UNIT 6.
WEB APPLICATIONS DEPLOYMENT

Web Applications Deployment
CFGS DAW

Author: Carlos Cacho López

Reviewed by: Lionel Tarazón Alcocer

lionel.tarazon@ceedcv.es

2019/2020

License

Nomenclature

In this unit we are going to use special symbols for some important content.

These symbols are:

INDEX

UT06. WEB APPLICATIONS DEPLOYMENT

1. INTRODUCTION

Deploying a web application means putting it in a production environment, that is, making it available for real users, companies, etc. In this unit we are going to learn how to deploy web applications in different environments.

2. WEB HOSTING

Web hosting is a service that provides Web users with online systems for storing information, images, videos or any content available via web. Web hosts are companies that provide space on a server they own for use by their clients.

First we are going to see “Traditional” web hosting and afterwards the “Cloud” hosting or computing.

2.1 Traditional hosting

There are three types of traditional web hosting:

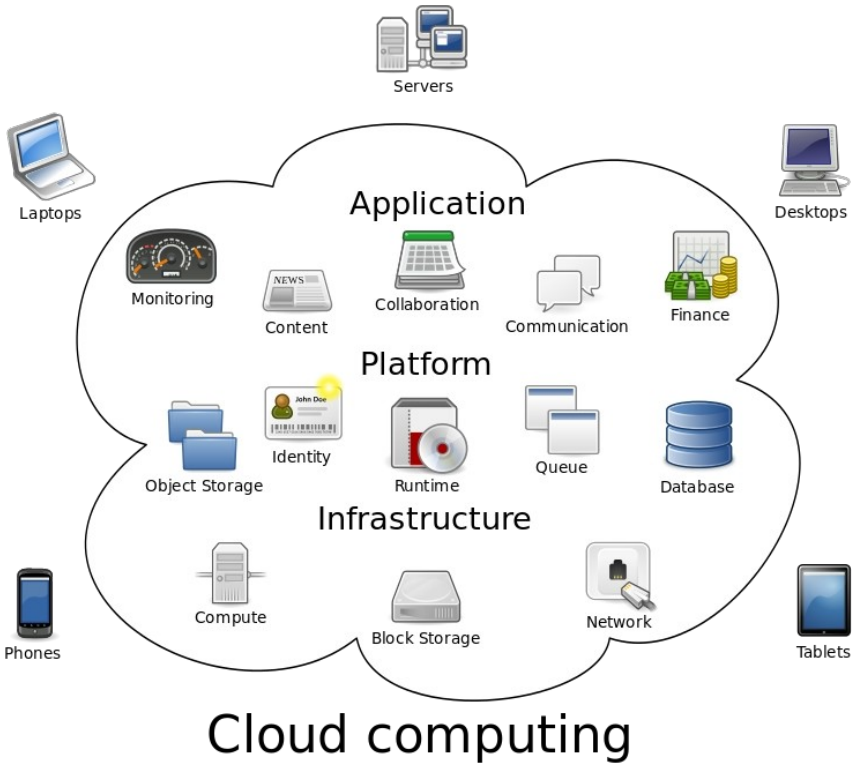
- **Shared**: many websites from different users allocated in the same web server.
- **Dedicated**: the client leases an entire physical server not shared with anyone else.
- **Virtual**: the client leases a virtual server (running in a physical server with other virtual servers).

Examples of these type of hostings:

- https://www.1and1.es/
- https://www.arsys.es/
- https://www.hostalia.com/

2.2 Cloud Computing

Cloud computing is a model that offers different services over the Internet. In this image we can see those kind of services: Application, Platform and Infraestructure.



By Sam Johnston - Created by Sam Johnston using OmniGroup's OmniGraffle and Inkscape (includes Computer.svg by Sasa Stefanovic)This vector image was created with Inkscape., CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=6080417

The big difference with the traditional hosting is that the user has more control over the different resources (servers, operating systems, copy backups, etc.). This can be made manually using a terminal or a mobile app or manage it automatically programming an API REST.

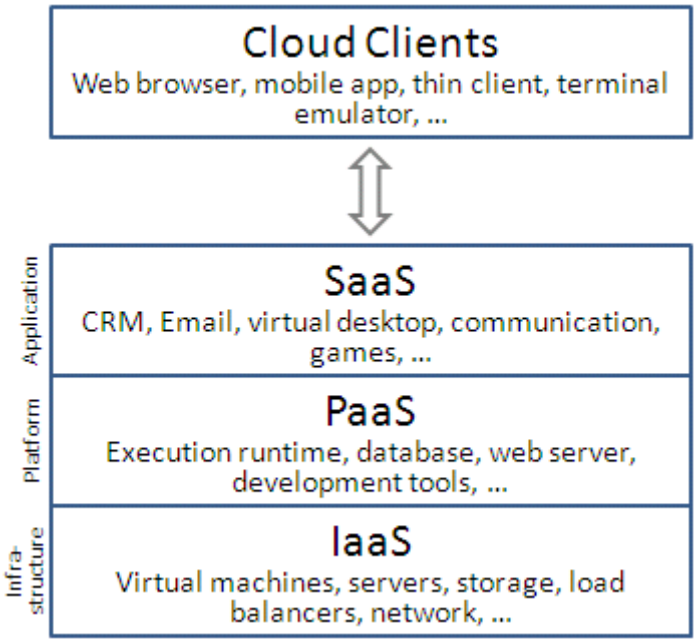
Some providers are:

- https://aws.amazon.com
- https://cloud.google.com
- https://azure.microsoft.com

Click on this video which briefly explains what is cloud computing and Amazon Web Services: https://www.youtube.com/watch?v=jOhbTAU4OPI

The servers offered by providers can be:

- High level: Software as a service (SaaS).
- Medium level: Platform as a service (PaaS).
- Low level: Infrastructure as a service (IaaS).



Public Domain, https://commons.wikimedia.org/w/index.php?curid=18327835

2.2.1 Infrastructure as a Service

Infrastructure as a Service (IaaS) refers to online services that provide high-level APIs used to dereference various low-level details of underlying network infrastructure like physical computing resources, location, data partitioning, scaling, security, backup etc.

Some of these services are:

- Servers (called instances).
- Load balancer.
- Operating Systems manage.
- DNS servers.

One of the most famous providers is Amazon Web Services (AWS).

2.2.2 Platform as a Service

Platform as a Service (PaaS) offers a platform to execute applications as web servers, data bases, monitoring, etc. In this level the consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

Some providers are *Amazon Web Services (AWS)*, *Heroku*, *OpenShif* or *Google App Engine*.

2.2.3 Software as a Service

Software as a Service (SaaS) offers different web applications to the final users. These applications are accessible from various client devices through either a thin client interface, such as a web browser or a program interface. Some examples are Google Apps, Microsoft Office 360 or Salesforce.com.

2.2.4 Container as a Service

Other special model is Container as a service (CaaS) which gives to the users the possibility to manage and deploy containers, applications and clusters through container-based virtualization. One possibility for this service is Docker.

2.2.4.1 Docker

Docker is a software tools thats make it easier to create, deploy, and run applications by using containers. Containers allow you to put together an application with all of the parts it needs in only one package.

Docker is similar to a virtual machine but it doesn't need to contain its own operating system because Docker allows applications to use the Linux kernel of the system where they are running. Therefore, applications in Docker are a lot more lightweight (require less CPU and memory resources). Docker applications contain everything they need to run and nothing else.

Docker uses images and containers. It is important to understand the difference:

- An **image** has everything necessary to run an application: code, libraries, configuration files, etc.
- A **container** is an instance of an image. Everytime we run an image a container is created (so if we run an image twice we will have two different containers).

We can find all the information in the official web page: https://docs.docker.com/get-started/

2.2.4.2 Docker Commands

Some important Docker commands:

- ***docker --version***: shows the Docker version installed.
- ***docker version*** or ***docker info***: shows more information about our Docker installation.
- ***docker run <image>***: runs a docker image in a new container (if we run the image twice we will have two different containers).
- ***docker run <image> -it***: runs the image in interactive mode (allows us to interact with the running container using a shell).
- ***docker run -p port1:port2 <image>***: runs a docker image and creates a container mapping port port1 to port2.
- ***docker image ls*** or ***docker images***: lists the images we have in our computer.
- ***docker image rm <image>***: remove the specified image of our computer.
- ***docker build -t <image> .***: creates an image using this directory's Dockerfile(the dot (.) is the current directory).
- ***docker pull <image>***: downloads the image (not the container).
- ***docker container ls*** or ***docker ps***: lists all running containers.
- ***docker container ls -a*** or ***docker ps -a***: lists all containers, even those not running.
- ***docker container start <id_container>***: starts the container with a specific id. We can get the id running the previous command.
- ***docker container start <id_container> -i***: starts the container with that id_container and we could access to it using the shell.
- ***docker exec -it <id_container> /bin/bash***: accesses to the shell of the container with that id_container.
- ***docker container stop <id_container>***: stops the specified container.
- ***docker container rm <id_container>***: remove the specified container of our computer.

Tip: Some of these commands need ***sudo***

2.2.4.3 Dockerfile

A Dockerfile is a simple text document where we can write all the desired Docker commands requiered to create a Docker image. This allows you to build build Docker images easily.

The file format is vey easy, it's just a list of *Instruction Arguments*

The most common instructions are:

- ***FROM <image>***: initializes a new build stage and sets the base of the image. For instance: FROM ubuntu:18.04
- ***RUN <command>***: runs the command. For instance: RUN apt-get update -y
- ***CMD ["executable", "param1", "param2", ...]***: default executable to run if the container is run without specifying a command. Only one CMD instruction is allowed. For instance: CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]
- ***EXPOSE <port>***: indicates the port where container listens. For instance: EXPOSE 80
- ***ENV <key value>*** or ***ENV <key>=<value>***: sets the environment variable (key) to the value. For instance: ENV APACHE_RUN_USER www-data
- ***ADD <source> <destination>***: copy files, directories or URLs from a source to a destination path in the filesystem of the image. For instance: ADD src /var/www/html
- ***COPY <source> <destination>***: copy files or directories from a source to a destination path in the filesystem of the container. For instance: COPY package.json ./
- ***WORKDIR <path>***: sets the working directory for do instructions as RUN, CMD, COPY or ADD in the Dockerfile.

In this link you can find all the instructions https://docs.docker.com/engine/reference/builder/

3. BIBLIOGRAPHY

[1] Gallego, Micael (2014): *Desarrollo de aplicaciones Web.*

[2] https://en.wikipedia.org/wiki/Cloud_computing

[3] https://docs.docker.com/