

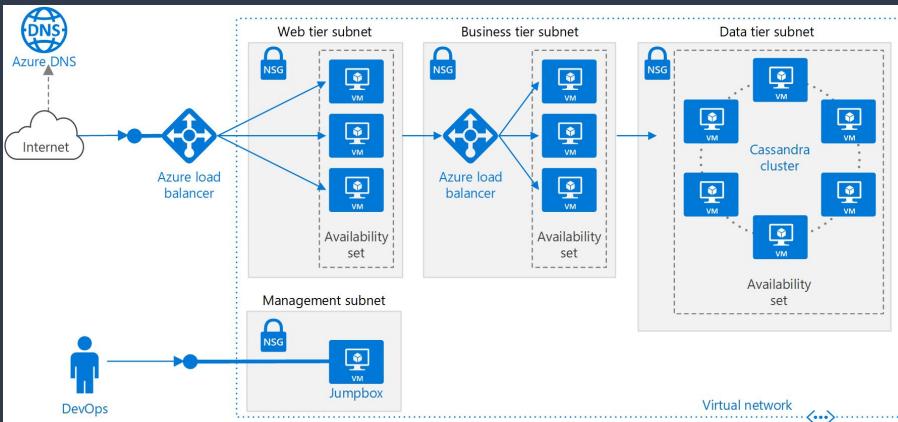
# Cloud Computing Usage Patterns

El punto de vista del cloud consumer

1er cuatrimestre 2023



# Cloudgenda



- Repaso intro a cloud
- Ejemplo Cloud Provider: AWS
- Estilos de Arquitectura habituales
  - N-tier
  - Web-Queue-Worker
  - Microservices
  - Event-Driven
- Usage Patterns
  - Auto-Scaling
  - Health Endpoint
  - Competing Consumers
  - Queue-Based Load Leveling
  - Priority Queue
  - Throttling
  - Retry
  - Circuit Breaker
  - Sharding
- Algunas arquitecturas de referencia AWS
  - Web App
  - Batch
  - Media sharing

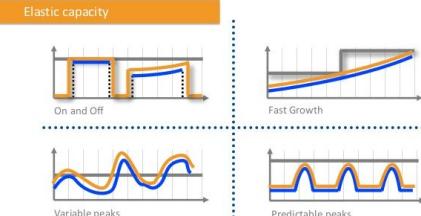
# Repaso

## Definición

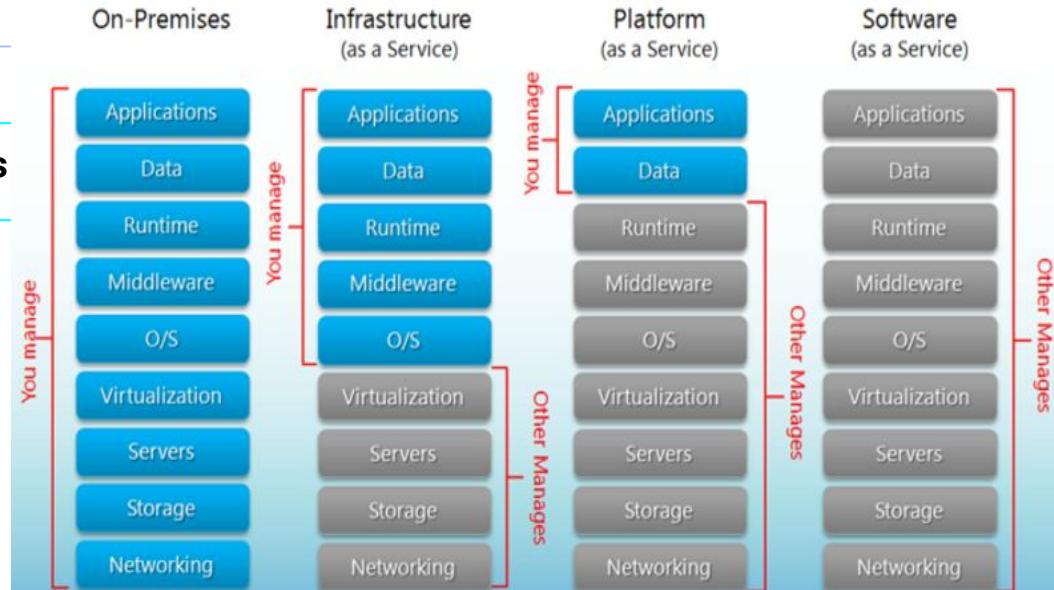
Forma especializada de **computación distribuida**, que introduce **modelos de utilización** para la **provisión remota** de **recursos escalables y medibles**

## Características

1. On-Demand usage
2. Ubiquitous Access
3. Multitenancy
4. Elasticity
5. Measured Usage
6. Resiliency

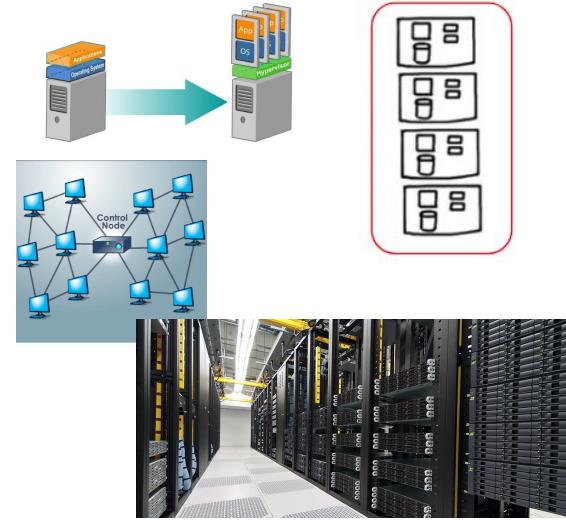


## Delivery Models



## Tecnologías

1. Virtualización
2. Clustering
3. Grid Computing
4. Internet
5. Web
6. Data Centers
7. Multitenant
8. Services



# Ejemplo de Cloud Provider: AWS – Amazon Web Services

# AWS EC2 Console

Provisión remota  
de recursos  
escalables y  
medibles

https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#instances:sort=instanceId

Services ▾ Resource Groups ▾ Guillermo\_Rugilo@... co... N. Virginia ▾ Support ▾

EC2 Dashboard Events Tags Reports Limits

INSTANCES Instances Launch Templates Spot Requests Reserved Instances Dedicated Hosts Scheduled Instances

IMAGES AMIs Bundle Tasks

ELASTIC BLOCK STORE Volumes Snapshots Lifecycle Manager

NETWORK & SECURITY Security Groups Elastic IPs Placement Groups Key Pairs Network Interfaces

LOAD BALANCING Load Balancers Target Groups

AUTO SCALING Launch Configurations

Launch Instance Connect Actions ▾

Filter by tags and attributes or search by keyword

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public IP
DEV-MCS-Fair-Queuing-Worker-develop	i-003a996d...	t2.small	us-east-1a	running	2/2 checks ...	None	
QA-MCS-CI-Files-Fry-UI	i-006a8aec...	t2.small	us-east-1c	terminated			
DEV-MCS-Access-Logs-ActivityProcessor	i-0120ad9d...	t2.small	us-east-1a	running	2/2 checks ...	None	
DEV-WOWZA-CF	i-0135b679...	t2.small	us-east-1c	running	2/2 checks ...	None	ec2-3...
DEV-MCS-CI-API-Public-develop	i-0174db3f...	t2.small	us-east-1c	running	2/2 checks ...	None	
QA-APODSTATS-PUSH-AS	i-018c9191...	t2.micro	us-east-1c	running	2/2 checks ...	None	
QA-MONGO-RSA2-26APR2017	i-01948855...	m3.large	us-east-1d	running	2/2 checks ...	OK	
DEV-MCS-CPS-WORKER-LINUX	i-01eaa809...	m3.2xlarge	us-east-1d	running	2/2 checks ...	None	
DEV-CI-UpdatePropagation-Worker-develop	i-026fd81a...	t2.small	us-east-1d	running	2/2 checks ...	None	

Instance: i-0174db3f9c3a1a0cb (DEV-MCS-CI-API-Public-develop) Private IP: 10.0.59.223

Description Status Checks Monitoring Tags

CloudWatch alarms: No alarms configured Create Alarm

CloudWatch metrics: Detailed monitoring. Disable Detailed Monitoring Showing data for: Last 24 Hours

Below are your CloudWatch metrics for the selected resources (a maximum of 10). Click on a graph to see an expanded view. All times shown are in UTC.

CPU Utilization (Percent)

Disk Reads (Bytes)

Disk Read Operations (Operations)

Disk Writes (Bytes)

Disk Write Operations (Operations)

Network In (Bytes)

Network Out (Bytes)

# AWS EC2 On-Demand Pricing

Modelos de utilización

Linux	RHEL	SLES	Windows	Windows with SQL Standard	Windows with SQL Web
Windows with SQL Enterprise	Linux with SQL Standard	Linux with SQL Web	Linux with SQL Enterprise		
Region: US East (N. Virginia) ▾					
vCPU	ECU	Memory (GiB)	Instance Storage (GB)	Linux/UNIX Usage	
<strong>General Purpose - Current Generation</strong>					
t3.nano	2	Variable	0.5 GiB	EBS Only	\$0.0052 per Hour
t3.micro	2	Variable	1 GiB	EBS Only	\$0.0104 per Hour
t3.small	2	Variable	2 GiB	EBS Only	\$0.0208 per Hour
t3.medium	2	Variable	4 GiB	EBS Only	\$0.0416 per Hour
t3.large	2	Variable	8 GiB	EBS Only	\$0.0832 per Hour
t3.xlarge	4	Variable	16 GiB	EBS Only	\$0.1664 per Hour
t3.2xlarge	8	Variable	32 GiB	EBS Only	\$0.3328 per Hour
t2.nano	1	Variable	0.5 GiB	EBS Only	\$0.0058 per Hour
t2.micro	1	Variable	1 GiB	EBS Only	\$0.0116 per Hour
t2.small	1	Variable	2 GiB	EBS Only	\$0.023 per Hour
t2.medium	2	Variable	4 GiB	EBS Only	\$0.0464 per Hour
t2.large	2	Variable	8 GiB	EBS Only	\$0.0928 per Hour
t2.xlarge	4	Variable	16 GiB	EBS Only	\$0.1856 per Hour
t2.2xlarge	8	Variable	32 GiB	EBS Only	\$0.3712 per Hour
m5.large	2	8	8 GiB	EBS Only	\$0.096 per Hour
m5.xlarge	4	16	16 GiB	EBS Only	\$0.192 per Hour

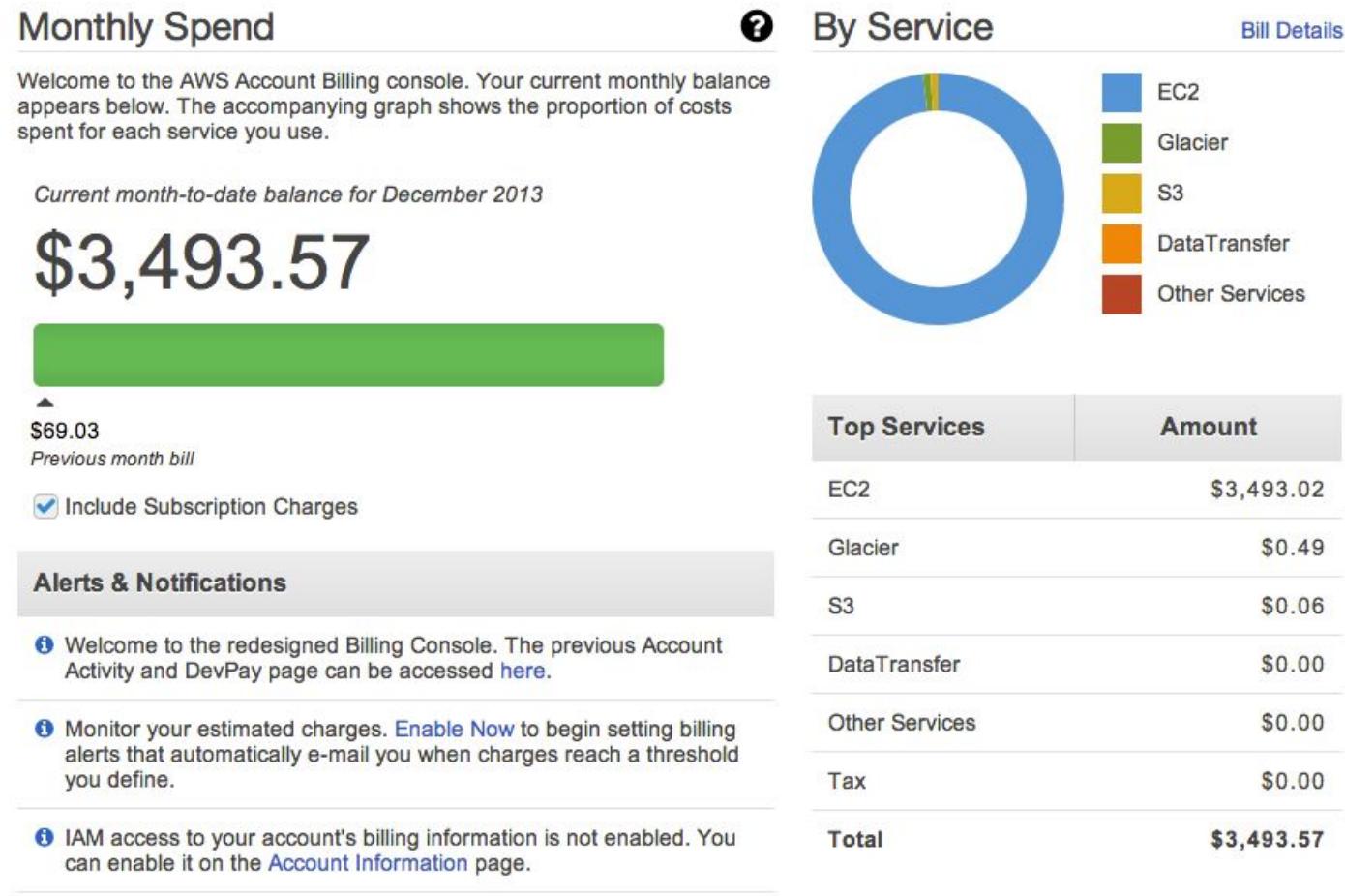
# AWS

Muchísimos cloud services

All services
 <a href="#">Compute</a>
 <a href="#">EC2</a>
 <a href="#">Lightsail</a>
 <a href="#">Lambda</a>
 <a href="#">Batch</a>
 <a href="#">Elastic Beanstalk</a>
 <a href="#">Serverless Application Repo...</a>
 <a href="#">AWS Outposts</a>
 <a href="#">EC2 Image Builder</a>
 <a href="#">AWS App Runner</a>
 <a href="#">Storage</a>
 <a href="#">S3</a>
 <a href="#">EFS</a>
 <a href="#">FSx</a>
 <a href="#">S3 Glacier</a>
 <a href="#">Storage Gateway</a>
 <a href="#">AWS Backup</a>
 <a href="#">Database</a>
 <a href="#">RDS</a>
 <a href="#">DynamoDB</a>
 <a href="#">ElastiCache</a>
 <a href="#">Neptune</a>
 <a href="#">Amazon QLDB</a>
 <a href="#">Amazon DocumentDB</a>
 <a href="#">Amazon Keyspaces</a>
 <a href="#">Amazon Timestream</a>
 <a href="#">Migration &amp; Transfer</a>
 <a href="#">AWS Migration Hub</a>
 <a href="#">AWS Application Migration ...</a>
 <a href="#">Application Discovery Service</a>
 <a href="#">Database Migration Service</a>
 <a href="#">Server Migration Service</a>
 <a href="#">AWS Transfer Family</a>
 <a href="#">AWS Snow Family</a>
 <a href="#">DataSync</a>
 <a href="#">Networking &amp; Content Delivery</a>
 <a href="#">VPC</a>
 <a href="#">CloudFront</a>
 <a href="#">Route 53</a>
 <a href="#">API Gateway</a>
 <a href="#">Direct Connect</a>
 <a href="#">AWS App Mesh</a>
 <a href="#">AWS Cloud Map</a>
 <a href="#">Global Accelerator</a>
 <a href="#">Developer Tools</a>
 <a href="#">CodeStar</a>
 <a href="#">CodeCommit</a>
 <a href="#">CodeArtifact</a>
 <a href="#">CodeBuild</a>
 <a href="#">CodeDeploy</a>
 <a href="#">CodePipeline</a>
 <a href="#">Cloud9</a>
 <a href="#">CloudShell</a>
 <a href="#">X-Ray</a>
 <a href="#">AWS FIS</a>
 <a href="#">Robotics</a>
 <a href="#">AWS RoboMaker</a>
 <a href="#">Customer Enablement</a>
 <a href="#">AWS IQ</a>
 <a href="#">Support</a>
 <a href="#">Managed Services</a>
 <a href="#">Activate for Startups</a>
 <a href="#">Blockchain</a>
 <a href="#">Amazon Managed Blockchain</a>
 <a href="#">Satellite</a>
 <a href="#">Ground Station</a>
 <a href="#">Quantum Technologies</a>
 <a href="#">Amazon Braket</a>
 <a href="#">Management &amp; Governance</a>
 <a href="#">AWS Organizations</a>
 <a href="#">CloudWatch</a>
 <a href="#">AWS Auto Scaling</a>
 <a href="#">CloudFormation</a>
 <a href="#">CloudTrail</a>
 <a href="#">Config</a>
 <a href="#">OpsWorks</a>
 <a href="#">Service Catalog</a>
 <a href="#">Systems Manager</a>
 <a href="#">AWS AppConfig</a>
 <a href="#">Trusted Advisor</a>
 <a href="#">Control Tower</a>
 <a href="#">AWS License Manager</a>
 <a href="#">AWS Well-Architected Tool</a>
 <a href="#">Personal Health Dashboard</a>
 <a href="#">AWS Chatbot</a>
 <a href="#">Launch Wizard</a>
 <a href="#">AWS Compute Optimizer</a>
 <a href="#">Resource Groups &amp; Tag Editor</a>
 <a href="#">Analytics</a>
 <a href="#">Athena</a>
 <a href="#">Amazon Redshift</a>
 <a href="#">EMR</a>
 <a href="#">CloudSearch</a>
 <a href="#">Elasticsearch Service</a>
 <a href="#">Kinesis</a>
 <a href="#">Quicksight</a>
 <a href="#">Data Pipeline</a>
 <a href="#">AWS Data Exchange</a>
 <a href="#">AWS Glue</a>
 <a href="#">AWS Lake Formation</a>
 <a href="#">MSK</a>
 <a href="#">AWS Glue DataBrew</a>
 <a href="#">Amazon FinSpace</a>
 <a href="#">Security, Identity, &amp; Compliance</a>
 <a href="#">IAM</a>
 <a href="#">Resource Access Manager</a>
 <a href="#">Cognito</a>
 <a href="#">Secrets Manager</a>
 <a href="#">GuardDuty</a>
 <a href="#">Inspector</a>
 <a href="#">Amazon Macie</a>
 <a href="#">AWS Single Sign-On</a>
 <a href="#">Certificate Manager</a>
 <a href="#">Key Management Service</a>
 <a href="#">CloudHSM</a>
 <a href="#">Directory Service</a>
 <a href="#">WAF &amp; Shield</a>
 <a href="#">AWS Firewall Manager</a>
 <a href="#">Artifact</a>
 <a href="#">Security Hub</a>
 <a href="#">Detective</a>
 <a href="#">AWS Audit Manager</a>
 <a href="#">AWS Signer</a>
 <a href="#">AWS Network Firewall</a>
 <a href="#">Front-end Web &amp; Mobile</a>
 <a href="#">AWS Amplify</a>
 <a href="#">Mobile Hub</a>
 <a href="#">AWS AppSync</a>
 <a href="#">Device Farm</a>
 <a href="#">Amazon Location Service</a>
 <a href="#">AR &amp; VR</a>
 <a href="#">Amazon Sumerian</a>
 <a href="#">Application Integration</a>
 <a href="#">Step Functions</a>
 <a href="#">Amazon AppFlow</a>
 <a href="#">Amazon EventBridge</a>
 <a href="#">Amazon MQ</a>
 <a href="#">Simple Notification Service</a>
 <a href="#">Simple Queue Service</a>
 <a href="#">SWF</a>
 <a href="#">Managed Apache Airflow</a>
 <a href="#">AWS Cost Management</a>
 <a href="#">AWS Cost Explorer</a>
 <a href="#">AWS Budgets</a>
 <a href="#">AWS Marketplace Subscripti...</a>
 <a href="#">AWS Application Cost Profiler</a>
 <a href="#">Business Applications</a>
 <a href="#">Amazon Connect</a>
 <a href="#">Amazon Pinpoint</a>
 <a href="#">Amazon Honeycode</a>
 <a href="#">Amazon Chime</a>
 <a href="#">Amazon Simple Email Service</a>
 <a href="#">Amazon WorkDocs</a>
 <a href="#">Amazon WorkMail</a>
 <a href="#">Alexa for Business</a>
 <a href="#">End User Computing</a>
 <a href="#">WorkSpaces</a>
 <a href="#">AppStream 2.0</a>
 <a href="#">WorkLink</a>
 <a href="#">Internet of Things</a>
 <a href="#">IoT Core</a>
 <a href="#">FreeRTOS</a>
 <a href="#">IoT 1-Click</a>
 <a href="#">IoT Analytics</a>
 <a href="#">IoT Device Defender</a>
 <a href="#">IoT Device Management</a>
 <a href="#">IoT Events</a>
 <a href="#">IoT Greengrass</a>
 <a href="#">IoT SiteWise</a>
 <a href="#">IoT Things Graph</a>
 <a href="#">Game Development</a>
 <a href="#">Amazon GameLift</a>
 <a href="#">Containers</a>
 <a href="#">Elastic Container Registry</a>
 <a href="#">Elastic Container Service</a>
 <a href="#">Elastic Kubernetes Service</a>
 <a href="#">Red Hat OpenShift Service ...</a>

# AWS Billing Console

Modelos de  
utilización



# Estilos de arquitectura habituales en Cloud

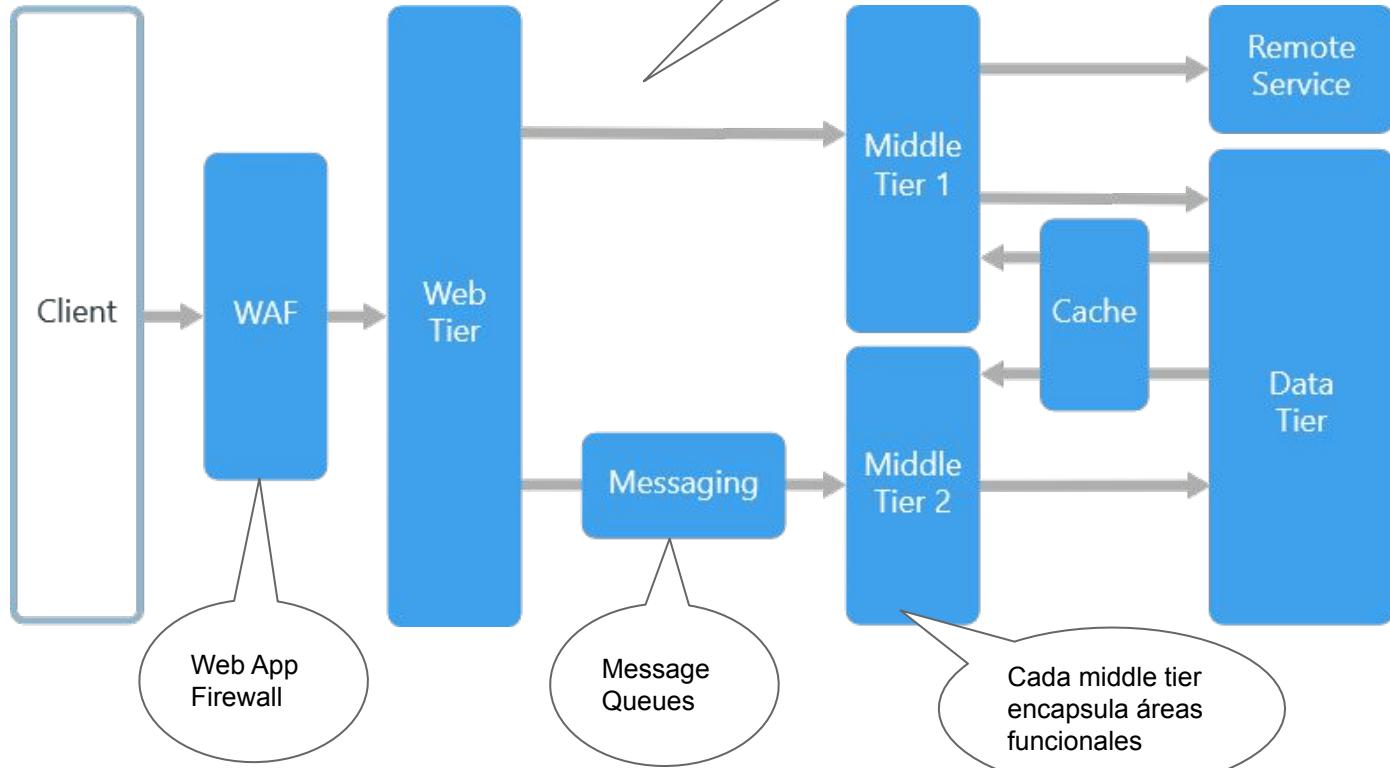
# N-tier

Las tiers son capas físicamente separadas

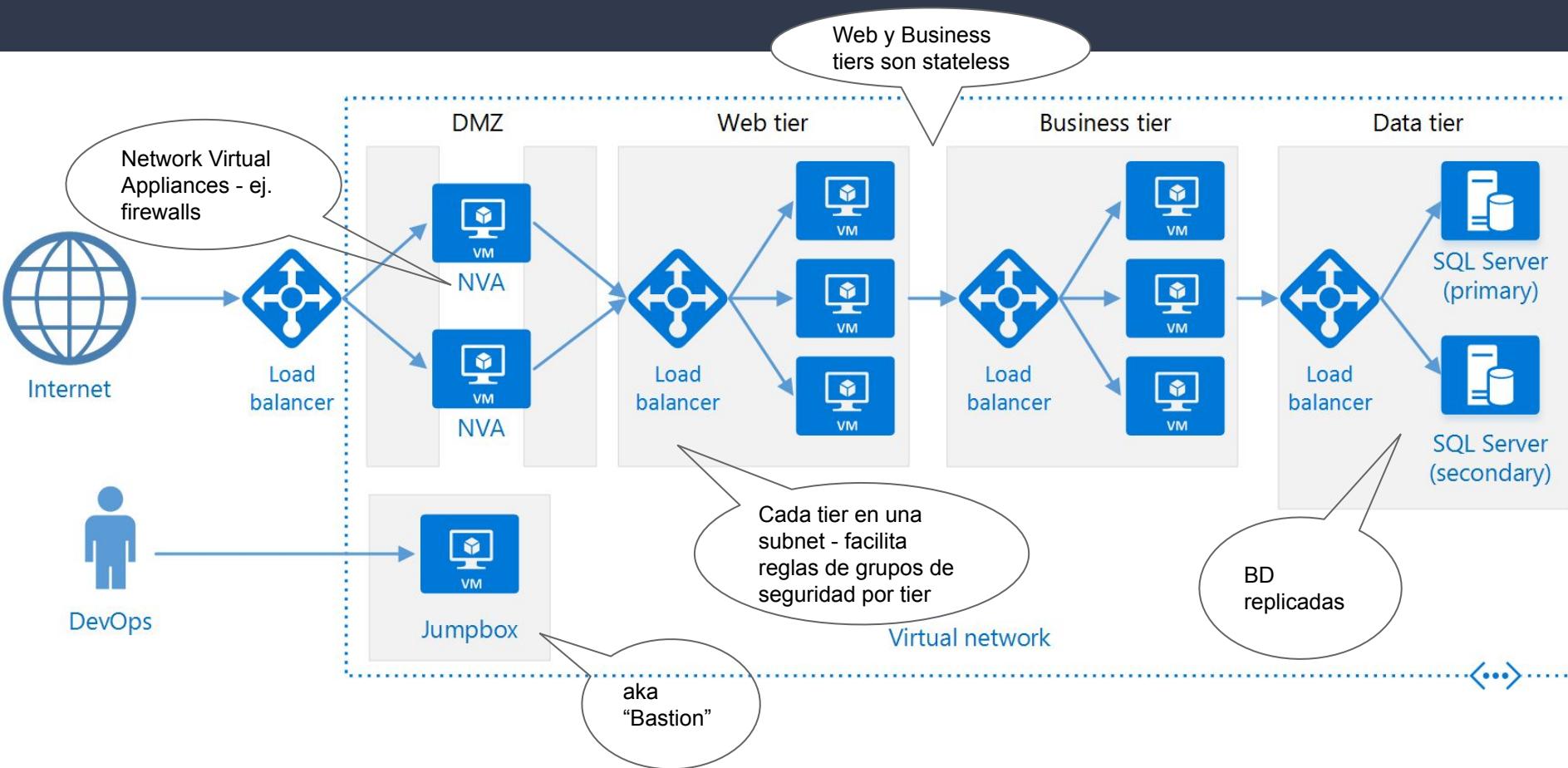
Las tiers pueden ser grupos de VMs manejadas por nosotros (IaaS) o servicios del cloud provider

## Layered System

Cada tier se puede escalar horizontalmente

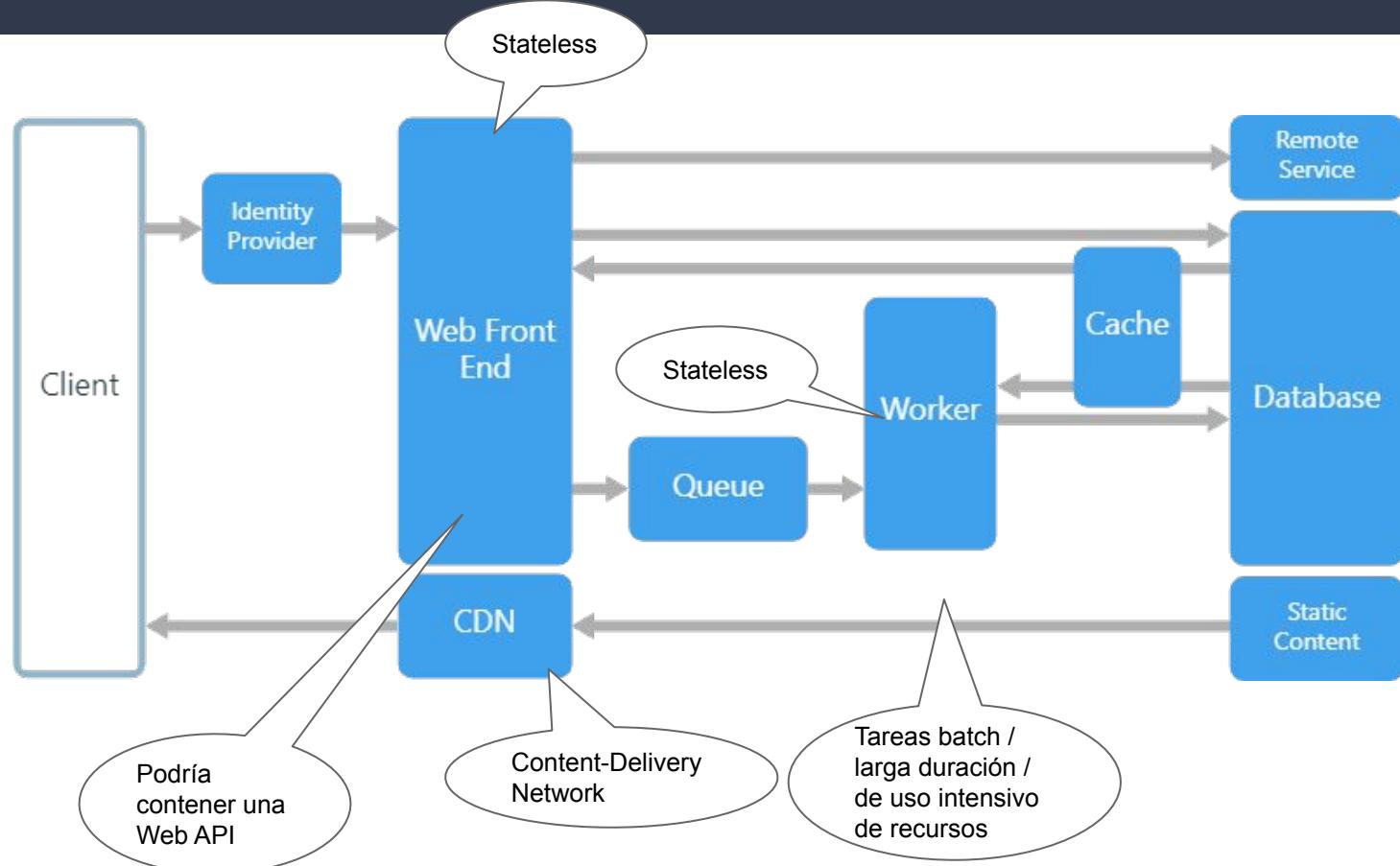


# N-tier usando IaaS en Azure

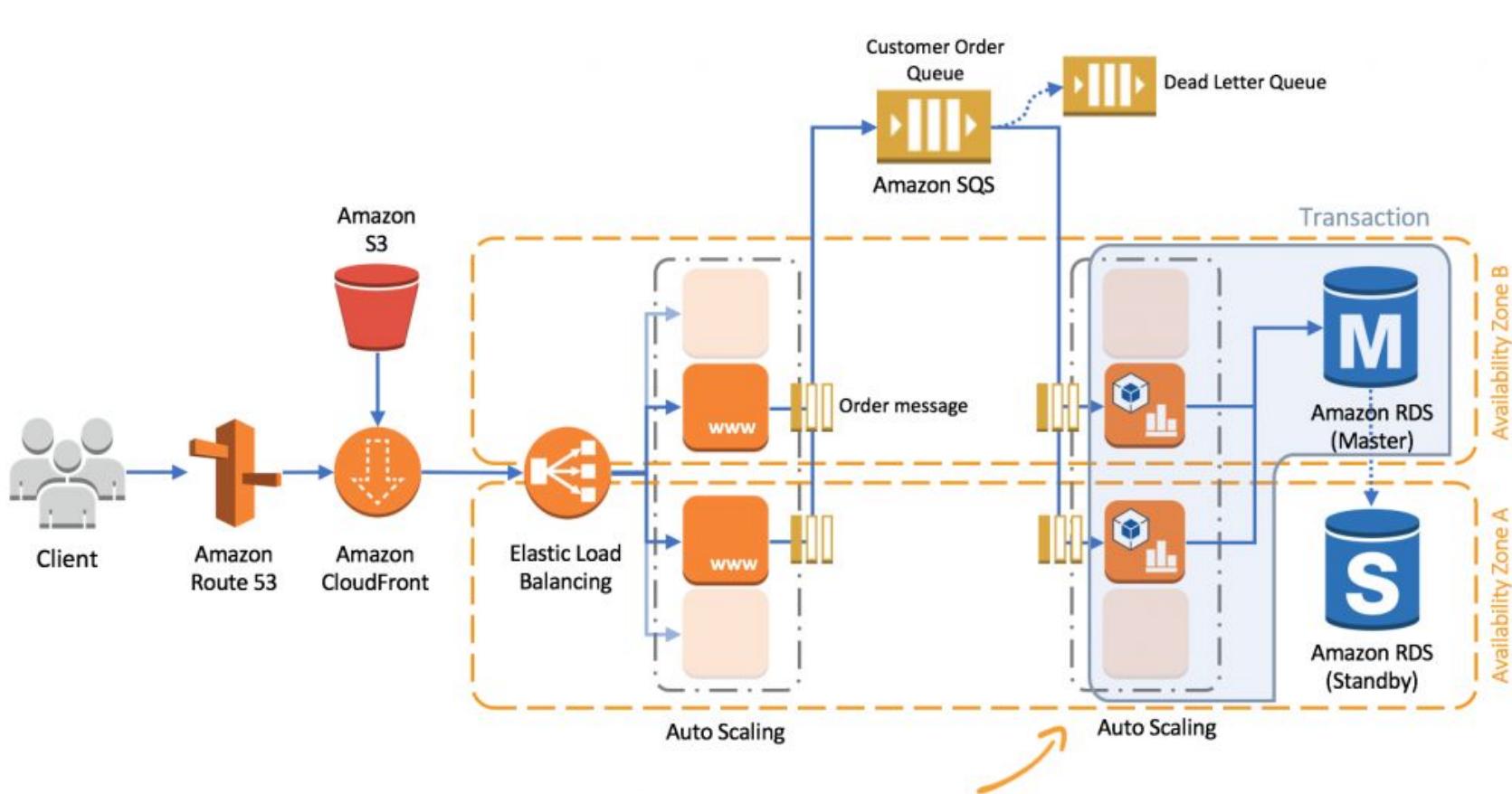


# Web-Queue-Worker

Front end y worker se podrían **escalar horizontalmente**



# AWS Web-Queue-Worker App



New processing nodes scaling independently, using

- `ApproximateNumberOfMessagesVisible`
- `ApproximateAgeOfOldestMessage`

# Event-driven



## Event Based Integration (EBI)

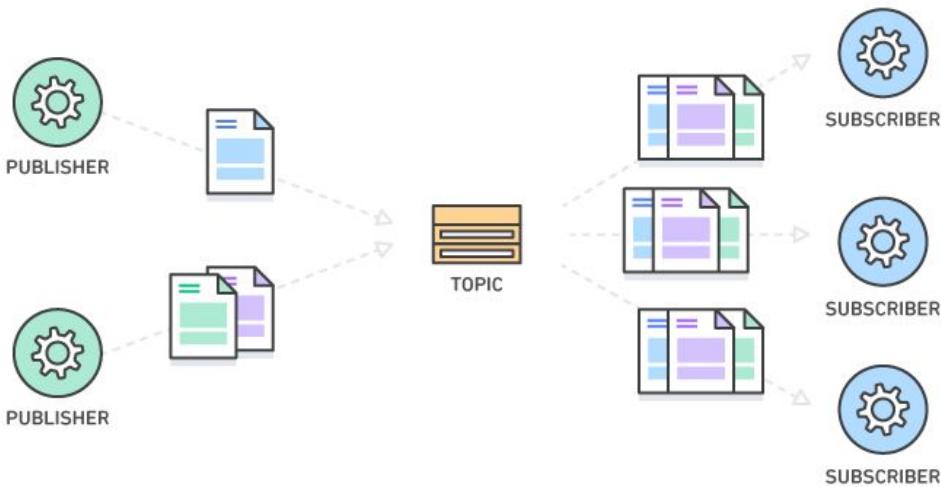
Producers desacoplados de consumers

Consumers desacoplados entre sí

Cada consumer ve todos los eventos

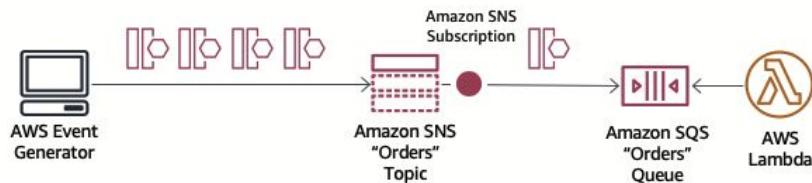
## Modelos

- **Pub/sub:** eventos no se pueden volver a reproducir (replay). Nuevos subscriptores no ven eventos anteriores.
- **Event Streaming:** se escriben en un log, ordenados, y persisten. Consumers leen de cualquier parte del stream. Hay replay.

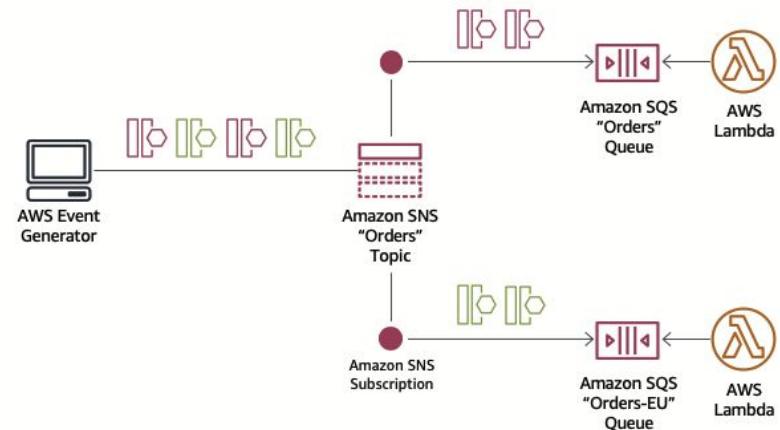


# Ejemplos AWS: Pub/Sub

Simple Pub/Sub

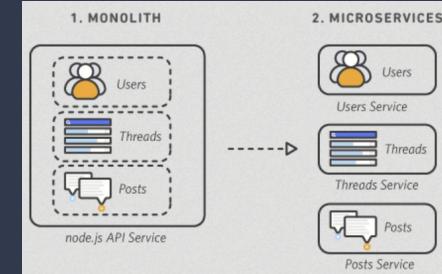


Pub/Sub con filtrado de mensajes



# Microservices

Servicios autónomos, pequeños, y desacoplados, cada cual implementa un área funcional de negocio



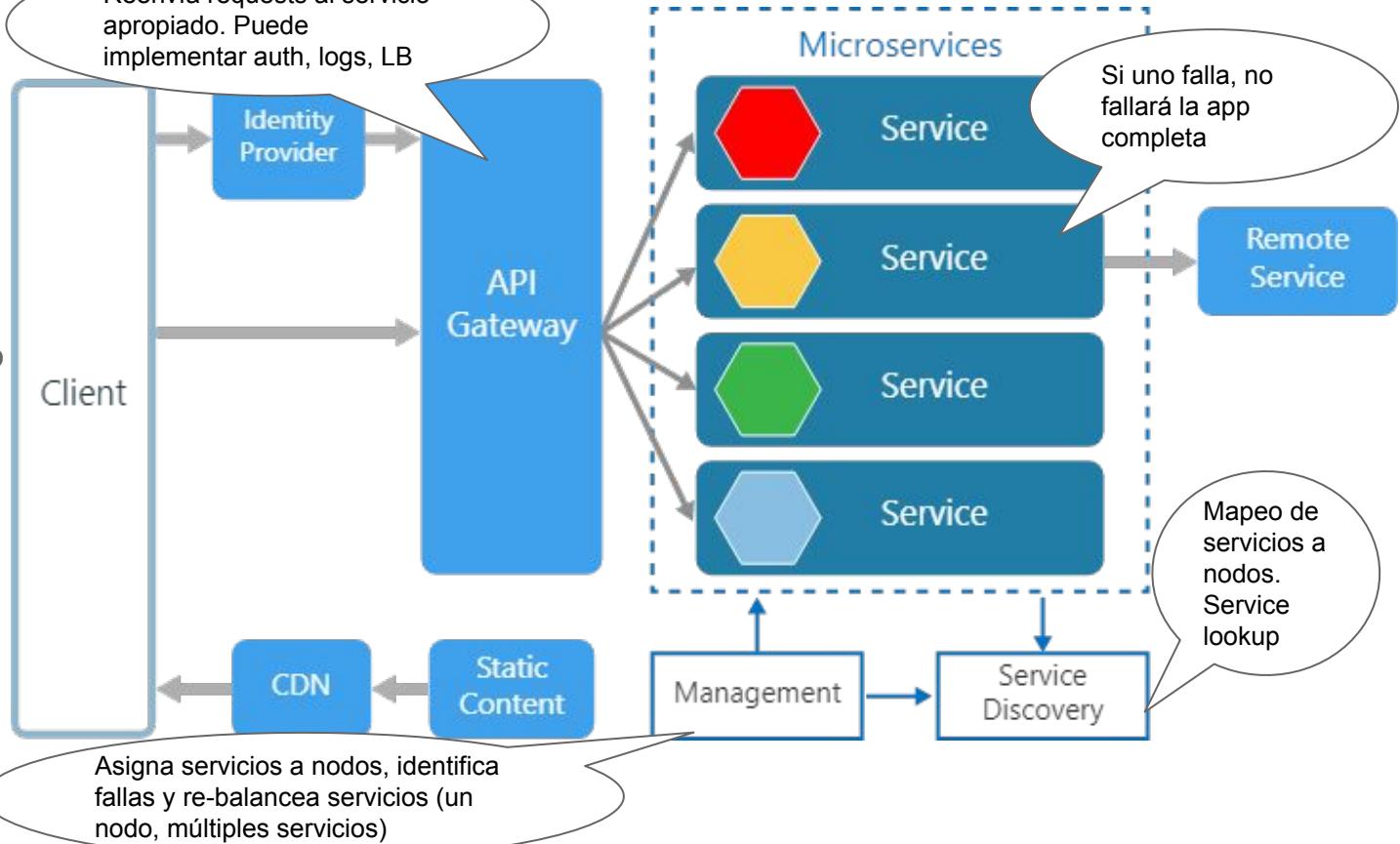
Cada microservice suele

- tener su propio codebase separado
- ser manejado por un pequeño grupo de dev
- ser deployado de forma independiente
- encargarse de su propio storage

Los microservices se comunican a través de APIs

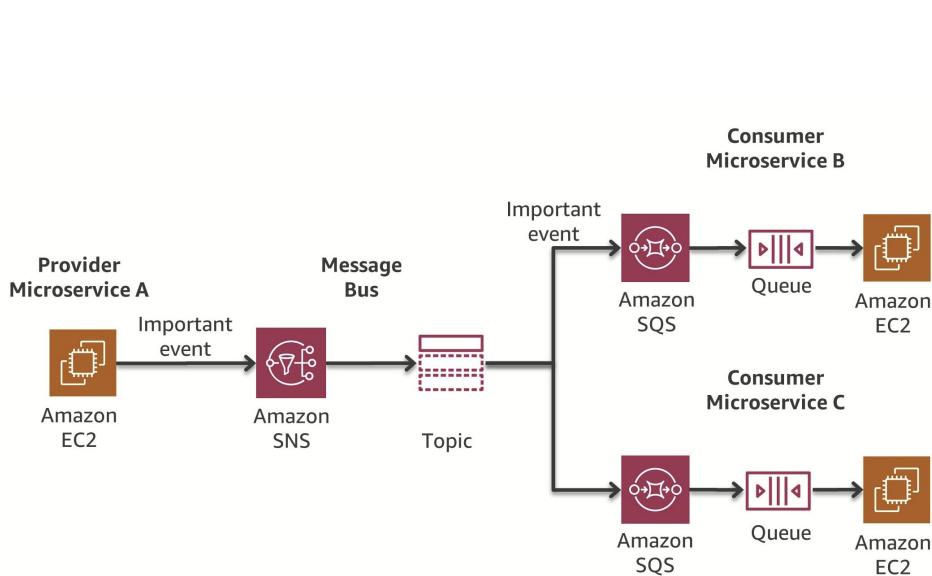
No necesariamente están implementados con la misma tecnología

Reenvía requests al servicio apropiado. Puede implementar auth, logs, LB



Asigna servicios a nodos, identifica fallas y re-balanza servicios (un nodo, múltiples servicios)

# Ejemplo AWS: comunicación async entre Microservices



## Comunicación entre Microservices

1. Sincrónica (llamados directos a APIs)
2. Asincrónica
  - Queues
  - EBI (aka “Message Bus pattern”)

- Servicios desacoplados
- No es necesario un Service Discovery (aka Service Directory)

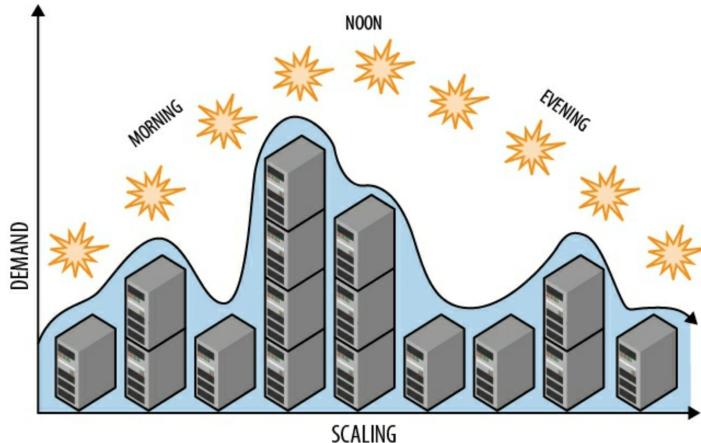
# Cloud usage Patterns

¿Cómo implementar “Match Strategy” de manera de optimizar el costo lo máximo posible y cumplir con la demanda?

# Auto-Scaling

Escalar manualmente usando la consola del cloud provider es posible y útil, pero:

- Conlleva esfuerzo manual rutinario
- Requiere administrador disponible
- El costo podría optimizarse más con una escalabilidad automática



*Disponer un proceso de asignación dinámica de recursos (nodos, storage, queues, etc.) para implementar “match strategy”*

- Por lo general, hablamos de **Horizontal Scaling**
- Reglas para eventos conocidos
  - A las 7PM, 2 web servers.
  - A las 7AM, 10 web servers.
  - A las 7PM de los días viernes, 1 web server.
- Reglas para reaccionar a eventos del ambiente

*“Si el % promedio de CPU utilizado durante la última hora es mayor a 50, incrementar en 1 la cantidad de instancias web server del pool”*

*“Si la cantidad promedio de mensajes en la queue en los últimos 20 minutos es menor a 5, decrementar en 1 la cantidad de workers”*

*“Si el tiempo promedio para confirmar la orden de un cliente en la última hora es mayor a 10 minutos, incrementar en 1 la cantidad de workers”*

# Auto-Scaling

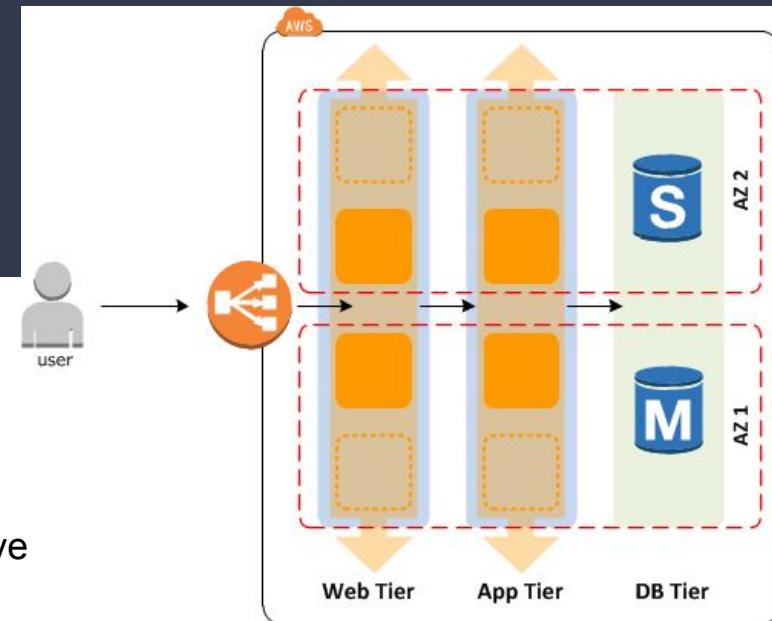
Consideraciones para la implementación

**El sistema debe ser escalable horizontalmente**

Evitar instance affinity, implementar Stateless

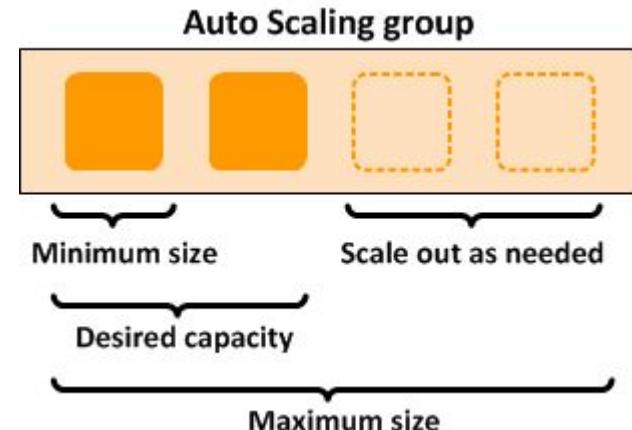
## Qué necesitamos

- Scale out/in APIs del cloud provider
- Sistemas de monitoreo que capturen las métricas clave
- Lógica que evalúe las métricas respecto a las reglas predefinidas, y decidan si escalar
- Testing y tuning de la estrategia de auto-scaling



## Buenas prácticas

- Utilizar las funcionalidades de auto-scaling del cloud provider; solución custom requiere esfuerzo significativo
- Escalar cada componente / subsistema de forma independiente
- Poner **límites inferiores y superiores** en las reglas
- Long-running tasks pueden causar problemas
  - Refactorizar hacia Pipes and Filters
  - Usar checkpoints en almacenamiento externo



# AWS Auto Scaling Groups

## Reglas de Auto Scaling

https://console.aws.amazon.com/ec2/autoscaling/home?region=us-east-1#AutoScalingGroups:id=DEV-MCS-Ci-API-Public-develop-CiApiASG-16UHGXNHAD3UX;view=policies

Services ▾ Resource Groups ▾ Guillermo\_Rugilo@... co... N. Virginia ▾ Support ▾

EC2 Dashboard Events Tags Reports Limits

INSTANCES Instances Launch Templates Spot Requests Reserved Instances Dedicated Hosts Scheduled Instances

IMAGES AMIs Bundle Tasks

ELASTIC BLOCK STORE

NETWORK & SECURITY Security Groups Elastic IPs Placement Groups Key Pairs Network Interfaces

LOAD BALANCING Load Balancers Target Groups

AUTO SCALING Launch Configurations **Auto Scaling Groups**

SYSTEMS MANAGER SERVICES

Create Auto Scaling group Actions ▾

Filter: Filter Auto Scaling groups... K < 1 to 50 of 90 Auto Scaling Groups > ▾

Name	Launch Configuration /	Instances	Desired	Min	Max	Availability Zones	Default Cooldown
QA-MCS-AutoScalingGroup-12...	QA-MCS-AutoScalingGroup...	1	1	1	5	us-east-1a, us-east-1c, us-e...	300
DEV-MCS-Ci-API-Content-devel...	DEV-MCS-Ci-API-Cont...	1	1	1	1	us-east-1a, us-east-1c, us-e...	300
DEV-MCS-APIS-3-0-McsApiASG...	DEV-MCS-APIS-3-0-Mc...	1	1	1	1	us-east-1a, us-east-1c, us-e...	300
<b>Public-APIs-ASG</b>	DEV-MCS-Ci-API-Publi...	1	1	1	1	us-east-1a, us-east-1c, us-e...	300
QA-MCS-Jobs2-20180501-Work...	QA-MCS-Jobs2-20180...	1	1	1	5	us-east-1d	300

Auto Scaling Group: Public-APIs-ASG

Details Activity History **Scaling Policies** Instances Monitoring Notifications Tags Scheduled Actions Lifecycle Hooks

Add policy

Public-APIs-WebServer-ScaleDown

Actions ▾

**Policy type:** Simple scaling  
**Execute policy when:** Public-APIs-Alarm-CPU-Low  
breaches the alarm threshold: CPUUtilization <= 40 for 2 consecutive periods of 60 seconds for the metric dimensions AutoScalingGroupName = Public-APIs-ASG

**Take the action:** Remove 1 instances  
**And then wait:** 450 seconds before allowing another scaling activity

Public-APIs-WebServer-ScaleUp

Actions ▾

**Policy type:** Simple scaling  
**Execute policy when:** Public-APIs-Alarm-CPU-High  
breaches the alarm threshold: CPUUtilization > 80 for 5 consecutive periods of 60 seconds for the metric dimensions AutoScalingGroupName = Public-APIs-ASG

**Take the action:** Add 2 instances  
**And then wait:** 600 seconds before allowing another scaling activity

# AWS CloudWatch

https://console.aws.amazon.com/cloudwatch/home?region=us-east-1#alarm:alarmFilter=ANY;name=DEV-MCS-Ci-API-Public-develop-AlarmCPUHigh-IFOF2VM73OLL

aws Services Resource Groups Actions

CloudWatch Dashboards Alarms

ALARMS 34 INSUFFICIENT 46 OK 121

Billing Events Rules Event Buses Logs Metrics Favorites + Add a dashboard

Create Alarm Add to Dashboard Actions

Filter: All alarms

State	Name	Threshold	Config Status
ALARM	DEV-MCS-Ci-API-Public-develop-AlarmCPULow-42TUPFTLXDX2	CPUUtilization <= 40 for 2 datapoints within 2 minutes	
INSUFFICIENT_DATA	DEV-MCS-Ci-API-Public-develop-AlarmELBLatencyExt-1H998XZO06X7C	TargetResponseTime >= 2 for 2 datapoints within 2 minutes	No notifications
OK	DEV-MCS-Ci-API-Public-develop-AlarmELBUnHealthyHostCountExt-WJ7GQQV9FLHG	UnHealthyHostCount >= 1 for 1 datapoints within 1 minute	
OK	DEV-MCS-Ci-API-Public-develop-AlarmELBUnHealthyHostCountInt-MFB7CBD7DMY9	UnHealthyHostCount >= 1 for 1 datapoints within 1 minute	
OK	Public-APIS-CPU-High	CPUUtilization > 80 for 5 datapoints within 5 minutes	No notifications

1 Alarm selected

**Alarm: Public-APIS-CPU-High**

Details History

State Details: State changed to OK at 2018/10/25. Reason: Threshold Crossed: 1 datapoint [15.3571428571429 (25/10/18 10:32:00)] was not greater than the threshold (80.0).

Description: Send Notification if CPU Utilization is above threshold

Threshold: CPUUtilization > 80 for 5 datapoints within 5 minutes

Actions: In ALARM:

- Send message to topic "OpsNotification-DEV" [REDACTED]
- For group DEV-MCS-Ci-API-Public-develop-CiApiASG-16UHGZNHAD3UX use policy DEV-MCS-Ci-API-Public-develop-WebServerScaleUpPolicy-E19LFQ4QCZLM (Add 2 instances)
- Send message to topic "AppSupportTeam-Notification" Send message to topic: "AppSupportTeam-Notification" (Warning: this alarm is not configured to notify. Please modify this alarm and add an email address.)

In INSUFFICIENT DATA: [REDACTED]

Namespace: AWS/EC2

Metric Name: CPUUtilization

Dimensions: AutoScalingGroupName = DEV-MCS-Ci-API-Public-develop-CiApiASG-16UHGZNHAD3UX

Statistic: Average

Period: 1 minute

Treat missing data missing as:

Percentiles with evaluate low samples:

Public-APIS-CPU-High

Percent

97.5

48.8

0

CPUUtilization > 80 for 5 datapoints within 5 minutes

11:00 12:00 13:00

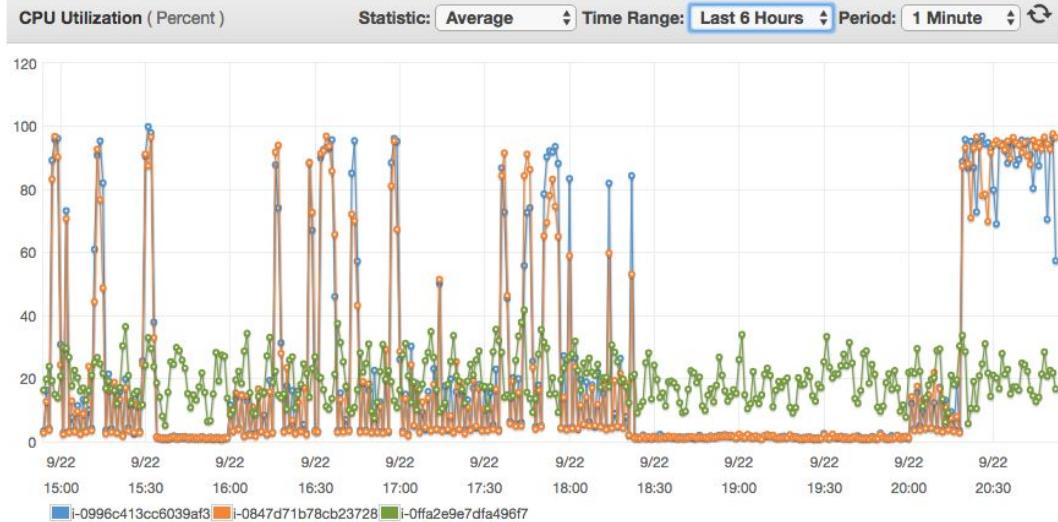
CPUUtilization

View in metrics

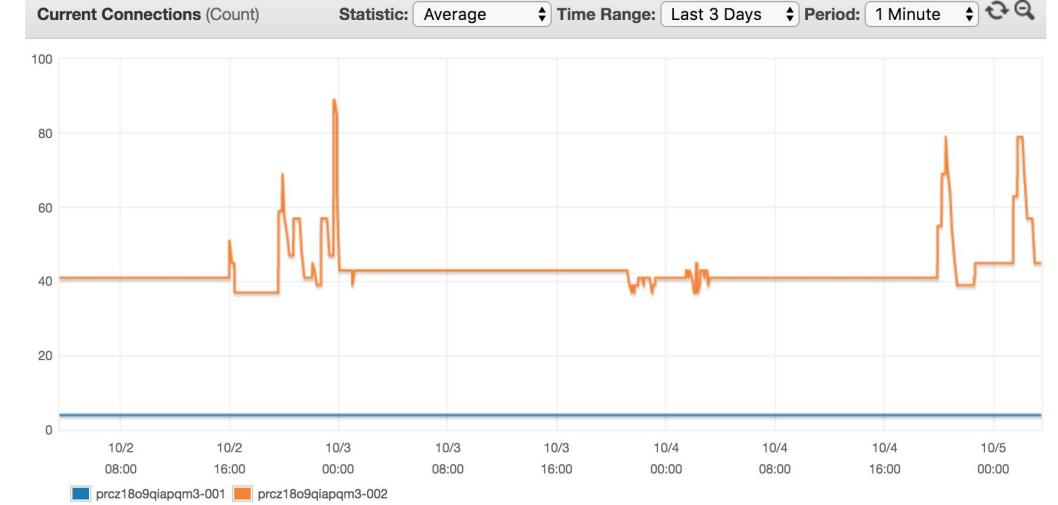
# AWS CloudWatch

## Análisis de métricas

### CloudWatch Monitoring Details



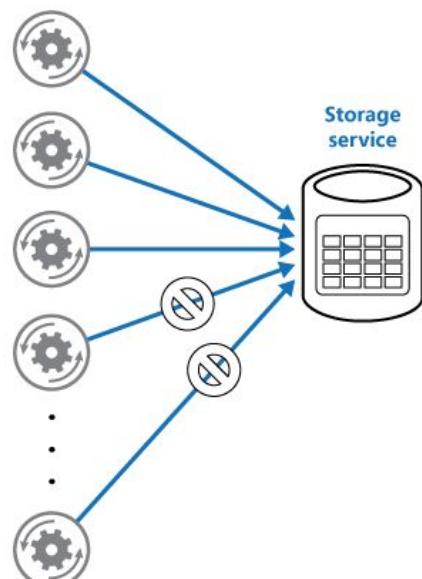
### CloudWatch Monitoring Details



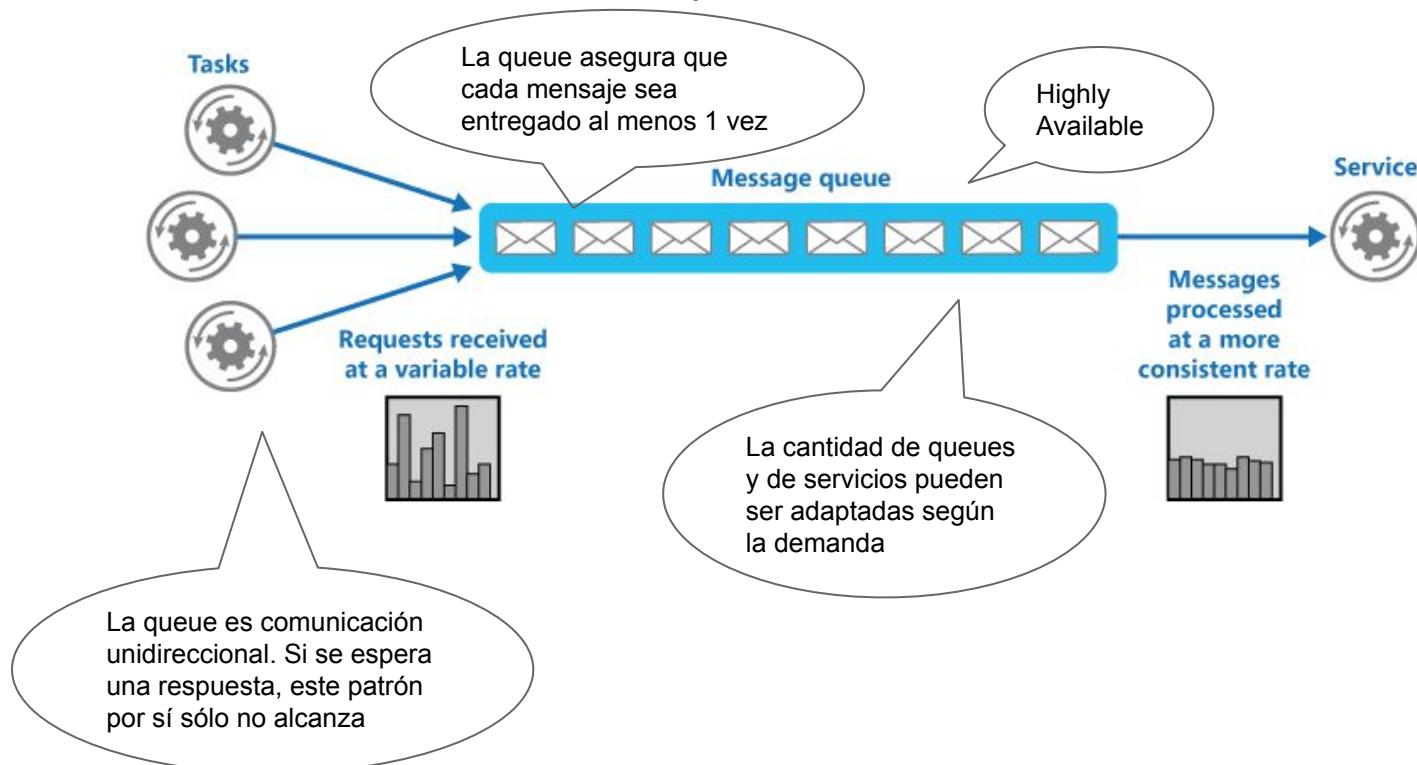
## ¿Cómo evitar la sobrecarga de un servicio sujeto a cargas grandes e intermitentes?

# Queue-Based Load Leveling

Picos en la demanda, muchos requests concurrentes



*Introducir una cola de mensajes (usualmente FIFO) entre los clientes del servicio y el servicio, actuando como buffer*



# Message Queues

## Consideraciones

El orden de los mensajes no suele estar garantizado

En modelo transaccional

- Algoritmo del consumer
- 1) Dequeue (get message)
- 2) Process
- 3) Delete message from queue

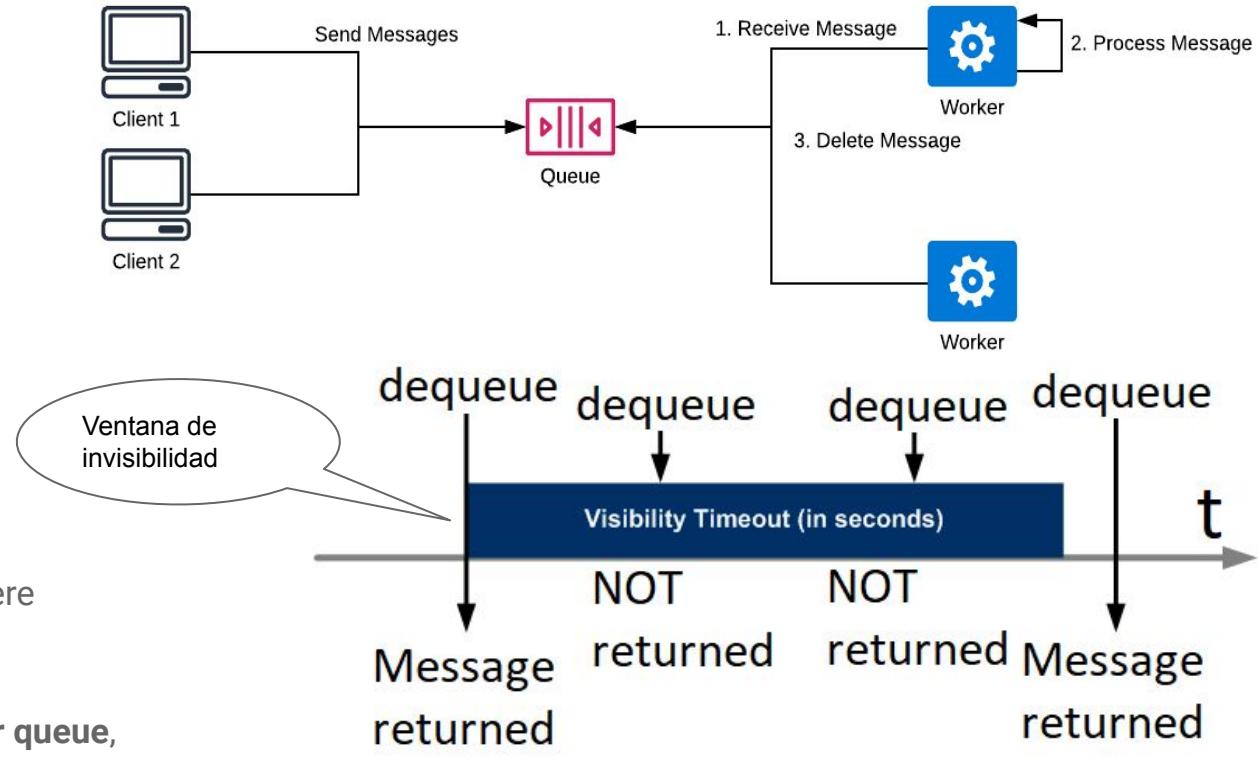
- Se usa **ventana de invisibilidad**

Un mensaje podría ser entregado **más de 1 vez** (ej. si un worker falla)

→ implementar **idempotencia**

**Poison messages:** mal formado, o requiere accesos denegados.

→ Guardar el mensaje en una **dead letter queue**, para posterior análisis



¿Cómo lograr que múltiples componentes “consumers” (workers) procesen mensajes de múltiples “producers” (apps) de forma concurrente?

# Competing Consumers

Gran número de requests

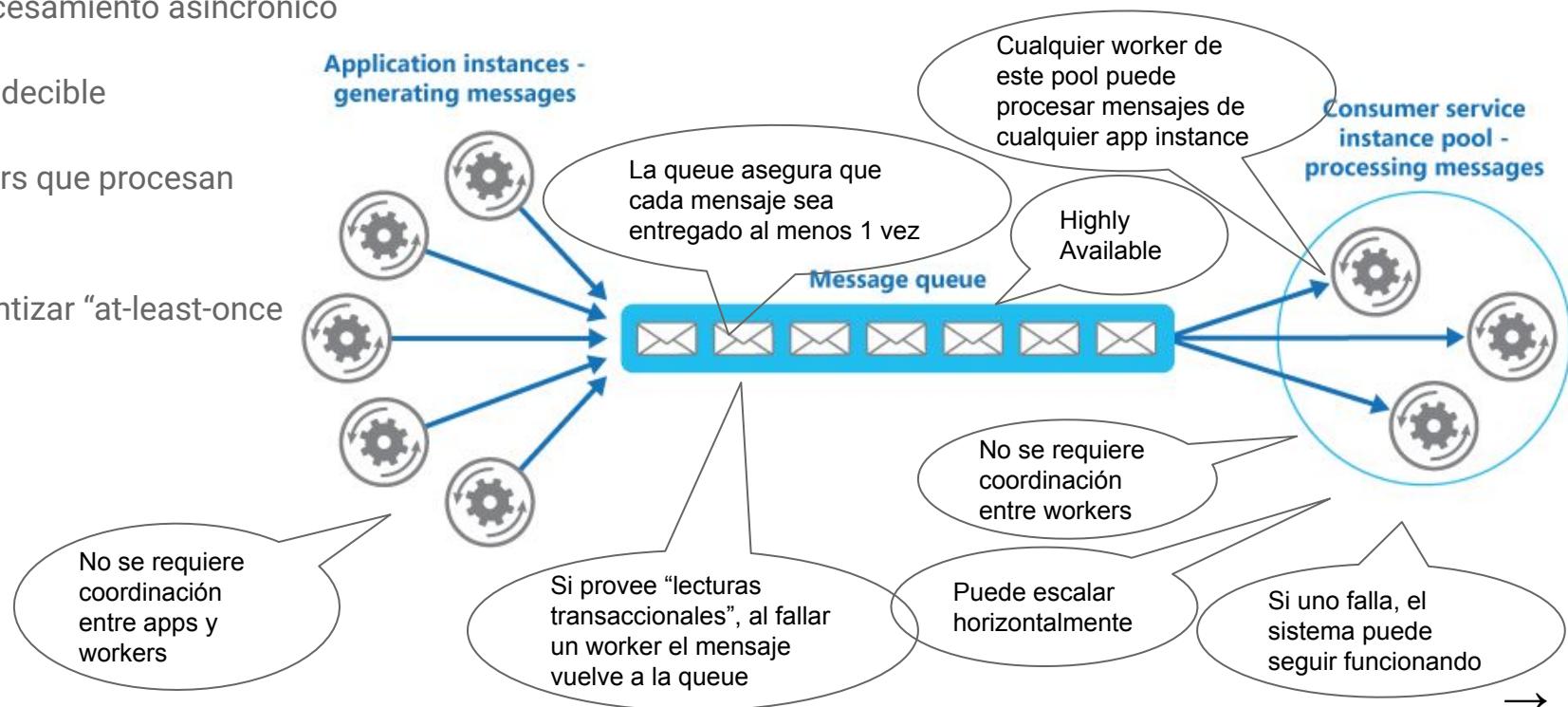
Se requiere procesamiento asincrónico

Demanda impredecible

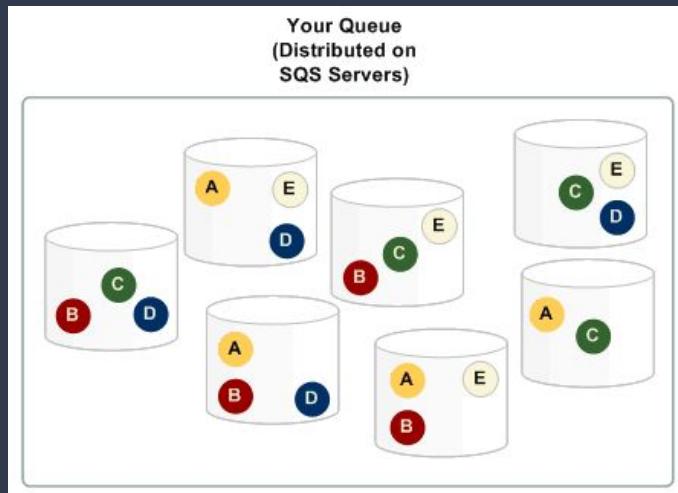
Múltiples workers que procesan requests

Queremos garantizar “at-least-once processing”

*Introducir una cola de mensajes entre las instancias de la aplicación y las instancias de los consumers*



# AWS SQS



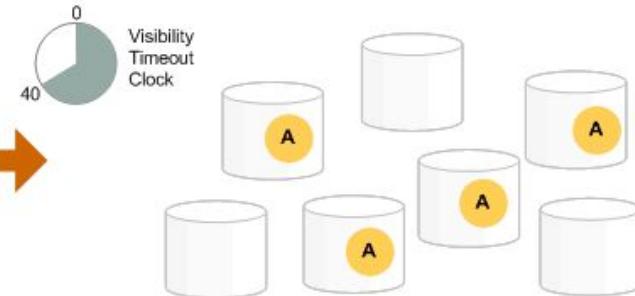
DB servers distribuidos con replicación

```
POST /MyQueue HTTP/1.1  
Host: sqs.us-east-2.amazonaws.com  
Content-Type: application/x-www-form-urlencoded
```

```
Action=SendMessage  
&MessageBody=Your+Message+Text  
&Expires=2020-10-15T12%3A00%3A00Z  
&Version=2012-11-05
```

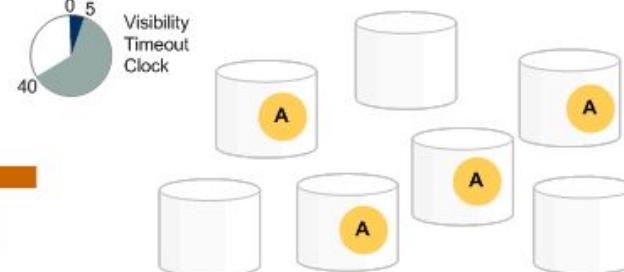
1

Component 1 sends  
Message A to the queue



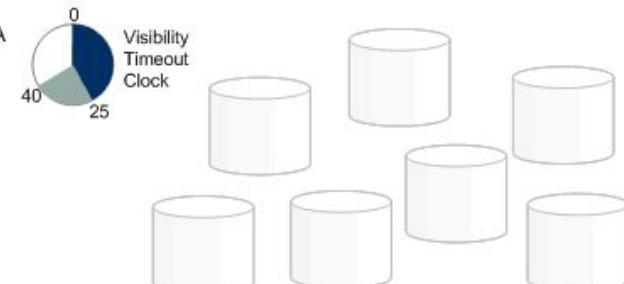
2

Component 2 retrieves Message A  
from the queue and the visibility  
timeout period starts



3

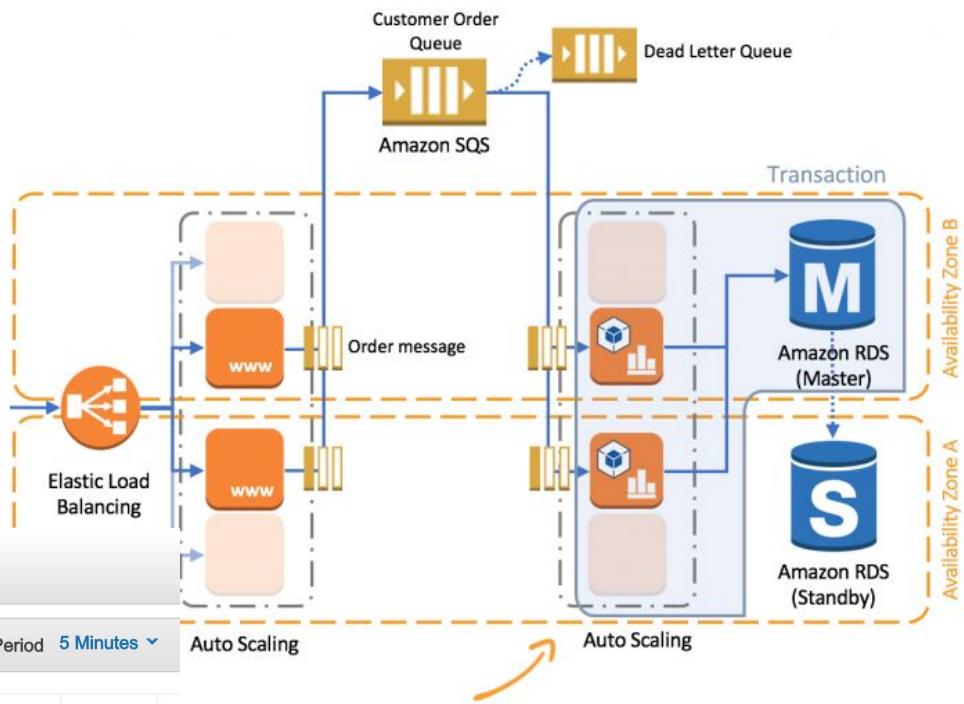
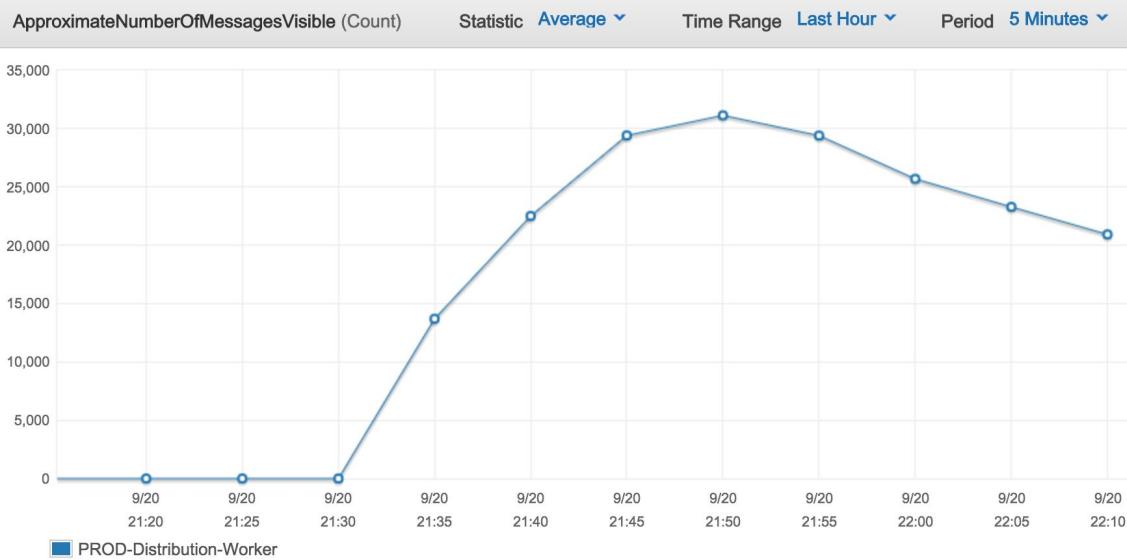
Component 2 processes Message A  
and then deletes it from the queue  
during the visibility timeout period



# AWS SQS

Auto-Scaling basado en métricas de las queues

## CloudWatch Monitoring Details



New processing nodes scaling independently, using

- ApproximateNumberOfMessagesVisible
- ApproximateAgeOfOldestMessage

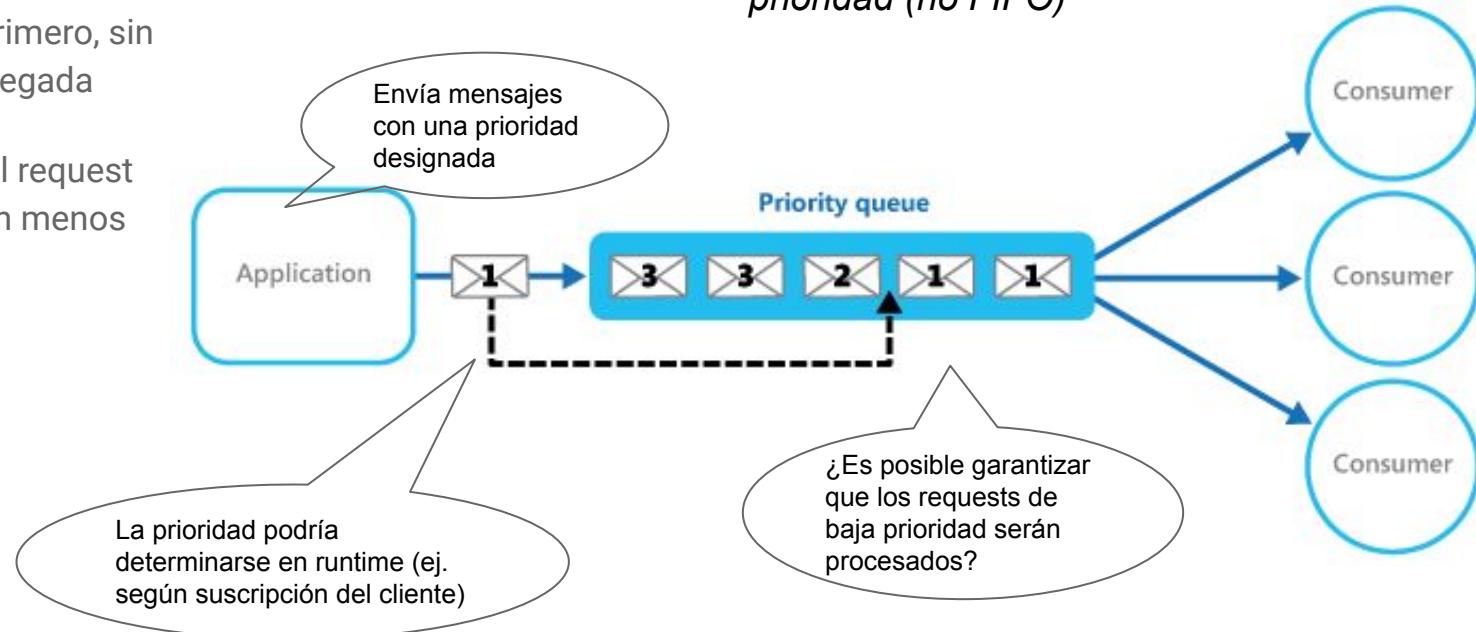
¿Cómo priorizar requests para respetar distintos SLAs para clientes particulares?

Los requests de “alta prioridad” deberían ejecutarse primero, sin importar el orden de llegada

Ej.: “Alta prioridad” = el request debe ser procesado en menos de 5 segundos

# Priority Queue

*Introducir una cola de prioridad que ordene los mensajes según prioridad (no FIFO)*

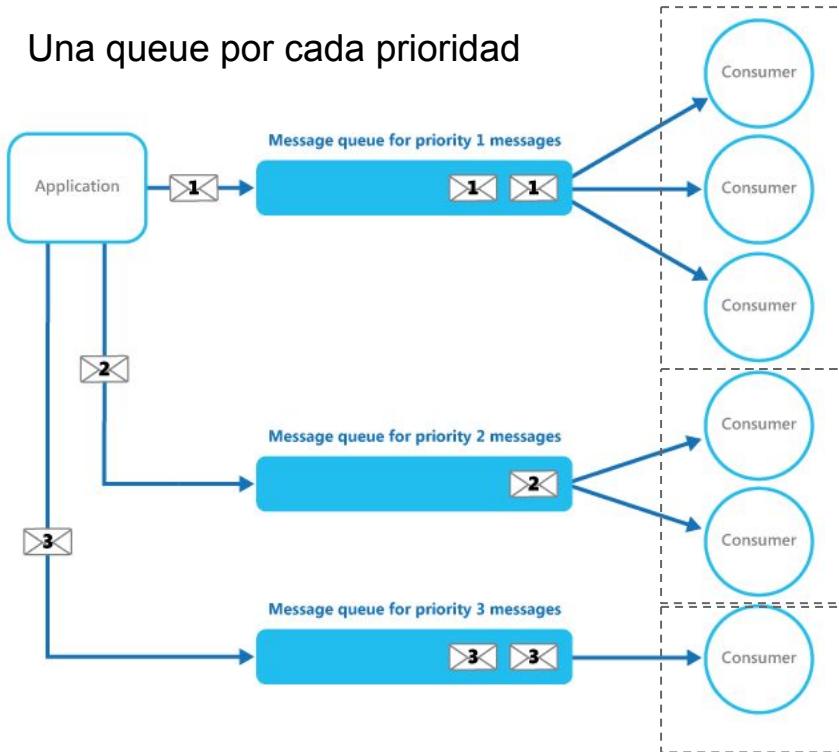


Ah sí, muy lindo, ¿y qué hago si no tengo una “Priority Queue” mágica?

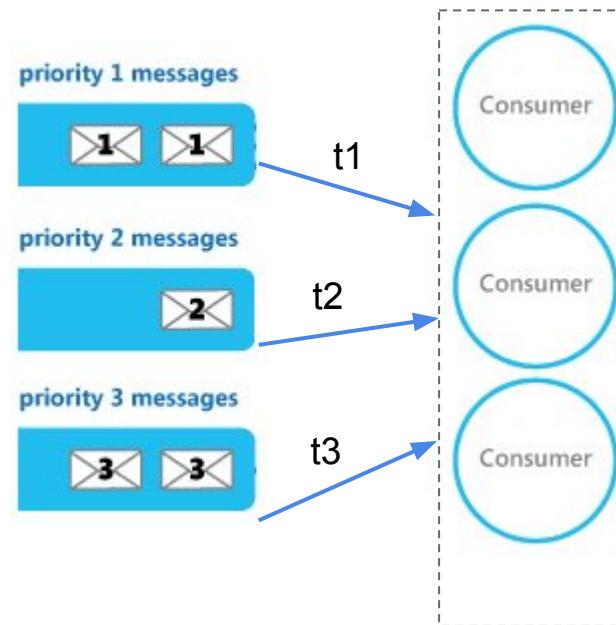
## Variantes de implementación

# Priority Queue

Una queue por cada prioridad

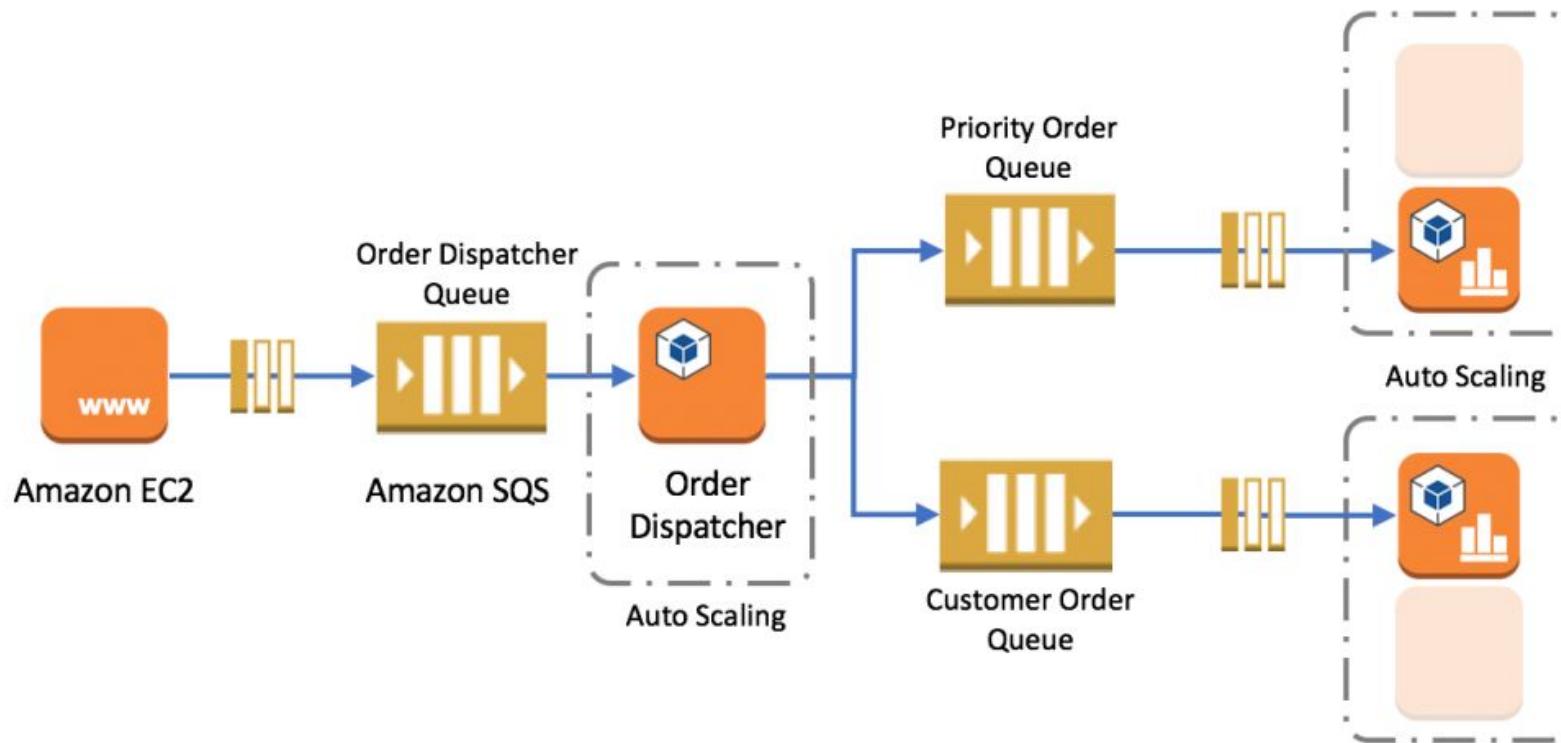


Un único pool de consumers. Sólo se procesan los P2 si no hay mensajes P1



Con 1 sólo pool de consumers no hay garantía de procesamiento de P2 y P3

# Priority Queue en AWS

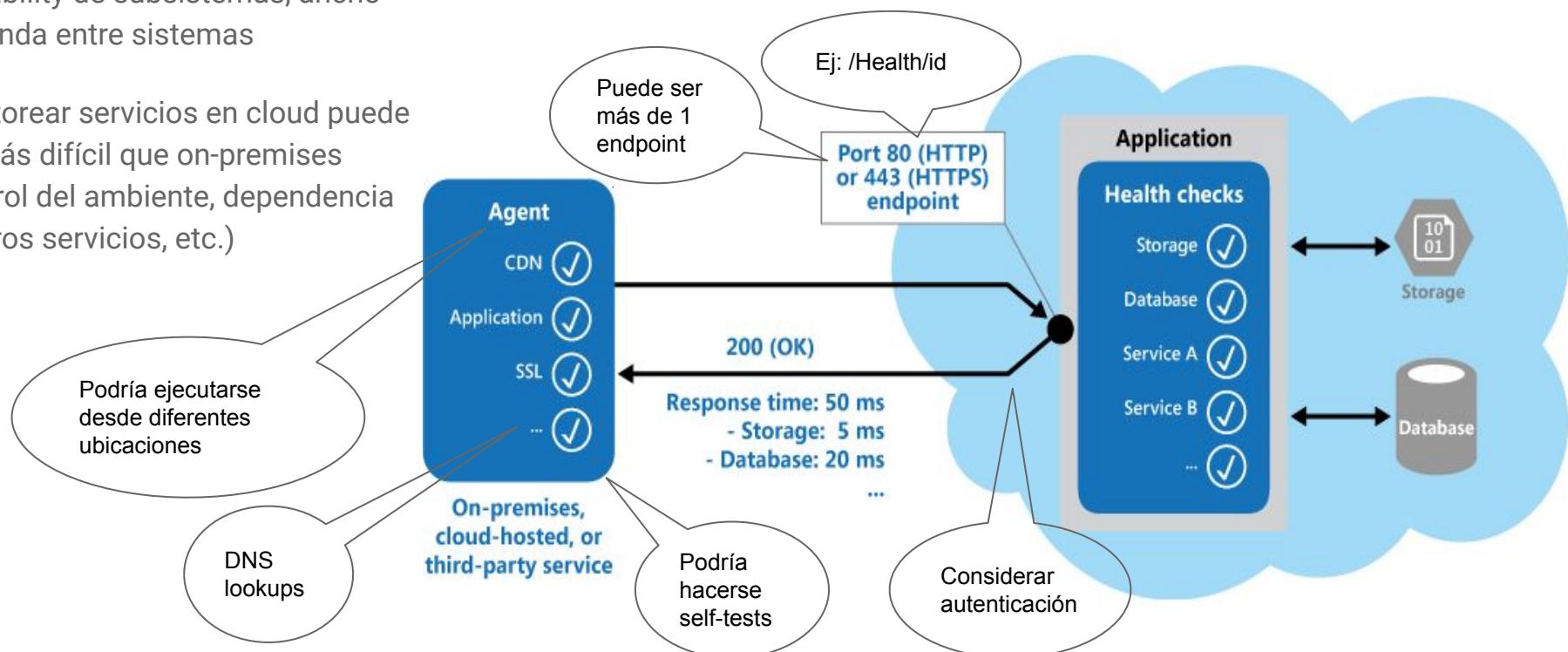


¿Cómo verificar si un servicio está cumpliendo con el nivel requerido de availability y performance?

Factores que afectan la app:

Latencia de red, performance / availability de subsistemas, ancho de banda entre sistemas

Monitorear servicios en cloud puede ser más difícil que on-premises (control del ambiente, dependencia de otros servicios, etc.)



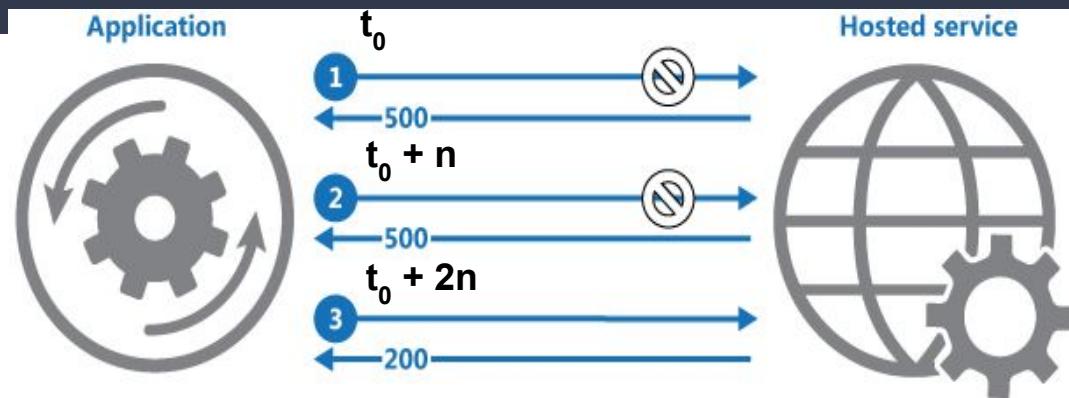
¿Cómo lograr que las aplicaciones manejen fallas temporales de los servicios que consumen?

En ambientes cloud, fallas temporales son comunes: conectividad, servicios no disponibles, timeouts por servicio ocupado

Esas fallas suelen corregirse “solas” (ej. throttling)

### ***Implementar lógica de reintentos (retry) que consideren un lapso de tiempo (delay) entre cada reinicio***

- Puede haber muchas instancias de la app reintentando
  - Se debe tratar de distribuir los reintentos lo más uniformemente posible



- La política de retry (cantidad, delays) debe ser escogida según la naturaleza de la interacción client-server
- Políticas agresivas pueden degradar más el servicio
- Considerar si la operación es **idempotente**: la falla puede darse al enviar la respuesta
- Mantener logs de los intentos fallidos

¿Cómo evitar que las aplicaciones invoquen servicios que están caídos, y que luego detecten cuando el problema ya está resuelto?

## Availability

# Circuit Breaker

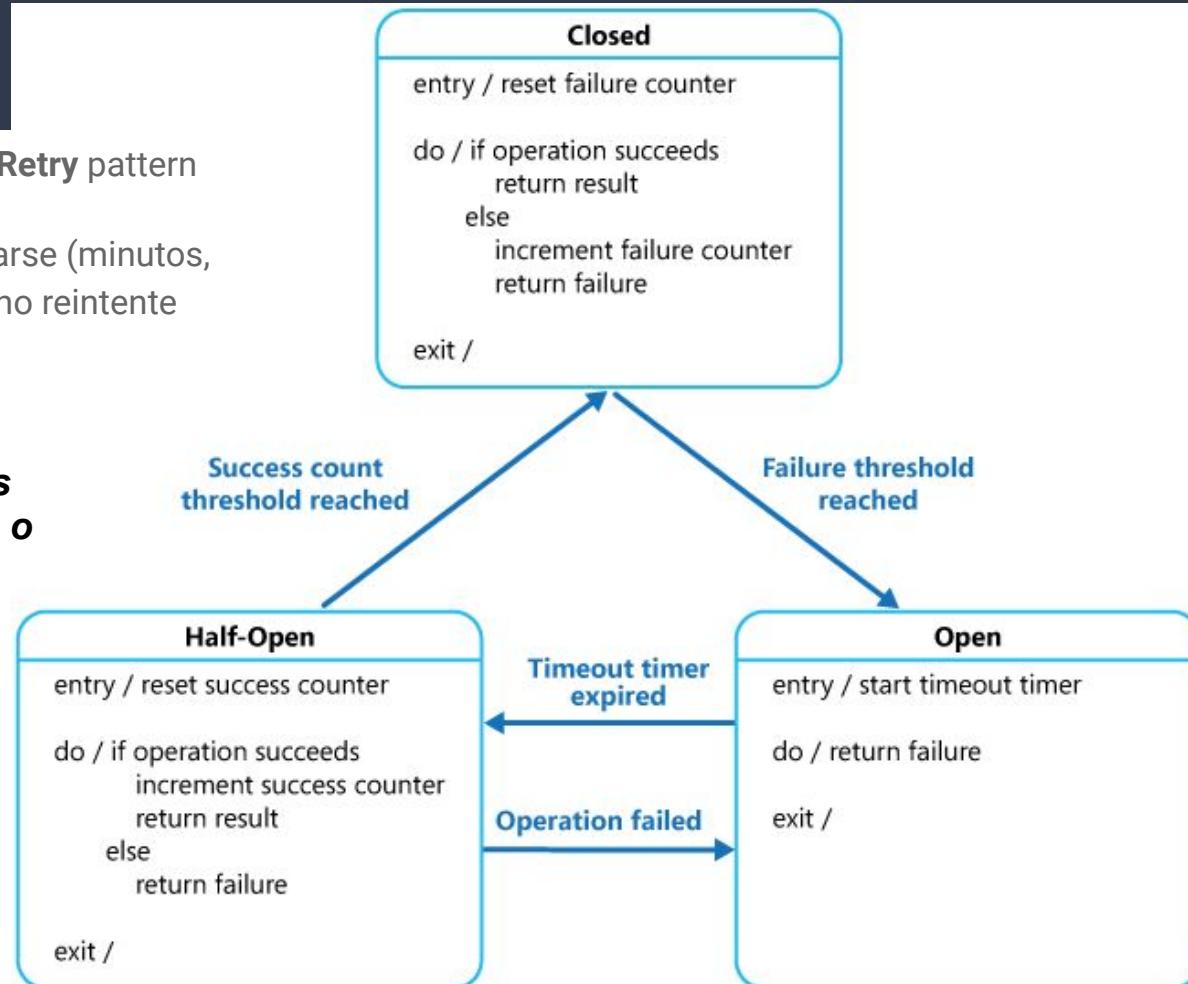
Las fallas temporales pueden manejarse con **Retry pattern**

Si la falla toma tiempo considerable en arreglarse (minutos, horas), sería ideal que la app acepte la falla y no reintente

**Implementar un proxy que registre la cantidad de fallas recientes y use esos datos para decidir si invocar el servicio o si fallar inmediatamente**

Estados del Circuit Breaker

- **Closed:** pasan todos los requests
- **Open:** devuelve excepción inmediatamente
- **Half-Open:** un número limitado de requests pasan, los otros fallan



# Throttling

¿Cómo controlar el uso de recursos para cumplir con SLAs sin utilizar únicamente autoscaling?

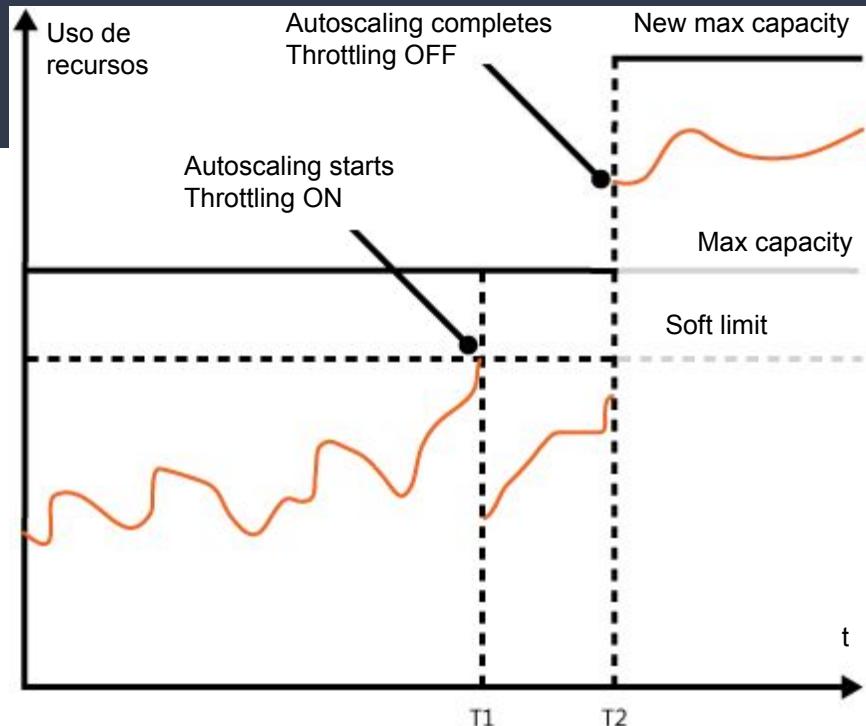
Un único cliente / app / servicio podría consumir todos los recursos con un burst de requests

Gran cantidad súbita de requests

Autoscaling no es inmediato, y aumenta los costos (respecto a no usarlo)

***Imponer un límite al uso de recursos, que al alcanzarse, el sistema comenzará a regular los requests de los clientes***

- Rechazar requests de clients que ya usaron la API más de N veces por segundo durante un lapso de t
- Degradar o deshabilitar funcionalidades no esenciales
- Priorizar requests según SLAs



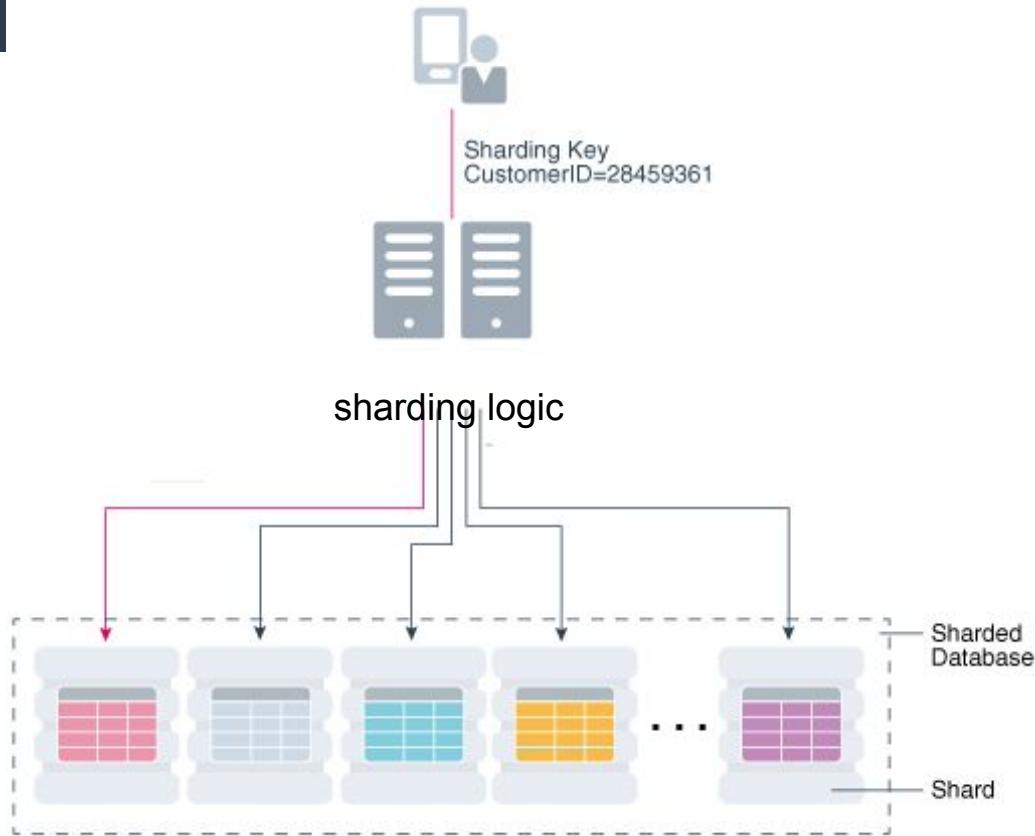
- La implementación puede tener gran impacto - en general es difícil implementar tarde en el desarrollo
- Devolver códigos de error específicos para throttling
- Debe poder ser ejecutado y detenido rápidamente

¿Cómo superar las limitaciones de espacio, recursos, ancho de banda y geográficas de un almacenamiento de datos en 1 server?

# Sharding

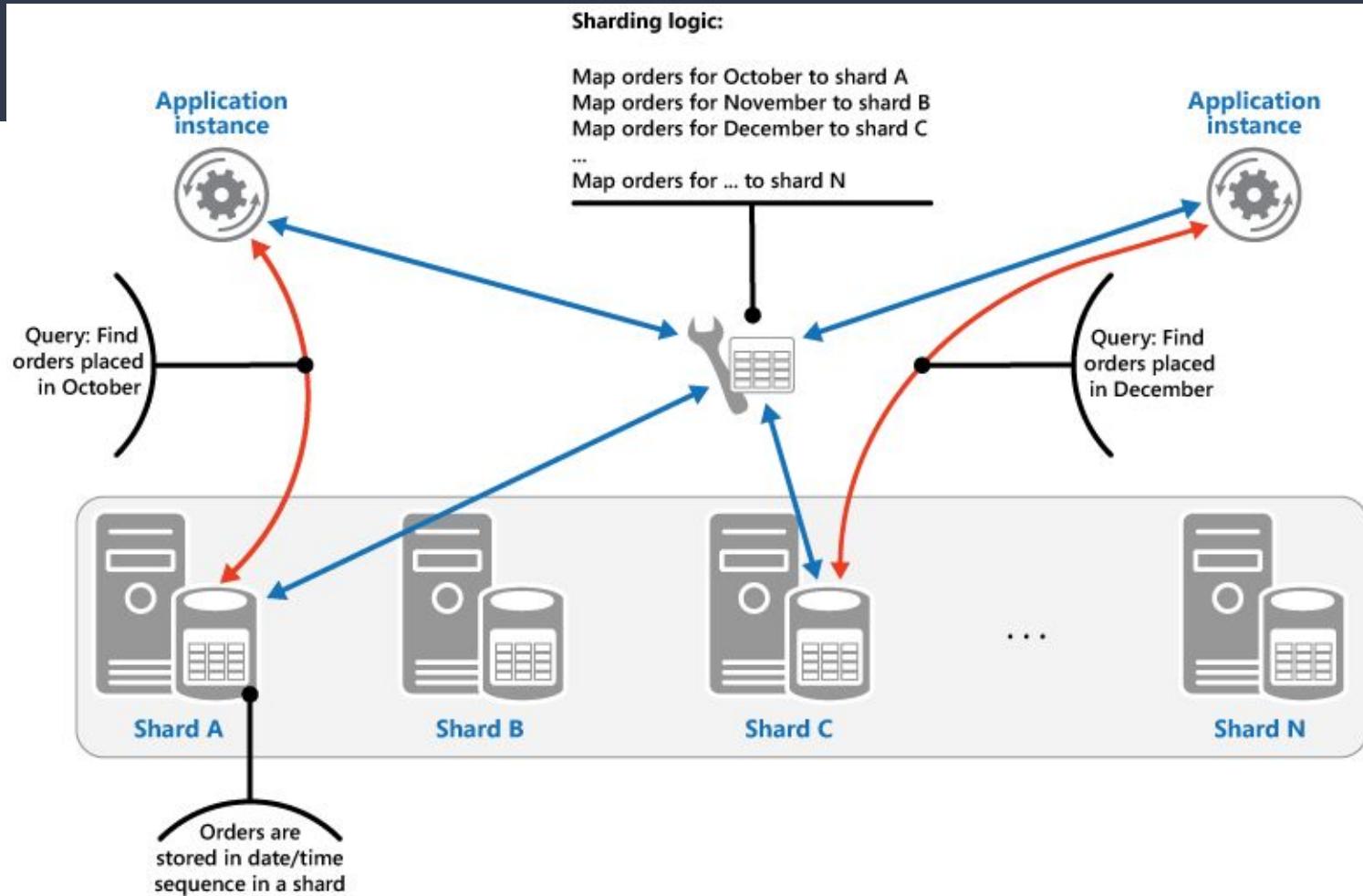
## **Dividir el almacenamiento de datos en particiones “horizontales” (shards)**

- Cada shard corre en su propio server y es independiente del resto
- Se puede escalar horizontalmente
- Se puede balancear la carga entre shards
- Los shards pueden ubicarse geográficamente cerca de los usuarios que lo usarán
- Cada shard contiene items que pertenecen a un rango, el cual es determinado por una shard key (o partition key)
  - Lookup strategy: mapa shardkey-shard
  - Range strategy: agrupar relacionados
  - Hash strategy:  $\text{Hash}(4) \Rightarrow \text{shard B}$
- Debe haber lógica de sharding, ya sea en el sistema de almacenamiento, o en la app



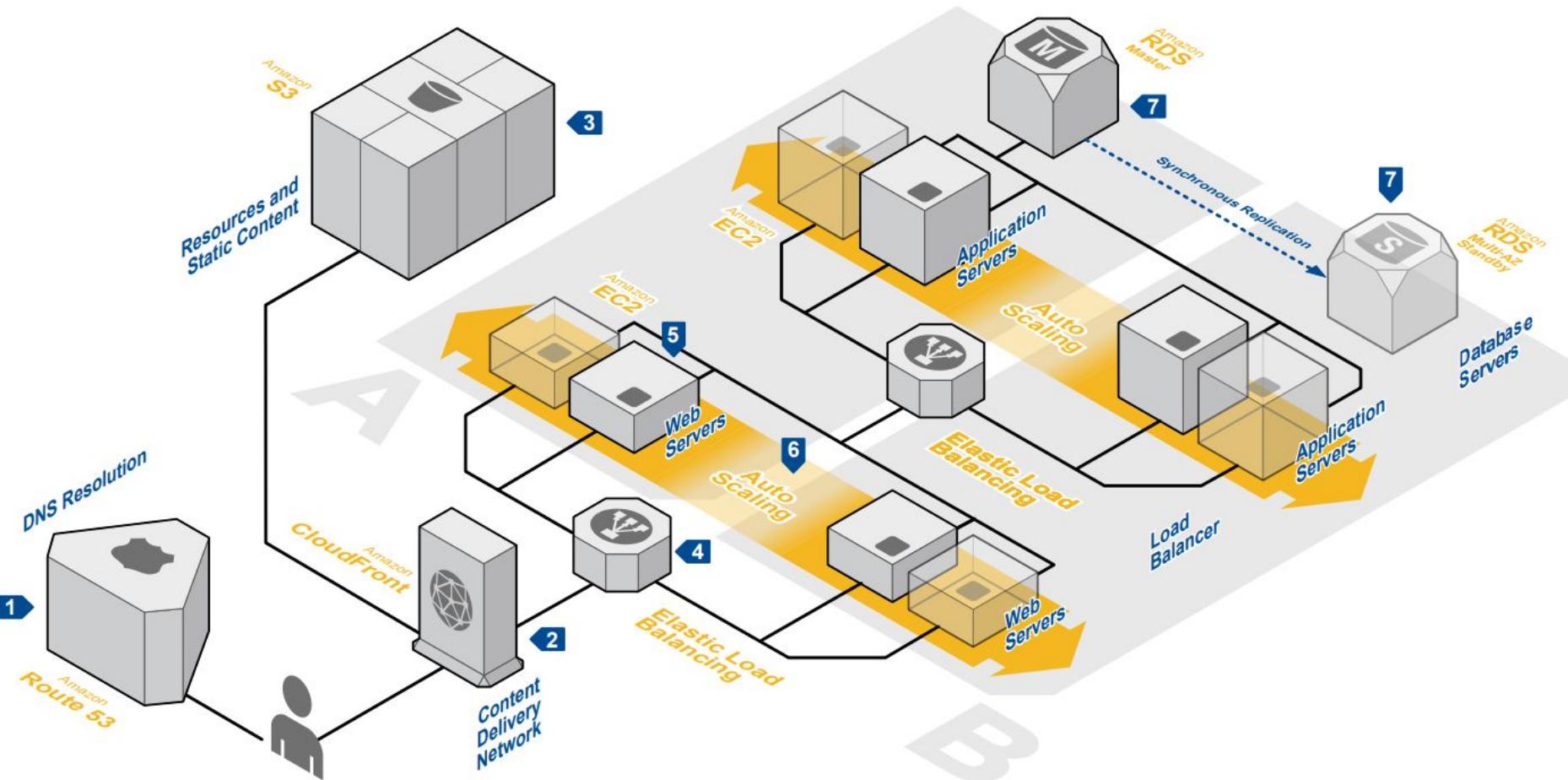
# Sharding

## Range Strategy - Ejemplo

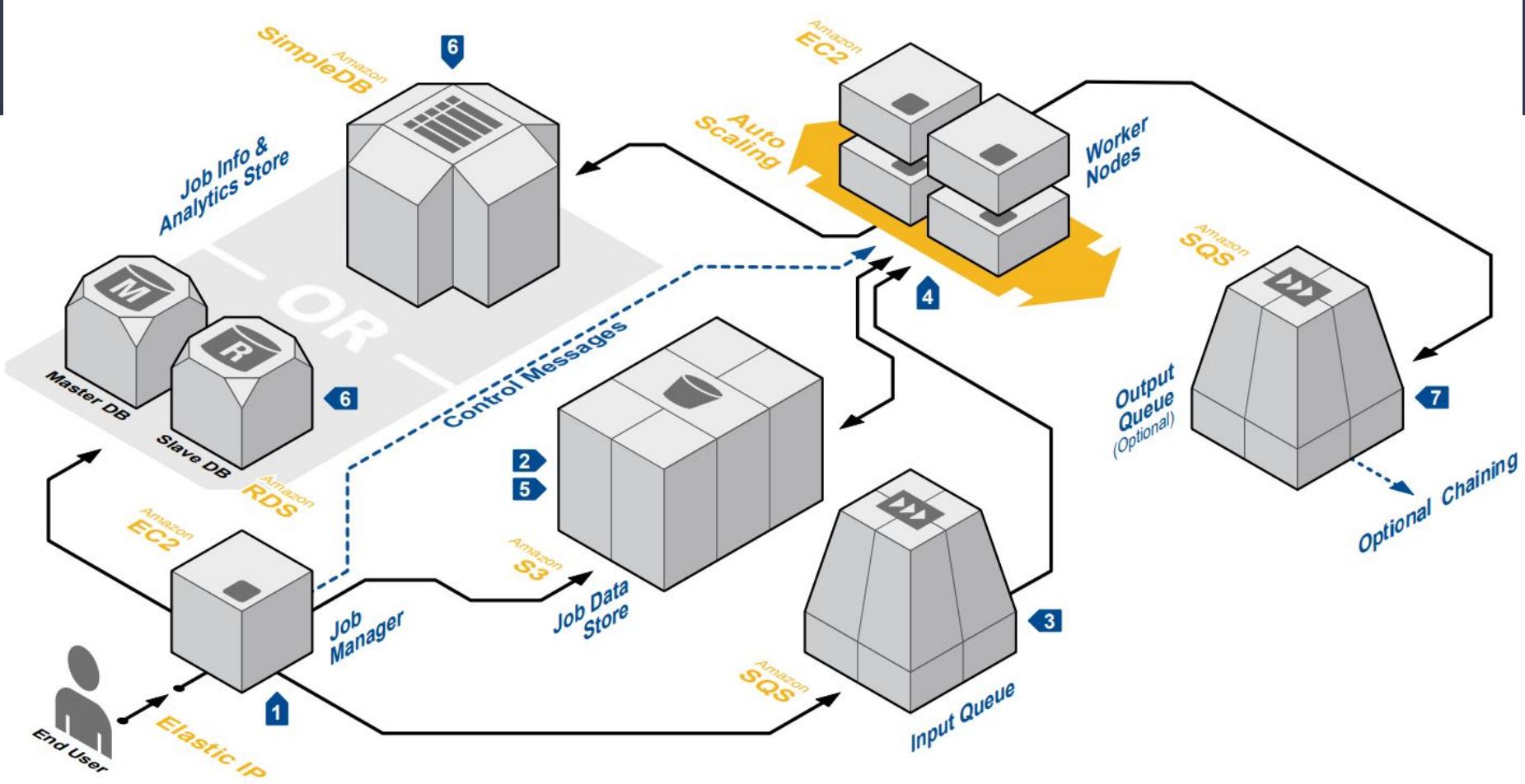


# Arquitecturas de referencia

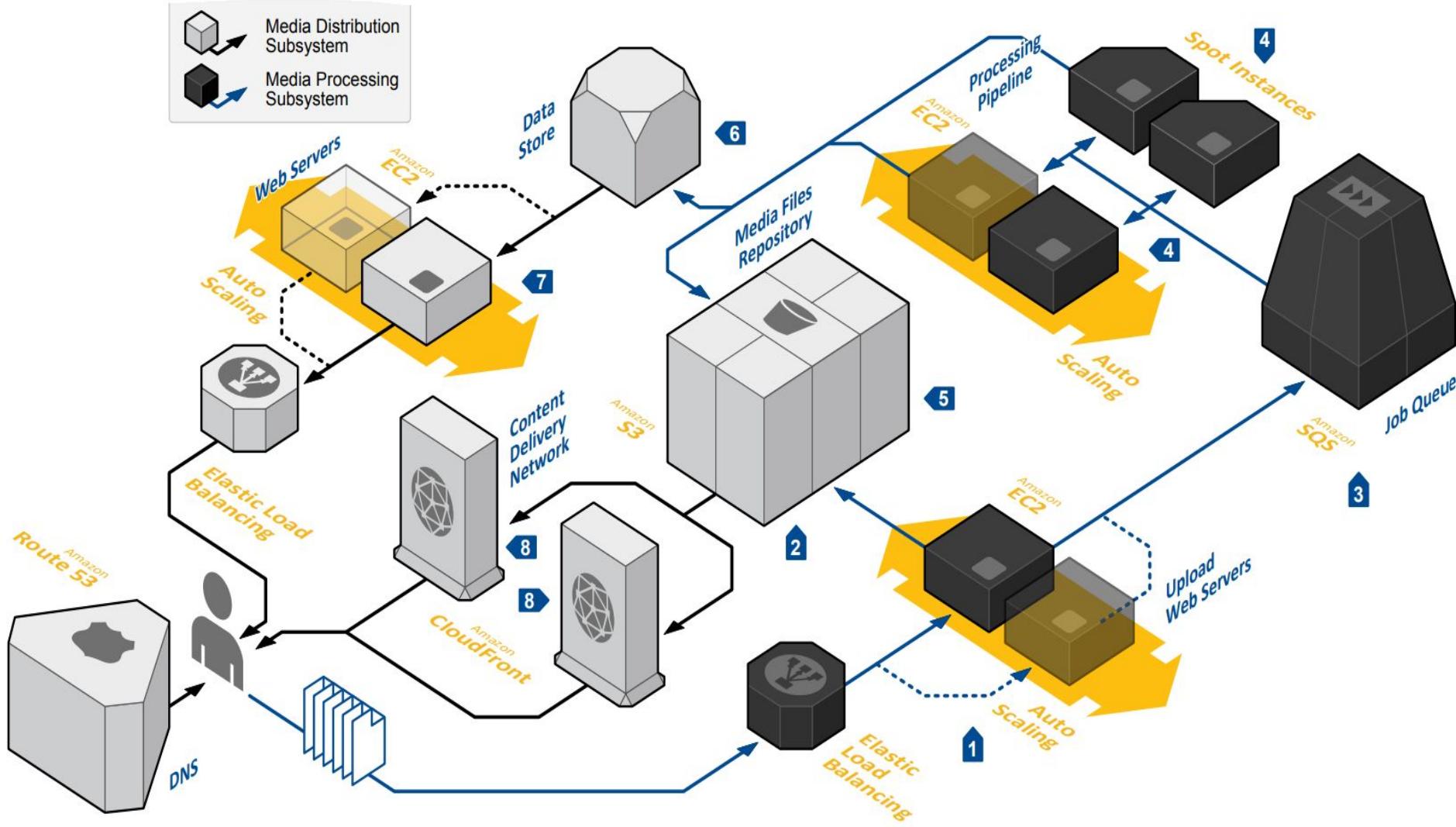
# Web App



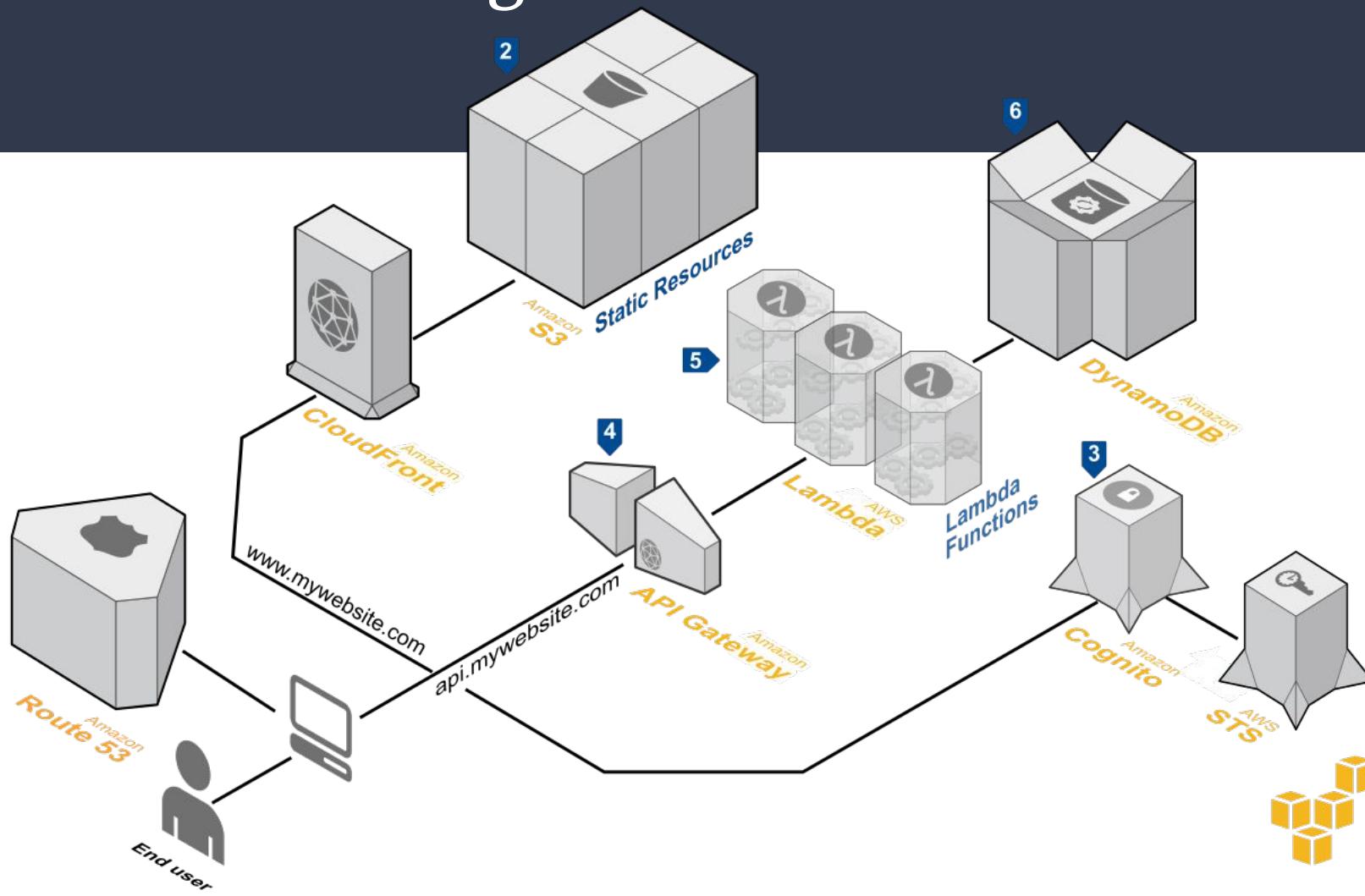
# Batch Processing



# Media Sharing



# Serverless blog



# ¿Consultas?

## Feedback

<https://goo.gl/forms/NvrORS12kuuBitpE3>

Guillermo Rugilo

[grugilo@fi.uba.ar](mailto:grugilo@fi.uba.ar)

¡Gracias!