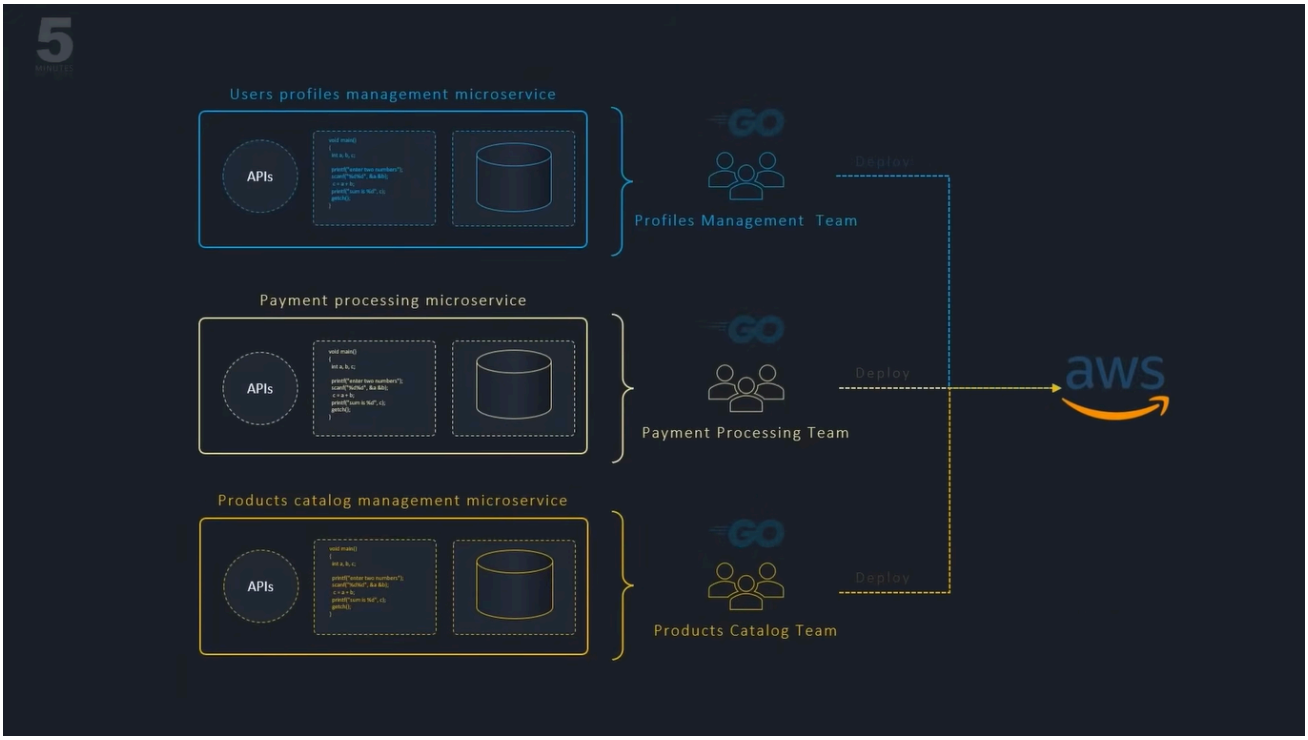
MicroServices Apps

MicroServices Definition

- So basically, in the old way of developing apps, we used the "Monolith" way, in which we put all our codes samples into one bloc (App).
- Like we run one command to activate all of our codes parts of our app...
- This is considered risky, as these parts can affect each other while modifying, and many other inconvenients such as obligation to use the same dependencies between development teams...
- (When we say app parts, for example, let's say we're developing an e-commerce app, the parts are : Products, Orders, Users...)
- Here, when we use MicroServices, we do each part individually in an app, and these apps connect with each other using networking...
- The main benefits here, are that when each development team takes charge of a part, they can implement their own dependencies (even if different from others), each part with it's own database to not interfere and collapse with the other parts data...
- It helps us also evolve our app specific part with minimal risks concerning the other parts (Something not featuring in the Monolith Architecture).
- This architecture helps manage the big loads of traffic...
- Example :



MicroServices App With Spring Boot

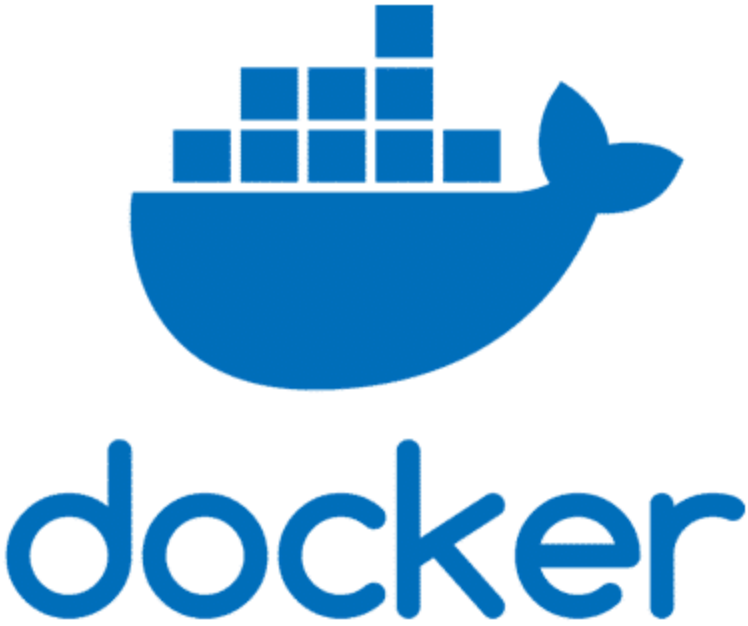
- So the thing is, usually, when I built apps using Spring Boot for the backend, I would create one Spring Project containing packages such as controllers, services, repositories...
- And all the app parts are divided to those packages in a single Spring Project...
- And I executed only one Spring Boot run command (Which was related to one port of my local network)...
- So this was Monolith
- Here, in the MicroServices Architecture, we create a Spring app for each app part individually, each one with it's own dependencies, port and stuff...
- We run all these Spring projects (instead of just one), and they all connect with each other and with the frontend which is also independant...
- Note : Each part has it's own database...
- 

Docker

- Now, comes one of the main characters of MicroServices Architecture : "Docker".
- So when we seperate our project into frontend, backend parts, databases, services (such as SQL for example), we put them into containers...
- Each container is configured from it's image "DockerFile" to match that app part requirements and the developement team methodologies
- Each container is affected to a port, and so this is how they communicate with each other...
- And they use also the APIs that are also independant in their own container...
- "Ports" are key for efficacy in communcation between our app components...
- Note : As we know, we can make a lot of containers from a single image... Why?, because of the loads of traffic that are gigantic and never stop... (We need availability and

consistency)...

- And this is gives us a sneak peak to Distributed Systems...



Kubernetes

- "Kuberenetes" is like an upgraded version of Docker. As it also uses containers in it's work...
- The difference is that Docker is for simple applications, so it only works locally, and stores containers in the same machine...
- The problem here is that even when we create a lot of containers, it may never be enough if we have a lot of traffic..
- So Kubernetes offers a lot more containers then Docker stored in a cluster architecture not only in our local machine...
- In an Automatic way of course...

kubernetes

Real World MicroServices Architecture

- So here, we're talking about when we want to deploy our MicroServices App to an actual Server for usage...
- Here, we'll use AWS Cloud Service...
- So basically, once you have your app perfectly architected in Docker, you need to deploy it in a server (CI/CD pipelines), as it will be given a real IP address to connect with real user instead of localhost from your machine in Docker
- There are two ways for this :

Method 1 (The simplest) :

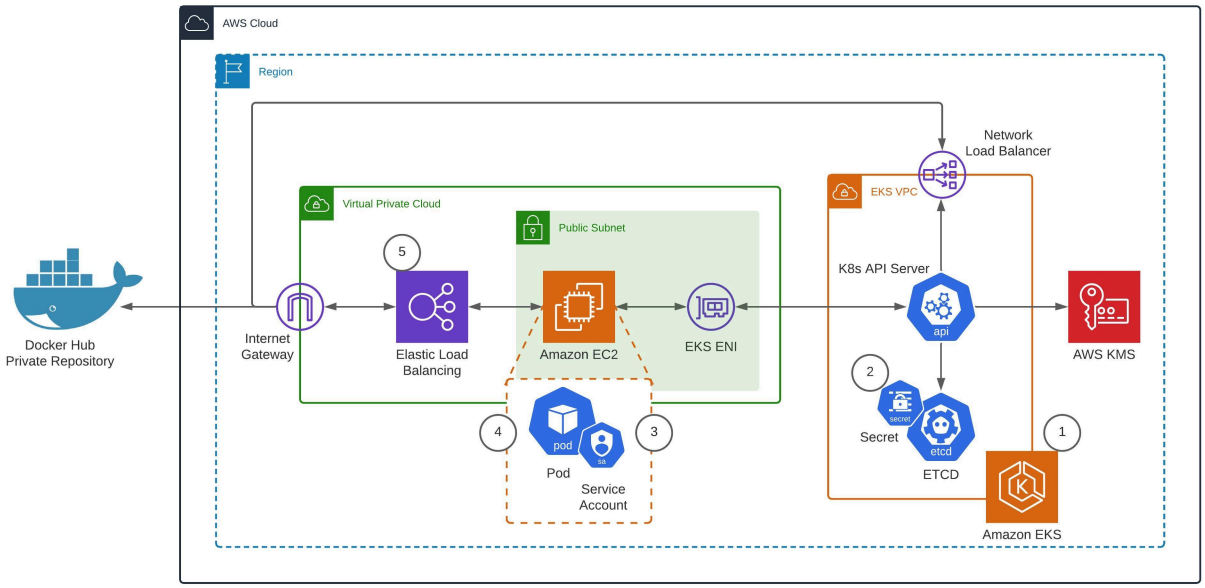
- You create an EC2 server instance, you give it a Linux OS, and you push your Docker architecture into it, and it runs with real network, and even the app parts use the real IP ports to communicate...

Method 2 (The hardest but the more efficient) :

- So as we know, AWS provides multiple services, each for a specific task...
- So actually, It offers services to make a similar Docker Architecture with Containerization directly in a efficient way and in the servers...

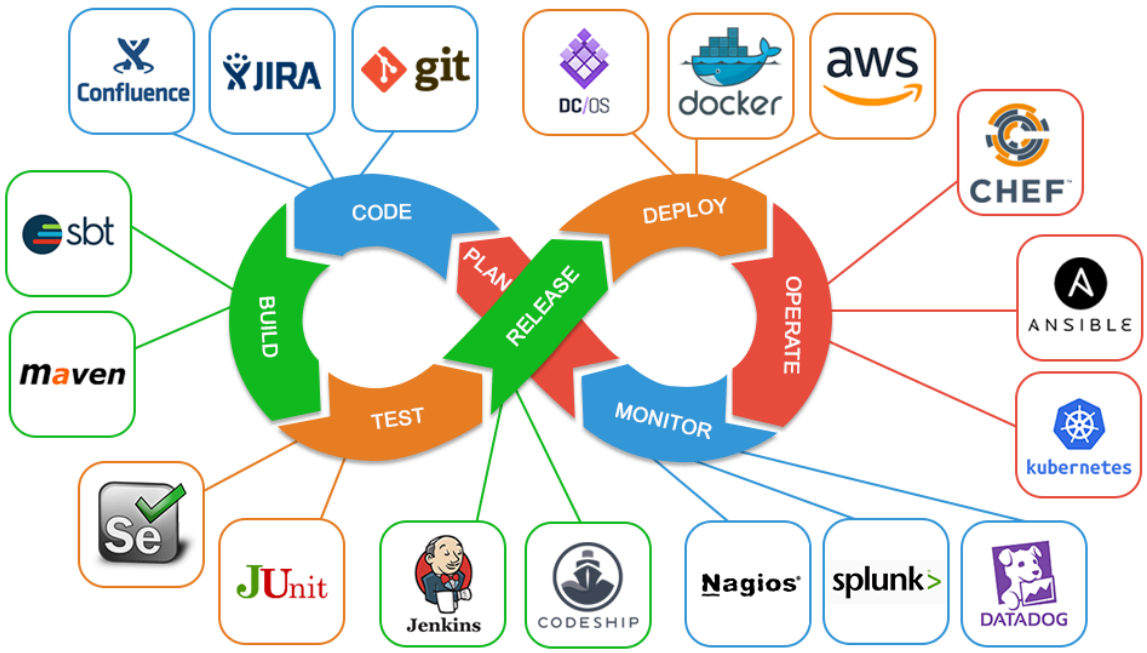
- Here are the differences:

Local	AWS Equivalent
Docker	Docker
Docker Compose	ECS
Local Network	VPC
Postgres Container	RDS
Redis Container	ElastiCache
Nginx Gateway	Load Balancer
Docker Hub	ECR



DevOps

- So DevOps actually makes all of this simple...
- It is a group of Best-Practices, tools and stuff to help us establish all of this smoothly...
- It provides us also with tools to do some tricks automatically instead of doing them manually (Which is a load of hard work and trouble)...


Confluence
JIRA
git
DC/OS
docker
aws
CHEF
sbt
maven
CODE
PLAN
BUILD
TEST
RELEASE
DEPLOY
OPERATE
MONITOR
ANSIBLE
kubernetes
Se
JUnit
Jenkins
CODESHIP
Nagios
splunk>
DATADOG

-
-