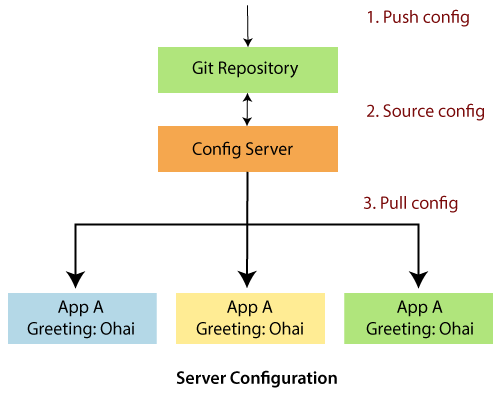
Spring Cloud Components

There are the following components:

* Configuration
* Service Discovery
* Circuit Breakers
* Routing and Messaging
* API Gateway
* Tracing
* CI Pipeline and Testing

## **Configuration**

Spring Cloud configuration components provide server-side and client-side support for externalized configuration in a distributed system. We can manage the external properties with config server for applications across all environments. Spring Cloud config server can use Git, SVN (Apache Subversion), filesystem, and Vault to Store config. Config clients (microservice app) retrieve the configuration client from the server on startup.



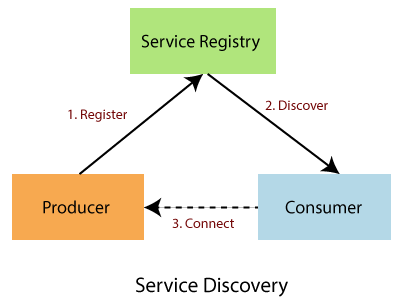
## **Service Discovery**

The service discovery is the automatic detection of devices and services over the network. In other words, service discovery is how an application and microservices connect in the distributed environment. Service discovery implementations include both:

* The **central server** that maintains a global view of the address.
* The **clients** that connect to the central server can update and retrieve the address.

There are **two** discovery patterns: **Client-side discovery** and **Server-side discovery**.

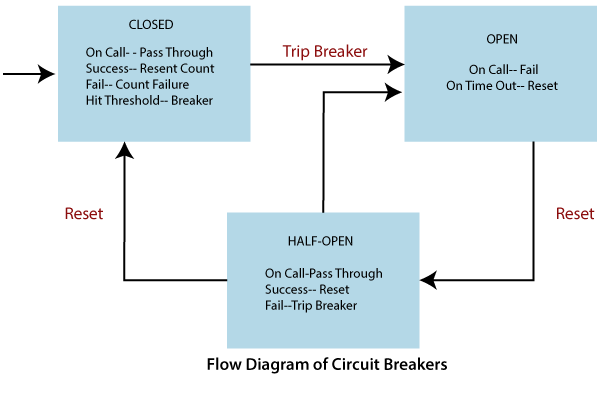
* **Client-side discovery:** In the Client-side discovery, client is responsible for determining the network location of available services. The client uses a **load-balancing algorithm** to select one of the available services and make a request. **Netflix OSS** is an example of a client-side discovery pattern.
* **Server-side discovery:** In the server-side discovery, the client makes an HTTP request to a service through a load balancer. The load balancer contacts to service registry and route each request to an available service instance. Similar to client-side discovery, service instances are registered and deregistered with the service registry. The **AWS ELB** (Elastic Load Balancer) is an example of server-side discovery. ELB balances the external traffic from the internet.



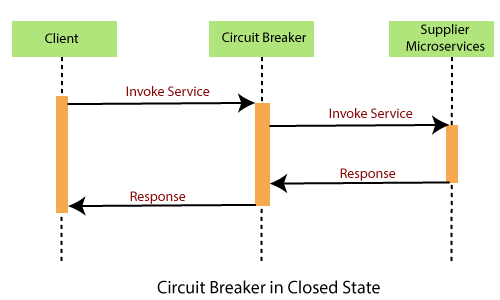
In the above figure producer is a software that sends a message to a message broker (Service Registry). A consumer is also a software that receives the message and processes it.

## **Circuit Breakers**

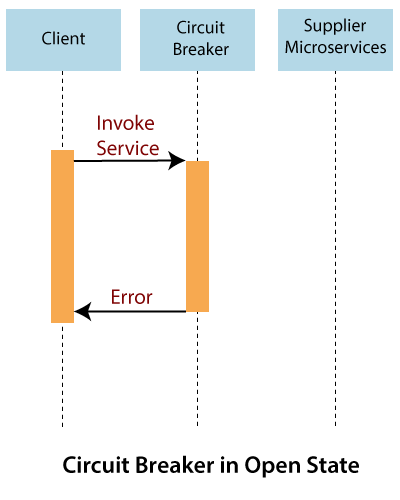
Netflix has created a library called **Hystrix**. It implements the circuit breakers pattern. Circuit breakers calculate when to open and close the circuit and what to do in case of failure. When all services fail at some point, the circuit breaker handles these failures gracefully. The circuit breakers have three states: **OPEN, CLOSED,** and **HALF-OPEN** State.



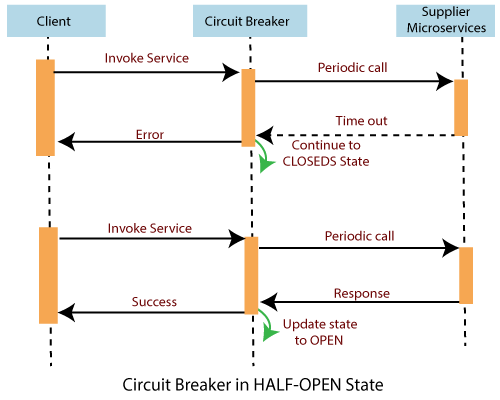
**CLOSED State:** If the Circuit breaker is in the CLOSED state and all calls pass through to the supplier microservices. It responds without any latency.



**OPEN State:** The circuit breaker returns an error calls without executing the function.

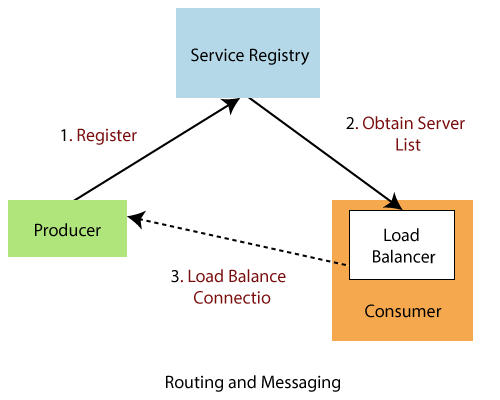


**HALF-OPEN State:** The circuit turns to HALF-OPEN state when a function execution is **timed out**. It test that underlying problem still exists or not. It is a **monitoring** and **feedback mechanism**. It makes a trial call to supplier microservices to check if it has recovered. If the call to the supplier is timed out, then the circuit remains in the **OPEN** state. If the call return success, the circuit-switched to the **CLOSED** state. The circuit breaker returns all external calls to the service with an error during the **HALF-OPEN** State.



## **Routing and Messaging**

The cloud application made up of many microservices so the communication will be critical. Spring Cloud supports communication via messaging or HTTP request. Routing uses **Netflix Ribbon** and **Open Feign while** messaging uses Kafka or Rabbit MQ.

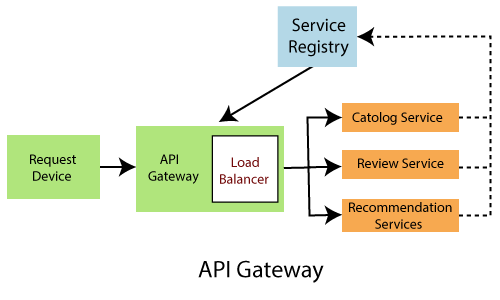


## **API Gateway**

API Gateway allows us to route API request (external or internal) to connect services. It also provides a library for building an API gateway on the top of Spring MVC. Its aims to provide cross-cutting concerns to them, such as **security** and **monitoring**.

**Features of API Gateway**

* Built on Spring framework 5, project reactor and Spring Boot 2.0
* Able to match routes on any requested attribute
* Predicates and filters are specific to routes
* Hystrix circuit Breaker integration
* Spring Cloud Discovery Client integration
* Easy to write Predicates and filters
* Request Rate Limiting
* Path rewriting



## **Tracing**

Spring Cloud's other functionality is **distributed tracing**. Tracing is a single request to get data from the application. Tracing results in an exponentially larger number of requests to various microservices.

We can add **Spring Cloud Sleuth** library in our project to enable tracing. Sleuth is responsible for recording **timing**, which is used for **latency analysis**. We can export this timing to Zipkin.

Zipkin is a distributed tracing tool specially designed for **analyzing latency problem** inside the microservice architecture. It exposes HTTP endpoint used for collecting input data. If we required to add tracing in our project, we should add the **spring-cloud-starter-zipkin** dependency.

In the microservices, the input traffic volume is so high, so we cannot collect an only certain amount of data. For that purpose, the Spring Cloud Sleuth provides a **sampling policy**. The sampling policy allows us how much input traffic is sent to Zipkin for analysis. To enable this feature, we have to add the **spring-cloud-sleuth-stream** dependency.

## **Cl Pipeline and Testing**

Spring Cloud pipeline is an opinionated (self-important) pipeline for Jenkins and Concourse, which creates pipeline automatically for the application. The building, testing, and deploying in various services is critical to having a successful cloud-native application.

The Jenkins pipeline provides a set of the tool designed for modeling simple and more advanced delivery pipeline as code. The definition of a pipeline is written into a text file called Jenkinsfile.

The pipeline has **two** syntaxes: **Declarative** and **Scripted** pipeline. These syntaxes are divided into two parts: Steps, and Stages. **Steps** are the fundamental part of the pipeline as they tell the Jenkins server what to do. **Stages** are the major part of a pipeline. Stages logically group a couple of steps, which displayed on the pipeline's result screen.