Sistemas Distribuidos I
(75.74)

Sistemas elásticos y de alta disponibilidad

Técnicas de Diseño. Elasticidad. Disponibilidad.

Docentes

- Pablo D. Roca
- Ezequiel Torres Feyuk
- Guido Albarello
- Ana Czarnitzki
- Cristian Raña

Agenda

- **Escalabilidad**
- Elasticidad
- Alta Disponibilidad

Quizá el objetivo más importante para los sistemas distribuidos modernos.

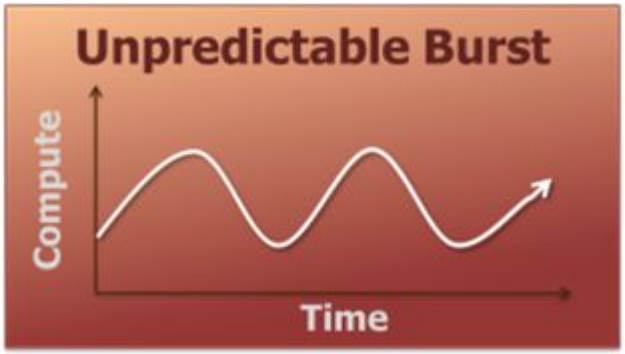
El objetivo de escalabilidad es el crecimiento,

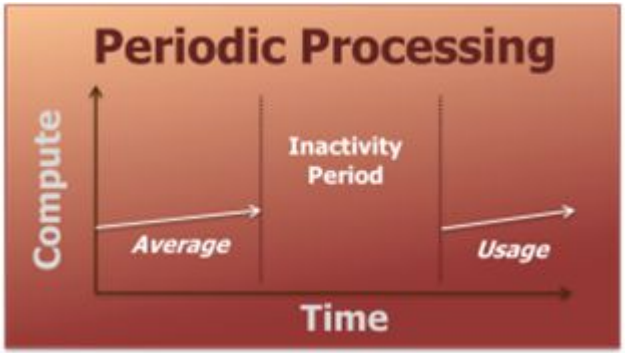
- ***Respecto del tamaño:*** Agregando usuarios o recursos a controlar.
- ***Respecto de la distribución geográfica:*** Permitiendo dispersión.
- ***Respecto de los objetivos administrativos del sistema:*** Nuevas sintaxis, semánticas y servicios ofrecidos.

Escalabilidad | Características de las Plataformas

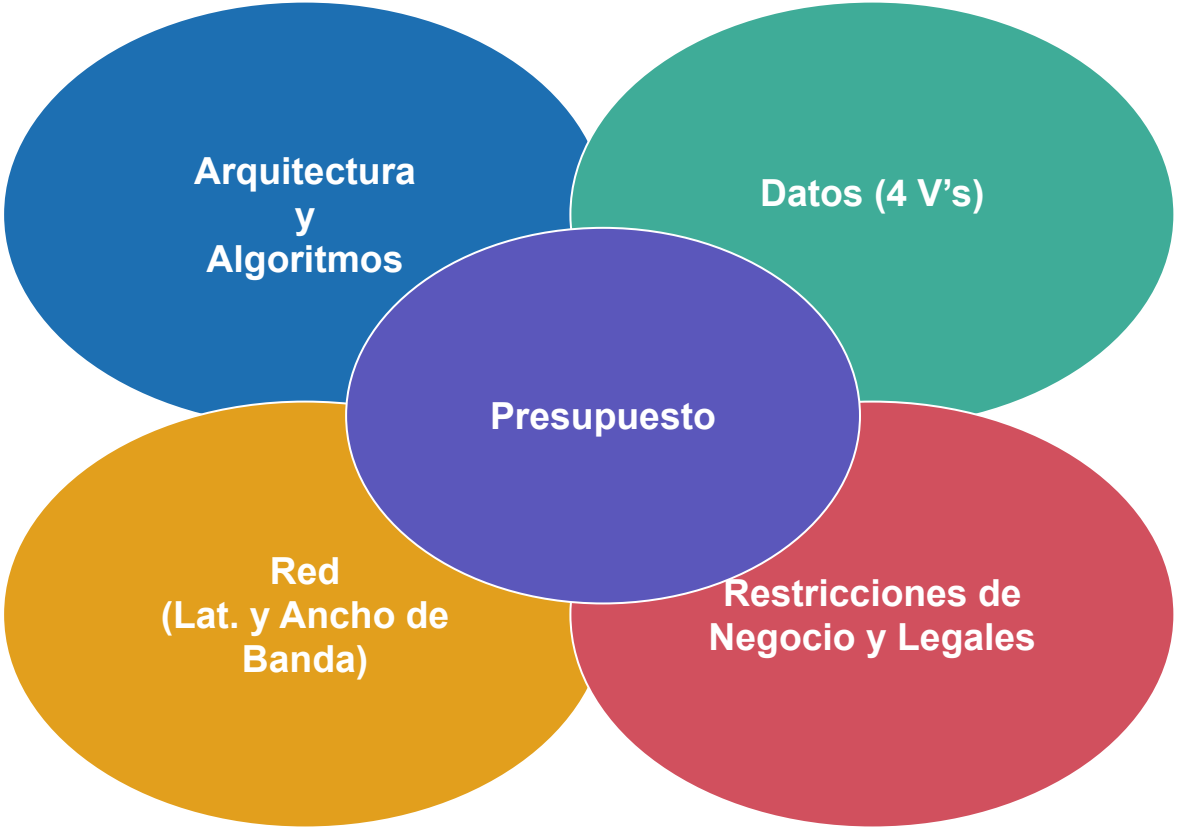
- **Plataformas para alta concurrencia**
  - Aplican patrones ya conocidos y probados
  - Escalamiento automático (con ciertos límites)
  - Vinculación fuerte con una infraestructura o producto
- **Arquitecturas Ad-Hoc y Personalizadas**
  - Necesidad de configuración y soporte
  - Escalamiento manual o automatizado por humanos
  - Posibilidad de migraciones a distintas plataformas

Escalabilidad | Patrón de Carga de Aplicaciones Web





Escalabilidad | Limitantes



Escalabilidad | Técnicas

- **Escalamiento vertical**
  - Agregar recursos a un nodo
- **Escalamiento horizontal**
  - Redundancia
  - Balanceadores de carga
  - Proximidad geográfica
- **Fragmentación de datos**
  - ¿Qué datos deben estar juntos?
- **Componentización**
  - Separar servicios
- **Optimizar algoritmos**
  - Performance
  - Mensajería
- **Asincronismo**
  - ¿Qué debe ser necesariamente sincronico?
  - Limitado por el negocio

Agenda

- Escalabilidad
- **Elasticidad**
- Alta Disponibilidad

Elasticidad | Introducción

- **Escalabilidad**
  - Capacidad de un sistema para poder adaptarse a diferentes ambientes (*environments*) modificando los recursos del sistema
  - Término utilizado en muchos contextos
- **Elasticidad**
  - Capacidad de un sistema para poder modificar dinámicamente los recursos del sistema adaptándose a patrones de carga
  - Término utilizado en Arquitecturas Cloud. Requiere soporte de la infraestructura

Elasticidad | Componentes

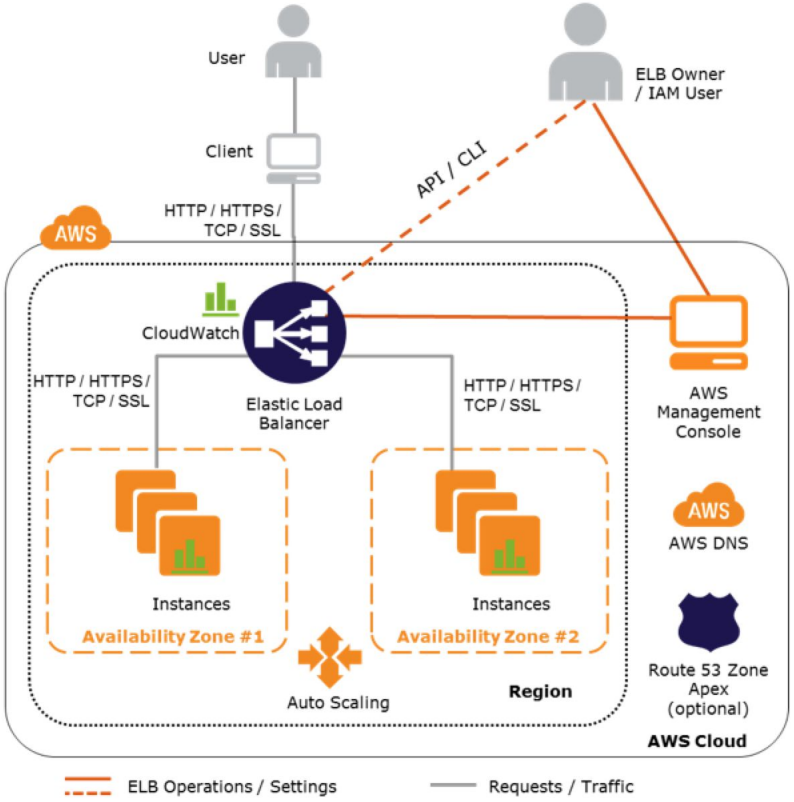
- ***Application Load Balancer***
  - Servicios / instancias nuevas reciben tráfico
  - Servicios / instancias dados de baja / caídos / degradados dejan de recibir tráfico
  - Cómo detectamos estado de un servicio / instancia ?
- ***Autoscaler***
  - *Scale In/Scale Out* en función de las métricas recolectadas
  - *Scale Out:* Incrementa instancias
  - *Scale In:* Decrementar instancias
- **Monitoring Automático**
  - Métricas sobre CPU, memoria, I/O, *networking*, etc. por cada servicio / instancia

Elasticidad | Ejemplo N#1 - AWS

- *Application Load Balancer (**Amazon Elastic Load Balancer**)*
  - Redirecciona tráfico a instancias (EC2), incluso en diferentes zonas
  - Interactúa con instancias para verificar estado (*/status* endpoint)
- *Autoscaler (**Amazon Autoscaling**)*
  - Autoscaling group permite definir *min*, *desired* and *max* instances
  - Diferentes policies para lograr diferentes objetivos (dynamic vs manual scaling)
- **Monitoring Automático (Amazon CloudWatch)**
  - Métricas globales automatizadas
  - Admite agregar métricas propias

A dynamic scaling policy instructs Amazon EC2 Auto Scaling to track a specific CloudWatch metric, and it defines what action to take when the associated CloudWatch alarm is in ALARM. The metrics that are used to trigger an alarm are an aggregation of metrics coming from all of the instances in the Auto Scaling group. When the policy is in effect, Amazon EC2 Auto Scaling adjusts the group's desired capacity up or down when the alarm is triggered.

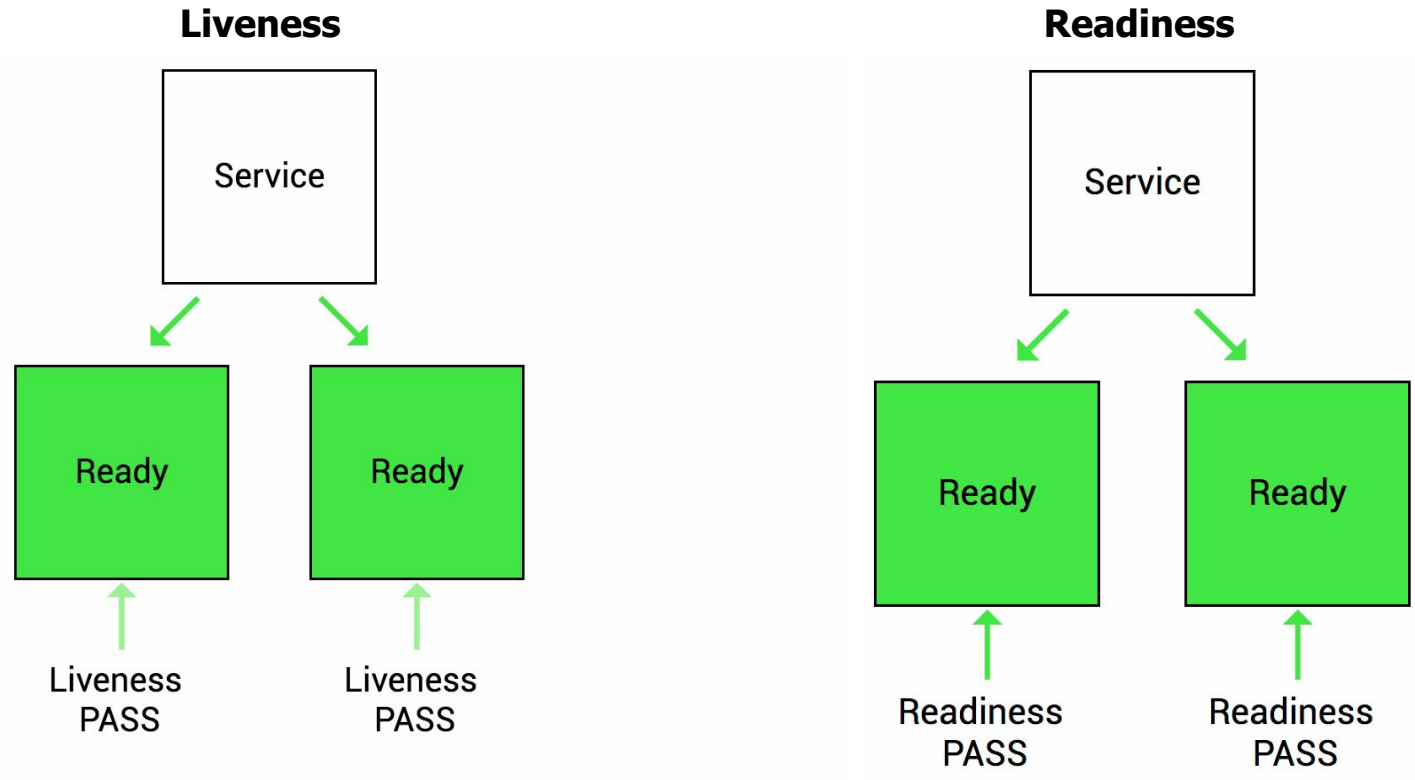
Elasticidad | Ejemplo N#1 - AWS



Elasticidad | Ejemplo N#2 - Kubernetes

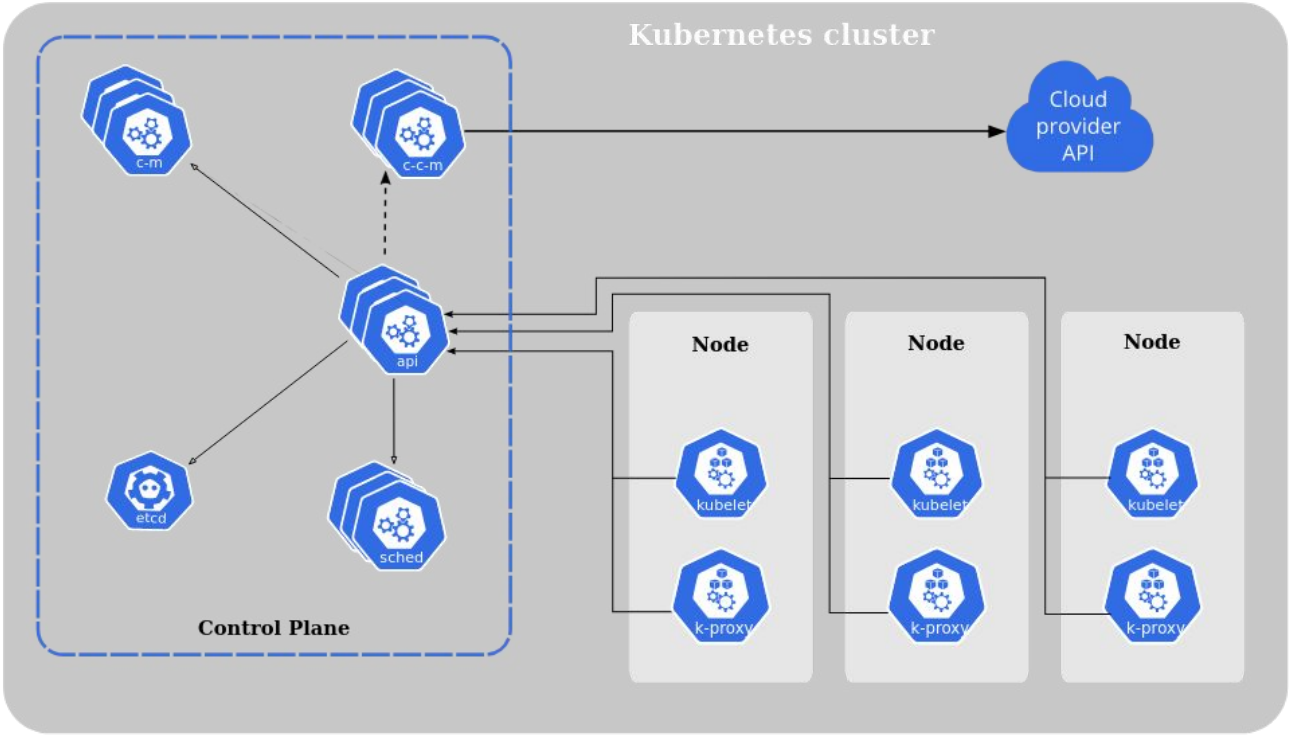
- *Application Load Balancer (**Kubernetes Service**)*
  - Redirecciona tráfico a pods en estado **Ready**
  - Liveness and Readiness probes permiten al usuario indicar si un pod debe recibir tráfico o no
- *Autoscaler (**Horizontal Pod Autoscale**)*
  - Se crea un HPA resource asociado a un deployment en función de alguna métrica
  - Similar a como funciona el autoscaler de AWS, se pondera las métricas de cada instancia/container
  - `desiredReplicas = ceil[currReplicas * ( currMetricValue / desiredMetricValue )]`
- **Monitoring Automático (Kubernetes Metrics Server)**
  - Collecta métricas de cada pod en el cluster y las expone en el API server
  - Diseñado específicamente para autoscaling

Elasticidad | Ejemplo N#2 - Kubernetes



https://cloud.google.com/blog/products/containers-kubernetes/kubernetes-best-practices-setting-up-health-checks-with-readiness-and-liveness-probes

Elasticidad | Ejemplo N#2 - Kubernetes



https://kubernetes.io/docs/concepts/overview/components/

Agenda

- Escalabilidad
- Elasticidad
- **Alta Disponibilidad**

Alta Disponibilidad | Introducción

Aún cuando ciertas propiedades de confiabilidad fallen para sistemas públicos, nos interesa que siempre estén disponibles:

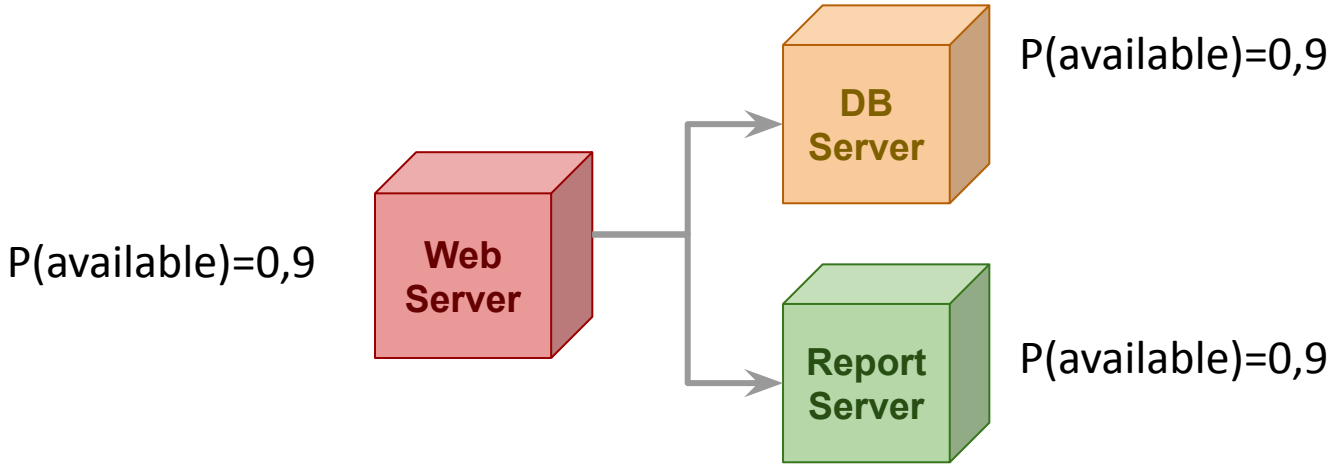
$$\mathbf{P(}\textit{system available}\mathbf{)} = 1 - P(\textit{failure})$$

Como **P(*system available*) < 1** siempre, lo importante es definir cuán cerca de 1 se encuentra:

- P(*system available*) ~ 0,9 => 36,5 días caídos al año
- P(*system available*) ~ 0,999 => 8,76 horas caídas al año

La disponibilidad de un sistema se mide por la cantidad de 9's

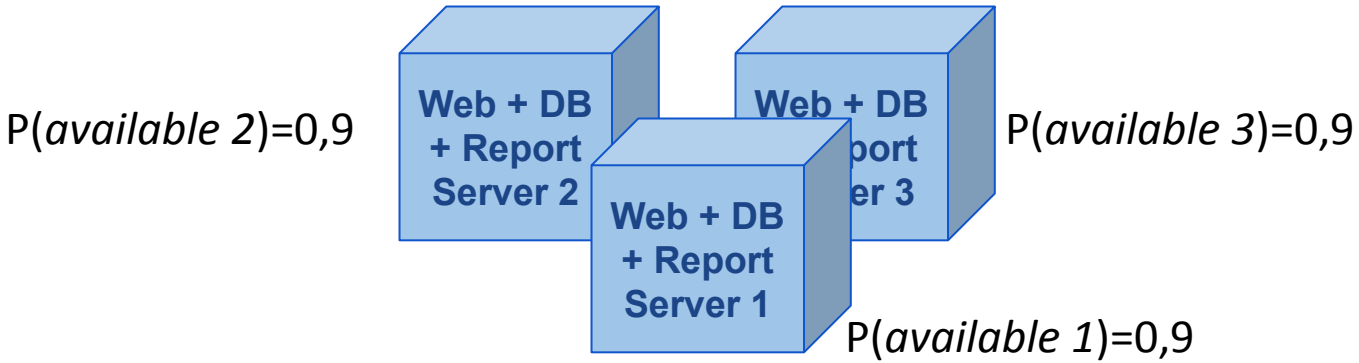
Escenario: Sistema distribuido con un componente por nodo



$$P(available) = P(webAvailable) \cdot P(dbAvailable) \cdot P(reportAvailable)$$

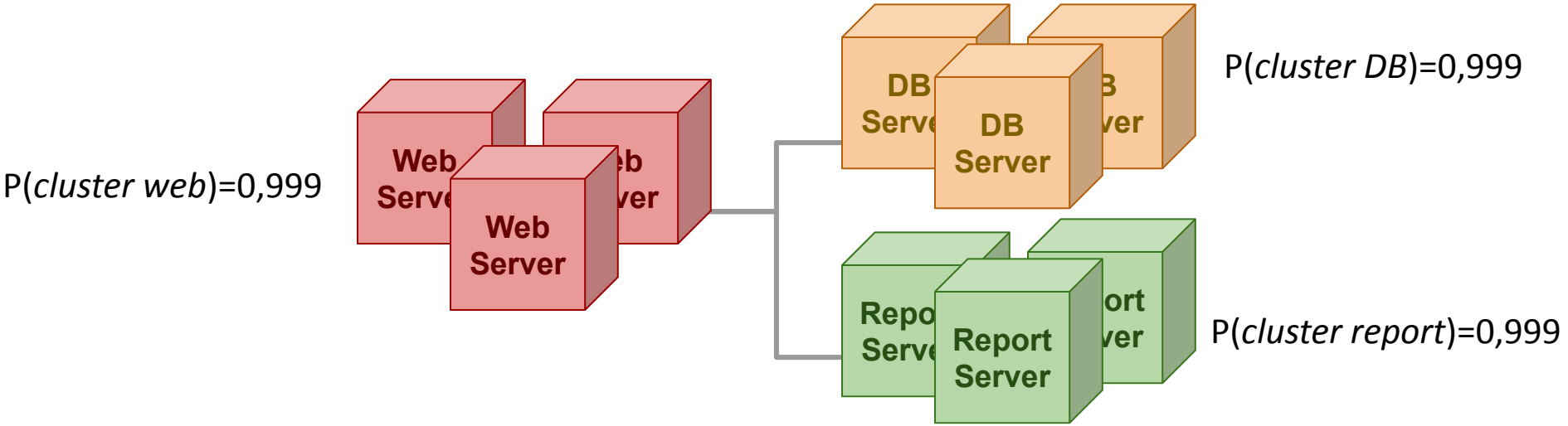
$$= 0{,}9 \cdot 0{,}9 \cdot 0{,}9 = \mathbf{0{,}729}$$

Escenario: Sistema con redundancia con todos los componentes en c/nodo



P(*available*) = 1 - P(*failure*) = 1 - P(*failure 1 & failure 2 & failure 3*)
= 1 - 0,1 . 0,1 . 0,1 = 1 - 0,001 = **0,999**

Escenario: Sistema distribuido con clusters para cada componente



P(*available*) = P(*cluster web*) . P(*cluster DB*) . P(*cluster report*)
= 0,999 . 0,999 . 0,999 = **0,997**

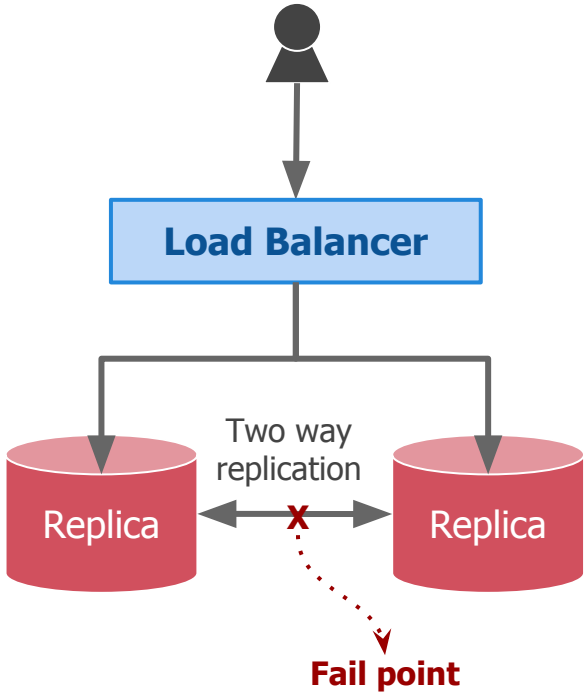
Alta Disponibilidad | Terminología en la industria

- **SLA: Service Level Agreement**
  - Contrato / Acuerdo de disponibilidad pactado con el cliente
  - Definición de qué sucede si el mismo no se respeta (e.g. BigQuery SLA)
- **SLO: Service Level Objectives**
  - Lo que se debe cumplir para no invalidar el SLA
  - E.g. Availability > 99.95%
- **SLI: Service Level Indicators**
  - Métricas a ser comparadas con los SLOs
  - Siempre deben ser superiores al threshold del SLO
  - Por lo general requiere una plataforma de observability
  - Analizar impacto del despliegue de los servicios

CAP *Theorem* | Problemas frente a particionamiento

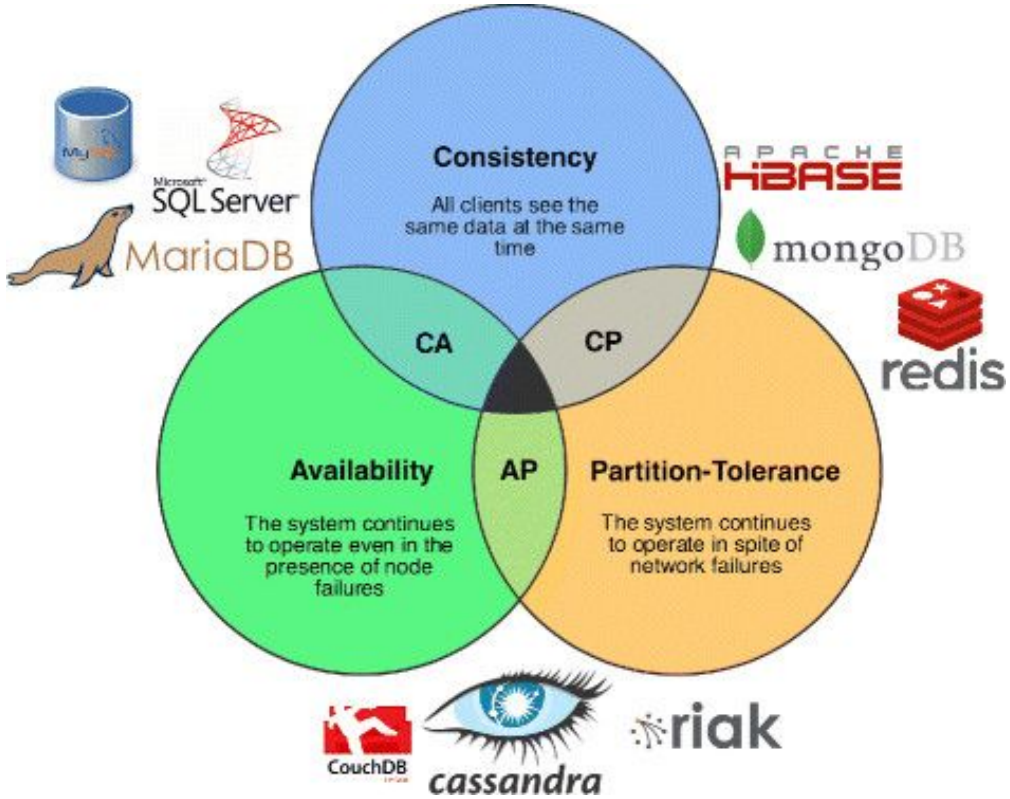
Eric Brewer ('98): en sistemas distribuidos con almacenamiento es posible garantizar sólo 2 de los siguientes atributos:

- ***Consistency:*** también conocido como repetibilidad de respuesta de todos los nodos frente a un mismo pedido.
- ***Availability:*** capacidad del sistema de responder a todo pedido.
- ***Partition Tolerance:*** capacidad de lidiar con la formación grupos aislados de nodos.



CAP *Theorem* | Ejemplos

- Se puede sacrificar consistency o availability. Pero no necesariamente es todo o nada.
- Sacrificar Partition-Tolerance significa no proveer un sistema distribuido.



Choosing the right NoSQL database for the job: a quality attribute evaluation, Lourenço et al. Journal of Big Data (2015)

Bibliografía

- G. Coulouris, J. Dollimore, t. Kindberg, G. Blair: Distributed Systems. Concepts and Design, 5th Edition, Addison Wesley, 2012.
  - Capítulo 18: Coordination and Agreement
- M. Van Steen, A. Tanenbaum: Distributed Systems. 3rd Edition. Pearson Education, 2017.
  - Capítulo 1.2: Introduction - Design Goals
  - Capítulo 8.1: Fault Tolerance - Introduction to Fault Tolerance