INCLUIT
empowering your business

Clase N°4 - Training Node.js

DDD - Domain Driven Design

DDD es un enfoque orientado al diseño de software en donde predomina un modelado del código lo más cerca posible al dominio del negocio. Este resultado en código puede ser fácilmente comprendido por el negocio y evolucionado a medida que las necesidades del negocio cambian. Dicha metodología cubre el ciclo de vida completo del software y busca atacar problemas comunes desde una perspectiva diferente.

Cabe aclarar, que no es necesario aplicar todos los patrones de DDD al proyecto. Una opción, es identificar conceptos del diseño estratégico e ir posteriormente incorporando esta metodología gradualmente.

DDD - Diseño estratégico

La importancia de cuándo diseñamos soluciones, separamos el problema de la solución. Un error común al diseñar software, lo más común es centrarnos en detalles técnicos primeramente como qué lenguaje o tecnología utilizar.

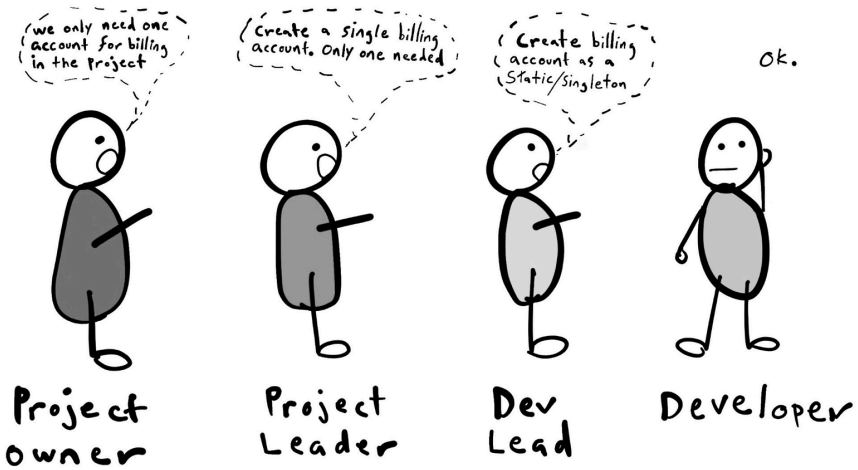
Domain

Es el problema del negocio. Identificar también los subdominios. Este lo podemos expresar en diferentes formas:

- Diagramas
- Documentación

Investigar lo que la compañía hace. ¿Qué servicios se les provee a los clientes? ¿En qué se diferencia esta de la competencia?. Reconocer el core del negocio.

Lenguaje Ubicuo



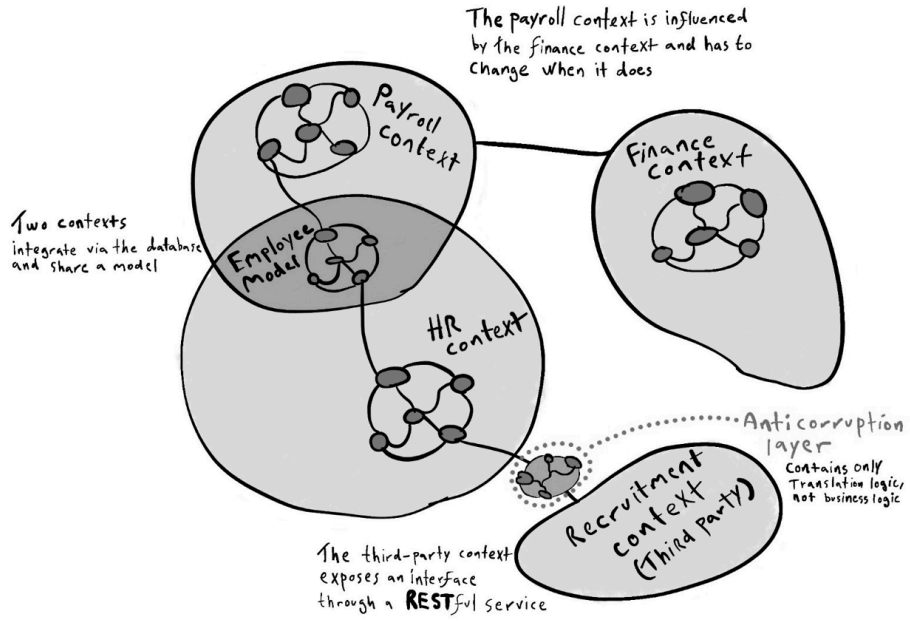
Important concepts get lost and misinterpreted when there isn't a joint effort on collaboration between the two sides. Proper ubiquitous language will help the two models to stay aligned. This is where the added importance of a feedback loop takes place.

Bounded Context

Son los que nos permiten dividir el domino en subdominios.
Separar en trozos más pequeños el problema. Principal beneficio es desacoplar subdominios.

Context Mapping

Los subdominios no están aislados, tienen dependencias entre ellos. Como se relacionan e interactúan los subdominios.



An example of a context map

DDD - Diseño táctico

Engloba todo lo que es código

- Servicios
- Entidades
- Objetos de valor
- Agregados
- Repositorios
- Eventos de dominio
- Factorias
- Arquitectura por capas

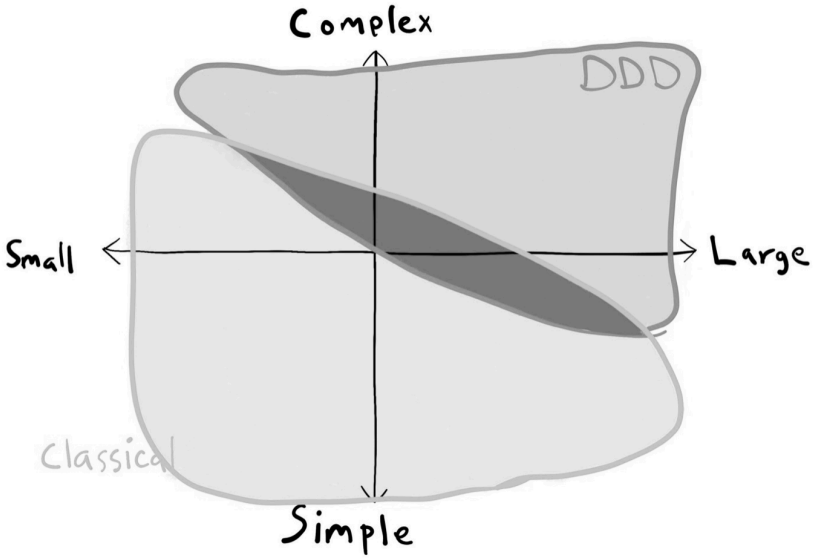
DDD en práctica

Event Storming: Técnica de grupo de trabajo para modelar y descubrir modelos con la finalidad de construir un lenguaje ubicuo que posteriormente será utilizado para las precedentes etapas. Lo recomendable aquí, es que participe un grupo diverso de personas del negocio: ingenieros, expertos del negocio, product owners, testers, diseñadores, personal de soporte, etc.

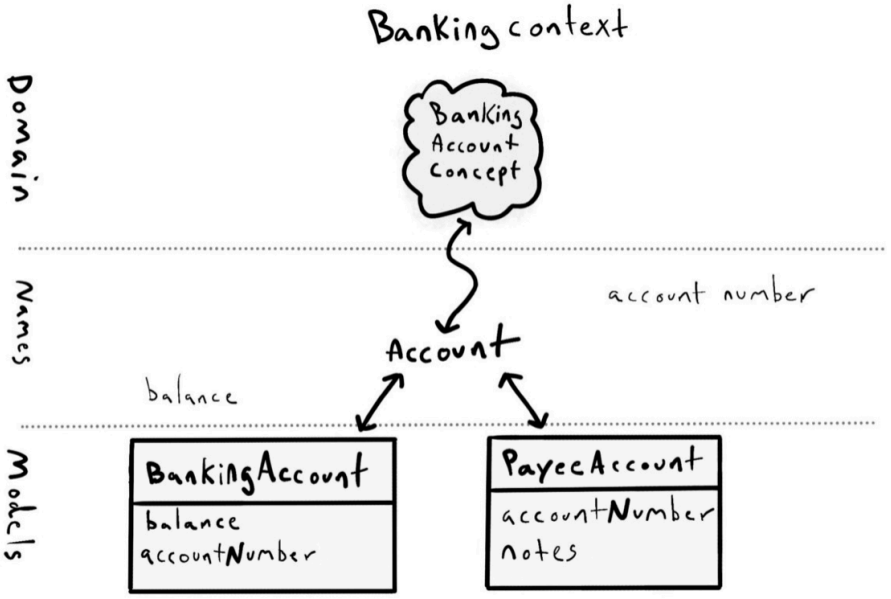
Beneficios de DDD

Las cualidades que destacamos

- Portabilidad: Capacidad que tiene de ser portado de una infraestructura a otra
- Escalabilidad



Matrix of when to use Domain Driven Design principles



Libros

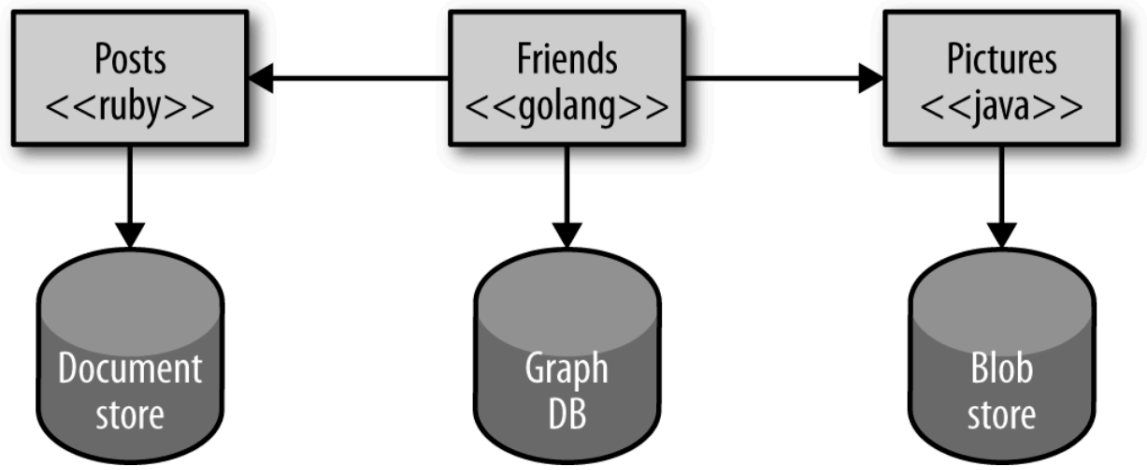
- Domain-Driven Design: Tackling Complexity in the Heart of Software
- Implementing Domain-Driven Design
- Domain-Driven Design Distilled
- Domain Driven Design Quickly

Links

- https://github.com/talyssonoc/node-api-boilerplate/wiki/Folder-structure
- https://github.com/node-ts/ddd

Microservicios

Esta es una arquitectura que permite desarrollar aplicaciones de software como un set modular de servicios. Piense en estos como si fuesen una persona que ejecuta una única tarea en especifico. Podríamos reforzar el concepto haciendo uso de la definición del principio de responsabilidad única que establece que una clase, componente o microservicios debe ser responsable de una sola cosa.



Beneficios

Su principal ventaja surge del hecho de que estos nos brindan mayor libertad y poder de respuesta ante cambios inevitables que surjan.

Heterogeneidad

Cada microservicio podría estar implementado en diferentes lenguajes y con la posibilidad de elegir la herramienta correcta para cada trabajo.

Resiliencia

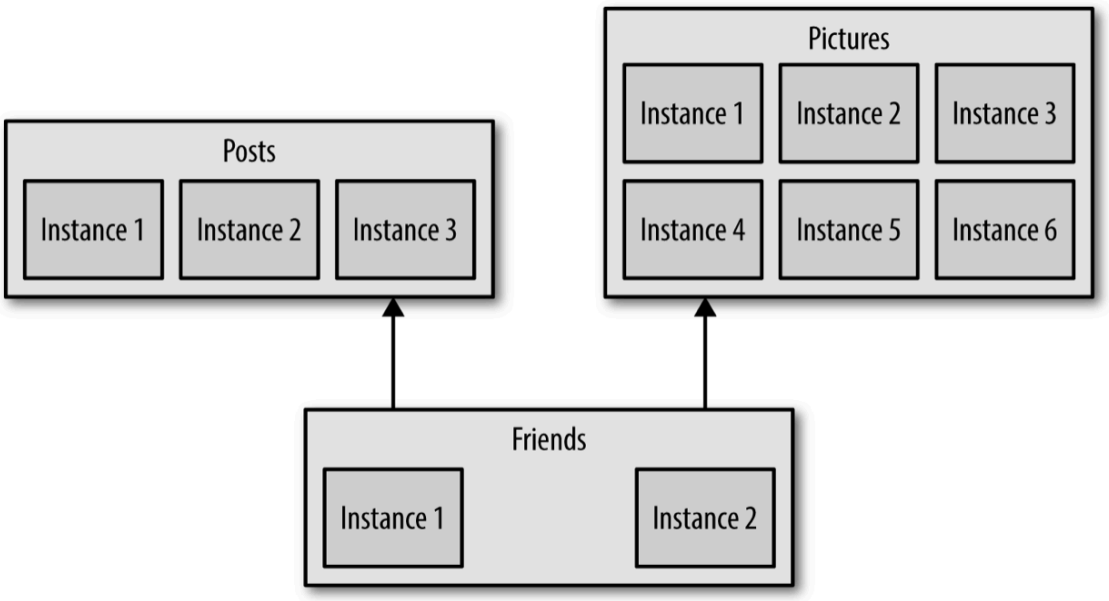
En el caso que uno de los componentes falle, no repercutirá en el sistema completo. Es posible aislar el problema del resto del sistema y trabajar sobre este sin afectar al resto.

Escalabilidad

Al trabajar con microservicios, tenemos la posibilidad de poder escalar solamente aquellos servicios que lo requieran.

Reusabilidad

Con los microservicios, nosotros permitimos que nuestra funcionalidad pueda ser consumida en diferentes formas para diferentes propósitos.

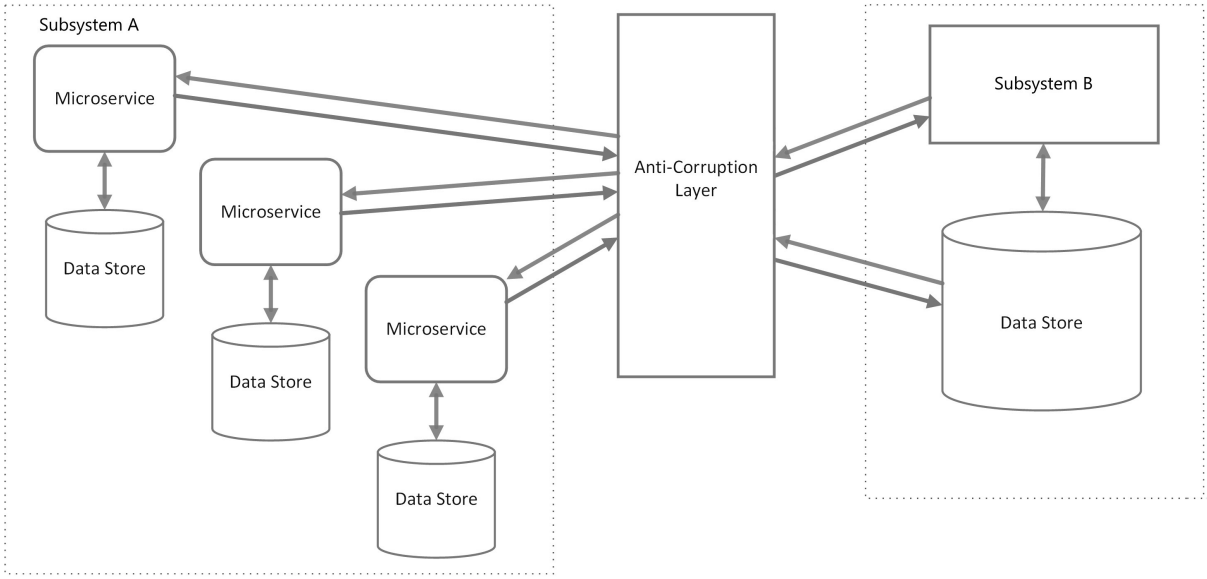


Libros

- TypeScript Microservices

ACL - Anticorruption Layer

Este patrón consiste en adicionar una capa intermedia entre diferentes subsistemas que no comparten una misma semántica. Esta capa se encarga de traducir las peticiones que un subsistema efectúa a otro. De esta forma, se asegura que el diseño de la aplicación no está limitada por el subsistema externo.



Problemas y consideraciones

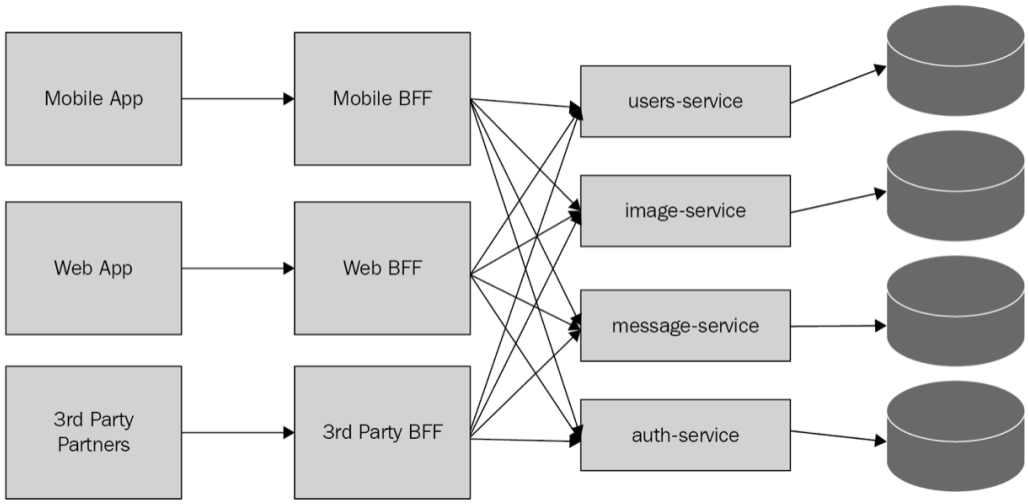
- La capa de anticorrupción podría adicionar latencia en la comunicación entre ambos sistemas
- La capa de anticorrupción adiciona un servicio adicional que debe de ser administrado y mantenido
- Considerar como la capa de anticorrupción debería de escalar
- La consistencia de los datos debe de ser mantenida y monitoreada

Links

- Anti-Corruption Layer Pattern

BFF - Backend for Frontend

Este es un patrón de arquitectura que introduce una capa entre un cliente y los servicios de backend con los que interactúa. Cada interfaz puede definir sus necesidades y requerimientos únicos sin necesidad de afectar en las implementaciones del frontend. Mediante este, evitamos tener un único backend para múltiples interfaces.

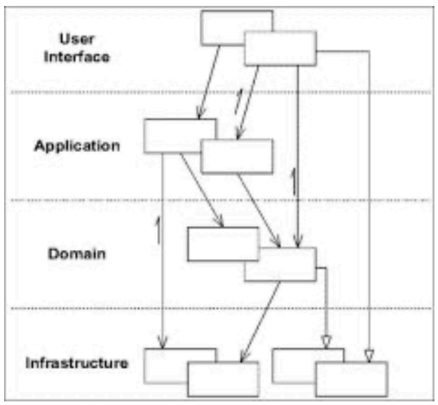


En un mundo donde proliferan diferentes dispositivos con conectividad a Internet, cada servicio debería de adecuarse a este sobre qué información retornar.

Libros

- Hands-On RESTful API Design Patterns and Best Practices

Layered Architecture



User Interface Layer	Accepts user commands and presents information back to the user
Application Layer	Manages transactions, translates DTOs, coordinates application activities, creates and accesses domain objects
Domain Layer	Contains the state and behavior of the domain
Infrastructure Layer	Supports all other layers, includes repositories, adapters, frameworks etc.

Una forma natural de diseñar software es siguiendo la naturaleza de la organización del negocio.

“Las organizaciones dedicadas al diseño de sistemas están abocadas a producir diseños que son copias de las estructuras de comunicación de dichas organizaciones” - Melvin Conway.

Para el desarrollo de los microservicios en Naranja, se utiliza una arquitectura en capas, cada una con la responsabilidad que le corresponde.

La topología de esta arquitectura está compuesta por las siguientes capas

- Aplicación (Application): Responsable de coordinar todos los elementos de la aplicación. No contiene lógica de negocio ni mantiene el estado de los objetos de negocio. Es responsable de mantener el estado de la aplicación y del flujo de esta.
- Dominio (Domain): Contiene la información sobre el dominio. Es el núcleo de la parte de la aplicación que contiene las reglas de negocio. Es responsable de mantener el estado de los objetos de negocio. (La persistencia de estos objetos se delega en la capa de infraestructura.
- Infraestructura (Infrastructure): Esta capa es la capa de soporte para el resto de capas. Provee la comunicación entre las otras capas, implementa la persistencia de los objetos de negocio y las librerías de soporte para las otras capas (Interface, Comunicación, Almacenamiento, etc…).

Libros

- Fundamentals of Software Architecture

Arquitecturas Serverless

Una arquitectura sin servidor es una manera de crear y ejecutar aplicaciones y servicios sin tener que administrar infraestructura. Ya no tiene que aprovisionar, escalar ni mantener servidores para ejecutar sus aplicaciones, bases de datos y sistemas de almacenamiento.

¿Qué beneficios tiene usar arquitecturas sin servidor?

Mediante el uso de una arquitectura sin servidor, los desarrolladores se pueden enfocar en el producto principal en lugar de preocuparse por la administración y el funcionamiento de los servidores, o los tiempos de ejecución, tanto en la nube como en las instalaciones. Gracias a esta reducción de gastos, los desarrolladores pueden emplear tiempo y energía en desarrollar productos increíbles que sean de confianza y se puedan escalar.

Otros beneficios

- Escalabilidad
- Alta disponibilidad
- Baja latencia
- Pago por uso

Desventajas

- Vendor Lock-In
- Cold Start
- Tiempo de computo
- Tamaño de la aplicación

Principales proveedores Cloud que ofrecen servicio bajo esta arquitectura

- Amazon Web Services
- Google Cloud Platform
- Microsoft Azure
- IBM Cloud

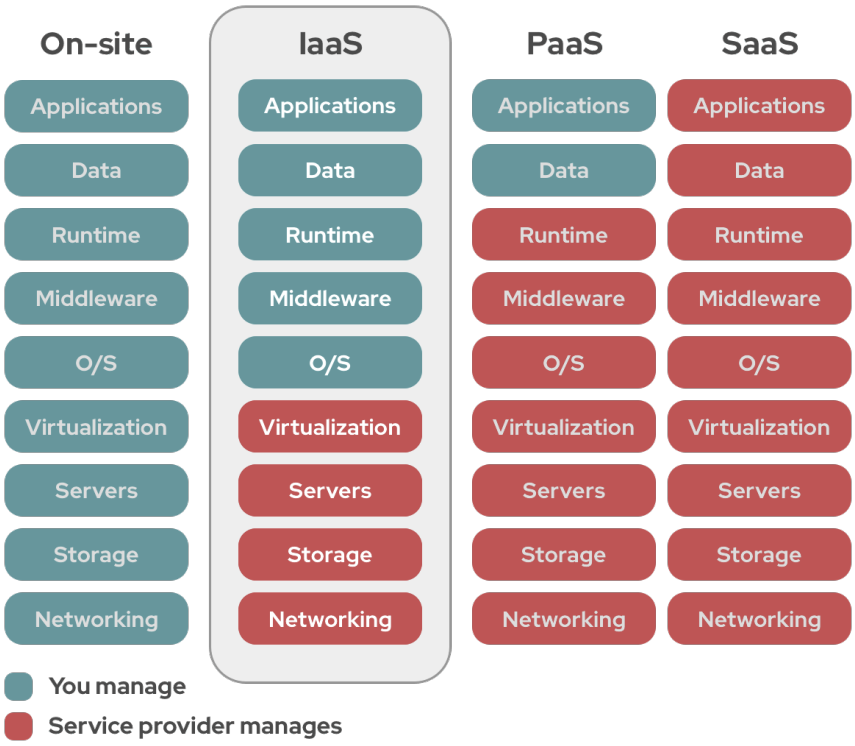
Escenarios recomendados de usos

- Tareas que puedan ser separadas en unidades independientes de trabajo
- Arquitecturas orientas a eventos

Interoperabilidad

Mediante Serverless Framework podremos atacar uno de los puntos débiles de esta arquitectura. De esta forma, no estamos atados a un proveedor cloud en especifico.

Modelos de informática en la nube



FaaS

Functions as a service

Las funciones son ejecutadas en instancias que almacenan de forma temporal el código y destruidas posteriormente ejecutado el código. Este concepto, nos ofrece elasticidad para poder soportar grandes cargas de tráfico. Dichas funciones son ejecutadas típicamente mediante eventos o peticiones HTTP. Cuando esta completa el procesamiento, puede retornar un valor o pasar el resultado a otra

función parte del flujo. Cada función debería de seguir el principio de responsabilidad única e idempotencia. A su vez, las funciones pueden ser asíncronas o asíncronas dependiendo de la necesidad de la tarea.

IaaS

Infrastructure as a service

La infraestructura como servicio (IaaS), también conocida como servicios de infraestructura en la nube, es una forma de cloud computing que ofrece a los usuarios finales una infraestructura de IT a través de Internet.

El proveedor de IaaS brinda la virtualización, el almacenamiento, la red y los servidores. De esta manera, el usuario no necesita tener un centro de datos on-premise ni debe preocuparse por actualizar o mantener físicamente estos elementos.

Este es el punto de entrada a todo organización que desee migrar a la nube fácil y rápidamente.

¿Cuándo usar IaaS?

El caso de uso más común para IaaS es cuando una organización ya tiene su aplicación, o tiene los recursos internos para desarrollarla, y simplemente necesita infraestructura para alojarla.

En AWS, la IaaS principal para alojar máquinas virtuales es Amazon EC2. AWS ofrece la gama más amplia de tipos de instancias entre los principales proveedores de la nube, con una gama de máquinas virtuales de propósito general, así como instancias adaptadas a la memoria, el cómputo y el almacenamiento.

Links

- AWS Types of Cloud Computing

Libros

- Software Architect's Handbook

PaaS

Platform as a service

La plataforma como servicio o PaaS es un conjunto de servicios basados en la nube que permite crear aplicaciones sin necesidad de preocuparse por la configuración y el mantenimiento de servidores, parches, actualizaciones y autenticaciones, entre muchas otras tareas: los usuarios pueden centrarse en crear la mejor experiencia de usuario posible.

SaaS

Software as a service

El software como servicio (SaaS) permite a los usuarios conectarse a aplicaciones basadas en la nube a través de Internet y usarlas.

¿Cómo funciona?

Por lo general, un proveedor de servicios de nube (como AWS, Azure o IBM Cloud) gestiona el entorno de nube en el cual se aloja el software. Además, el proveedor de SaaS se encarga de las actualizaciones del software, las correcciones de errores y demás tareas de mantenimiento general de las aplicaciones.

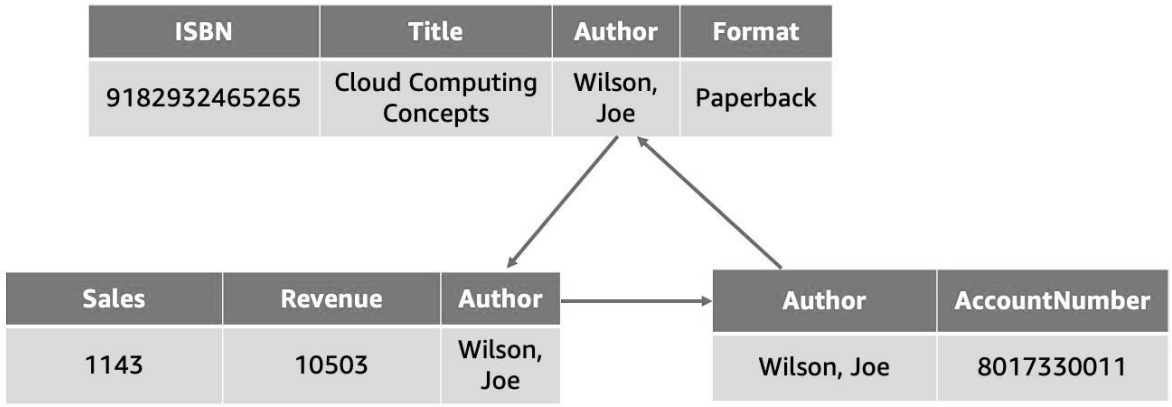
Ejemplos de soluciones SaaS

- Software de gestión de contenido de WordPress
- Servicio de mensajería de Slack
- Servicio de ofimatica Microsoft Office 365
- Servicio de almacenamiento de archivos de Dropbox

Base de datos

Relacionales

Una base de datos relaciónale organiza datos en tablas. Datos en una tabla pueden ser vinculados a otras tablas. Una tabla almacena datos en filas y columnas. Una fila, comúnmente llamada registro, contiene toda la información especifica de una entrada. Las columnas describen atributos de este registro.



RDBMS - Relational Database Management System

Este es un sistema que nos permite crear, actualizar y administrar una base de datos relacional.

- MySQL
- PostgreSQL
- Oracle
- SQL Server
- Amazon Aurora

Tipos de base de datos no relacionales

- Key/value
- Document databases
- Graph databases
- Search databases
- In-memory databases

SQL en comparación con Terminología NoSQL

La siguiente tabla compara la terminología utilizada por las bases de datos NoSQL seleccionadas con la terminología utilizada por las bases de datos SQL.

SQL	MongoDB	DynamoDB	Cassandra	Couchbase
Tabla	Conjunto	Tabla	Tabla	Bucket de datos
Fila	Documento	Elemento	Fila	Documento
Columna	Campo	Atributo	Columna	Campo
Clave principal	ObjectId	Clave principal	Clave principal	ID del documento
Índice	Índice	Índice secundario	Índice	Índice

Links

- AWS - Base de datos NoSQL

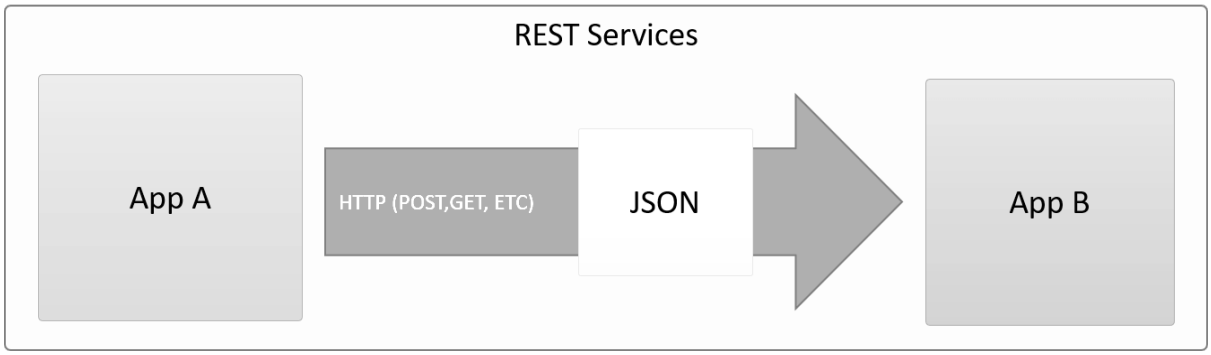
APIs

Una API es la interfaz que permite el intercambio de información entre dos componentes de software independientes.

- API internas o privadas
- API externas o abiertas

REST - Representational State Transfer

REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON



RFC 2616

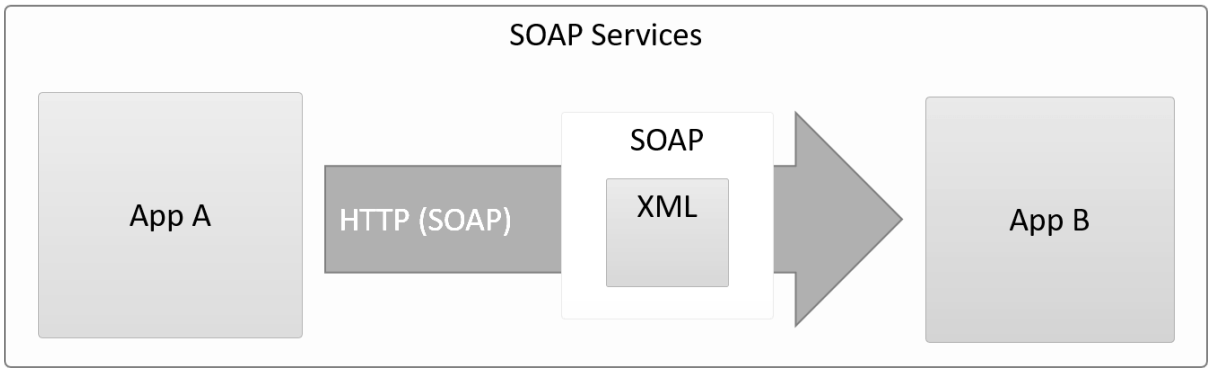
https://tools.ietf.org/html/rfc2616

RESTful

Hace referencia a un servicio web que implementa la arquitectura REST

SOAP - Simple Object Access Protocol

Los servicios SOAP o mejor conocimos simplemente como Web Services, son servicios que basan su comunicación bajo el protocolo SOAP. Los servicios SOAP funcionan por lo general por el protocolo HTTP que es lo más común cuando invocamos un Web Services, sin embargo, SOAP no está limitado a este protocolo, si no que puede ser enviado por FTP, POP3, TCP, Colas de mensajería (JMS, MQ, etc). Pero como comentaba, HTTP es el protocolo principal.



Códigos de respuestas HTTP

Los códigos de estado de respuesta HTTP el estado de una solicitud HTTP específica. Las respuestas se agrupan en cinco clases

- Respuestas informativas (100 – 199)
- Respuestas satisfactorias (200 – 299)
- Redirecciones (300 – 399)
- Errores de los clientes (400 – 499)
- Errores de los servidores (500 – 599)

Documentación

- RFC 7231

Links

- https://http.cat
- https://httpstat.us

Contratos de APIs en Swagger con OpenAPI 2 y 3

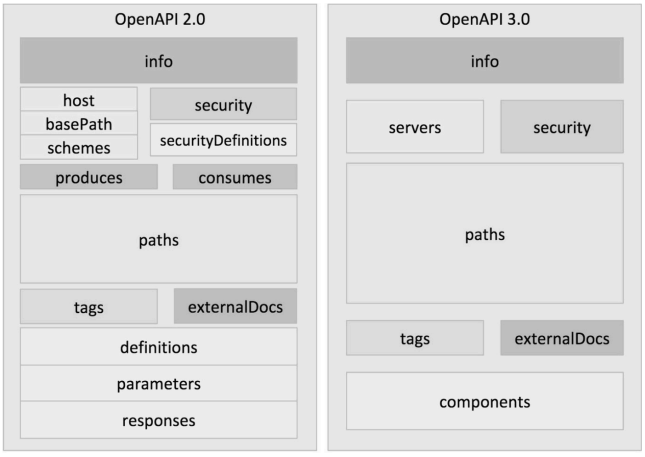
Cuando hablamos de Swagger nos referimos a una serie de reglas, especificaciones y herramientas que nos ayudan a documentar nuestras APIs. De esta manera, podemos realizar documentación que sea realmente útil para las personas que la necesitan. Swagger nos ayuda a crear documentación que todo el mundo entienda.

¿Qué es OpenAPI?

Es un estándar para definir contratos de API totalmente agnóstico en cuanto al lenguaje se refiere, y que establece un marco común sobre cómo construir y mantener las APIs.

Estructura de una interfaz basada en Swagger

Las secciones que tenemos que tener en cuenta para documentar una API en formato Swagger son



Links

- Audit Swagger Contract
- Swagger Official Page