Introduction to Spring Cloud

LXFT LISTED NYSE

Prepared by Orkhan Gasimov
Feb. 2017

LUXOFT

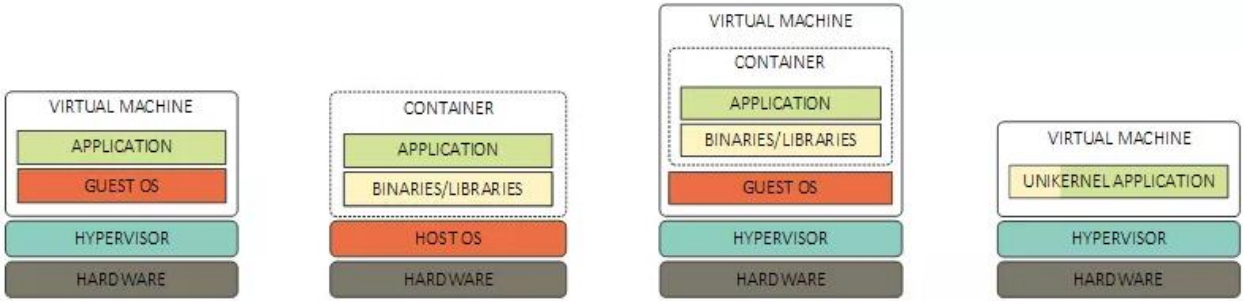
Spring Cloud

Spring Cloud

- Spring Cloud is a set of tools to build distributed systems with common patterns.
  - E.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, one-time tokens, global locks, leadership election, distributed sessions, cluster state.
- Spring Cloud focuses on providing out of box experience for typical use cases and extensibility mechanisms.

Spring Cloud

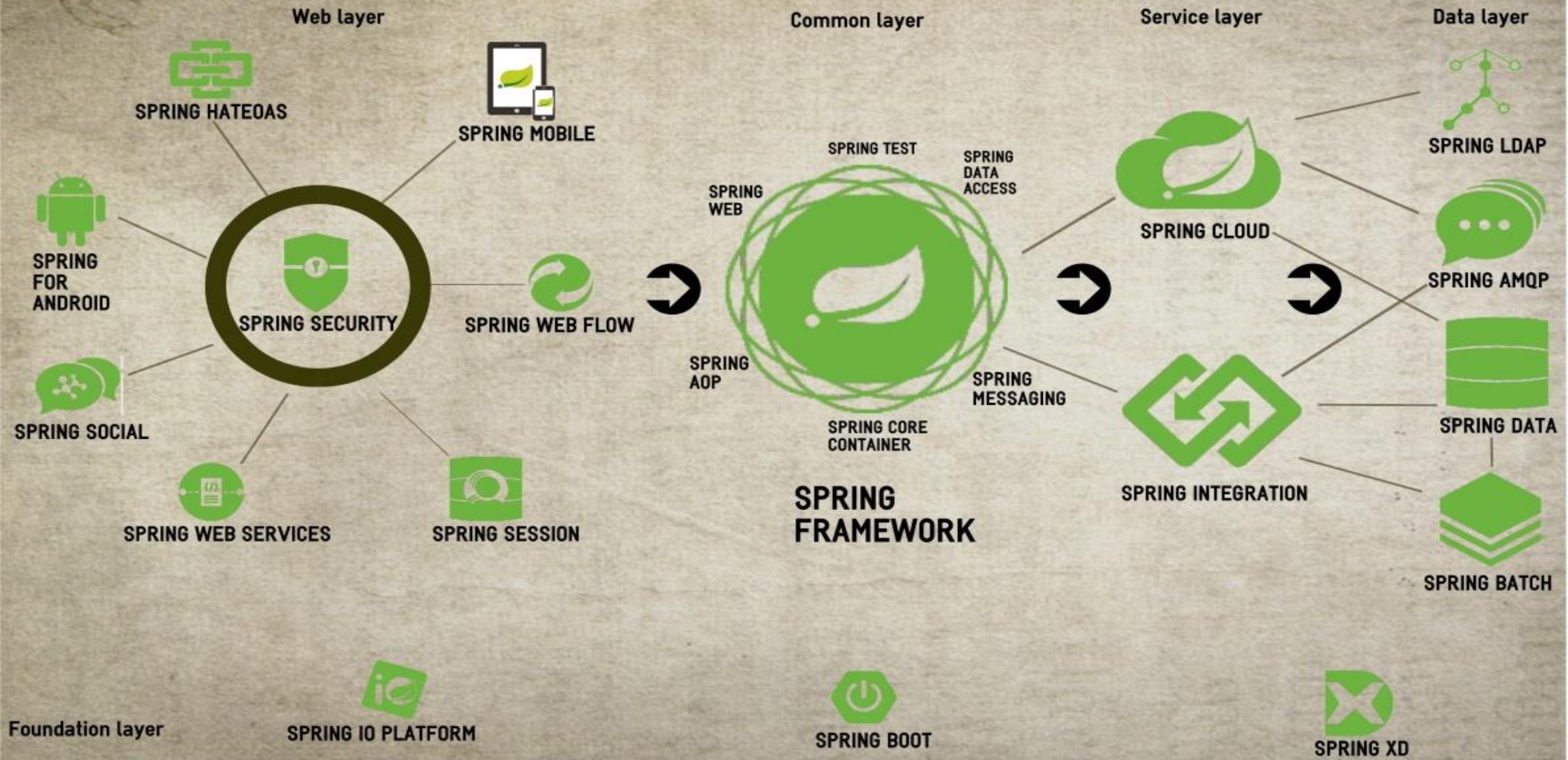
- Spring Cloud helps developers to quickly stand up services and applications that implement boilerplate patterns for coordination of distributed systems.
  - They will work well in any distributed environment, including the developer's own laptop, bare metal data centers, and managed platforms.



Spring Cloud

- Spring Cloud is based on Spring Boot and takes a very declarative approach.
  - Often you get a lot of features with just a classpath change and/or an annotation.

Spring Framework Ecosystem



Architectural Pattern & Principles

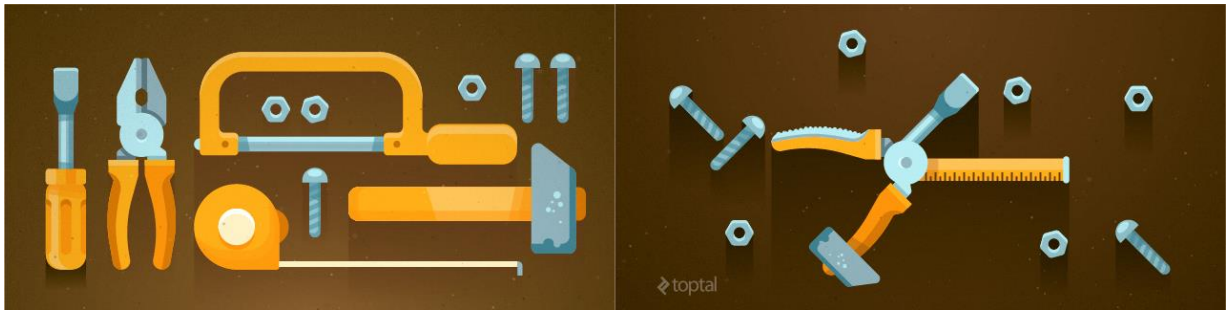
Architectural Patterns & Principles

- Some of basic architectural patterns and principles followed by projects built with Spring Cloud are:
  - Single Responsibility Principle
  - Share-Nothing Architecture
  - Asynchronous Message-Passing
  - Microservice Architecture
  - Service Discovery Pattern

Single Responsibility Principle

Single Responsibility Principle

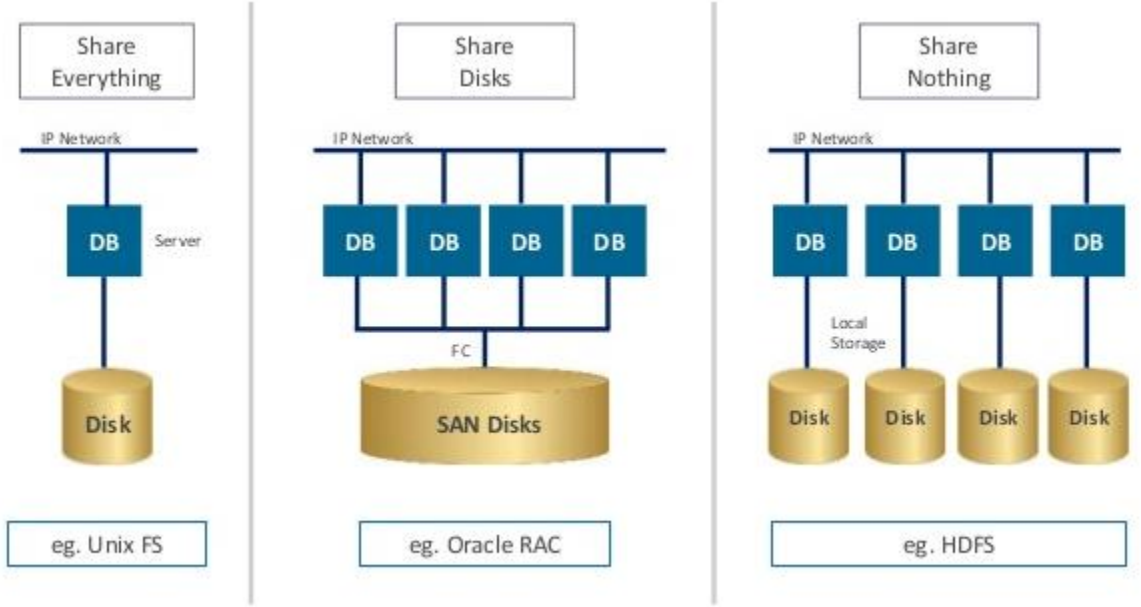
- A single service should have responsibility over a single part of the functionality provided by the software and that responsibility should be entirely encapsulated by the service.



- Just because you can, doesn't mean you should!

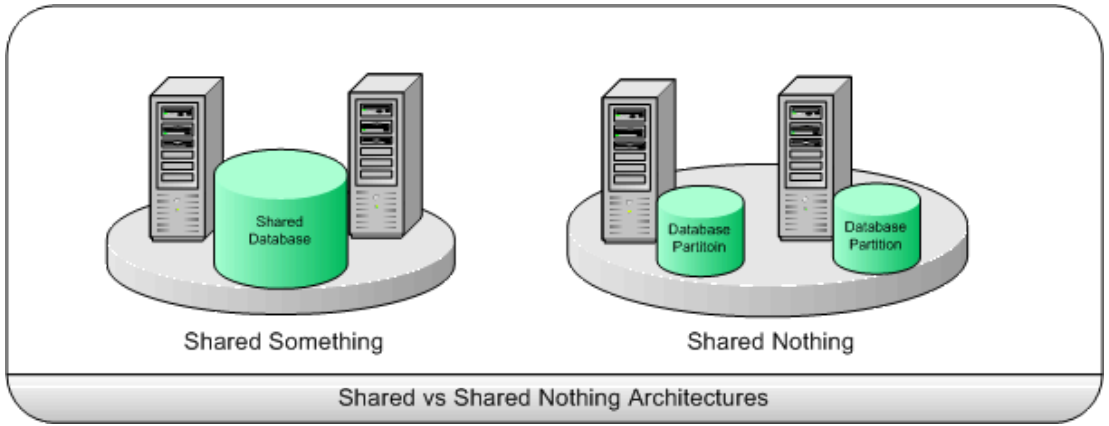
Share-Nothing Architecture

Share-Nothing Architecture



Share-Nothing Architecture

- A shared nothing architecture is a distributed computing architecture in which each node is independent and self-sufficient, and there is no single point of contention across the system.
  - More specifically, none of the nodes share memory or disk storage.



Share-Nothing Architecture

- The advantages of share-nothing architecture versus a controller-based architecture include:
  - eliminating any single point of failure;
  - allowing self-healing capabilities;
  - providing an advantage with offering non-disruptive upgrade.
- Share-nothing is popular for web development because of its scalability.

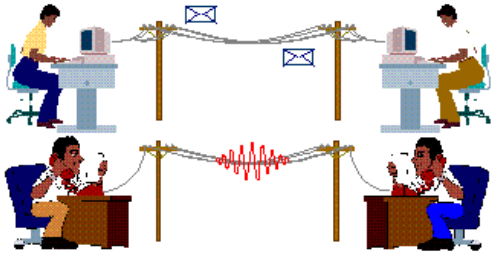
Share-Nothing Architecture

- There is some doubt about whether a web application with many independent web nodes but a single, shared database (clustered or otherwise) should be counted as share-nothing.
- One of the approaches to achieve share-nothing architecture for stateful applications is the use of a data grid, also known as distributed caching.
  - However, this still leaves the centralized database as a single point of failure.

Asynchronous Message-Passing

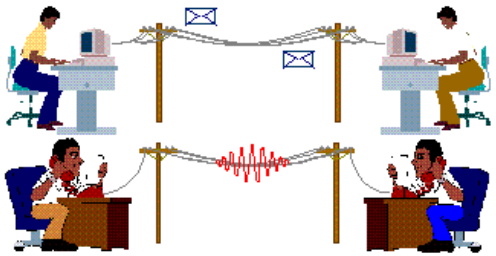
Asynchronous Message-Passing

- Asynchronous Message-Passing is similar to e-mail.
- In this pattern sender and receiver are decoupled.
- Sender knows only the address of receiver.
  - Usually this is the address of the mailbox that receiver checks for new messages.
- Sender reports his address with the message he sends, so that receiver can reply if necessary.
  - Sender, Receiver, Mailbox and Message are decoupled components.
  - They work independently and may have various swappable implementations.



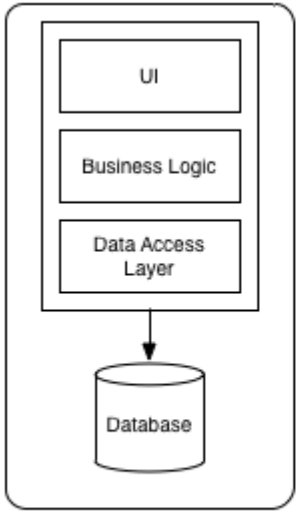
Asynchronous Message-Passing

- Asynchronous message passing leads to non-blocking API.
- Pattern implementations gain higher modifiability and scalability.
- For instance:
  - A sequential mailbox implementation for Receiver can be swapped to a prioritized or another one depending on requirements.
  - Sender and Receiver are loosely coupled.
  - Both could be scaled up horizontally.

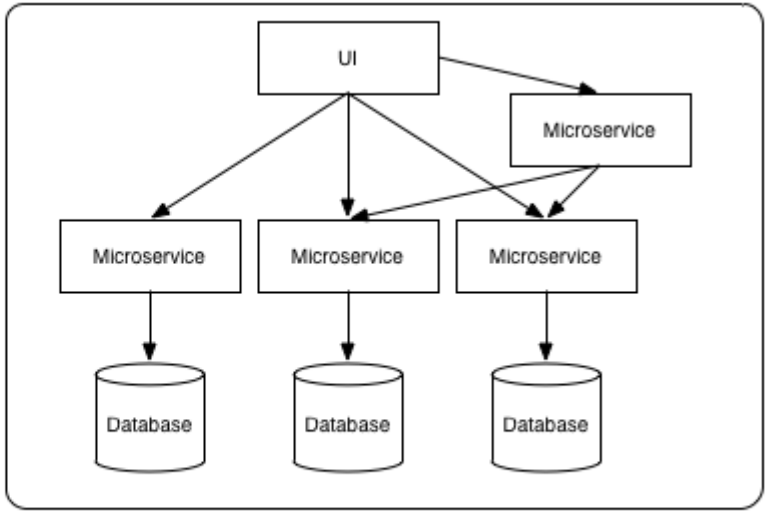


Microservice Architecture

Microservice Architecture



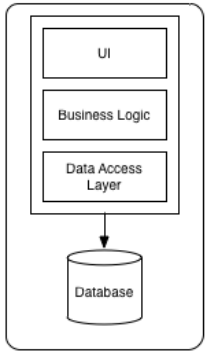
Monolithic Architecture



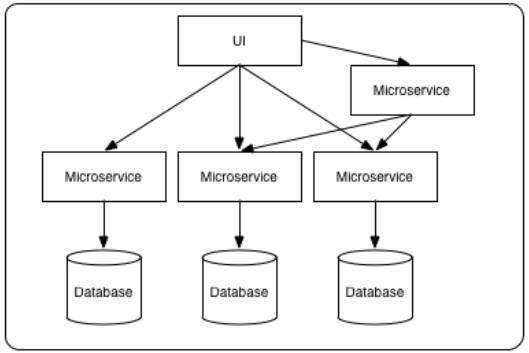
Microservices Architecture

Microservice Architecture

- Essentially, microservice architecture is a method of developing software applications as a suite of independently deployable, small, modular services in which each service runs a unique process and communicates through a well-defined, lightweight mechanism to serve a business goal.



Monolithic Architecture



Microservices Architecture

Microservice Architecture

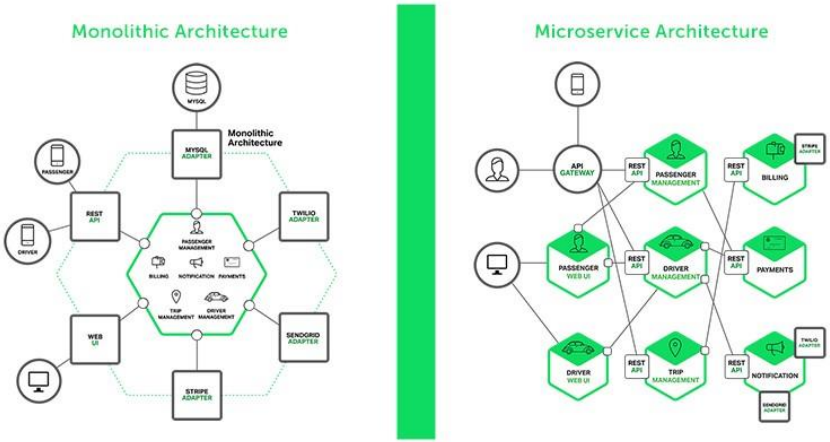
- How the services communicate with each other depends on your application's requirements, but many developers use HTTP/REST with JSON or Protobuf.
  - DevOps professionals are, of course, free to choose any communication protocol they deem suitable, but in most situations, REST (Representational State Transfer) is a useful integration method because of its comparatively lower complexity over other protocols.

Microservice Architecture

- Unlike microservices, a monolith application is always built as a single, autonomous unit.
  - In a client-server model, the server-side application is a monolith that handles the HTTP requests, executes logic, and retrieves/updates the data in the underlying database.
- The problem with a monolithic architecture, though, is that all change cycles usually end up being tied to one another.
  - A modification made to a small section of an application might require building and deploying an entirely new version.

Microservice Architecture

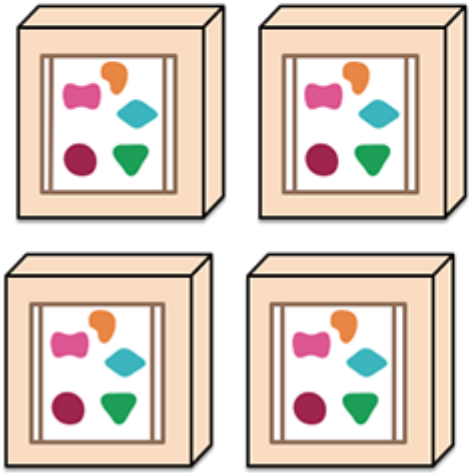
- If you need to scale specific functions of an application, you may have to scale the entire application instead of just the desired components.
- This is where creating microservices can come to the rescue.



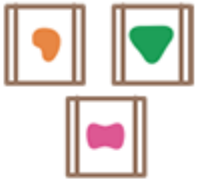
Microservice Architecture

A monolithic application puts all its functionality into a single process...

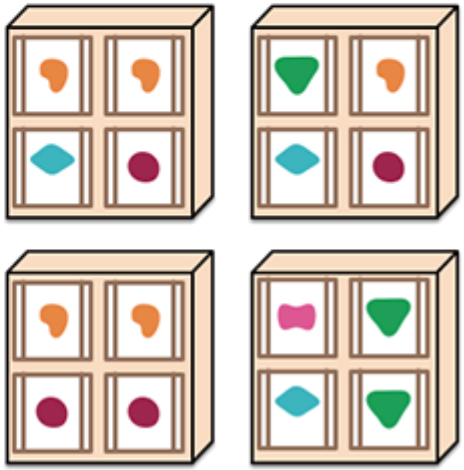
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.



Microservice Architecture

- Isn’t this just another name for SOA?
- SOA and microservice architecture bears a number of similarities, however, traditional SOA is a broader framework and can mean a wide variety of things.
  - The typical SOA model, usually has more dependent ESBs, with microservices using faster messaging mechanisms.
  - SOA also focuses on imperative programming, whereas microservices architecture focuses on a responsive-actor programming style.
  - Moreover, SOA models tend to have an outsized relational database, while microservices frequently use NoSQL or micro-SQL databases.

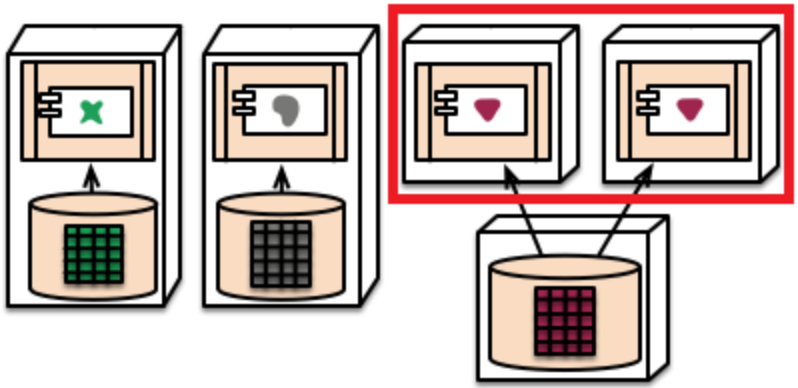
Service Discovery Pattern

Service Discovery Pattern

- Services typically need to call one another.
- In a monolithic application, services invoke one another through language-level method or procedure calls.
- In a traditional distributed system deployment, services run at fixed, well known locations (hosts and ports) and so can easily call one another using HTTP/REST or some RPC mechanism.

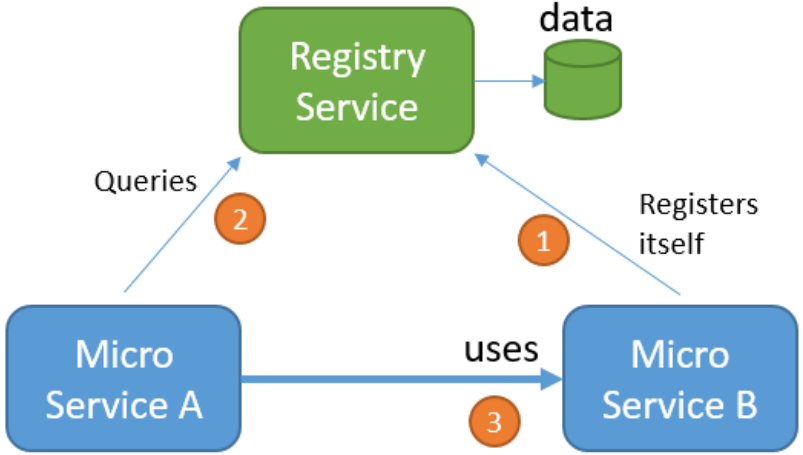
Service Discovery Pattern

- However, a modern microservices-based application typically runs in a virtualized or containerized environments where the number of instances of a service and their locations changes dynamically.



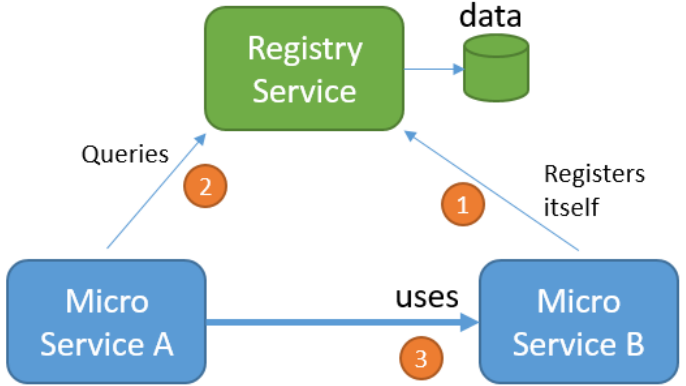
Service Discovery Pattern

- Consequently, you must implement a mechanism that enables the clients of service to make requests to a dynamically changing set of ephemeral service instances.
  - There is a Registry Service.
  - Every service registers itself.
  - Clients query registry first.
  - Directly call available instances.



Service Discovery Pattern

- With Service Discovery, services get loosely coupled.
- Multiple copies of a service can be registered.
- Enables horizontal scalability.
- Systems get more fault tolerance.



Thank You!

LXFT
LISTED
NYSE

LUXOFT