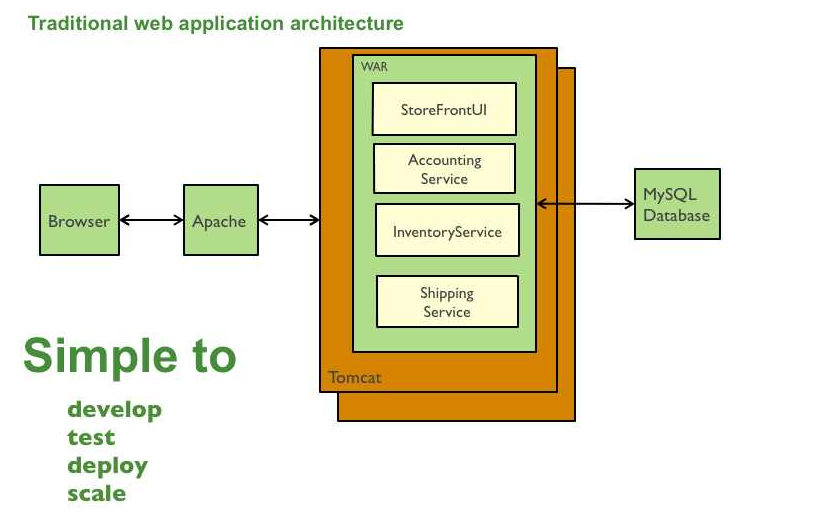
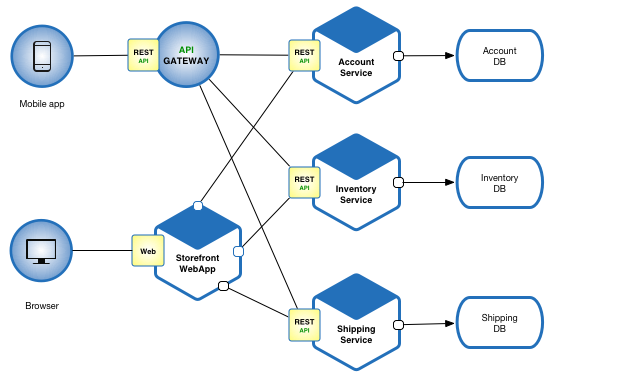
**MicroServices:**

Monolithic:



* Simple to develop - the goal of current development tools and IDEs is to support the development of monolithic applications
* Simple to deploy - you simply need to deploy the WAR file (or directory hierarchy) on the appropriate runtime
* Simple to scale - you can scale the application by running multiple copies of the application behind a load balancer
* **DrawBacks:**
* The large monolithic code base intimidates developers, especially ones who are new to the team. The application can be difficult to understand and modify. 4
* Overloaded IDE
* Overloaded web container - the larger the application the longer it takes to start up. This had have a huge impact on developer productivity because of time wasted waiting for the container to start. It also impacts deployment too.
* Continuous deployment is difficult - a large monolithic application is also an obstacle to frequent deployments. In order to update one component you have to redeploy the entire application. This will interrupt background tasks (e.g. Quartz jobs in a Java application),
* Requires a long-term commitment to a technology stack

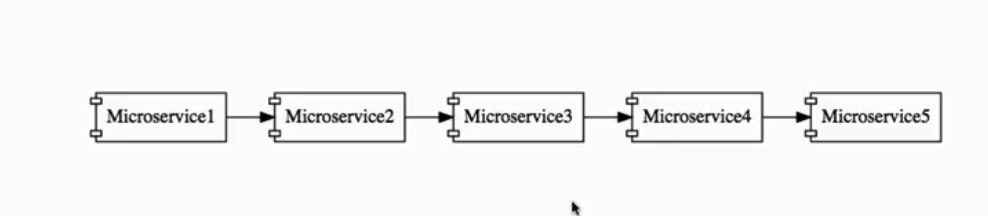


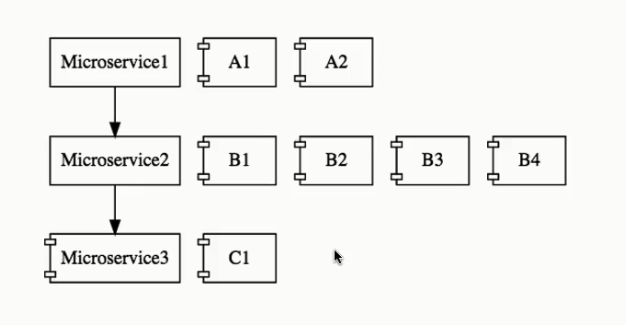
Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities. The microservice architecture enables the continuous delivery/deployment of large, complex applications. It also enables an organization to evolve its technology stack.

In Short,

* Services exposed By REST
* Small well-chosen deployable unit
* Cloud Enabled

Its means set of micro services integrating each other.





**II.Challanges when Building MS**

**Bounded Context:**

**H**ow to we identify to do each of these micro services. What should we do, what should we not do.

What are right boundaries is one of the challenges

**Configuration Management:**

We would have n no.of microservies, each ms have multiple instances in each env.

**Dynamic Scale up/down:**

Loads on the MS may be different at diffent instances of time .Some point of time I may need more instances. Should able to bring up/down based on the load

**VISIBILITY:**

Functionality is distributed in n.nof microservies then finding bug is very dicfficult.We have centeralize log we can look it.Montiroing of MS

**PACK of Cards:** one ms is calling another and its calligng other and so on. If one of these MS fails all fails.

Its if very important to take care about faulttolerance.How do I build faulttolerance to my app.

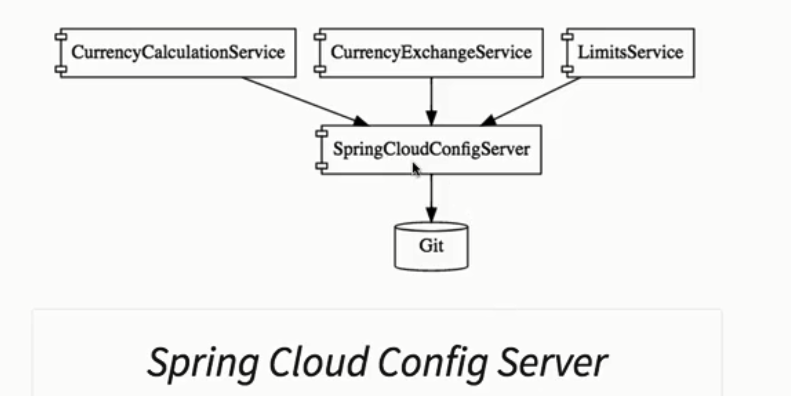
**III.Spring cloud**

Have a look: <http://projects.spring.io/spring-cloud/>

Spring cloud provides solutions for the above challenges

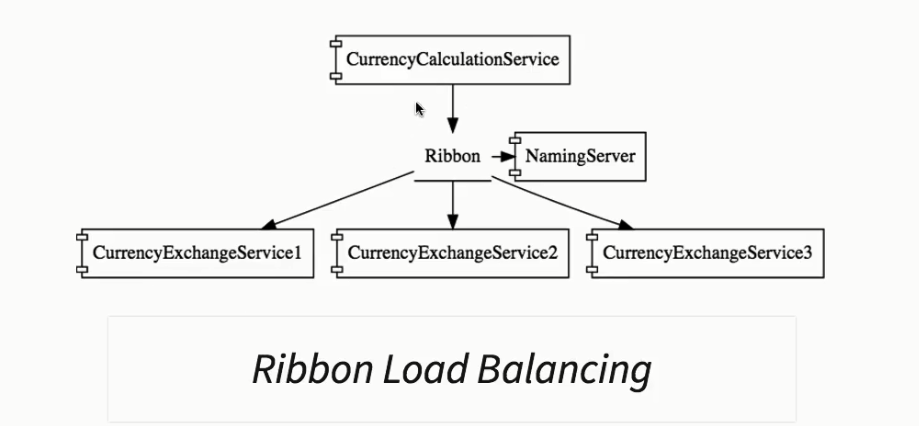
1.Configuration Management: Spring Cloud Config Server

-> Store all configutions diff env fro diff ms store in github

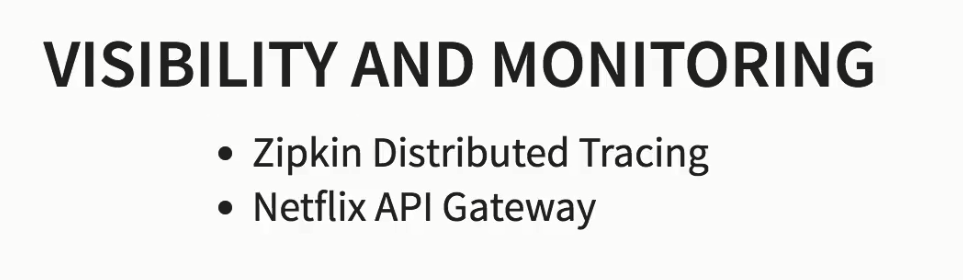


Dynamic Scale UP/Down:





**VISIBILTY AND MONITORING:**





**MS Advantages:**

* **New technology and process adoption**
* **Dynamic Sclaing**
* **Faster realase**