Microservices

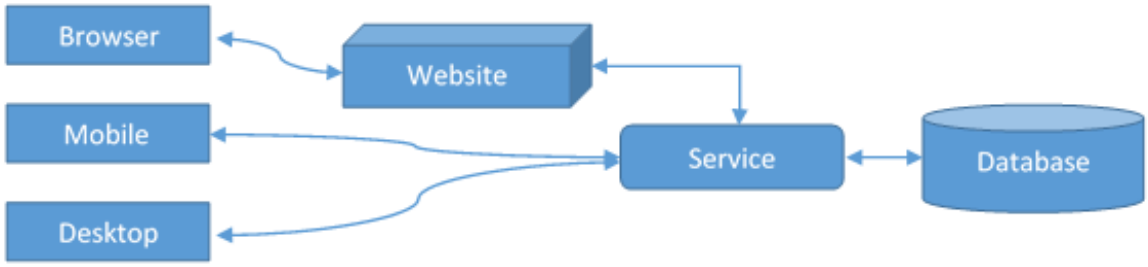
acarlstein.com/

Posted by Alejandro G. Carlstein Ramos Mejia on November 28, 2016 January 18, 2017 About Programming / Architecture / Microservices

Microservices: Design Principles Introduction

What is a Service?

A service is a piece of software that provides functionality to another pieces of software.



- Provide reusability of functionality.
- A service can provide functionality to any application.
  Such a web server, mobile, or desktop application
- A service can be use by another service
- Service-oriented Architecture (SOA)
  - Instead of using package modules within each client application, we have a service which provide the same functionality to different client applications.
  - Allow for new services and application to use the same functionality, reusing it, in the future.
  - Allows for scale up our software when demand increasing.
    i.e. Load balancer which have multiple copies of the same service on multiple servers. When demand increases, we increase the instance of the service running across servers.
- As long as the signature of a service, contract and interface, doesn't change when the service changes, we can upgrade our service without having to update our clients.
- A service is stateless. When a request comes in, that instance of the service does not have to remember the previous request from that specific client.

Microservice Architecture

Traditional SOA resulted in monolithic services. You needed to know how to size a service. The micro sized services provide an efficiently, flexible, and high performance applications.

A microservice architecture is an application which is powered by multiple microservices, in which each microservice provides a set of functions (or related functions) to a specific part of the application.

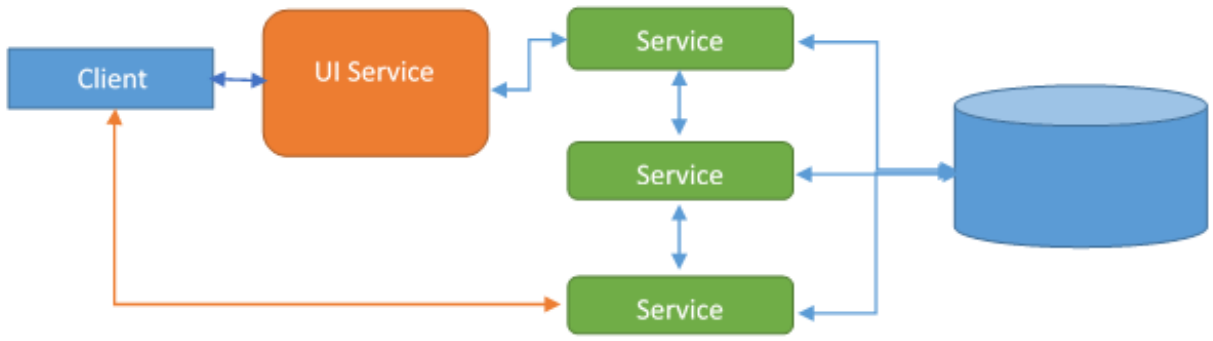
A microservice have a single focus. It does one thing only and it does it well.

Microservice architecture are used in lightweight and quick communication mechanisms (such as REST) between client and services; as well as, service to service.

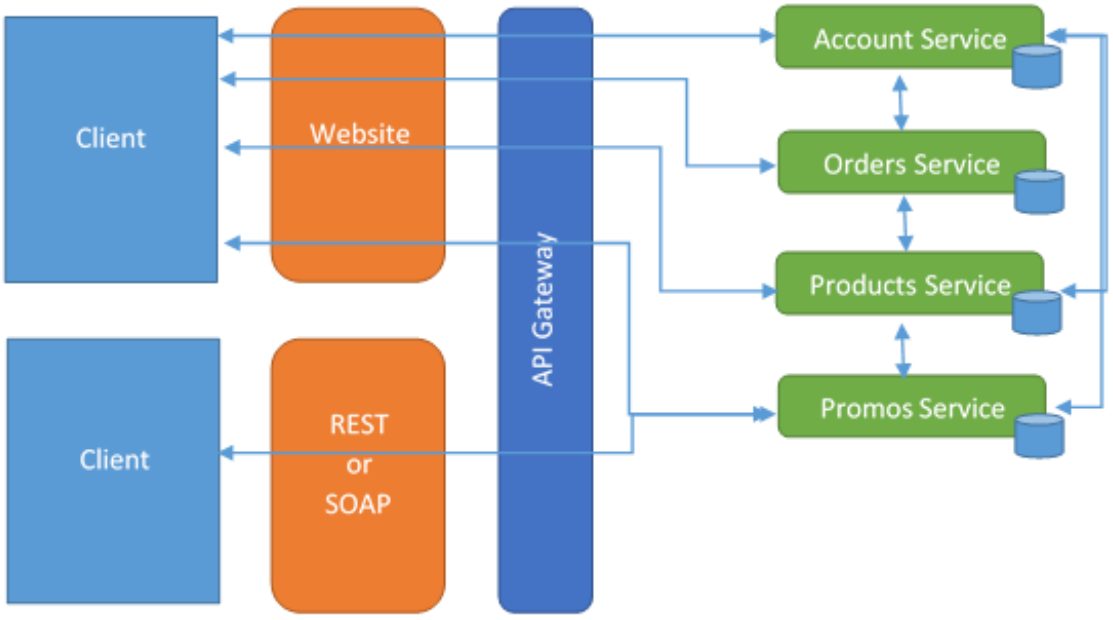
A microservice needs technology agnostic API which mean to use an open communication so it doesn't dictate the technology that the client needs to use.

At difference to a monolithic service where there is a central database used to share data between applications and services, in microservices architecture, each microservice has its own data storage.

Monolithic Service Example



Microservice Example



A microservice is independently changeable. This means that we can upgrade or fix a microservice without forcing any changes to the clients or services.

A microservice needs to be independently deployable. You should be able to deploy a microservice without having to deploy anything else.

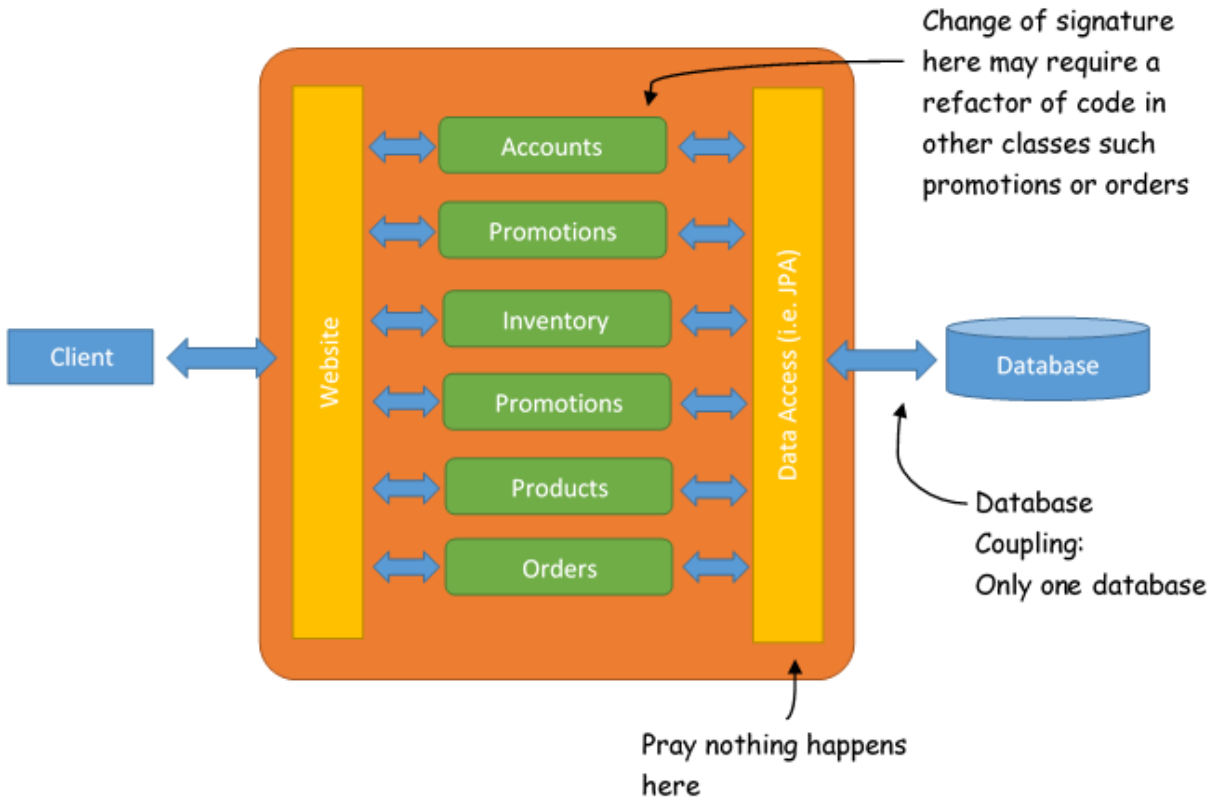
⚠ Note: You may need a centralized tooling for management of microservices.

The Monolithic System

Typical Enterprise Application:

- Large website with all modules packaged in together into one package.
- A service which talks to a website in which the service itself is a large service with all modules packaged together as one executable.
- As you add features and stuff, your application keeps growing.
- There is no restriction in size; as well as, there is no division.
- There is always one package which basically contains everything.
- Large code base which makes it harder and time consuming to add new functionality.
- Code can become intertwined making difficult to make changes without creating side effects to other parts of the systems.
- Testing can be challenging.
- Features may be in deep in the system that they can be utilized for external use.
- Commonly stuck with one technology stack which may bring restriction such as implementing some new technology that may be different to the current
- technology stack.
- The system is less competitive because it restrict the adoption of new technology.
- Large package with high levels of coupling which means that changes may produce a ripple effect (side effects). This coupling happens in all levels such as modules, services, and objects.
- A failure in one part of the system may affect the whole system.
- Duplication of the whole may require scaling.
- The system may requires a long time to compile. The larger the system gets, the longer the compilation time required.
- Any changes, no matter how minor, may requires a complete rebuild.

Monolithic System Example



Emergence of Microservices

- Need to respond to change quickly.
- Can split a large system into parts that can be upgraded and enhanced individually
- The entire system will not break if one part breaks.

- It allows for business domain-driven design.
- It takes advantage of automated test tools.
- Since transactions are distributed, each transaction will be processed by multiple services before it's completed.
    - The integration between those services requires to be tested.
    - Instead of testing these microservices manually, we can automated the test.
- Release and deployment of microservices may become complex; however, there are tools available to easier the work.
- We can host microservices using on-demand technology such using virtual machines to host our microservice.
    - Physical servers are no longer required in order to deploy our software.
    - On-demand hosting is simpler in these days with cloud services available.
    - We can clone these virtual machines.
- We can move our microservice from one technology stack to another technology stack.
- It allows for asynchronous communication technology.
    The distributed transaction do not have to wait for other services to complete their task.
- We have simpler server side and client side technology, as well as many open communication protocols available which allows communication between different technology stacks.

Key Benefits

- Shorter development times
- The split up of the system allows to work individually in one part or assign different parts to different people and/or teams.
- Due the size and the concept of single focus, an individual or team has less to worry about in terms of scope.
- They only need to focus on their scope and not the whole system
    As long as the contracts (interfaces) between the services remain.
- Developers can rework, change and deploy individual components without affecting the system (or need to redeploy everything) since the services are loosely coupled.
- Deployment is more reliable and faster.
- Allows for shorter development times, reliable and faster deployment; therefore, frequent updates.
    Frequent updates provide a competitive edge.
- It allows us to decouple changeable parts.
- Increases security since each microservice has its own database and its own security mechanism.
    - Data is distributed which makes the data more secure.
    - The monolithic system may have one database. By hacking that one system, you can gain access to the data.
- Quicker to identify which service is having the problem.

- Highly scalable and better performance.
    Scale part individually instead of the whole system.
- Easier to change ownership of each microservice.
- Each microservice have their own database and code base.
- Enables distributed teams.