Arquitecturas orientadas a Servicio

Monolíticas

`-Request-> ReverseProxy (<-> Static Files) -> WebServer -Query-> DBServer`

Escalables

- Web Requests: **+Web Servers**.
  - (+) Routeo y Escalabilidad
  - (-) SPOF.
- Data Queries: **+DB Servers**.
  - (+) Throughput lectura.
  - (-) Throughput escritura.

Service Oriented (SOA)

- **Paradigma** orientado al **ámbito corporativo**.
- **Business Process Management (BPM).** Disciplina involucrando modelado, automatización, ejecución, control, métricas y optimización de los flujos de negocio p/ objetivos de empresa.

Características

- **Tecnologías:**
  - Web Services (SOAP + HTTP).
  - **Enterprise Service Bus** para **eventos**.
  - Service **Repository** & **Discovery**.
    - Comunicación punto a punto.
- **Servicios y Procesos:**
  - Interfaces.
  - Contratos.
  - Implementación: business logic + data management.

Microservicios

- +Granularidad.
- Escalabilidad Horizontal.
- Flexibilidad de Negocio.
- Monitoreo y Disponibilidad parcial.

Serverless

Cloud

- Todo lo que se puede consumir más allá del firewall.
- `Networking + Infra + Nuevas plataformas + Servicios`

Niveles de abstracción

- **IaaS.**
  - Almacenamiento y **virtualización de equipos**.
  - Definir redes y adaptación frente a carga.
  - **Customer Managed:** Apps, Security, Databases, OS.
  - *Ej. Google Cloud Storage.*
- **PaaS.**

  - Frameworks y plataformas p/ desarrollar aplicaciones *Cloud Ready*.
  - Recursos expuestos como **servicios p/ desarrollo** y **manejo de ciclo de vida** (logs,monitoreo).
  - **Customer Managed:** Apps.
  - *Ej. Google AppEngine.*
- **SaaS.**
  - Alquiler de servicios, **software a demanda**.
  - Soluciones **genéricas** y **adaptables**.
  - Arquitecturas pensadas p/ **integración**.
  - **Customer Managed:** NADA.
  - *Ej. Google Apps.*

Beneficios

- **Accesibilidad.** Movilidad y visibilidad constante.
- **Time-to-Market.** Recursos instantaneos.
- **Escalabilidad.** Capacidad *ilimitada* de recursos.
- **Costos.** Pay-as-you-go. Controles de gasto.

Resistencia al cambio

- Factores **políticos**:
  - Locación y jurisdicción de datos,
  - Incapacidad de influir sobre decisiones de HW.
- Factores **técnicos**:
  - Costos p/ migraciones,
  - Exposición de datos sensibles,

PaaS

Tener una plataforma para desarrollar. Viene con:

- **Infraestructura.** Ya está en la plataforma, todo como IaaS.
- **Plataforma de Desarrollo.** SOs, librerías especiales, middlewares.
- **Persistencia.** Bases de datos, archivos blobs, colas de mensajes.
- **Monitoreo.**
- **Escalabilidad.** Balanceo, elasticidad.

Google AppEngine

Plataforma basada en buenas prácticas (forzadas).

Buenas prácticas

- Sistemas **granulares**.
- Escalamiento **horizontal**.
- **Requests breves**, los largos son encolables.
- Independencia de SO / HW.

Servicios integrados

- Cache.
- Colas de mensajes.
- Elasticidad.
- Versionado.
- Herramientas de log, debugging, monitoreo.

- Modelos No-Relacionales (Datastore, BigTable).

Microservicios

- **Aplicaciones**:
  - **Servicios**, pueden o no hablar entre ellos.
  - C/ servicio tiene capa de:
    - Memcached.
    - Datastore.
    - Task Queues.

Componentes

- **Servicios:** módulos.
- **Instancias (AppServers):** servidores de backend.
  - Unidad de procesamiento.
  - **Dinámicas:** creadas x requests.
  - **Residentes:** escaladas manualmente.
  - Depende de:
    - Alguien que se encargue de bufferear cosas,
    - Que no sea importante el estado dentro de las instancias.
  - Evidentemente BUENO tener **stateless**.
  - Si mi req tarda mucho, tiran la instancia.

Arquitectura

- Primer Data Center.
  - Google **front end**.
    - 1st level validations.
  - **Edge Cache.**
- Segundo Data Center.
  - Tiene a la App en sí.
  - **AppEngine front end.**
    - CDN.
    - App Servers.
      - Instancias.

Comunicación interna

- **Push Queues:** cuando algo llega, la **cola es algo activo**.
  - Puede invocar cosas.
  - La cola creaba los handlers si no estaban.
  - Si habían pocos workers, la cola creaba más.
  - La cola hacía el balanceo.
  - El mensaje debe tener una URL (p/ atenderlo).
- **Pull Queues:** las de siempre.
  - Procesamientos largos.
  - Payload, Etiqueta.
- Se comparte **Datastore** y **Memcached**.
  - Todo el resto es **serverless**.
- Mensajes en modo **leasing** (ACK de Rabbit).

Almacenamiento

- Datastore.

- Basado en BigTable.
- Clave-Valores.
- Atomicidad? **Particionado**.
  - Guardar las cosas que component mi transaccion todas juntas.
  - Transacciones con poco delay.
  - Localidad espacial p/ acceso.