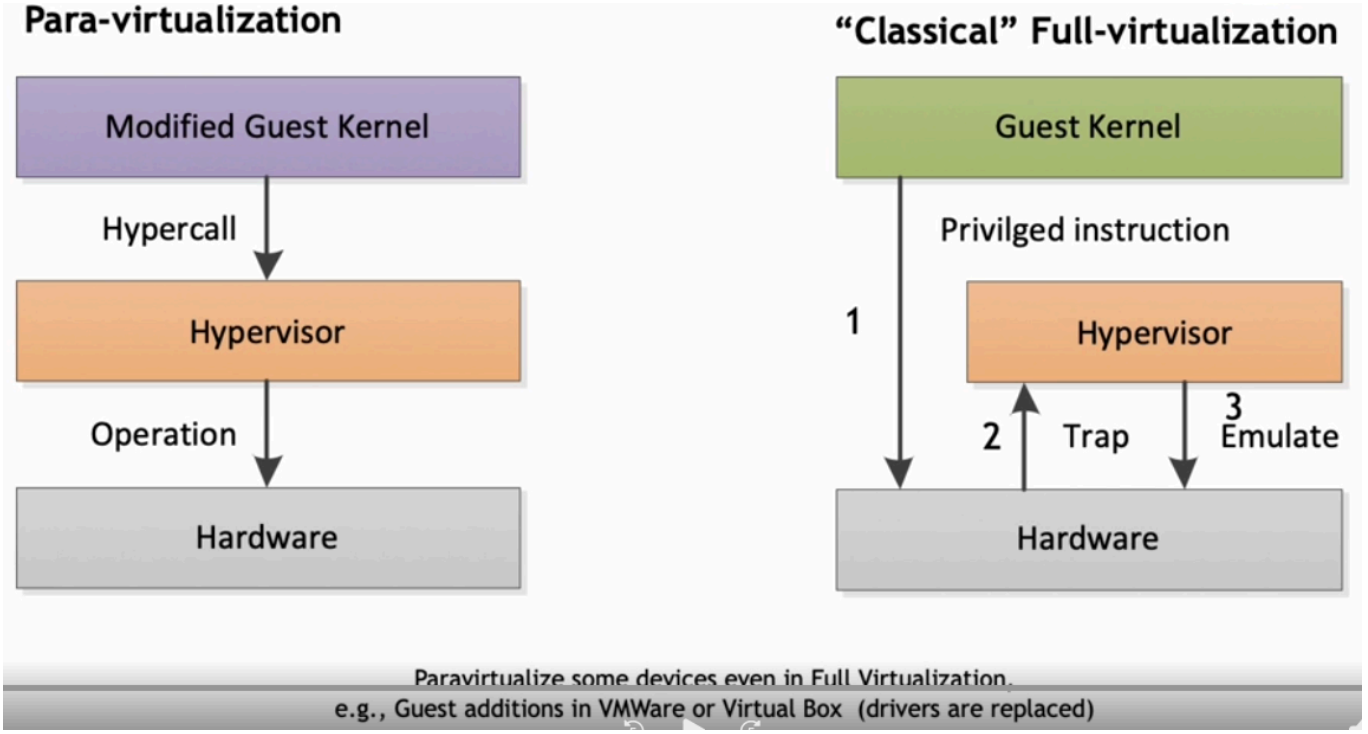
Containers

Different way to implement virtualization, the focus by commercial products is on system level same ISA because the overhead by the system is limited/negligible.
Two way to realize it:

- Paravirtualization: the HW is not assumed so there is needed collaboration between Guest OS and Host OS, Guest applications need to be modified by the Host to be runned
- Full Virtualization: we have almost a complete simulation of the actual HW, in a way that the Guest OS run unmodified
  In Paravirtualized models Host and Guest OS collaborate, core idea is to reduce the operation runned by Guest OS to access resources, instead raise a call to the hypervisor to have a straightforward access to the disk(simplified access), almost same performance than baremetal. The other side of the coins is that with the modification of the Guest OS you have to raise API calls to the hypervisor(can have problem if they aren't open source). The other solution is to run in full simulation, the Guest OS has access to all the instructions to access memory. The advantages is that you can run a full OS but there will be performance degradation as the hypervisor has to mediate the access of the Guest OS to the resources and there could be incompatability with some piece of HW.



Today there are some HW extension for full virtualized systems.

Containers

Started from an analysis of the overhead introduced by hypervisor. Running multiple VM on a same machine introduce some performance degradation. Some chunks of resources are

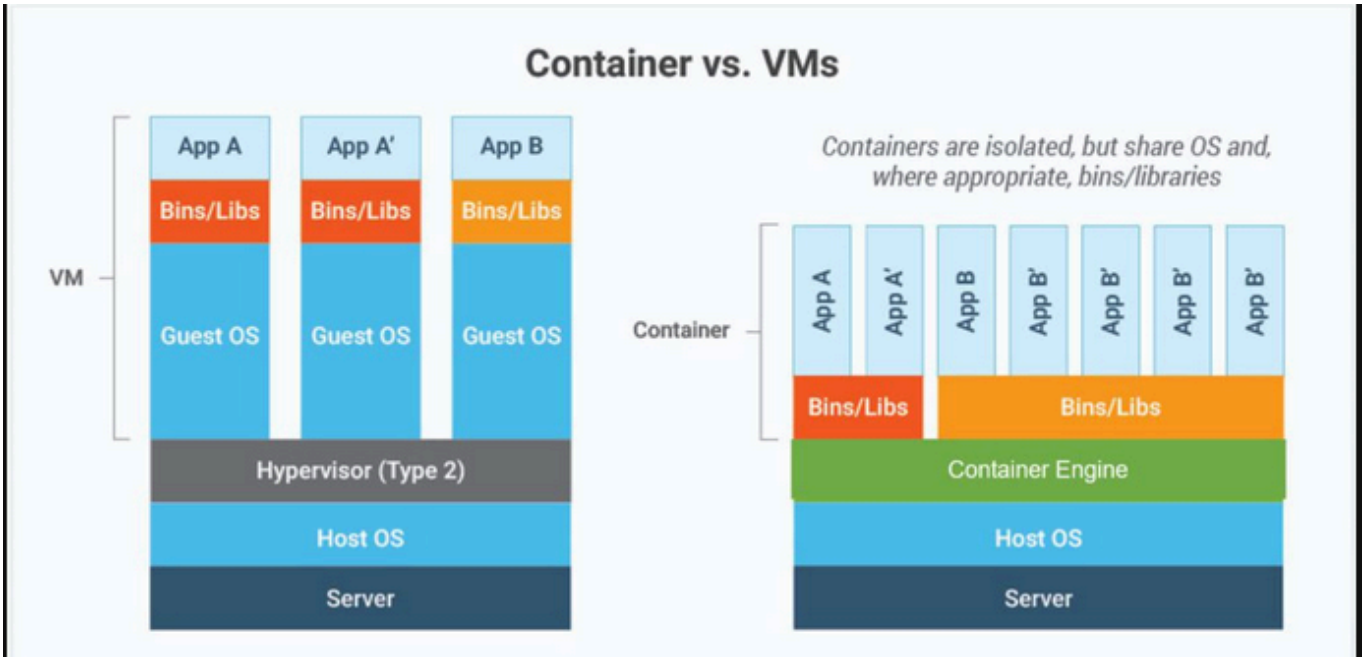
consumed by Guest OS even if it isn't doing nothing. Can take minutes to boot-up and sometimes it is critical in some real-time applications. Also hypervisors are not as efficient as the host OS.

Let's consider how we develop code. Nowaday we have the runtime environment, we use libraries developed by others and we have to not hardcode some features and we need to save some logs.

In deployment and simplified scenario we develop code in our laptop then we deploy and test the code in a staging server that simulates the HW of the production server and then the code is put on the production server. The environments are becoming more and more complex and the inconsistencies between development and production environment is becoming larger and larger(it worked on my computer problem). The real world is composed of a multitude of HW and OS combinations and so the multiplicity of system where you have to deploy is very large. We want a way to move the software developed to this multitude of configuration in a safe way. The solution is to use containers, as they mimic the transporter object. The key point is to have the SW moved safely between environment. We don't have the issue of asking hoe to handle the things contained in the container but we only care in how to handle the container.

Docker gives an environment where you can put all your code and then create the container to transport it.

The key difference between VM and Containers is that in VM we have virtualization supported by hypervisor and viceversa in Containers we provide virtualization at the operating system level. As the containers share the Host OS they can run applications without wasting resources. The container are faster than VM as they share the same kernel. The disadvantages is that you cannot run different OS as you are sharing a single kernel.



Docker permits tov run different version of the same application without wasting resources. In VM we have to pay to run every OS even if it is only a copy but required to run the applications. In Docker it is only stored the difference between the same application versions.

The concept is that Docker have a Docker file containing all the information, a Docker image

that is a file starting everything that is installed, but it only contains the binaries and the libraries of the application and a Docker Container Environment where the applications are runned.

Working with containers

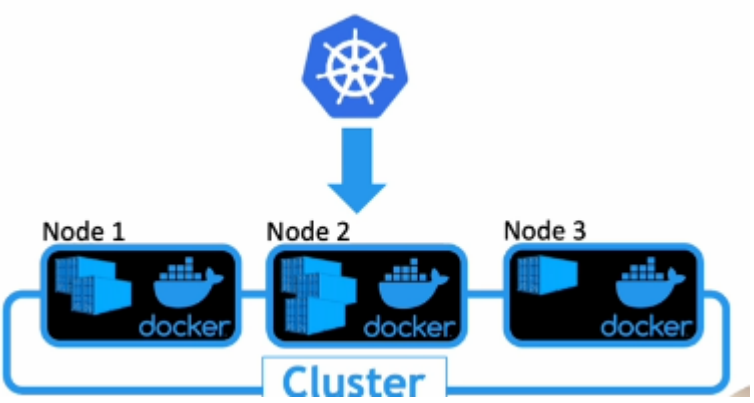
- docker build: to build an image from a Dockerfile
- docker run: to start a new container from an image
- docker ps: to list all the running containers
- docker stop: to stop a running container
- docker rm: to remove a stopped container
- docker images: to list all the images on the system
- docker pull: to download an image from, a registry
- docker exec: to execute a command inside a running container
- docker-compose: to manage multi-container applications
  The cycle is that we have a Docker engine, a Docker file for the application stored and through the docker build command we can generate a local image and with docker push we can store the image in a registry similar to the github one. With docker run you can run how much instances of the image. If something changes in the image we can create a version of the container saving only the delta, we can also perform the docker push(only deltas are stored and when there is the docker update you download the deltas and then the docker update, after some time, will be at the last version of the application)
  There are some limitations to Docker:
- It needs privileges for an external daemon to run(possible security concern)
- Enterprise-oriented as it allows easy micro-service virtualization, but it is not compatible with traditional High Performance Computing(HPC) systems
  There are some alternatives: Podman, LXD(LXC), Singularity
  Singularity
  Doesn't need root privileges, doesn't use a daemon. Born for scientific application as it is designed for general scientific use cases. It is easier to match resources requirements. HPC-oriented as its containers offer native support for GPUs, InfiniBand and MPI

What happens when we have many containers?

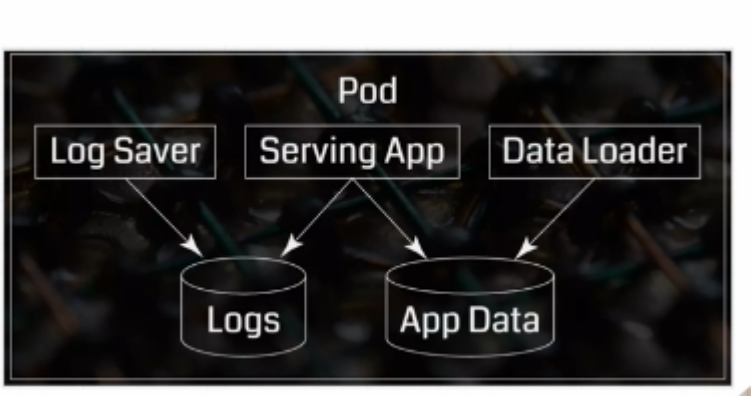
If we have many containers we risk to congeste the system we are considering. The real issue is that when we have to manage multiple containers the manual work increases(to scale up services, to fix crashing nodes, to run services), the complexity increases(to run new elements in production, to correctly scale with many containers) and the cost increases(in terms of human cost and public cloud cost).
The solution is container orchestration as you want to manage container to deploy in cluster. The main benefit are:

- Deployment: optimal resources usage, automatic scalability
- Networking: auto-discovery, accessibility from outside

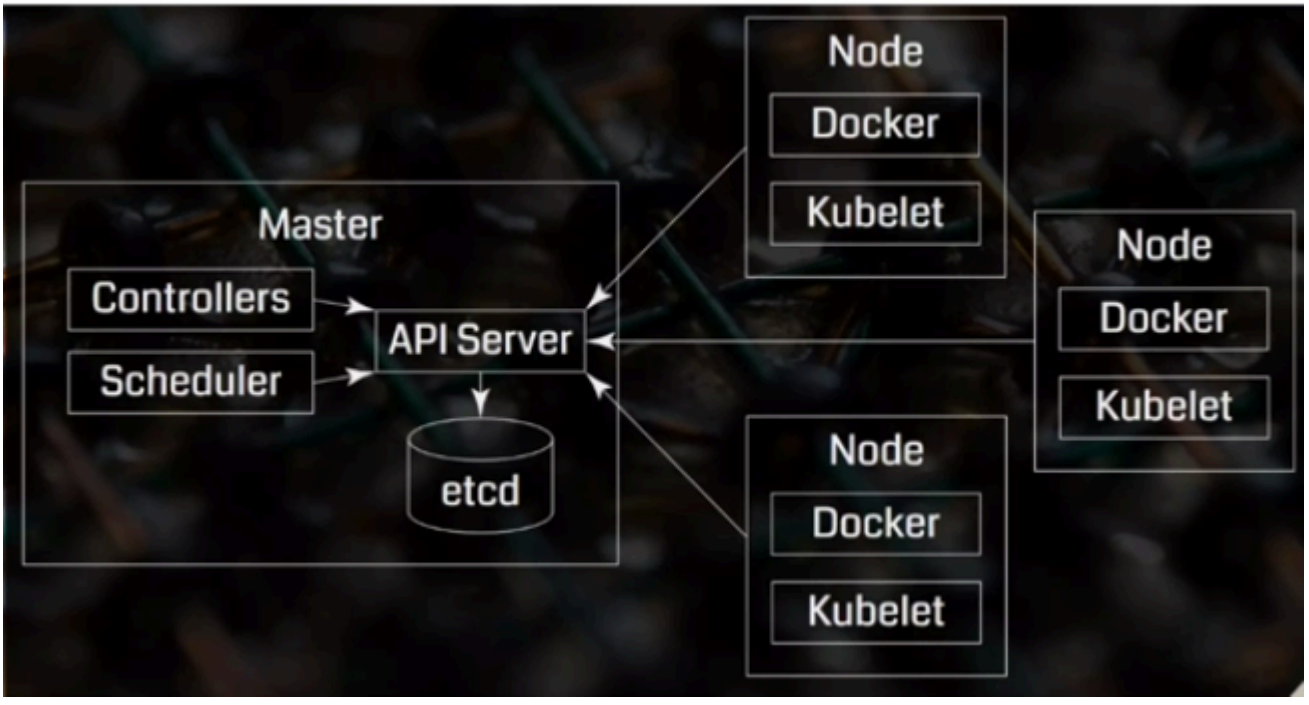
- Management: load balancing, fault tolerance, updates/rollbacks
  Kubernetes
  Largest container orchestration platform, open-sources, automatic.



It manages a single cluster(extensions can permit to handle multiple domains). The most important concept is the Pod: most simple unit that you can deploy. Constituted of multiple containers that share the same virtual environment and run in the same node.



The Pods are defined with json or YAML. Pods have a name defined as Label that identifies them. Then there is the kubernetes service that define the function of what the Pod can do.



The most important component is the Kubelet and its job is to run the Pod, double check that the containers are running correctly and restart the ones that are faulting.
The cAdvisors performs a low level monitoring of the resources gathered a container level.
In the master node there is an important component called etcd that metadata service that implements a distributed key-value store and store and replicate data used by Kubernetes across the entire cluster. It manages the state of cluster. Then all the services that kubernetes offers can be accessed by its API and it is the one distributing the pods where the cluster are running. Then there is the scheduler that search for new created pods to assign them to the nodes. It tries to balance the resources utilization, you can specify nodes affinity/anti-affinity. It is a bit lazy. The last component is Proxy that is a load balancer for pods, it implements the TCP, UDP stream forwarding or round robin forwarding across a set of backends.
At last the Replication Controller manages the replication of pods. In the manifest file we can decide how many replicas we want for every pod. The point is if we change the replica set in the manifest file we can augment/reduce the nodes allocated.