Despliegues de sistemas de ML orientado a microservicios

Clase 2

FAMAF Facultad de Matemática, Astronomía, Física y Computación
UNC
Córdoba Technology Cluster
CCAD Centro de Computación de Alto Desempeño de la Universidad Nacional de Córdoba

Temas clase 2

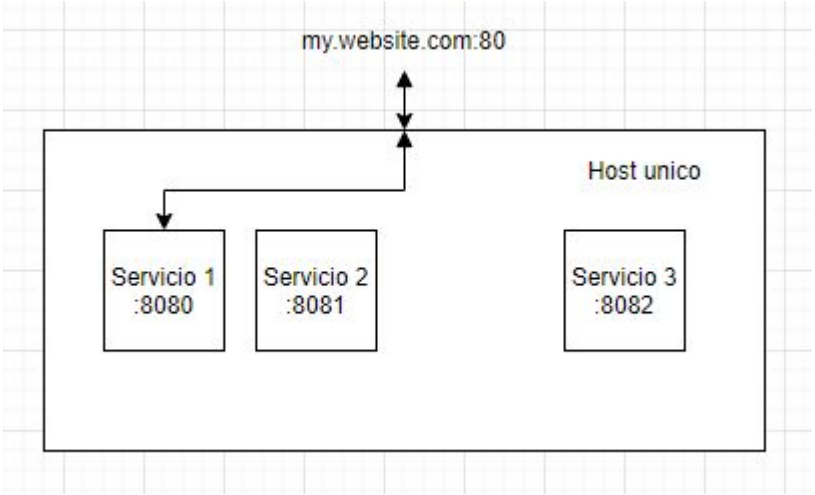
- Balanceadores de carga y DNS resolver
- Comunicación entre servicios.
  - Redis
  - Kafka
  - RabbitMQ, Nats, Mosquitto, ...

- Actividad: kafka cluster

Balanceadores de carga y DNS resolver

Hasta aquí tenemos lo siguiente

Servicios independientes y aislados en un host único pero en distintos puertos. El problema es que si quisiéramos alojar distintos sitios con un mismo DNS no podríamos pues solo podemos hacer un redireccionamiento en este caso solo al servicio 1.



Balanceadores de carga

En caso de utilizar réplicas es muy probable que quisiéramos realizar un balance en la carga de nuestros modelos para lo cual vamos a necesitar un **load balancer..** Existen muchas alternativas para esta tarea, estos son los más populares..

Balanceadores de carga

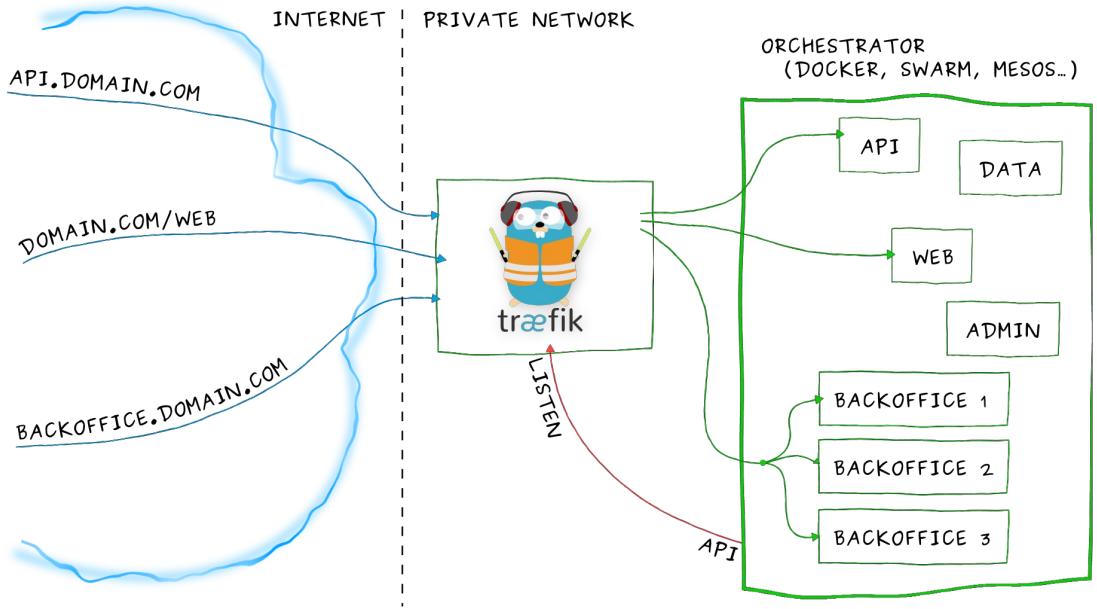
En caso de utilizar replicas es muy probable
nuestros modelos para lo cual vamos a nec
para esta tarea, estos son los mas populares

Los nuevos workers se reportan automáticamente a través de un label. No hay necesidad de cambio de archivos de configuración estáticos.

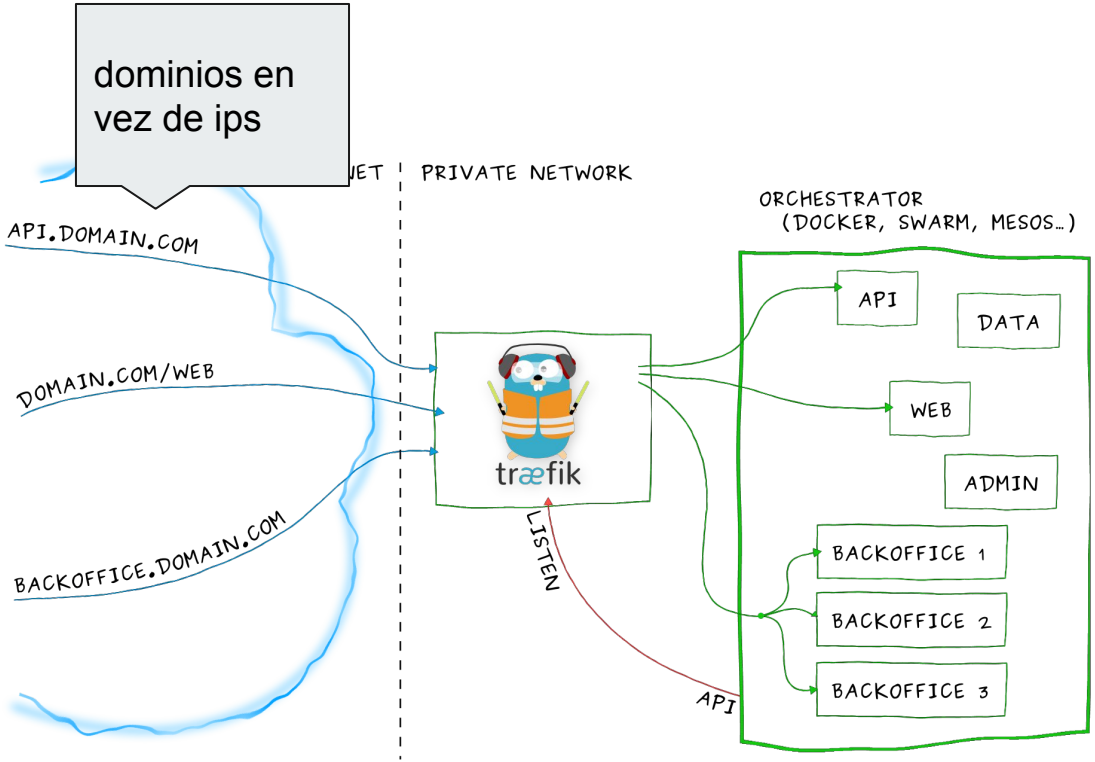
NGINX

APACHE
HTTP SERVER PROJECT

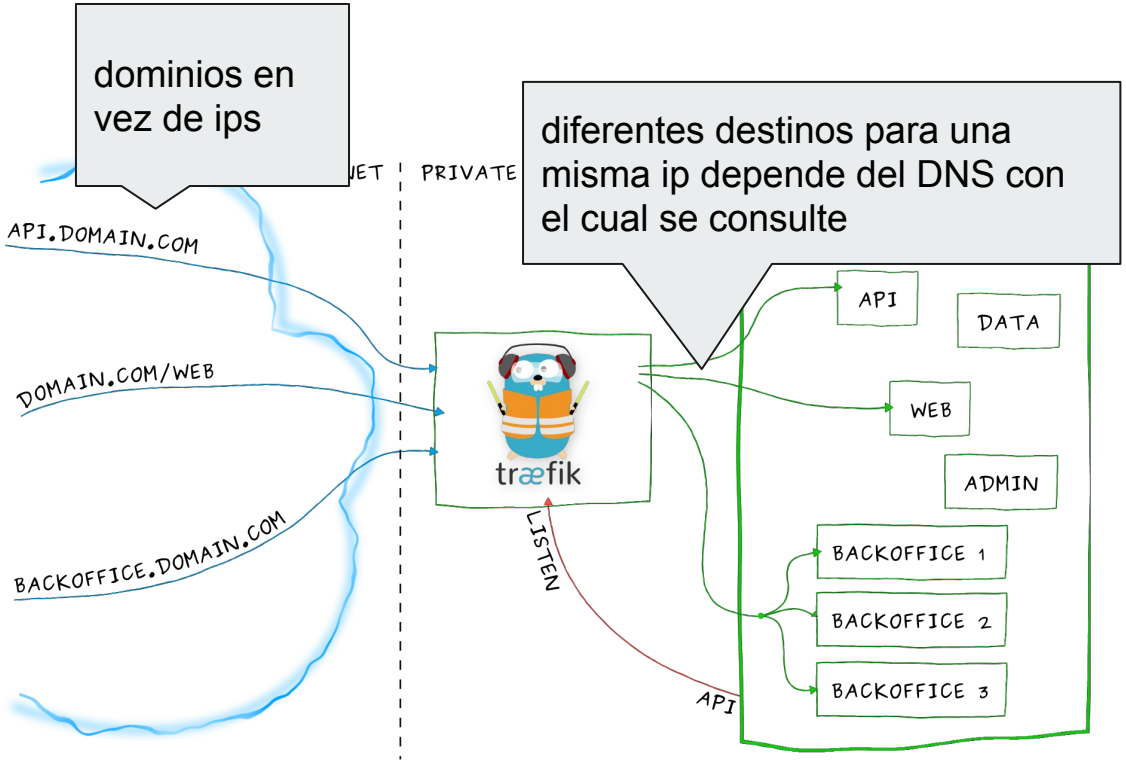
Balanceador de carga y DNS resolver dinámico



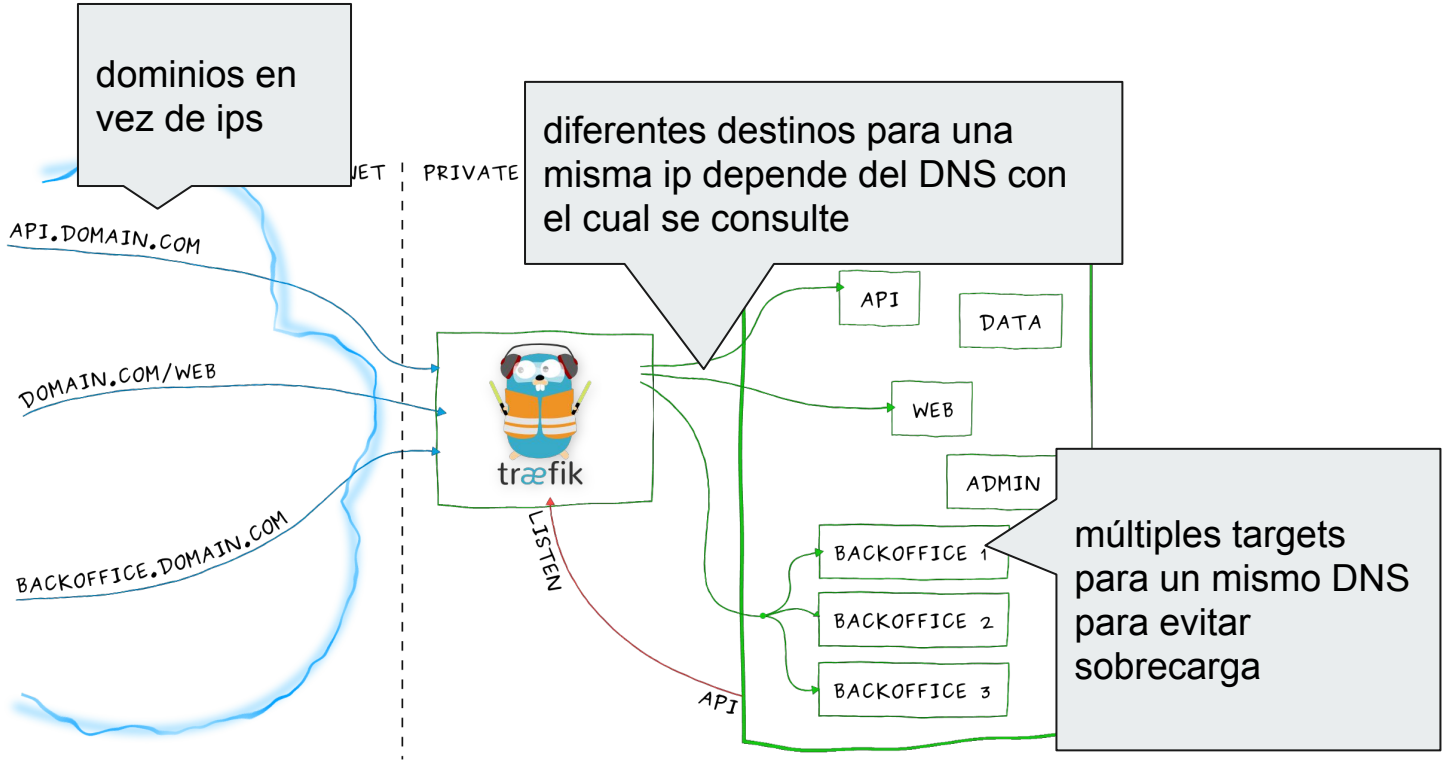
Balanceador de carga y DNS resolver dinámico



Balanceador de carga y DNS resolver dinámico



Balanceador de carga y DNS resolver dinámico

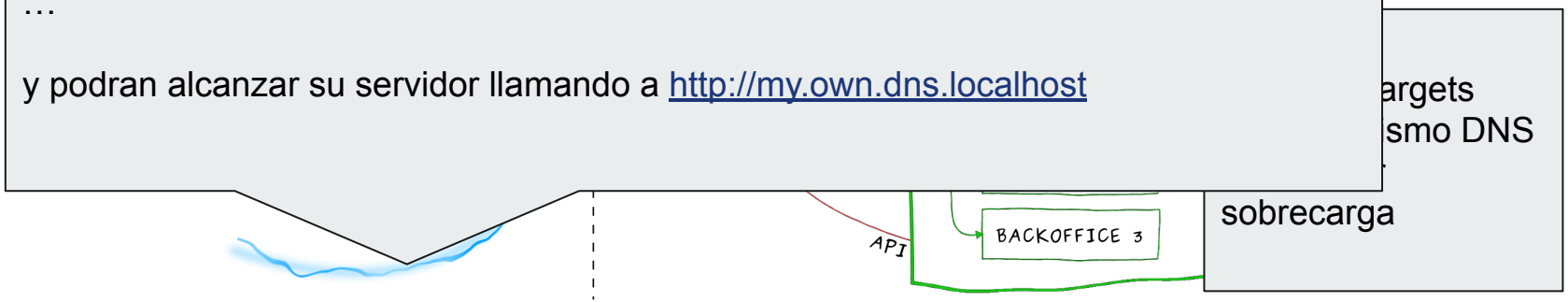


Balanceador de carga y DNS resolver dinámico

Todo solo instanciando traefik y agregando estas labels en su contenedor destino..

```
...
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.api.rule=Host(`my.own.dns.localhost`)"
      - "traefik.http.routers.api.entrypoints=web"
...
```

y podran alcanzar su servidor llamando a http://my.own.dns.localhost



Balanceador de carga y DNS resolver dinámico

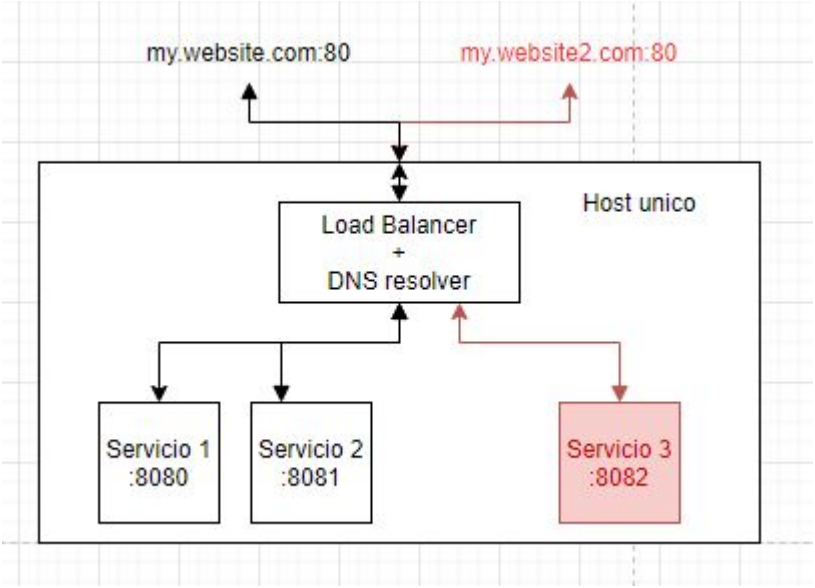
```yaml
version: "3.3"

services:

  traefik:
    image: "traefik:v2.3"
    container_name: "traefik"
    command:
      #- "--log.level=DEBUG"
      - "--api.insecure=true"
      - "--providers.docker=true"
      - "--providers.docker.exposedbydefault=false"
      - "--entrypoints.web.address=:80"
    ports:
      - "80:80"
      - "8080:8080"
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock:ro"

  whoami:
    image: "traefik/whoami"
    container_name: "simple-service"
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.whoami.rule=Host(`whoami.localhost`)"
      - "traefik.http.routers.whoami.entrypoints=web"
```

Mas info

https://doc.traefik.io/traefik/user-guides/docker-compose/basic-example/

Comunicación entre servicios

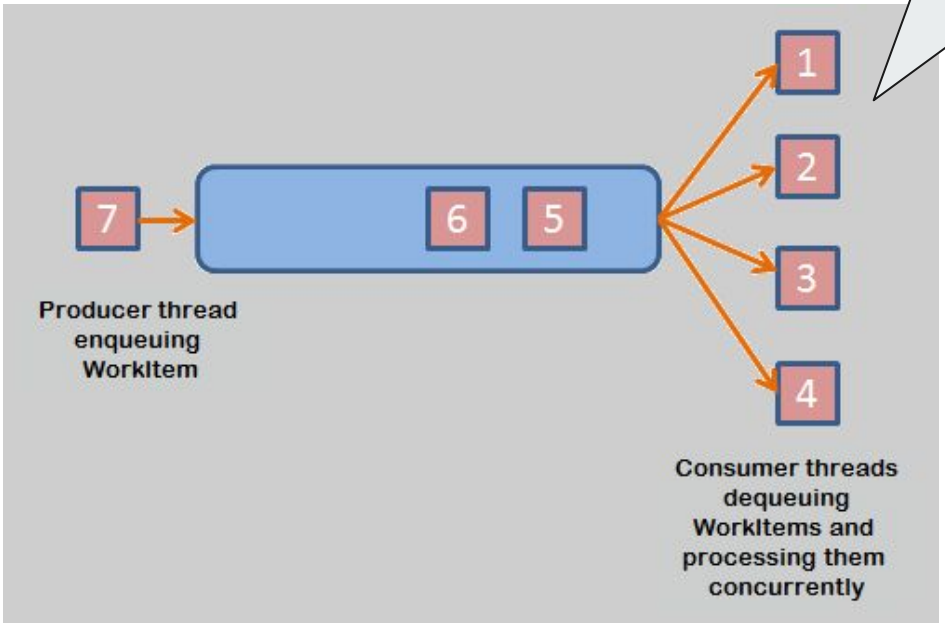
Hasta aquí tenemos lo siguiente



Comunicación (colas de mensajes)

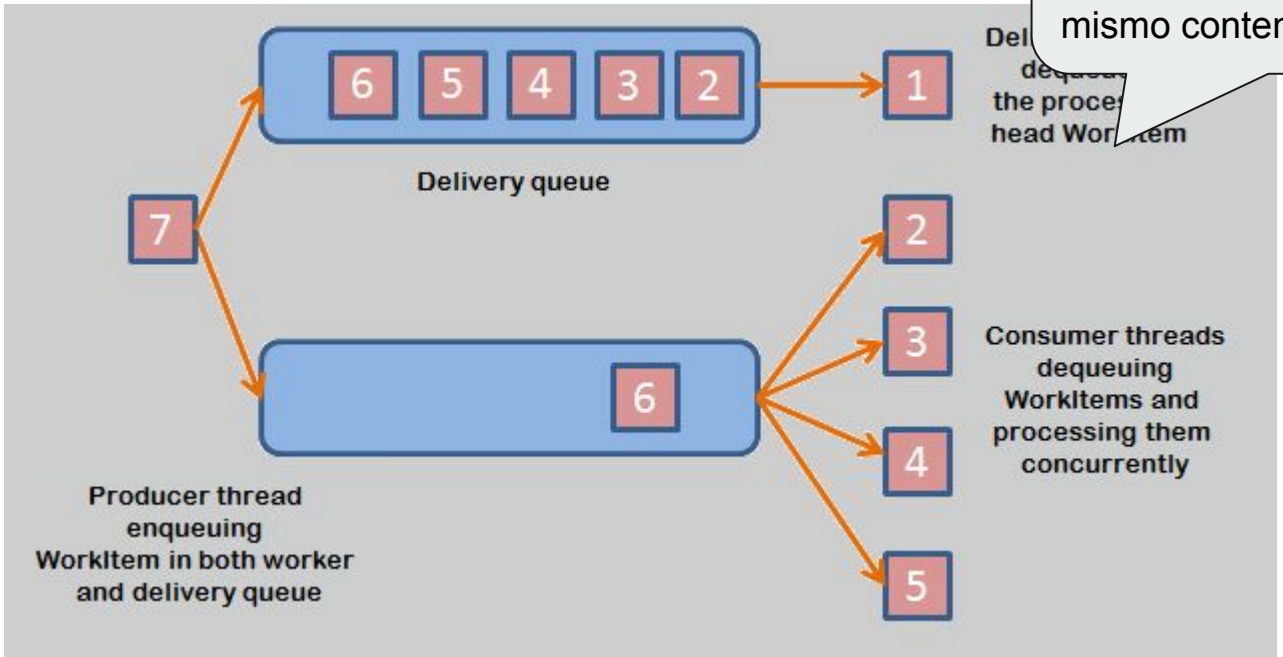
Comunicación (colas de mensajes)

Cómo hacemos para evitar solapamientos en los distintos consumidores?



Comunicación (colas de mensajes)

Cómo hacemos para asegurar que todos los grupos de consumidores consuman el mismo contenido?



Comunicación (conceptos a comparar)

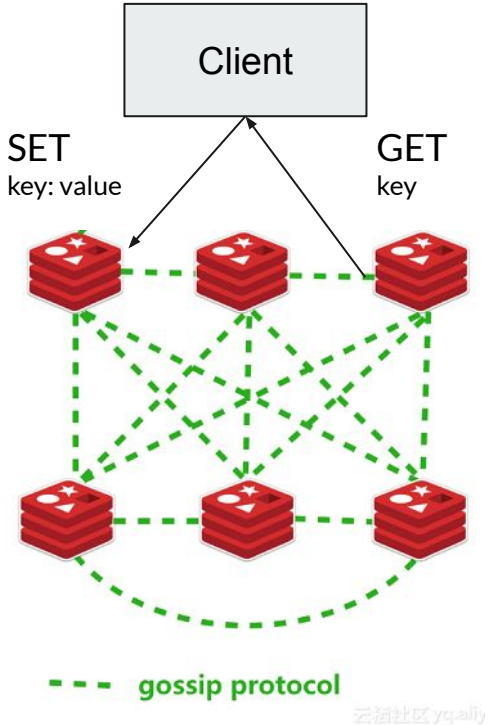
- **Método de suscripción**: Pull-based o Push-based
- **Paralelismo:** es o no thread-safe
- **Retención de mensajes:** Si tiene persistencia de datos o es in-memory
- **Replicación:** Para garantizar la resiliencia de los datos.
- **Velocidad:** Rapidez en el consumo y publicación de mensajes
- **Cantidad de datos:** Cantidad de datos que es capaz de manejar y si es escalable.
- **Casos de uso.**

Comunicación: Redis

No es una cola de mensajes aunque muchas veces se use para intercomunicar servicios. Es una **base de datos** de tipo NoSQL que nos permite almacenar **valores** los cuales se accedan a través de una **clave.**

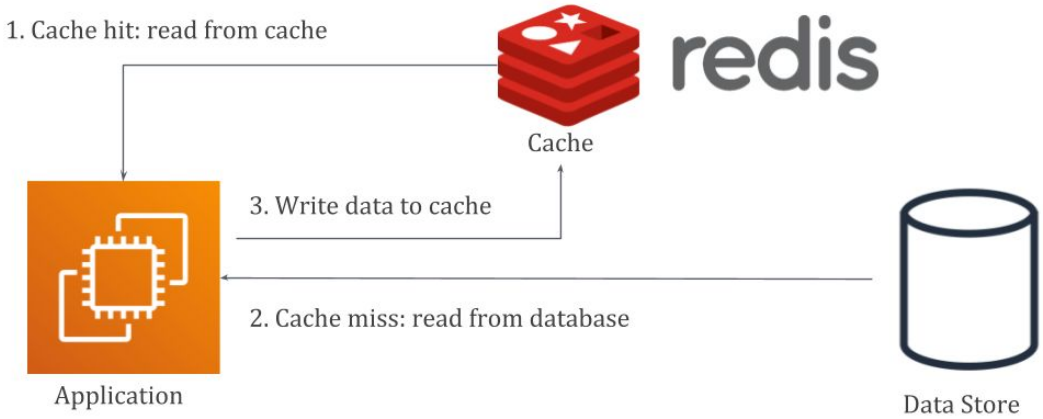
Almacenamiento: en memoria temporal
Velocidad: La más alta
Arquitectura: simple aunque con replicación y distribución en modo cluster
Thread-safe: no por defecto.
Curva de aprendizaje: baja

Comunicación: Redis



Comunicación: Redis

Muchas veces es utilizado como **cache** debido a que podríamos alojar con un TTL determinado la respuesta de cada una de las requests. De esta manera ante una misma petición solo deberíamos devolver la respuesta ya computada evitando recalcular los ítems para la respuesta.



Comunicación: RabbitMQ

Cola de mensajes versátil y escalable. Mayormente usada como broker para productor/consumidor para información **transaccional (eg: en un e-commerce las compras/ventas realizadas).**

Almacenamiento: persistente y replicado
Velocidad: Alta
Arquitectura: dumb consumer/ smart broker
Thread-safe: Si.
Curva de aprendizaje: mediana
Cantidad garantizada de mensajes permitidos: entre 4k/sec y 10k/sec
Retención: mediante ACK.

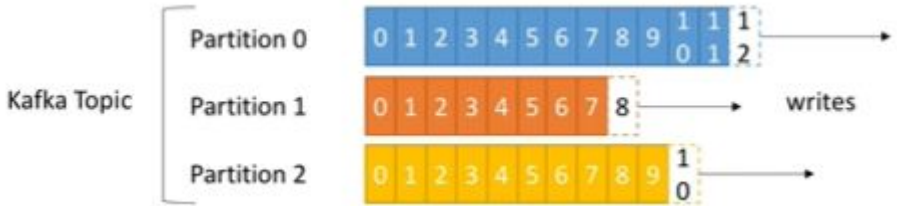
Colas de mensajes: Kafka

Cola de mensajes robusta y confiable. Con una implementación más complicada para un despliegue doméstico, es mayormente usada como broker en sistemas más grandes para información **operacional(eg: en un e-commerce información sobre productos visitados o el tracking de un cliente).** Su fuerte resiliencia y confiabilidad frente a grandes volúmenes de datos de una manera eficiente y distribuida.

Almacenamiento: persistente y replicado
Velocidad: Alta
Arquitectura: smart consumer/ dumb broker
Thread-safe: Si.
Curva de aprendizaje: alta
Cantidad garantizada de mensajes permitidos: 1 Millón/sec de mensajes
Retención: mediante políticas.
Rebalanceo: si

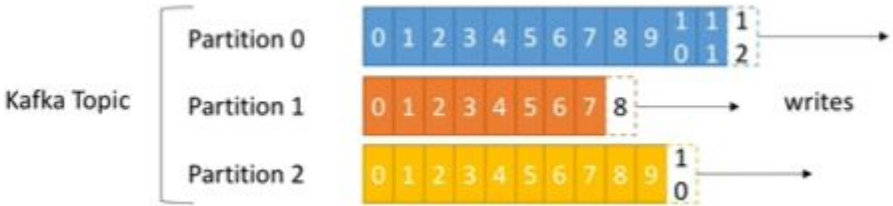
Colas de mensajes: Kafka

Un **tópico** es similar a un tema o a una tabla de información.
Cada tópico tiene **particiones** es decir unidades en las cuales es dividido el tópico.
Y cada partición tendrá un **offset** que indicará el último índice leído de cada partición.
Todos los datos en cada partición son **inmutables.**



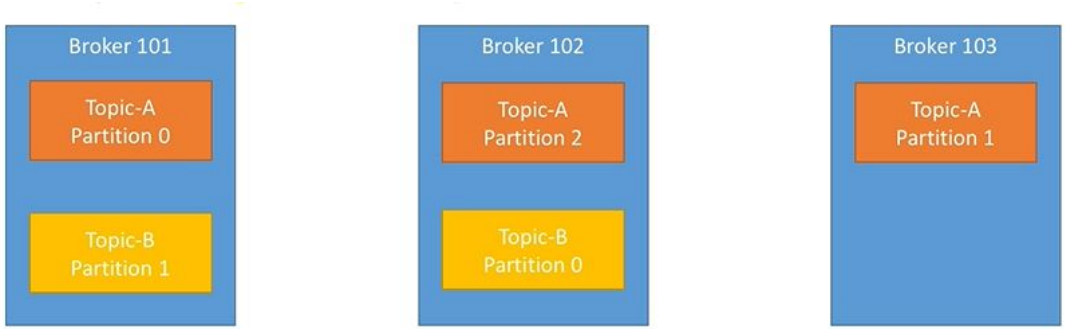
Colas de mensajes: Kafka

Los datos en cada partición son asignados indistintamente es decir al publicar algo en el tópico no sabremos a qué partición se asignará. De esta forma **no podremos asegurar el orden de los mensajes entre las particiones.** Es decir podría ser que el mensaje en la posición 7 de la partición 0 haya sido publicado antes que el mensaje 10 de la partición 2



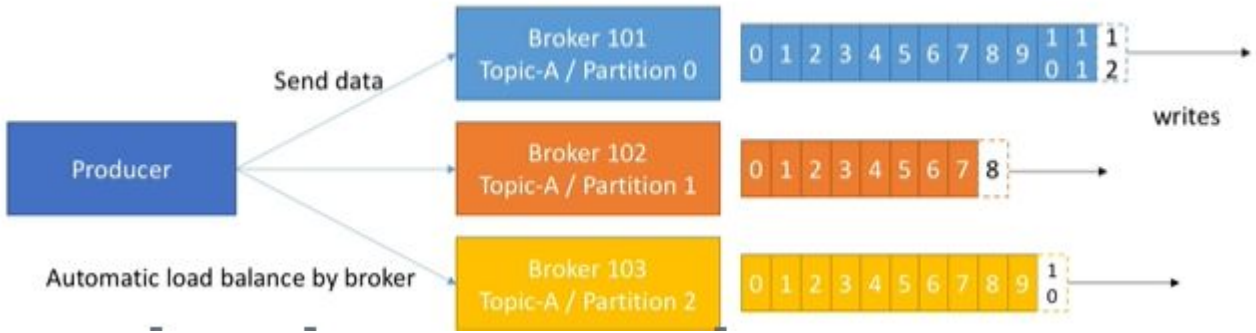
Colas de mensajes: Kafka

Un **broker** será asignado a cada partición de cada tópico de manera tal que todos conozcan la totalidad de la información aunque cada uno será responsable de su partición y hará tanto lecturas como escrituras a ese mismo slot de información.



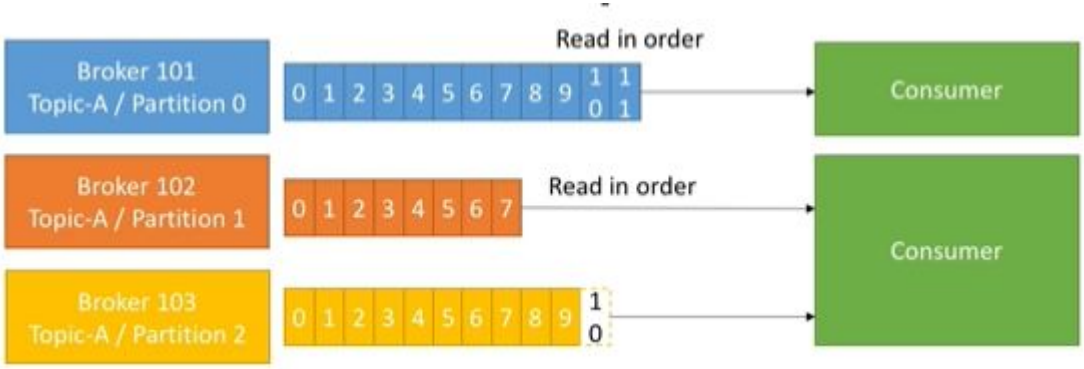
Colas de mensajes: Kafka

Un **productor** publicará a través de un broker en cualquiera de las particiones de un tópico de manera aleatoria al menos que etiquete el mensaje con una **clave.** Los mensajes que se emitan bajo esta misma clave serán destinados siempre a la misma partición.



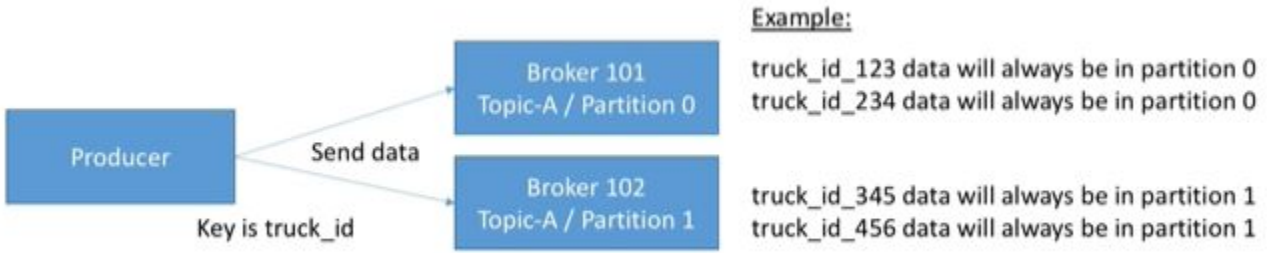
Colas de mensajes: Kafka

Un **consumidor** se conectará a uno o más brokers para obtener la información de manera ordenada. Pues si bien entre particiones no podemos asegurar el orden de la información si **podemos asegurar el orden dentro de una misma partición**. De esta forma si el productor emitió la información con una etiqueta particular entonces el sistema, independientemente de qué broker tome el mensaje, **asegurará el procesamiento ordenado de los mensajes etiquetados con dicha clave**.



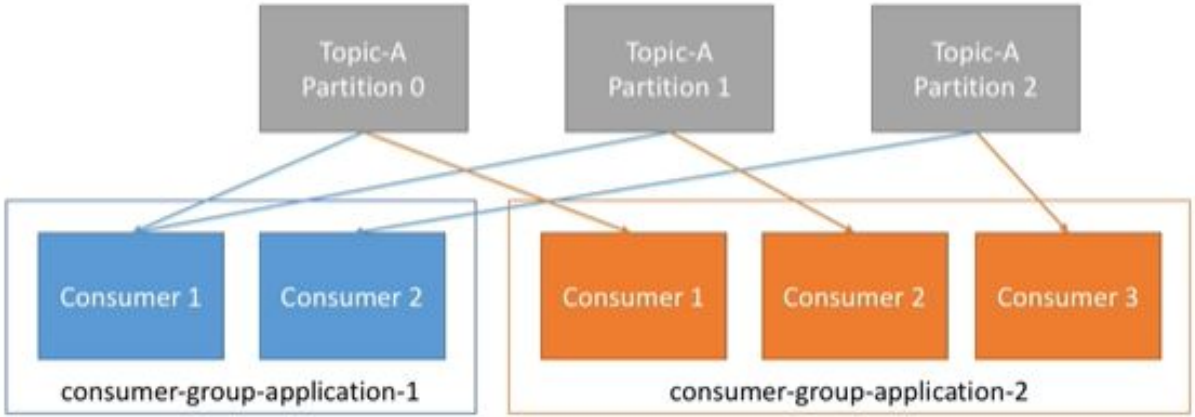
Colas de mensajes: Kafka (ejemplo)

Dado una flota de camiones que tengan un GPS el cual reporta su ubicación en tiempo real.
Cada camión debería emitir sus coordenadas etiquetadas bajo la clave **truck_id.**
De esta forma veremos que si tenemos 4 camiones 123, 234, 345 y 456 y nuestro tópico "truck_coordinates" con 2 particiones veremos que esta podría ser una disposición posible.



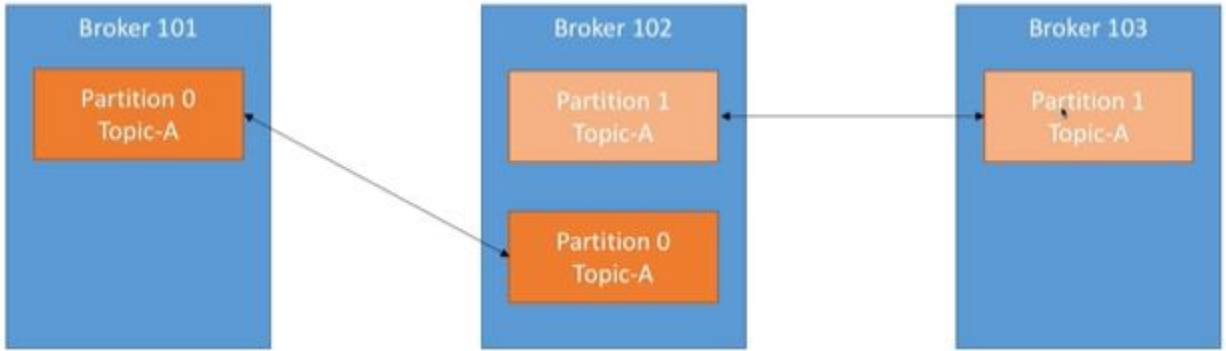
Colas de mensajes: Kafka (ejemplo)

Siguiendo con el ejemplo vemos que del otro podemos tener **grupos de consumidores** que hagan distintas tareas. Los diversos grupos de consumidores pueden ser distintos en su cardinalidad y en caso de ser menos que las particiones aceptaran la lectura proveniente de más de una partición a la vez. Recordemos que esto no supone un desorden en la lectura puesto que las particiones preservan orden.



Colas de mensajes: Kafka

Kafka además cuenta con un sistema de **replicación** para la persistencia de la información incluso ante fallos de uno o más brokers.



Colas de mensajes: Kafka

Ante la falla del broker 102 no perderíamos la información ni de la partición 0 ni de la partición 1 pues ambas pueden encontrarse en los brokers 101 y 103 respectivamente

Colas de mensajes: Kafka + Zookeeper

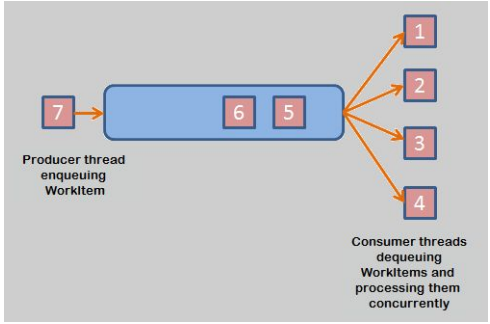
Ante la falla del broker 102 no perderiamos l… la partición 1 pues ambas pueden encontrar… te

Actividad 1

Configuración de kafka

Actividad 1

El objetivo de esta actividad es entender cómo configurar un cluster de Kafka de un solo nodo con Zookeeper. Una vez instanciado el cluster enviar mensajes con un productor y consumir los mensajes con al menos 2 grupos de consumidores para entender el funcionamiento.



Opcional: Usar además un cliente python para consumir del cluster. Pueden usar jupyter notebooks para esto.

Hint

- Instalar java8
- Descargar kafka desde https://kafka.apache.org/downloads
- Descomprimir los archivos y agregar el path en bashrc (opcional)
- Inicializar zookeeper cambiando la configuración para persistencia de datos
- Inicializar kafka cambiando la configuración para persistencia de datos.
- Utilizar el CLI incluido como training para crear tópicos producir y consumir contenido

- Resolucion: https://github.com/matisilva/kafka-kickstart

<FIN DE CLASE>