 países.
- Cloudflare: Un CDN y plataforma de seguridad en la nube con más de 200 millones de sitios web protegidos y más de 200 data centers en todo el mundo.
- Fastly: Un CDN y plataforma de optimización de contenidos en la nube con más de 70 data centers en todo el mundo.

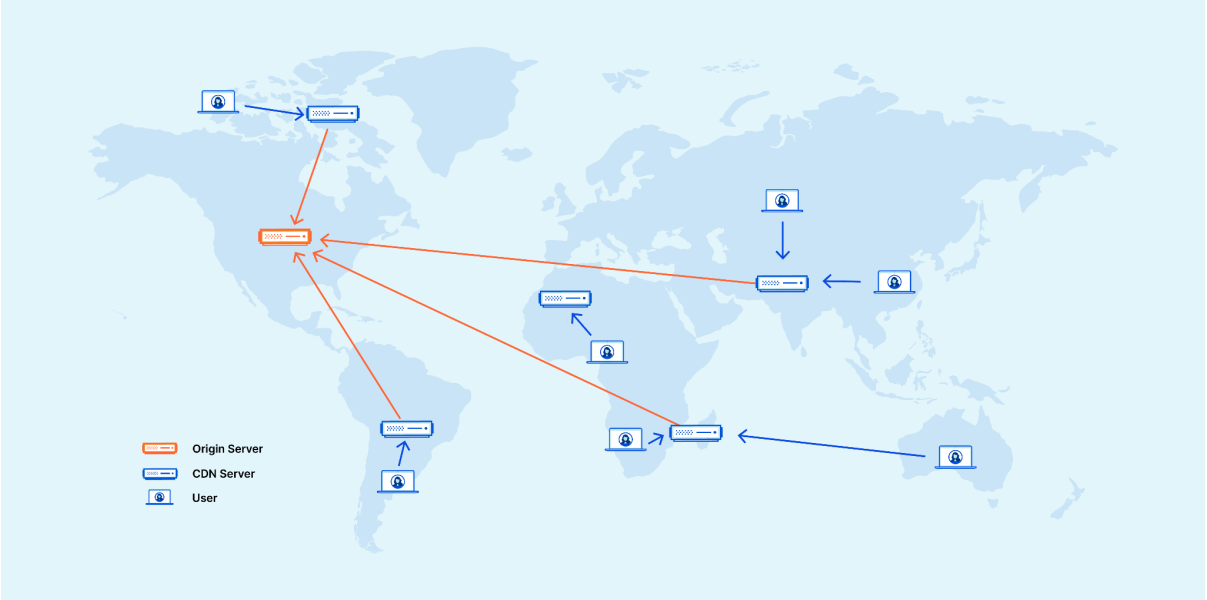
- Amazon Web Services (AWS): La plataforma de nube de Amazon ofrece un CDN conocido como Amazon CloudFront, con más de 100 puntos de presencia en todo el mundo.
- Google Cloud: La plataforma de nube de Google ofrece un CDN conocido como Google Cloud CDN, con más de 70 puntos de presencia en todo el mundo.

Cloudflare CDN

Cloudflare CDN es una red de entrega de contenidos que mejora el rendimiento de los sitios web al reducir la distancia que los datos deben viajar para llegar al usuario final. Esto se logra mediante el almacenamiento en caché del contenido del sitio web en servidores de borde ubicados en todo el mundo.

Las ventajas de Cloudflare CDN incluyen un mayor rendimiento, seguridad mejorada, reducción del ancho de banda y menor latencia. Sin embargo, algunas desventajas incluyen la posible pérdida de control sobre los datos almacenados en caché y la posibilidad de que se produzcan errores en la entrega de contenido si la red de borde de Cloudflare no está actualizada.

Cloudflare CDN ofrece planes gratuitos y pagos. El plan gratuito proporciona servicios básicos de CDN y seguridad, mientras que los planes pagados ofrecen características más avanzadas y opciones de personalización.



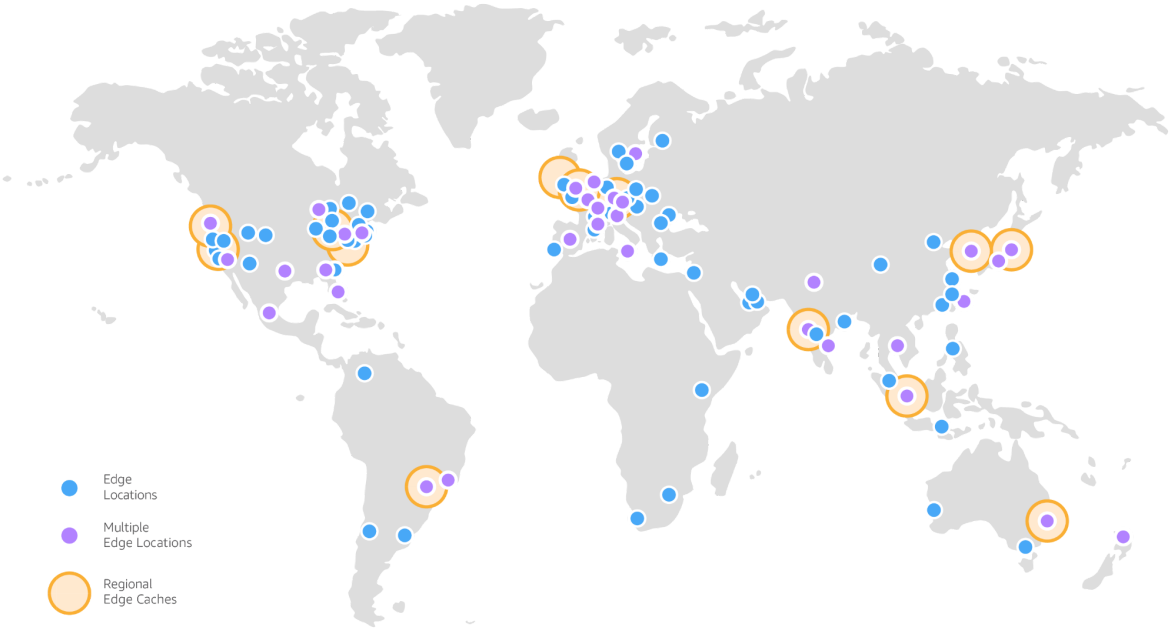
CDN Endpoints de Cloudflare CDN

AWS CloudFront

CloudFront también ofrece otras características útiles, como la posibilidad de utilizar SSL/TLS para cifrar la comunicación entre el usuario y los servidores de CloudFront, la integración con servicios de aceleración de aplicaciones web como Amazon EC2 y Elastic

Load Balancing, y la opción de utilizar reglas de restricción de acceso basadas en la ubicación del usuario o el tipo de dispositivo.

Para utilizar CloudFront, se debe configurar una distribución y especificar la ubicación de los contenidos originales (por ejemplo, un servidor web o un almacenamiento en la nube como Amazon S3 o Amazon EBS). Luego, se puede utilizar una URL de CloudFront para acceder a los contenidos a través de la red de servidores de CloudFront en lugar de acceder directamente a la ubicación original de los contenidos.



Edge locations de AWS CloudFront

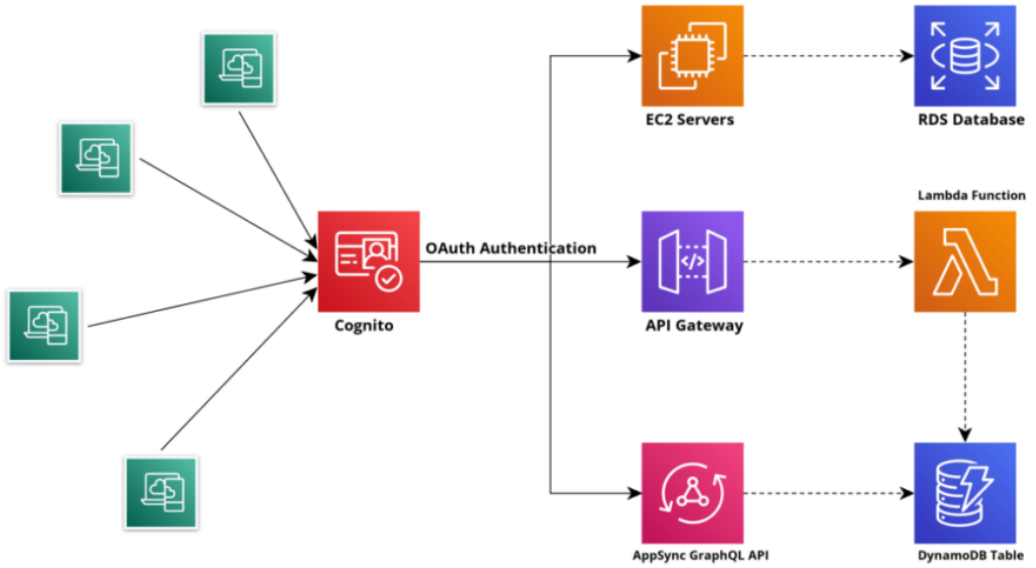
IAM - Identity and Access Management

AWS Cognito

Amazon Cognito es un servicio que proporciona autenticación, autorización y gestión de usuarios en aplicaciones móviles y web. Con Cognito, se puede simplificar la gestión de usuarios y la autenticación de aplicaciones móviles y web.
Cognito ofrece dos principales funcionalidades:

- Gestión de usuarios: Cognito permite agregar fácilmente un registro e inicio de sesión a tus aplicaciones móviles y web. También se puede permitir que los usuarios se registren utilizando sus credenciales de redes sociales (por ejemplo, Facebook, Google, Amazon) o proporcionando un nombre de usuario y contraseña.
- Autenticación de usuarios: Cognito ofrece autenticación de dos factores y verificación por SMS para aumentar la seguridad de los usuarios. Además, es posible integrar Cognito con servicios de identidad existentes (por ejemplo, Active Directory, SAML) para permitir a los usuarios iniciar sesión utilizando sus credenciales de identidad existentes.

En resumen, Amazon Cognito es una solución completa para la gestión de usuarios y la autenticación en aplicaciones móviles y web.



KeyCloak

Keycloak es una plataforma de gestión de identidades y acceso (Identity and Access Management, IAM) open source. Sirve como intermediario entre una aplicación y diferentes proveedores de identidades externos, como Google, Facebook y Twitter.

Keycloak es especialmente útil en entornos empresariales, ya que proporciona una capa adicional de seguridad y facilita la administración de usuarios y permisos en aplicaciones de gran escala. También es muy flexible y se puede integrar con una amplia variedad de aplicaciones y servicios.

Algunas desventajas de Keycloak que se pueden mencionar son:

- Es una plataforma open source, lo que significa que no cuenta con el mismo nivel de soporte técnico y servicios profesionales que otras plataformas de gestión de identidades y acceso comerciales.
- Puede ser un poco compleja de configurar y administrar para usuarios sin conocimientos técnicos avanzados.
- Aunque Keycloak es compatible con una amplia variedad de proveedores de identidades externos, puede ser un poco limitado en términos de integraciones con otros servicios y aplicaciones.
- Al ser una plataforma open source, es posible que no cuente con las mismas funcionalidades y características avanzadas que otras plataformas de gestión de identidades y acceso comerciales.

CI/CD

Para automatizar la implementación de aplicaciones y garantizar una entrega constante de software, se utilizan servicios de Integración Continua/Entrega Continua (CI/CD). En el

mercado existen diversas opciones para desplegar tanto el código de la aplicación como el de la infraestructura de manera eficiente. En esta sección se presentan los servicios de CI/CD más comunes utilizados actualmente.

Jenkins

Jenkins es una popular herramienta de integración continua (CI) y entrega continua (CD) de código abierto que se usa ampliamente para crear e implementar proyectos de software. Es altamente configurable y se puede utilizar para una amplia gama de proyectos y propósitos. Aquí hay algunas ventajas y desventajas de Jenkins:
Ventajas:

- Jenkins es de código abierto y de uso gratuito, lo que lo convierte en una solución rentable para las organizaciones.
- Es altamente personalizable y puede configurarse para adaptarse a las necesidades de un proyecto u organización específicos.
- Jenkins tiene una comunidad grande y activa de usuarios y desarrolladores, por lo que cuenta con un buen soporte y hay muchos recursos disponibles para solucionar problemas y aprender a usarlo.
- Jenkins se integra con una amplia gama de herramientas y tecnologías, incluidos sistemas de control de versiones, marcos de prueba y herramientas de implementación.
- Tiene una interfaz fácil de usar y configurar, incluso para usuarios con poca o ninguna experiencia técnica.
- Jenkins dispone de la funcionalidad de extenderse mediante plugins. Existen variedad de plugins que permiten cambiar el comportamiento de Jenkins o añadir nuevas funcionalidades.

Desventajas:

- Jenkins puede ser complejo de instalar y configurar, especialmente para proyectos grandes o complejos.
- Requiere un servidor o host dedicado para ejecutarse, lo que puede ser una carga para organizaciones o equipos pequeños.
- Jenkins puede consumir muchos recursos y puede requerir hardware e infraestructura significativos para funcionar de manera eficiente.
- Puede ser difícil de mantener y actualizar, especialmente si no se usa regularmente o si no está configurado correctamente.

CircleCI

CircleCI es una herramienta de CI/CD que admite el rápido desarrollo y publicación de software. CircleCI permite la automatización en toda la canalización del usuario, desde la creación de código, las pruebas hasta la implementación.
Puede integrar CircleCI con GitHub, GitHub Enterprise y Bitbucket para crear compilaciones cuando se confirman nuevas líneas de código. CircleCI también aloja la integración continua bajo la opción administrada en la nube o se ejecuta detrás de un firewall en una infraestructura privada.

Características clave de CircleCI:

- Se integra con Bitbucket, GitHub y GitHub Enterprise
- Ejecuta compilaciones usando un contenedor o una máquina virtual
- Fácil depuración
- Pruebas rápidas
- Correo electrónico personalizado y notificaciones mediante mensajería instantánea
- Despliegue continuo y específico para cada branch
- Altamente personalizable
- Merge automatizado y comandos personalizados para la carga de paquetes
- Configuración rápida y compilaciones ilimitadas

GitLab CI

GitLab CI es una parte de GitLab. Es una aplicación web con una API que almacena su estado en una base de datos. Es una de las mejores herramientas para la Integración Continua que gestiona proyectos y proporciona una interfaz de usuario amigable, además de ofrecer la ventaja de todas las características de GitLab.
Características:

- GitLab Container Registry es un registro seguro para imágenes de Docker
- Proporciona API para la mayoría de las funciones, por lo que permite a los desarrolladores crear integraciones más profundas con el producto.
- Ayuda a los desarrolladores a poner su idea en producción al encontrar áreas de mejora en su proceso de desarrollo.
- Le ayuda a mantener su información segura con problemas confidenciales
- Los proyectos internos en GitLab permiten promover el abastecimiento de repositorios internos.

TeamCity

TeamCity es un servicio de gestión de compilación e integración continua de JetBrains que ayuda a construir e implementar diferentes tipos de proyectos. TeamCity se ejecuta en un entorno Java. La herramienta se puede instalar en servidores Windows y Linux, admite proyectos .NET y de pila abierta.

TeamCity 2019.1 proporciona una nueva interfaz de usuario e integración nativa de GitLab. También es compatible con las pull requests desde GitLab y Bitbucket.
Características clave:

- Proporciona varias formas de reutilizar los ajustes y configuraciones de un proyecto principal en un subproyecto
- Ejecuta compilaciones paralelas simultáneamente en diferentes entornos
- Permite ejecutar compilaciones de historial, ver informes de historial de pruebas, fijar, etiquetar y agregar compilaciones a favoritos
- Fácil de personalizar, interactuar y ampliar el servidor
- Mantiene el servidor CI funcional y estable
- Gestión de usuarios flexible, asignación de roles de usuario, clasificación de usuarios en grupos, diferentes formas de autenticación de usuarios y un registro con todas las acciones de los usuarios para la transparencia de todas las actividades en el servidor

Travis CI

Travis CI es un servicio de CI que se utiliza para crear y testear proyectos. Travis CI detecta automáticamente los commits realizados a un repositorio GitHub y automáticamente compila el proyecto y ejecuta las pruebas correspondientes.

La herramienta brinda soporte para muchas configuraciones de compilación y lenguajes como Node, PHP, Python, Java, Perl, etc.

Características de Travis CI:

- Configuración rápida
- Vistas de compilación en vivo para el monitoreo de proyectos de GitHub
- Soporte de solicitud de extracción
- Implementación en múltiples servicios en la nube
- Servicios de base de datos preinstalados
- Despliegues automáticos en compilaciones pasadas
- Cada compilación se ejecuta en una máquina virtual limpia.
- Admite varios lenguajes, como Android, C, C#, C++, Java, JavaScript (con Node.js), Perl, PHP, Python, R, Ruby, etc.

GitHub Actions

GitHub Actions es una plataforma de integración y entrega continua (CI/CD) que permite a los desarrolladores automatizar sus flujos de trabajo de desarrollo de software en la plataforma de GitHub. Está diseñada para ser fácil de usar e integrarse con otras herramientas y tecnologías, lo que la convierte en una opción popular para los desarrolladores que desean automatizar sus procesos de compilación, prueba e implementación.

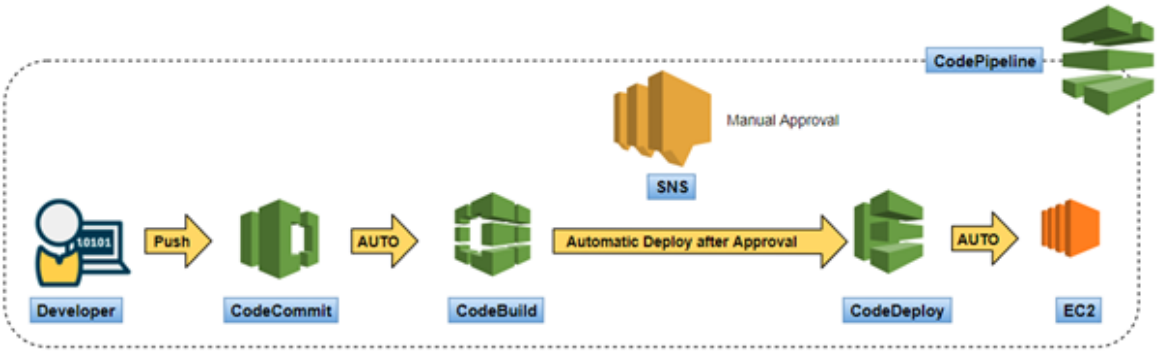
Algunas ventajas de usar GitHub Actions incluyen:

- Está integrado con GitHub, lo que significa que los desarrolladores pueden usarlo para automatizar sus flujos de trabajo directamente desde la interfaz de GitHub.
- Es fácil de configurar y usar, con un gran número de acciones pre construidas disponibles para tareas comunes como compilar e implementar código en variedad de lenguajes y plataformas.
- GitHub Actions es altamente escalable y se puede usar para automatizar flujos de trabajo para proyectos de cualquier tamaño.
- Se integra con una amplia gama de herramientas y tecnologías, incluyendo las populares herramientas CI/CD y otras herramientas para desarrolladores.
- Es gratis de usar para proyectos de código abierto e incluye varios planes de pago para proyectos privados.

Algunos posibles inconvenientes de usar GitHub Actions incluyen:

- Es una herramienta relativamente nueva, lo que significa que puede no contar con el mismo nivel de soporte y recursos que otras herramientas CI/CD más establecidas.
- Puede no ser adecuado para proyectos con requisitos de seguridad o cumplimiento estrictos, ya que está alojado en la plataforma de GitHub.
- Puede no ofrecer tantas funciones avanzadas o opciones de personalización como otras herramientas CI/CD.
- Puede no ser la mejor opción para las organizaciones que necesitan alojar su propia infraestructura CI/CD o tienen requisitos específicos para el alojamiento y la implementación.

En AWS



AWS CodePipeline

AWS CodePipeline es un servicio de entrega continua completamente administrado que ayuda a automatizar pipelines de despliegues para obtener actualizaciones de aplicaciones rápidas y confiables. Se integra con otros servicios de AWS, como CodeCommit, CodeBuild y CodeDeploy.

AWS CodeCommit

CodeCommit es un servicio de control de versiones que le facilita alojar repositorios de Git en la nube. Le permite almacenar, rastrear y administrar los cambios de código a lo largo del tiempo, lo que le permite colaborar con su equipo y crear software de manera más eficiente.

AWS CodeBuild

CodeBuild es un servicio de compilación completamente administrado que compila código fuente, ejecuta pruebas y produce paquetes de software que están listos para implementar. Se puede utilizar como un servicio independiente o como parte de una canalización de CI/CD.

AWS CodeDeploy

CodeDeploy es un servicio de implementación que ayuda a automatizar la implementación de aplicaciones en instancias de Amazon EC2, instancias locales o funciones de AWS Lambda sin servidor. Se puede usar para implementar cambios de código en una variedad de entornos, incluidos producción, preparación y prueba.

Juntos, estos servicios forman un poderoso conjunto de herramientas que se pueden utilizar para automatizar y optimizar sus procesos de desarrollo e implementación de software en AWS. Al usar CodePipeline para orquestar el flujo de cambios de código a través de su canalización, puede ofrecer nuevas funciones y actualizaciones a sus usuarios de manera más rápida y confiable.

En Azure

Azure DevOps es una colección de servicios proporcionados por Microsoft Azure. Proporciona servicios de desarrollo para que un equipo soporte, planifique, colabore, cree e implemente aplicaciones. Proporciona funciones integradas en un navegador o un IDE (Entorno de desarrollo integrado). Algunos de los servicios para desarrolladores son:

- Azure Repos
- Azure Pipelines
- Azure Boards
- Azure Test Plans
- Azure Artifacts

Azure Pipelines

Azure Pipelines simplifica la integración continua y la entrega continua en el proceso de desarrollo de aplicaciones. Puede comenzar desde la etapa de origen con el código existente en GitHub o en contenedores locales. Azure Repos puede mantener un repositorio central y los pipelines de Azure mantienen canalizaciones de compilación y lanzamiento para el proyecto determinado. El proceso de CI/CD de Azure DevOps es un proceso crucial con todos los servicios de desarrollo necesarios.

Además de la integración continua y la implementación continua con Azure DevOps, estas canalizaciones se usan para construir flujos de trabajo de compilación, implementación y prueba que se usan principalmente en pruebas continuas (CT). Esto prueba los cambios en una rutina rápida y escalable.

Ventajas de Azure Pipelines:
Azure Pipelines puede tener varios factores y, en la práctica de CI/CD de Azure DevOps, ofrece varias ventajas:

- Sistemas de control de versiones: tener el código en un sistema de control de versiones es el primer paso para crear una canalización de Azure CI/CD. Puede administrar su código fuente en GitHub, Bitbucket, Subversion o cualquier otro

repositorio de Git. También es compatible con Team Foundation Version Control (TFVC).
- Lenguajes de programación y tipos de aplicaciones: puede usar diferentes lenguajes con canalizaciones de Azure como Java, Ruby, C, C++, Python, PHP, Go y JavaScript.
- Destinos de implementación: las aplicaciones con pipelines de Azure se pueden implementar en varios entornos de destino. Esto incluye máquinas virtuales, contenedores o cualquier plataforma local o en la nube.
- Precios: es gratis para proyectos públicos. Pero, para proyectos privados, puede ejecutar hasta 1800 minutos de pipelines gratis por mes.

En Google Cloud Platform

Google Cloud Platform (GCP) ofrece una serie de herramientas y servicios para implementar integración y entrega continua (CI/CD) en proyectos de software. Algunas opciones populares incluyen:

Cloud Build

Es un servicio de compilación en la nube que permite automatizar el proceso de compilación y prueba de proyectos de software. Se integra con una amplia gama de herramientas y tecnologías y se puede usar para implementar código directamente a Google Kubernetes Engine (GKE) o a cualquier otro destino.

Cloud Code

Es un conjunto de herramientas y extensiones de integración y entrega continua (CI/CD) para el entorno de desarrollo de integración continua (IDE) de Google Cloud. Se integra con Cloud Build y otros servicios de GCP y permite a los desarrolladores automatizar su flujo de trabajo de CI/CD directamente desde su IDE.

Cloud Deployment Manager

Es un servicio de implementación en la nube que permite a los desarrolladores implementar y gestionar aplicaciones y servicios en GCP de forma automatizada. Se puede usar para implementar aplicaciones directamente a GKE o a cualquier otro destino.

Además, GCP ofrece una amplia gama de herramientas y servicios para el monitoreo, la depuración y la optimización de aplicaciones y servicios en producción, como Stackdriver y Cloud Monitoring. Estas herramientas se pueden usar para garantizar la estabilidad y el rendimiento de las aplicaciones implementadas mediante CI/CD.

Implementación

La implementación final se realizó sobre AWS por una serie de factores, entre el costo comparado con otras alternativas en la nube (ver sección costos), además de tener una mejor afinidad con AWS a la vez de que sus servicios se adaptan mejor a lo que necesita este proyecto puntualmente ya que solo requería implementar 2 servicios fuera de los provistos y administrados por Amazon, mientras que en Azure y GCP requiere levantar al menos el stack ELK completo con docker, lo que aumenta la complejidad y los costos.
Los servicios a utilizar son los siguientes:

Servicio	Alternativa	AWS
Container Service	Docker	AWS ECS
Cola de mensajes	RabbitMQ	AWS MQ
Service Discovery	Hashicorp Consul	AWS Cloud Map
SQL Database	MySQL	AWS RDS o AWS Aurora
NoSQL Database	MongoDB	AWS DocumentDB
Data Collector	Logstash	AWS Cloudwatch
Data Indexer	ElasticSearch	AWS OpenSearch
Data Visualizer	Kibana	AWS OpenSearch Dashboards
Load Balancing	Nginx LB	AWS Application Load Balancer
Certificate Issue	Let’s Encrypt Certbot	AWS Certificate Manager
Firewall	NGINX ModSecurity WAF	AWS WAF on ALB
Almacenamiento		AWS S3 o AWS EFS o AWS EBS
Registro de contenedores	DockerHub	AWS ECR
Servicio DNS	Cloudflare	AWS Route53
Content Distribution Network	Cloudflare CDN	AWS CloudFront

A continuación, se presentan los detalles de la implementación:

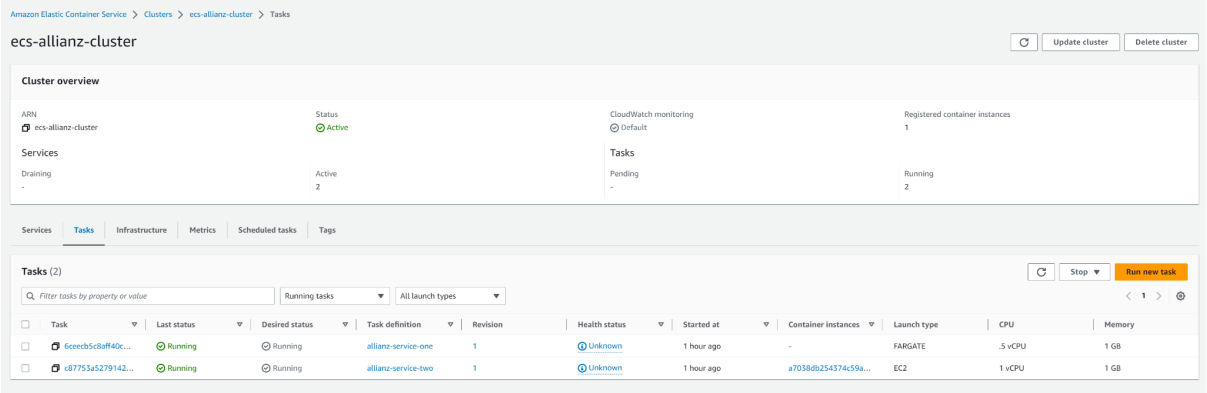
Container Service: AWS ECS

Se implementó los dos servicios de la infraestructura original, “service-one” y “service-two”, en AWS ECS:

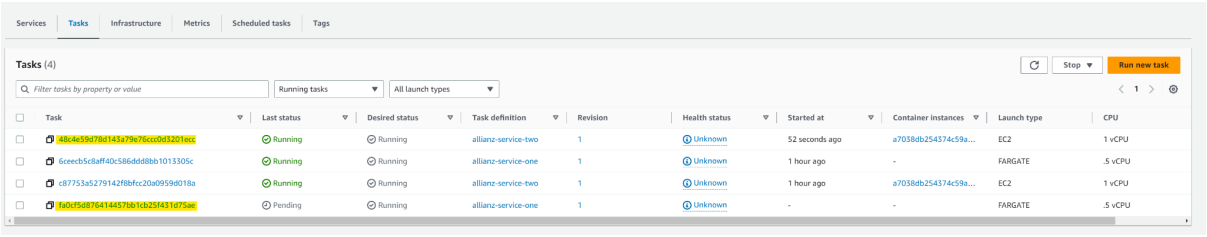
- Service-one se implementó utilizando ECS con EC2, lo que permite que ECS administre el container y su estado dentro de una instancia EC2
- Service-two se implementó utilizando ECS con Fargate, lo que delega en AWS la administración tanto del container y su estado como de la infraestructura sobre la que corre.

Se optó por esta implementación ya que la alternativa de instalar Docker en una instancia EC2 manualmente y correr los containers requiere varias configuraciones manuales mientras que ECS se encarga de toda la administración de contenedores, lo que significa que no es necesario preocuparse por aspectos como el aprovisionamiento de servidores, la orquestación de contenedores, el equilibrio de carga y la escalabilidad. Esto ahorra tiempo y esfuerzo.

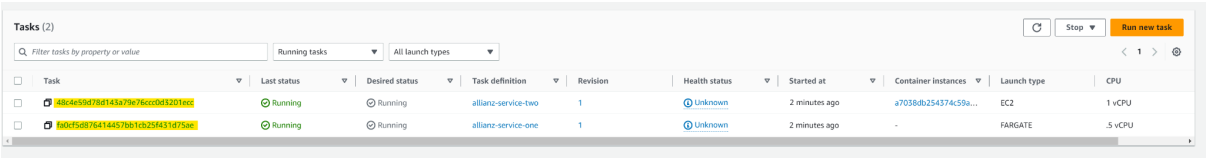
Además, ECS implementa los deployments green/blue, vistos anteriormente. A continuación un ejemplo de este tipo de despliegues en el cluster ECS:
Inicialmente las 2 tareas se encuentran corriendo con normalidad:



Al realizarse un cambio en el código, luego de completarse el proceso de build y push de la imagen, se ejecuta un nuevo despliegue dentro del cluster con las nuevas imágenes, esto es, se crean instancias nuevas de los servicios corriendo paralelamente a los originales.



Luego de que los health checks de los nuevos servicios responden correctamente, indicando que se encuentran listos para recibir peticiones, ECS detiene las instancias anteriores de los servicios, quedando solamente las nuevas versiones



Cola de mensajes: AWS MQ

Amazon provee un servicio completamente administrado compatible con diferentes servicios de cola de mensajes, entre ellos RabbitMQ, con el cual la aplicación original está desarrollada para utilizar, este servicio autoadministrado permite descansar en que AWS administra los recursos para mantener el servicio funcionando correctamente.

Service Discovery: Consul

La aplicación tiene una fuerte integración con Hashicorp Consul, por lo que modificarla para tener integración con AWS Cloud Map requería un importante esfuerzo de desarrollo para modificar y testear el código, se optó por correr un cluster de Consul de 3 nodos en una instancia EC2, Consul es muy eficiente en su consumo de recursos (≈ 50MB de consumo de memoria RAM por nodo), por lo que fue sencillo implementar el cluster en una pequeña instancia EC2 tipo “t2.nano” (1 CPU, 512 MB de RAM). Fuera de levantar el container se

expone el puerto 8500/tcp que es el utilizado por Consul para que las aplicaciones se registren y el servicio quedó funcionando correctamente. A continuación se muestra el archivo “user_data_consul.tpl”, que es el script bash que corre al iniciarse la instancia EC2, este se encarga de actualizar la librería de repositorios e instalar curl, el cual permite luego descargar el script “get-docker.sh” que instala el servicio de docker. Las siguientes líneas definen una red de tipo bridge que utilizaran los 3 containers del cluster para comunicarse, seguidas de 3 lineas de comandos “docker run” que inician los 3 containers el primero, “consul”, inicia como el maestro del cluster y los siguientes 2, “consul2” y “consul3”, se inician configurándose para unirse al maestro.

```
#!/bin/bash

apt-get update
apt-get install curl -y
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh

docker network create -d bridge backend

docker run -d --restart unless-stopped --name=consul --network="backend"
--hostname=consul -p 8500:8500 -p 8600:8600 consul:1.12.2 consul agent -server
-client 0.0.0.0 -ui -bootstrap-expect=3 -data-dir=/consul/data
-retry-join=consul2 -retry-join=consul3 -datacenter=blr

docker run -d --restart unless-stopped --name=consul2 --network="backend"
--hostname=consul2 --expose=8500 --expose=8600 consul:1.12.2 consul agent
-server -data-dir=/consul/data -retry-join=consul -retry-join=consul3
-datacenter=blr

docker run -d --restart unless-stopped --name=consul3 --network="backend"
--hostname=consul3 --expose=8500 --expose=8600 consul:1.12.2 consul agent
-server -data-dir=/consul/data -retry-join=consul -retry-join=consul2
-datacenter=blr
```

SQL Database: RDS MySQL

Para el caso de una base de datos relacional se optó por RDS MySQL, que es un servicio administrado de base de datos de AWS que corre un DBMS de MySQL 8 y no requiere realizar adaptaciones en el los microservicios para que se conecten correctamente, dejando en manos del servicio de RDS el mantenimiento de la base de datos. Se optó por RDS en vez de Aurora por las siguientes razones:

- Coste: RDS MySQL suele ser más económico que Aurora MySQL, lo que lo convierte en una buena opción para aquellos usuarios que tienen un presupuesto más limitado.
- Compatibilidad con MySQL: RDS MySQL es una base de datos MySQL "estándar", lo que significa que es compatible con todas las aplicaciones que se ejecutan en una base de datos MySQL. Aurora MySQL, por otro lado, tiene algunas características propietarias que no están presentes en MySQL, lo que podría causar problemas de compatibilidad en algunos casos.

- Flexibilidad: RDS MySQL ofrece más opciones de configuración que Aurora MySQL, lo que permite a los usuarios personalizar la configuración de la base de datos según necesidades específicas.

La estructura de base de datos es bastante simple, dado que almacena su propio UUID y el generado por el otro microservicio:

```
+---------------+--------------------------------------+
| original_name | original_value                       |
+---------------+--------------------------------------+
| service-one   | 77a9d62f-ad50-4fa3-9feb-33248709436e |
| service-two   | f9133b60-2339-4fbd-a3df-c75a6796ccc9 |
+---------------+--------------------------------------+
```

NoSQL Database: DocumentDB

Para la alternativa NoSQL se optó por AWS DocumentDB, que es un servicio que corre un motor de base de datos no relacional MongoDB completamente administrado, simplificando las tareas de instalación y mantenimiento comparado con instalar mongoDB en una instancia Linux o con Docker. La única modificación que se necesitó hacer fue modificar un parámetro en el string de conexión con la base de datos, agregando el la opción "retryWrites=false" ya que DocumentDB no es compatible con esta opción[3], luego de este ajuste el microservicio "service-one" se pudo conectar sin problemas a la base de datos.

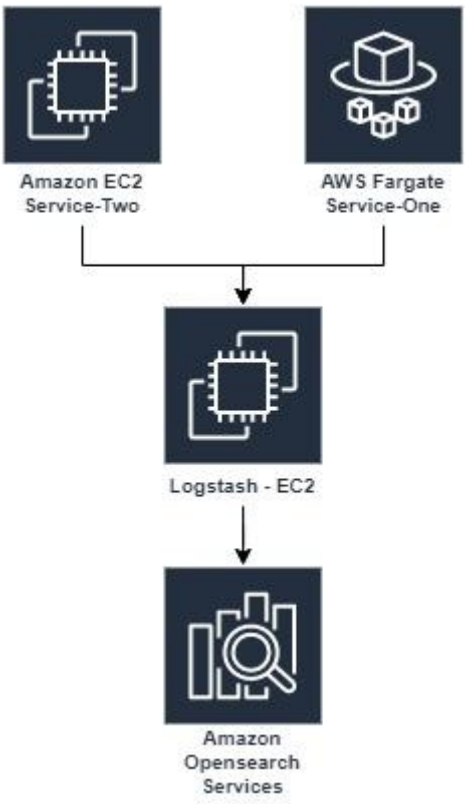
```
  data:
    mongodb.uri:
mongodb://${MONGO_USER}:${MONGO_PASS}@${MONGO_HOST}:27017/db?retryW
rites=false
```

Recopilación de logs

La infraestructura inicial utiliza el stack ELK (Elastic-Logstash-Kibana) para la recopilación, indexado y visualización de logs y métricas. Si bien era una alternativa viable levantar el stack completo de los 3 servicios en una instancia EC2 con docker, la pila ELK suele requerir configuraciones y ajustes en archivos ".yml" para llegar al funcionamiento deseado de cada uno de sus componentes, lo cual aunque funcionara en el primer intento, puede traer problemas a futuro para aplicar actualizaciones y mejoras, o incluso a la hora de resolver algún problema o inestabilidad que surja durante el funcionamiento, a la vez que la pila corre sobre Java y suele ser exigente en recursos del host, por lo que requeriría constante monitoreo de la instancia sobre la que corren los servicios o se podrían llegar a detener por falta de memoria o CPU.
Para implementar la solución se dividió el stack en 2 partes, una consta de ElasticSearch y Kibana, que se optó por una solución open source que es un fork de Elastic ofrecida por

[3] Documentación en el sitio de AWS sobre DocumentDB y compatibilidad con MongoDB: AWS Documentation

AWS llamada OpenSearch (ElasticSeach) y Opensearch Dashboards (Kibana), se implementó estos 2 servicios de manera administrada en AWS, quien se encarga de administrar la salud del servicio y aplicar parches y actualizaciones, mientras que la segunda parte, Logstash, al ser poco exigente en recursos, corre en un container en una pequeña instancia EC2, luego a los microservicios se les pasa como variable de entorno la IP y puerto de esta instancia para que puedan hacer el logshipping correctamente, el container de Logstash recibe como parámetro la URL y credenciales del cluster de OpenSearch al cual le envía los logs mediante HTTPS.



Load Balancing: AWS Application Load Balancer

La decisión de qué LB utilizar fue sencilla, si bien un balanceador con NGINX es relativamente fácil de implementar y muy eficiente en el utilización de recursos, es poco dinámico a la hora de apuntar hacia containers administrados por ECS, cuyas IP son dinámicas, a la vez que ECS, al ser un servicio de AWS, tiene una perfecta y sencilla integración con Application Load Balancers y Target Groups, reduciendo sustancialmente el tiempo de implementación y la complejidad de mantenimiento si lo comparamos con un LB NGINX.
La integración de ECS-ALB-Target Groups es tan buena que, cuando se define el target group al que el balanceador apunta, se le definen health checks, que son básicamente tests para verificar que la aplicación está corriendo, en este se define un puerto y un rango de códigos HTTP que son considerados “healthy”, por lo general código HTTP 200. Si la aplicación no responde con un código esperado por un número definido de intentos, se considera “unhealthy” y se envía un mensaje al servicio de ECS para que detenga el container y cree uno nuevo, en un intento de recuperar el estado operativo del servicio.

Certificate Issue: AWS ACM

Basados en la decisión de balanceador se procede a definir cómo será el proceso para obtener certificados TLS, ya que las alternativas planteadas tienen una fuerte integración con el Balanceador además del servicio que administra el DNS. El servicio de Let's Encrypt permite instalarse como plugin de NGINX y obtener automáticamente certificados para los dominios que el LB tiene definidos, tiene varios métodos pero el más común es utilizando una dirección del web server del tipo "http://<DOMAIN>/.well-known/acme-challenge/<TOKEN>" para hacer la verificación del dominio, este caso se llama "HTTP-01 challenge"[4].
El caso de Amazon Certificates Manager (ACM) tiene una integración tanto con Route53 como con ALB, ya que utiliza el primero para crear automáticamente registros DNS del tipo CNAME para verificar el dominio (DNS Challenge), y una vez obtenido el certificado se lo puede integrar sencillamente a ALB para trabajar con HTTPS en cada endpoint deseado. Por los beneficios que provee, se optó por ACM, además de que ninguno de los servicios tiene un costo por crear certificados.

Firewall: WAF on ALB

La implementación de una barrera de seguridad para la aplicación requirió considerar tanto la complejidad de implementación como la de integración con los demás servicios, como el LB definido fue el ALB de AWS, la implementación más sencilla era utilizar las reglas de Web Application Firewall que AWS ya tiene predefinidas y si se desea se puede crear reglas personalizadas. La alternativa de utilizar ModSec requiere un profundo conocimiento de networking, protocolos, del tráfico que iba a manejar el balanceador y demás, es muy sencillo que una regla genere problemas involuntariamente y altamente complejo y consumidor de tiempo intentar resolverlo, por lo que se optó la integrar reglas WAF al ALB.

Las reglas administradas por AWS que se decidió utilizar son las siguientes:

- AWSManagedRulesCommonRuleSet: El grupo de reglas del conjunto de reglas básicas (CRS) contiene reglas que son generalmente aplicables a las aplicaciones web. Esto proporciona protección contra la explotación de una amplia gama de vulnerabilidades, incluyendo algunas de las vulnerabilidades de alto riesgo y comúnmente encontradas descritas en publicaciones de OWASP como el OWASP Top 10. Se recomienda la implementación de este grupo de reglas para cualquier caso de uso de AWS WAF.
- AWSManagedRulesAmazonIpReputationList: El grupo de reglas de la lista de reputación de IP de Amazon contiene reglas que se basan en la inteligencia de amenazas internas de Amazon. Esto es útil si desea bloquear direcciones IP generalmente asociadas con bots u otras amenazas. El bloqueo de estas direcciones IP puede ayudar a mitigar bots y reducir el riesgo de que un actor malintencionado descubra una aplicación vulnerable.
- AWSManagedRulesAnonymousIpList: El grupo de reglas de la lista de IP anónimas contiene reglas para bloquear solicitudes de servicios que permiten la obfuscación de la identidad del usuario. Estos incluyen solicitudes de VPN, proxies, nodos Tor y proveedores de alojamiento. Este grupo de reglas es útil si desea filtrar a los

[4] Documentación de Lets Encrypt sobre los métodos de verificación de dominio: Let's Encrypt Docs

usuarios que intentan ocultar su identidad en su aplicación. El bloqueo de las direcciones IP de estos servicios puede ayudar a mitigar bots y la evasión de restricciones geográficas.

Se consideraron otras alternativas de grupos de reglas de AWS como:

- AWSManagedRulesSQLiRuleSet: El grupo de reglas de la base de datos SQL contiene reglas para bloquear patrones de solicitud asociados con la explotación de bases de datos SQL, como los ataques de inyección SQL. Esto puede ayudar a prevenir la inyección remota de consultas no autorizadas. Evalúe este grupo de reglas para su uso si su aplicación interactúa con una base de datos SQL.
- AWSManagedRulesKnownBadInputsRuleSet: El grupo de reglas de entradas sospechosas conocidas contiene reglas para bloquear patrones de solicitud que se sabe que son inválidos y están asociados con la explotación o el descubrimiento de vulnerabilidades. Esto puede ayudar a reducir el riesgo de que un actor malintencionado descubra una aplicación vulnerable.

Dado que la aplicación solo devuelve información en respuesta a solicitudes y no procesa ninguna entrada a través de sus endpoints, en este caso no sería útil implementarlos.

Almacenamiento

Si bien la aplicación es stateless (no requiere almacenar archivos para funcionar) se utilizó el servicio de AWS Simple Storage Service (S3) para la implementación del frontend estático, donde CloudFront obtiene los archivos del sitio estático desde el bucket S3. Fuera de esto, los servicios de bases de datos DocumentDB y RDS MySQL usan almacenamientos del tipo EBS (Elastic Block Service), que el tamaño es administrado por cada servicio según se necesite.

Servicio DNS: Route53

Para la administración del DNS se optó por Route53 (R53) de AWS, que como se mencionó antes tiene una gran implementación con los demás servicios de AWS, permitiendo por ejemplo, crear automáticamente los registros DNS para validar el certificado de ACM para HTTPS, o crear los registros para apuntar a balanceadores de carga.

Servicio CDN: AWS CloudFront

La elección de CloudFront como servicio de distribución de contenidos se basó en varios factores. En primer lugar, se valoró la capacidad de CloudFront para integrarse fácilmente con el ecosistema de AWS, lo que simplifica la gestión de recursos en la nube. En segundo lugar, se tuvo en cuenta su capacidad para exponer archivos almacenados en un bucket S3, lo que permite una distribución de contenidos escalable y eficiente junto al bajo costo y la amplia red de Edge Locations de CloudFront en todo el mundo.

CI/CD

La implementación de los pipelines para desplegar el código se realizó en GitHub Actions por la sencillez de configuración e integración con GitHub. La configuración se realiza en archivos “yml”, a continuación se proporciona una descripción detallada del archivo necesario para compilar y cargar la imagen de Docker de "service-one" en AWS ECR.:

```yaml
name: Deploy to ECR
on: workflow_dispatch
jobs:
  build:
    name: Build Image
    runs-on: ubuntu-latest
    steps:
    - name: Check out code
      uses: actions/checkout@v2
    - name: Configure AWS credentials
      uses: aws-actions/configure-aws-credentials@v1
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: us-east-1
    - name: Login to Amazon ECR
      id: login-ecr
      uses: aws-actions/amazon-ecr-login@v1
    - name: Build, tag, and push image to Amazon ECR
      env:
        ECR_REGISTRY: ${{ secrets.ECR_REGISTRY }}
        ECR_REPOSITORY: service-one
        IMAGE_TAG: ${{github.run_number}}
      run: |
        docker build -t $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG .
        docker tag $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG
$ECR_REGISTRY/$ECR_REPOSITORY:latest
        docker push $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG
        docker push $ECR_REGISTRY/$ECR_REPOSITORY:latest
    - name: Update ECS deployment with new ECR image
      run: aws ecs update-service --force-new-deployment --service
allianz-service-one-service --cluster ecs-allianz-cluster
```

Configuración del pipeline en GitHub Actions para hacer build y push de la imagen de docker a Amazon ECR

El archivo define los pasos en el siguiente orden:

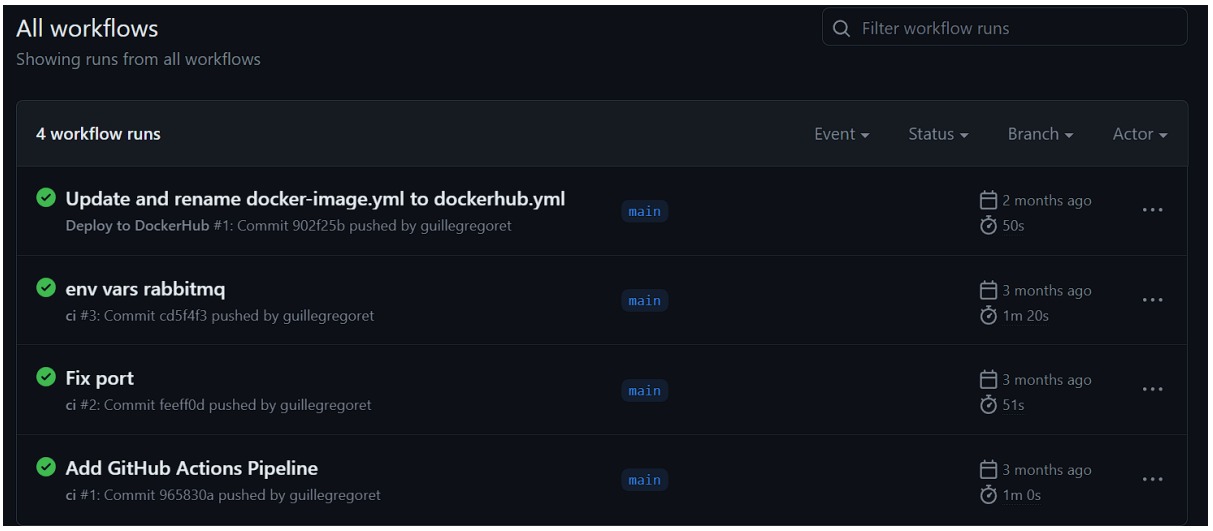
1. Checkout code: Realiza un checkout del código dentro de la instancia temporal que ejecuta el pipeline.
2. Configure AWS credentials: Configura localmente las credenciales para acceder a los recursos dentro de la cuenta de AWS destino.
3. Login to Amazon ECR: Accede al servicio de Elastic Container Repository en AWS.
4. Build, tag, and push image to Amazon ECR: corre los comandos de “docker build”, “docker tag” y “docker push”
5. Update ECS deployment with new ECR image: Ejecuta un comando por la cli de AWS mediante el cual ECS (Elastic Container Service) lanza un nuevo deploy de container del servicio “allianz-service-one-service” dentro del cluster “ecs-allianz-cluster”, lo que en resumen hará que se cree un container con la imagen recientemente pusheada al repositorio y cuando esta empiece a responder, detendrá la imagen anterior.

De manera similar, en el siguiente archivo se muestra como se hace el proceso de build y push de una imagen de Docker a DockerHub:

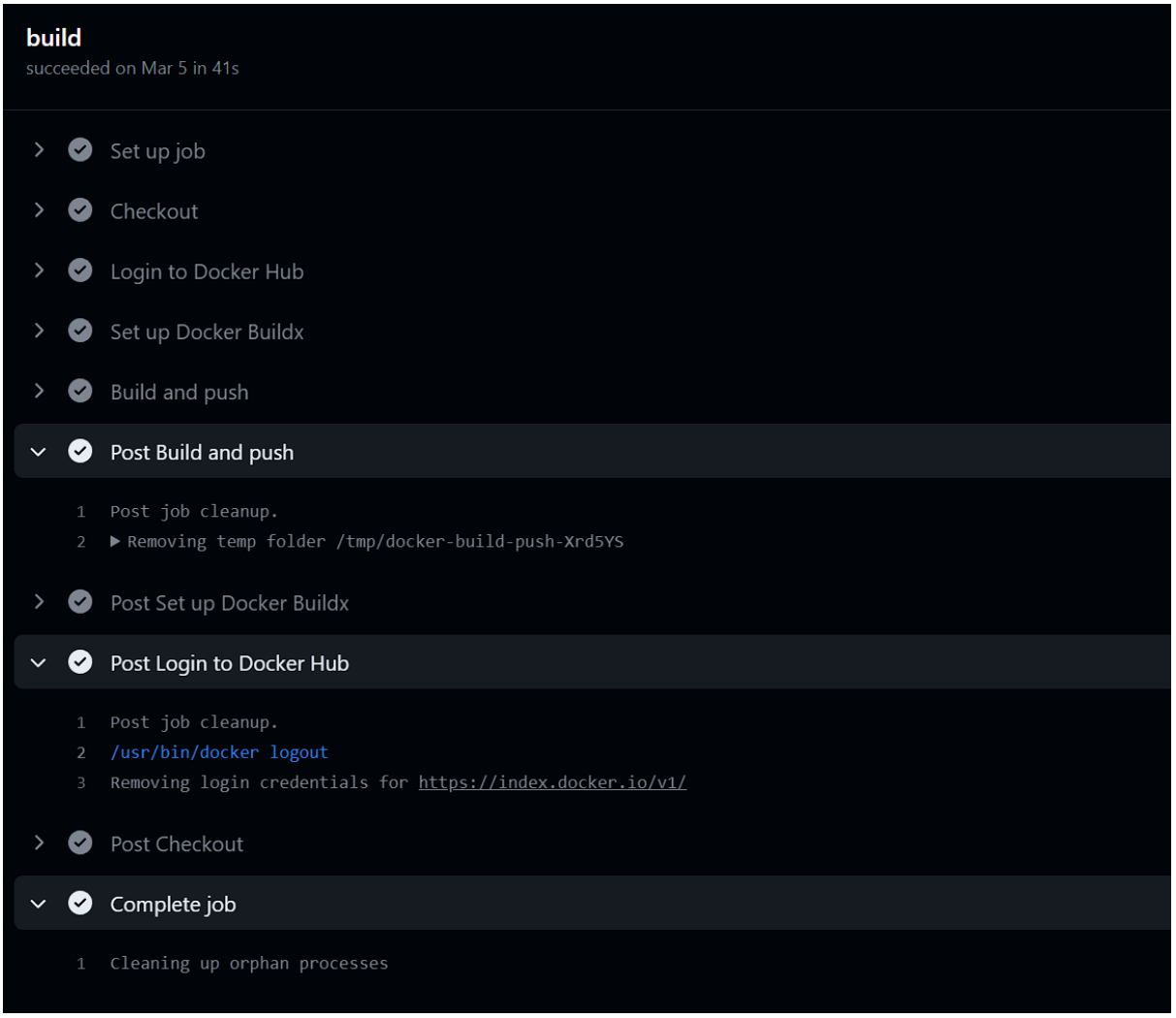
```yaml
name: Deploy to DockerHub
'on':
  push:
    branches:
      - main
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3
      - name: Login to Docker Hub
        uses: docker/login-action@v2
        with:
          username: '${{ secrets.DOCKERHUB_USER }}'
          password: '${{ secrets.DOCKERHUB_PASSWORD }}'
      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2
      - name: Build and push
        uses: docker/build-push-action@v4
        with:
          context: .
          file: ./Dockerfile
          push: true
          tags: '${{ secrets.DOCKERHUB_USER
}}/allianz-serviceone:latest'
```

Recurso en GitHub: dockerhub.yml

En ambos casos el proceso de deploy inicia automáticamente al hacer un push a la branch indicada, en el caso de GitHub Actions se tiene una pestaña en el repositorio dedicada a los workflows de CI/CD:



Dentro de cada proceso se puede obtener un detalle de cada paso ejecutado en el proceso de ejecución del pipeline:



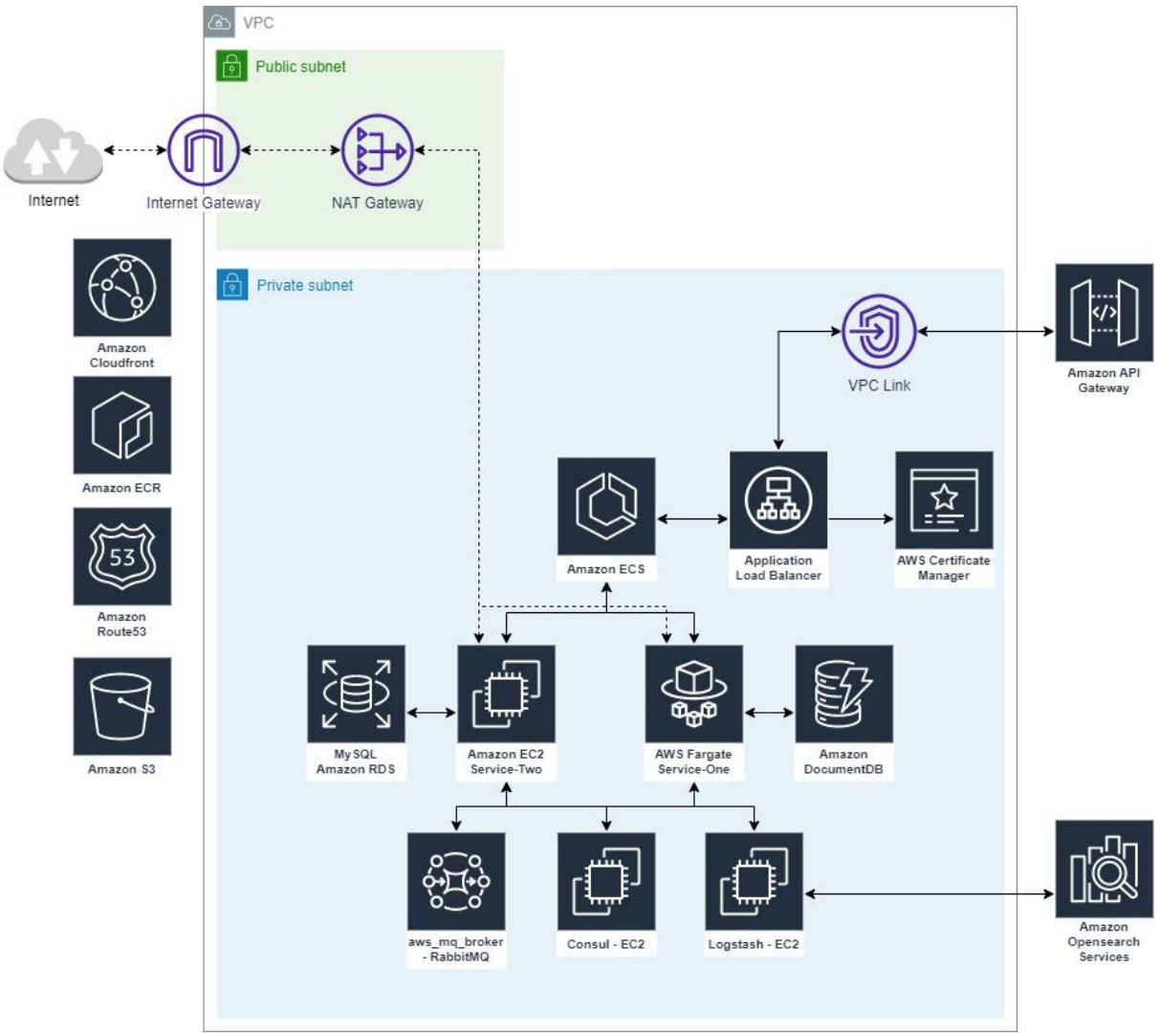
Como se habló en puntos anteriores, la Integración Continua (CI) se refiere a la práctica de integrar el código de los desarrolladores en un repositorio central de manera frecuente, lo que permite detectar errores de manera temprana. La Entrega Continua (CD) se refiere al proceso de automatizar la entrega del código a través de diferentes entornos, como de desarrollo a prueba o de prueba a producción.

La combinación de estos dos procesos, CI/CD, puede ayudar a las empresas a ahorrar tiempo en el despliegue de aplicaciones de varias maneras:

- Automatización de tareas: El uso de procesos de CI/CD permite automatizar muchas de las tareas relacionadas con el despliegue de aplicaciones, como la compilación, las pruebas y la implementación. Esto puede ayudar a reducir el tiempo que se necesita para implementar una nueva versión de la aplicación, ya que las tareas se realizan automáticamente en lugar de tener que ser realizadas manualmente por un equipo de desarrolladores.
- Mayor eficiencia: Los procesos de CI/CD pueden ayudar a mejorar la eficiencia del equipo de desarrollo, ya que permiten a los desarrolladores centrarse en escribir código en lugar de tener que realizar tareas manuales relacionadas con el despliegue de aplicaciones. Esto puede ayudar a reducir el tiempo que se necesita para implementar una nueva versión de la aplicación.
- Mejora en la calidad del software: Al automatizar las pruebas y la implementación, los procesos de CI/CD pueden ayudar a mejorar la calidad del software. Esto se debe a que las pruebas se realizan automáticamente después de cada integración, lo que ayuda a detectar errores y problemas de manera temprana.

Topología final

A continuación se presenta una descripción simplificada de los servicios utilizados, así como las interacciones existentes entre ellos.


VPC
Public subnet
Internet
Internet Gateway
NAT Gateway
Private subnet
Amazon Cloudfront
Amazon ECR
Amazon Route53
Amazon S3
VPC Link
Amazon API Gateway
Amazon ECS
Application Load Balancer
AWS Certificate Manager
MySQL Amazon RDS
Amazon EC2 Service-Two
AWS Fargate Service-One
Amazon DocumentDB
aws_mq_broker - RabbitMQ
Consul - EC2
Logstash - EC2
Amazon Opensearch Services

Costos

Se estimó la infraestructura propuesta en AWS, Azure y Google Cloud:

Servicio	Alternativa	GCP	Azure	AWS
Container Service	Docker	GCP Cloud Run	Azure Container Instances	AWS ECS
Cola de mensajes	RabbitMQ	GCP Cloud Pub/Sub o Cloud Tasks	Azure Service Bus, Azure Storage Queues	AWS MQ o SQS
Service Discovery	Hashicorp Consul	Consul on Docker	Consul on Docker	Consul on Docker
API Gateway	Netflix Zuul	GCP API Gateway	Azure API Management	AWS API Gateway
Database	MySQL	GCP Cloud SQL	Azure Database for MySQL	AWS RDS o AWS Aurora
	MongoDB	GCP Datastore o Cloud Bigtable	Azure Cosmos DB	AWS DocumentDB
Data Collector	Logstash	ELK on Docker	ELK on Docker	Logstash on Docker
Data Indexer	ElasticSearch			AWS OpenSearch
Data Visualizer	Kibana			AWS OpenSearch Dashboards
Load Balancing	Nginx LB	GCP Cloud Load Balancing	Azure Load Balancer	AWS Application Load Balancer
Certificate Issue	Let's Encrypt Certbot	GCP Certificate Authority Service	Azure App Service Managed Certificate	AWS Certificate Manager
Firewall	NGINX ModSecurity WAF	Google Cloud Armor	Azure WAF, Azure Front Door	AWS WAF on ALB
Almacenamiento		GCP Cloud Storage o Persistent Disk o Filestore	Azure Blob Storage o Disk Storage o Azure Files	AWS S3 o AWS EFS o AWS EBS
Registro de contenedores	DockerHub	GCP Container Registry	Azure Container Registry	AWS ECR
IAM	KeyCloak	Firebase Authentication	Azure Active Directory Service	AWS Cognito
Servicio DNS	Cloudflare	GCP Cloud DNS	Azure DNS	AWS Route53
Content Distribution Network	Cloudflare CDN	GCP Cloud CDN	Azure CDN	AWS CloudFront

El costo mensual obtenido es el siguiente. Se anexa el cálculo realizado en cada proveedor.

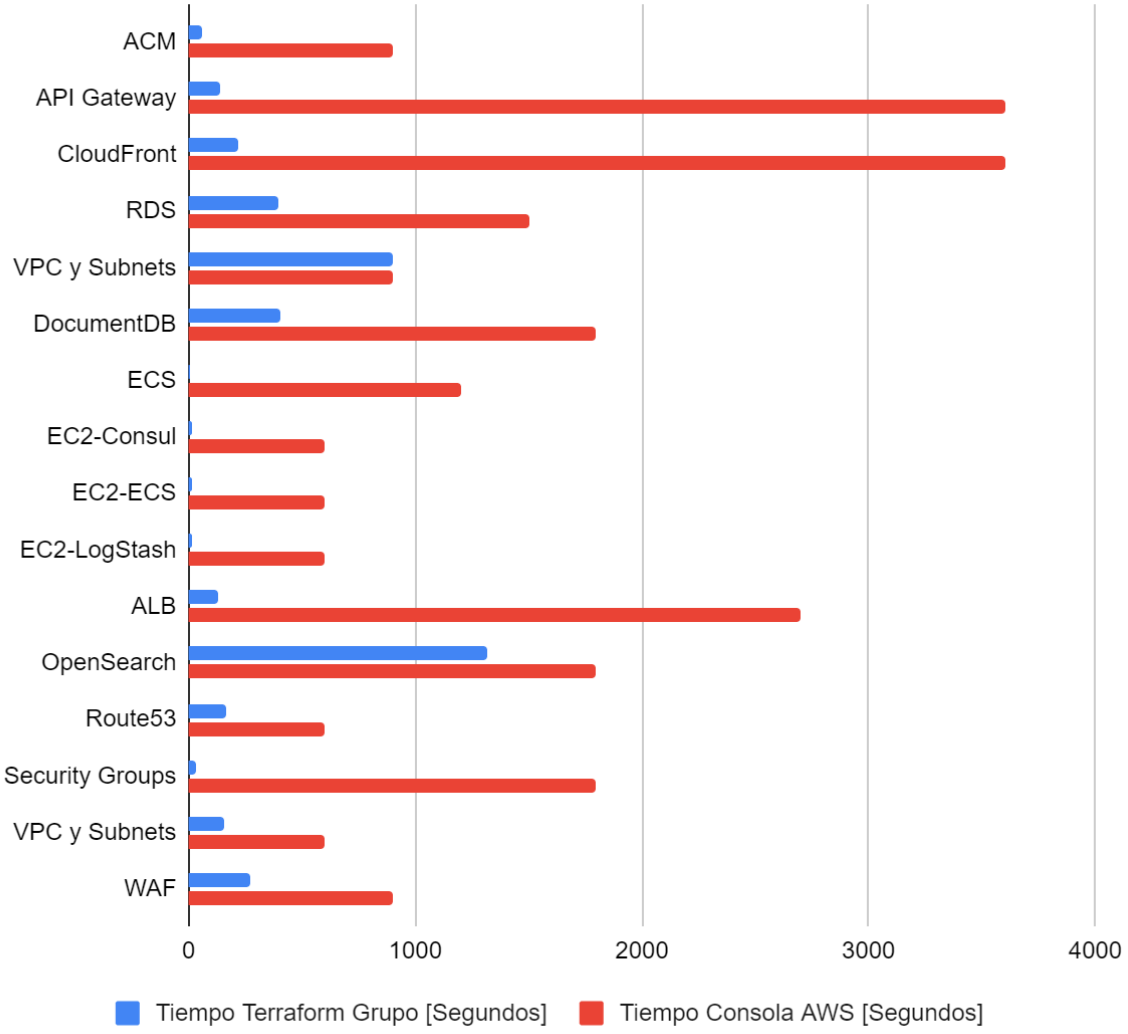
Proveedor	Costo Mensual
AWS	259.83 USD
Azure	546.49 USD
Google Cloud	294.57 USD

Optimización del tiempo al utilizar IaC.

A continuación se presenta el análisis de los tiempos de implementación de la infraestructura, realizado comparando dos enfoques: la implementación manual desde la consola web de AWS y la utilización de Terraform.

Servicio	Recurso en Terraform	Tiempo Terraform [Segundos]	Tiempo Terraform Grupo [Segundos]	Tiempo Consola AWS [Segundos]
ACM	aws_acm_certificate_validation	59	59	900
API Gateway	aws_apigatewayv2_api_mapping	1		
	aws_apigatewayv2_domain_name	5		
	aws_apigatewayv2_integration	1		
	aws_apigatewayv2_route	0		
	aws_apigatewayv2_vpc_link	129	136	3600
CloudFront	aws_cloudfront_distribution	218	218	3600
RDS	aws_db_instance	392	392	1500
VPC y Subnets	aws_db_subnet_group	1	900	900
DocumentDB	aws_docdb_cluster	53		
	aws_docdb_cluster_instance	352		
	aws_docdb_subnet_group	1	406	1800
ECS	aws_ecs_service	4		
	aws_ecs_task_definition	2	6	1200
EC2-Consul	aws_instance_consul	15	15	600
EC2-ECS	aws_instance_ec2	15	15	600
EC2-LogStash	aws_instance_logstash	15	15	600
ALB	aws_lb	124		
	aws_lb_listener	2		
	aws_lb_target_group	5	131	2700
OpenSearch	aws_opensearch_domain	1322	1322	1800
Route53	aws_route53_record	122		
	aws_route53_zone	43	165	600
Security Groups	aws_security_group	15		
	aws_security_group_rule	12		
	security_group_rds	4	31	1800
VPC y Subnets	vpc	154	154	600
WAF	aws_wafv2_web_acl	2		
	aws_wafv2_web_acl_association	271	273	900

En el siguiente gráfico se presenta una comparación de tiempo recurso a recurso, donde se evidencia claramente la ventaja de utilizar Terraform.



Como se observa en las métricas previas, se evidencia una notable reducción en el tiempo de despliegue al utilizar Terraform en comparación con la implementación manual. Además, al utilizar Terraform, los recursos se crean en paralelo al resolver las dependencias entre ellos, a diferencia del método manual donde se crean en serie. Mientras que la implementación manual requirió aproximadamente 380 minutos o 6 horas y media, Terraform sólo necesita alrededor de 26 minutos en promedio para crear los recursos. Aunque es necesario desarrollar el código completo para que Terraform funcione, existe documentación y módulos que permiten implementar conjuntos de recursos de manera fácil y rápida. Además, en el caso de tener que desplegar múltiples entornos idénticos, como dev, staging y producción, el código de un solo entorno se puede utilizar para desplegar todos ellos.

Ejemplos prácticos

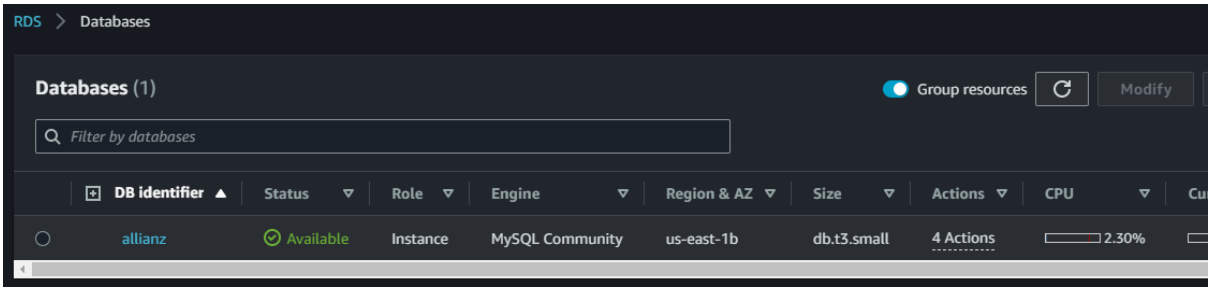
Un ejemplo concreto que ilustra las ventajas de implementar infraestructura como código es el siguiente: Supongamos que el equipo de desarrollo necesita realizar modificaciones en la base de datos, tanto en su capacidad (tamaño de la instancia de cómputo) como en el espacio de almacenamiento. En un escenario tradicional, donde se cuenta con un rol de DevOps o SysOps, se debería seguir un proceso que incluye los siguientes pasos:

1. Creación de un documento detallado con las especificaciones y requisitos de la modificación.
2. Envío del documento a través de múltiples canales de comunicación, como correo electrónico y sistemas de gestión de tickets.
3. Espera a que el documento sea revisado y aprobado por diferentes niveles de jerarquía y departamentos.
4. Asignación del ticket a la persona encargada de ejecutar la tarea, quien puede tener una carga de trabajo considerable y no estar disponible de inmediato.
5. Notificación al equipo de desarrollo de que el ticket ha sido asignado y que deben esperar a que se complete antes de poder continuar con su trabajo.
6. Seguimiento constante del estado del ticket y comunicación adicional con el encargado para obtener actualizaciones sobre el progreso.
7. Finalmente, después de un periodo de tiempo prolongado, la tarea es ejecutada y se notifica al equipo de desarrollo.

Todo este proceso engorroso implica una demora significativa en el avance del proyecto, ya que el equipo de desarrollo queda bloqueado hasta que la tarea de modificación de la base de datos se complete. En contraste, al implementar infraestructura como código, estos pasos son eliminados, permitiendo una ejecución más rápida y eficiente de las modificaciones requeridas. A continuación dos ejemplos:

Cambio de tamaño de instancia

Inicialmente la base de datos en RDS tiene un tamaño “db.t3.small” que cuenta con 2 CPU y 2 GB de RAM, se desea pasar a una instancia tipo “db.t3.medium” que también cuenta con 2 CPU pero con 4 GB de RAM



En el código, dentro del bloque donde se define la base de datos se modifica puntualmente el parámetro `instance_class`:

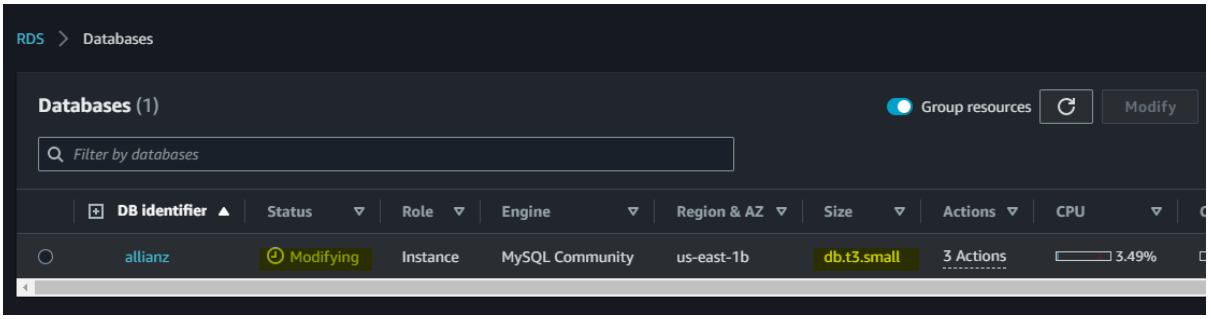
```
24
25  resource "aws_db_instance" "allianz_mysql" {
26    allocated_storage           = 20
27    storage_type                = "gp3"
28    engine                      = "mysql"
29    engine_version              = "8.0.28"
30-   instance_class              = "db.t3.small"
31    identifier                  = var.rds_db_name
32    db_name                     = var.rds_db_name
33    username                    = var.rds_username
34    password                    = var.rds_password
```

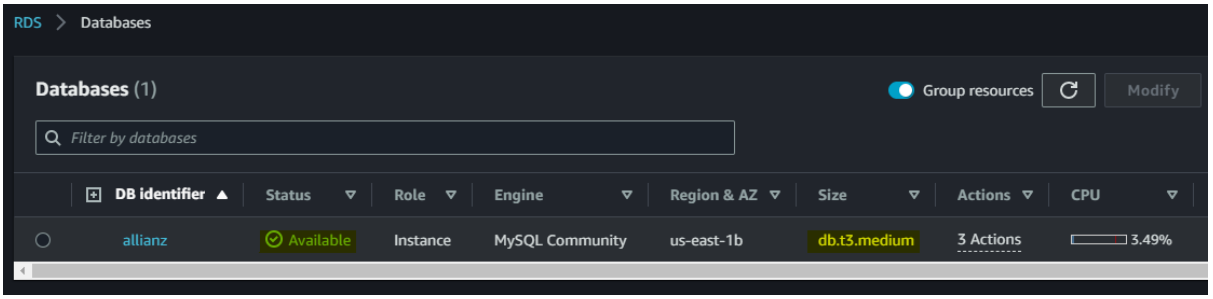
```
24
25  resource "aws_db_instance" "allianz_mysql" {
26    allocated_storage           = 20
27    storage_type                = "gp3"
28    engine                      = "mysql"
29    engine_version              = "8.0.28"
30+   instance_class              = "db.t3.medium"
31    identifier                  = var.rds_db_name
32    db_name                     = var.rds_db_name
33    username                    = var.rds_username
34    password                    = var.rds_password
```

Se ejecuta el código de terraform con terraform apply

```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_db_instance.allianz_mysql: Modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 10s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 20s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 30s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 40s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 50s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 1m0s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 1m10s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 1m20s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 1m30s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 1m40s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 1m50s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 2m0s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 2m10s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 2m20s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 2m30s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 2m40s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 2m50s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 3m0s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 3m10s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 3m20s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 3m30s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 3m40s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 3m50s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 4m0s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 4m10s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 4m20s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 4m30s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 4m40s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 4m50s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 5m0s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 5m10s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y, 5m20s elapsed]
aws_db_instance.allianz_mysql: Modifications complete after 5m30s [id=db-E4EKYR5K3GNSDRXVTTRWL4ON5Y]

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

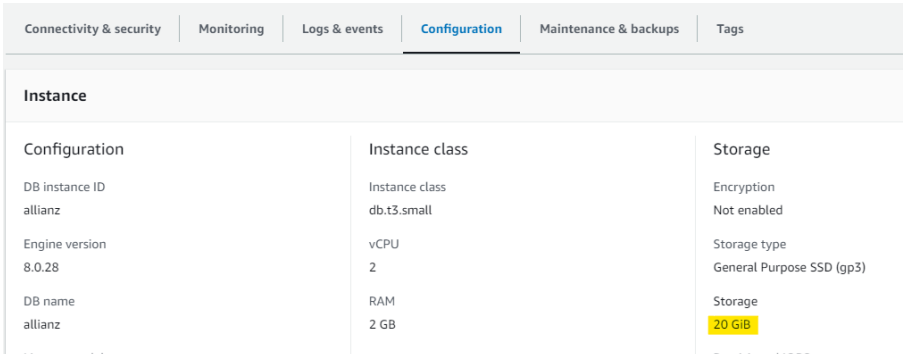
La base de datos cambiará a estado “Modificando” por lo que dure el proceso, luego volverá a estar disponible con el nuevo tamaño.



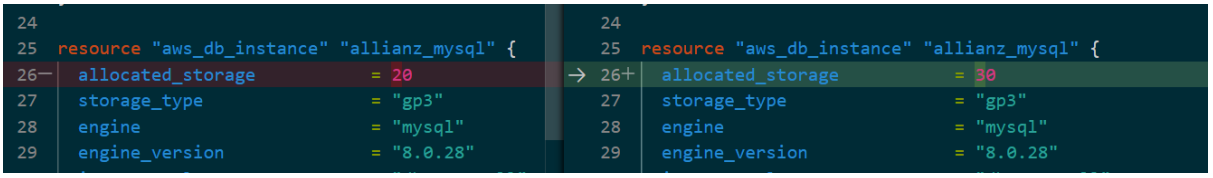


Aumento del tamaño del almacenamiento

Para el caso de expandir el almacenamiento de la base de datos el caso es similar, para el ejemplo se pasa el almacenamiento de 20 GB a 30 GB:



Se modifica el parámetro `allocated_storage`

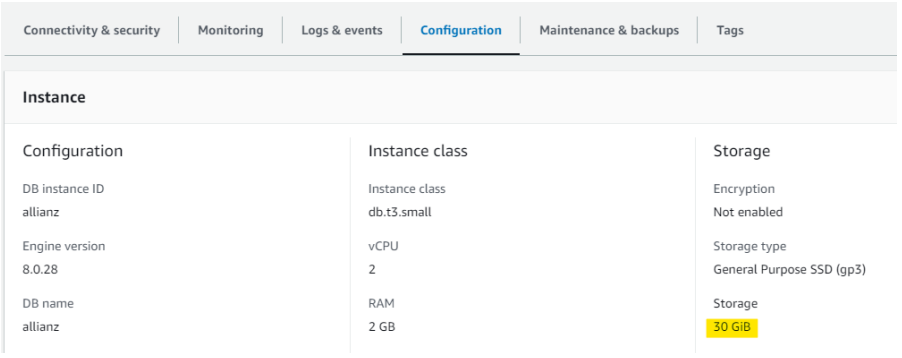


Se corre los cambios en terraform

```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_db_instance.allianz_mysql: Modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 10s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 20s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 30s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 40s elapsed]
```

```
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 50s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 1m0s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 1m10s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 1m20s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 1m30s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 1m40s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 1m50s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 2m0s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 2m10s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 2m20s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 2m30s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 2m40s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 2m50s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 3m0s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 3m10s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 3m20s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 3m30s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 3m40s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 3m50s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 4m0s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 4m10s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 4m20s elapsed]
aws_db_instance.allianz_mysql: Still modifying... [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ, 4m30s elapsed]
aws_db_instance.allianz_mysql: Modifications complete after 4m38s [id=db-35EEHW5BX5W4LUMSOAF6O3UOGQ]

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

Al finalizar se verá reflejado el nuevo tamaño de almacenamiento, en una operación que no demoró más de 5 minutos.



Conclusión

Además de las habilidades técnicas adquiridas durante el desarrollo de este proyecto, también he mejorado mis habilidades de gestión de proyectos.

También he tenido la oportunidad de poner en práctica mis habilidades en la investigación y análisis de problemas, lo que me ha permitido encontrar soluciones eficaces y eficientes a los desafíos técnicos enfrentados en el desarrollo del proyecto.

Además, este proyecto ha sido una oportunidad para mí para aprender y aplicar las mejores prácticas de la industria en términos de seguridad, rendimiento y escalabilidad de la aplicación, la infraestructura diseñada, planteada y desarrollada con terraform levanta un entorno completo y funcional en aproximadamente 25 minutos. He comprendido la importancia de implementar estas prácticas desde el inicio del proyecto para garantizar que la aplicación sea **segura, eficiente y escalable** a medida que crezca el número de usuarios y se agreguen más funcionalidades.

Este proyecto ha sido una experiencia invaluable, no solo en términos de mejora de habilidades técnicas, sino también en términos de habilidades de gestión de proyectos, trabajo en equipo, resolución de problemas y comprensión de las mejores prácticas de la industria. Estoy seguro de que estos conocimientos y habilidades me serán útiles en mi carrera profesional y en cualquier proyecto futuro en el que participe.

Bibliografía

- Migrating to AWS: A Manager's Guide by Jeff Armstrong - 2020
- Terraform: Up & Running, 2nd Edition by Yevgeniy Brikman - 2019
- Navin Sabharwal, Sarvesh Pandey, Piyush Pandey - “Infrastructure-as-Code Automation Using Terraform, Packer, Vault, Nomad and Consul”
- Bases de datos no relacionales
- Redis Explained
- Amazon ECS task definitions - Amazon Elastic Container Service
- What happens in a TLS handshake? | SSL handshake | Cloudflare
- ELB vs. ALB vs. NLB: Choosing the Best AWS Load Balancer for Your Needs - IOD - The Content Engineers
- AWS Managed Rules rule groups list

Anexos

Anexo I: Código de Terraform

waf.tf

```
# WAF and WebACL Rules
resource "aws_wafv2_web_acl_association" "waf_alb" {
resource_arn = aws_lb.public-load-balancer.arn
web_acl_arn = aws_wafv2_web_acl.external.arn
}

resource "aws_wafv2_web_acl" "external" {
name = "ExternalACL"
scope = "REGIONAL"

default_action {
allow {}
}

rule {
name = "AWS-AWSManagedRulesCommonRuleSet"
priority = 1

override_action {
none {}
}

statement {
managed_rule_group_statement {
name = "AWSManagedRulesCommonRuleSet"
vendor_name = "AWS"
}
}

visibility_config {
cloudwatch_metrics_enabled = true
```

```
32 metric_name = "AWS-AWSManagedRulesCommonRuleSet"
33 sampled_requests_enabled = true
34 }
35 }
36
37 rule {
38 name = "AWS-AWSManagedRulesSQLiRuleSet"
39 priority = 2
40
41 override_action {
42 none {}
43 }
44
45 statement {
46 managed_rule_group_statement {
47 name = "AWSManagedRulesSQLiRuleSet"
48 vendor_name = "AWS"
49 }
50 }
51
52 visibility_config {
53 cloudwatch_metrics_enabled = true
54 metric_name = "AWS-AWSManagedRulesSQLiRuleSet"
55 sampled_requests_enabled = true
56 }
57 }
58
59 rule {
60 name = "AWS-AWSManagedRulesKnownBadInputsRuleSet"
61 priority = 3
62
63 override_action {
64 none {}
65 }
66
67 statement {
```

```
68  managed_rule_group_statement {
69  name = "AWSManagedRulesKnownBadInputsRuleSet"
70  vendor_name = "AWS"
71  }
72  }
73
74  visibility_config {
75  cloudwatch_metrics_enabled = true
76  metric_name = "AWS-AWSManagedRulesKnownBadInputsRuleSet"
77  sampled_requests_enabled = true
78  }
79  }
80
81  rule {
82  name = "AWS-AWSManagedRulesAmazonIpReputationList"
83  priority = 4
84
85  override_action {
86  none {}
87  }
88
89  statement {
90  managed_rule_group_statement {
91  name = "AWSManagedRulesAmazonIpReputationList"
92  vendor_name = "AWS"
93  }
94  }
95
96  visibility_config {
97  cloudwatch_metrics_enabled = true
98  metric_name = "AWS-AWSManagedRulesAmazonIpReputationList"
99  sampled_requests_enabled = true
100 }
101 }
102
```

```
rule {
name = "AWS-AWSManagedRulesAnonymousIpList"
priority = 5

override_action {
none {}
}

statement {
managed_rule_group_statement {
name = "AWSManagedRulesAnonymousIpList"
vendor_name = "AWS"
}
}

visibility_config {
cloudwatch_metrics_enabled = true
metric_name = "AWS-AWSManagedRulesAnonymousIpList"
sampled_requests_enabled = true
}
}


visibility_config {
cloudwatch_metrics_enabled = true
metric_name = "ExternalACL"
sampled_requests_enabled = true
}

}
```

variables.tf

```
variable "domain_name" {
type = string
}
variable "project_name" {
```

```
type = string
}
variable "bucket_name" {
type = string
}
variable "rds_username" {
type = string
}
variable "rds_password" {
type = string
}
variable "rds_db_name" {
type = string
}
variable "consul_port" {
type = string
}
variable "docdb_username" {
type = string
}
variable "docdb_password" {
type = string
}
variable "docdb_db" {
type = string
}
variable "opensearch_user" {
type = string
}
variable "opensearch_password" {
type = string
}
variable "opensearch_url" {
type = string
}
variable "logstash_port" {
type = string
```

```
42 }
43 variable "rabbit_host" {
44 type = string
45 }
46 variable "rabbit_user" {
47 type = string
48 }
49 variable "rabbit_pass" {
50 type = string
51 }
52 variable
   "service_one_docker_image" {
53 type = string
54 }
55 variable
   "service_two_docker_image" {
56 type = string
57 }
58 variable "ecs_cluster_name" {
59 type = string
60 }
61 variable "cloudflare_api_token" {
62 type = string
63 }
64 variable "cloudflare_zone_id" {
65 type = string
66 }
67
```

user_data.tpl

```
1 #!/bin/bash
2
3 # Update all packages
4
5 sudo yum update -y
6 sudo yum install -y ecs-init
7 sudo service docker start
```

```
sudo start ecs

#Adding cluster name in ecs config
echo ECS_CLUSTER=${ECS_CLUSTER_NAME} >>
/etc/ecs/ecs.config
cat /etc/ecs/ecs.config | grep "ECS_CLUSTER"
```

user_data_logstash.tpl

```
#!/bin/bash

# Update all packages
apt-get update
apt-get install curl -y
sysctl -w vm.max_map_count=262144
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
docker run -d -p 5001:5001 --restart unless-stopped --name
logstash-allianz -e OS_USER='${OS_USER}' -e OS_PASSWD='${OS_PASSWD}'
-e OS_URL='${OS_URL}' guillegregoret/allianz-logstash:latest

```

user_data_consul.tpl

```
#!/bin/bash

# Update all packages
apt-get update
apt-get install curl -y
hostnamectl set-hostname consul_cluster
echo "Changing Hostname"
hostname "consul_cluster"
echo "consul_cluster" > /etc/hostname
sysctl -w vm.max_map_count=262144
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh

docker network create -d bridge backend
```

```
15 docker run -d --restart unless-stopped --name=consul
   --network="backend" --hostname=consul -p 8500:8500 -p 8600:8600
   consul:1.12.2 consul agent -server -client 0.0.0.0 -ui
   -bootstrap-expect=3 -data-dir=/consul/data -retry-join=consul2
   -retry-join=consul3 -datacenter=blr
16 docker run -d --restart unless-stopped --name=consul2
   --network="backend" --hostname=consul2 --expose=8500 --expose=8600
   consul:1.12.2 consul agent -server -data-dir=/consul/data
   -retry-join=consul -retry-join=consul3 -datacenter=blr
17 docker run -d --restart unless-stopped --name=consul3
   --network="backend" --hostname=consul3 --expose=8500 --expose=8600
   consul:1.12.2 consul agent -server -data-dir=/consul/data
   -retry-join=consul -retry-join=consul2 -datacenter=blr
```

upload_frontend.tf

```
1  resource "aws_s3_bucket_object"
   "frontend_html" {
2  for_each = fileset("frontend/", "*.html")
3  bucket = aws_s3_bucket.root_bucket.id
4  key = each.value
5  source = "frontend/${each.value}"
6  etag = filemd5("frontend/${each.value}")
7  content_type = "text/html"
8  }
9  resource "aws_s3_bucket_object" "frontend_js"
   {
10 for_each = fileset("frontend/", "*.js")
11 bucket = aws_s3_bucket.root_bucket.id
12 key = each.value
13 source = "frontend/${each.value}"
14 etag = filemd5("frontend/${each.value}")
15 content_type = "text/javascript"
16 }
17 resource "aws_s3_bucket_object" "frontend_css"
   {
18 for_each = fileset("frontend/", "*.css")
19 bucket = aws_s3_bucket.root_bucket.id
20 key = each.value
21 source = "frontend/${each.value}"
22 etag = filemd5("frontend/${each.value}")
23 content_type = "text/css"
```

```
}
resource "aws_s3_bucket_object" "favicon" {
for_each = fileset("frontend/", "*.gif")
bucket = aws_s3_bucket.root_bucket.id
key = each.value
source = "frontend/${each.value}"
etag = filemd5("frontend/${each.value}")

}
```

task_definition_service_two.json

```
[
{
"name": "service-two",
"image": "${IMAGE}",
"cpu": 1024,
"memory": 1024,
"environment": [
{"name": "SPRING_PROFILES_ACTIVE", "value": "docker"},
{"name": "CONSUL_HOST", "value": "${CONSUL_HOST}"},
{"name": "CONSUL_PORT", "value": "${CONSUL_PORT}"},
{"name": "RDS_HOST", "value": "${RDS_HOST}"},
{"name": "RDS_DB", "value": "${RDS_DB}"},
{"name": "RDS_USER", "value": "${RDS_USER}"},
{"name": "RDS_PASS", "value": "${RDS_PASS}"},
{"name": "RABBIT_USER", "value": "${RABBIT_USER}"},
{"name": "RABBIT_PASS", "value": "${RABBIT_PASS}"},
{"name": "RABBIT_HOST", "value": "${RABBIT_HOST}"},
{"name": "LOGSTASH_HOST", "value": "${LOGSTASH_HOST}"},
{"name": "LOGSTASH_PORT", "value": "${LOGSTASH_PORT}"}
],
"logConfiguration": {
"logDriver": "awslogs",
"options": {
"awslogs-group": "service-two",
"awslogs-create-group": "true",
```

```
"awslogs-region": "us-east-1",
"awslogs-stream-prefix": "ecs"
}
},
"links": [],
"portMappings": [
{
"hostPort": 0,
"containerPort": 8084,
"protocol": "tcp"
}
],
"essential": true,
"entryPoint": [],
"command": [],
"mountPoints": [],
"volumesFrom": []
}
]
```

task_definition_service_one.json

```
[
{
"name": "service-one",
"image": "${IMAGE}",
"cpu": 512,
"memory": 1024,
"networkMode": "awsvpc",
"environment": [
{"name": "SPRING_PROFILES_ACTIVE", "value": "docker"},
{"name": "CONSUL_HOST", "value": "${CONSUL_HOST}"},
{"name": "CONSUL_PORT", "value": "${CONSUL_PORT}"},
{"name": "MONGO_HOST", "value": "${MONGO_HOST}"},
{"name": "MONGO_DB", "value": "${MONGO_DB}"},
{"name": "MONGO_USER", "value": "${MONGO_USER}"},
{"name": "MONGO_PASS", "value": "${MONGO_PASS}"},
```

```
{"name": "RABBIT_USER", "value": "${RABBIT_USER}"},
{"name": "RABBIT_PASS", "value": "${RABBIT_PASS}"},
{"name": "RABBIT_HOST", "value": "${RABBIT_HOST}"},
{"name": "LOGSTASH_HOST", "value": "${LOGSTASH_HOST}"},
{"name": "LOGSTASH_PORT", "value": "${LOGSTASH_PORT}"}
],
"logConfiguration": {
"logDriver": "awslogs",
"options": {
"awslogs-group": "service-one",
"awslogs-create-group": "true",
"awslogs-region": "us-east-1",
"awslogs-stream-prefix": "ecs"
}
},
"links": [],
"portMappings": [
{
"hostPort": 8082,
"containerPort": 8082,
"protocol": "tcp"
}
],
"essential": true,
"entryPoint": [],
"command": [],
"mountPoints": [],
"volumesFrom": []
}
]
```

s3.tf

```

# S3 Bucket for static frontend

resource "aws_s3_bucket" "root_bucket" {
```

```
5
6  bucket = "${var.bucket_name}-${random_string.random_suffix.result}"
7  #policy = templatefile("templates/s3-policy.json", { bucket =
   "${var.bucket_name}-${random_string.random_suffix.result}" })
8  cors_rule {
9  allowed_headers = ["Authorization", "Content-Length"]
10 allowed_methods = ["GET", "POST"]
11 allowed_origins = ["https://${var.domain_name}"]
12 max_age_seconds = 3000
13 }
14 }
15 resource "aws_s3_bucket_website_configuration" "root_bucket" {
16 bucket = aws_s3_bucket.root_bucket.id
17
18 index_document {
19 suffix = "index.html"
20 }
21
22 error_document {
23 key = "404.html"
24 }
25 }
26
27 resource "aws_s3_bucket_public_access_block" "pab" {
28 bucket = aws_s3_bucket.root_bucket.id
29
30 block_public_acls = false
31 block_public_policy = false
32 ignore_public_acls = false
33 restrict_public_buckets = false
34 }
35
36 resource "aws_s3_bucket_ownership_controls" "ownership-controls" {
37 bucket = aws_s3_bucket.root_bucket.id
38 rule {
39 object_ownership = "BucketOwnerPreferred"
40 }
```

```
}

resource "aws_s3_bucket_acl" "bucket-acl" {
depends_on = [
aws_s3_bucket_public_access_block.pab,
aws_s3_bucket_ownership_controls.ownership-controls,
]

bucket = aws_s3_bucket.root_bucket.id
acl = "public-read"
}

resource "aws_s3_bucket_policy" "public_read_access" {
bucket = aws_s3_bucket.root_bucket.id
policy = <<EOF
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "PublicReadGetObject",
"Effect": "Allow",
"Principal": "*",
"Action": "s3:GetObject",
"Resource":
"arn:aws:s3:::${var.bucket_name}-${random_string.random_suffix.result
}/*"
}
]
}
EOF
depends_on = [aws_s3_bucket.root_bucket]
}

resource "random_string" "random_suffix" {
length = 8
special = false
upper = false
```

```
}
```

rds.tf

```
# RDS
resource "aws_db_subnet_group" "rds-private-subnet" {
name = "rds-private-subnet-group"
subnet_ids = [module.vpc.private_subnets[0],
module.vpc.private_subnets[1]]
}

resource "aws_security_group" "rds-sg" {
name = "my-rds-sg"
vpc_id = module.vpc.vpc_id

}

# Ingress Security Port 3306
resource "aws_security_group_rule" "mysql_inbound_access" {
from_port = 3306
protocol = "tcp"
security_group_id = aws_security_group.rds-sg.id
to_port = 3306
type = "ingress"
cidr_blocks = ["0.0.0.0/0"]
}

resource "aws_db_instance" "mysql" {
allocated_storage = 20
storage_type = "gp3"
engine = "mysql"
engine_version = "8.0.28"
instance_class = "db.t3.medium"
identifier = var.rds_db_name
db_name = var.rds_db_name
username = var.rds_username
password = var.rds_password
```

```
parameter_group_name = "default.mysql8.0"
db_subnet_group_name = aws_db_subnet_group.rds-private-subnet.name

vpc_security_group_ids = ["${aws_security_group.rds-sg.id}"]
allow_major_version_upgrade = true
auto_minor_version_upgrade = true
backup_retention_period = 1
backup_window = "22:00-23:00"
maintenance_window = "Sat:00:00-Sat:03:00"
multi_az = false
skip_final_snapshot = true
apply_immediately = true

blue_green_update {
enabled = true
}
}
```

r53.tf

```
##Route53
resource "aws_route53_zone" "main" {
name = var.domain_name
}

### R53 Records ###
resource "aws_route53_record" "api" {
name =
aws_apigatewayv2_domain_name.apigateway-domain-name.domain_name
type = "A"
zone_id = aws_route53_zone.main.zone_id

alias {
name =
aws_apigatewayv2_domain_name.apigateway-domain-name.domain_name_confi
guration[0].target_domain_name
zone_id =
aws_apigatewayv2_domain_name.apigateway-domain-name.domain_name_confi
guration[0].hosted_zone_id
```

```
15 evaluate_target_health = true
16 }
17 }
18
19 resource "aws_route53_record" "root_domain" {
20 zone_id = aws_route53_zone.main.zone_id
21 name = var.domain_name
22 type = "A"
23
24 alias {
25 name = aws_cloudfront_distribution.root_s3_distribution.domain_name
26 zone_id =
   aws_cloudfront_distribution.root_s3_distribution.hosted_zone_id
27 evaluate_target_health = false
28 }
29 }
30
31 resource "aws_route53_record" "certificate_validation_record" {
32 allow_overwrite = true
33 name =
   tolist(aws_acm_certificate.ssl_certificate.domain_validation_options)
   [0].resource_record_name
34 records =
   [tolist(aws_acm_certificate.ssl_certificate.domain_validation_options
   )[0].resource_record_value]
35 type =
   tolist(aws_acm_certificate.ssl_certificate.domain_validation_options)
   [0].resource_record_type
36 zone_id = aws_route53_zone.main.zone_id
37 ttl = 60
38 }
39
40 ##### CloudFlare
41
42 provider "cloudflare" {
43 api_token = var.cloudflare_api_token
44 }
45 resource "cloudflare_record" "NS0" {
46 zone_id = var.cloudflare_zone_id
```

```
47 name = var.project_name
48 value = aws_route53_zone.main.name_servers.0
49 type = "NS"
50 ttl = 600
51 }
52 resource "cloudflare_record" "NS1" {
53 zone_id = var.cloudflare_zone_id
54 name = var.project_name
55 value = aws_route53_zone.main.name_servers.1
56 type = "NS"
57 ttl = 600
58 }
59 resource "cloudflare_record" "NS2" {
60 zone_id = var.cloudflare_zone_id
61 name = var.project_name
62 value = aws_route53_zone.main.name_servers.2
63 type = "NS"
64 ttl = 600
65 }
66 resource "cloudflare_record" "NS3" {
67 zone_id = var.cloudflare_zone_id
68 name = var.project_name
69 value = aws_route53_zone.main.name_servers.3
70 type = "NS"
71 ttl = 600
72 }
```

output.tf

```
1 output "target_domain_name" {
2 value =
aws_apigatewayv2_domain_name.apigateway-domain-name.domain_name_config
uration[0].target_domain_name
3 }
4 output "api_gateway_endpoint" {
5 value = aws_apigatewayv2_api.apigateway.api_endpoint
6 }
```

opensearch.tf

```
1
2  # OpenSeach Service
3  resource "aws_opensearch_domain"
   "allianz_opensearch" {
4  domain_name = "opensearch-${var.project_name}"
5  engine_version = "OpenSearch_2.3"
6
7  cluster_config {
8  instance_type = "t3.small.search"
9  }
10 encrypt_at_rest {
11 enabled = true
12 }
13
14 domain_endpoint_options {
15 enforce_https = true
16 tls_security_policy = "Policy-Min-TLS-1-2-2019-07"
17 }
18 node_to_node_encryption {
19 enabled = true
20 }
21 advanced_security_options {
22 enabled = true
23 anonymous_auth_enabled = false
24 internal_user_database_enabled = true
25 master_user_options {
26 master_user_name = var.opensearch_user
27 master_user_password = var.opensearch_password
28 }
29 }
30 ebs_options {
31 ebs_enabled = true
32 volume_size = 10
33 }
34
```

```
35 }
36
```

mq.tf

```
1
2  # RabbitMQ Service
3  resource "aws_mq_broker" "allianz-mq" {
4  broker_name = "${var.project_name}-mq"
5
6  engine_type = "RabbitMQ"
7  engine_version = "3.10.10"
8  host_instance_type = "mq.t3.micro"
9  publicly_accessible = true
10 #security_groups = [aws_security_group.test.id]
11
12 user {
13 username = var.rabbit_user
14 password = var.rabbit_pass
15 }
16 }
17
```

main.tf

```
1  terraform {
2  required_version = ">= 1.0"
3
4  required_providers {
5  aws = {
6  source = "hashicorp/aws"
7  version = ">= 4.40"
8  }
9  cloudflare = {
10 source = "cloudflare/cloudflare"
11 version = "~> 3.0"
12 }
```

```
13 }
14 }
15
16 ## Locals
17
18 locals {
19 name = var.project_name
20 rds_name = "${var.project_name}_rds"
21 region = "us-east-1"
22 tags = {
23 Owner = "${var.project_name}"
24 Environment = "staging"
25 }
26 }
27 ##
28
29 provider "aws" {
30 region = "us-east-1"
31 profile = "acloudguru"
32 }
33
34 provider "aws" {
35 alias = "acm_provider"
36 region = "us-east-1"
37 profile = "acloudguru"
38 }
39
40 # VPC Config
41 module "vpc" {
42 source = "terraform-aws-modules/vpc/aws"
43 version = "~> 2"
44
45 name = local.name
46 cidr = "10.99.0.0/18"
47
48 azs = ["${local.region}a", "${local.region}b", "${local.region}c"]
```

```
49 public_subnets = ["10.99.0.0/24", "10.99.1.0/24", "10.99.2.0/24"]
50 private_subnets = ["10.99.3.0/24", "10.99.4.0/24", "10.99.5.0/24"]
51 #database_subnets = ["10.99.7.0/24", "10.99.8.0/24", "10.99.9.0/24"]
52
53 create_database_subnet_group = false
54 enable_dns_hostnames = true
55
56 enable_nat_gateway = true
57 single_nat_gateway = true
58 reuse_nat_ips = true # <= Skip creation of EIPs for the NAT Gateways
59 external_nat_ip_ids = aws_eip.eip_nat.*.id # <= IPs specified here as
   input to the module
60
61 }
62
63 #Elastic IP for NAT Gateway
64 resource "aws_eip" "eip_nat" {
65 vpc = true
66 }
67
68 #Security Groups
69 module "security_group_rds" {
70 source = "terraform-aws-modules/security-group/aws"
71 version = "~> 4"
72
73 name = local.name
74 description = "Complete PostgreSQL security group"
75 vpc_id = module.vpc.vpc_id
76
77 # ingress
78 ingress_with_cidr_blocks = [
79 {
80 from_port = 5432
81 to_port = 5432
82 protocol = "tcp"
83 description = "PostgreSQL access from within VPC"
84 cidr_blocks = module.vpc.vpc_cidr_block
```

```
},
]

tags = local.tags
}

resource "aws_security_group" "public" {
name = "Allow public HTTP/HTTPS ALB"
description = "Public internet access"
vpc_id = module.vpc.vpc_id

}

resource "aws_security_group_rule" "public_out" {
type = "egress"
from_port = 0
to_port = 0
protocol = "-1"
cidr_blocks = ["0.0.0.0/0"]

security_group_id = aws_security_group.public.id
}

resource "aws_security_group" "private_alb" {
name = "Allow private traffic ALB"
#description = "Public internet access"
vpc_id = module.vpc.vpc_id

}

resource "aws_security_group_rule" "private_network_rule" {
type = "ingress"
from_port = 0
to_port = 0
protocol = "-1"
cidr_blocks = ["10.99.0.0/18"]
```

```
121
122 security_group_id = aws_security_group.private_alb.id
123 }
124
125 resource "aws_security_group_rule" "public_in_https_svc_one" {
126 type = "ingress"
127 from_port = 8082
128 to_port = 8082
129 protocol = "tcp"
130 cidr_blocks = ["0.0.0.0/0"]
131 security_group_id = aws_security_group.public.id
132 }
133
134 resource "aws_security_group_rule" "public_in_https_svc_two" {
135 type = "ingress"
136 from_port = 8084
137 to_port = 8084
138 protocol = "tcp"
139 cidr_blocks = ["0.0.0.0/0"]
140 security_group_id = aws_security_group.public.id
141 }
142
143 resource "aws_security_group" "sg-ec2-ecs" {
144 name = "ec2-ecs instance"
145 description = "Public internet access"
146 vpc_id = module.vpc.vpc_id
147
148 }
149
150 resource "aws_security_group_rule" "public_out_ec2_ecs" {
151 type = "egress"
152 from_port = 0
153 to_port = 0
154 protocol = "-1"
155 cidr_blocks = ["0.0.0.0/0"]
156
```

```
157 security_group_id = aws_security_group.sg-ec2-ecs.id
158 }
159
160 resource "aws_security_group_rule" "private_in" {
161 type = "ingress"
162 from_port = 0
163 to_port = 0
164 protocol = "-1"
165 cidr_blocks = ["10.99.0.0/18"]
166
167 security_group_id = aws_security_group.sg-ec2-ecs.id
168 }
169 resource "aws_key_pair" "terraform_ec2_key" {
170 key_name = "terraform_ec2_key"
171 public_key = "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQC2/QlUK8DpNUyIXHMVtfH6XxesN0b9pH0ooPTuFR
uMNXn6vdGWkG2s3sIby4A6yVTTkmwQ47YhTc/B9/B+Ui4mAZG5l6xTOyZp/mPgXJZWNMfH
OMIu6tWU/PkahhhICDUVfnPcLSa8rT9ywi5w5nn2nKyujbpwMTwTJMKH6HR0OyT37g2FyP
nrnVXW6LRIelGYGWWWkCAqN768Wo9TV2UqSzA5kRW2pWb38iszWaYvdDhL82MffJpJeJ1x
Urna4SmeRqyuoiH9E6WPCn3wx++rNoyqLpuwyvEvJncN6UHGyTOrUfHmoK8u80dHHJQ8ce
nWWIFugfFR6bmo8xowqvi/ ggregoret"
172 }
```

ecs.tf

```
1 # IAM Roles and Policies
2 resource "aws_iam_role" "ecs-instance-role" {
3 name = "ecs-instance-role"
4 path = "/"
5 assume_role_policy =
data.aws_iam_policy_document.ecs-instance-policy.json
6 }
7
8 data "aws_iam_policy_document" "ecs-instance-policy" {
9 statement {
10 actions = ["sts:AssumeRole"]
11 principals {
12 type = "Service"
13 identifiers = ["ec2.amazonaws.com"]
```

```
14 }
15 }
16 }
17 data "aws_iam_policy_document" "ecsInstanceRole-policy" {
18 statement {
19 actions = ["sts:AssumeRole"]
20 principals {
21 type = "Service"
22 identifiers = ["ec2.amazonaws.com"]
23 }
24 }
25 }
26 resource "aws_iam_role_policy_attachment"
   "ecs-instance-role-attachment" {
27 role = aws_iam_role.ecs-instance-role.name
28 policy_arn =
   "arn:aws:iam::aws:policy/service-role/AmazonEC2ContainerServiceforEC
   2Role"
29 }
30
31 resource "aws_iam_instance_profile" "ecs-instance-profile" {
32 name = "ecs-instance-profile"
33 path = "/"
34 role = aws_iam_role.ecs-instance-role.id
35 }
36
37 resource "aws_iam_role" "ecs-service-role" {
38 name = "ecs-service-role"
39 path = "/"
40 assume_role_policy =
   data.aws_iam_policy_document.ecs-service-policy.json
41 }
42
43 resource "aws_iam_role_policy_attachment"
   "ecs-service-role-attachment" {
44 role = aws_iam_role.ecs-service-role.name
45 policy_arn =
   "arn:aws:iam::aws:policy/service-role/AmazonEC2ContainerServiceRole"
46 }
```

```
47
48 data "aws_iam_policy_document" "ecs-service-policy" {
49 statement {
50 actions = ["sts:AssumeRole"]
51 principals {
52 type = "Service"
53 identifiers = ["ecs.amazonaws.com"]
54 }
55 }
56 }
57
58 data "aws_iam_policy_document" "ecs_agent" {
59 statement {
60 actions = ["sts:AssumeRole"]
61
62 principals {
63 type = "Service"
64 identifiers = ["ec2.amazonaws.com"]
65 }
66 }
67 }
68
69 resource "aws_iam_role" "ecs_agent" {
70 name = "ecs-agent"
71 assume_role_policy = data.aws_iam_policy_document.ecs_agent.json
72 }
73
74 resource "aws_iam_role_policy_attachment" "Cloudwatch_FullAccess" {
75 role = aws_iam_role.ecs_agent.name
76 policy_arn = "arn:aws:iam::aws:policy/CloudWatchLogsFullAccess"
77 }
78
79 resource "aws_iam_role_policy_attachment"
   "Cloudwatch_FullAccess_task_role" {
80 role = aws_iam_role.ecs_task_role.name
81 policy_arn = "arn:aws:iam::aws:policy/CloudWatchLogsFullAccess"
82 }
```

```
83
84  resource "aws_iam_role_policy_attachment"
    "Cloudwatch_FullAccess_task_execution" {
85  role = aws_iam_role.ecs_task_execution_role.name
86  policy_arn = "arn:aws:iam::aws:policy/CloudWatchLogsFullAccess"
87  }
88
89  resource "aws_iam_role_policy_attachment" "ecs_agent" {
90  role = aws_iam_role.ecs_agent.name
91  policy_arn =
    "arn:aws:iam::aws:policy/service-role/AmazonEC2ContainerServiceforEC
    2Role"
92  depends_on = [aws_iam_role.ecs_agent]
93  }
94
95  resource "aws_iam_instance_profile" "ecs_agent" {
96  name = "ecs-agent"
97  role = aws_iam_role.ecs_agent.name
98  }
99
100 ##### ECS-Cluster #####
101 resource "aws_ecs_cluster" "cluster" {
102 name = var.ecs_cluster_name
103 }
104
105 ##### ECS Cluster - EC2 Instance #####
106 resource "aws_instance" "ec2_instance" {
107 ami = "ami-02861932a7d48032d"
108 subnet_id = module.vpc.private_subnets[0]
109 instance_type = "t3a.small"
110 iam_instance_profile = aws_iam_instance_profile.ecs_agent.name
111 vpc_security_group_ids = [aws_security_group.sg-ec2-ecs.id]
112 ebs_optimized = "false"
113 source_dest_check = "false"
114 key_name = "terraform_ec2_key"
115 user_data = data.template_file.user_data.rendered
116 root_block_device {
```

```
volume_type = "gp2"
volume_size = "10"
delete_on_termination = "true"
}

lifecycle {
ignore_changes = [key_name, ebs_optimized, private_ip]
}
}

data "template_file" "user_data" {
template = file("user_data.tpl")
vars = {
ECS_CLUSTER_NAME = var.ecs_cluster_name
}
}

##### ECS - Service Two #####
##### ECS Task Definition #####

resource "aws_ecs_task_definition" "task_definition_service_two" {
container_definitions =
data.template_file.task_definition_service_two_json.rendered # task
definition json file location
family = "service-two" # task name
network_mode = "bridge" # network mode awsvpc, brigde
memory = "1024"
cpu = "1024"
requires_compatibilities = ["EC2"] # Fargate or EC2
depends_on = [aws_db_instance.mysql, aws_instance.consul_instance,
aws_instance.logstash_instance]
}

data "template_file" "task_definition_service_two_json" {
template = file("task_definition_service_two.json")

vars = {
```

```
151 CONSUL_HOST = aws_instance.consul_instance.private_dns
152 CONSUL_PORT = var.consul_port
153 RDS_HOST = aws_db_instance.mysql.endpoint
154 RDS_DB = var.rds_db_name
155 RDS_USER = var.rds_username
156 RDS_PASS = var.rds_password
157 LOGSTASH_HOST = aws_instance.logstash_instance.private_dns
158 LOGSTASH_PORT = var.logstash_port
159 RABBIT_HOST = var.rabbit_host
160 RABBIT_USER = var.rabbit_user
161 RABBIT_PASS = var.rabbit_pass
162 PROJECT_NAME = var.project_name
163 IMAGE = var.service_two_docker_image
164 }
165
166 }
167
168 ##### ECS Service #####
169
170
171 resource "aws_ecs_service" "service-two-service" {
172 cluster = aws_ecs_cluster.cluster.id # ecs cluster id
173 desired_count = 1 # no of task running
174 launch_type = "EC2" # Cluster type ECS OR FARGATE
175 name = "service-two-service" # Name of service
176 task_definition =
    aws_ecs_task_definition.task_definition_service_two.arn # Attaching
    Task to service
177
178
179 load_balancer {
180 container_name = "service-two"
181 container_port = "8084"
182 target_group_arn = aws_alb_target_group.service-two-public.arn #
    attaching load_balancer target group to ecs
```

```
183 }
184 depends_on = [aws_security_group.sg-ec2-ecs,
    time_sleep.wait_120_seconds]
185 }
186
187 ##### ECS - Service One #####
188 ##### ECS Task Definition #####
189
190 resource "aws_ecs_task_definition" "task_definition_service_one" {
191 container_definitions =
    data.template_file.task_definition_service_one_json.rendered # task
    definition json file location
192 execution_role_arn = aws_iam_role.ecs_task_execution_role.arn
193 task_role_arn = aws_iam_role.ecs_task_role.arn
194 family = "service-one" # task name
195 network_mode = "awsvpc" # network mode awsvpc, brigde
196 memory = "1024"
197 cpu = "512"
198 requires_compatibilities = ["FARGATE"] # Fargate or EC2
199
200 depends_on = [aws_db_instance.mysql, aws_instance.consul_instance,
    aws_instance.logstash_instance,
    aws_docdb_cluster_instance.cluster_instances]
201 }
202
203 data "template_file" "task_definition_service_one_json" {
204 template = file("task_definition_service_one.json")
205
206 vars = {
207 CONSUL_HOST = aws_instance.consul_instance.private_dns
208 CONSUL_PORT = var.consul_port
209 MONGO_HOST = aws_docdb_cluster_instance.cluster_instances.endpoint
210 MONGO_DB = var.docdb_db
211 MONGO_USER = var.docdb_username
212 MONGO_PASS = var.docdb_password
213 LOGSTASH_HOST = aws_instance.logstash_instance.private_dns
214 LOGSTASH_PORT = var.logstash_port
215 RABBIT_HOST = var.rabbit_host
```

```
RABBIT_USER = var.rabbit_user
RABBIT_PASS = var.rabbit_pass
PROJECT_NAME = var.project_name
IMAGE = var.service_one_docker_image
}
}

##### ECS Service #####
resource "aws_ecs_service" "service-one-service" {
cluster = aws_ecs_cluster.cluster.id # ecs cluster id

desired_count = 1 # no of task running

launch_type = "FARGATE" # Cluster type ECS OR FARGATE

name = "service-one-service" # Name of service

task_definition =
aws_ecs_task_definition.task_definition_service_one.arn # Attaching
Task to service

network_configuration {
security_groups = [aws_security_group.sg-ec2-ecs.id]
subnets = [module.vpc.private_subnets[0]]
assign_public_ip = false
}

load_balancer {
container_name = "service-one"
container_port = "8082"
target_group_arn = aws_alb_target_group.service-one-public.arn #
attaching load_balancer target group to ecs
}
depends_on = [aws_security_group.sg-ec2-ecs,
time_sleep.wait_120_seconds]
}



##### IAM Role for Fargate #####
resource "aws_iam_role" "ecs_task_execution_role" {
```

```
name = "service-one-ecsTaskExecutionRole"

assume_role_policy = <<EOF
{
"Version": "2012-10-17",
"Statement": [
{
"Action": "sts:AssumeRole",
"Principal": {
"Service": "ecs-tasks.amazonaws.com"
},
"Effect": "Allow",
"Sid": ""
}
]
}
EOF
}

resource "aws_iam_role_policy_attachment"
"ecs-task-execution-role-policy-attachment" {
role = aws_iam_role.ecs_task_execution_role.name
policy_arn =
"arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePoli
cy"
}
resource "aws_iam_role" "ecs_task_role" {
name = "service-one-ecsTaskRole"

assume_role_policy = <<EOF
{
"Version": "2012-10-17",
"Statement": [
{
"Action": "sts:AssumeRole",
"Principal": {
"Service": "ecs-tasks.amazonaws.com"
},
```

```
"Effect": "Allow",
"Sid": ""
}
]
}
EOF
}

resource "aws_iam_policy" "dynamodb" {
name = "service-one-task-policy-dynamodb"
description = "Policy that allows access to DynamoDB"

policy = <<EOF
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": [
"dynamodb:CreateTable",
"dynamodb:UpdateTimeToLive",
"dynamodb:PutItem",
"dynamodb:DescribeTable",
"dynamodb:ListTables",
"dynamodb:DeleteItem",
"dynamodb:GetItem",
"dynamodb:Scan",
"dynamodb:Query",
"dynamodb:UpdateItem",
"dynamodb:UpdateTable"
],
"Resource": "*"
}
]
}
EOF
```

```
319 }
320
321 resource "aws_iam_role_policy_attachment"
    "ecs-task-role-policy-attachment" {
322 role = aws_iam_role.ecs_task_role.name
323 policy_arn = aws_iam_policy.dynamodb.arn
324 }
325
326 resource "time_sleep" "wait_120_seconds" {
327 depends_on = [aws_db_instance.mysql,
    aws_docdb_cluster.docdb_cluster,
    aws_docdb_cluster_instance.cluster_instances]
328
329 create_duration = "120s"
330 }
331
332 resource "null_resource" "next" {
333 depends_on = [time_sleep.wait_120_seconds]
334 }
335
336 resource "time_sleep" "wait_120_seconds_services" {
337 depends_on = [aws_ecs_service.service-one-service,
    aws_ecs_service.service-two-service]
338
339 create_duration = "120s"
340 }
341
```

ec2-logstash.tf

```
1 # Template
2 data "template_file" "user_data_logstash" {
3 template = file("user_data_logstash.tpl")
4 vars = {
5 OS_USER = var.opensearch_user
6 OS_PASSWD = var.opensearch_password
7 OS_URL = var.opensearch_url
8 }
```

```
}
# EC2 Instance - Logstash
resource "aws_instance" "logstash_instance" {
ami = "ami-04fba13aa6da74c92"
subnet_id = module.vpc.private_subnets[0]
instance_type = "t3.micro"
private_ip = "10.99.3.202"
iam_instance_profile = aws_iam_instance_profile.ecs_agent.name
vpc_security_group_ids = [aws_security_group.sg-ec2-ecs.id]
ebs_optimized = "false"
source_dest_check = "false"
key_name = "terraform_ec2_key"
user_data = data.template_file.user_data_logstash.rendered
private_dns_name_options {
enable_resource_name_dns_a_record = true
hostname_type = "ip-name"
}
root_block_device {
volume_type = "gp2"
volume_size = "10"
delete_on_termination = "true"
}


}

```

ec2-consul.tf

```
# Template
data "template_file" "user_data_consul" {
template = file("user_data_consul.tpl")
}
# EC2 Instance - Consul
resource "aws_instance" "consul_instance" {
ami = "ami-04fba13aa6da74c92"
subnet_id = module.vpc.private_subnets[0]
instance_type = "t3a.nano"
```

```
private_ip = "10.99.3.200"
iam_instance_profile = aws_iam_instance_profile.ecs_agent.name
vpc_security_group_ids = [aws_security_group.sg-ec2-ecs.id]
ebs_optimized = "false"
source_dest_check = "false"
key_name = "terraform_ec2_key"
user_data = data.template_file.user_data_consul.rendered
private_dns_name_options {
enable_resource_name_dns_a_record = true
hostname_type = "ip-name"
}
root_block_device {
volume_type = "gp2"
volume_size = "10"
delete_on_termination = "true"
}

}
```

documentdb.tf

```
# Security Group DocumentDB
resource "aws_security_group" "docdb-sg" {
name = "my-docdb-sg"
vpc_id = module.vpc.vpc_id
}

# Ingress Security Port 27017
resource "aws_security_group_rule" "docdb_inbound_access" {
from_port = 27017
protocol = "tcp"
security_group_id = aws_security_group.docdb-sg.id
to_port = 27017
type = "ingress"
cidr_blocks = ["0.0.0.0/0"]
}

```

```
# DocDB Cluster Instance
resource "aws_docdb_cluster_instance" "cluster_instances" {
identifier = "docdb-cluster-${var.project_name}"
cluster_identifier = aws_docdb_cluster.docdb_cluster.id
instance_class = "db.t3.medium"

lifecycle {
ignore_changes = [identifier]
}
}

# DocDB Cluster
resource "aws_docdb_cluster" "docdb_cluster" {
cluster_identifier = "docdb-cluster-${var.project_name}"
availability_zones = ["us-east-1a"]
master_username = var.docdb_username
master_password = var.docdb_password
db_cluster_parameter_group_name =
aws_docdb_cluster_parameter_group.docdb-paramgroup.id
vpc_security_group_ids = ["${aws_security_group.docdb-sg.id}"]
db_subnet_group_name = aws_docdb_subnet_group.docdb_subnetgroup.name

skip_final_snapshot = true
lifecycle {
ignore_changes = [availability_zones]
}


}

# DocDB Parameter Group
resource "aws_docdb_cluster_parameter_group" "docdb-paramgroup" {
family = "docdb5.0"
name = "docdb-paramgroup"
description = "docdb cluster parameter group"

parameter {
name = "tls"
```

```
value = "disabled"
}
parameter {
name = "ttl_monitor"
value = "disabled"
}
}

resource "aws_docdb_subnet_group" "docdb_subnetgroup" {
name = "${var.project_name}-docdb-subnetgroup"
subnet_ids = [module.vpc.private_subnets[0],
module.vpc.private_subnets[1]]
}
```

cloudfront.tf

```
# Cloudfront - Frontend

resource "aws_cloudfront_distribution" "root_s3_distribution" {
origin {
domain_name =
aws_s3_bucket_website_configuration.root_bucket.website_endpoint
origin_id = "S3-.${var.bucket_name}"
custom_origin_config {
http_port = 80
https_port = 443
origin_protocol_policy = "http-only"
origin_ssl_protocols = ["TLSv1", "TLSv1.1", "TLSv1.2"]
}
}

enabled = true
is_ipv6_enabled = true
default_root_object = "index.html"
aliases = [var.domain_name]

custom_error_response {
```

```
error_caching_min_ttl = 0
error_code = 404
response_code = 200
response_page_path = "/index.html"
}

default_cache_behavior {
allowed_methods = ["GET", "HEAD"]
cached_methods = ["GET", "HEAD"]
target_origin_id = "S3-.${var.bucket_name}"

forwarded_values {
query_string = true

cookies {
forward = "none"
}

headers = ["Origin"]
}

viewer_protocol_policy = "allow-all"
min_ttl = 0
default_ttl = 86400
max_ttl = 31536000
}

restrictions {
geo_restriction {
restriction_type = "none"
}
}

viewer_certificate {
acm_certificate_arn =
aws_acm_certificate_validation.cert_validation.certificate_arn
ssl_support_method = "sni-only"
```

```
minimum_protocol_version = "TLSv1.1_2016"
}
}
```

apigw.tf

```
resource "aws_apigatewayv2_api" "apigateway" {
name = "${var.project_name}-api-gateway"
protocol_type = "HTTP"
#depends_on = [time_sleep.wait_120_seconds_services]
}
# Ownership of domain name
resource "aws_apigatewayv2_domain_name" "apigateway-domain-name" {
domain_name = "api.${var.project_name}.gregoret.com.ar"

domain_name_configuration {
certificate_arn = aws_acm_certificate.ssl_certificate.arn
endpoint_type = "REGIONAL"
security_policy = "TLS_1_2"
}
depends_on = [aws_acm_certificate_validation.cert_validation]
}
# Domain Mapping
resource "aws_apigatewayv2_api_mapping" "api-mapping" {
api_id = aws_apigatewayv2_api.apigateway.id
domain_name = aws_apigatewayv2_domain_name.apigateway-domain-name.id
stage = aws_apigatewayv2_stage.apigw-stage.id
}

# Service One API
resource "aws_apigatewayv2_integration" "service-one-integration" {
api_id = aws_apigatewayv2_api.apigateway.id
description = "Service one integration with API Gateway"
integration_type = "HTTP_PROXY"
integration_uri = aws_lb_listener.service-one-lb-listener.arn

```

```
integration_method = "ANY"
connection_type = "VPC_LINK"
connection_id = aws_apigatewayv2_vpc_link.vpc-link.id

tls_config {
server_name_to_verify = "api.${var.project_name}.gregoret.com.ar"
}

request_parameters = {
"overwrite:path" = "$request.path.proxy"
}
}

resource "aws_apigatewayv2_route" "service-one-route" {
api_id = aws_apigatewayv2_api.apigateway.id
route_key = "ANY /service-one/{proxy+}"

target =
"integrations/${aws_apigatewayv2_integration.service-one-integration.
id}"
}

# Service Two API
resource "aws_apigatewayv2_integration" "service-two-integration" {
api_id = aws_apigatewayv2_api.apigateway.id
description = "Service two integration with API Gateway"
integration_type = "HTTP_PROXY"
integration_uri = aws_lb_listener.service-two-lb-listener.arn

integration_method = "ANY"
connection_type = "VPC_LINK"
connection_id = aws_apigatewayv2_vpc_link.vpc-link.id

tls_config {
server_name_to_verify = "api.${var.project_name}.gregoret.com.ar"
}

```

```
66  request_parameters = {
67  "overwrite:path" = "$request.path.proxy"
68  }
69  }
70
71  resource "aws_apigatewayv2_route" "service-two-route" {
72  api_id = aws_apigatewayv2_api.apigateway.id
73  route_key = "ANY /service-two/{proxy+}"
74
75  target =
    "integrations/${aws_apigatewayv2_integration.service-two-integration.
    id}"
76  }
77
78  #General
79  resource "aws_apigatewayv2_vpc_link" "vpc-link" {
80  name = "vpc-link"
81  security_group_ids = [aws_security_group.public.id]
82  subnet_ids = [module.vpc.private_subnets[0],
    module.vpc.private_subnets[1], module.vpc.private_subnets[2]]
83  }
84
85  resource "aws_apigatewayv2_stage" "apigw-stage" {
86  api_id = aws_apigatewayv2_api.apigateway.id
87  name = "$default"
88  auto_deploy = true
89  }
```

alb.tf

```
1  ##### ALB - Application Load Balancing #####
2  ### ALB Public ###
3  resource "aws_lb" "public-load-balancer" {
4  internal = true # internal = true else false
5  name = "${var.project_name}-public-alb"
6  load_balancer_type = "application"
7  subnets = [module.vpc.private_subnets[0],
   module.vpc.private_subnets[1], module.vpc.private_subnets[2]] #
   Subnets privadas
```

```
8  security_groups = [aws_security_group.public.id]
9  }
10
11 ##### ALB - Target Groups #####
12
13 resource "aws_alb_target_group" "service-one-public" {
14 name = "service-one-tg"
15 port = "8082"
16 protocol = "HTTP"
17 vpc_id = module.vpc.vpc_id
18 target_type = "ip"
19
20
21 health_check {
22 healthy_threshold = "2"
23 interval = "15"
24 path = "/actuator/health"
25 protocol = "HTTP"
26 unhealthy_threshold = "8"
27 timeout = "6"
28 }
29 }
30
31 resource "aws_alb_target_group" "service-two-public" {
32 name = "service-two-tg"
33 port = "8084"
34 protocol = "HTTP"
35 vpc_id = module.vpc.vpc_id
36
37
38 health_check {
39 healthy_threshold = "2"
40 interval = "15"
41 path = "/actuator/health"
42 protocol = "HTTP"
43 unhealthy_threshold = "8"
```

```
44 timeout = "6"
45 }
46 }
47
48
49 ##### ALB - Listeners #####
50 resource "aws_lb_listener" "service-one-lb-listener" {
51 load_balancer_arn = aws_lb.public-load-balancer.arn
52 port = "8082"
53 protocol = "HTTPS"
54 ssl_policy = "ELBSecurityPolicy-2016-08"
55 certificate_arn = aws_acm_certificate.ssl_certificate.arn
56
57 default_action {
58 type = "forward"
59 target_group_arn = aws_alb_target_group.service-one-public.id
60 }
61 }
62
63 resource "aws_lb_listener" "service-two-lb-listener" {
64 load_balancer_arn = aws_lb.public-load-balancer.arn
65 port = "8084"
66 protocol = "HTTPS"
67 ssl_policy = "ELBSecurityPolicy-2016-08"
68 certificate_arn = aws_acm_certificate.ssl_certificate.arn
69
70 default_action {
71 type = "forward"
72 target_group_arn = aws_alb_target_group.service-two-public.id
73 }
74 }
75
76
77
```

acm.tf

```
# SSL Certificate
resource "aws_acm_certificate" "ssl_certificate" {
provider = aws.acm_provider
domain_name = var.domain_name
subject_alternative_names = ["*.${var.domain_name}"]
validation_method = "DNS"
lifecycle {
create_before_destroy = true
}
}

resource "aws_acm_certificate_validation"
"cert_validation" {
provider = aws.acm_provider
certificate_arn = aws_acm_certificate.ssl_certificate.arn
}
```

templates\s3-policy.json

```
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "PublicReadGetObject",
"Effect": "Allow",
"Principal": "*",
"Action": "s3:GetObject",
"Resource": "arn:aws:s3:::${bucket}/*"
}
]
}
```

Anexo II: Diagnóstico inicial del cliente

Lenguaje del backend

- ☐ Java
- ☐ NodeJS
- ☐ Python
- ☐ PHP
- ☐ Go
- ☐ Perl
- ☐ Kotlin
- ☐ JavaScript
- ☐ C#
- ☐ Laravel
- ☐ Otro:_____

Lenguaje del frontend

- ☐ JavaScript
- ☐ ReactJS
- ☐ NextJS
- ☐ Redux
- ☐ Bootstrap
- ☐ Angular
- ☐ ReactJS
- ☐ VueJS
- ☐ Otro:_____

Motor de bases de datos

- ☐ MySQL
- ☐ MariaDB
- ☐ SQL Server
- ☐ PostgreSQL
- ☐ Oracle
- ☐ MongoDB
- ☐ Cassandra
- ☐ FireBase
- ☐ Otro:_____

Networking

¿Utiliza certificados SSL? (Si/No) _____
Dominio de internet: _____

Servicio que administra el DNS:_____
Puertos utilizados (*servicio:puerto*)

1. _____
2. _____
3. _____
4. _____
5. _____
6. _____

Repositorios de código

Plataforma de repositorios y control de versiones

- ☐ GitHub
- ☐ GitLab
- ☐ BitBucket
- ☐ AWS CodeCommit
- ☐ Otro:_____

¿Implementan GitFlow? (Branches Main, Develop, etc)

- ☐ Si
- ☐ No

Plataforma de servicios en la nube

- ☐ Amazon Web Services
- ☐ Microsoft Azure
- ☐ Google Cloud
- ☐ Digital Ocean
- ☐ On-Premises
- ☐ Otro:_____

Integración continua y despliegue continuo

Usan pipelines o algún proceso de automatización de despliegues

- ☐ Si:_____
- ☐ No

Realizan tests de el código

- ☐ Si
- ☐ No

Arquitectura

- ☐ Microservicios
- ☐ Monolítico

Anexo III: Estimación de costos por plataforma

Azure

Microsoft Azure Estimate				
Estimate				
Service category	**Service type**	**Custom name**	**Description**	**Estimated monthly cost**
Integration	Service Bus	MQ	Basic tier: 1 million messaging operations	$0.05
Compute	Virtual Machines	Consul	1 B1ls (1 Core, 0.5 GB RAM) x 730 Hours (Pay as you go), Linux, (Pay as you go); 1 managed disk – P3; Inter Region transfer type, 5 GB outbound data transfer from East US to East Asia	$6.20
Web	API Management	API Gateway	Developer tier, 1 unit(s), 730 Hours, 0 x 730 Hours x 5 overage workspaces	$48.03
Databases	Azure Database for MySQL	MySQL - Relational DB	Single Server Deployment, Basic Tier, 1 Gen 5 (2 vCore) x 730 Hours, 10 GB Storage with ZRS redundancy, 0 GB Additional Backup storage - LRS redundancy	$50.64
Databases	Azure Cosmos DB	MongoDB - No Relational DB	Azure Cosmos DB for MongoDB, Standard provisioned throughput (manual), Always-free quantity disabled, Single Region Write (Single-Master) - East US (Write Region), 400 RU/s x 730 Hours, 10 GB transactional storage, Analytical storage disabled, 7-day continuous backup storage	$25.86

Compute	Virtual Machines	Elastic Stack (ELK) - ElasticSearch Kibana Logstash	1 B4ms (4 Cores, 16 GB RAM) x 730 Hours (Pay as you go), Linux, (Pay as you go); 1 managed disk – P4; Inter Region transfer type, 5 GB outbound data transfer from East US to East Asia	$126.46
Networking	Load Balancer		Basic Load Balancer is free of charge	$0.00
Networking	Azure Firewall		Basic tier, 1 Logical firewall units x 730 Hours, 0 GB Data processed	$288.35
Identity	Azure Active Directory External Identities	Azure IdP	Premium P2 tier: 50,000 monthly active user(s), 0 SMS/Phone Events	$0.00
Networking	Azure DNS		Zone 1, DNS, Public; 1 hosted DNS zone, 1 DNS query	$0.90
Support			0	$0.00
		Licensing Program	**Microsoft Customer Agreement (MCA)**	
		Billing Account		
		Billing Profile		
		Total		**$546.49**

AWS

Contact your AWS representative: Contact Sales

Export date: **5/12/2023** Language: **English**

Estimate title: **My Estimate**

Estimate URL: **https://calculator.aws/#/estimate?id=9d01f3d4ded06539b7460697e21e5497860dc77b**

Estimate summary

Upfront cost	Monthly cost	Total 12 months cost
0.00 USD	**259.83 USD**	**3,117.97 USD** Includes upfront cost

Detailed Estimate

Name	Group	Region	Upfront cost	Monthly cost
Amazon EC2	No group applied	US East (N. Virginia)	0.00 USD	4.43 USD
Description: EC2 - Consul Cluster **Config summary**: Tenancy (Shared Instances), Operating system (Linux), Workload (Consistent, Number of instances: 1), Advance EC2 instance (t3a.nano), Pricing strategy (On-Demand Utilization: 100 %Utilized/Month), Enable monitoring (disabled), EBS Storage amount (10 GB), DT Inbound: Not selected (0 TB per month), DT Outbound: Not selected (0 TB per month), DT Intra-Region: (0 TB per month)				
Amazon DocumentDB (with MongoDB compatibility)	No group applied	US East (N. Virginia)	0.00 USD	57.94 USD
Description: DocumentDB Cluster **Config summary**: Quantity (1), Server utilization (730 Hours/Month), Instance type (db.t3.medium), Storage (10 GB)				

Amazon EC2	No group applied	US East (N. Virginia)	0.00 USD	8.59 USD

Description: EC2 - Logstash
Config summary: Tenancy (Shared Instances), Operating system (Linux), Workload (Consistent, Number of instances: 1), Advance EC2 instance (t3.micro), Pricing strategy (On-Demand Utilization: 100 %Utilized/Month), Enable monitoring (disabled), EBS Storage amount (10 GB), DT Inbound: Not selected (0 TB per month), DT Outbound: Not selected (0 TB per month), DT Intra-Region: (0 TB per month)

Amazon EC2	No group applied	US East (N. Virginia)	0.00 USD	28.45 USD

Description: EC2 - ECS Cluster - ServiceTwo
Config summary: Tenancy (Shared Instances), Operating system (Linux), Workload (Consistent, Number of instances: 1), Advance EC2 instance (t3a.medium), Pricing strategy (On-Demand Utilization: 100 %Utilized/Month), Enable monitoring (disabled), EBS Storage amount (10 GB), DT Inbound: Not selected (0 TB per month), DT Outbound: Not selected (0 TB per month), DT Intra-Region: (0 TB per month)

Amazon MQ	No group applied	US East (N. Virginia)	0.00 USD	20.76 USD

Description: RabbitMQ
Config summary: Broker type (Single-instance Broker), DT Inbound: All other regions (0 TB per month), DT Outbound: Not selected (0 TB per month), DT Intra-Region: (1 GB per month), Amazon RabbitMQ Broker Instance (mq.t3.micro), Storage per Broker (10 GB), Number of Brokers running (1)

Amazon OpenSearch Service	No group applied	US East (N. Virginia)	0.00 USD	26.28 USD

Description: OpenSearch
Config summary: Number of instances (1), Storage for each Amazon OpenSearch Service instance (General Purpose SSD (gp3)), UltraWarm storage cost (0), Nodes (1), Instance type (t3.small.search), Utilization (On-Demand only) (100 %Utilized/Month), Instance Node Type (General purpose), Storage Type (EBS Only), Pricing strategy (OnDemand), Nodes (0), Instance type (r5.2xlarge.search), Utilization (On-Demand only) (100 %Utilized/Month), Instance Node Type (Memory optimized), Storage Type (EBS Only), Pricing strategy (OnDemand), Number of nodes (0), Instance type (ultrawarm1.large.search), Utilization (On-Demand only) (100 %Utilized/Month), Pricing strategy (OnDemand)

Amazon RDS for MySQL	No group applied	US East (N. Virginia)	0.00 USD	25.97 USD
Description: RDS **Config summary**: Storage for each RDS instance (General Purpose SSD (gp2)), Storage amount (10 GB), Quantity (1), Instance type (db.t3.small), Utilization (On-Demand only) (100 %Utilized/Month), Deployment option (Single-AZ), Pricing strategy (OnDemand)				
Amazon Simple Storage Service (S3)	No group applied	US East (N. Virginia)	0.00 USD	0.08 USD
Description: S3 Bucket **Config summary**: S3 Standard storage (1 GB per month)				
Amazon Route 53	No group applied	US East (N. Virginia)	0.00 USD	0.91 USD
Description: **Config summary**: Hosted Zones (1), Additional Records in Hosted Zones (5)				
AWS Fargate	No group applied	US East (N. Virginia)	0.00 USD	17.77 USD
Description: Fargate - ServiceOne **Config summary**: Operating system (Linux), CPU Architecture (x86), Average duration (30 days), Number of tasks or pods (1 per month), Amount of ephemeral storage allocated for Amazon ECS (20 GB), Amount of memory allocated (1 GB)				
Amazon API Gateway	No group applied	US East (N. Virginia)	0.00 USD	1.00 USD
Description: API Gateway HTTP **Config summary**: HTTP API requests units (millions), Average size of each request (34 KB), REST API request units (millions), Cache memory size (GB) (None), WebSocket message units (thousands), Average message size (32 KB), Requests (1 per month)				
Elastic Load Balancing	No group applied	US East (N. Virginia)	0.00 USD	16.44 USD
Description: Application Load Balancer **Config summary**: Number of Application Load Balancers (1)				
Amazon Virtual Private Cloud (VPC)	No group applied	US East (N. Virginia)	0.00 USD	40.61 USD
Description: VPC - NAT **Config summary**: Number of NAT Gateways (1) Number of VPC Interface endpoints per AWS region (1)				

AWS Web Application Firewall (WAF)	No group applied	US East (N. Virginia)	0.00 USD	10.60 USD

Description: WAF on ALB
Config summary: Number of Web Access Control Lists (Web ACLs) utilized (1 per month), Number of Rules added per Web ACL (5 per month)

Acknowledgement

AWS Pricing Calculator provides only an estimate of your AWS fees and doesn't include any taxes that might apply. Your actual fees depend on a variety of factors, including your actual usage of AWS services. Learn more

Google Cloud

Google Cloud Pricing Calculator - Estimate

Compute Engine

1 x Consul

Region: Iowa	
730 total hours per month	
Provisioning model: Regular	
Instance type: e2-micro	USD 6.11
Operating System / Software: Free	
Estimated Component Cost: USD 6.11 per 1 month	

1 x Elastic Stack (ELK) - ElasticSearch Kibana Logstash

Region: Iowa	
730 total hours per month	
Provisioning model: Regular	
Instance type: e2-standard-2	USD 48.92
Operating System / Software: Free	
Static public IP: 1 (730 hours)	USD 2.92
Estimated Component Cost: USD 51.84 per 1 month	

Cloud Load Balancing

Iowa

Forwarding rules: 2	USD 18.25
Inbound data processed: 1 GiB	USD 0.01
Outbound data processed: 1 GiB	USD 0.01
USD 18.27	

Cloud NAT

Number of assigned VM instances: 2	USD 2.04
Traffic Proccessed: 10 GiB	USD 0.45
USD 2.49	

Google Cloud Armor

Commitment type: Standard (Pay as you go)	
Policies: 5	USD 25.00
Rules: 5	USD 5.00
Incoming requests: 10,000	USD 0.01
USD 30.01	

Datastore

Iowa

Stored data: 1 GiB	USD 0.00
Entity Reads: 100,000	USD 0.00
Entity Writes: 100,000	USD 0.00
Entity Deletes: 100,000	USD 0.00
USD 0.00	

A part of your estimate fits within the Datastore free tier.

Cloud SQL for MySQL

MYSQL - RELATIONALDB

Number of instances: 1	
Location: Iowa	
Total hours per month: 730.0	
Instance type: db-lightweight-2	USD 79.46
SSD Storage: 10.0 GiB	USD 1.70
Backup: 0.0 GiB	USD 0.00

USD 81.16

Cloud DNS

Managed Zones: 1	USD 0.20
Queries: 10,000	USD 0.00
USD 0.20	

Pub/Sub

Service Queue

Message delivery type: Basic	
Volume: 250 MiB	USD 0.00
Subscriptions: 2	USD 0.19
USD 0.19	

Cloud Run

Services one and two

Region: Iowa	
CPU allocation type: CPU is always allocated	
CPU: 1	
Memory: 1 GiB	
CPU allocation time: 5,256,000 vCPU-second	USD 90.29
Memory allocation time: 5,256,000 GiB-second	USD 9.61
Number of instances at peak: 2	
Minimum number of instances: 2	
USD 99.90	

Persistent Disk (Accompanying)

1 x boot disk

Product accompanying: Compute Engine	
	USD 1.00

Zonal balanced PD: 10 GiB

USD 1.00

1 x boot disk

Product accompanying: Compute Engine

Zonal SSD PD: 20 GiB	USD 3.40

USD 3.40

Total Estimated Cost: USD 294.57 per 1 month

Estimate Currency

USD - US Dollar