Setting up Spring Cloud

Spring Cloud Config is Spring's client/server approach for storing and serving distributed configurations across multiple applications and environments.

Spring Cloud

- Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems (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**).
- Coordination of distributed systems leads to **boiler plate patterns**, and using **Spring Cloud developers** can quickly stand up services and applications that implement those patterns.
- They will work well in any distributed environment, including the developer's **own laptop**, **bare metal data centres**, and managed platforms such as **Cloud Foundry**.

Spring Cloud Features

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

Cloud Config

- This configuration store is ideally versioned under Git version control and can be modified at application runtime. While it fits very well in Spring applications using all the supported configuration file formats together with constructs like **Environment**, **PropertySource** or **@Value**, it can be used in any environment running any programming language.
- We will create two different projects to implement this

Release train Spring Boot compatibility

Release Train	Boot Version
2020.0.x aka Ilford	2.4.x, 2.5.x (Starting with 2020.0.3)
Hoxton	2.2.x, 2.3.x (Starting with SR5)
Greenwich	2.1.x
Finchley	2.0.x
Edgware	1.5.x
Dalston	1.5.x

Main Projects of Cloud

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.

Spring Cloud Netflix

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

Spring Cloud Bus

An event bus for linking services and service instances together with distributed messaging. Useful for propagating state changes across a cluster (e.g. config change events).

Spring Cloud Cloudfoundry

Integrates your application with Pivotal Cloud Foundry. Provides a service discovery implementation and also makes it easy to implement SSO and OAuth2 protected resources.

Spring Cloud Open Service Broker

Provides a starting point for building a service broker that implements the Open Service Broker API.

Main Projects of Cloud

Spring Cloud Cluster

Leadership election and common stateful patterns with an abstraction and implementation for Zookeeper, Redis, Hazelcast, Consul.

Spring Cloud Consul

Service discovery and configuration management with Hashicorp Consul.

Spring Cloud Security

Provides support for load-balanced OAuth2 rest client and authentication header relays in a Zuul proxy.

Spring Cloud Sleuth

Distributed tracing for Spring Cloud applications, compatible with Zipkin, HTrace and log-based (e.g. ELK) tracing.

Spring Cloud Data Flow

A cloud-native orchestration service for composable microservice applications on modern runtimes. Easy-to-use DSL, drag-and-drop GUI, and REST-APIs together simplifies the overall orchestration of microservice based data pipelines.

Main Projects of Cloud

Spring Cloud Stream

A lightweight event-driven microservices framework to quickly build applications that can connect to external systems. Simple declarative model to send and receive messages using Apache Kafka or RabbitMQ between Spring Boot apps.

Spring Cloud Stream Applications

Spring Cloud Stream Applications are out of the box Spring Boot applications providing integration with external middleware systems such as Apache Kafka, RabbitMQ etc. using the binder abstraction in Spring Cloud Stream.

Spring Cloud Task

A short-lived microservices framework to quickly build applications that perform finite amounts of data processing. Simple declarative for adding both functional and non-functional features to Spring Boot apps.

Spring Cloud Task App Starters

Spring Cloud Task App Starters are Spring Boot applications that may be any process including Spring Batch jobs that do not run forever, and they end/stop after a finite period of data processing.

Spring Cloud Zookeeper

Service discovery and configuration management with Apache Zookeeper.

Main Projects of Cloud

Spring Cloud Connectors

Makes it easy for PaaS applications in a variety of platforms to connect to backend services like databases and message brokers (the project formerly known as "Spring Cloud").

Spring Cloud Starters

Spring Boot-style starter projects to ease dependency management for consumers of Spring Cloud. (Discontinued as a project and merged with the other projects after Angel.SR2.)

Spring Cloud CLI

Spring Boot CLI plugin for creating Spring Cloud component applications quickly in Groovy

Spring Cloud Contract

Spring Cloud Contract is an umbrella project holding solutions that help users in successfully implementing the Consumer Driven Contracts approach.

Spring Cloud Gateway

Spring Cloud Gateway is an intelligent and programmable router based on Project Reactor.

Main Projects of Cloud

Spring Cloud OpenFeign

Spring Cloud OpenFeign provides integrations for Spring Boot apps through autoconfiguration and binding to the Spring Environment and other Spring programming model idioms.

Spring Cloud Pipelines

Spring Cloud Pipelines provides an opinionated deployment pipeline with steps to ensure that your application can be deployed in zero downtime fashion and easily rolled back of something goes wrong.

Spring Cloud Function

Spring Cloud Function promotes the implementation of business logic via functions. It supports a uniform programming model across serverless providers, as well as the ability to run standalone (locally or in a PaaS).

Netflix Dependency

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

Listing Spring Cloud Dependencies

```
<properties>
    <spring.cloud-version>Hoxton.SR8</spring.cloud-version>
</properties>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring.cloud-version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Required Dependencies(Server)

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-config-server</artifactId>
    <version>3.0.4</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Required Dependencies(Client)

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-config</artifactId>
    <version>3.0.4</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Steps to Create server

- Step 1: Create a Maven project using Spring Initializr https://start.spring.io/
- Step 2: Choose the Spring Boot version 2.2.0 M6 or higher version. Do not choose the snapshot version.
- Step 3: Provide the Group name. In our case, com.tarkesh.microservices.
- Step 4: Provide the Artifact id. We have provided spring-cloud-config-server.
- Step 5: Add the Spring Boot DevTools and Config Server dependencies.
- Step 6: Click on Generate the project button. A zip file will download, extract it in the hard disk. And open it with any IDE

Git Installation

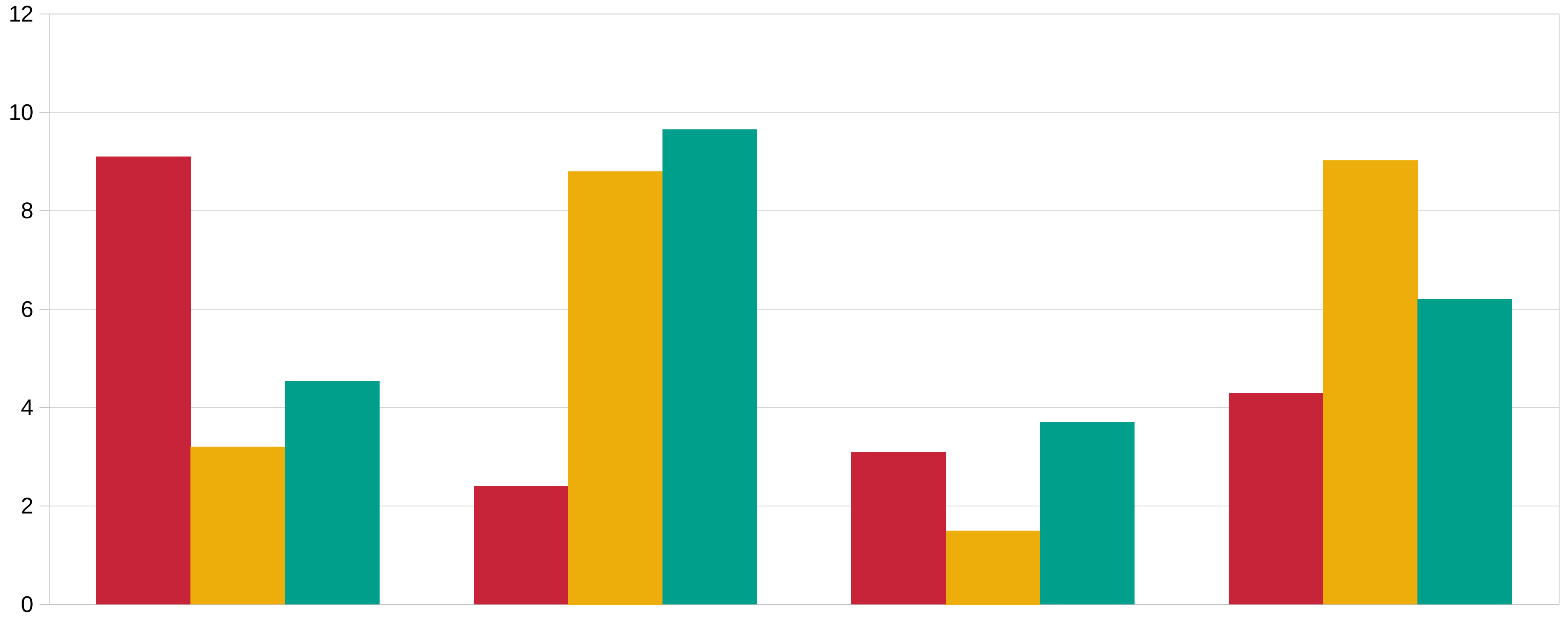
- Step 1: Download Git from https://git-scm.com/ and install it.
- Step 2: Create a Git repository and store the files that we want to be able to configure a limits-service. We will try to access them from the spring-cloud-config-server. Open the Git bash and type the following commands:
- Creating a new directory:
- mkdir git-localconfig-repo
- cd git-localconfig-repo/
- Initializing a new Git repository:
- git init

Git Installation

- Step 3: Now move to the spring-cloud-config-server project and add a link to the specific folder.
- Right-click on the spring-cloud-config-server project.
- Click on Build Path->Configure Build Path…
- Select the Source tab.
- Click on Link Source and browse the folder git-localconfig-repo.
- Right click on the folder-> New -> Other -> File -> Next -> Provide the file name: limits-service-properties-> Finish.
- Now write the following code in the properties file:
- limits-service.minimum=8
- limits-service.maximum=888

Git Installation

- Step 4: Configure the user name and user email:
- git config -global user.email abc@example.com
- git config -global user.name "abc"
- The command commits any file we have added with the git add command and also commits any files we have changed since then.
- git add -A
- Now execute the command to commit the changes in the repository. It records or snapshots the file permanently in the version history.
- git commit -m "first commit" Step 4: Configure the user name and user email:
- git config -global user.email abc@example.com
- git config -global user.name "abc"
- The command commits any file we have added with the git add command and also commits any files we have changed since then.
- git add -A
- Now execute the command to commit the changes in the repository. It records or snapshots the file permanently in the version history.
- git commit -m "first commit"


12
10
8
6
4
2
0