Cloud !

Πίνακας περιεχομένων

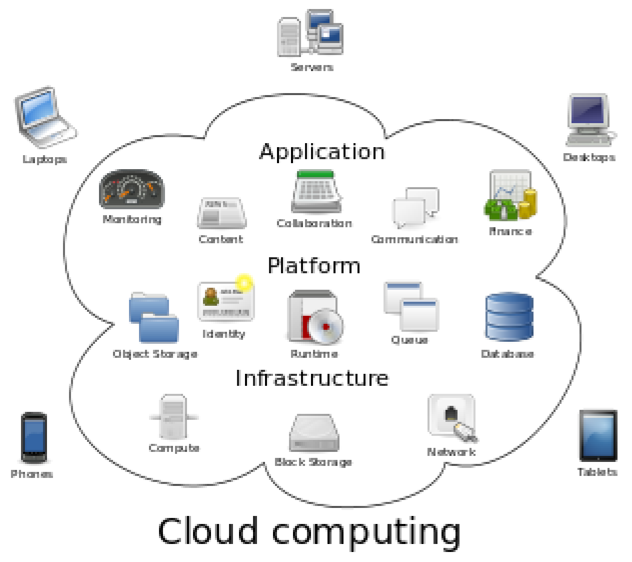
1. Cloud - Intro

Cloud computing is the **on-demand availability of computer system resources**, especially data storage and computing power, without direct active management by the user. The term is generally used to describe data centers available to many users over the Internet.

Large clouds, predominant today, often have functions distributed over multiple locations from central servers. If the connection to the user is relatively close, it may be designated an edge server.

Clouds may be limited to a single organization (enterprise clouds), or be available to many organizations (public cloud).

Cloud computing relies on sharing of resources to achieve coherence and economies of scale.



From Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/Cloud_computing

1.1. Cloud Computing Tutorial for Beginners

- Cloud Computing Tutorial for Beginners

  ▶ https://www.youtube.com/watch?v=RWgW-CgdIk0 *(YouTube video)*

2. Cloud computing architecture

Cloud computing architecture refers to the components and subcomponents required for cloud computing. These components typically consist of a front end platform (fat client, thin client, mobile device), back end platforms (servers, storage), a cloud based delivery, and a network (Internet, Intranet, Intercloud). Combined, these components make up cloud computing architecture.

2.1. Virtualization

In computing, virtualization refers to the act of creating a virtual (rather than actual) version of something, including virtual computer hardware platforms, storage devices, and computer network resources.

2.2. Containerization

Containerization has become a major trend in software development as an alternative or companion to virtualization. It involves encapsulating or packaging up software code and all its dependencies so that it can run uniformly and consistently on any infrastructure. The technology is quickly maturing, resulting in measurable benefits for developers and operations teams as well as overall software infrastructure.

2.3. Virtual Machines vs Docker Containers

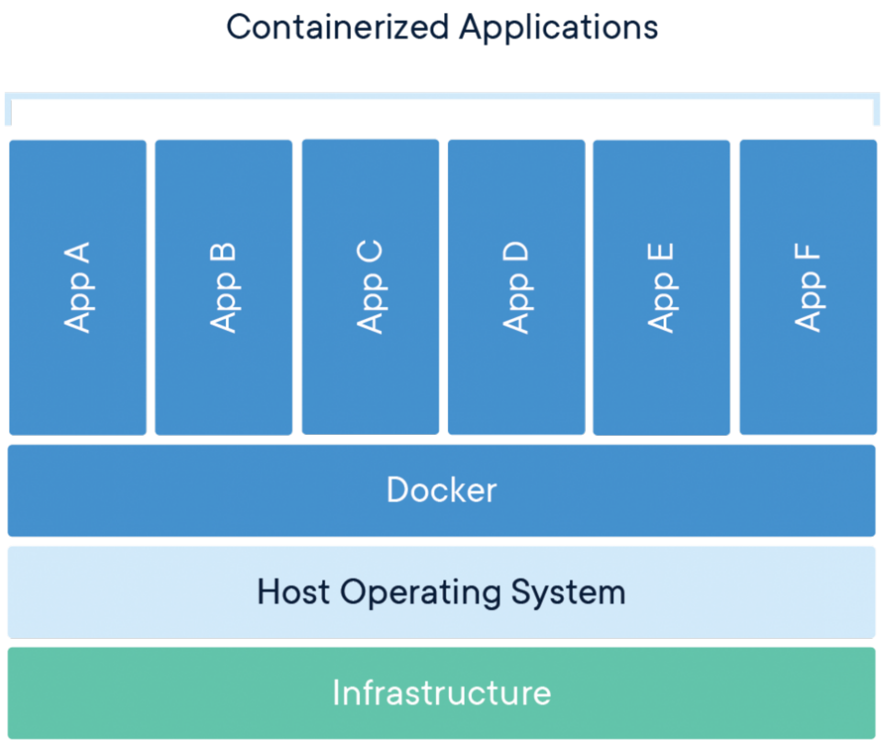
- A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it.

Docker is the service to run multiple containers on a machine (node) which can be on a vitual machine or on a physical machine.

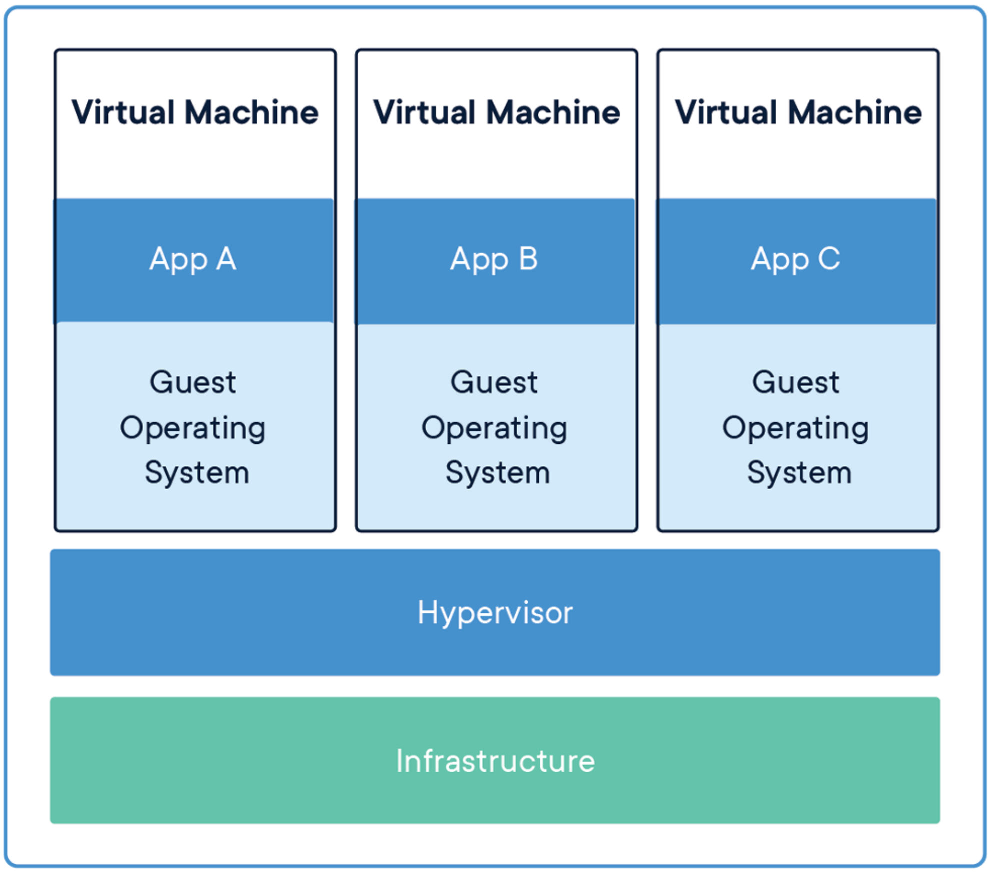
- A virtual machine is an entire operating system (which normally is not lightweight).
    - Virtual Machines vs Docker Containers

      ▶ https://www.youtube.com/watch?v=TvnZTi_gaNc *(YouTube video)*

2.3.1. Docker Containers



2.3.2. Virtual Machines



3. Orchestration

Container orchestration automates the deployment, management, scaling, and networking of containers. Enterprises that need to deploy and manage hundreds or thousands of Linux® containers and hosts can benefit from container orchestration. Container orchestration can be used in any environment where you use containers. It can help you to deploy the same application across different environments without needing to redesign it. And microservices in containers make it easier to orchestrate services, including storage, networking, and security.

Containers give your microservice-based apps an ideal application deployment unit and self-contained execution environment. They make it possible to run multiple parts of an app independently in microservices, on the same hardware, with much greater control over individual pieces and life cycles.

Managing the lifecycle of containers with orchestration also supports DevOps teams who integrate it into CI/CD workflows. Along with application programming interfaces (APIs) and DevOps teams, containerized microservices are the foundation for cloud-native applications.

Container orchestration used for:

- Provisioning and deployment
- Configuration and scheduling
- Resource allocation
- Container availability
- Scaling or removing containers based on balancing workloads across your infrastructure
- Load balancing and traffic routing
- Monitoring container health
- Configuring applications based on the container in which they will run
- Keeping interactions between containers secure

3.1. Kubernetes vs Docker Swarm

- Kubernetes vs Docker Swarm

  ▶ https://www.youtube.com/watch?v=FmrAGliHvzQ *(YouTube video)*

3.2. Technical Comparisons

Kubernetes	Docker Swarm
Reported Scalability	
5,000 Node Cluster With 150,000 pods	1,000 Node Cluster With 30,000 containers
Service Discovery	
• Services, each given a VIP, group Pods together • These services can be reached through environment variables provided in a pod. For example: {SVCNAME_SERVICE_HOST} • DNS is optional. This includes the third party offerings such as SkyDNS, CoreDNS	• Docker Engine, while participating in the Swarm, becomes a service registry • Two Methods for Discovery • 1) Relies on embedded DNS and virtual IP and operates on Network and Transport Layer • 2) Application Layer DNS round robin with third party support
Networking	
• Kubernetes uses an overlay network that lets pods communicate across multiple nodes • Containers inside pods share a network stack and can communicate via local host • Services group together pods via a network proxy and are assigned a virtual IP	• Overlay networks connect Docker daemons and use the overlay network driver • Services connect to this overlay network • Uses a customizable ingress network to load balance among service - the module which does this is called IPVS

Persistent Data Volumes

• Volumes are directly attached to pods and share its lifecycle • Containers in Pod can access volume • Provides support for multiple types of Volumes: nsf, awsElasticBlockStore, configmap, vSphereVolume, and others	• Two methods of persistent storage which are familiar to Docker users. • 1) Bind Mounts: File or directory from the machine is mounted onto containers • 2) Volumes: Mountable storage managed by docker itself that exist beyond container lifecycle

Monitoring and Logging

• Comprehensive solution for monitoring and logging included with Kubernetes itself • Default Kubernetes GUI for logging and monitoring • Option to add third party solutions	• Does not include default solution for logging and monitoring • Many third party solutions that can be integrated with Docker Swarm - yet complexity of implementation varies

High Availability

• High Availability Supported • Services perform health checks directly on pods • Workload Objects (Replication Controller, Deployments, etc..) can be deployed across multiple nodes • Etcd is run across multiple nodes and can tolerate failures	• High Availability Supported • Perform health checks on docker daemon hosts • Restarts containers on a new host if host failure occurs. • Uses RAFT algorithm ensure cluster fault tolerance (requires majority of nodes to be available)

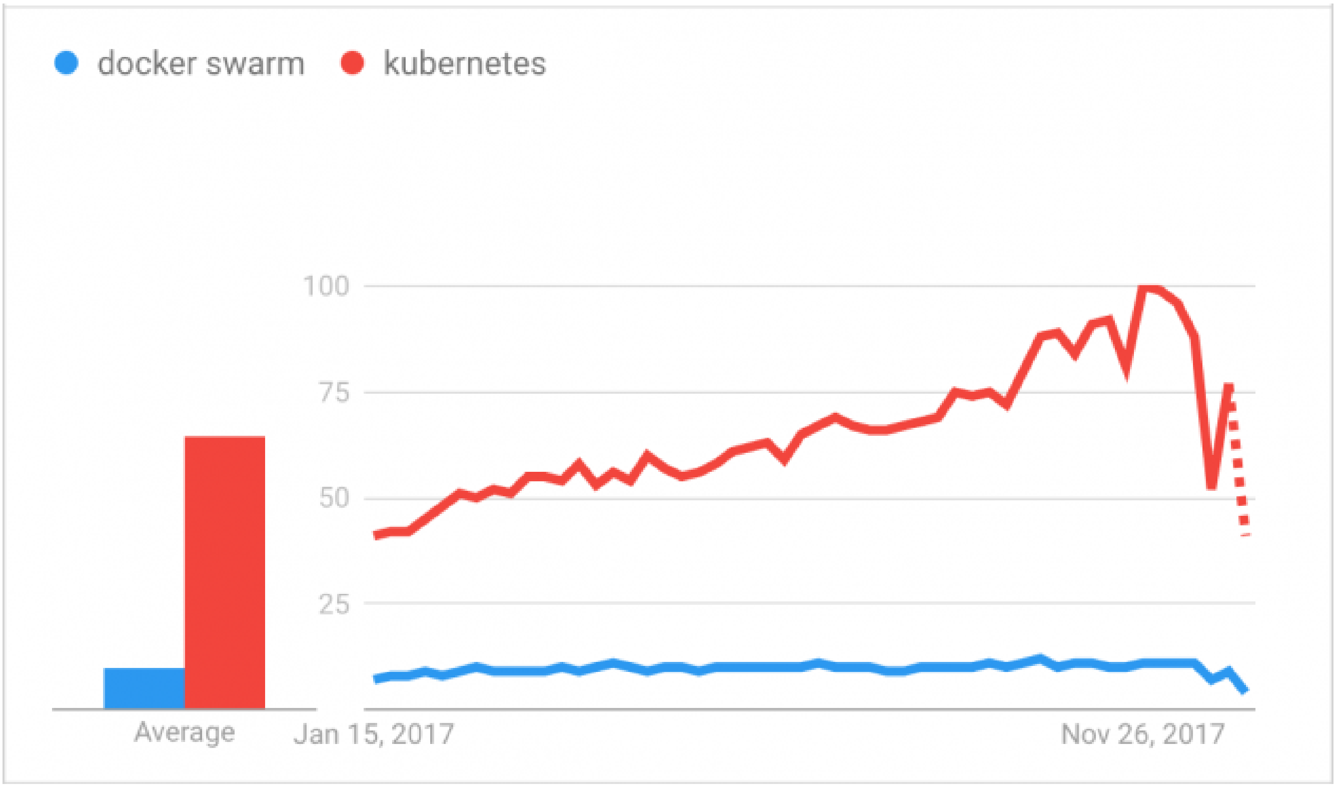
3.3. Conclusion

When comparing Docker Swarm vs Kubernetes, it becomes apparent that the origins of both platforms have played key roles in shaping their features and communities today.

Docker, realizing the strength of its container technology, decided to build a platform that made it simple for Docker users to begin orchestrating their container workloads across multiple nodes. However, their desire to preserve this tight coupling can be said to have limited the extensibility of the platform.

Kubernetes, on the other hand, took key concepts taken from Google Borg, and, from a high level perspective, decided to make containerization fit into the former platform's existing workload orchestration model. This resulted in Kubernetes emphasis on reliability, sometimes at the cost of simplicity and performance.

3.4. Popularity of searches for each platform



Origin: https://www.nirmata.com/2018/01/15/orchestration-platforms-in-the-ring-kubernetes-vs-docker-swarm

3.5. Short_answer

Info

- Docker:![1]
  - Docker is the container technology that allows you to **containerize your applications.**
  - Docker is **the core of using other technologies.**
- Docker Compose
  - Docker Compose allows configuring and starting **multiple Docker containers.**
  - Docker Compose is mostly used as a helper when you want to start multiple Docker containers and doesn't want to start each one separately using docker run ….
  - Docker Compose is used for **starting containers on the SAME host.**
  - Docker Compose is used instead of all optional parameters when building and running a single docker container.
- Docker Swarm
  - Docker swarm is for **running** and connecting containers on **multiple hosts.**
  - Docker swarm is a **container cluster management and orchestration tool.**
    - It manages containers running on multiple hosts and does things like scaling, starting a new container when one crashes, networking containers …
  - Docker swarm is docker in **production.**
  - It is the **native docker orchestration tool** that is embedded in the Docker Engine.
  - The docker swarm file named stack file is very similar to a Docker compose file.
- Kubernetes
  - Kubernetes is a **container orchestration tool** developed by Google.
  - Kubernete's goal is **very similar** to that for Docker swarm.

Update

Docker support docker stack deploy --orchestrator=kubernetes options

4. Docker

4.1. Images

In Docker, everything is based on Images. An image is a combination of a file system and parameters.

4.1.1. Dockerfile

A Dockerfile is a simple text file that contains a list of commands that the Docker client calls while creating an image. It's a simple way to automate the image creation process. The best part is that the commands you write in a Dockerfile are almost identical to their equivalent Linux commands. This means you don't really have to learn new syntax to create your own dockerfiles.

Dockerfile

```
FROM ubuntu:16.04
ENV DEBIAN_FRONTEND noninteractive
RUN apt-get update -y && \
    apt-get -y install gcc && \
    rm -rf /var/lib/apt/lists/*
```

4.1.2. docker build

docker build

```
docker build  -t ImageName:TagName dir

Options

    -t □ is to mention a tag to the image

    ImageName □ This is the name you want to give to your image.

    TagName □ This is the tag you want to give to your image.

    Dir □ The directory where the Docker File is present.
```

docker build example

```
docker build □t myimage:0.1 .
```

4.1.3. Displaying Docker Images

To see the list of Docker images on the system, you can issue the following command.

docker images

```
docker images
```

This command is used to display all the images currently installed on the system.

Output:

- TAG – This is used to logically tag images.
- Image ID – This is used to uniquely identify the image.
- Created – The number of days since the image was created.
- Virtual Size – The size of the image.

4.1.4. Removing Docker Images

The Docker images on the system can be removed via the docker rmi command.

docker images

```
docker rmi

This command is used to remove Docker images.
Syntax

docker rmi ImageID
```

4.1.5. Docker Hub

Docker Hub is a registry service on the cloud that allows you to download Docker images that are built by other communities. You can also upload your own Docker built images to Docker hub.

To run apache, you need to run the following command:

run docker image from Docker Hub

```
docker run -p 8080:80 apache

Note the following points about the above  command 


    Here, apache is the name of the image we want to download from Docker hub and
install on our Ubuntu machine.

    -p is used to map the port number of the internal Docker image to our main Ubuntu
server so that we can access the container accordingly.
```

4.2. Containers

Containers are instances of Docker images that can be run using the Docker run command. The basic purpose of Docker is to run containers.

4.2.1. Running a Container

Running of containers is managed with the Docker run command. To run a container in an interactive mode, first launch the Docker container.

run docker image

```
docker run –it myimage /bin/bash
```

4.2.2. Listing of Containers

One can list all of the containers on the machine via the docker ps command. This command is used to return the currently running containers.

run docker image

```
docker ps
```

4.2.3. Display the running processes of a container

With this command, you can see the top processes within a container. Syntax

docker top

```
docker top ContainerID

Options

    ContainerID – This is the Container ID for which you want to see the top
processes.
```

4.2.4. Stop a running container

This command is used to stop a running container.

docker stop

```
docker stop ContainerID

Options

    ContainerID – This is the Container ID which needs to be stopped.
```

4.2.5. Attach a running container

This command is used to attach to a running container.

docker

```
docker attach ContainerID

Options

    ContainerID  This is the Container ID to which you need to attach.
```

4.2.6. Delete container

This command is used to delete a container.

docker rm

```
docker rm ContainerID

Options

    ContainerID  This is the Container ID which needs to be removed.
```

4.2.7. Container Logging

Logging is also available at the container level.

docker log

```
Docker logs containerID

Parameters

    containerID  This is the ID of the container for which you need to see the logs.
```

4.3. Volumes

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers.

docker volumes

```
docker run -d --name mycontainer -v /var/www/html:/var/html nginx:latest
```

4.4. repositories

You might have the need to have your own private repositories. You may not want to host the repositories on Docker Hub. For this, there is a repository container itself from Docker. Let’s see how we can download and use the container for registry.

4.4.1. Create

docker registry

```
docker run □d □p 5000:5000 □-name registry registry:2

The following points need to be noted about the above command:

    Registry is the container managed by Docker which can be used to host private
repositories.

    The port number exposed by the container is 5000. Hence with the □p command, we
are mapping the same port number to the 5000 port number on our localhost.

    We are just tagging the registry container as □2□, to differentiate it on the
Docker host.

    The □d option is used to run the container in detached mode. This is so that the
container can run in the background
```

4.4.2. Push

use the Docker push command to push the image to our private repository.

docker registry

```
docker push localhost:5000/myimage
```

4.4.3. Pull

use the following Docker pull command to pull image from our private repository.

docker registry

```
docker pull localhost:5000/myimage
```

Reminder

Caminante, no hay camino,
se hace camino al andar.

Wanderer, there is no path,
the path is made by walking.

Antonio Machado Campos de Castilla

[1] origin info