**Micro-service Architecture**

It is a method of developing as a set of independently deployable, smaller modular service

in which each service runs a unique process

Split domain design is based on 🡪 **bounded Context**

----------------------------------------------------------------------------------------------------------------------------------------

**Bounded context:**

* It is domain driven design
* System represents the entire business
* Each system contains multiple subsystems
* Each sub system could be a bounded context in micro-services

**Typical Microservice-module**

* Gateway “ 8080
* Registry server ( service discovery )
* Config server
* API services

**Discovery-server**

In cloud we find a running instance from within Discovery server

**Server Instance**

* **Traditional network**

server instances are static 🡪we need to know IP address and port

* **Cloud**

It is dynamic 🡪 there are so many instances running 🡪 Solution is Discovery server

**Discovery-Server type**

1. **Client-side discovery**

* **Client is responsible** to find available service instance by **query** the **discovery server**
* Then client is using load balancing algorithm to select one of them
* The network location of a service instance is registered with the **service registry** when it starts up
* [Netflix OSS](https://netflix.github.io/) provides client‑side discovery pattern. [Netflix Eureka](https://github.com/Netflix/eureka) is a **service registry.**
* [Netflix Eureka](https://github.com/Netflix/eureka) provides REST API to register and query

1. **Server-side discovery**

* The client makes a request to a service via a load balancer.(router)
* Load balancer query service registry.
* The [AWS Elastic Load Balancer](https://aws.amazon.com/elasticloadbalancing/) is an example of Server-side Discovery

**Eureka**

It is server to register other services in Micro-service platform   
 Prepare instance to look up by Micro-services

**ZULL proxy**

It is server-side load balancer by Netflix   
 Allows a browser to consume services from multiple hosts

without managing cross-origin-resource

**Ribbon:**

Client-side load balancer setup in gateway

**Ribbon + Feign client**

Offer client side load balancer

**Circuit-breaker**

Circuit breakers prevents repeatedly calling a failing service

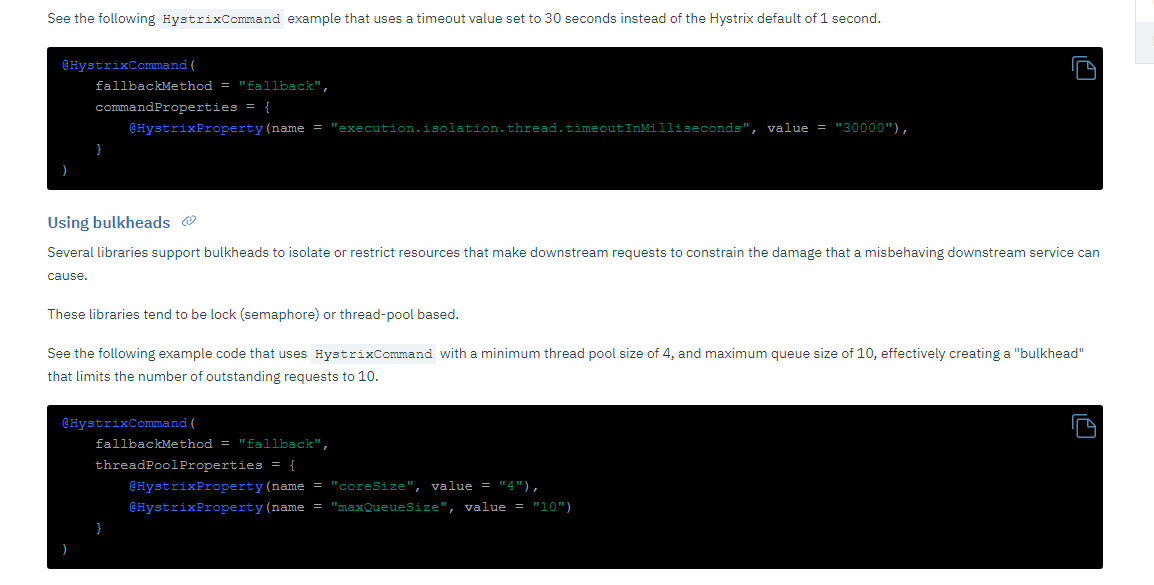
Circuit breaker 🡪 it is a design pattern for fault tolerance, latency and error

Fall back method for error handling

**Hystrics**

It is an implemented product from Netflix in Circuit breaker to reduce impact of failure and latency.

<https://console.bluemix.net/docs/java-spring/fault_tolerance.html#tolerance>



@HystricsCommand has

fallback method name,

command property

Hystricproperty

**Distributed Transaction**

The issue in micro-service is distributed Transaction

2 solutions to fix the issue in Micro-services

1. **2 phase commit in Database level** 🡪
   1. Issue is that it is 2pc is synchronous (blocking).   
       This prevents the customer from applying new orders.
   2. 2pc allows read-write isolation.
2. **Saga**

It is Asynch by local 🡪 when it is done send a message to event bus

Support for a long-lived transaction

Saga pattern does not have read isolation

**Saga**

**Chorography Saga –message broker, it is not centeralzied**

* Services are connected to message bus and subscribe events that they are interested in.
* The first service execute a transaction and publishes an event
* This event listened by one more services which executes local transaction
* The service that listed the first event executes the second events
* The distributed transaction ends when the last service executes its local transaction and does not publish new events

Issue:

* + Business flow is spreading across multiple services
  + Any changes to business flow will lead to change in multiple services
  + Difficult to rollback

**Orchestration Saga –**

[**https://blog.couchbase.com/saga-pattern-implement-business-transactions-using-microservices-part-2/**](https://blog.couchbase.com/saga-pattern-implement-business-transactions-using-microservices-part-2/)

1. Saga orchestrator module will tells others what to do and when
2. If anything fails then orchestrator is responsible to coordinate rollback by sending undo comment to the previous operation.
3. Orchestrator invokes participant but participant does not call the Saga
4. Centralized orchestration
5. Drawback is adding more complexity to orchestrator
6. Risk of constating too much logic
7. For traceability we caret 🡪 unique identifier to pass around each messages
8. Add reply address to in message so that each participant can reply multiple orchetsrators

**Scaling micro-services**

1. Devops, Auto deployment - Continues integration
2. Infrastructure , no room for new instance, discuss to cloud AWS and Google cloud
3. Microservices must be stateless with zero dependencies

DataDog, DynaTrace

**Distributed tracing pattern** This pattern used for trouble shooting.   
 Assigning a unique id for every request including start-date, end\_date and print in log file as well.

**Cross-cutting concern**

health check  
 Externalaized configuration  
 Logging   
 Metrcis

**Spring boot Actuator**

Spring boot has built-in API for health check

https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-endpoints.html

Micro-service development management (Actuator)

Generate UI for our end points , has 2 parts UI and server side.

SwaggerConfig : Enable swagger is based on io.springfox which creates Docket  
Swagger UI: Io.springfox-swagger-ui

**Health check pattern:**

It is an end-points that comes with spring to check whether or not service is down,

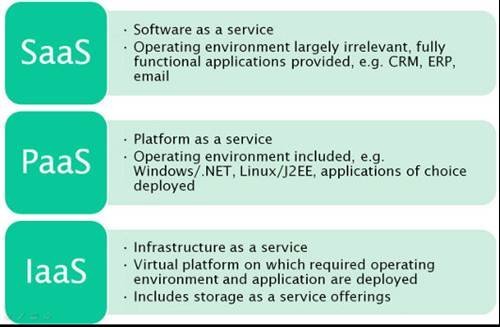
All clientside and server side load balancers are calling this endpoint frequently

**What is Cloud** :

The practice of using a network of remote servers hosted on the Internet to store,

manage, and process data, rather than a local server or a personal computer.

**Infrastructure as a service (IaaS)  
Platform as a service (PaaS)  
Software as a service (SaaS)**



**Spring cloud config server**

Provides an HTTP resource-based API for external configuration

@**EnableConfigServer**

By default is runs on 8080

**Bootstarp.yml** 🡪 contain name of application and profile environment

**Environment variables in ConfigServer :**

Environmnet repository is where we save our property values

Structure of repository :

**The default implementation of Environment Repository is GIT back-end.**

This is good for Audit changes or any other physical update

To change the location of repository we can use

spring.cloud.conifg.server.git.uri = file:// 🡨 local

spring.cloud.conifg.server.git.uri = GIT url

Difference between Local and GIT:

Server directly goes to local repository without cloning from GIT

Even in that case, it is better to use the ssh:

 protocol for a shared filesystem repository,

so that the server can clone it and use a local working copy as a cache.

Place holder:

Spring Config su[pports place holder for Apppliction,label and profile

spring:

cloud:

config:

server:

git:

uri: https://github.com/myorg/{application**}**

### Pattern Matching and Multiple Repositories

Comes with pattern matching on the application and profile name

spring:

cloud:

config:

server:

git:

uri: https://github.com/spring-cloud-samples/config-repo

repos:

simple: https://github.com/simple/config-repo

special:

pattern: special\*/dev\*,\*special\*/dev\*

uri: https://github.com/special/config-repo

local:

pattern: local\*

uri: file:/home/configsvc/config-repo

The pattern property in the repo is actually an array, so you can use a YAML array (or [0], [1], etc. suffixes in properties files

Type of ConfigServer URI :

It could be SSH or HTTPS

Force-pull :

If the local copy of repository gets dirty then forc-pull = true will save the problem

Delete untrack branches

In order to keep local repository branches clean and up to remote - deleteUntrackedBranches property could be set. It will make Spring Cloud Config Server **force**delete untracked branches from local repository. Example:

Vault backend:

It is secure for certain properties such as password we use Vault profile

By default Vault server runs on 8200

**Health indicator**

## Health Indicator

Config Server comes with a Health Indicator that checks whether the configured EnvironmentRepository is working

spring:

cloud:

config:

server:

health:

repositories:

myservice:

label: mylabel

myservice-dev:

name: myservice

profiles: development

Security in Config server:

For repository the same as GIT we have

SSH

Basic auth

If it is basic Auth we can have user name and password in conifgServer yml file

git:

uri: https://github.com/spring-cloud-samples/config-repo

username: trolley

password: strongpassword

for SSH we can have private key copied in there.

Cross-Origin Resource Sharing ([CORS](https://developer.mozilla.org/en-US/docs/Glossary/CORS))   
It is a mechanism that uses additional [HTTP](https://developer.mozilla.org/en-US/docs/Glossary/HTTP) headers to tell a browser to let a web application running

at one origin (domain) have permission to access selected resources from a server at a different origin.

Browsers restrict cross-origin HTTP requests initiated from within scripts.

<https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

Unless the response from the other origin includes the right CORS headers.

