ZenDesk Project

**Ideal Requirements**

Build microservices

Java,scala

Load balancing, monitoring <https://dzone.com/articles/7-things-to-consider-while-moving-to-a-microservic>

Docker, kubernetees

Upload to gcp and try use for demo

| grep jpa look up linux things, refresh on pipes etc

**Notes**

You need an eureka service because it’s a discovery mechanism that your microservices can register and allows them to communicate.

Monolithic architecture – hard to make changes as application grows without affecting the rest of the code

Microservice architecture – load balancing, microservice problems are common/generic meaning that there are frameworks to deal with these issues.

Find common problems for microservices and try optimize project that way.

--

Service discovery is a design pattern.

Webservice vs microservice – webservices doesn’t know who or what it will be used for it just is there for people to use, microservice has an intended purpose and goal.

In Spring boot each microservice is a spring boot app.

Services have heart beats that you can see in the console to show they are still up, they ping the server, if the server doesn’t get a ping in x time the service is cut off because it can not be found

To ensure APIs are backward compatible wrap things in an object when you return a list and an API is expecting an object you an compromise but when its expecting a list and you send it an object it will break, likewise when you change this any service depending on this

Introducing external apis adds more points of failure to the system

**ToDos**

Find out how do you manage versioning in microservices

Find out how to not hard code urls

Do load balancing for example one service might be used often duplicate it several times, this is why hard coding urls is bad because now which url are you going to use ? they have to be allocated dynamically

**Part 2**

Fault tolerance – what is the impact of a fault, how much tolerance a system has for a specific fault

What will happen when 1 service goes down.

Resilience – how many faults can a system tolerate before it halts, also how much can it bounce back when these faults occur. Is there a mechanism where the system can correct itself?

Time out threads so that they don’t clog up the server much like the issue with are you dead yet attacks on apache servers.

Pattern for fault tolerance **Circuit breaker pattern**

-Detect something is wrong

-Take temp steps to avoid it, eg don’t send requests to it, use another instance of service

Interrupt current flow after a fault is detected

What are the circuit trip parameters?

-Last n requests to consider for the decision

- how many of those should fail

-timeout duration

Find more later

**When does it bounce back ?**

Give it time to not take requests ie sleep for n time

Throw an error

Return a fallback response.

Cache previous responses and use when possible.

Fail fast so you can recover.

How to impl pattern

When to break circuit | what to do when it breaks | when to resume requests

It needs concurrency programming use framework called Hystrix to make it easier.

Performance testing can help you find parameters for circuit breaks.

**Histrix**

Implements circuit breaker pattern

Give it config param and it will do the work

Adaptive fault tolerance