Easy App Deployment Environment

Fog and Cloud Computing 2021 - Group 19
Claudio Facchinetti <claudio.facchinetti@studenti.unitn.it>
Matteo Franzil <matteo.franzil@studenti.unitn.it>

May 16, 2021

1 Introduction

This document outlines the idea for the final project from group 19 for the FCC-21 course.
Some time ago deployment was done on bare metal by developers. This approach shortly showed its limits: managing replication, load balancing, and more proved convoluted. In the last years a new cloud based approach took over, with virtualization and orchestration solutions being provided by big market players such as Amazon or Microsoft.
The goal of this project is to outline an architecture offering some of the key services available on the market while showing how simple it is to adapt a legacy application to this context, by applying the concepts shown in class.

2 Sample Application

Our first building block will be an application developed by Matteo Franzil as the final project for a web programming course. It was originally devised to run on a Tomcat application server on bare metal, with no cloud approach in mind.
Its purpose is to allow citizens to access their sanitary information in an intuitive and simple portal while allowing doctors, chemists' and local sanitary services to interact with patients, prescriptions, and more. It integrated features such as Excel report generation, prescription PDF generations with QR codes, and more.
The application is logically divided into three components. Their deployment will be shown in later sections: the **web server**, handling the logic and providing the HTML dynamic pages, the **database**, for storing data about patients, doctors, prescriptions and so on, and the **resources**, like reports, prescriptions, and patient images.

3 Platform as a Service

In this section we are explaining how we are applying the concepts and technologies gathered during the PaaS laboratory.

We plan to create both a new user and a new cluster, in order to simulate a company working on a fresh Kubernetes environment.

3.1 Kubernetes

We plan to use the PaaS machine as a resource orchestrator using Kubernetes. We will create two main ReplicaSets:

- **web server** RS: containing 3 to 5 pods and an alias, in order to ensure maximum performance and load balancing, depending on usage;
- **database** RS: containing just one pod for the moment. We are locked to using Postgres as a DBMS: if we were to switch to a distributed or NoSQL DBMS we may scale number of pods to 2 and include an alias and a load balancer. In either case we plan to deploy a volume on Kubernetes for data persistency through the pod's destruction and re-creation.

Of course, both pods will feature "packaged" web server and database docker images so that they can be completely decoupled. Finally, we are still investigating whether to deploy a single or two different namespaces (one for testing and one for production), and we may plan to stick to two in order to demonstrate how flexible Kubernetes is and how easy it is to create two namespaces for the same application but with just a different purpose.
Depending on the result of the DNS Resolver findings we could also create an ingress for the load balancer service for the production deployment "Web Server" so that it is accessible from the outside world via a URL.

4 Infrastructure as a Service

In this section we are explaining how we are applying the concepts and technologies gathered during the IaaS laboratory.

We are going to create a new user from scratch for the OpenStack management and also a new associated project, so that we will simulate someone working on a clean OpenStack project; we will only setup the `PaaS` and `IaaS` connectivity in advance.

4.1 Local artifact manager

Since using Kubernetes implies the usage of Docker images, we wanted to avoid storing them on the `PaaS` machine - as we want it to be fully dedicated to our web server and database for maximum performance.
We therefore plan to use OpenStack to deploy an instance which will serve as an *image manager*, enabling us to create the images and push them to the instance via the command.
For this we are currently exploring alternatives: we found some commercial software, such as JFrog, and also some open source software like Harbor and Docker itself; we are exploring the features of every artifact manager and will choose one that will best fit our needs on both performance and security.
This instance will be running a currently undefined version of Ubuntu and will be exposed via a floating IP and/or a static URL, depending on the result of our findings about the DNS Resolver.

4.2 DNS resolver

Since in 2021 proposing a Web Application demo with raw IP addresses connections sounded a bit old school, we plan to deploy a DNS Resolver to make all the services accessible as URLs and not as IP address. Our idea is that every instance on the project will be reachable through a `FQDN`, including both machines, the web server, the database, and so forth, enabling flexibility on app deployment and resource utilization.
This is the part about which we are more worried: we are currently exploring the capabilities of the *Neutron* component of OpenStack and deciding whether it is better to use it or to deploy a dedicated instance, running probably a Debian-based Linux distribution; of course this will need to have a Floating IP in order to make it accessible from the outside.

4.3 Resource server

This will be the last component of the web application: it will an instance deployed on OpenStack with a volume attached containing all the resources (e.g. images, PDFs, reports...). It will assist the web server in delivering content without setting up a whole NoSQL database. Depending on our findings on the DNS Resolver it may be associated with a Floating IP and/or to a public URL through the resolver.

4.4 Flavors and resource usage

We plan to use the following, custom-defined flavors for the three machines. In all three cases, the ephemeral disk and swap values will be set to `0`, and the root disk size will be small to allocate larger sizes for the volumes. CPU overcommitting allows us to use more than the four cores attached to the `IaaS` machine. The artifact manager needs balanced performance, a DNS server is usually low on RAM requirements but multiple `vCPUs` might help, and the resource server has higher overall requirements in order to provide better performance to the web server.

Name	`art-flv`	`dns-flv`	`res-fv`
VCPUs	`2`	`4`	`6`
Memory	`2GB`	`1GB`	`4GB`
Root Disk	`4GB`	`2GB`	`4GB`

Attachment A Diagrams

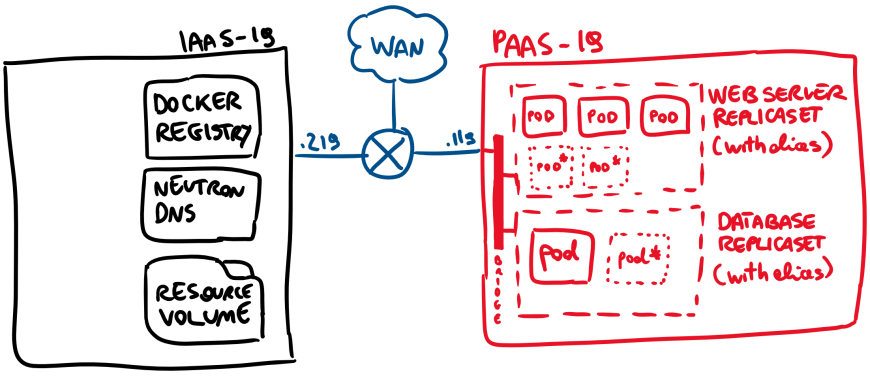

Figure 1 Diagram of the environment, showing the two machines and internal structure

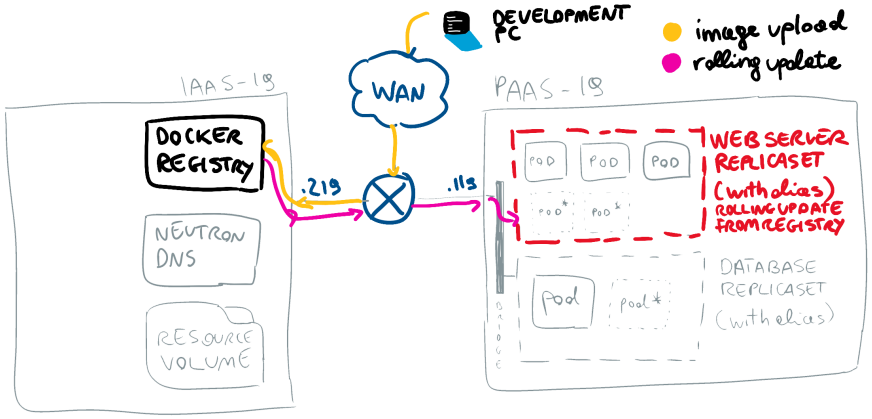

Figure 2 Involved components during a code deployment and rolling update for the web server.

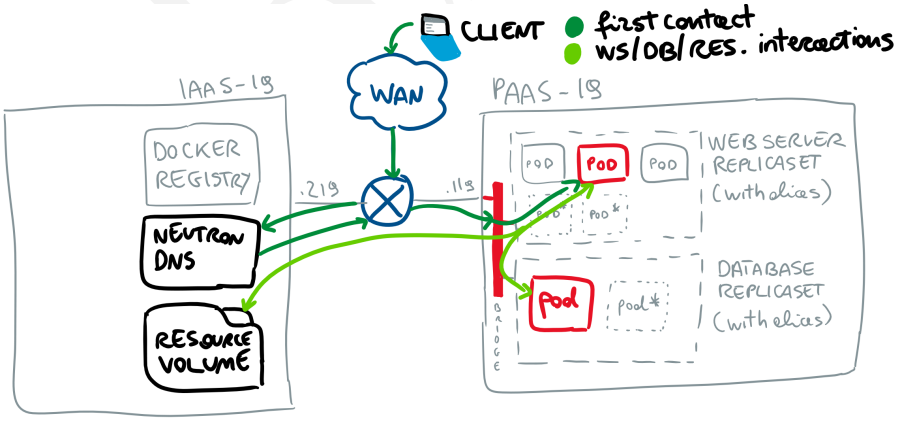

Figure 3 Involved components during day-to-day server operations.