Infraestructura

TACS

Infraestructura

Todo recurso, generalmente hardware, que da soporte a una aplicación productiva

- Servidores físicos
- Contenedores de aplicación
- Networking
- Bases de datos

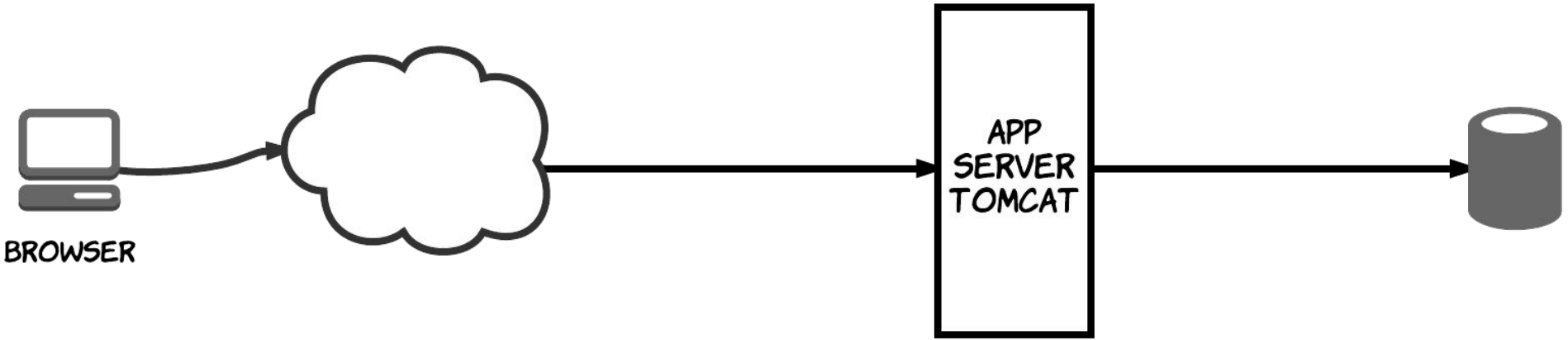
Infraestructura Tradicional

TACS

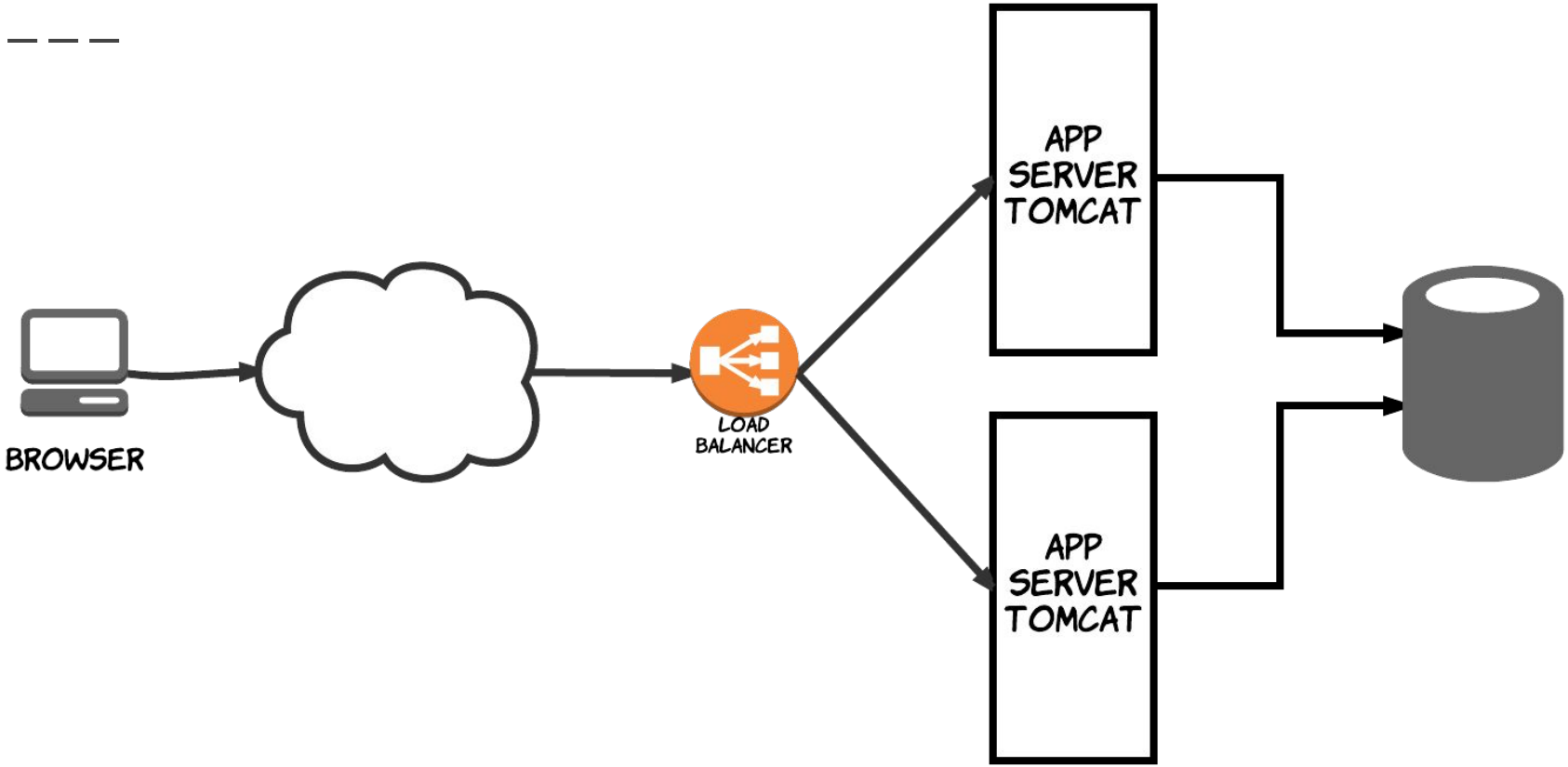
Infraestructura típica de una app web

Volvamos 10, 15 años atrás ¿Cómo se imaginan una arquitectura típica de una app cliente servidor, para 10 usuarios, para mi sistema de venta?

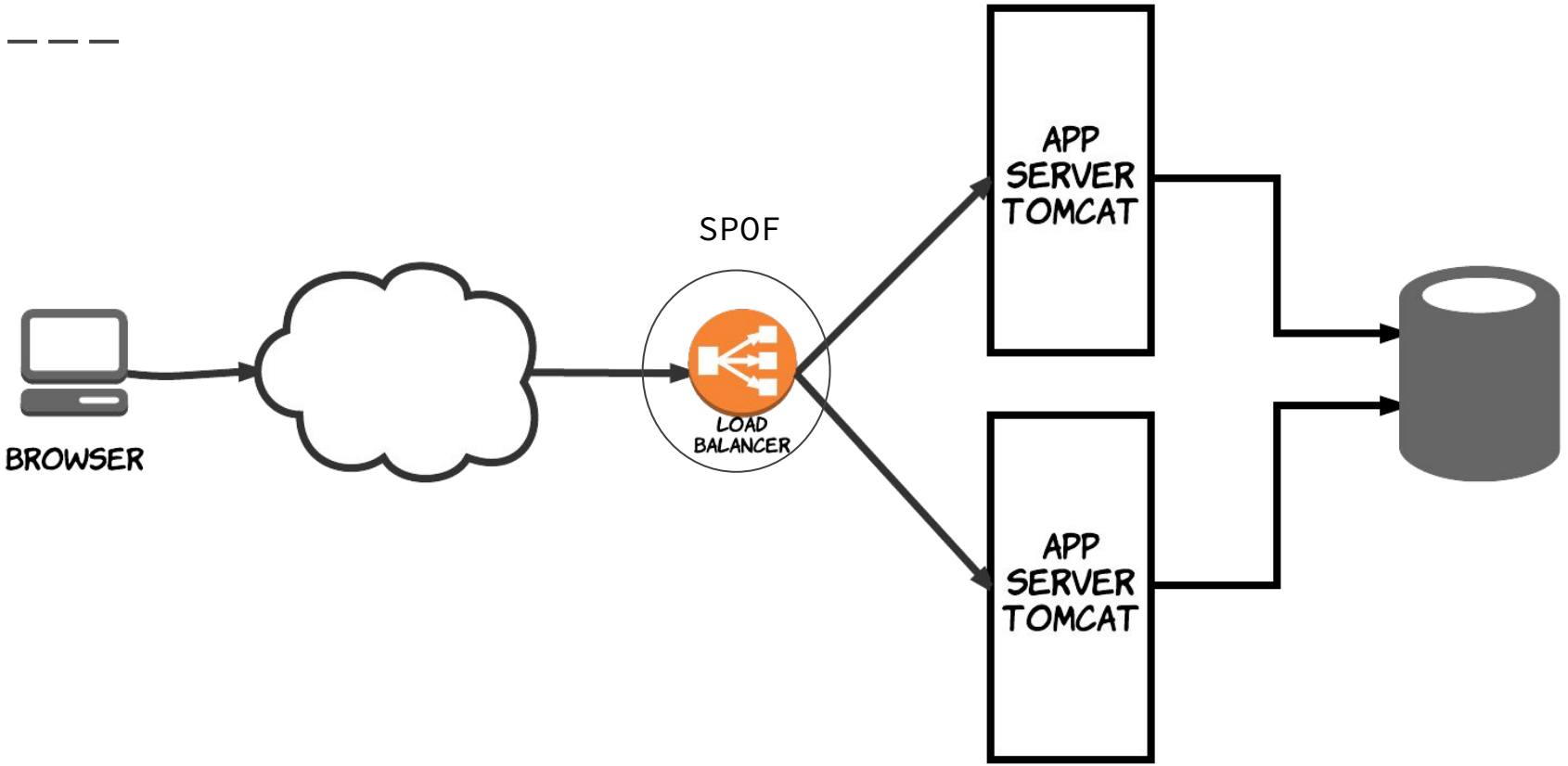
Infraestructura típica de una app web



Infraestructura típica de una app web



Infraestructura típica de una app web



Performance

Performance.

- Cantidad de trabajo útil completado por un sistema.
- Una forma de medir la performance es por Tiempo de respuesta
  - De servicios de la Aplicación (Application Services)
  - De experiencia del usuario final (End User Experience)

Escalabilidad

Escalabilidad.
Propiedad de un sistema de adaptarse al crecimiento del uso (mayor **throughput**), manteniendo la calidad de servicio o **tiempos de respuesta** (Performance).

- Cuando los servidores empiezan a decaer en performance, puede producirse un efecto en cadena degradando todo el sistema

- Picos de carga implica mayor uso de recursos, luego están en desuso.

Tipos de Escalabilidad

- Vertical (+ power)
- Horizontal (+ nodos)

Escalabilidad Horizontal

Clustering

- Alta disponibilidad. (Uptime)
- Load Balancing (para mantener performance, distribución de carga)

Alta Disponibilidad

Habilidad del sistema para continuar funcionando luego de la falla de uno o más servidores.

Disponibilidad o **Uptime** = % tiempo online

Alta disponibilidad se mide en 9s, 99,99%

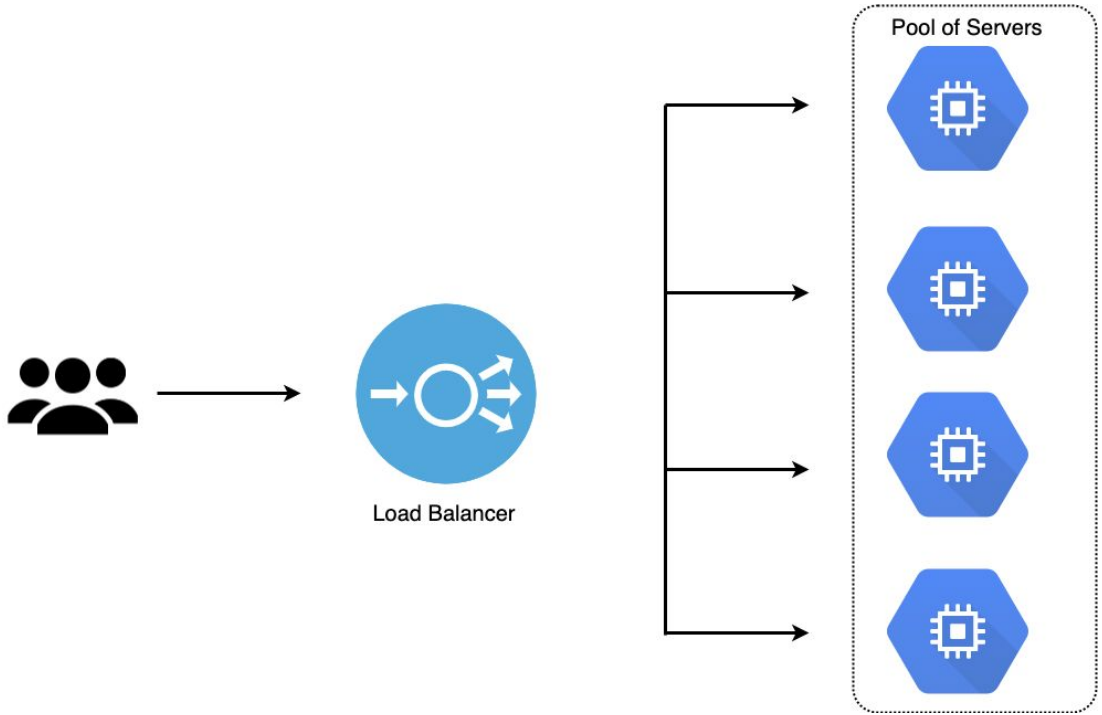
Alta Disponibilidad

- Promedio de tiempo entre fallas **ATTF**
- Promedio de tiempo en recuperarse **ATTR**
- **Availability** = **(ATTF) / (ATTF + ATTR)** 24 / (24+1) = 0,96
  - Para cumplir con cierto Availability **ATTR < (ATTF - AV * ATTF) / AV**
    - Availability **0,999** por hora implica que el ATTR < (24-0.999*24)/0.999) = 1:26 minutos
- Tipos de aplicación / Horarios de uso
- Estrategias al deployar
  - Canary deploy
  - Blue Green Deploy
  - A/B Testing
  - Disabled Features
- Ghost Replication. Active - Passive Cluster (Failover)

Load Balancing

- Es el proceso de distribuir eficientemente el tráfico de red entre múltiples servidores.
- Es las forma más directa de escalar una aplicación.
- Tipos de Load Balancer
  - Físicos
  - Software

Load balancer



Algoritmos de Load Balancing

- Round Robin
- Load Balance Factors
  - Menor cantidad de conexiones
  - Menor tiempo de respuesta
  - Menor ancho de banda
- IP Hash

Load Balancing

- Stateless apps vs Stateful apps
- Stickiness
    - Pérdida de sesiones ante fallas.
- Centralized
    - Salto de red y SPOF (Redis / Memcached)
- Session Replication
    - Overhead en memoria y networking

Load Balancing - Session Replication

- Sincrónica o Asincrónica
  - Replication Triggers (Cuando considero dirty la session):
    - SET
    - SET_AND_GET
    - SET_AND_NON_PRIMITIVE_GET
    - ACCESS
  - Replication Granularity
    - **Attribute**: sólo dirty attribs + lastTimeStamp
    - **Session**: Toda la session serializada viaja
    - **Field**: solo viajan los campos modificados de los atributos

Tomcat / Jboss Session Replication Triggers

Level	Description	Speed
SET	This is the best option for performance. **The session is replicated only if** an attributed is explicity modified with **setAttribute** method.	**good**
SET_AND_GET	With this option **any attributed that is get/set is marked as dirty** even if it's not written back to the session. This leads to a significant performance degradation.	**slow**
SET_AND_NON_PRIMITIVE_GET	This is the **default** option. **It works the same as SET_AND_GET except for primitive system types** (String, Integer, Long). Since they are immutable objects they are not replicated when a get is issued.	**average**
ACCESS	This is the most conservative option. **It causes the session to be marked as dirty whenever it is accessed.** Since a the session is accessed during each HTTP request, it will be replicated with each request. Note that use of this option can have a significant performance impact, so use it with caution	**v.slow**

Dirty Session

Command	SET	SET_AND_NP_GET	SET_AND_GET	ACCESS
`HttpSession session;`	NO	NO	NO	YES
`String s = (String)session.getAttribute("x");`	NO	NO	YES	YES
`Person p = (Person)session.getAttribute("p");`	NO	YES	YES	YES
`Person p = (Person)session.getAttribute("p");` `session.setAttribute(p);`	YES	YES	YES	YES

Content Delivery Network

- https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.8.0/angular.min.js
- Para contenido estático, **más cerca del cliente**.
- Reduce la carga de los servidores.
- Red de tráfico distribuida.
- Reduce la **latencia**.
- Incrementa el ancho de banda.
- Mejora web caching.

Failover / Transparent Failover

Para lograr HA es importante el mecanismo de **failover**

Failover > la habilidad de que las conexiones del cliente migren de un servidor a otro, en eventos de fallas de servidor, por lo que las apps clientes pueden continuar operando.
Transparent Failover > El cliente no se entera de la caída del servidor. El balanceador redirige el request a otro server disponible. Puede ser a nivel request o nivel server

Anti Patterns para Clusterización

- Singleton
- Abusar del estado
- Uso de recursos remotos sin cachear (DB, APIS)
  - Tipos
    - Local, en memoria (hashmaps, Infinispan)
    - Centralizada (MemCached)
  - Concerns
    - Invalidación de entradas
    - Eficiencia en uso de memoria

Hardware vs Virtualizado

- Costo/beneficio
  - Reducción de costos de infra.
  - Minimizar o eliminar downtime.
  - Mejora la productividad, la eficiencia, la agilidad y la respuesta de IT.
  - Acelera y simplifica el aprovisionamiento de aplicaciones y recursos.
  - Soporta business continuity y disaster recovery.
- Permite un manejo centralizado.

Infraestructura Cloud

TACS

Cloud Computing

Un modelo que da acceso **on-demand** a un pool compartido de **recursos** computacionales que puede ser **rápidamente provisionado**, con **baja o nula interacción** del proveedor del servicio.

- Network
- Servers
- Storage
- Services

Características Cloud

- **On-demand self-service**. Capacidad para empezar a servir, sin las demoras de IT tradicional.
- **Acceso a través de internet**. Es accesible desde cualquier plataforma (browsers, teléfonos, etc)
- **Resource pooling**. Los recursos son compartidos entre distintos clientes transparentemente
- **Rapid elasticity**. Capacidad de escalar para soportar picos de tráfico
- **Measured Service**. La factura es medida. Pago por lo que consumo.

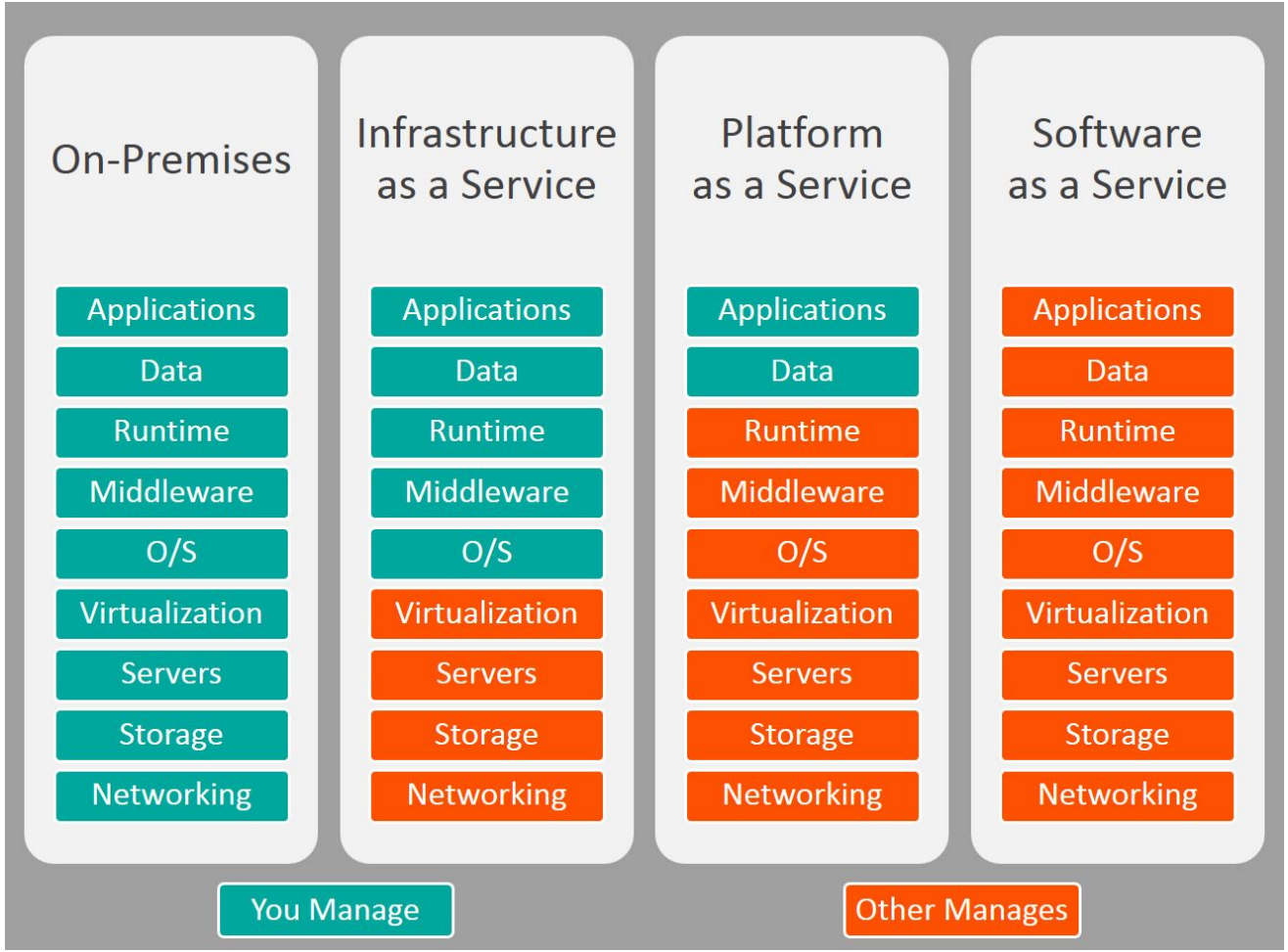
Características Cloud

- Agilidad
- Costo
- Mantenimiento
- Centralización
- Capacidad de respuesta ante picos de carga
- Utilización y eficiencia
- Performance
- Productividad
- Confiabilidad
- Escalabilidad y elasticidad
- Seguridad*

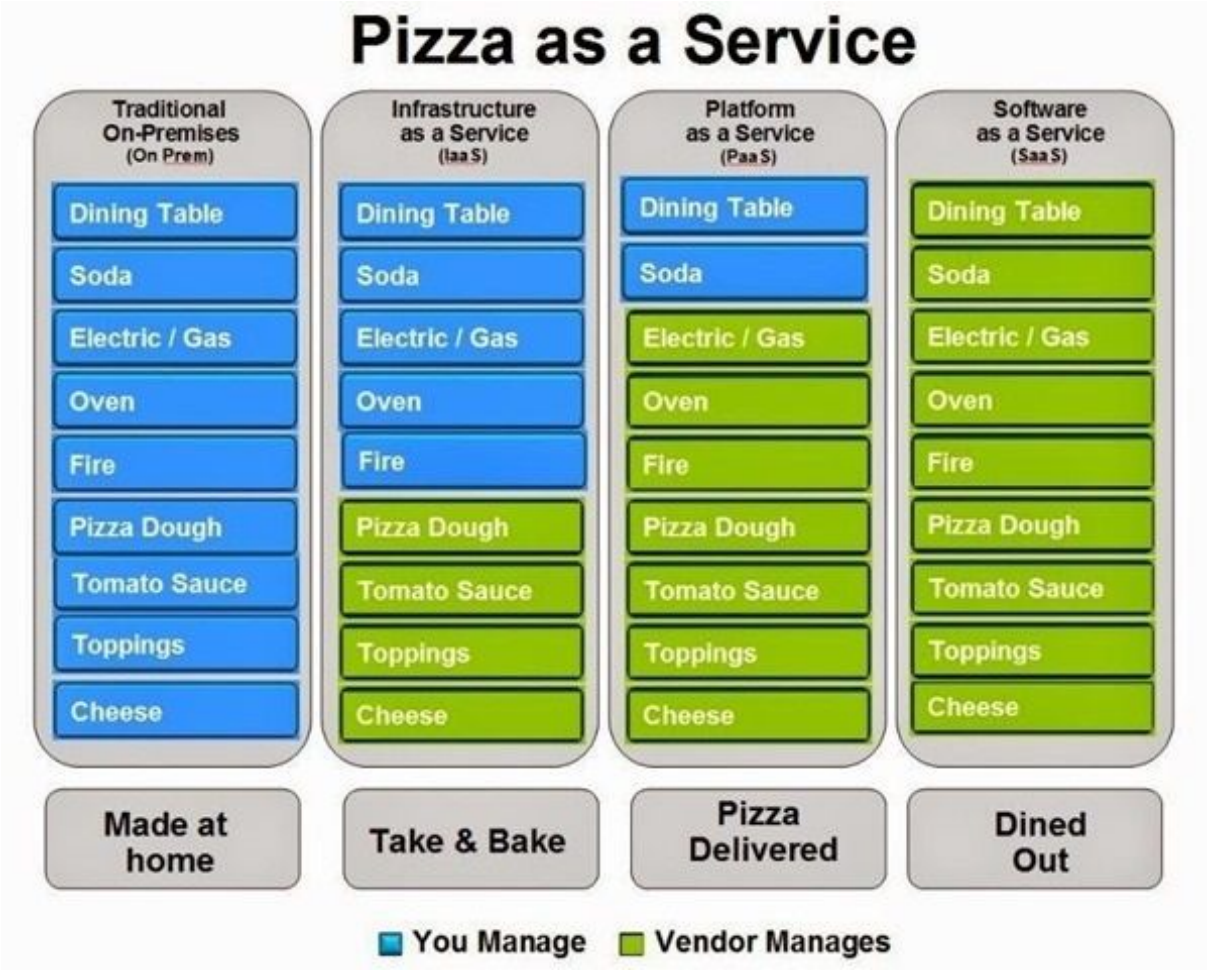
Flavors

IAAS	Virtual Machines, servers, Load Balancers, networks...
PAAS	Execution Runtime, database, webservers, Development Tools
SAAS	EMail, CRMs, Virtual Desktop...

Flavors


On-Premises
Infrastructure as a Service
Platform as a Service
Software as a Service
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
You Manage
Other Manages

Flavors


Pizza as a Service
Traditional On-Premises (On Prem)
Infrastructure as a Service (IaaS)
Platform as a Service (PaaS)
Software as a Service (SaaS)
Dining Table
Soda
Electric / Gas
Oven
Fire
Pizza Dough
Tomato Sauce
Toppings
Cheese
Made at home
Take & Bake
Pizza Delivered
Dined Out
You Manage
Vendor Manages

Cloud vs Infra Tradicional

Pros

- Elastic scale
- Servicios provistos por el proveedor (mails, colas, DBs, etc)
- Puedo empezar con bajo costo e ir creciendo
- Fácil evolucionar / “refactorizar”
- Distribuido geográficamente (comentar CDN)

Cloud vs Infra Tradicional

Cons

- No tengo los datos en mi poder
- Dependo de un externo (si se cae, pierdo)
- Dependiendo el tamaño tengo que ver el costo
- Me cobran por todo lo que hago (CPU, red, tiempo, etc)
- Tengo menos flexibilidad, no siempre puedo hacer todo lo que quiero

Infrastructure as a Service (IaaS)

Infrastructure as a Service (IaaS) es una forma de entregar **infraestructura** - servers, storage, network and operating systems - **como un servicio on-demand.**

En vez de comprar servers, software, espacio en datacenters o equipos de red, los clientes **compran estos recursos como un service on demand totalmente outsource.**

IaaS

Pros

- Puedo hacer lo que quiero
  - Correr procesos
  - Instalar paquetes
  - Levantar cualquier lenguaje o aplicación que tenga
- Fácil para portar aplicaciones existentes.
- Servicios provistos por la plataforma (no pasa en un Infra tradicional).

IaaS

Cons

- Escalabilidad manual (más trabajo).
- Tengo que tener determinados perfiles en el equipo (devops, infra, depende que use DBAs).
- Tengo un costo de licencias

IaaS

Ejemplo

- Amazon EC2
- GCP
- Azure

Platform as a Service (PaaS)

PaaS puede ser definido como una **plataforma computacional** que permite la **creación de apps webs rápida y fácilmente** sin la complejidad de comprar y mantener el software y la infraestructura por detrás de éstas.

Platform as a Service (PaaS)

- Servicios para desarrollar, testear, deployar, hostear y mantener aplicaciones, están integradas en el mismo ambiente de desarrollo.
- Consolas Web UI para crear, modificar, testear y deployar apps
- Arquitectura Multi-tenant donde multiple usuarios concurrentes utilizan la misma app de desarrollo

Platform as a Service (PaaS)

- **Autoscaling** de la app incluído por default, incluyendo **Load Balancing** y **Failover**
- Soporte para desarrollo colaborativo - project planning y herramientas de comunicación
- Herramientas para manejo de cobros y suscripciones

PaaS Vs IaaS

Pros

- Escala solo
- Menos costo de mantenimiento
- Poco conocimiento de licencias, infra, etc.
- Servicios provistos por la plataforma (no pasa en un Infra tradicional)

PaaS Vs IaaS

Cons

- App acoplada a la plataforma
  (Vendor lock-in)
- Requerimientos acotados a lo provisto por la plataforma.
  - Ej.: No sockets, no files, no threads.
- Request con timeouts.
- BB.DD. supeditada a la plataforma.
- Subset de la JDK (esto pasa con GAE)
- No es aplicable a cualquier Framework.

Platform as a service (PaaS)

Ejemplos

- Google App Engine
- Heroku
- AWS Beanstalk

Software as a service (SaaS)

- El proveedor de este software licencia la aplicación a los clientes como un servicio on demand, a través de dos formas:
  - Una suscripción, en un modelo “pay-as-you-go”
  - Sin cargo, cuando se pueden generar ingresos por publicidad o similares

SaaS - Características

Acceso Web a software comercial

- Se maneja desde una ubicación central
- Modelo “one to many”
- Los usuarios no tienen que encargarse de upgrades y patches
- Integración vía APIs

Donde SaaS tiene sentido

- “Vanilla” Software
- Email newsletter campaign software
- Necesidad de acceso web o mobile. Ej. sales management software
- Software que va a ser usado por un período corto
- Software con altos picos una vez por mes(liquidación de sueldos, impuestos)

Donde SaaS puede no ser la mejor opción

- Cuando se requiera procesamiento ultra rápido o datos en **tiempo real**
- Donde exista una **legislación u otras regulaciones que** no permitan mantener la data hosteada externamente
- Donde **ya exista una solución on-premise** que resuelva todas las necesidades de la empresa

Software as a Service (SaaS)

- Google Apps
- Salesforce.com
- Trello.com
- Slack.com

Cloud - Pricing

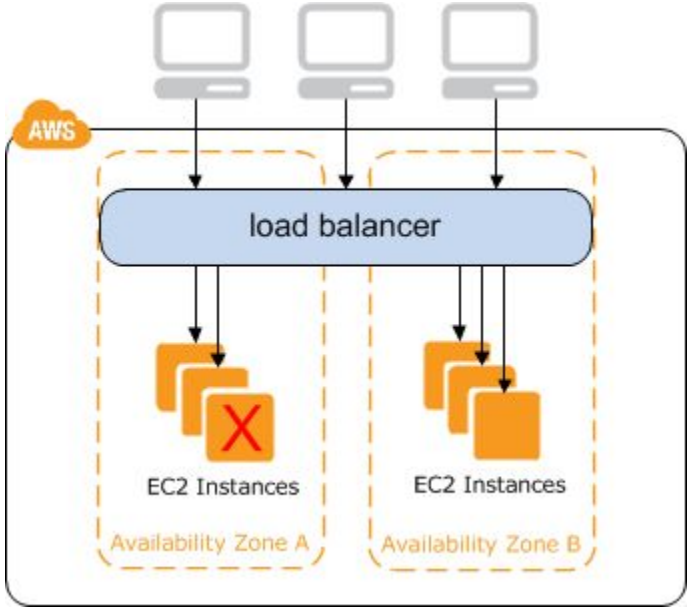
En qué punto una empresa va por un cloud público a uno privado o híbrido

AWS - IAAS

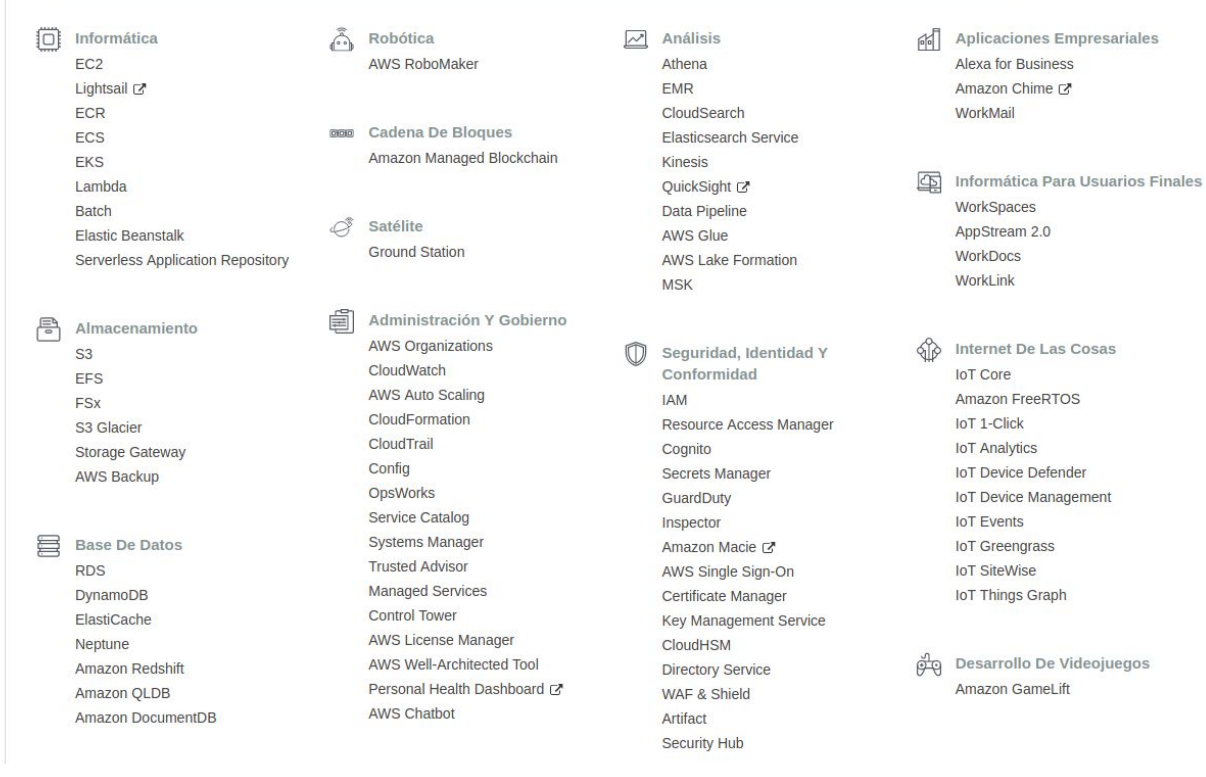
EC2

- Infraestructura de cómputo
  - Instances
  - Launch templates
  - **A**uto **S**caling **G**roups, Cloudwatch
- Load Balancers
  - **ApplicationLB**(http/https), **NetworkLB**(tcp/udp/tls)
- Network
  - **V**irtual **P**rivate **C**loud
  - **A**vailavility **Z**ones
  - **S**ecurity **G**roups

EC2


AWS
load balancer
EC2 Instances
Availability Zone A
EC2 Instances
Availability Zone B

Productos AWS



AWS - Facturación

Cost Explorer

-Gasto por cuenta

-Gasto diario

-Gasto por servicio

Nuevas tendencias

- Function as a Service (serverless)(Ej Lambda)
- Backend as a Service
  - Auth0
  - Firebase (ya no es solamente una DB, Serverless Apps)

Práctica

IaaS

Preguntas?

Material Extra

- http://highscalability.com/blog/2016/3/30/should-apple-build-their-own-cloud.html
- https://medium.com/@davidmytton/how-much-is-spotify-paying-google-cloud-ebb3bf180f15
- https://medium.com/@davidmytton/aws-vs-google-cloud-flexibility-vs-operational-simplicity-dca4324b03d4
- https://medium.com/simone-brunozzi/the-cloud-wars-of-2016-3f87e0a03d18

Links útiles para la práctica

EBT

https://aws.amazon.com/es/elasticbeanstalk/getting-started/

https://aws.amazon.com/es/blogs/devops/deploying-a-spring-boot-application-on-aws-using-aws-elastic-beanstalk/

https://docs.aws.amazon.com/es_es/elasticbeanstalk/latest/dg/eb3-cli-git.html

Links útiles para la práctica

ECS y ECR

https://docs.aws.amazon.com/AmazonECR/latest/userguide/repository-create.html

https://console.aws.amazon.com/ecs/home#/firstRun

https://aws.amazon.com/es/getting-started/hands-on/deploy-docker-containers/

https://cloudonaut.io/ecs-vs-fargate-whats-the-difference/

Gracias!

TACS