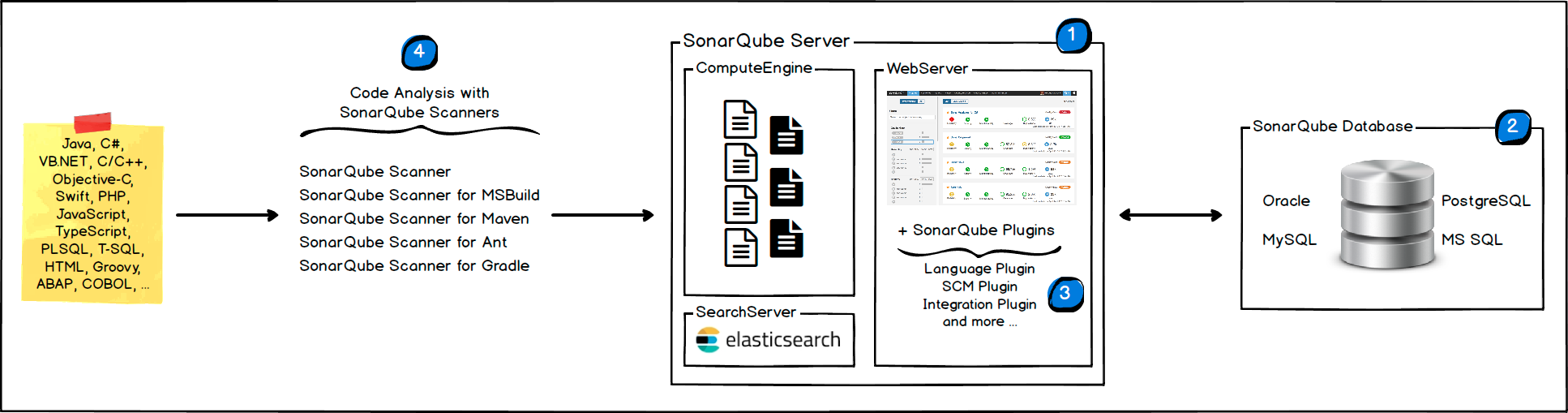
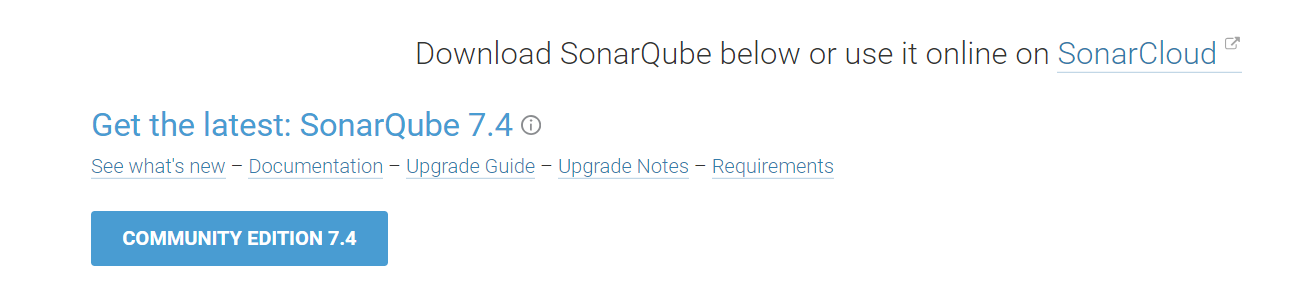
The SonarQube Platform is made of 4 components:  


1. One SonarQube Server starting 3 main processes:
   * Web Server for developers, managers to browse quality snapshots and configure the SonarQube instance
   * Search Server based on Elasticsearch to back searches from the UI
   * Compute Engine Server in charge of processing code analysis reports and saving them in the SonarQube Database
2. One SonarQube Database to store:
   * The configuration of the SonarQube instance (security, plugins settings, etc.)
   * The quality snapshots of projects, views, etc.
3. Multiple SonarQube Plugins installed on the server, possibly including language, SCM, integration, authentication, and governance plugins
4. One or more SonarScanners running on your Build / Continuous Integration Servers to analyze projects

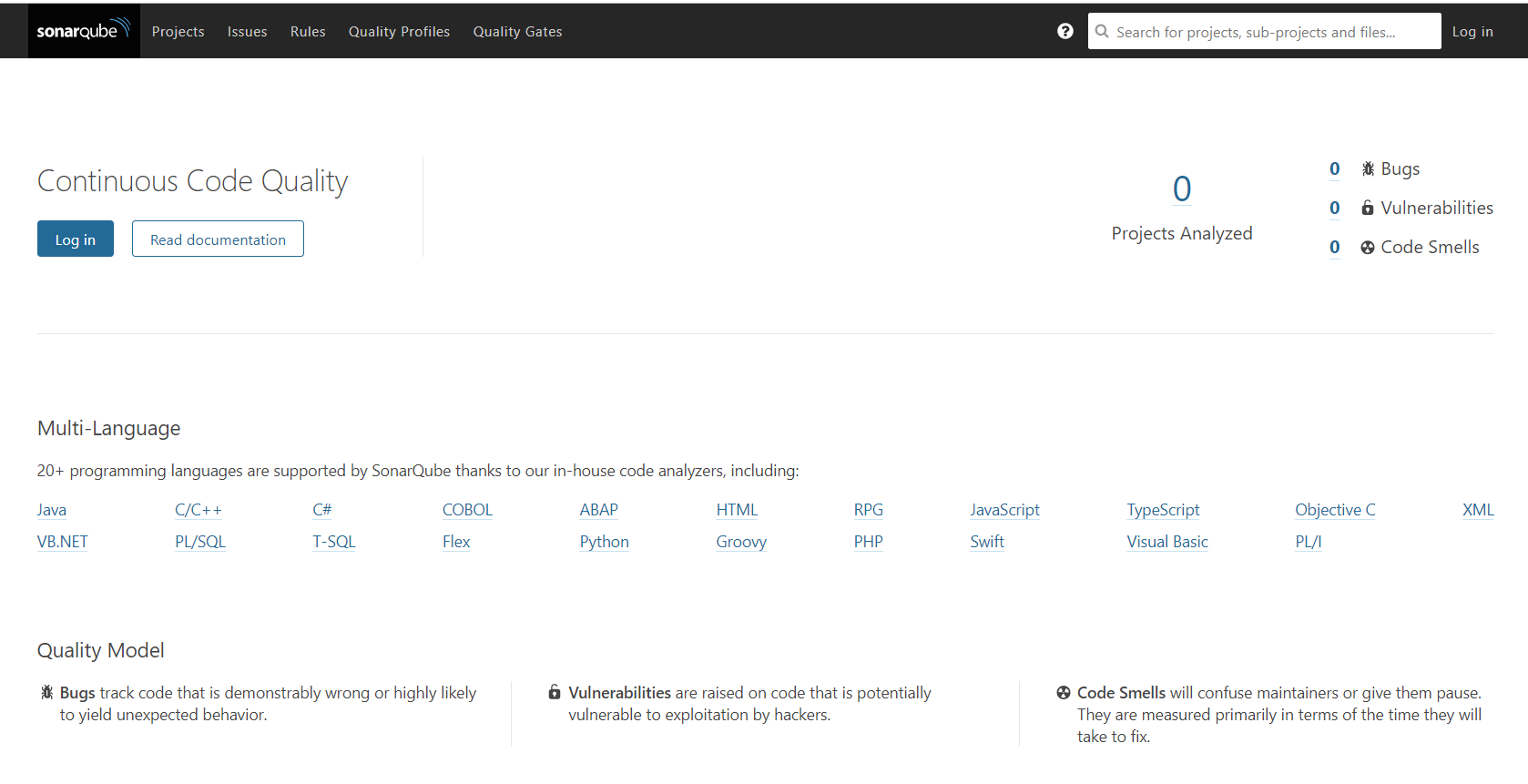
We can work online on SonarQube in the last versions but we will be installing it locally for this project, let’s start with installing and lunching SonarQube:



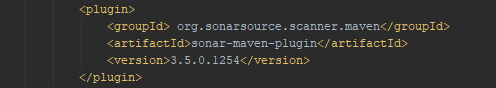
After unzipping the content of the zip downloaded in C:\SonarQube-7.4 foe example, we can start it with a simple command

C:\SonarQube-7.4\bin\windows-x86-64\StartSonar.bat

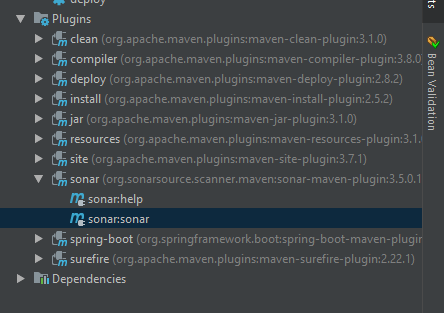
Then going to <http://localhost:9000> we will see the home page of SonarQube



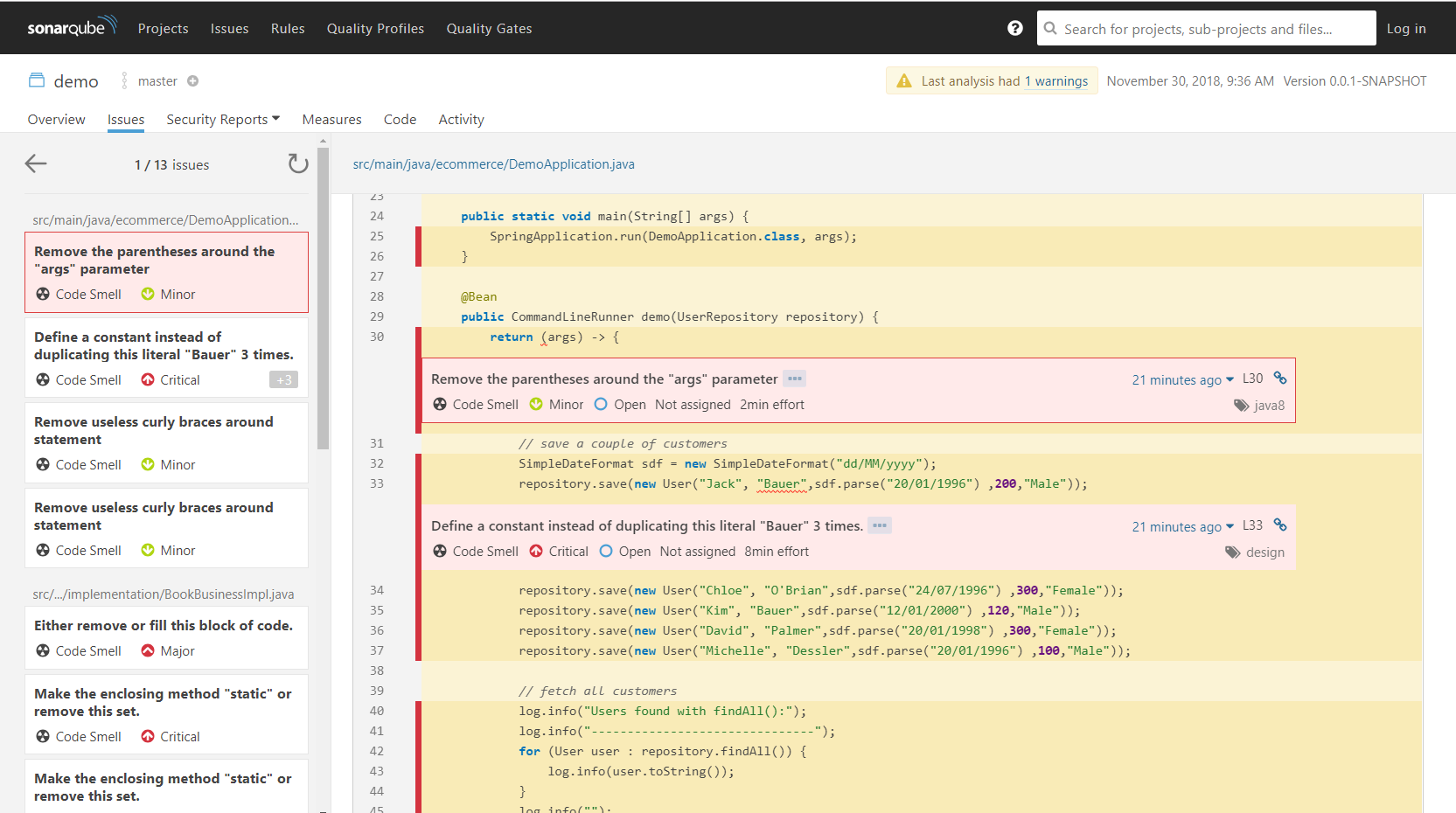
To test SonarQube we will be using maven project, so we need to add sonar plugin into pom.xml



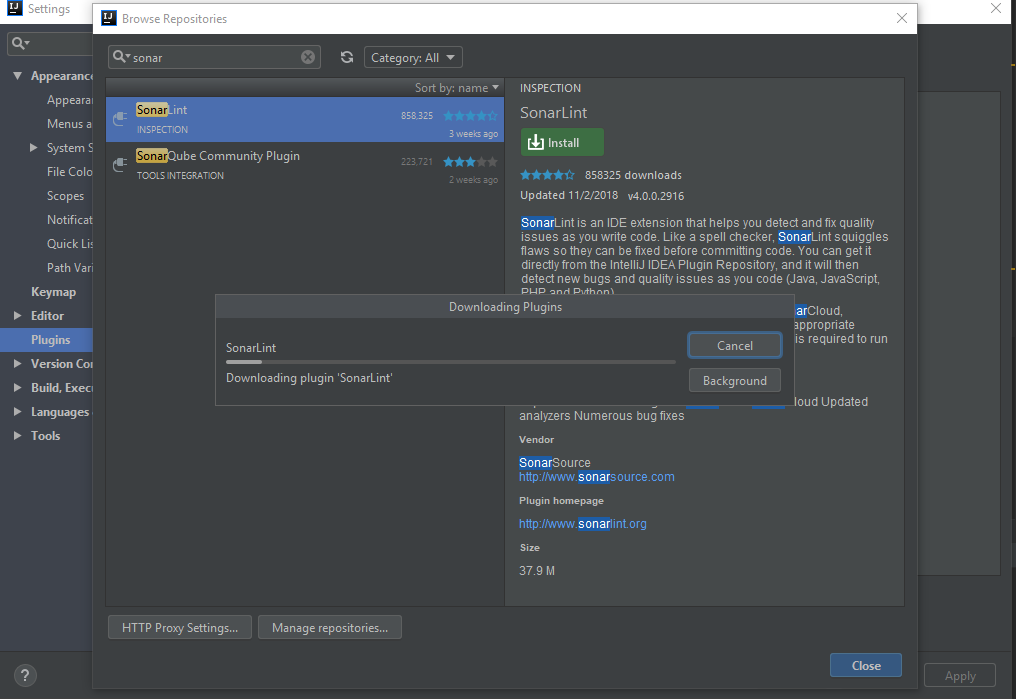
After running clean and install, we can run sonar:sonar plugin



After a build success we can now open <http://localhost:9000> to observe this interface that contains the projects analyzed by SonarQube and the detected issues

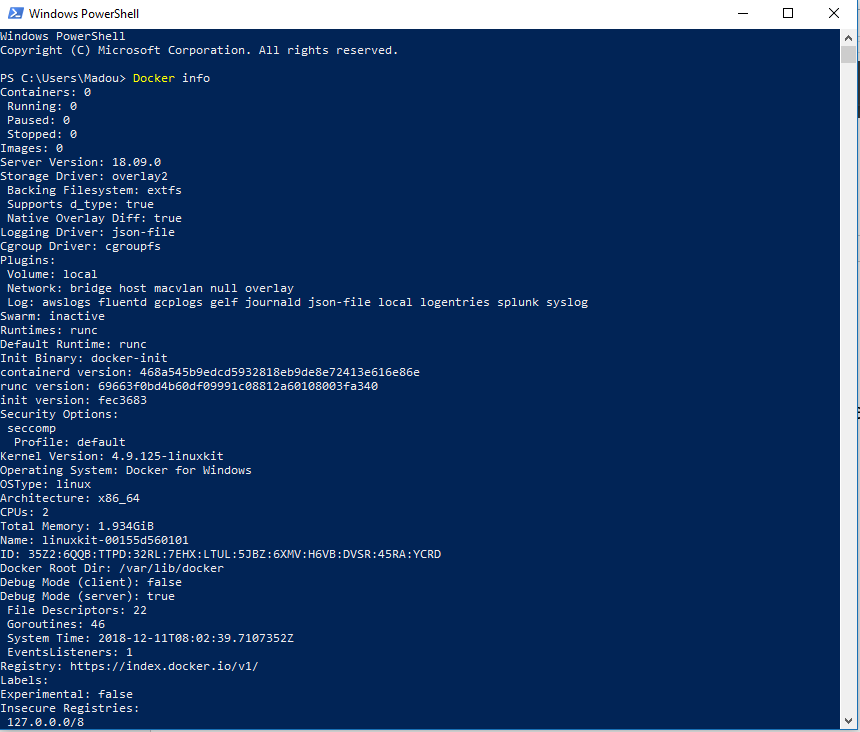


Thus in intelIj we can add a plugin called SonarLint that do scan the code for issues and errors



Now to Docker:

After downloading Docker for windows and installing it we need to check the installation by opening a Windows PowerShell and type “Docker info”



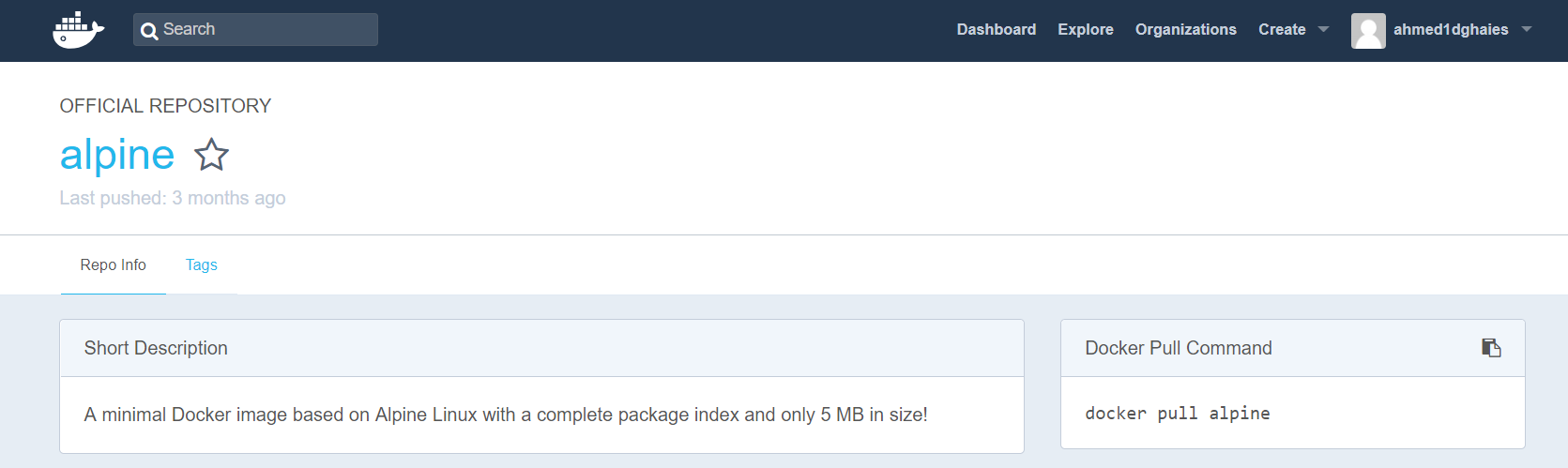
We can see here that we are using Docker on windows but the OSType is linux

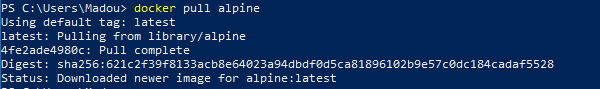


For the time being we have an empty Docker:



Now let’s look for a small image to download and try going to hub.docker.com

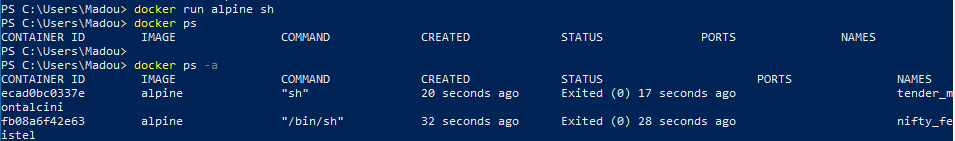




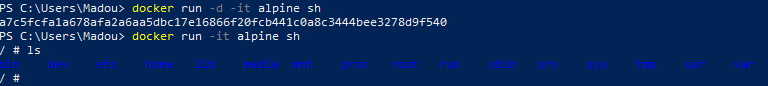
Now we can see the new images



Now if we tried to run the alpine image it will be closed just after running it, because we didn’t run the docker container in the interacting mode



So we need to run this command instead “docker run -d -it alpine sh” and then “docker run -it alpine sh”. Now we can run linux commands:



We can give a simple task for that container like echo “hello”

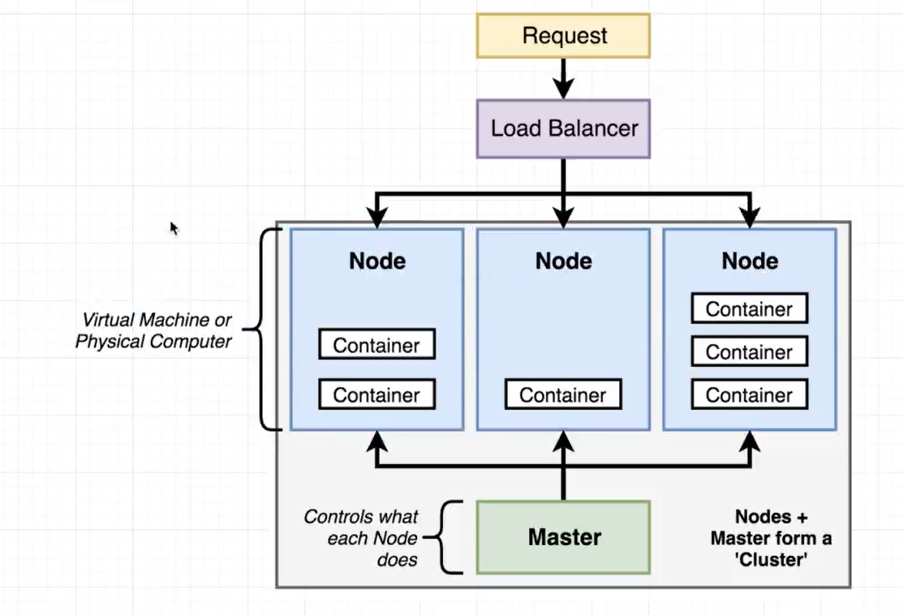


Now let’s move on to kubernetes, but first we need to answer two question:   
1- What is kubernetes:

Kubernetes is a system for running many different containers over multiple different machines

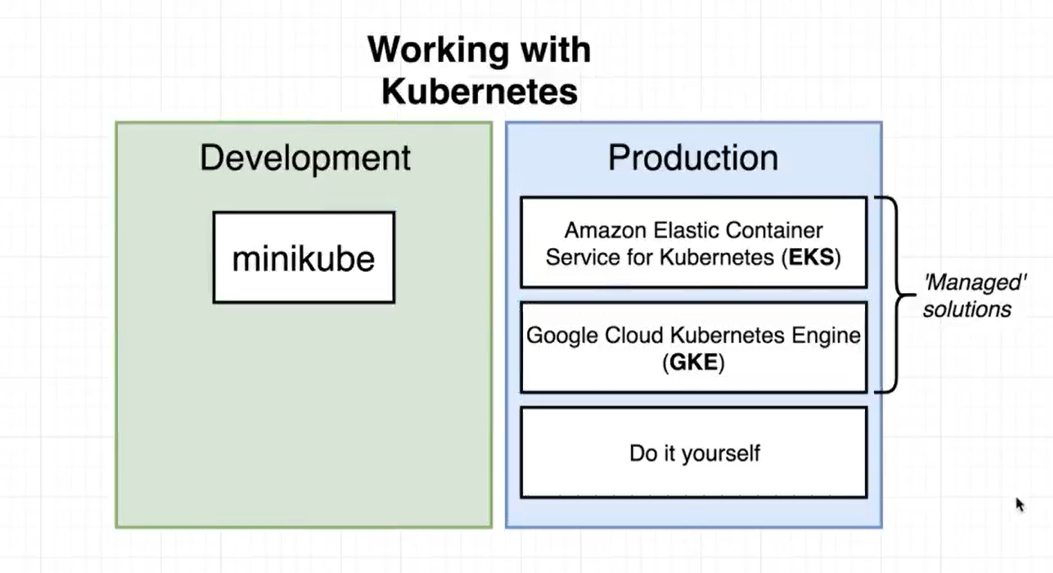
2- Why use kubernetes:

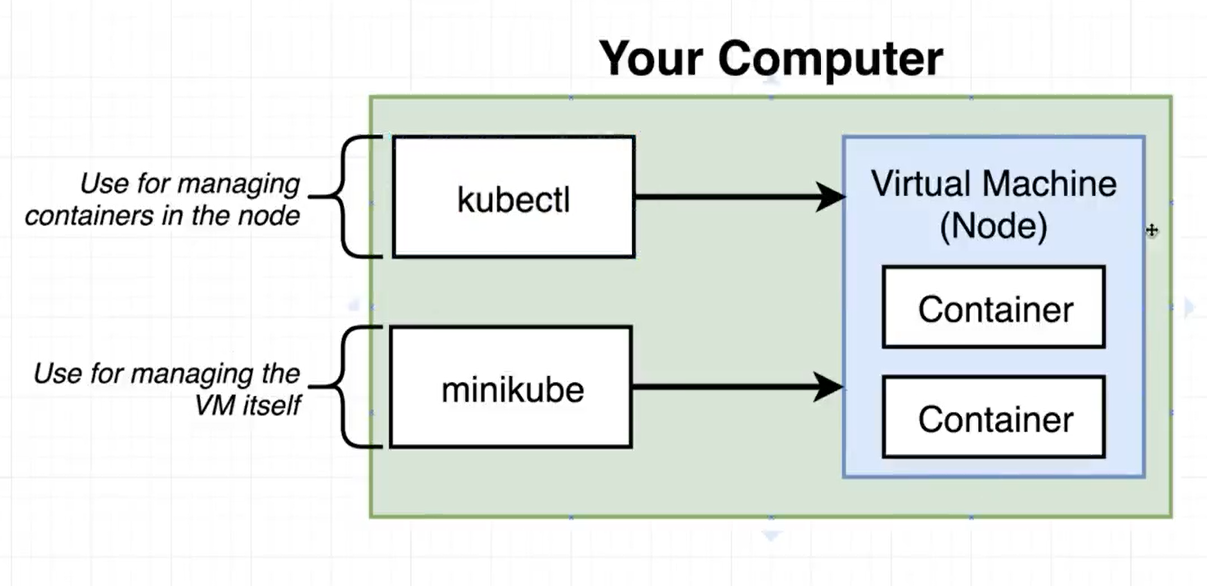
When we need to run many different containers with different images.



This figure is a diagram of a kubernetes cluster. It contains a certain number of nodes that represent a virtual machine or a physical computer, these nodes are controlled by a master. Then we have the load balancer that receives request and assign them to nodes

Now let’s move on to how we can use kubernetes. Working with kubernetes can be divided into two big categories, working locally in the development process or during the production, the next figure can explain what each one needs





We are going to use kubectl any time that we want to manage what the node is doing, but minikube is used to just create and run a local cluster.