**WHAT ARE MICROSERVICES?**

Microservices involve an **architectural approach** that emphasizes the **decomposition** of applications into **single-purpose**, **highly cohesive** and **loosely coupled** services managed by cross-functional teams, for **delivering and maintaining complex software systems with the velocity and quality** required by today’s digital businesses.

Microservices are language-, platform-, and operating system-agnostic.

They can be used to break down a big monolithic application into smaller and simpler services.

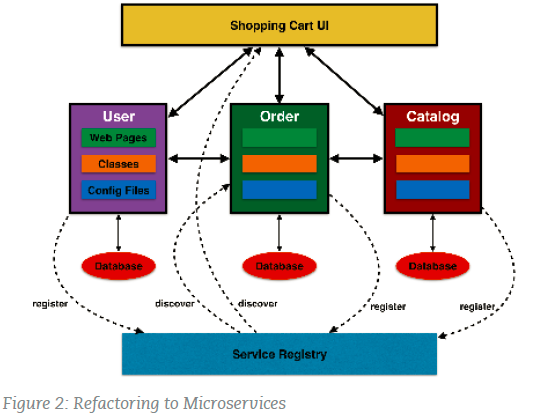
Each microservice does one thing, and does it well, so the “micro” in microservices refers to the **scope of the services’** functionalities, not the number of Lines of Code (LOC).

Microservices may **not be suitable for simpler applications** and are better suited for complex applications that have grown over a period of time, with large codebases and teams.

**Single Responsibility Principle.** Each service is responsible for a single part of the functionality, and does it well.

**Independent DURS (Deploy, Update, Replace, Scale).** Each service can be independently deployed, updated, replaced, and scaled.

**Group code by functionality**, not by layer . Each service has **a business capability**.



**BENEFITS OF MICROSERVICES**

INDEPENDENT SCALING

-They can be scaled up or down based on demand.

POLYGLOT

-Developers are free to pick the language and stack that are best suited for their service.

INDEPENDENT **Deploy, Update, Replace, Scale:**

-Each service can be independently deployed, updated, replaced, and scaled.

EASY MAINTENANCE

**-Single Responsibility Principle.** Each service is responsible for a single part of the functionality, and does it well. Hence it has a smaller and less complex domain.

**RESILIENCE**

-A misbehaving service, such as a memory leak or an unclosed database connection, will only affect that service, as opposed to an entire monolithic application

**AGILITY:**

-Use Convention over Configuration (CoC). Keep configuration to a minimum by using or establishing conventions.

**CENTRAL CONFIG:**

-Make your code stateless and externalize your application’s state.

**Spring Cloud :**

builds on Spring Boot by providing a bunch of libraries that enhance the behaviour of an application when added to the classpath. You can take advantage of the basic default behaviour to get started really quickly, and then when you need to, you can configure or extend to create a custom solution.

**Features**

Spring Cloud focuses on providing good out of box experience for typical use cases and extensibility mechanism to cover others.

• Distributed/versioned configuration

• Service registration and discovery

• Routing

• Service-to-service calls

• Load balancing

• Circuit Breakers

• Global locks

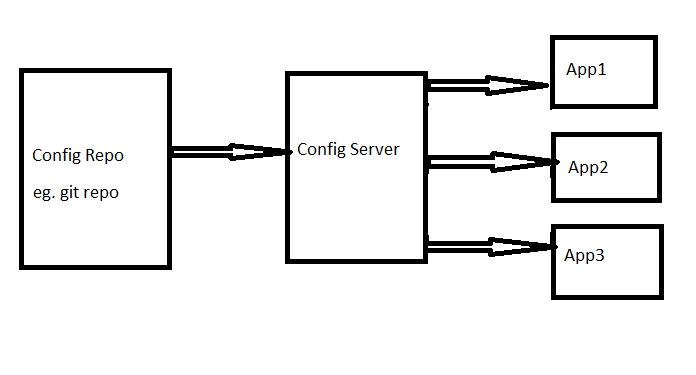
• Leadership election and cluster state

• Distributed messaging

# Main Projects:

**Spring Cloud Config**

Centralized external configuration management backed by a git repository. The configuration resources map directly to Spring `Environment` but could be used by non-Spring applications if desired.



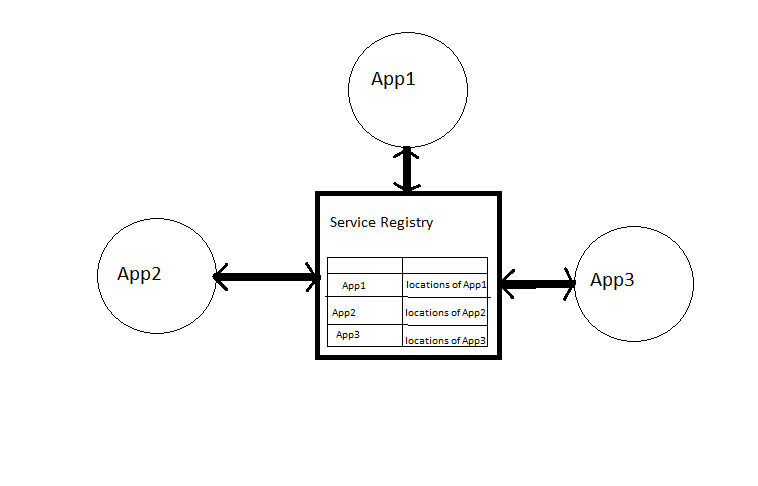
* Supports version control
* Dynamic config update
* Supports config encryption
* Can update configuration of different apps from a single place.

**Spring Cloud Netflix**

Integration with various Netflix OSS components (Eureka, Hystrix, Zuul etc.)

**Spring Cloud Netflix features:**

**Service Discovery:** Eureka instances can be registered and clients can discover the instances using Spring-managed beans



**Circuit Breaker:** Hystrix clients can be built with a simple annotation-driven method decorator

**Declarative REST Client:** Feign creates a dynamic implementation of an interface decorated with JAX-RS or Spring MVC annotations

**Client Side Load Balancer:** Ribbon

**External Configuration:** a bridge from the Spring Environment to Archaius (enables native configuration of Netflix components using Spring Boot conventions)

**Router and Filter:** automatic regsitration of Zuul filters, and a simple convention over configuration approach to reverse proxy creation