



UNIVERSIDAD TECNICA  
FEDERICO SANTA MARIA

Departamento de Informática

# Comunicación y Virtualización

## VIRTUALIZACION Y CONTENEDORES

PH.D. Erika Rosas Olivos

[erosas@inf.utfsm.cl](mailto:erosas@inf.utfsm.cl)

Septiembre 2023



# ¿Qué es virtualización?

La virtualización usa software para crear una capa de abstracción sobre el hardware del computador, permitiendo que los elementos de hardware de un solo computador (procesador, memoria y almacenamiento) sean divididos en múltiples computadores virtuales, comúnmente llamados máquinas virtuales.

- Es la tecnología que impulsa la economía de la computación en la nube.



**Ref. [ibm.com](http://ibm.com)**



# Propiedades de la virtualización

## Aislamiento:

- Aislamiento de fallas
- Aislamiento de software
- Aislamiento de rendimiento

## Encapsulación:

- El estado de la VM puede ser capturado en un archivo.
- La complejidad es proporcional al hardware y es independiente de la configuración de software invitado.

## Interposición:

- Todas las acciones del invitado pasan a través del software de virtualización, que puede inspeccionar, modificar y denegar operaciones.



# ¿Cuáles son los beneficios de la virtualización?

## **Eficiencia de recursos:**

- La virtualización permite correr varias aplicaciones en una maquina física sin sacrificar confiabilidad.

## **Administración simplificada:**

- Administración de políticas por software.
- Se pueden crear plantillas en software para aplicaciones y máquinas virtuales.

## **Tiempo de inactividad mínimo:**

- Se pueden ejecutar VMs redundantes para ejecutar commutación por falla (failover)

## **Aprovisionamiento rápido:**

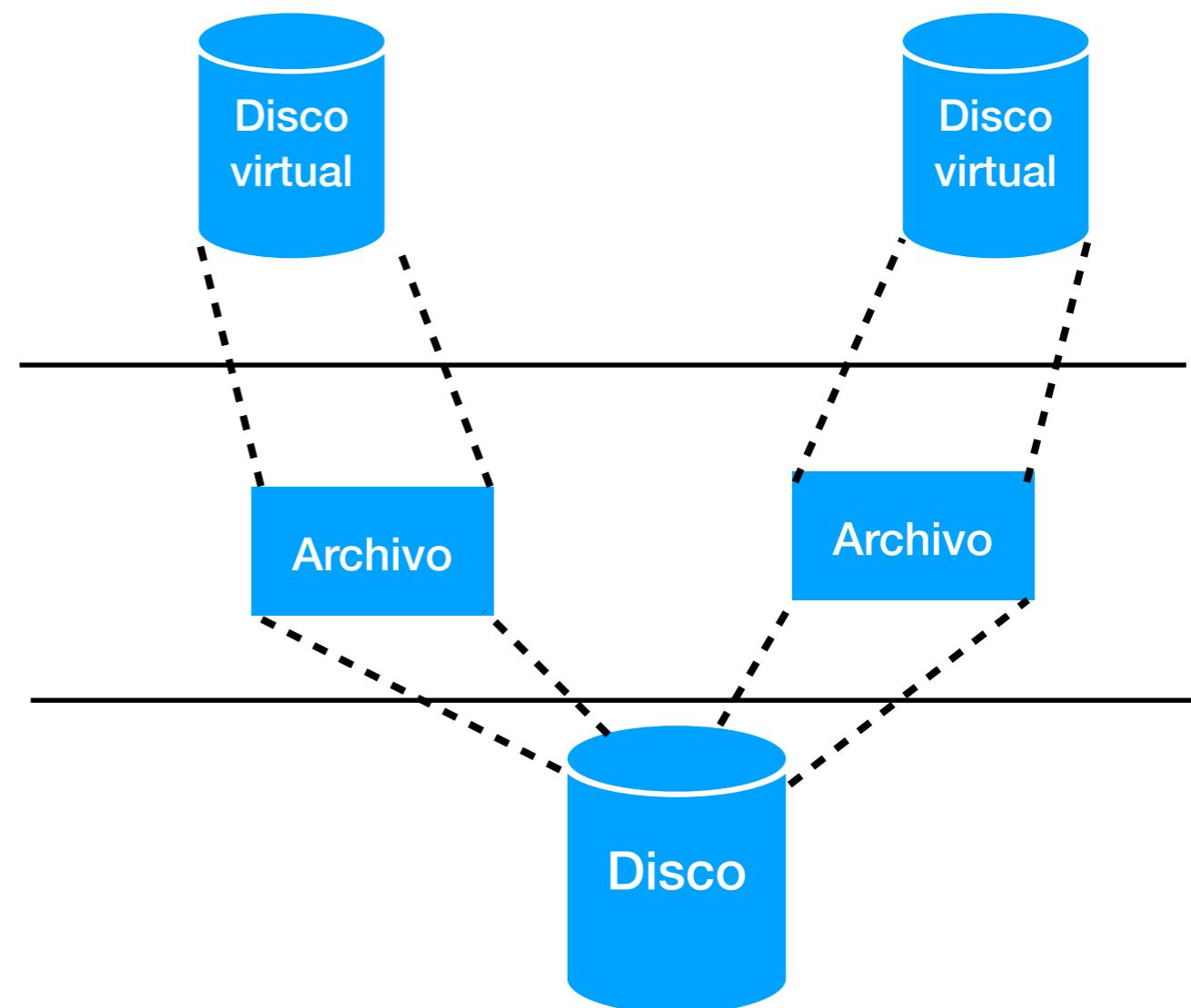
- Más rápido crear VM que comprar, instalar y configurar una máquina física para una aplicación.



# ¿Qué hace posible la virtualización?

## Abstracción:

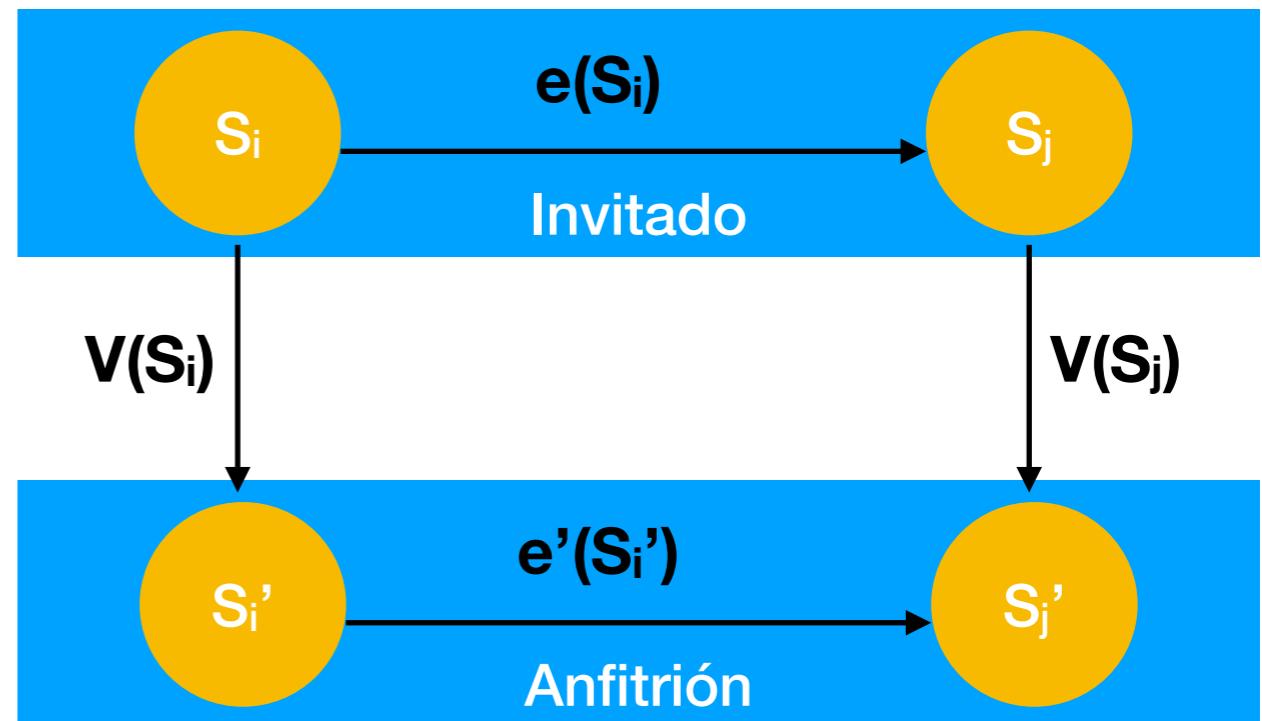
- Lo clave para manejar la complejidad de sistemas es la división en niveles de abstracción, separados por interfaces bien definidas.
- La virtualización usa abstracción, pero no oculta detalles, generalmente es el mismo detalle que el sistema anfitrión.



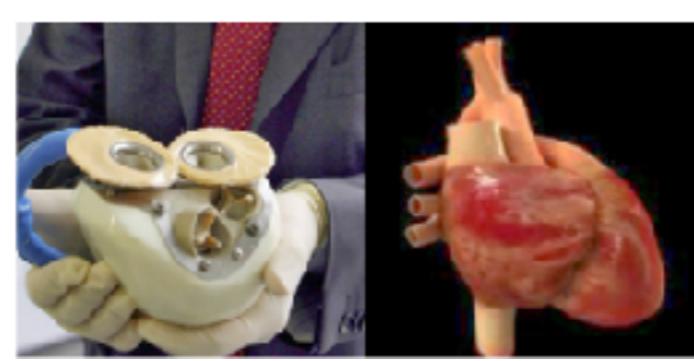


# ¿Qué es la virtualización?

- Formalmente, la virtualización involucra la construcción de un isomorfismo que mapea un sistema *invitado* virtual a un sistema *anfitrión* real



Isomorfismo: construcción de modelos de sistemas similares al original.



- Función  $V$  mapea el estado del invitado al estado del anfitrión.
- Para una secuencia de operaciones  $e$ , que modifican el estado del invitado, hay operaciones correspondientes  $e'$  en el anfitrión que realizan una modificación equivalente.



# ¿Qué es la virtualización?

- Los sistemas operativos usan instrucciones especiales que no están disponibles para aplicaciones.
- Un procesador es virtualizable cuando el acceso a estas instrucciones causa un error que software con acceso privilegiado puede interceptar o capturar.
- La mayoría de las instrucciones se ejecutan directamente, sin necesidad de emulación, por eso estos sistemas tienen buen rendimiento en la actualidad.

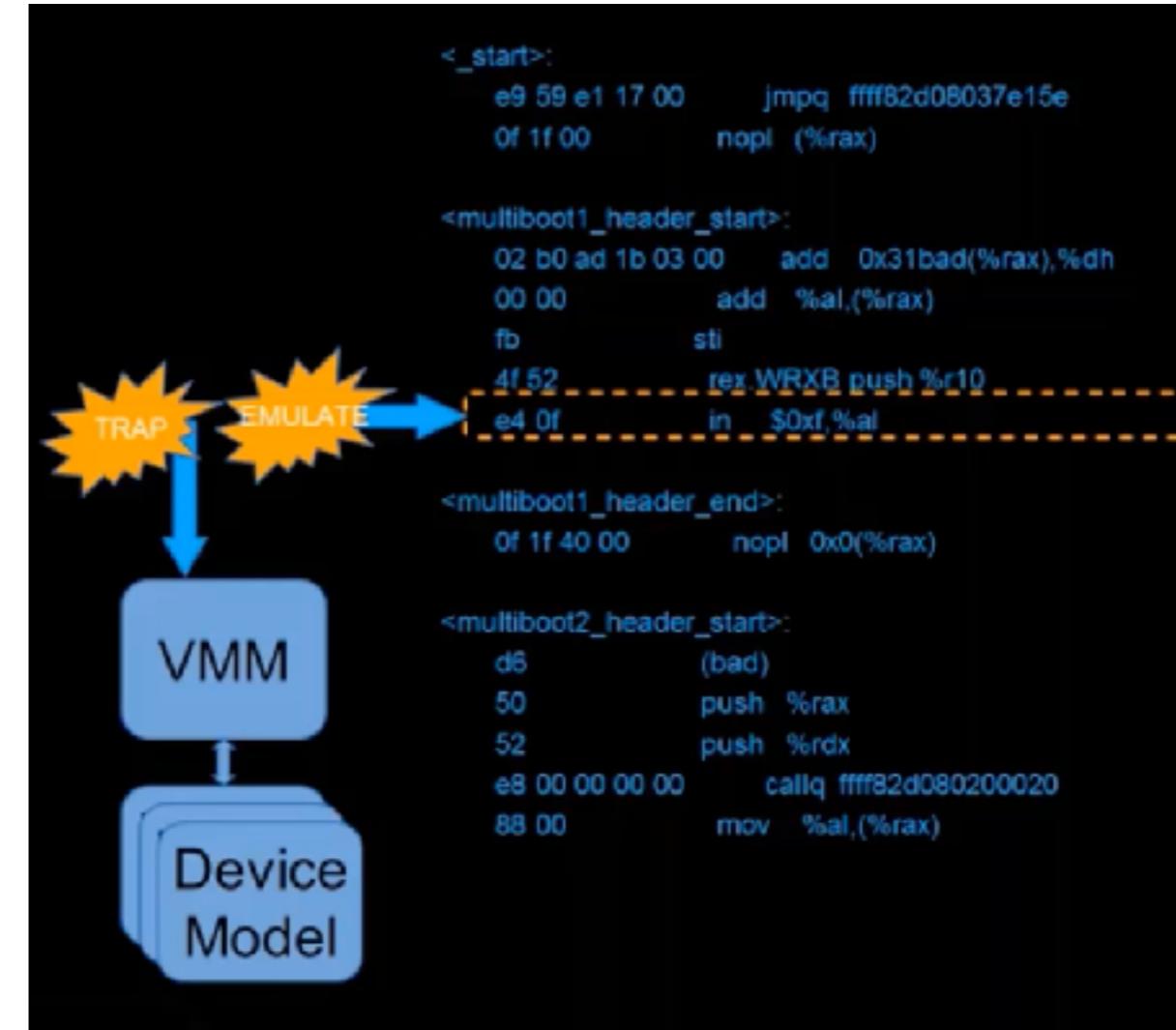


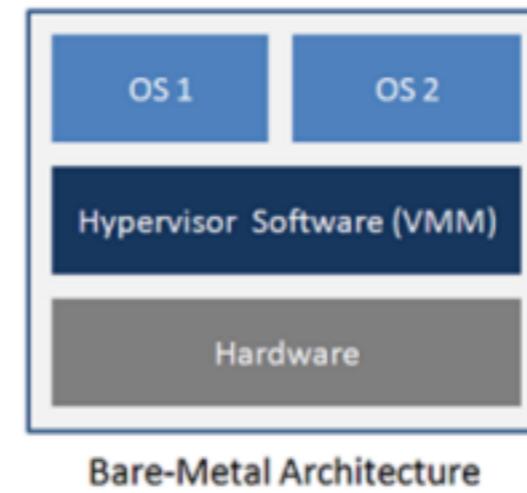
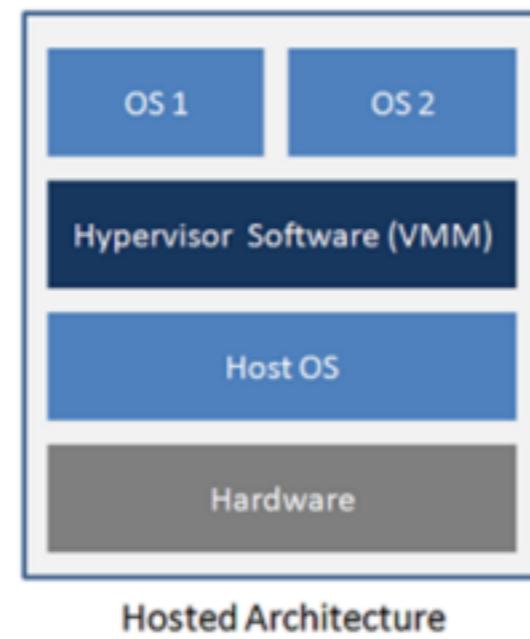


# Hypervisors

Software usado para virtualizar el OS.

- Monitor de la máquina virtual (VMM):
  - Captura y emula el conjunto de instrucciones privilegiadas del sistema
- Device Model (10 a 100s): Usado para virtualizar los dispositivos de I/O.
- Scheduler, memory manager, etc.





- Este enfoque virtualiza el sistema operativo completo.
- La abstracción son dispositivos virtuales como discos virtuales, CPU virtuales, tarjetas de red virtuales.
- Múltiples sistemas operativos pueden correr en el mismo hardware.
- Las máquinas virtuales corren en un espacio de nombres virtuales aislado por hardware con menos privilegios que el host anfitrión.

Ref <https://www.ni.com/white-paper/9629/en/>



# Paravirtualización

- Paravirtualización modifica la captura de instrucciones del Sistema operativo.
- Usa Hypercall invocan directamente la VMM.
- El sistema operativo invitado no está completamente aislado.
- Es más rápido, pero menos portable y compatible.

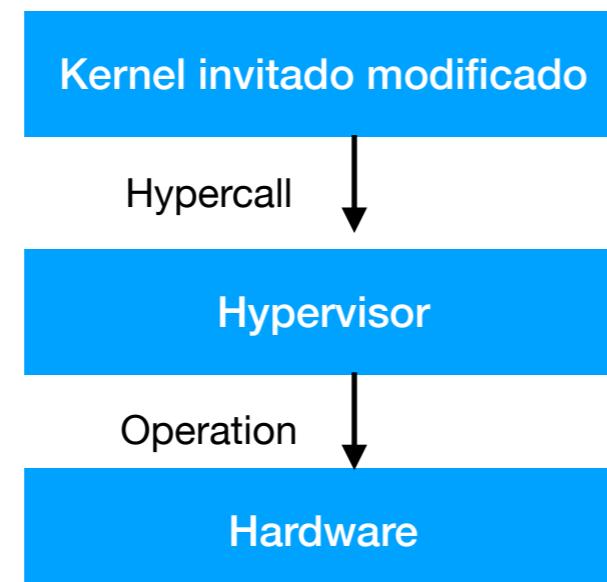
```
<_start>:
e9 59 e1 17 00    jmpq  ffff82d08037e15e
0f 1f 00          nopl  (%rax)

<multiboot1_header_start>:
02 b0 ad 1b 03 00    add   0x31bad(%rax),%dh
00 00              add   %al,(%rax)
fb                 sti
4f 52              rex.WRXB_push %r10
e4 0f              HYPERCALL io_in

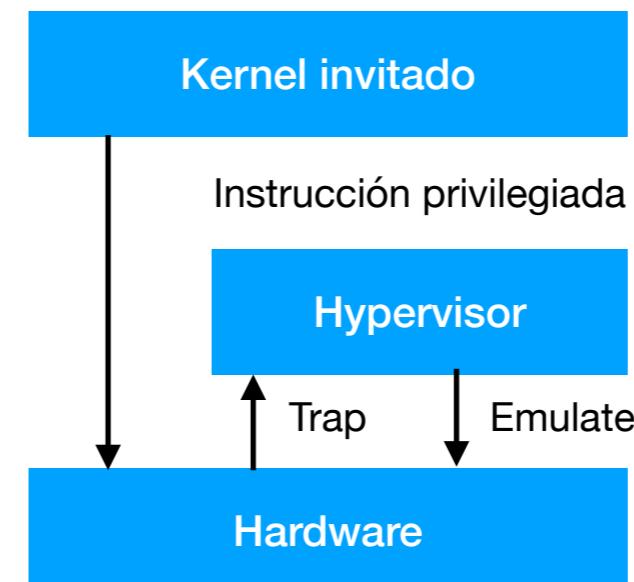
<multiboot1_header_end>:
0f 1f 40 00          nopl  0x0(%rax)

<multiboot2_header_start>:
d6                 (bad)
50                 push  %rax
```

## Paravirtualización



## Virtualización clásica





UNIVERSIDAD TECNICA  
FEDERICO SANTA MARIA

Departamento de Informática

# Introducción a contenedores

- Los contenedores comparten recursos con el OS del anfitrión, mejorando la eficiencia.
- Son rápidos de iniciar y detener.
- Tienen poco overhead comparado con aplicaciones corriendo nítidamente en el OS anfitrión.
- Son portables y livianos.

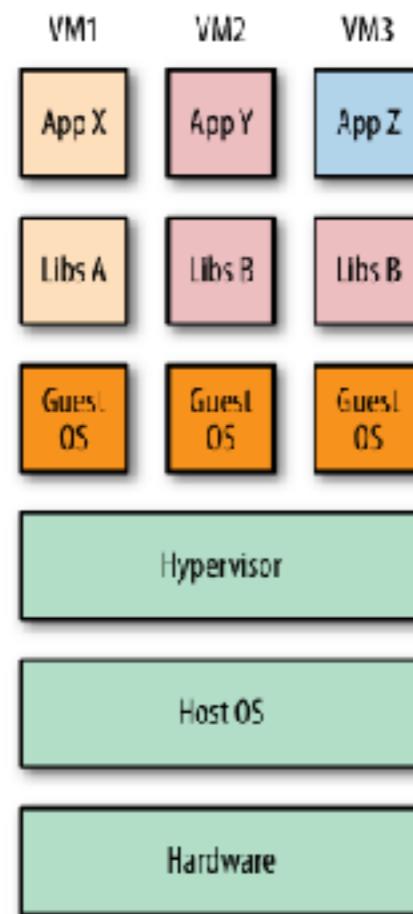




# Introducción a contenedores

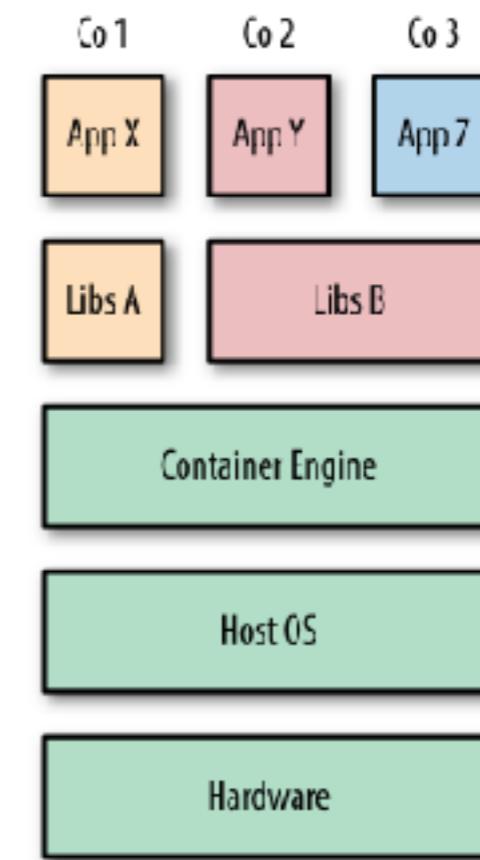
## Máquinas Virtuales

Objetivo: Emular un ambiente foraneo.



## Contenedores

Objetivo: Hacer aplicaciones portables y auto-contenidas.

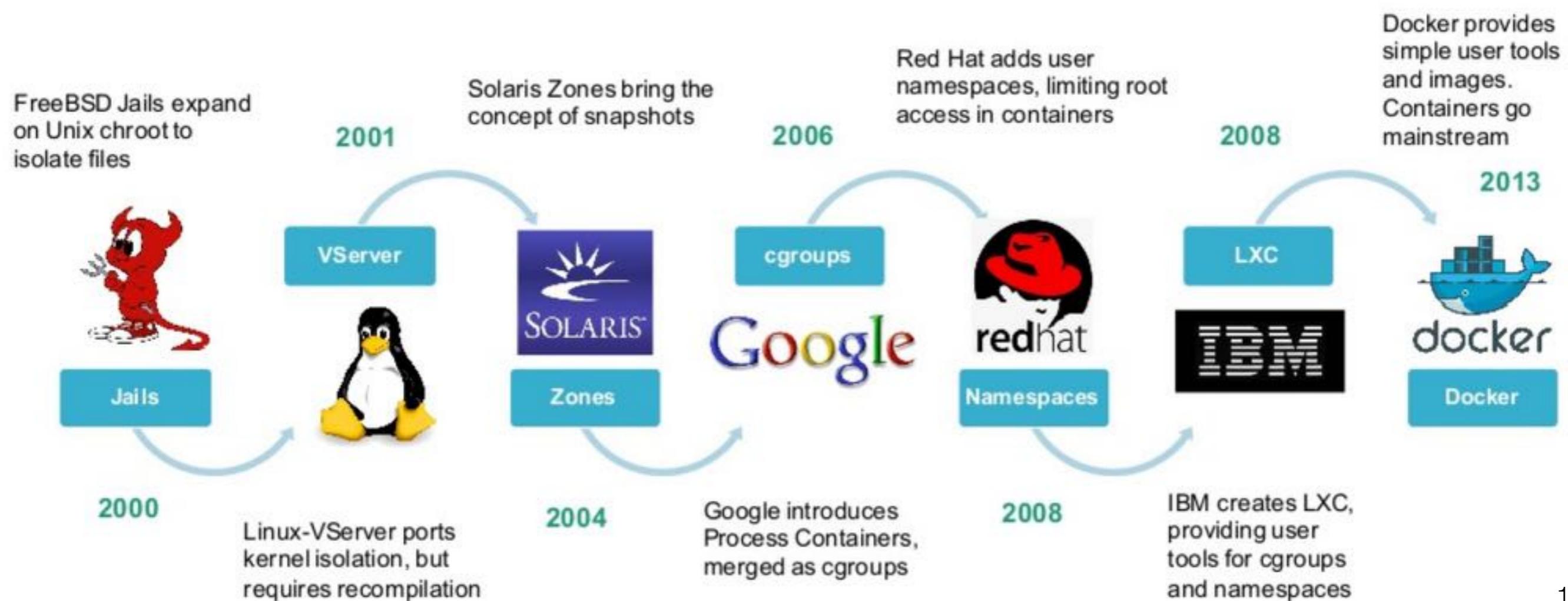




# Introducción a contenedores

Qué tecnologías posibilitaron el crecimiento masivo de contenedores:

- **Kernel namespaces:** El kernel de Linux aísla los accesos a diferentes recursos.
- **Control groups:** Se separa el árbol de procesos, sistema de archivos del root, etc.
- **Union filesystems:** Permite superponer varios sistemas de archivos, apareciendo al usuario como uno.
- Docker: contenedores simples, accesibles a todos.





- La compañía Docker, Inc. : Fundada en San Francisco por Salomon Hykes en el 2013. Partió como PaaS.
- La tecnología Docker: La tecnología que corre contenedores.

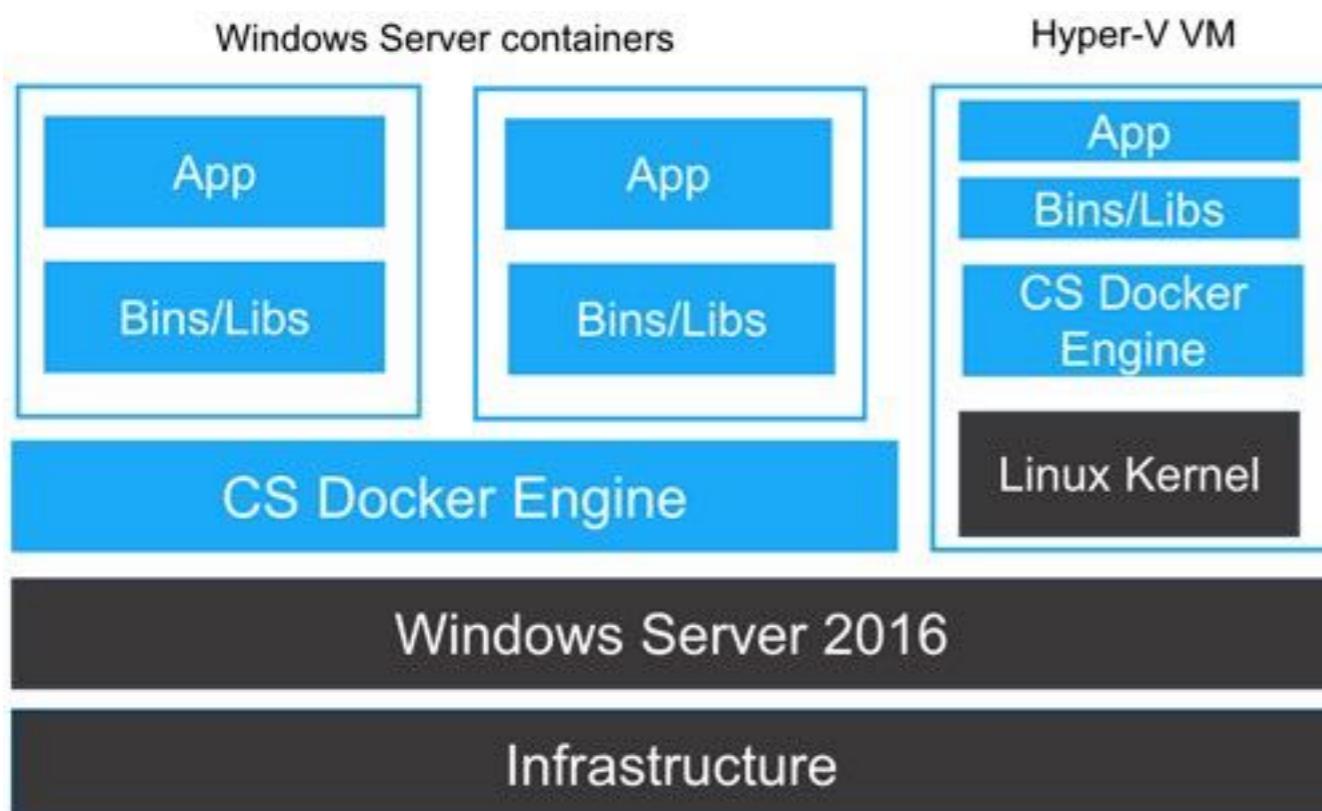
## Docker Momentum





# Windows containers? Mac containers?

- También hay Windows containers
- Los contenedores de Windows deben correr en el OS windows, los contenedores de linux deben correr en el OS Linux.
- Puedo correr contenedores de Linux en infraestructura Windows usando Hyper-V VM

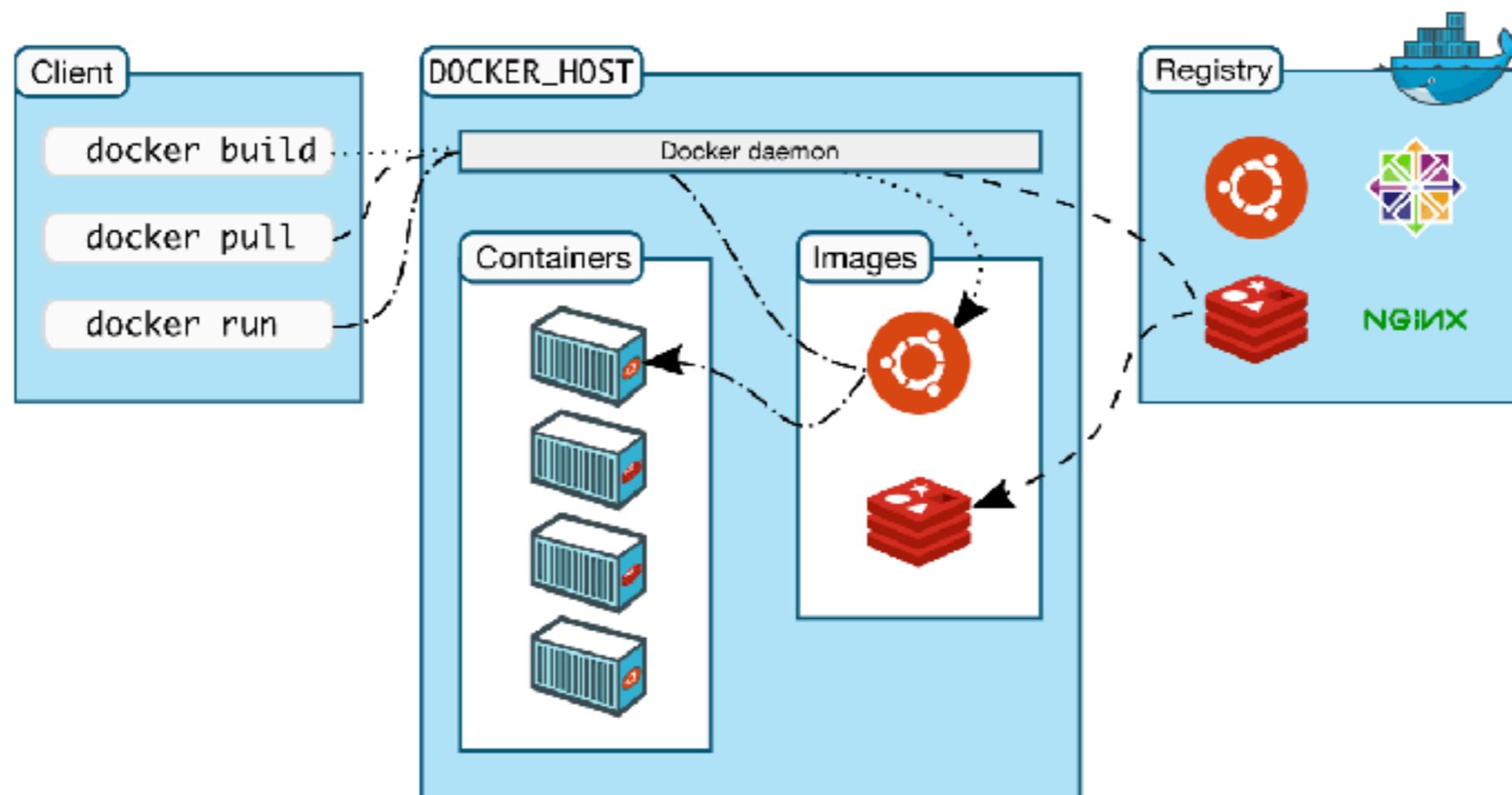


- Contenedores Mac:
  - No existen.
  - Se pueden ejecutar contenedores de linux en Mac usando Docker Desktop.
  - Usa una máquina virtual liviana de linux en Mac: LinuxKit.



# Arquitectura de Docker

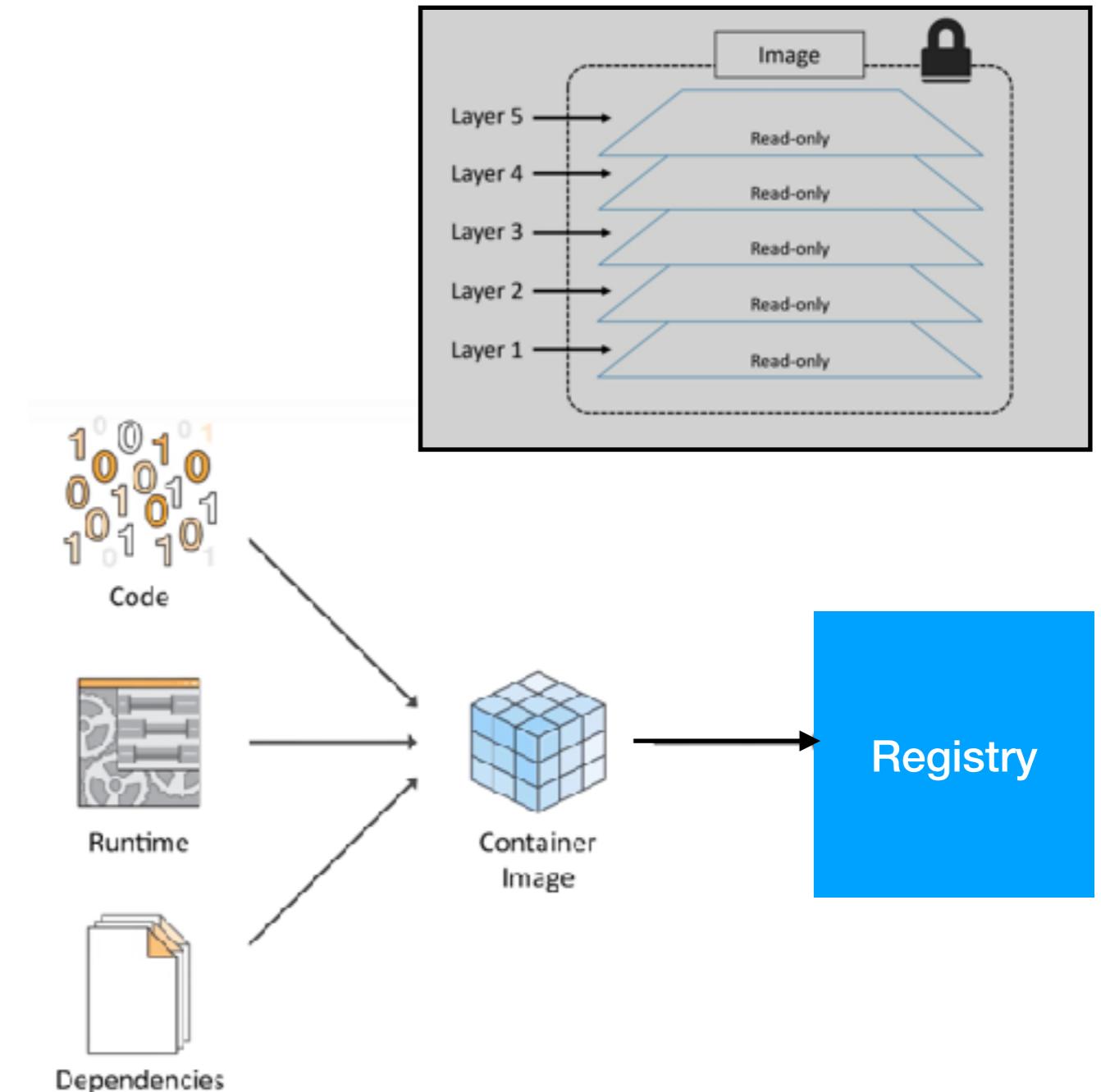
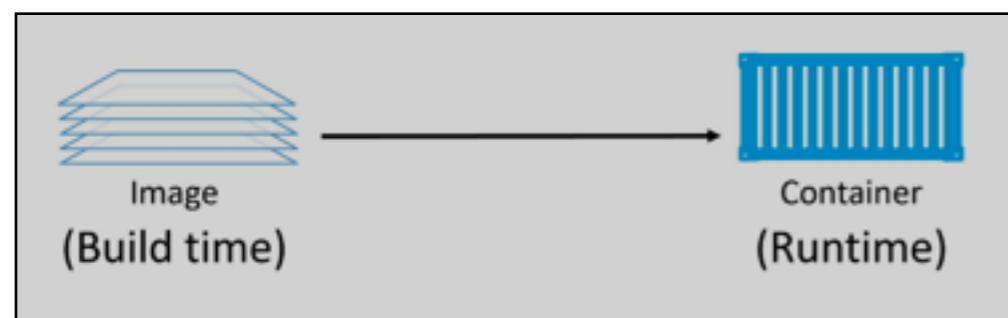
- Usa arquitectura cliente-servidor.
- El cliente y servidor se comunican via API REST sobre sockets de linux.
- Docker registry almacena imágenes de Docker.
  - Docker Hub es un ejemplo de registro público pero hay más.





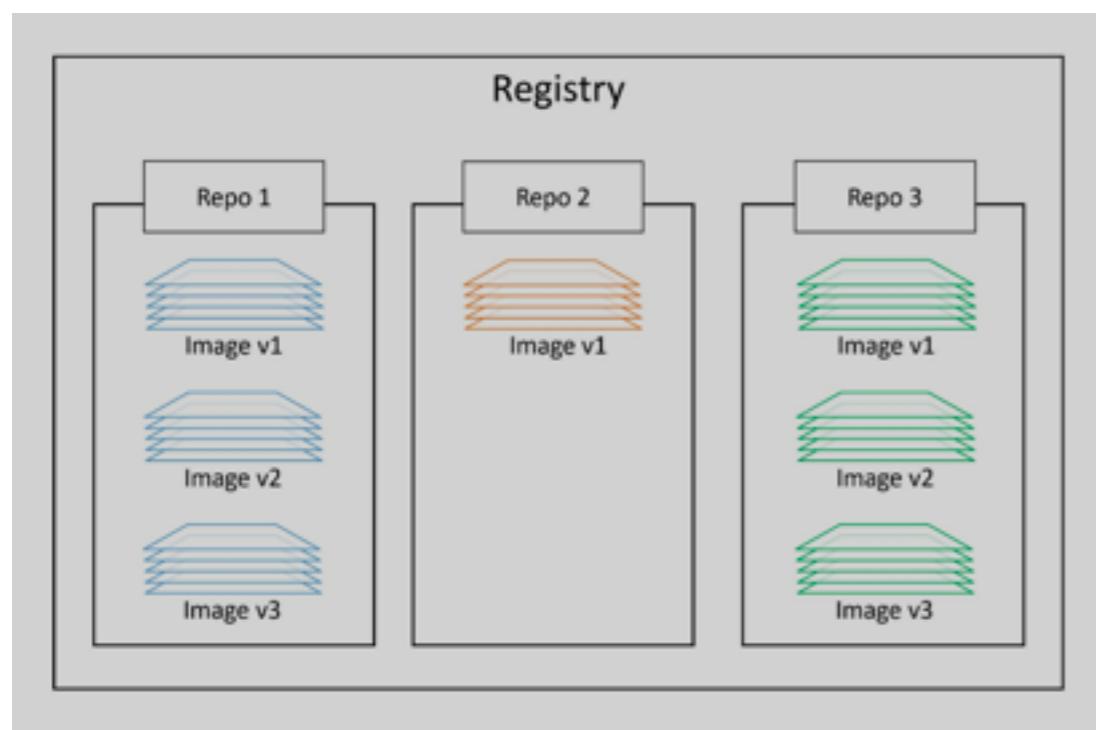
# ¿Qué es un contenedor? ¿Qué es una imagen?

- Un contenedor es una instancia que está siendo ejecutada de una imagen.
- Un *Imagen de Docker* es una unidad de empaquetamiento que contiene todo lo requerido para ejecutar una aplicación.





- Necesitamos imágenes para crear contenedores.
- Las imágenes tienden a ser pequeñas.
- Se puede ver como un objeto compuesto por capas.
- Tenemos que descargar imagen de un registro, por defecto Docker Hub.
- Descarguemos la imagen Alpine Linux Docker que pesa 5MB.



# Manejo de imágenes

<https://labs.play-with-docker.com/>

Listar imágenes del repositorio local

```
$docker image ls
```

Pull image from registry

```
$docker image pull <repository>:<tag>
$docker image pull alpine:latest
```

```
$docker image pull gcr.io/google-containers/
git-sync:v3.1.5
```

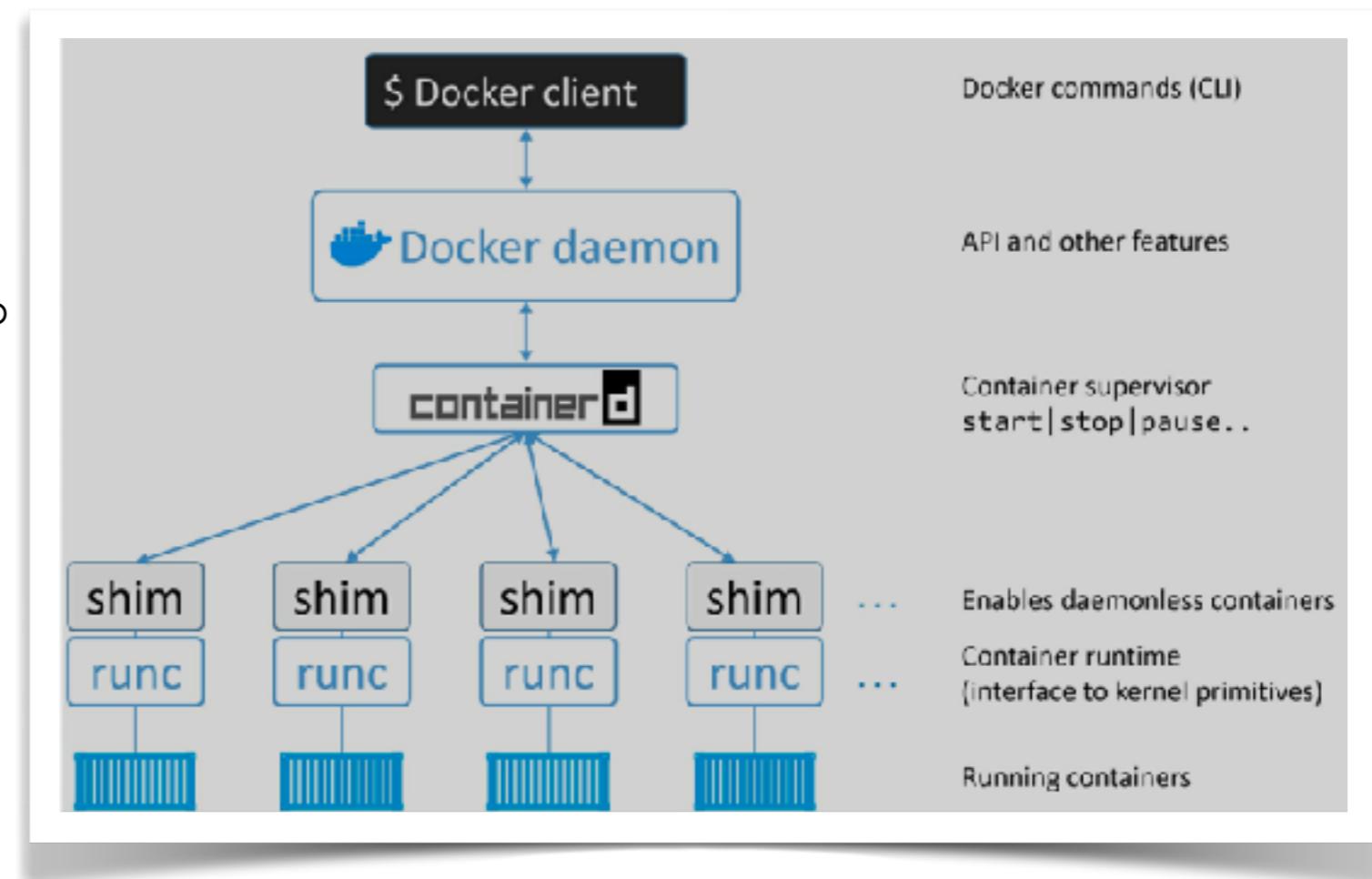
Borrar imagen

```
$docker rmi nombre_image
```



# Docker Engine

- No se ejecutan contenedores en Docker daemon.
- **Docker daemon** realiza administración de imágenes, REST API, autenticación, manejo de red, orquestación.
- **containerd** - función es manejar operaciones en contenedores - start, stop, pause, rm
- **runc** - función es crear contenedores.
- Daemon se comunica con containerd via gRPC.

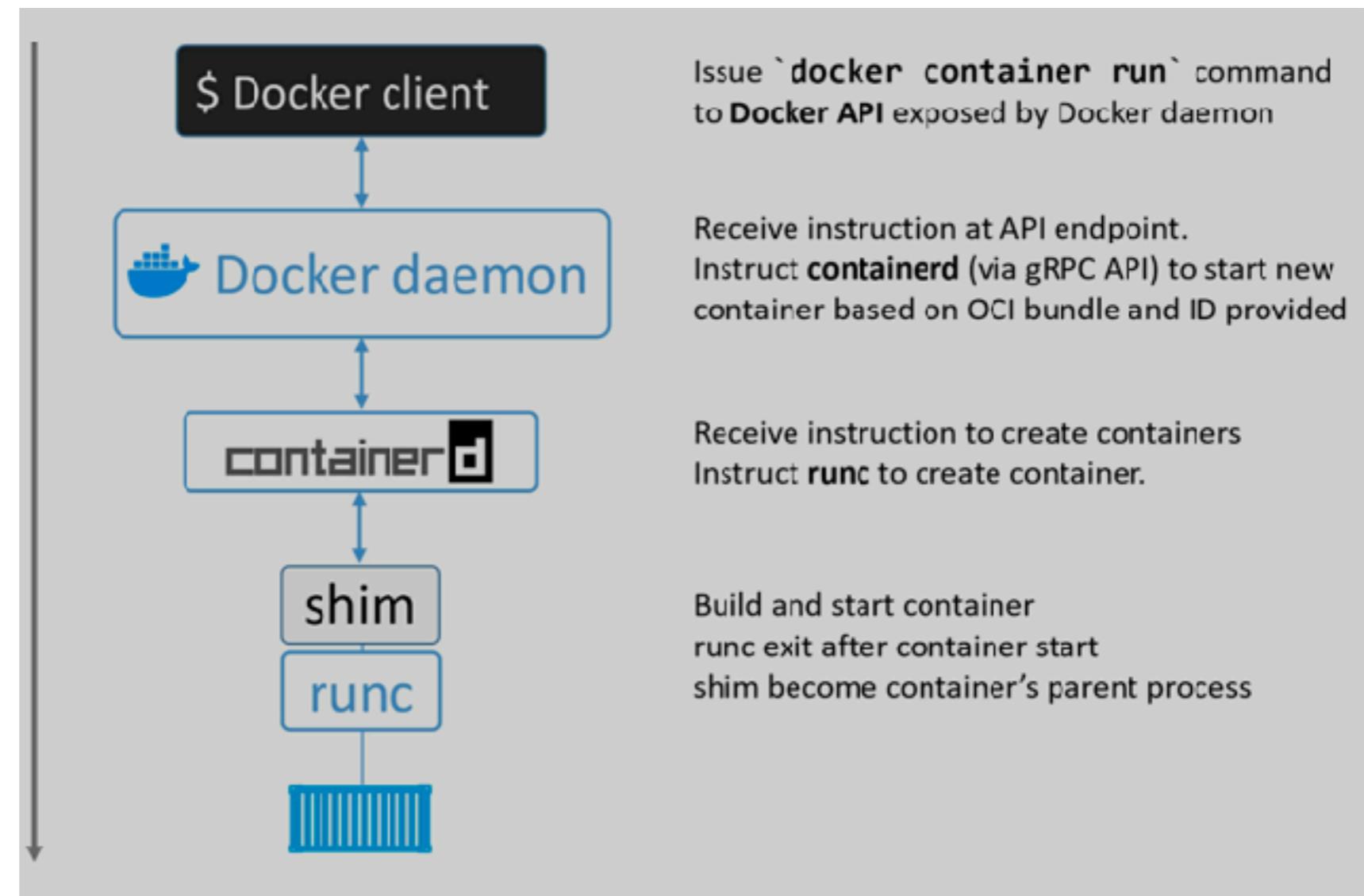


Ref. Docker deep dive, Nigel Poulton, Mayo 2020



# Docker Engine

- Proceso daemon *dockerd*
- Docker API



Ref. Docker deep dive, Nigel Poulton, Mayo 2020



UNIVERSIDAD TECNICA  
FEDERICO SANTA MARIA

Departamento de Informática

Comunicación con Docker  
desde un programa

Comunicación con Docker  
por línea de comando

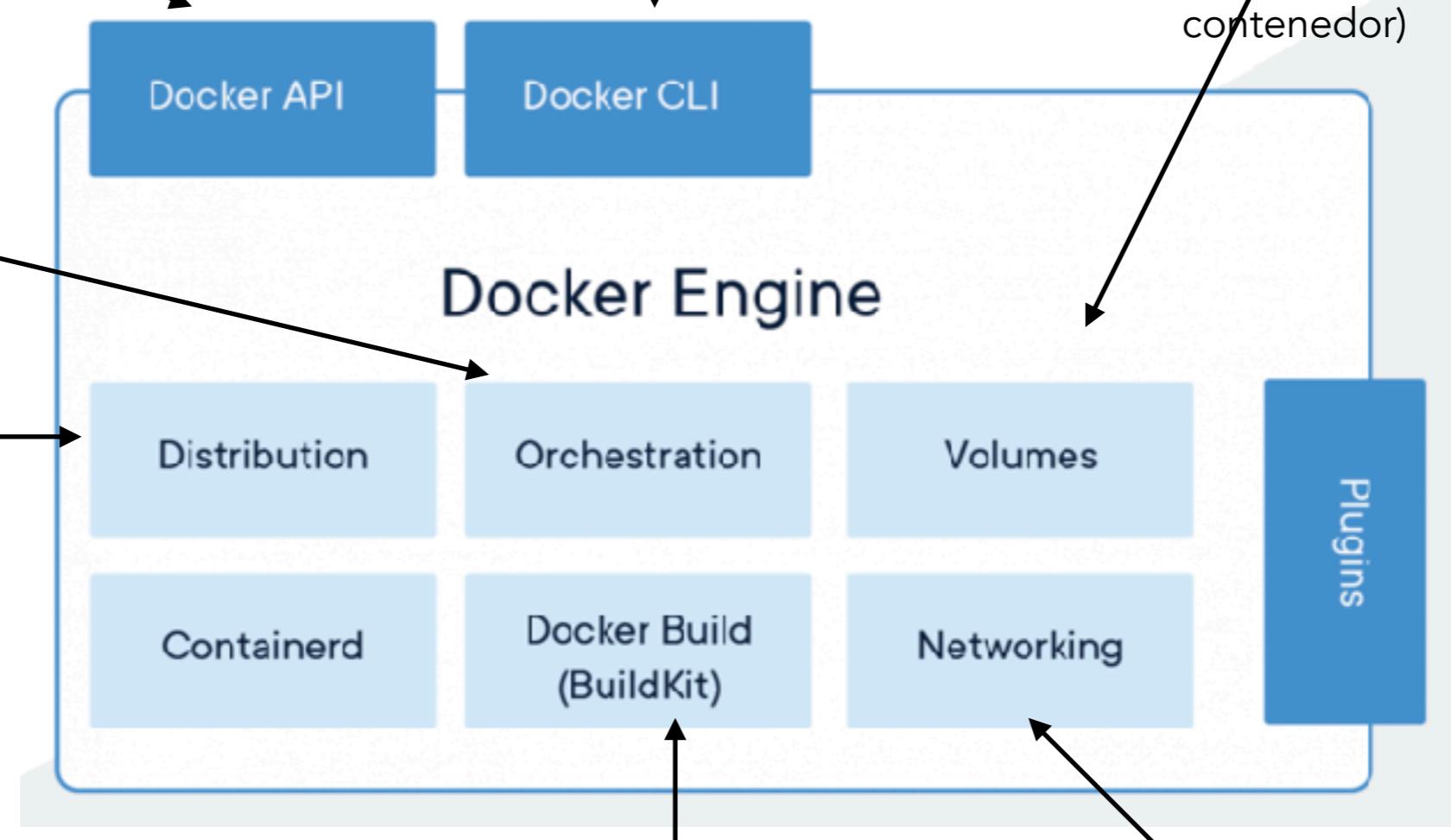
# Docker Engine

Plugin para el  
almacenamiento los  
contenedores  
(Son montados en el  
contenedor)

Manejo de múltiples  
contenedores

Distribución de los  
contenedores

## Docker Engine





# Algunos comandos básicos

- docker container **run -it**  
ubuntu:latest /bin/bash
- docker **run -d -p 80:80** docker/  
getting-started
- docker **ps**
- docker container **stop**  
nombre\_contenedor
- docker container **rm**  
nombre\_contenedor

-it container interactivo y  
atulado a la consola.

-d modo detach

-p exponer el puerto 80 del  
contenedor al 80 del host

ls listar los contenedores locales



# Crear imágenes: Manual

- docker pull centos:latest
- docker run it --name mywebserver centos:latest
- yum install http -y
- vi /var/www/html/index.html
- exit

```
docker commit mywebsebver mywebserver:v1
```

```
docker run -p 80:80 mywebserver:v1 /usr/sbin/httpd -D FOREGROUND
```

SHARING:

```
docker push <hub-user>/<repo-name>:<tag>
```



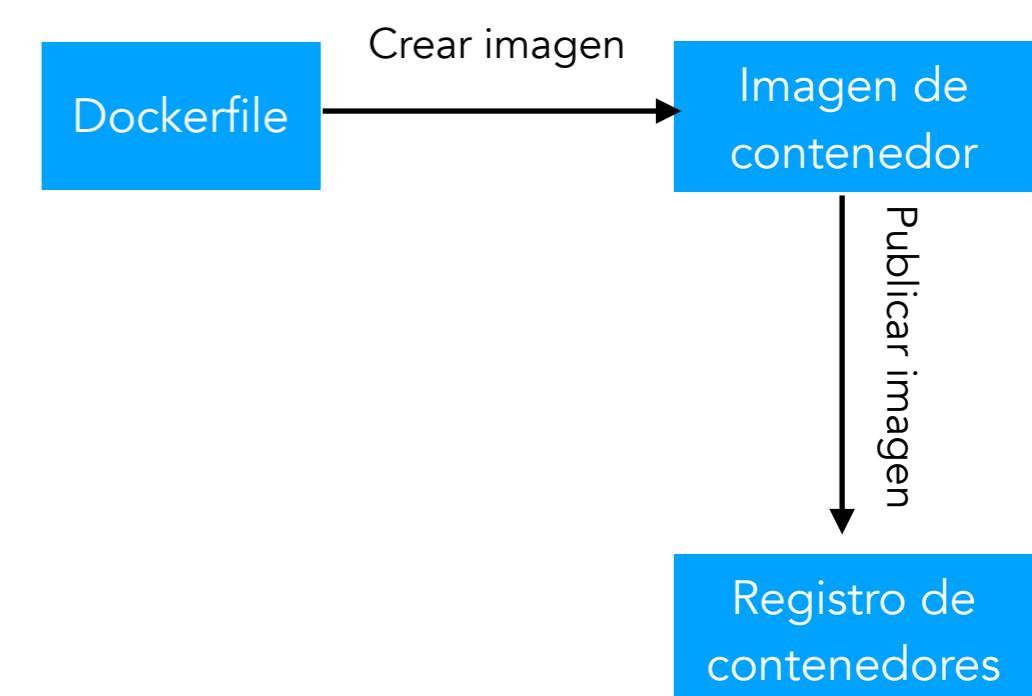
- Se crean imágenes automáticamente leyendo instrucciones de un Dockerfile.
- Dockerfile es un documento de texto con todos los comando a ejecutar para formar la imagen.
- Ejemplo Dockerfile

```
FROM ubuntu
MAINTAINER EmpresaxyZ
RUN apt-get update
RUN apt-get install -y apache2
RUN apt-get install -y apache2-utils
RUN apt-get clean
EXPOSE 80 CMD ["apache2ctl", "-D", "foreground"]
```

# Crear imágenes: DockerFile

```
docker build -t servidorapache:v1 .
```

```
docker container run -d --name web1 --
publish 80:80 servidorapache
```





# Crear múltiples contenedores

- Docker compose: Despliega y administra aplicación con múltiples contenedores en nodos Docker.
- Es un binario externo de Python que se instala en el host de Docker.
- Se definen aplicaciones con múltiples contenedores en un archivo YAML.

```
$docker-compose up -d  
$docker compose down  
$docker-compose ps  
$docker-compose restart
```

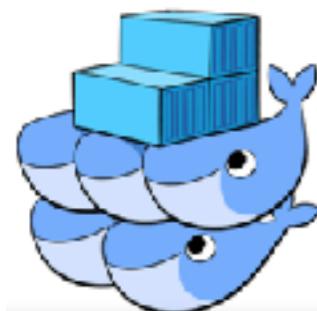
```
version: "3.8"  
services:  
  
  web-fe:  
    build: .  
    command: python app.py  
    ports:  
      - target: 5000  
        published: 5000  
    networks:  
      - counter-net  
  volumes:  
    - type: volume  
      source: counter-vol  
      target: /code  
  
  redis:  
    image: "redis:alpine"  
    networks:  
      counter-net:  
  
    networks:  
      counter-net:  
  
    volumes:  
      counter-vol:
```



# Orquestación de contenedores

Los orquestadores como Kubernetes ayudan en la administración y escalabilidad de las aplicaciones que funcionan en contenedores.

Es un conjunto de APIs que ayuda a desplegar contenedores en un conjunto de nodos llamados cluster.



Pequeño a mediano



Administrador de  
contenedores de Red  
Hat



**kubernetes**

Gran escala, mayor  
comunidad.

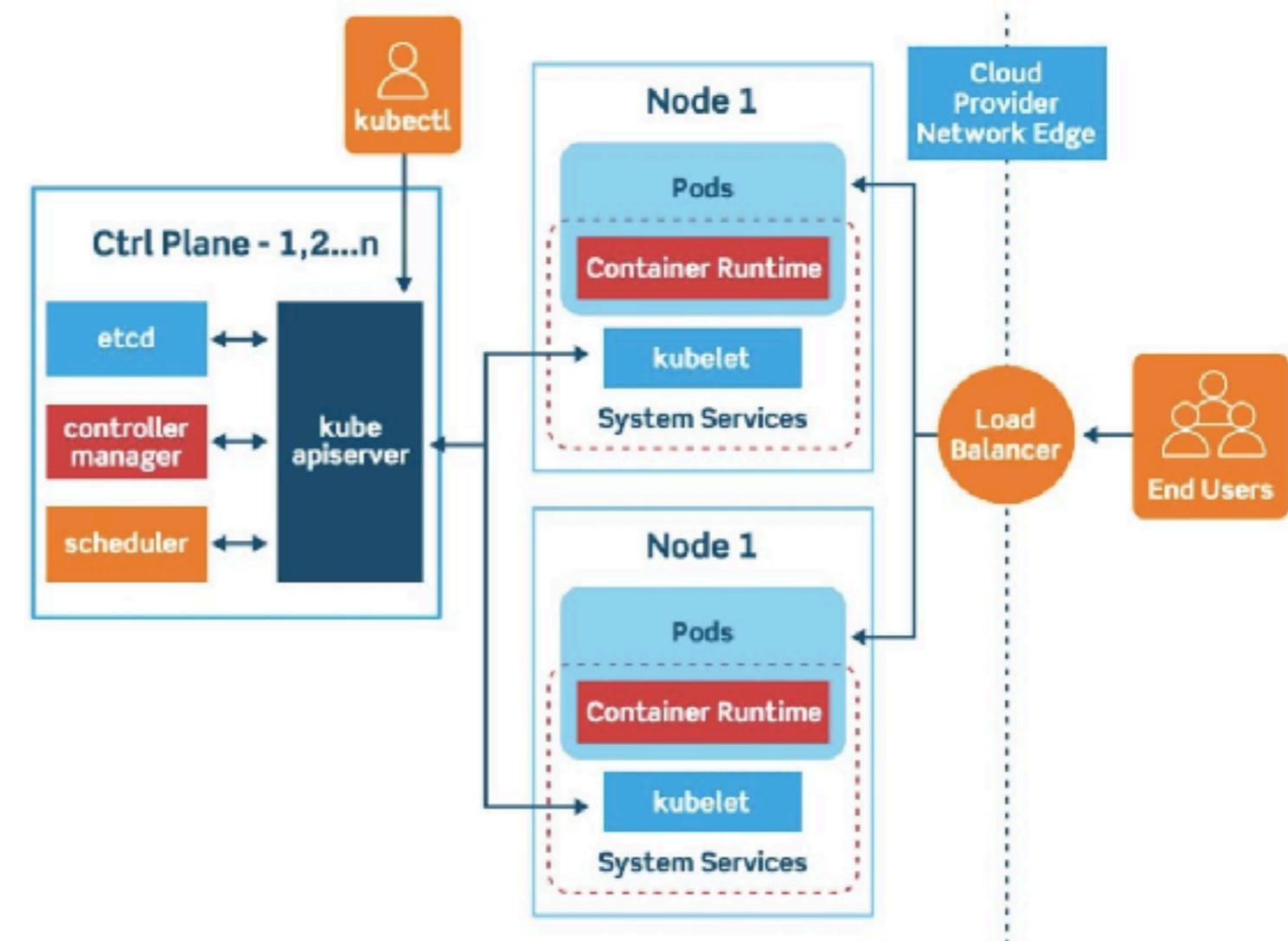


Servicio de Kubernetes  
en Azure



# Arquitectura Kubernetes

- Nodo controlador: Ctrl Plane
- Nodos del cluster con kubelet.



<https://platform9.com/blog/kubernetes-enterprise-chapter-2-kubernetes-architecture-concepts/>



# Impacto contenedores en Data Science

- Hace de los proyectos agnósticos del hardware
- Evita *Cloudlock*
- Facilita el uso de librerías con configuraciones complejas.
- Facilita compartir el trabajo.
- Ejemplos
  - Jupyter notebook en contenedores.
  - Desarrollo de aplicaciones en TensorFlow en contenedores como microservicios.



<https://towardsdatascience.com/how-to-run-jupyter-notebook-on-docker-7c9748ed209f>

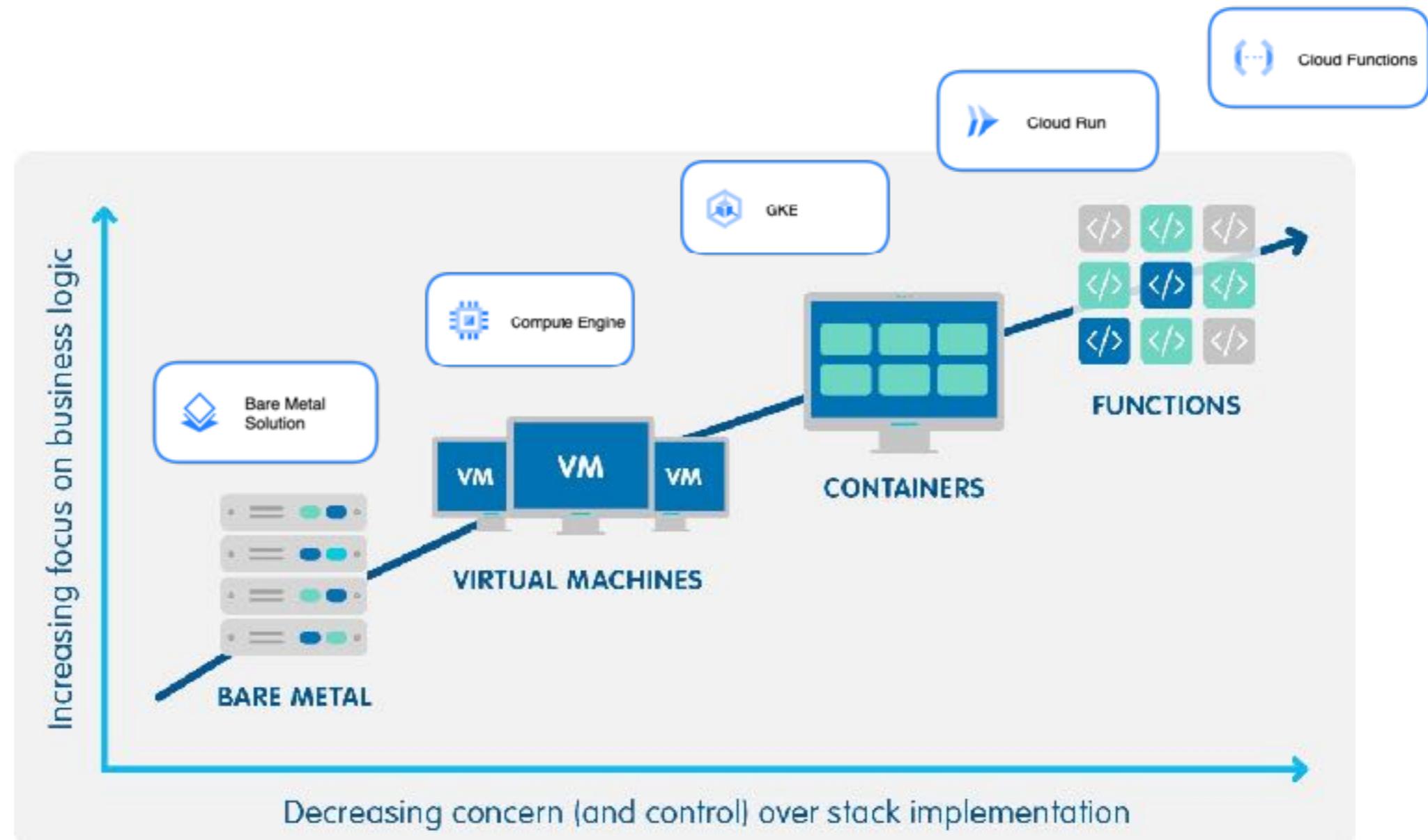
<https://jupyter-docker-stacks.readthedocs.io/en/latest/>



UNIVERSIDAD TECNICA  
FEDERICO SANTA MARIA

Departamento de Informática

# Enfoques en Cloud Computing: GCP





UNIVERSIDAD TECNICA  
FEDERICO SANTA MARIA

Departamento de Informática

# Comunicación y Virtualización

COMUNICACIÓN

PH.D. Erika Rosas Olivos

[erosas@inf.utfsm.cl](mailto:erosas@inf.utfsm.cl)

Septiembre 2023



# ¿Porqué se necesita comunicación y transferir datos?

- Mover datos desde la fuente a los clusters de procesamiento.
  - Adquisición de datos
- Almacenar datos procesados a fuentes donde puedan ser consultadas o visualizadas.
  - Mover a Bases de Datos de servicio.
  - Mover a frameworks de visualización.
- Diseminar notificaciones de eventos.
  - Enviar eventos a sistemas/usuarios que los requieran.
- Mover datos dentro de un mismo sistema distribuido.
  - Ej. Transmisión de datos entre worker nodes.



<https://www.sealpath.com/blog/protecting-the-three-states-of-data/>



# Representación de datos externos y serialización

- La información en los programas es representada como **estructuras de datos** y la información en los mensajes consiste en una **secuencia de bytes**.
- La estructura de datos se debe convertir antes de la transmisión y debe ser reconstruida a la llegada.
- Problemas: orden little-endian o big-endian; estándar ASCII o Unicode.

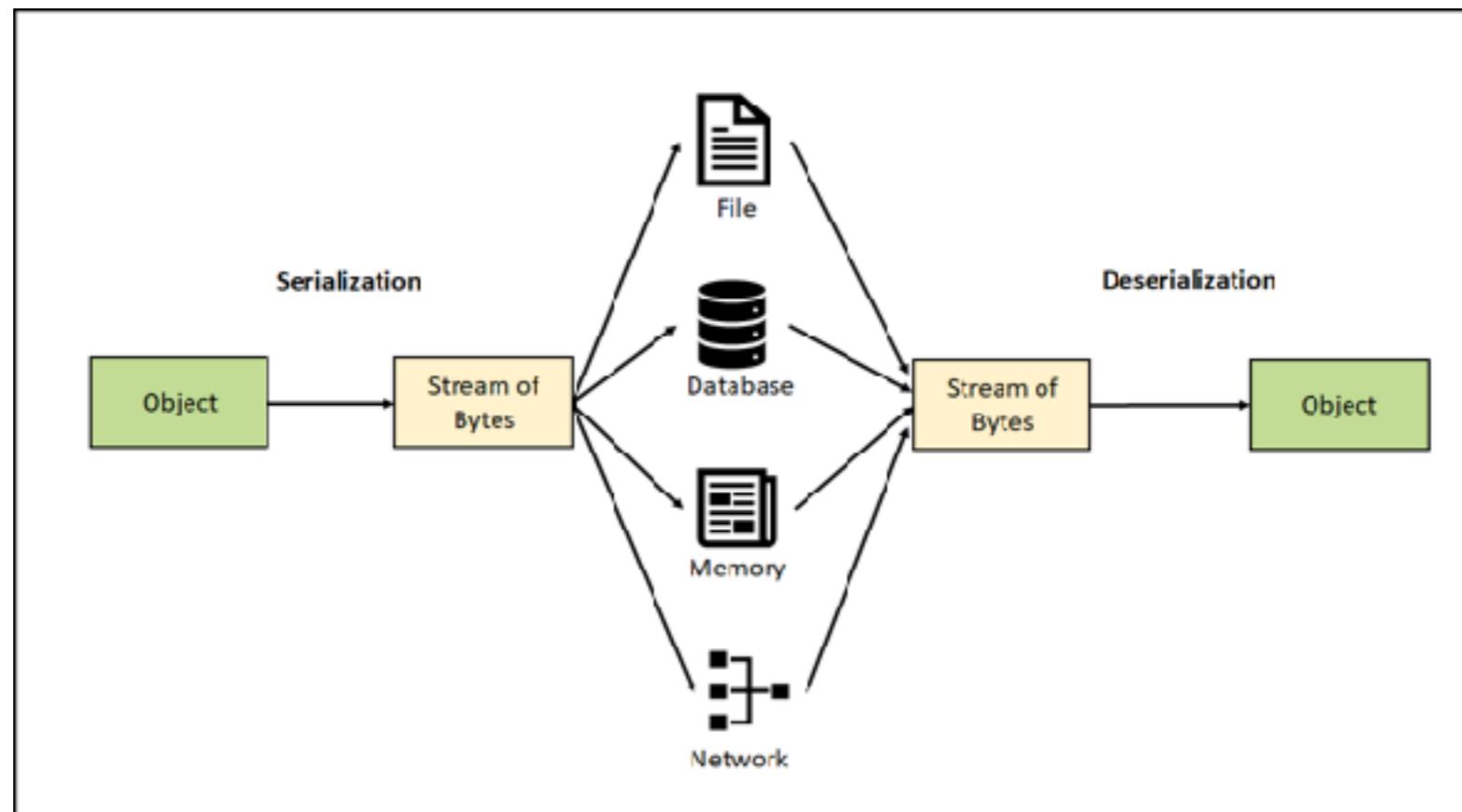




# Representación de datos externos y serialización

**Marshalling/Serialización:** es el proceso de tomar una colección de elementos de datos y ensamblarlos en una forma adecuada para su transmisión en un mensaje.

**Unmarshalling/Deserialización:** es el proceso de desensamblar los elementos a la llegada para producir una colección de datos equivalentes en el destino.





UNIVERSIDAD TECNICA  
FEDERICO SANTA MARIA

Departamento de Informática

# Serialización específica del lenguaje



Java -> `java.io.Serializable`



Ruby -> `Marshal`



Python -> `pickle`

Cómodas y convenientes, no se agrega casi nada de código.

- Problemas: Dependiente del lenguaje, leerlo desde un lenguaje diferente es difícil.
- Problemas de seguridad.
- Problemas de rendimiento.



# JSON, XML y Variantes Binarias



## XML

```
<empInfo>
  <employees>
    <employee>
      <name>James Kirk</name>
      <age>10</age>
    </employee>
    <employee>
      <name>Jean-Luc Picard</name>
      <age>45</age>
    </employee>
    <employee>
      <name>Wesley Crusher</name>
      <age>27</age>
    </employee>
  </employees>
</empInfo>
```

## JSON

```
[{"empInfo": {
  "employees": [
    {"name": "James Kirk", "age": 40},
    {"name": "Jean-Luc Picard", "age": 45},
    {"name": "Wesley Crusher", "age": 27}
  ]
}}
```

- JSON, XML, CSV formatos textuales.
- Ambigüedad al codificar números y string que muestran números, o entre tipos de números como enteros y punto flotante.
- Se puede soportar esquema opcionalmente en JSON y XML.
- Suficientemente buenos para múltiples propósitos.



# Serialización binaria

- Para comunicación interna en una organización se puede usar un formato más compacto y rápido.
- Para datasets pequeños la ganancia no es mucha, pero si, si se hablan de terabytes de datos.
- Codificación binaria de JSON (MessagePack, BSON, BISON, Etc.)
- Ejemplo MessagePack bajar de 81 bytes a 66 bytes

## MessagePack

Byte sequence (66 bytes):

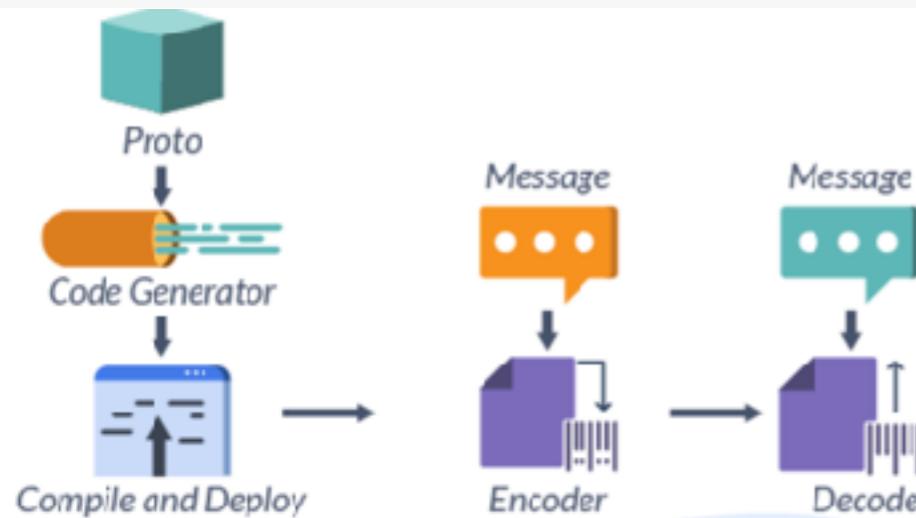
83	a8	75	73	65	72	4e	61	6d	65	a6	4d	61	72	74	69	6e	ae	66	61
76	6f	72	69	74	65	4e	75	6d	62	65	72	cd	05	39	a9	69	6e	74	65
72	65	73	74	73	92	ab	64	61	79	64	72	65	61	6d	69	6e	67	a7	68
61	63	6b	69	6e	67														

Breakdown:

object (3 entries)	string (length 8)	u s e r N a m e	string (length 6)	M a r t i n
83	a8	75 73 65 72 4e 61 6d 65	a6	4d 61 72 74 69 6e
	ae	66 61 76 6f 72 69 74 65 4e 75 6d 62 65 72		
uint16	1337	string (length 9)	interes	ts
cd	05 39	a9	69 6e 74 65 72 65 73 74 73	
array (2 entries)	string (length 11)	d a y d r e a m i n g		
92	ab	64 61 79 64 72 65 61 6d 69 6e 67		
	a7	h a c k i n g		
		68 61 63 6b 69 6e 67		



```
message Person {  
    required string name = 1;  
    required int32 id = 2;  
    optional string email = 3;  
  
    enum PhoneType {  
        MOBILE = 0;  
        HOME = 1;  
        WORK = 2;  
    }  
  
    message PhoneNumber {  
        required string number = 1;  
        optional PhoneType type = 2 [default = HOME];  
    }  
  
    repeated PhoneNumber phone = 4;  
}
```



```
Person person;  
person.set_name("John Doe");  
person.set_id(1234);  
person.set_email("jdoe@example.com");  
fstream output("myfile", ios::out | ios::binary);  
person.SerializeToOstream(&output);  
  
fstream input("myfile", ios::in | ios::binary);  
Person person;  
person.ParseFromIstream(&input);  
cout << "Name: " << person.name() << endl;  
cout << "E-mail: " << person.email() << endl;
```

Ventajas sobre XML  
Simple  
3 a 10 veces más pequeño  
20 a 100 veces más rápido  
Menos ambiguo pero no es flexible al esquema

Fuente: <https://developers.google.com/protocol-buffers/docs/overview>



## JSON

```
{  
  "userName": "Martin",  
  "favouriteNumber": 1337,  
  "interests": ["daydreaming", "hacking"]  
}
```

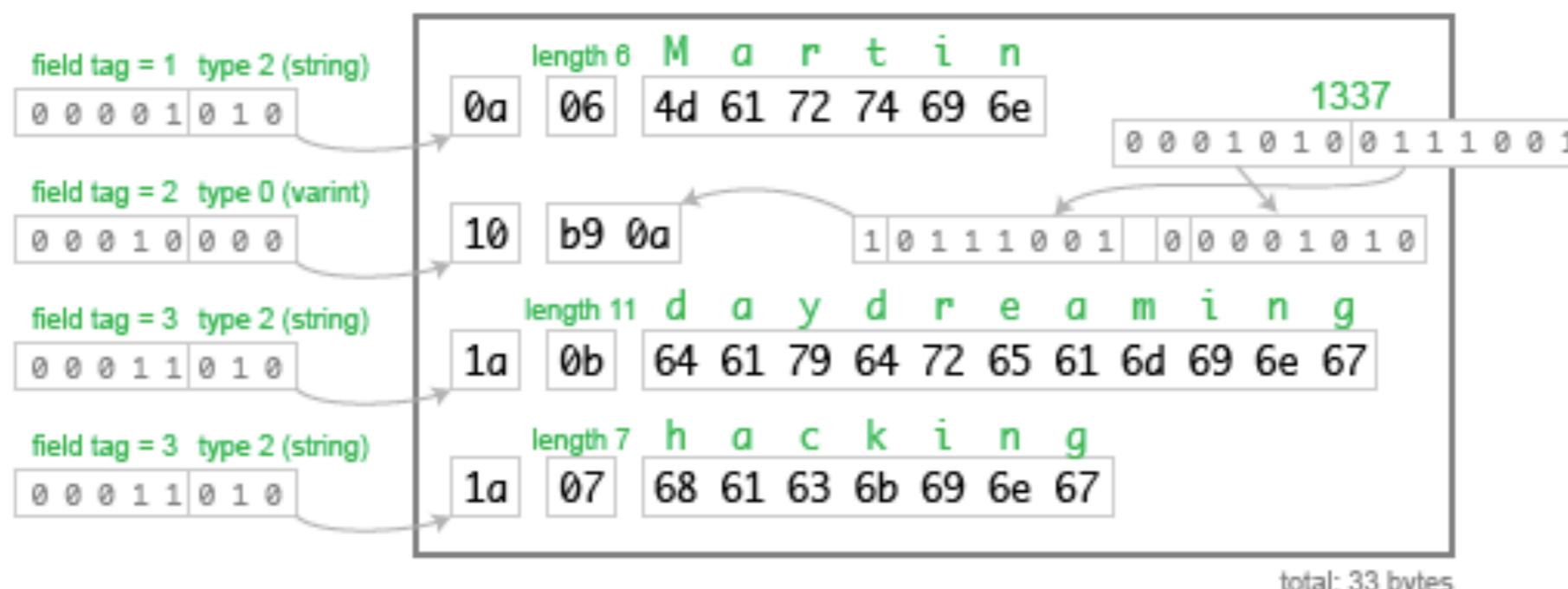
Encoded in 84 bytes

## Protocol buffers

```
message Person {  
  required string user_name = 1;  
  optional int64 favourite_number = 2;  
  repeated string interests = 3;  
}
```

Encoded in 33 bytes

### Protocol Buffers





# Comparación de rendimiento

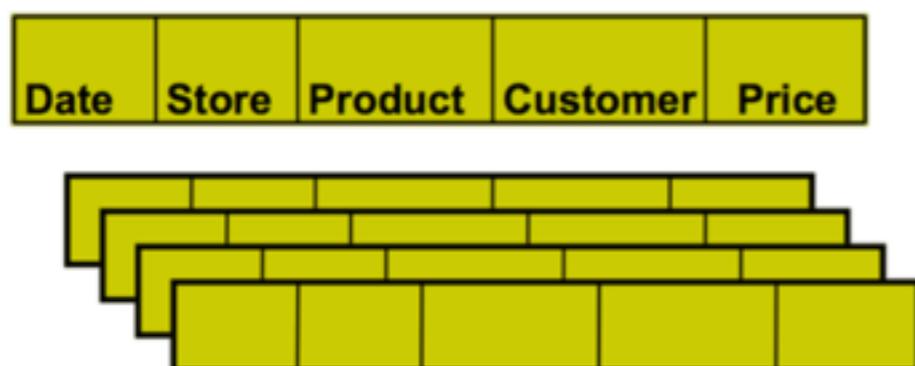
	CPU del Servidor %	CPU del Cliente Promedio %	Tiempo Promedio
<b>REST - XML</b>	12%	80.75 %	05:27.45
<b>REST - JSON</b>	20%	75%	04:44.83
<b>RMI</b>	16%	46.5%	02:14.54
<b>Protocol Buffers</b>	30%	37.75%	01:19.48
<b>Thrift - TBinaryProtocol</b>	33%	21%	01:13.65
<b>Thrift - TCompactProtocol</b>	30%	22.5%	01:05.12



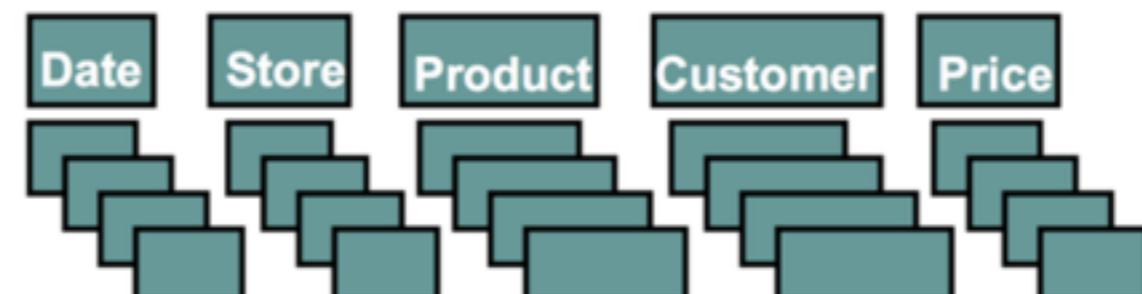
# Almacenamiento Orientado a Columnas

- Problema: Trillones de filas y petabytes de datos in tablas.
- Consultas: acceso a un gran número de filas pero pocas columnas, de las muchas que forman la tabla (+ de 100s)
- A pesar de los indices que hacen que no se carga en memoria toda la tabla, igualmente se necesita cargar la fila completa.
- Solución: Almacena cada columna en un archivo separado.

**row-store**



**column-store**





# Almacenamiento Orientado a Columnas

fact\_sales table

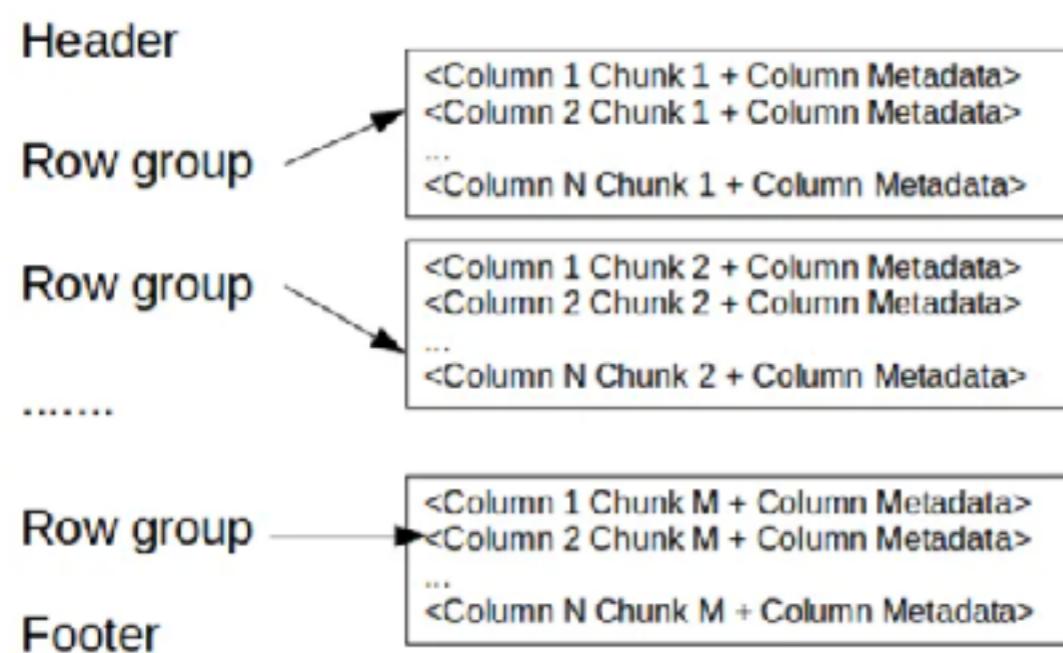
date_key	product_sk	store_sk	promotion_sk	customer_sk	quantity	net_price	discount_price
140102	69	4	NULL	NULL	1	13.99	13.99
140102	69	5	19	NULL	3	14.99	9.99
140102	69	5	NULL	191	1	14.99	14.99
140102	74	3	23	202	5	0.99	0.89
140103	31	2	NULL	NULL	1	2.49	2.49
140103	31	3	NULL	NULL	3	14.99	9.99
140103	31	3	21	123	1	49.99	39.99
140103	31	8	NULL	233	1	0.99	0.99

Columnar storage layout:

date\_key file contents: 140102, 140102, 140102, 140102, 140103, 140103, 140103, 140103  
product\_sk file contents: 69, 69, 69, 74, 31, 31, 31, 31  
store\_sk file contents: 4, 5, 5, 3, 2, 3, 3, 8  
promotion\_sk file contents: NULL, 19, NULL, 23, NULL, NULL, 21, NULL  
customer\_sk file contents: NULL, NULL, 191, 202, NULL, NULL, 123, 233  
quantity file contents: 1, 3, 1, 5, 1, 3, 1, 1  
net\_price file contents: 13.99, 14.99, 14.99, 0.99, 2.49, 14.99, 49.99, 0.99  
discount\_price file contents: 13.99, 9.99, 14.99, 0.89, 2.49, 9.99, 39.99, 0.99



# Parquet



- Formato de almacenamiento columnar que soporta modelos de datos basado en documentos.
- Codificación binaria como Protocol Buffers o Apache Thrift.
- **Row group:** almacena todos los valores de la columna para un rango de filas en formato columnar.
- **Column chunk:** Contiene todos los valores de una columna en el Row group.
- **Footer:** Contiene detalles del esquema, metadatos del modelo y metadatos de los grupos de filas y columnas.



# Comparación Formatos para almacenar BIG DATA

- Todos poseen un nivel de compresión.
- Ninguno es legible por humanos.
- Todos pueden dividirse en múltiples discos para escalabilidad y procesamiento paralelo.
- Todos se auto-describen, tienen el esquema integrado.

BIG DATA FORMATS COMPARISON



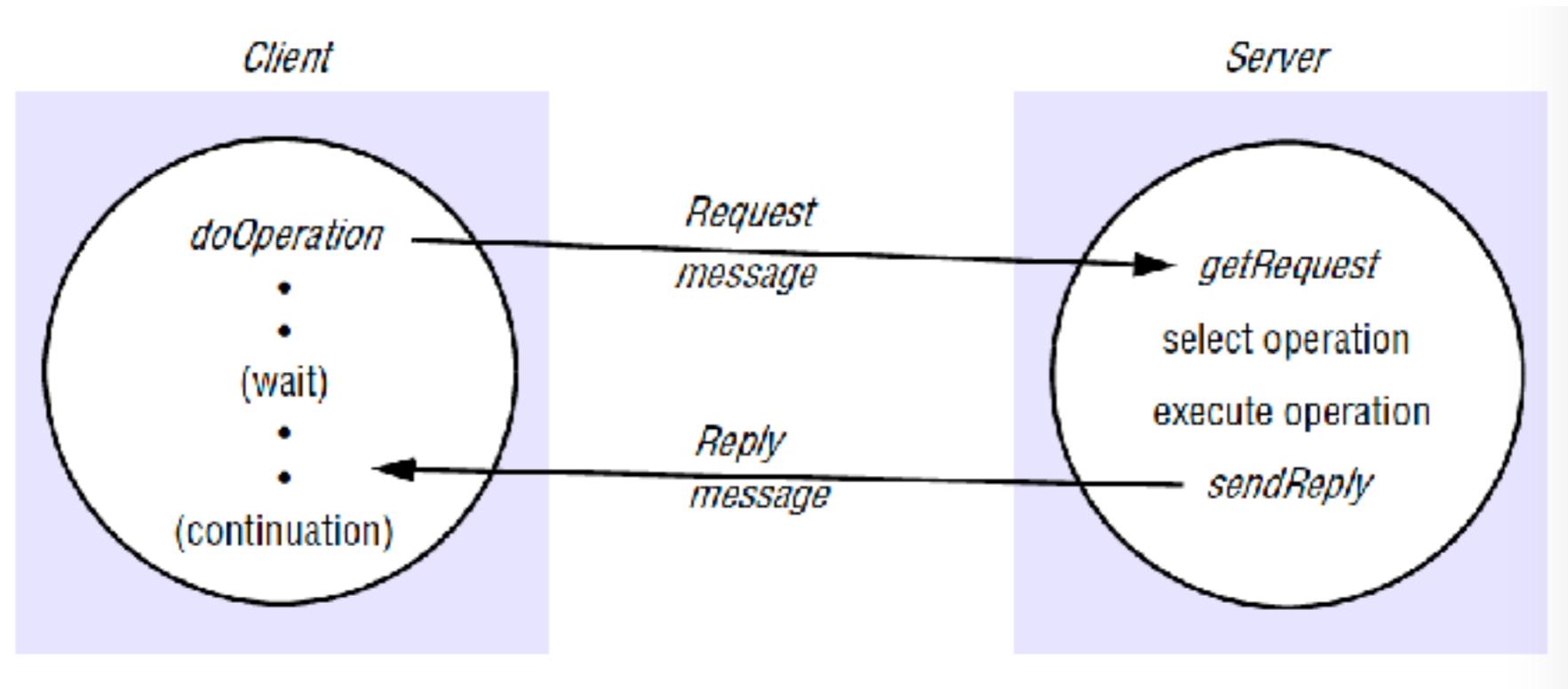
Source: Nexla analysis, April 2018

<https://www.datanami.com/2018/05/16/big-data-file-formats-demystified/>



# Request-reply protocols

- For typical **client-server** interactions: Two-way exchange of messages.
- Generally synchronous and reliable.
- When UDP is used we avoid overhead of TCP due to:
  - ACK are redundant, since request are followed by replies in this model.
  - Establishing a connection involves two extra pairs of messages in addition to the pair required for a request and reply.





# Remote procedure call (RPC)

- Extiende la abstracción de una llamada a procedimiento a sistemas distribuidos.
- **Procedimientos** pueden ser llamados como si estuvieran en el **espacio local de direcciones**.
- Oculta el paso de mensajes de codificación, decodificación de parámetros y resultados.



# Semantics and transparency in RPC

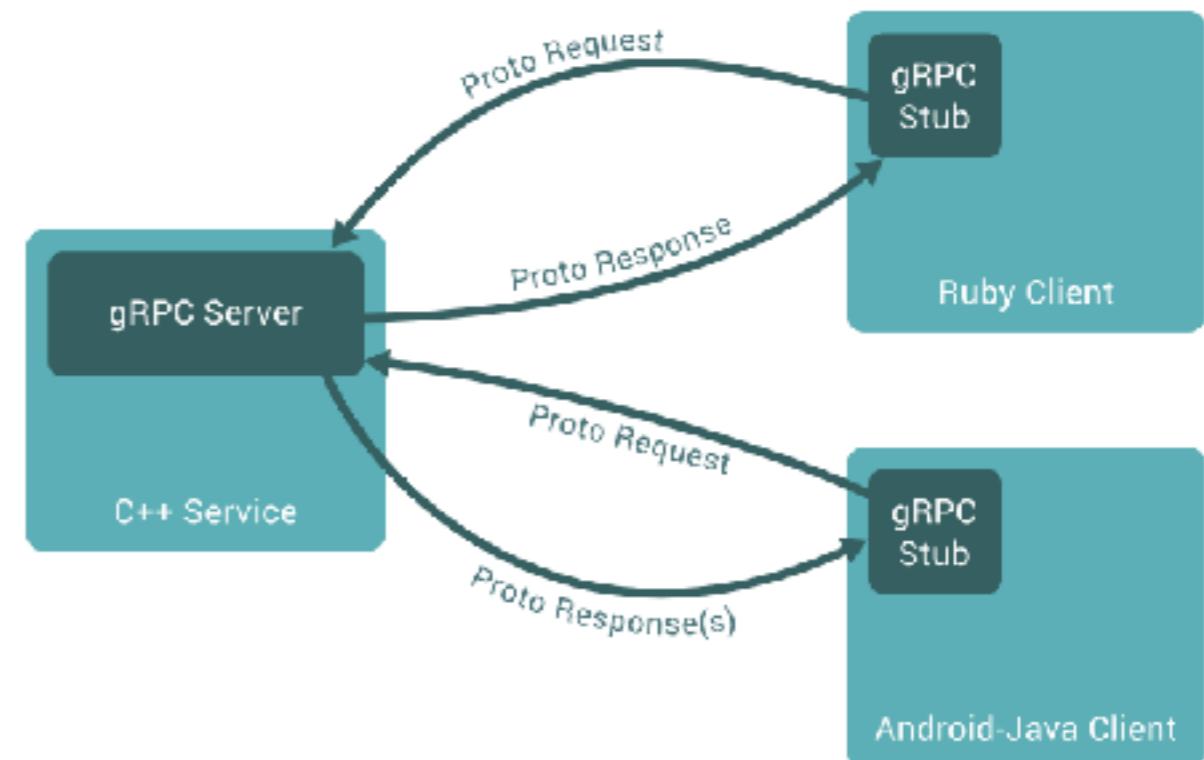
<i>Fault tolerance measures</i>			<i>Call semantics</i>	
<i>Retransmit request message</i>	<i>Duplicate filtering</i>	<i>Re-execute procedure or retransmit reply</i>	<i>Maybe</i>	Fallas ocaciones son aceptables
No	Not applicable	Not applicable	<i>At-least-once</i>	Para operaciones idempotentes
Yes	No	Re-execute procedure	<i>At-most-once</i>	Para operaciones no-idempotentes
Yes	Yes	Retransmit reply		

- **Fallas** son más problemáticas sue en el espacio local.
- **Latencia** de RPC is varios órdenes de magnitud mayor que los del espacio local.
- El que llama debe tener la posibilidad de abortar la llamada RPC.
- RPC no ofrece paso de parámetros por referencia.



- High performance RPC framework that can run in any environment (10 languages).
- Efficiently connect services in and across data centers
- Pluggable support for load balancing, tracing, health checking and authentication.
- Also for mobile devices, browsers and backend services.
- Is built on HTTP/2 transport.
- Use Protocol buffers as IDL and underlying message interchange format.

# Example: gRPC



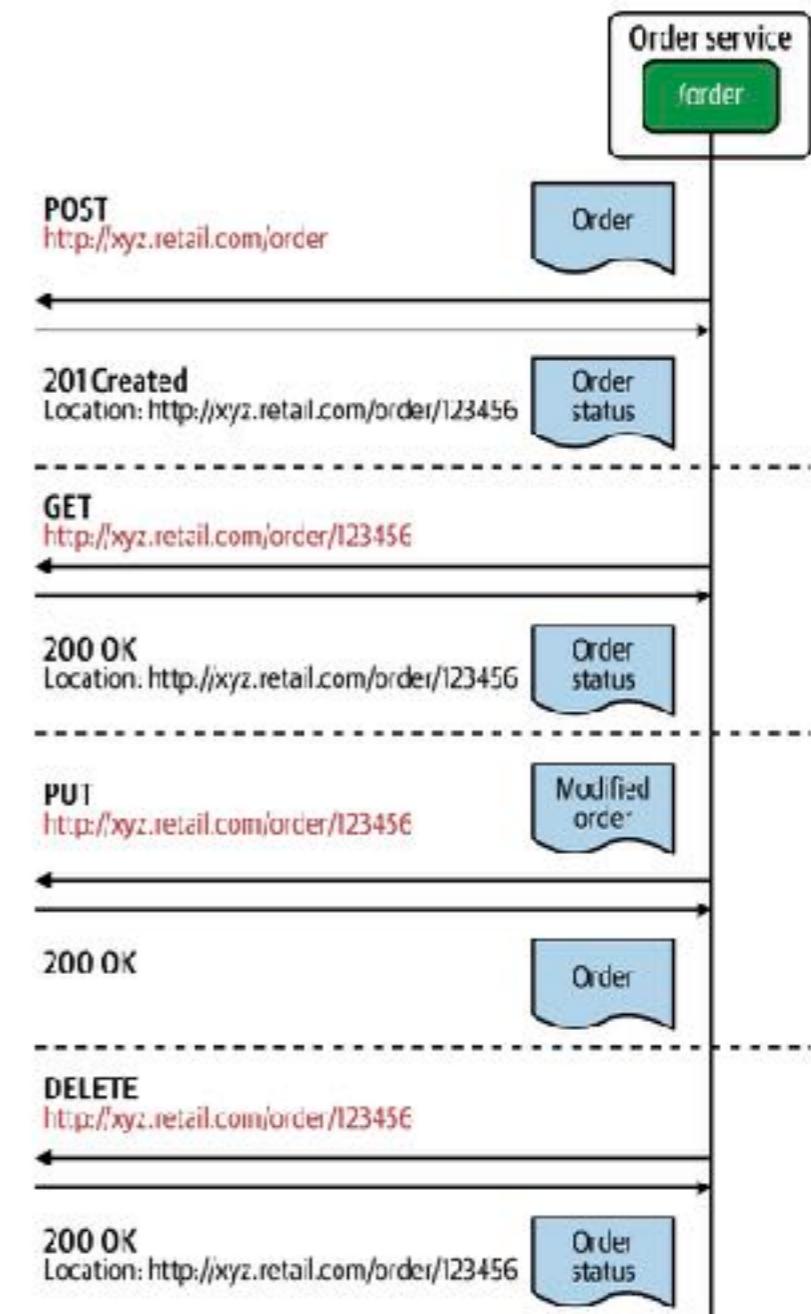
<https://grpc.io/docs/languages/go/basics/>

Source: <https://grpc.io/about/>



# Flujos de datos a través de servicios: REST - RPC - GraphQL

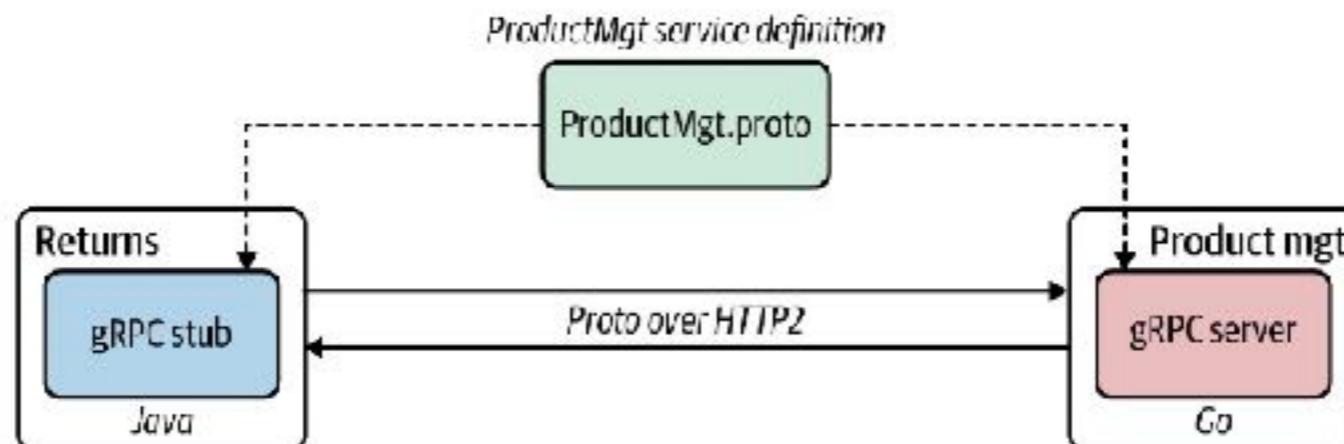
- Paradigmas de API Request-Response
  - REST
    - No es un protocolo, sino que una filosofía de diseño.
    - Centrado en recursos identificados con URLs con operaciones CRUD (Create, Read, Update, Delete)
    - Típicamente JSON como respuesta de la API
    - Para almacenar y rescatar información.
    - Sobre HTTP2 funciona bien pues se pasa a binario el JSON.
    - Predominante para API pública.
  - RPC
  - GraphQL





# Flujos de datos a través de servicios: REST - RPC - GraphQL

- REST
- RPC
  - Nueva generación de frameworks gRPC, Avro, Thrift.
  - Realizan codificado binario.
  - Liviano y de alto rendimiento, se usa cuando la eficiencia y throughput es importante.
  - Usar cuando la comunicación es entre dos componentes de un sistema distribuido controlado por una misma organización, típicamente en un mismo datacenter.
  - Hay un acoplamiento en ambos lados de la comunicación.
- GraphQL

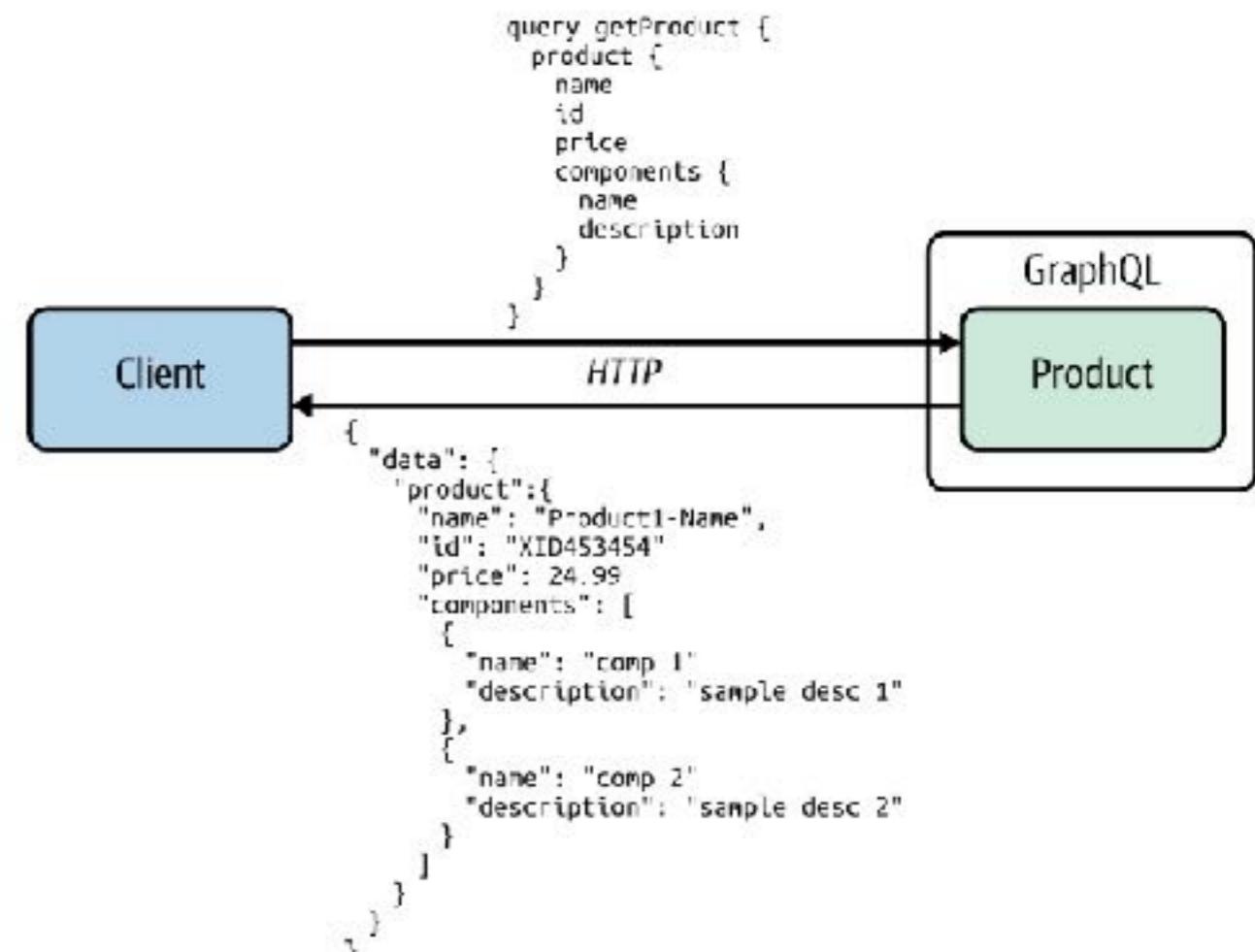


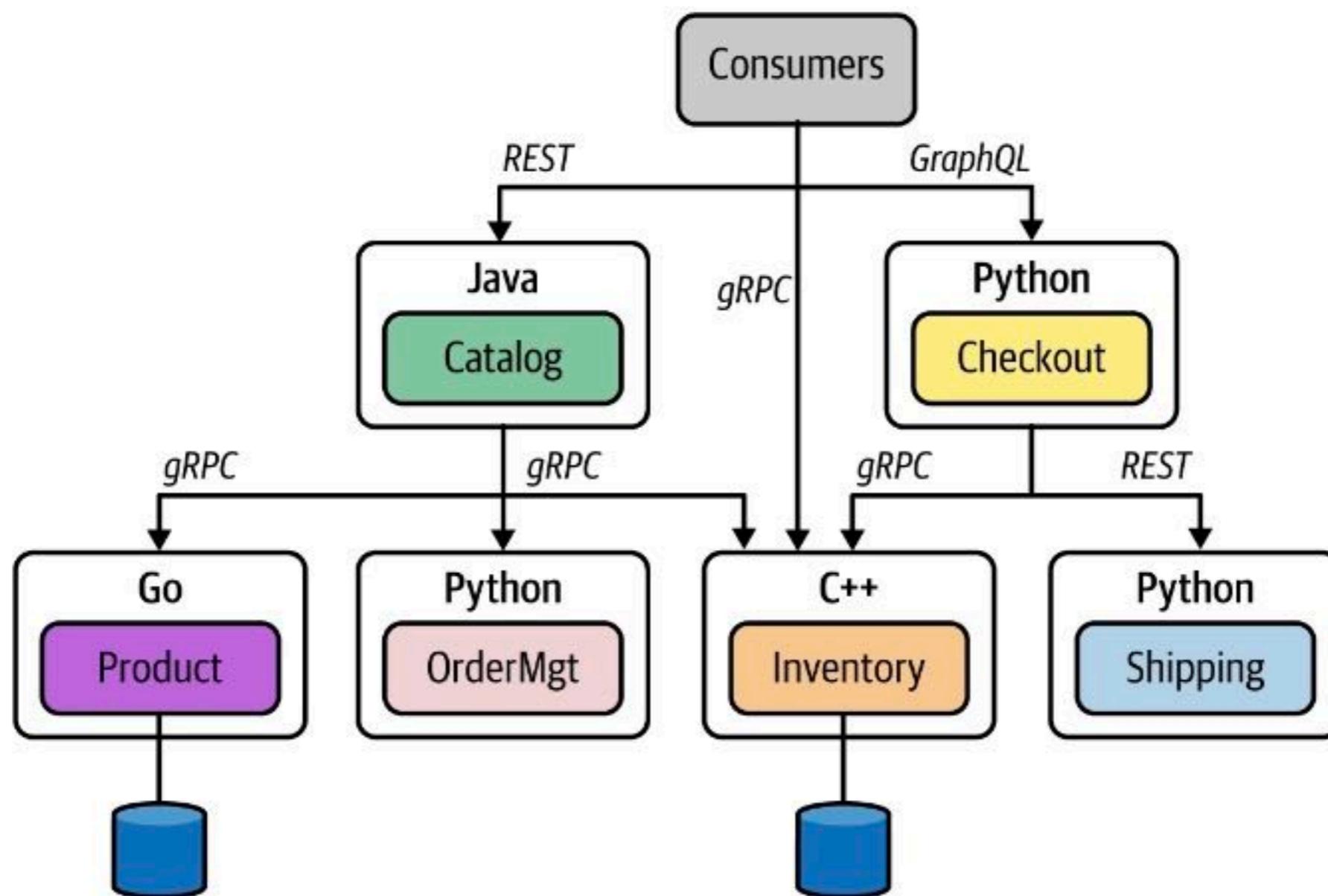
*Figure 2-12. An example of microservices communication using gRPC*



# Flujos de datos a través de servicios: REST - RPC - GraphQL

- REST
- RPC
- GraphQL
  - Casos con consultas/respuestas más complejas, por ejemplo altamente dimensionales en datasets grandes.
  - Se traspasa la cantidad de información exacta que se necesita mejorando costos de red y eficiencia.
  - Un solo Endpoint para múltiples tipos de consultas.
  - Uso cuando una REST API se hace muy compleja.
  - No ideales si la API es simple, dado que agregan complejidad.



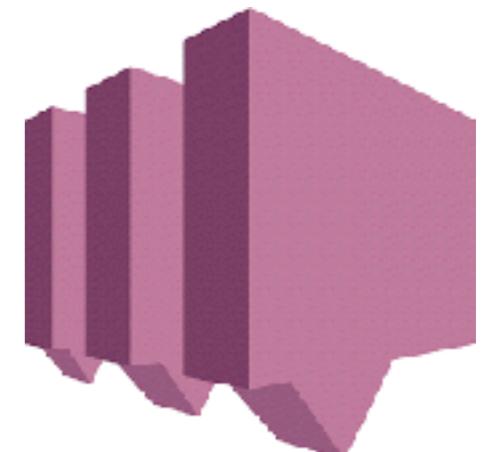


Design Patterns for Cloud Native Applications: Patterns in Practice using APIs, Datta, Events and Streams, Kasun Indrasiri, Sriskandarajah Suhothayan, O'Reilly.



# Comunicación indirecta

1. Sistemas Publish/Subscribe (Apache Kafka, Pub/Sub, Amazon SNS)
2. Sistemas de recolección (Apache Flume)
3. Sistemas de Colas (RabbitMQ)



Todos son sistemas de comunicación indirecta, dado que proveen escalabilidad para el contexto de Big Data.



# Comunicación Indirecta

Comunicación entre entidades en un sistema distribuido a través de un intermediario, sin acoplamiento directo entre el emisor y receptor.

- **Propiedades**

- Desacoplamiento de espacio: El emisor no conoce la identidad de los receptores y viceversa.
- Desacoplamiento de Tiempo: El emisor y receptor pueden tener tiempos de vida diferentes.

- **Desventajas:**

- Se introduce un overhead en el rendimiento debido al nivel de indirección.

Las dimensiones y niveles de desacoplamiento varían de sistema a sistema.

Que pasa con la comunicación asíncrona?



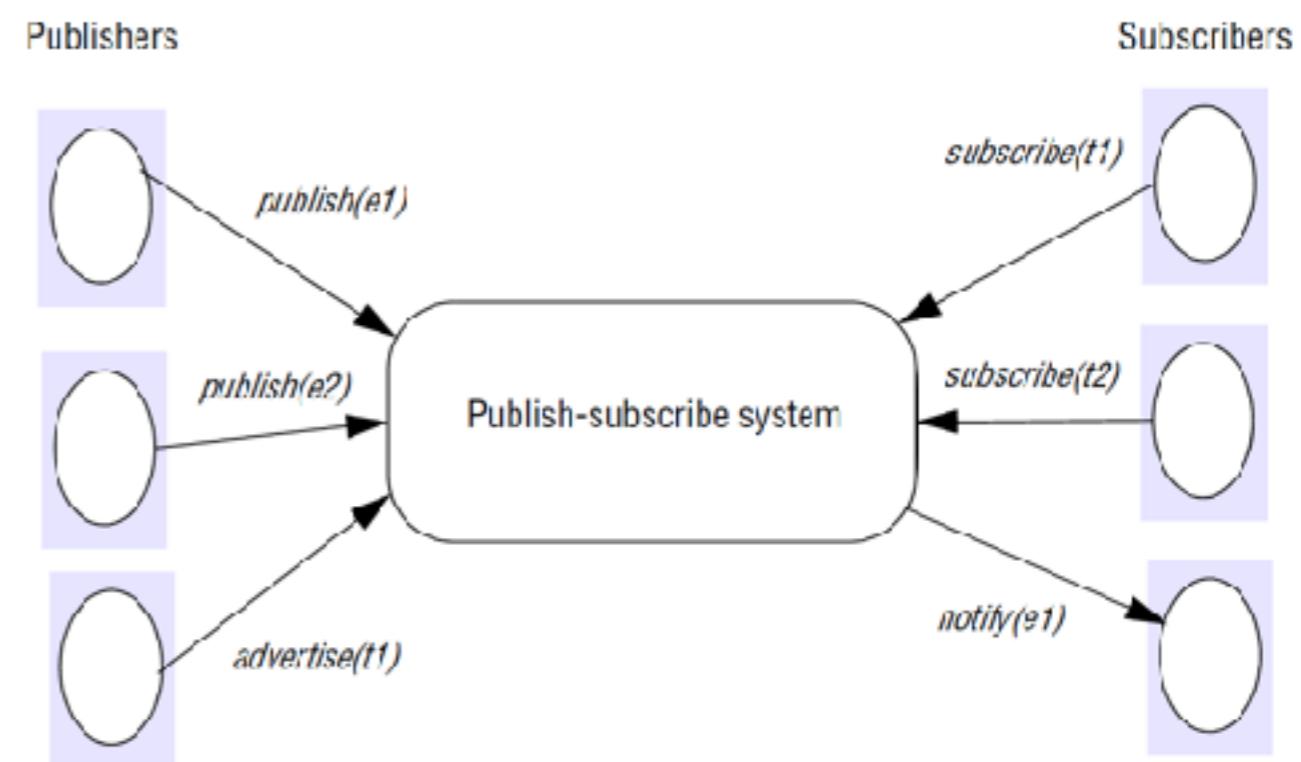
# Frameworks de Publish/Subscribe

- Familia de sistemas que compartir la característica común de diseminar eventos a múltiples receptores a través de un intermediario.
- Los **publicadores** publican eventos estructurados a un servicio de eventos.
- Los **subscriptores** expresan su interés en eventos particulares a través de suscripciones, las cuales pueden ser patrones arbitrarios sobre los eventos estructurados.
- El sistema hace calce de las suscripciones con los eventos publicados y asegura la entrega de las notificaciones de los eventos.



# Modelo de programación Publish/ Subscribe

- Publishers diseminan un evento e usando la operación **publish(e)**
- Subscribers expresan un interés en un conjunto de eventos a través de suscripciones **subscribe(f)**, donde **f** se refiere a un filtro (patrón).
- **Unsubscribe(f)** revoca un interés en un filtro **f**.
- Un evento **e** llega a un subscriptor usando la operación **notify(e)**.
- **advertise(f)** -> los publicadores declaran la naturaleza de los eventos.
- **unadvertise(f)** -> Publicadores revocan **advertise**.





# Modelos de suscripción

- Basado en canal: Publishers publican eventos a canales con nombre y los subscriptores se suscriben a los canales para recibir todos los eventos que son enviados al canal.
- Basado en tópico: Cada evento es tagueado con un tópico, las suscripciones definen tópicos de interés. Se puede usar una jerarquía de organización de tópicos.
- Basado en contenido: Las suscripciones se expresan sobre un rango de campos en una notificación de evento. Las suscripciones hacen calce con una consulta sobre los valores de los atributos de un evento.
- Basado en tipo: En los enfoques basado en objetos, los objetos tienen un tipo. Los eventos tienen un tipo y las suscripciones hacen calce con tipos o subtipos.



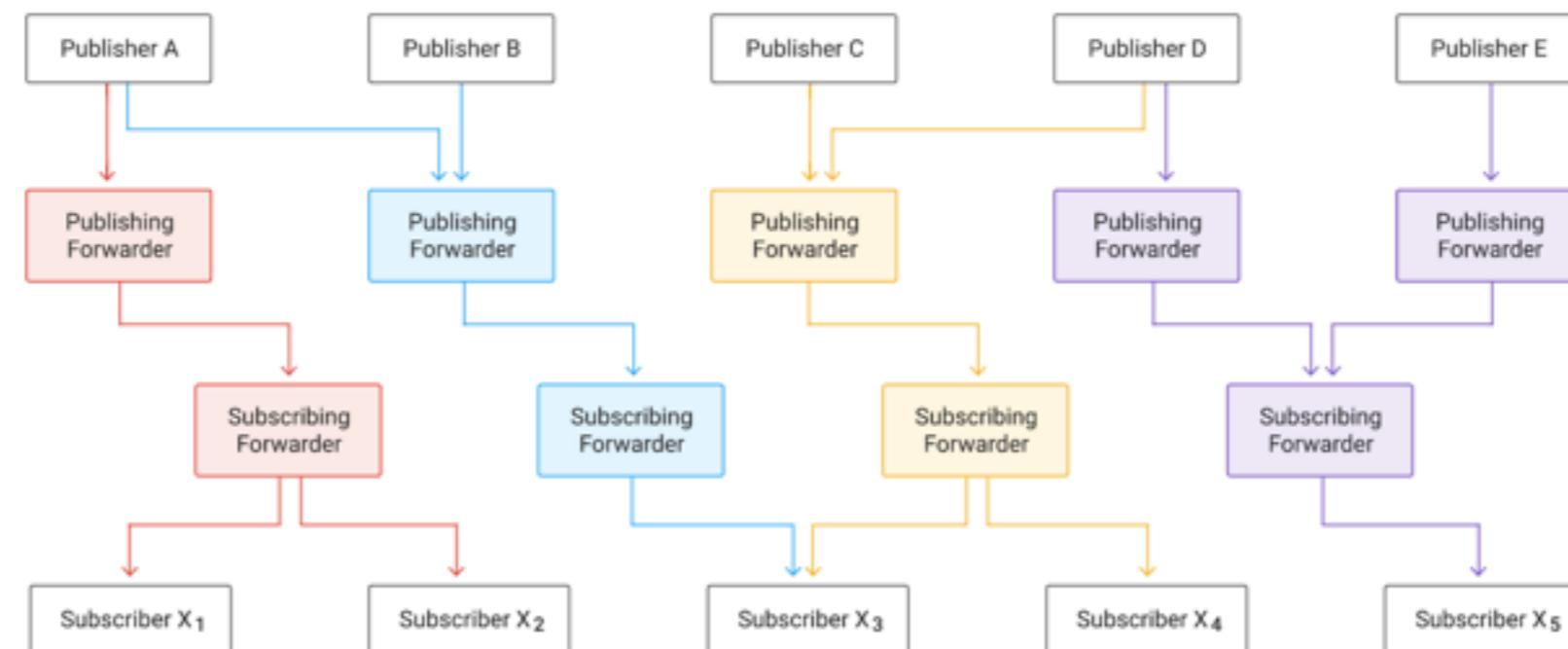
# Pub/Sub

- Servicio administrado, escala de manera automática.
- Ideal para arquitecturas basadas en eventos.
- Entrega mensajes en orden con semántica at least once.
- Periodo de retención de hasta 31 días, gratuito hasta 7 días.
- Costos tienen que ver con throughput, los costos de egreso de la red, y almacenamiento.
- Uso para comunicación servicio a servicio más que cliente-servidor.





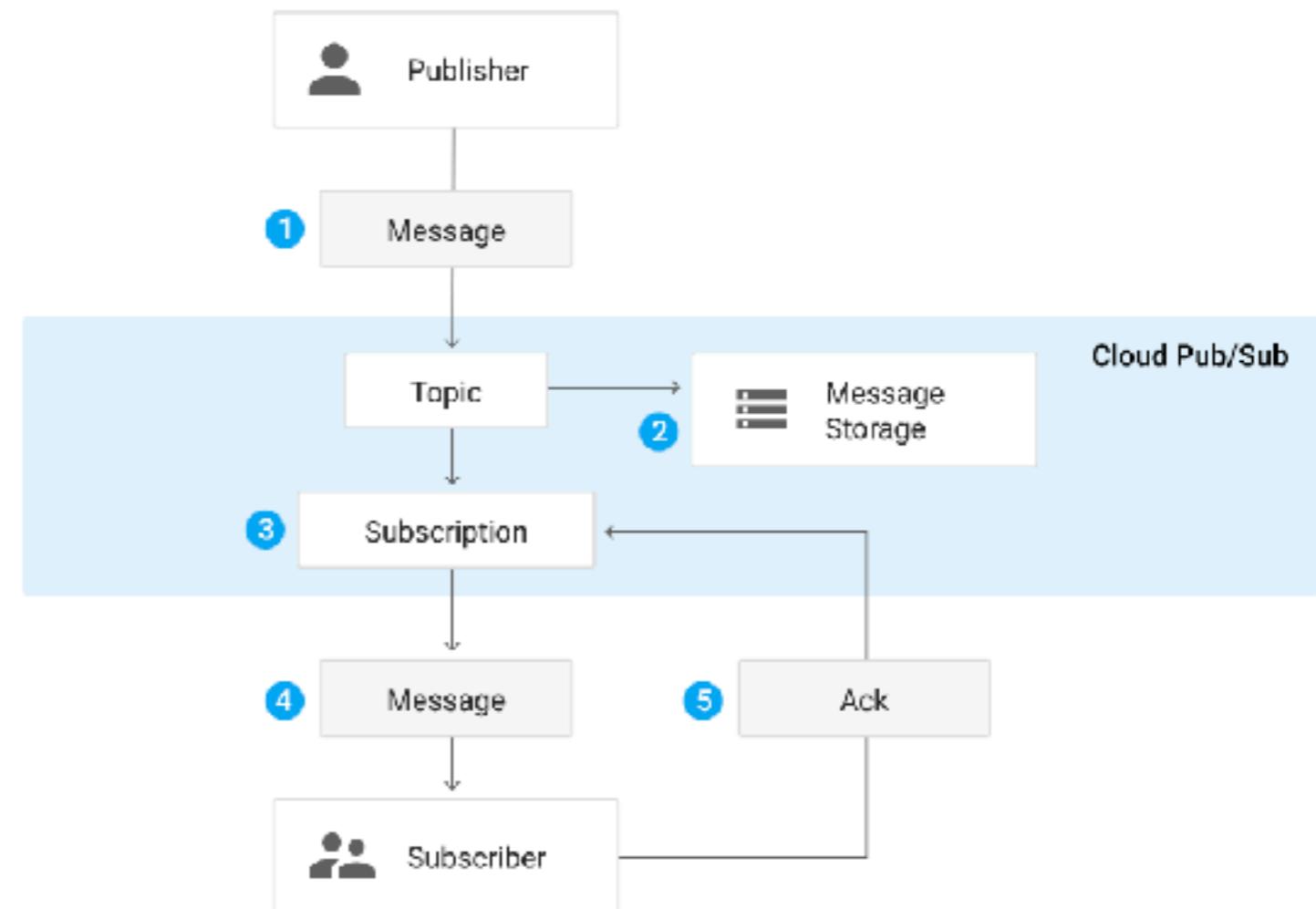
- Internamente hay servidores en el plano de control y en el plano de datos.
- El **plano de datos** forwardear los mensajes de publicadores a subscriptores. Servidores forwarders.
- En el **plano de control** se maneja la asignación de publicadores - subscriptores a los servidores del plano de datos. Servidores routers. Buscan balance entre consistencia y uniformidad.





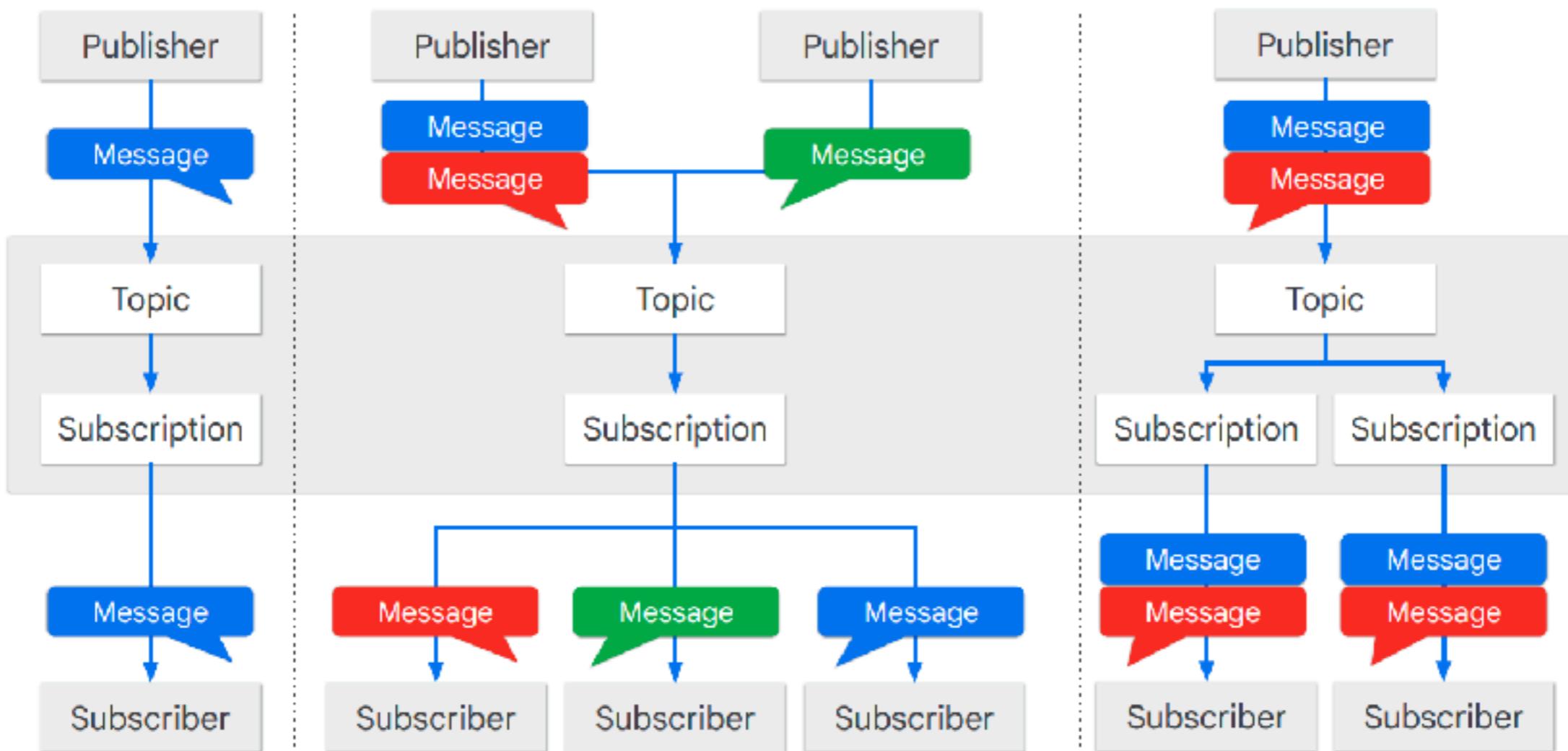
# Pub/Sub

- Through publisher por región 120,000,000 KB por minuto en regiones grandes y 10 veces menos en pequeña.
- Pull throughput suscriptor 240,000,000 KB en regiones grandes y 10 veces menos en regiones pequeñas.
- Push throughout suscriptor por region 8,400,000 kB en regiones grandes y 1,200,000 en pequeñas.





# Patrones comunes de Pub/Sub





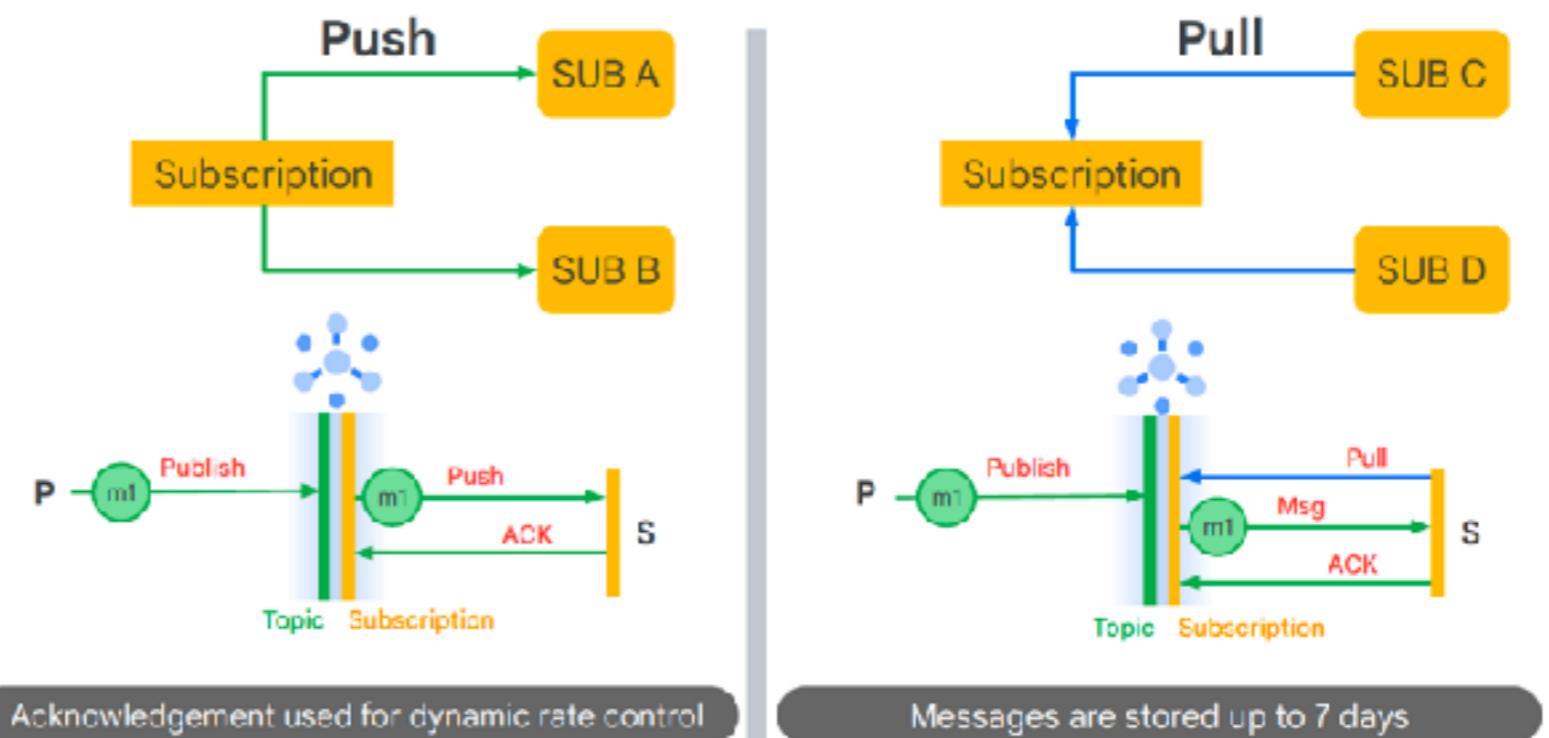
# Subscripciones Push/Pull

## Pull

- Gran volumen de mensajes
- El throughput y eficiencia son críticos.
- No hay un end-point HTTPS público disponible en subscriber.

## Push

- Hay múltiples tópicos para el mismo webhook
- App Engine y Cloud Functions
- No se pueden configurar dependencias GCP





UNIVERSIDAD TECNICA  
FEDERICO SANTA MARIA

Departamento de Informática

# Kafka: commit log



- Sistema de publish/subscribe: un log de commit distribuido.
- Fue diseñado para proveer registros durables de todas las transacciones.
- Los datos almacenados de manera durable, en orden, inmutable y pueden ser leído de manera determinista.
- La unidad de datos es un mensaje.
- Para eficiencia, los mensajes se escriben en batches: una colección de mensajes del mismo tópico.

Powered by:



**CISCO**™



**LinkedIn**

**NETFLIX**

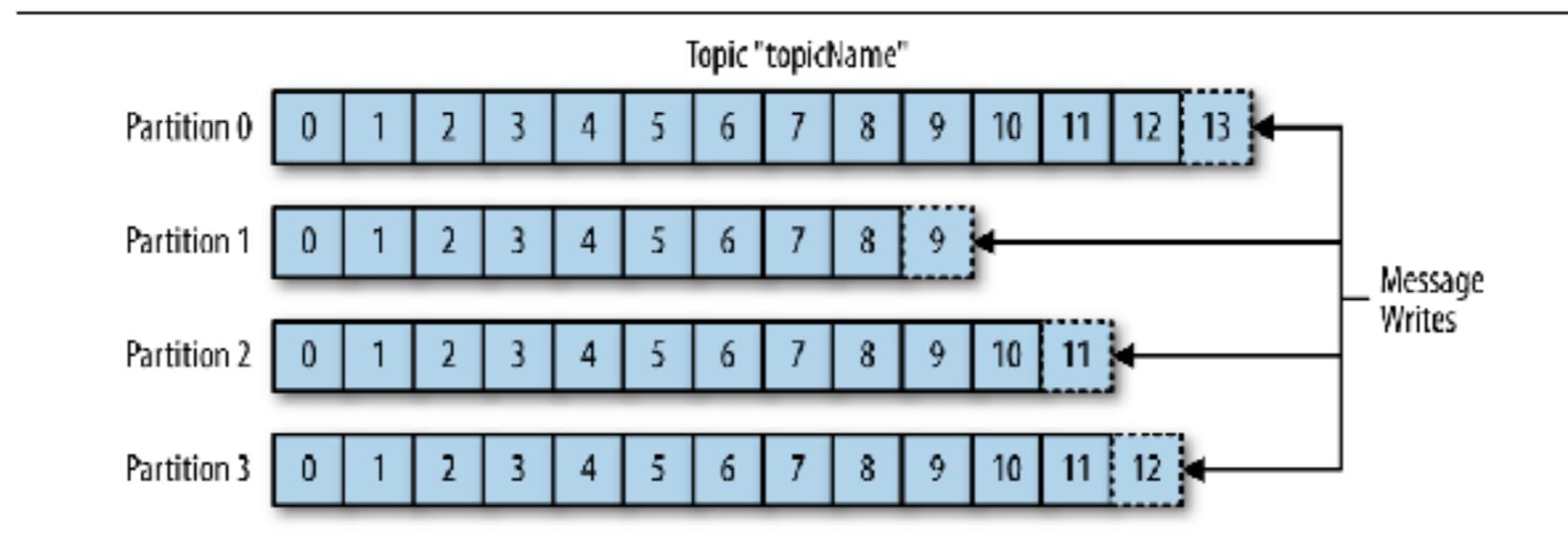


**Itaú**



# Kafka: tópicos y particiones

- **Tópico:** Categorización de mensajes.
- **Partición:** Los tópicos se dividen en un número de particiones. Una partición es un log individual.
- Los mensajes se escriben a las particiones como solo agregar (append), y son leídos en orden.
- No hay garantías de orden de acceso a todo el tópico, solo a una partición.
- Cada partición puede ser alojada en un servidor diferente (escalar horizontalmente)

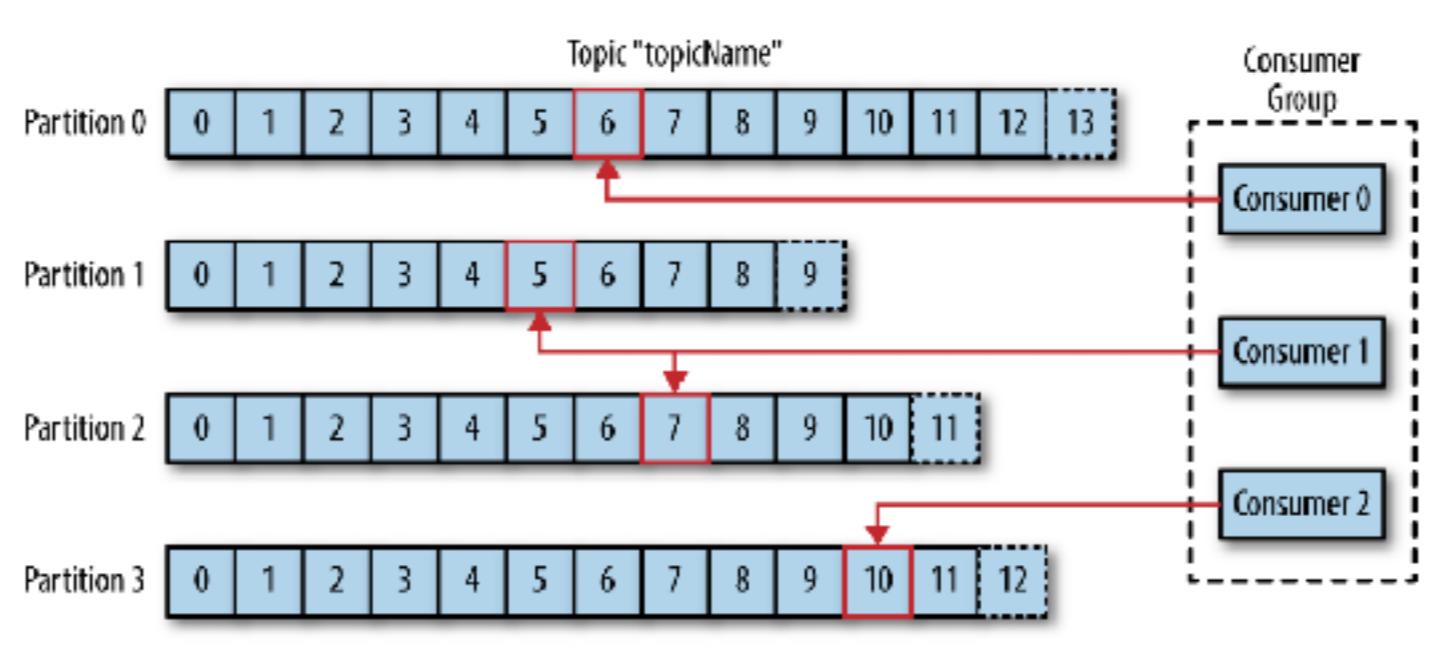




# Kafka: Productores y Consumidores



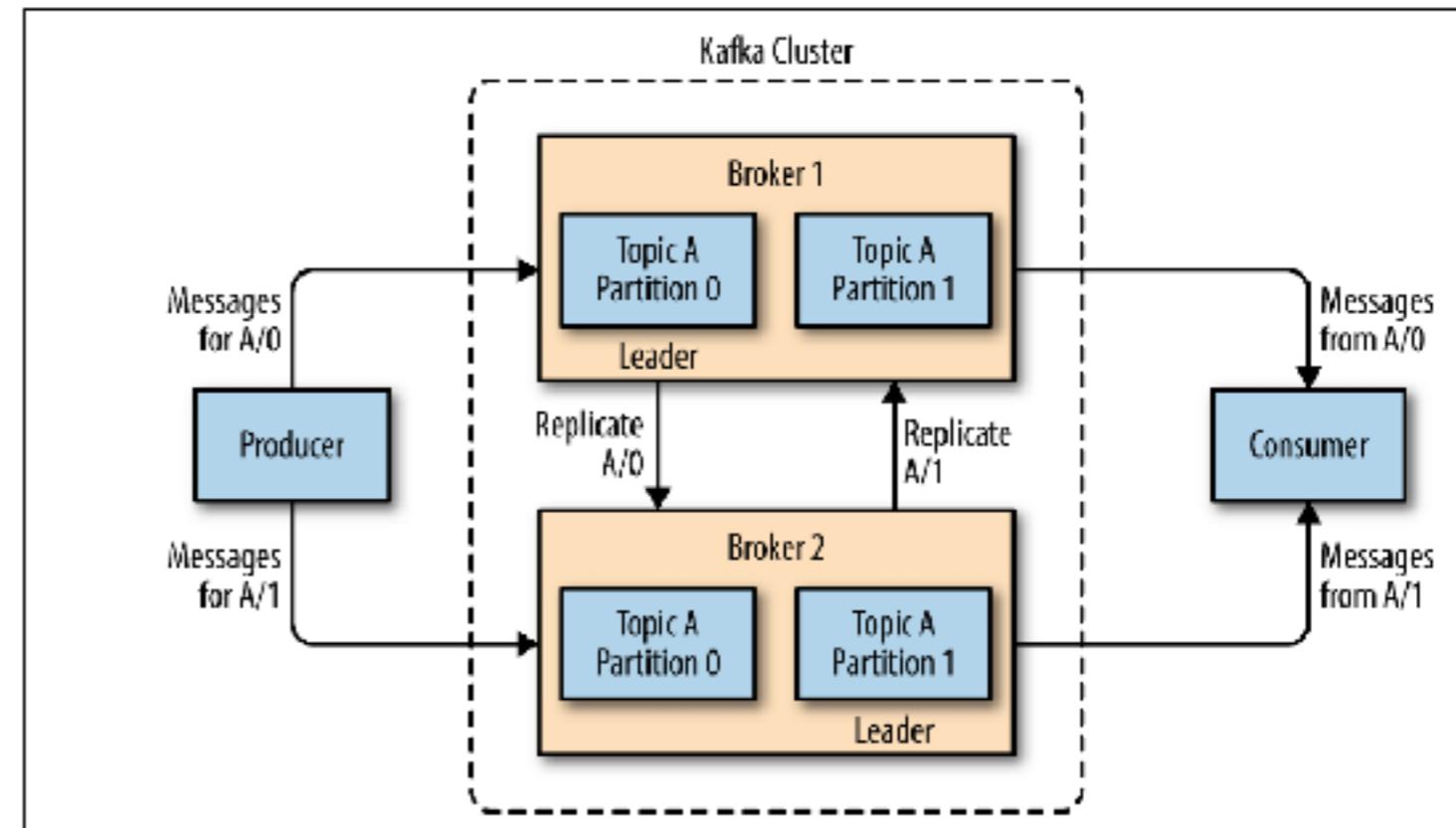
- **Productores:** Crea nuevos mensajes (publicadores o escritores)
  - Generalmente no importa que partición del tópico recibe el mensaje.
  - En particiones basadas en key, todos los mensajes de una key son recibidos por una misma partición.
- **Consumidores:** Leen los mensajes (subscriptores o lectores)
  - Pueden subscribirse a uno o más tópicos, leen los mensajes en orden.
  - Con el offset hace seguimiento de que mensajes ha consumido.
  - Grupos de consumidores asegurar que una partición ha sido consumida por un solo miembro del grupo.





# Kafka: Arquitectura

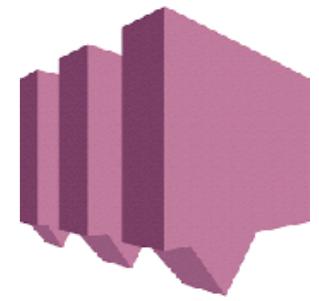
- **Broker:** Un servidor de Kafka.
  - Recibe mensajes de productores.
  - Asigna offset a mensajes
  - Hace commit de mensajes a almacenamiento en disco.
  - Un solo broker puede administrar miles de particiones y millones de mensajes por segundo.



- **Cluster:** Es construido con Kafka brokers, un broker actua como controlador del cluster.
  - Un cluster controla la asignación de particiones a brokers y monitorear fallas.
  - Una partición pertenece a un solo broker (leader), pero puede ser asignada a múltiples (replicación)

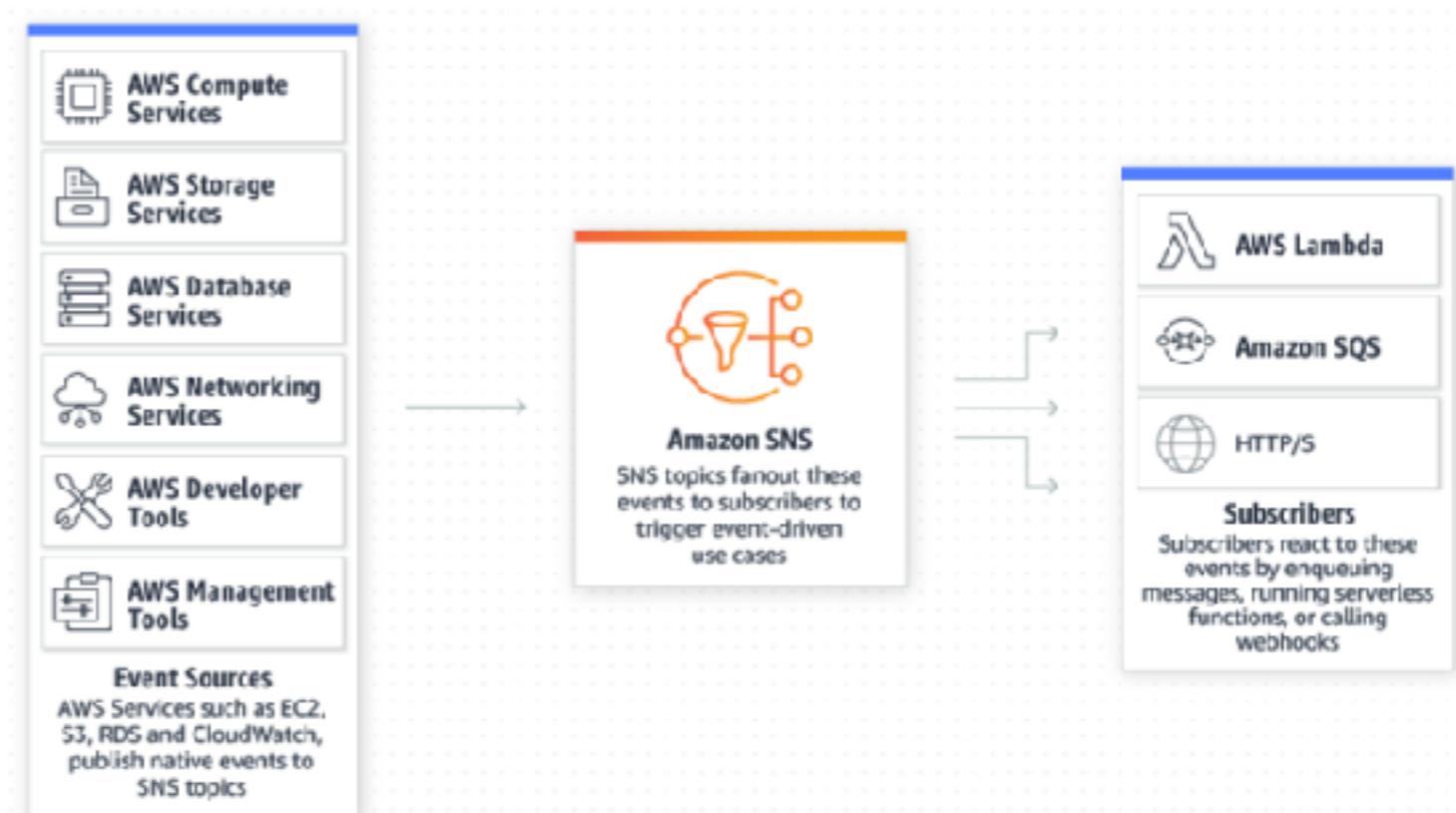


# Amazon SNS



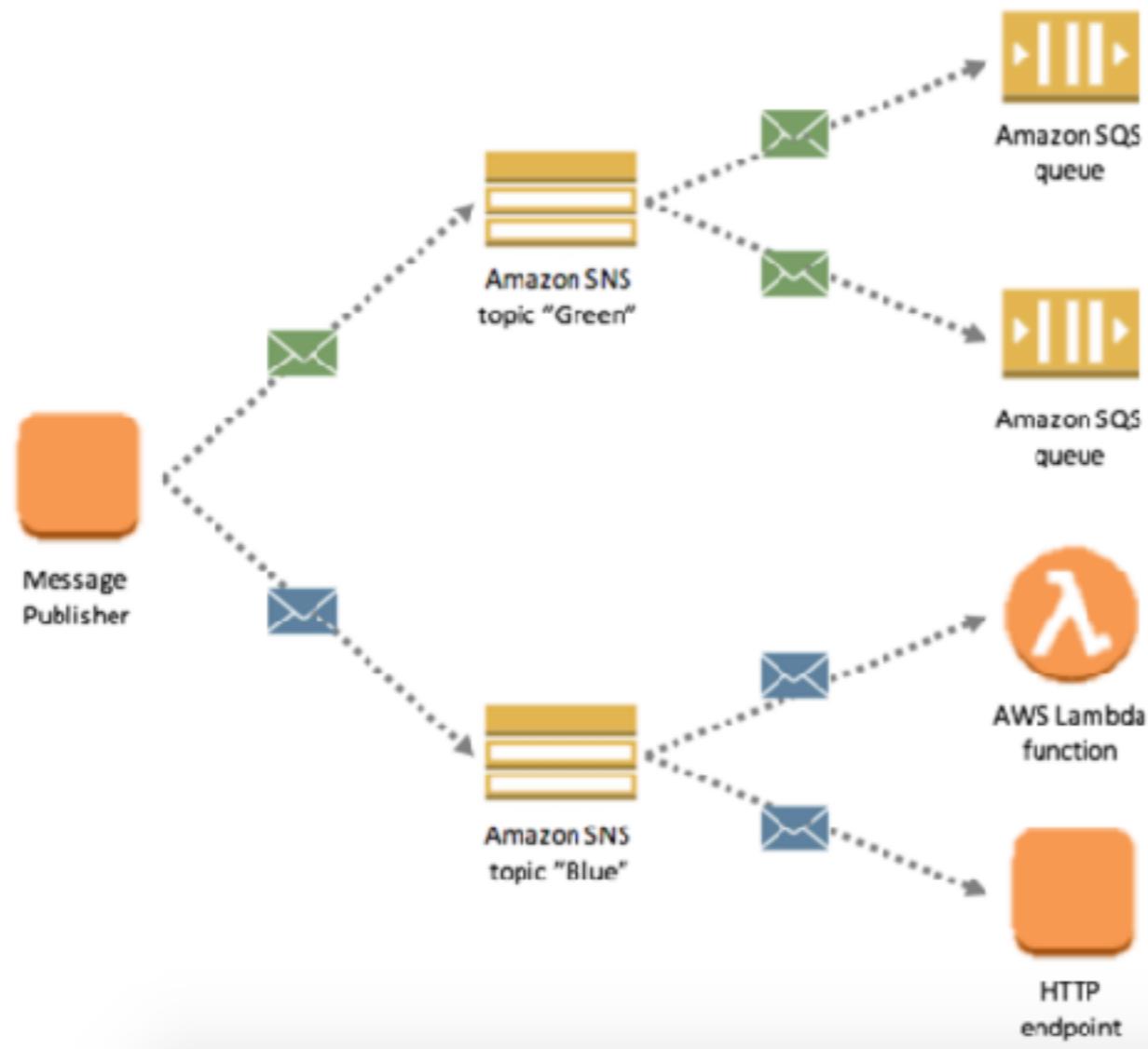
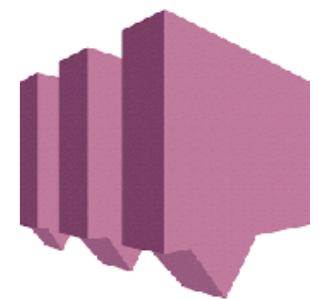
<https://aws.amazon.com/sns/features/>

- Amazon Simple Notification Service.
- Sistema P/S integrado nativamente con fuentes de eventos de AWS (Amazon EC2, Amazon S3, Amazon RDS) -> Publishers
- Integrado con destinos de eventos de AWS (subscriptores) como Amazon SQS y Lambda

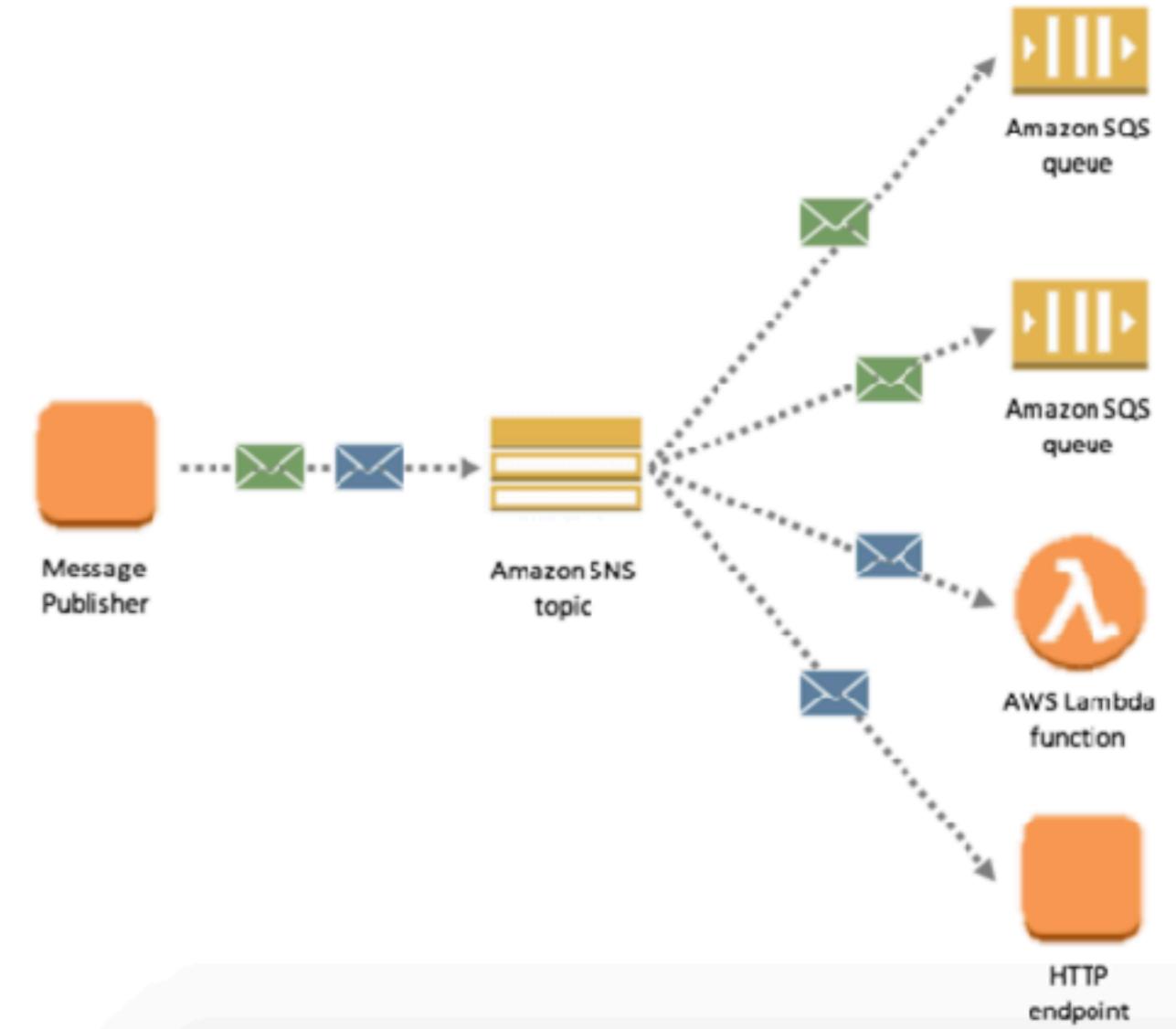




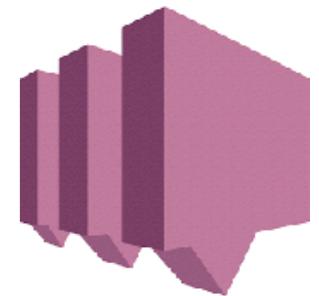
# Amazon SNS



Topic-based



Attribute-based



<https://aws.amazon.com/sns/features/>

- Sport merchandise e-commerce website
- Events: checkout events, buyer navigation events

## Subscripción endpoints

```
payment_gateway_subscription_arn = sns.subscribe(  
    TopicArn = topic_arn,  
    Protocol = 'sqS',  
    Endpoint = 'arn:aws:sqS:ap-southeast-2:123456789012:PaymentQueue'  
)[ 'SubscriptionArn' ]  
  
sns.set_subscription_attributes(  
    SubscriptionArn = payment_gateway_subscription_arn,  
    AttributeName = 'FilterPolicy',  
    AttributeValue = '{"event_type": ["order_placed", "order_cancelled"]}'  
)  
search_engine_subscription_arn = sns.subscribe(  
    TopicArn = topic_arn,  
    Protocol = 'lambda',  
    Endpoint = 'arn:aws:lambda:ap-southeast-2:123456789012:function:SearchIndex'  
)[ 'SubscriptionArn' ]
```

## Create topic

Python

```
topic_arn = sns.create_topic(  
    Name='ShoppingEvents'  
)[ 'TopicArn' ]
```



```
message = '{"product": {"id": 1251, "status": "in_stock"},' \
    ' "buyer": {"id": 4454}}'

sns.publish(
    TopicArn = topic_arn,
    Subject = 'Product Visited #1251',
    Message = message,
    MessageAttributes = {
        'event_type': {
            'DataType': 'String',
            'StringValue': 'product_page_visited'
        }
    }
)

message = '{"order": {"id": 5678, "status": "confirmed", "items": [' \
    ' {"code": "P-9012", "product": "Santos FC Jersey", "units": 1},' \
    ' {"code": "P-3156", "product": "Soccer Ball", "units": 2}]}},' \
    ' "buyer": {"id": 4454}}'

sns.publish(
    TopicArn = topic_arn,
    Subject = 'Order Placed #5678',
    Message = message,
    MessageAttributes = {
        'event_type': {
            'DataType': 'String',
            'StringValue': 'order_placed'
        }
    }
)
```

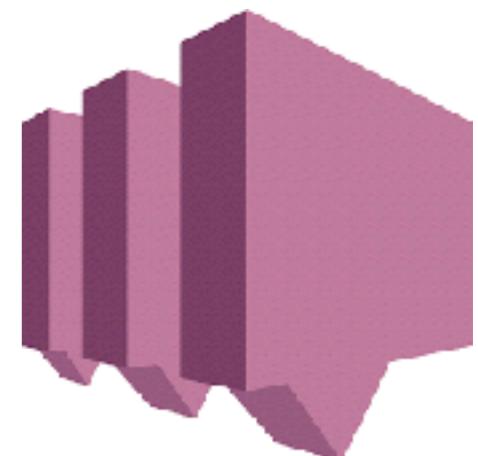




# Comunicación indirecta

1. Sistemas Publish/Subscribe (Apache Kafka, Amazon SNS)
2. Sistemas de recolección (Apache Flume)
3. Sistemas de Colas (RabbitMQ)

Todos son sistemas de comunicación indirecta, dado que proveen escalabilidad para el contexto de Big Data.





# Sistemas de recolección de datos

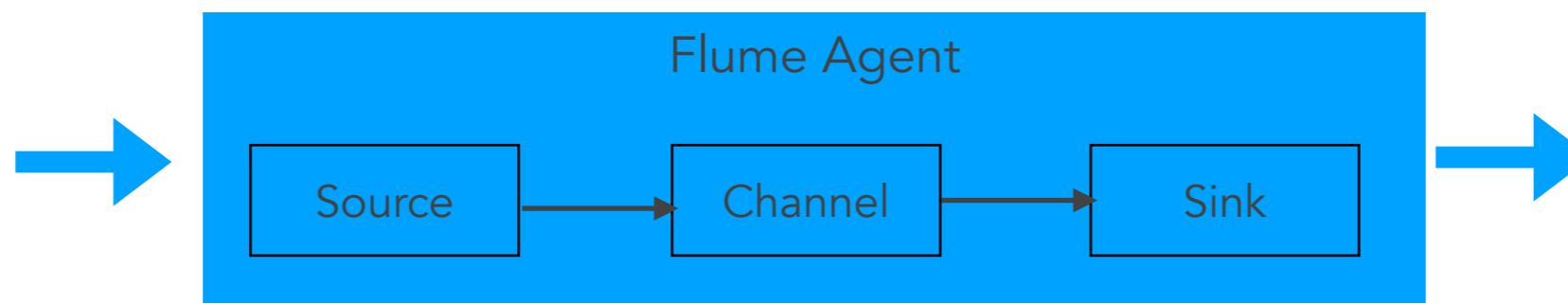
- Permiten recolectar, agregar y mover datos de diferentes fuentes.
- Ejemplos de fuentes: logs de servidores, de redes sociales, de streaming de sensores, de IoT.
- Envío a algún centro de datos centralizado como sistema de archivos distribuido o base de datos.



Sources
Avro Source
Thrift Source
Exec Source
JMS Source
Spooling Directory Source
Twitter Source
NetCat Source
Sequence Generator Source
Syslog Source
HTTP Source
Custom Source

# Apache Flume: Arquitectura

Un flujo puede estar compuesto por múltiples canales, donde una fuente de datos escribe los datos a múltiples canales.



Sinks
HDFS Sink
Avro Sink
Thrift Sink
File Roll Sink
Logger Sink
IRC Sink
HBase Sink
ElasticSearch Sink
Custom Sink

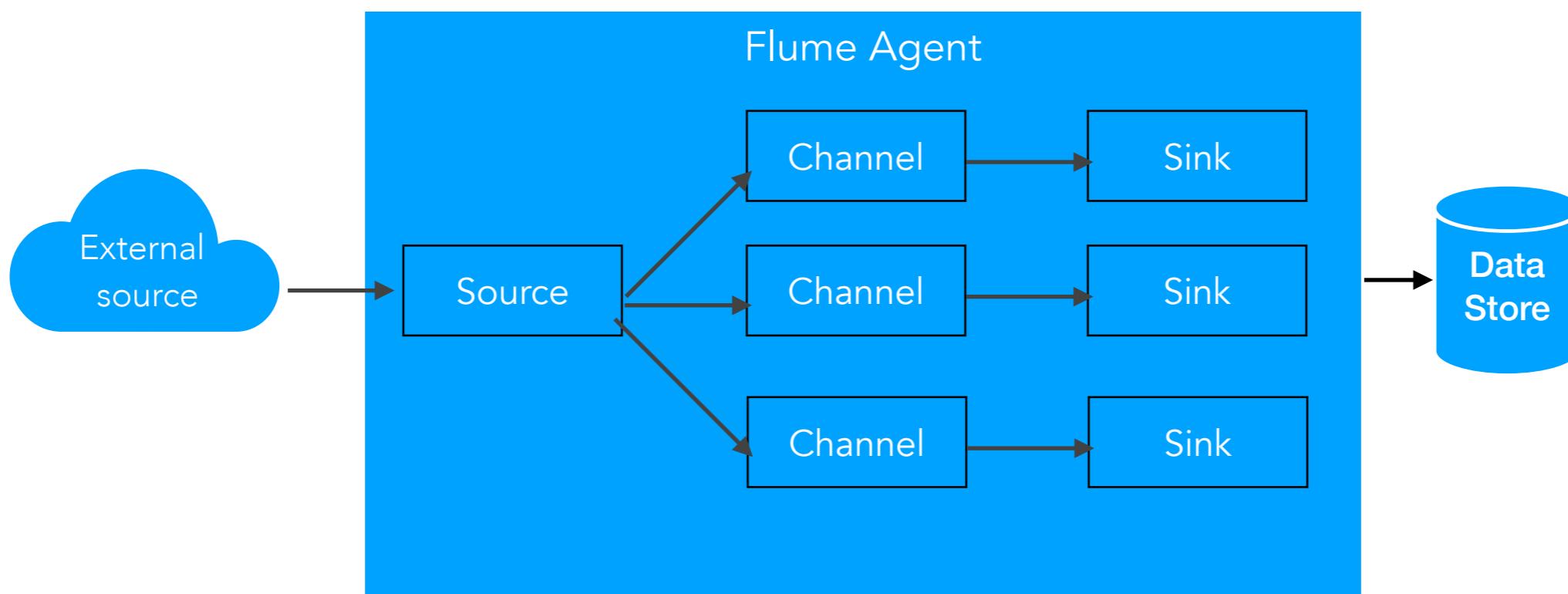
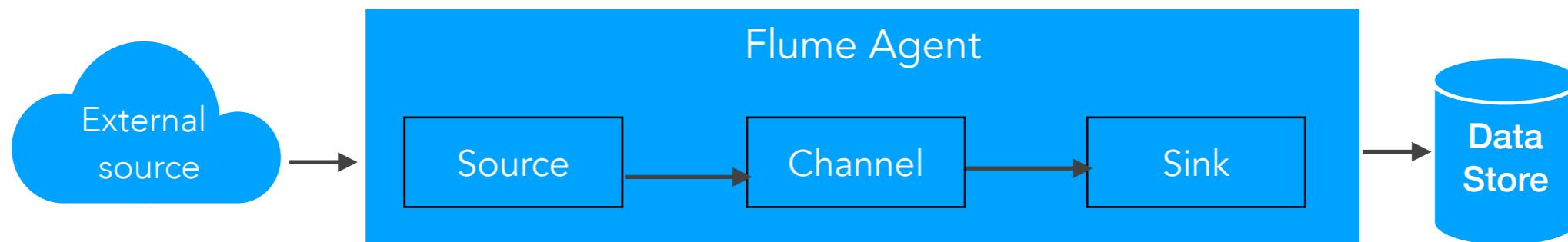
- **Source:** Recibe o consulta (polls) datos de fuentes externas.
- **Channel:** La fuente pasa los datos a un canal, por donde son transmitidos los datos. Cada canal se une a un Sink.
- **Sink:** Componente que drena los datos del canal a un almacenamiento.
- **Agente:** Es un proceso que aloja fuentes, canales y sinks.
- **Evento:** Unidad de datos que fluye, opcionalmente con atributos.



UNIVERSIDAD TECNICA  
FEDERICO SANTA MARIA

Departamento de Informática

# Ejemplos de flujos de datos en Flume

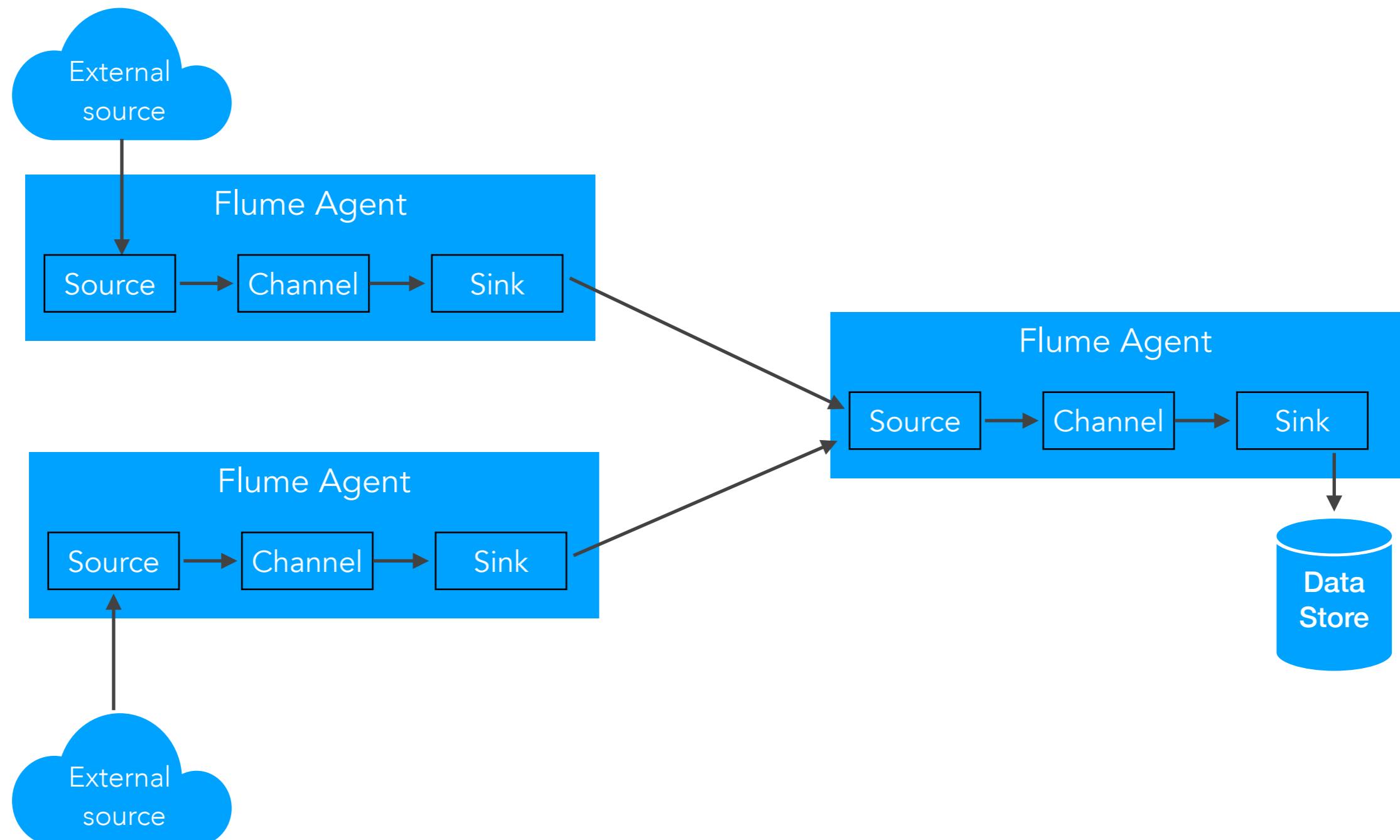




UNIVERSIDAD TECNICA  
FEDERICO SANTA MARIA

Departamento de Informática

# Ejemplos de flujos de datos en Flume

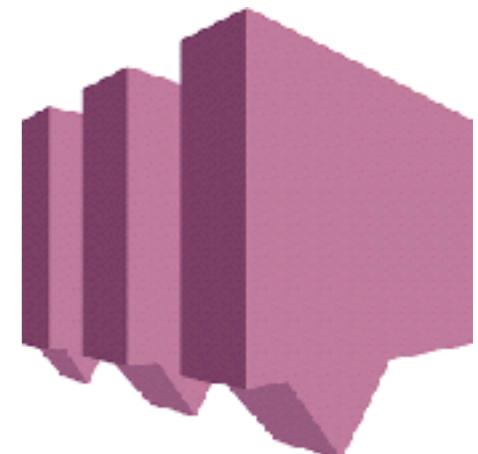




# Comunicación indirecta

1. Sistemas Publish/Subscribe (Apache Kafka, Amazon SNS)
2. Sistemas de recolección (Apache Flume)
3. Sistemas de Colas (RabbitMQ)

Todos son sistemas de comunicación indirecta, dado que proveen escalabilidad para el contexto de Big Data.





# Sistemas de colas de mensajes

- Los sistemas de colas de mensajes ofrecen capacidad de almacenamiento de término intermedio para los mensajes, sin requerir que el emisor o receptor estén activos durante la transmisión.
- Generalmente no hay garantías de cuando el mensaje será leído por el receptor.

Los mensajes son dirigidos a la abstracción de una cola, con receptores que extraen mensajes de las colas.



# Sistemas de colas de mensajes

- **Modelo de programación:**
  - Proceso *productor* puede enviar mensajes a una cola específica.
  - Proceso *consumidor* puede recibir mensajes de esta cola.
- **Estilos de recepción:**
  - Recepción *bloqueante*: el receptor se bloquea hasta que el mensaje está disponible.
  - Recepción *no-bloqueante*: Operación polling, se chequea el estado de la cola y se retorna el mensaje si está disponible.
  - Operación de *notificación*: se usa una notificación de evento cuando un mensaje está disponible en la cola.



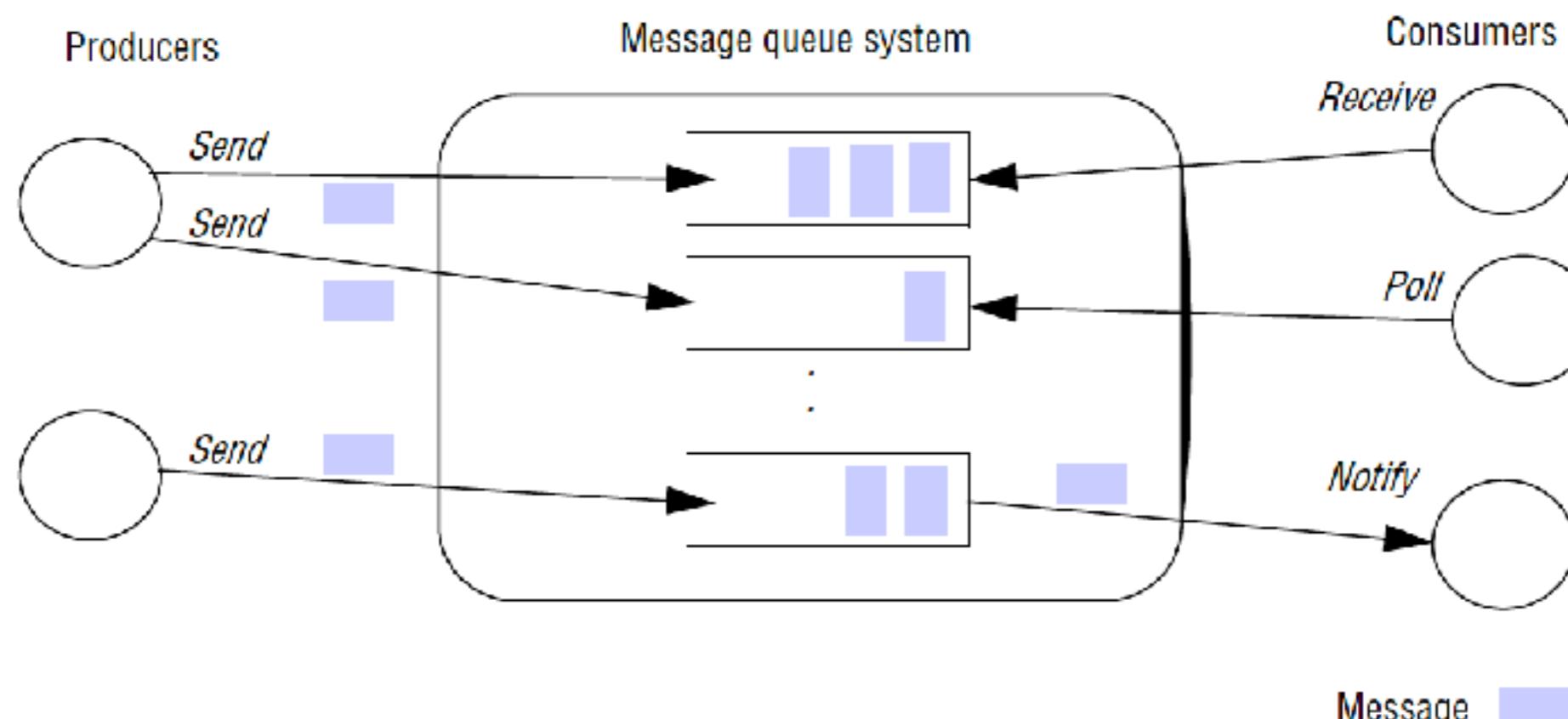
# Interfaz básica de una cola

Operación	Descripción
<b>put</b>	Añade (append) un mensaje a una cola específica.
<b>get</b>	Se bloquea hasta que la cola especificada no está vacía y remueve el primer mensaje.
<b>poll</b>	Se chequea una cola especificada por mensajes, y se remueve el primer. Nunca se bloquea.
<b>Notify</b>	Se instala un handlers para llamar (notificar) cuando un mensaje sea puesto en la cola específica.



# Modelo de programación

- La política de encolamiento es normalmente first-in-first-out (FIFO), pero la mayoría de los sistemas también soportan “prioridad”
- Prioridad: Los mensajes con prioridad mayor son entregados primero.
- Los consumidores pueden seleccionar mensajes de la cola basado en las propiedades del mensaje.



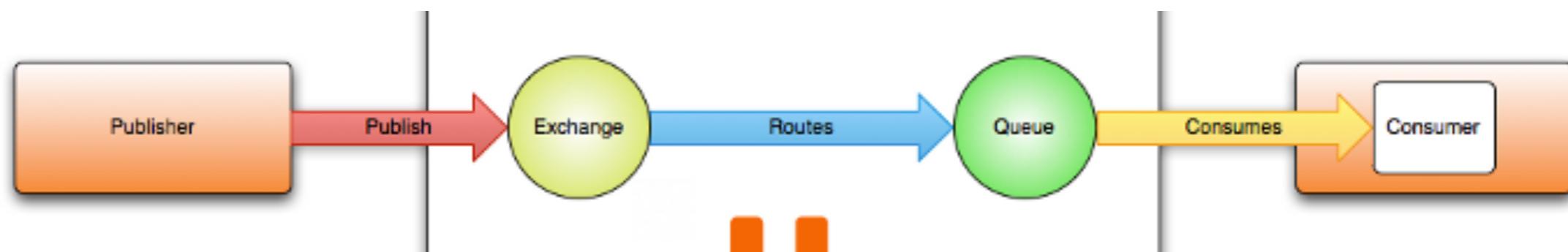


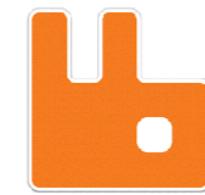
# RabbitMQ

<https://www.rabbitmq.com/>

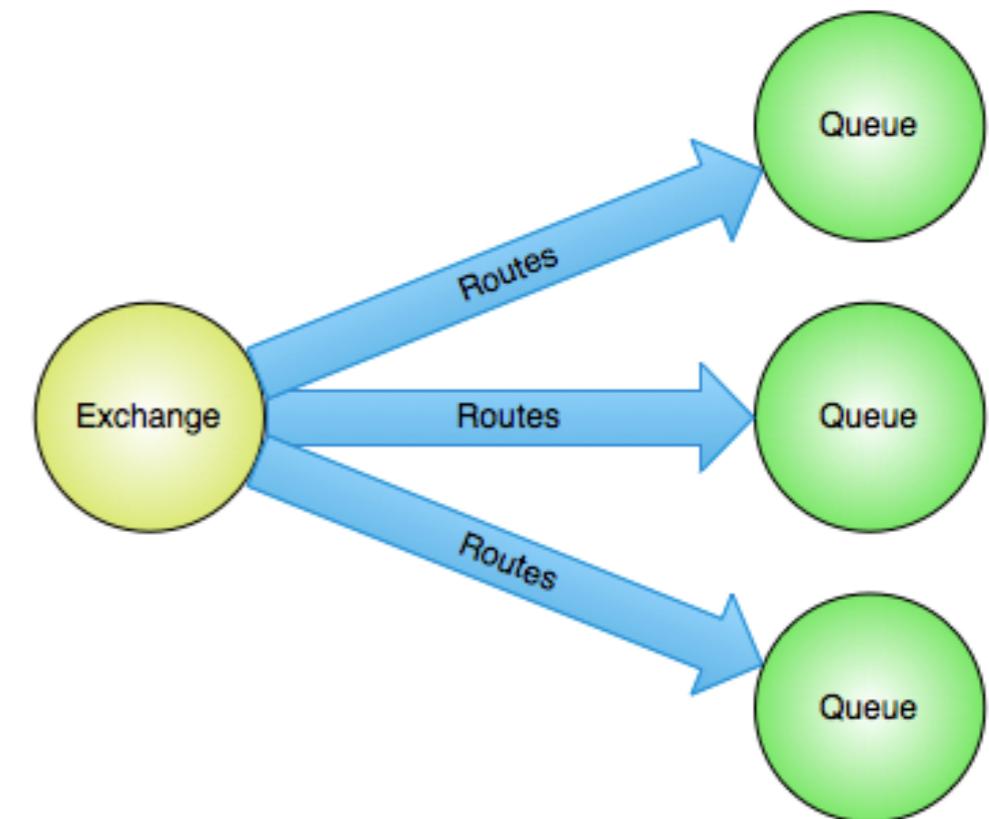
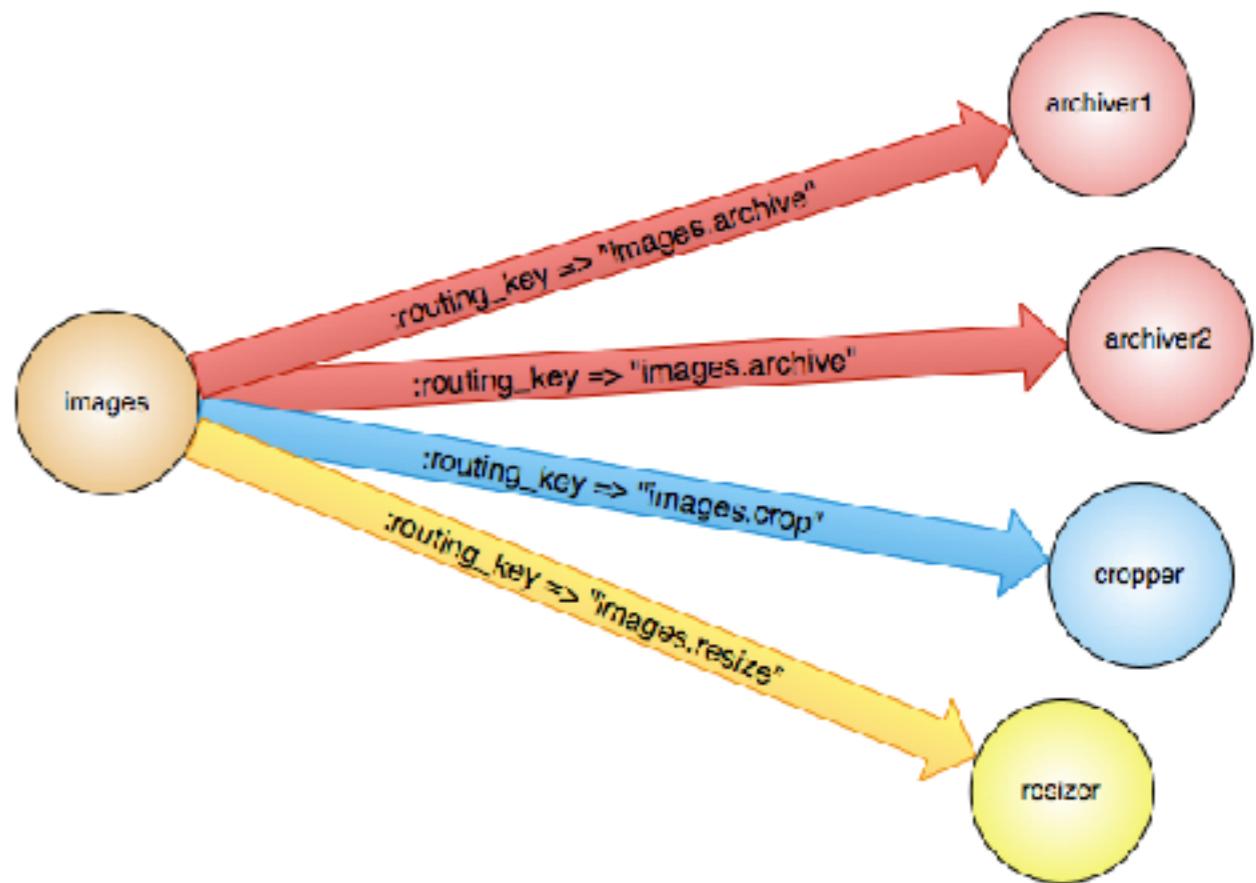
- Usa Message broker
- Se puede ejecutar en cualquier plataforma que soporta Erlang.
- Modelo AMQP (Advanced Message Queuing Protocol) protocolo que permite la comunicación a través de brokers de mensajes (middleware)
- Brokers: reciben mensajes del productor y lo enruta al consumidor.

- Mensajes son publicados en “exchanges” como mailboxes (Poniendo atributos/ metadata)
- Exchanges distribuyen copias del mensaje a colas usando reglas a las que se les llama “bindings”.
- Distintas opciones de entrega: Entrega a consumidores o los consumidores van a buscar el mensaje desde las colas a demanda.
- Usa ACK cuando un mensaje es entregado a un consumidor. Solo se elimina un mensaje cuando todos los ACKs han llegado





- Ejemplos de exchanges
  - Direct exchange: message routing key.
  - Fanout exchange: Encamina a todas las colas.
- Propiedades Colas: Nombre, durable, exclusiva, auto-delete.
- Bindings: Routing key (filtro)





## Sending Hello World.

- Para Go -> uso de paquete amqp
- Un mensaje enviado desde un productor es llamado "Publishing".
- La declaración de una cola es idempotente.
- El contenido del mensaje es un byte array (listo para enviar).

```
func main() {
    conn, err := amqp.Dial("amqp://guest:guest@localhost:5672/")
    failOnErr(err, "Failed to connect to RabbitMQ")
    defer conn.Close()

    ch, err := conn.Channel()
    failOnErr(err, "Failed to open a channel")
    defer ch.Close()

    q, err := ch.QueueDeclare(
        "hello", // name
        false,   // durable
        false,   // delete when unused
        false,   // exclusive
        false,   // no-wait
        nil,     // arguments
    )
    failOnErr(err, "Failed to declare a queue")

    body := "Hello World!"
    err = ch.Publish(
        "",      // exchange
        q.Name, // routing key
        false,   // mandatory
        false,   // immediate
        amqp.Publishing{
            ContentType: "text/plain",
            Body:         []byte(body),
        })
    log.Printf(" [x] Sent %s", body)
    failOnErr(err, "Failed to publish a message")
}
```

Comienza conexión

Channel: pool de conexiones

Declaración de la cola

Publica un mensaje en la cola



## Receiving Hello World.

- Se conecta a Rabbit y canal igual que el productor.
- Porqué se declara denuevo la cola
- Un mensaje recibido por un consumidor es llamado "delivery".

Itera sobre los  
mensajes  
recibidos

```
q, err := ch.QueueDeclare(  
    "hello", // name  
    false, // durable  
    false, // delete when unused  
    false, // exclusive  
    false, // no-wait  
    nil, // arguments  
)  
failOnError(err, "Failed to declare a queue")  
  
msgs, err := ch.Consume(  
    q.Name, // queue  
    "", // consumer  
    true, // auto-ack  
    false, // exclusive  
    false, // no-local  
    false, // no-wait  
    nil, // args  
)  
failOnError(err, "Failed to register a consumer")  
  
forever := make(chan bool)  
go func() {  
    for d := range msgs {  
        log.Printf("Received a message: %s", d.Body)  
    }  
}()  
  
log.Printf(" [*] Waiting for messages. To exit press CTRL+C")  
<-forever
```

Declaración de la cola

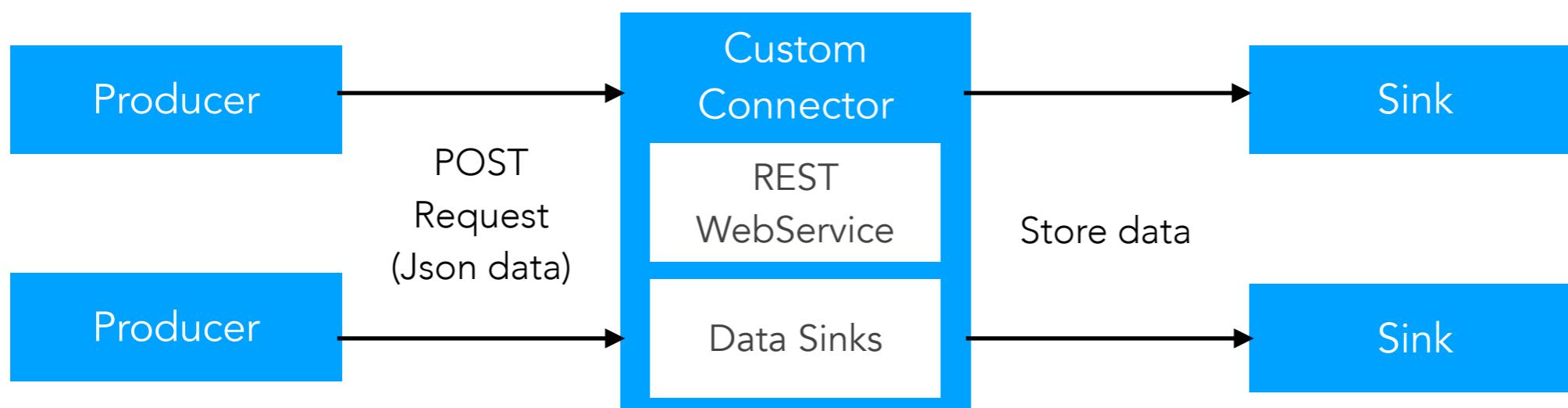
Lee los mensajes de la cola

Para que no se salga sin esperar



# Conectores basados en REST

- Se ha desarrollado conectores personalizados para requerimientos específicos de aplicaciones.
  - Conectores basados en REST exponen el servicio web REST.
  - Los productores pueden publicar datos usando HTTP POST.
  - Los datos se almacenan en el Sink (sistema de archivos, cloud, etc).
  - La consulta HTTP se procesa y se almacenan datos.
  - Ventajas, cualquier cliente puse usarlo y no tienen estado.
  - Desventaja, no adecuado para alto rendimiento.





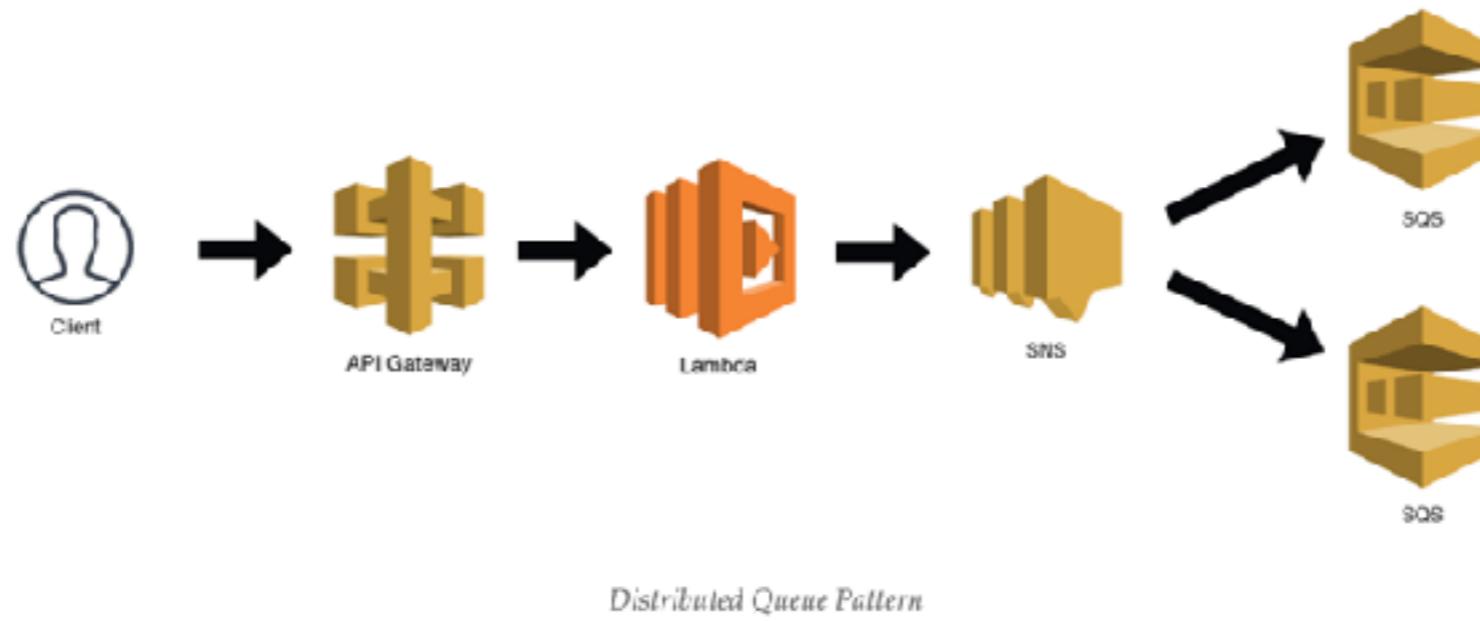
# Conectores basados en MQTT

- MQTT significa MQ Telemetry Transport y es un protocolo de publish/subscribe liviano, especialmente diseñado para dispositivos con restringidos.
- Se usa en IoT para enviar lo que sienten los sensores a un servidor o a la nube.
- Componentes:
  - Publisher: Publica datos a un tópico que administra un broker.
  - Broker/Servidor: Administra un tópico y redirecciona los datos recibidos en un tópico a todos los subscriptores de ese tópico.
  - Subscriber: Subscrive a tópicos y recibe datos publicados en estos por los publicadores.



# Examples of indirect communication

## Distributed Queue Pattern



- **Standard Queue:** Unlimited throughput, at least once delivery, best effort ordering.
- **FIFO Queue:** 300 messages per second (batch may increase this two 3000), exactly once delivery, FIFO order.



# Examples of indirect communication

## Dead Letter Queue

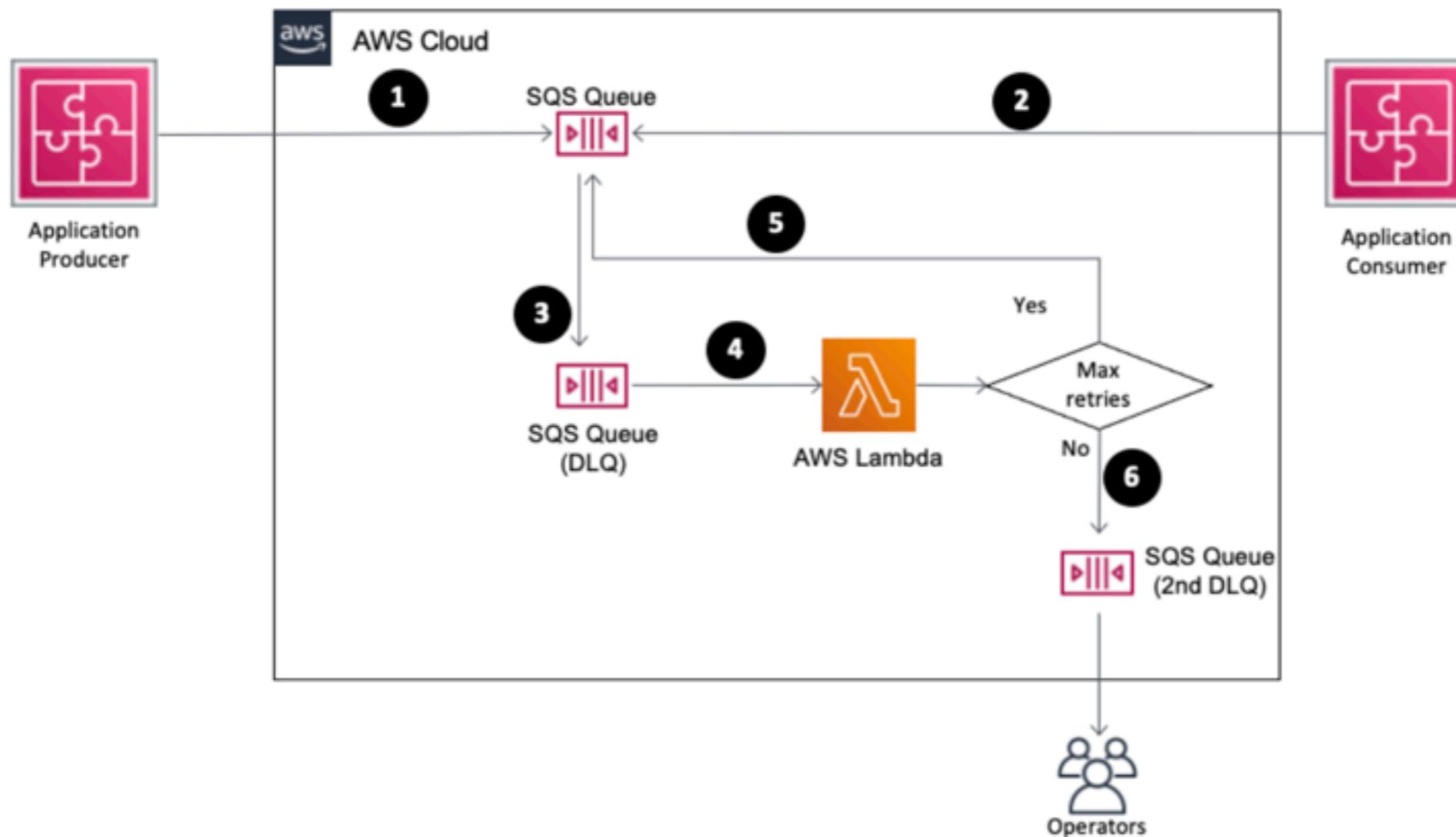


- Twilio API: API Rest to send SMS messages
- DLQ: Dead Letter Queue
- RDS Aurora: Rela



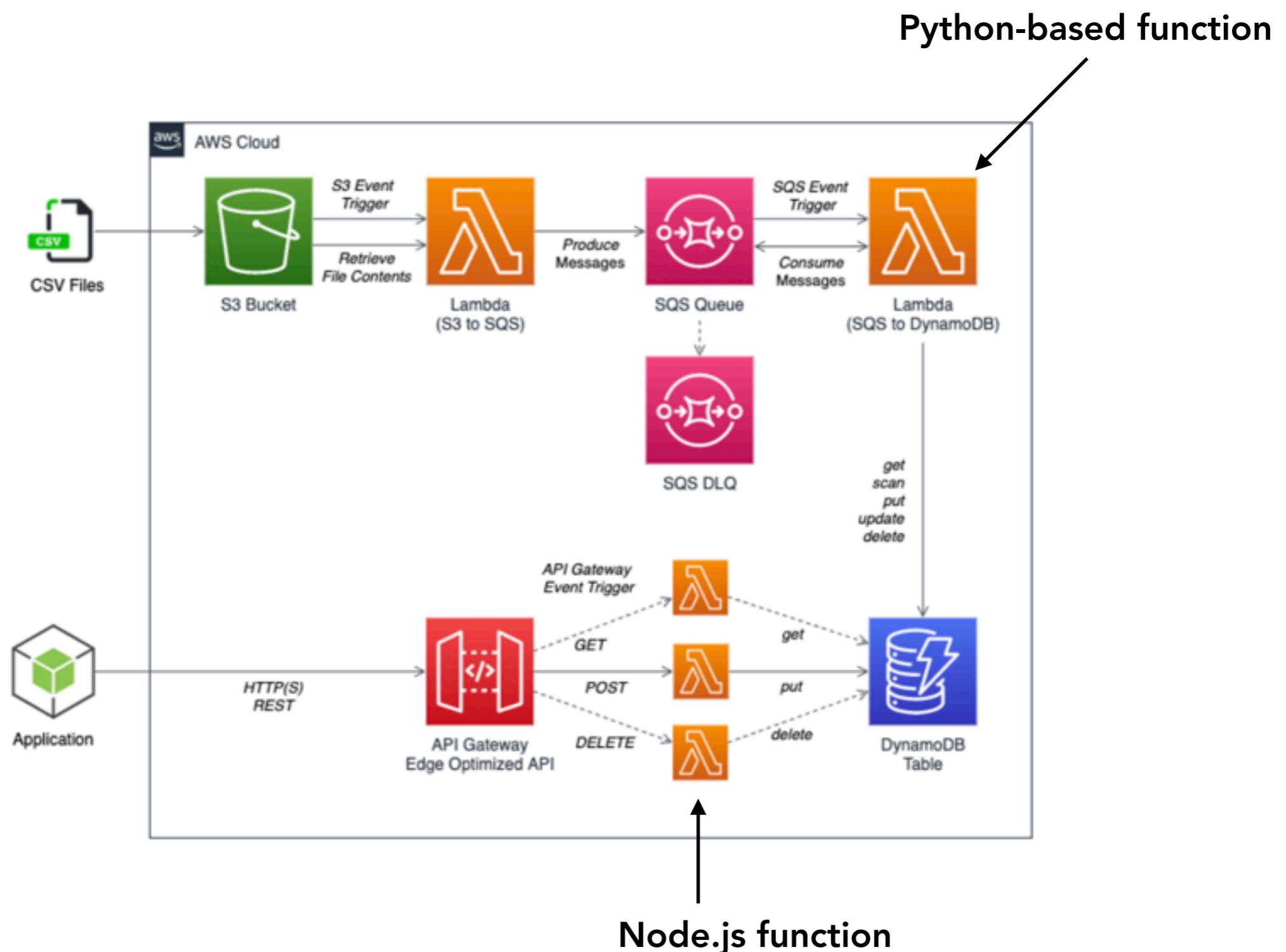
# Examples of indirect communication

## 2nd Dead Letter Queue





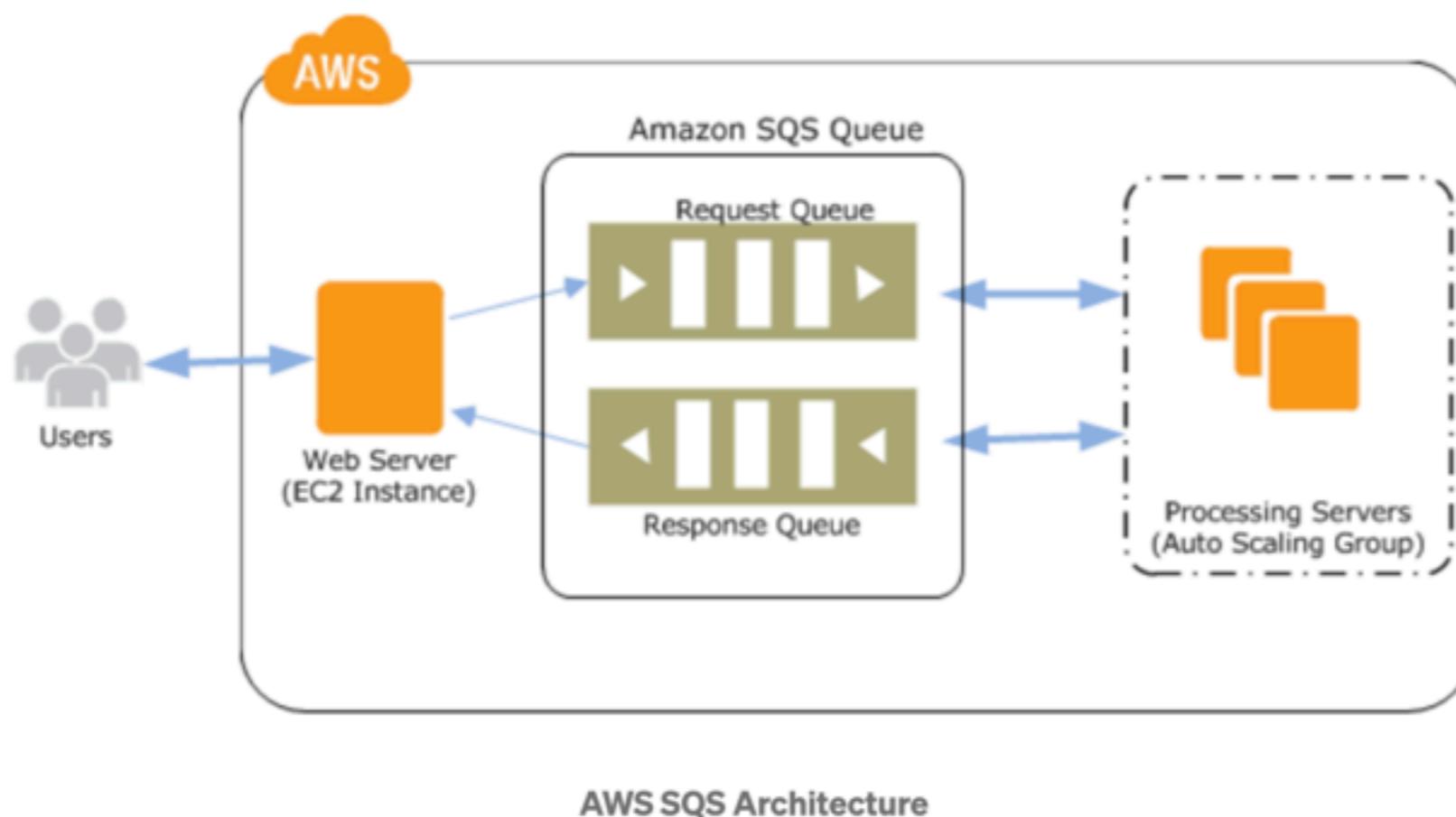
# Event-driven serverless architecture





# Examples of indirect communication

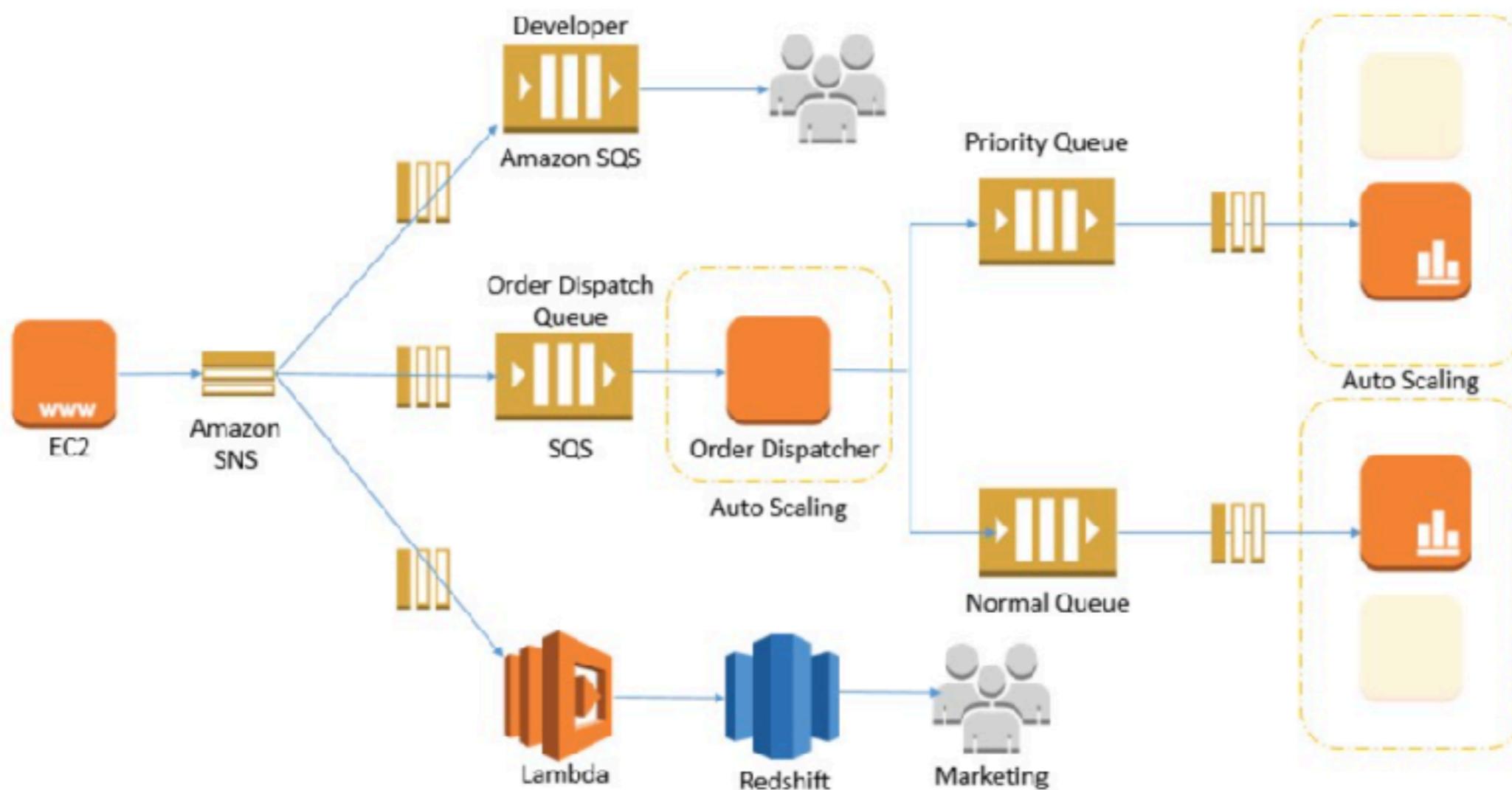
## Not only serverless





# Examples of indirect communication

## Not only serverless





# SNS Example:Receive Phone Call Alerts for AWS Account Security Events With Amazon Polly

