Containerisation

Barry Denby

Griffith College Dublin

April 16, 2020

What will be discussed this week

- Up to this point most of our discussion around cloud has used virtual machines as the main way of managing and distributing workload.
- However, recently another approach has been gaining traction in the form of containerisation.
- As another method of virtualising and load balancing workloads across machines.
- However as you will see containerisation and virtualisation are different approaches and have their advantages and disadvantages.

Definition of Containerization

- Operating-system-level virtualisation, also known as containerization refers to an operating system feature in which the kernel allow the existence of multiple isolated user-space instances.
- Such instances are referred to as containers and may look like real computers from the point of view of the program running inside them.
  - both of the above taken from wiki.

Definition of Containerization

- Programs that run inside the container will only ever see the resources allocated to them by the machine they are running on.
- Unless there is a security issue that enables the application to break out of the contained environment.

Containerization

- One of the earliest basic examples of this is a chroot (change root) jail.
- chroot permits you to choose anywhere in the filesystem to simulate as root (/) for the program to run in the chroot jail.
- For example if I made a chroot jail for a program at /home/barry/chroot-jail the program will only have access to everything in that directory.
- It would see /home/barry/chroot-jail as / and would not be able to go further up the directory structure.

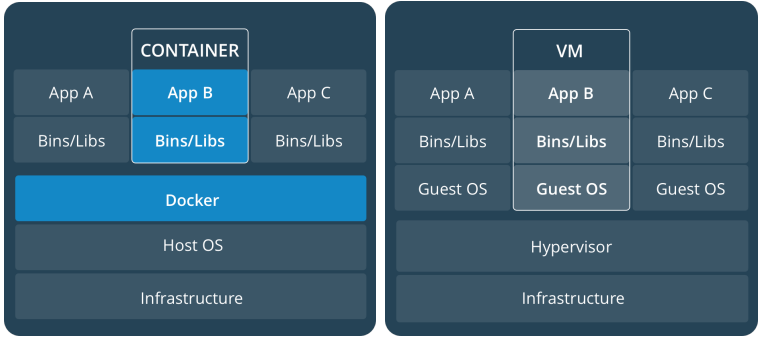
Containerization

- This is a very basic form of containerization, for something suitable for the Cloud we need something more advanced.
- The containerization software needs to be aware of the following:
  - The capabilities of the hardware it runs on.
  - The mechanisms through which it can read and write data (including network access).
  - Any connected peripheral devices.
- The containerisation software also needs to be able to provide a user defined subset of the above to a container.
- It also needs to hide anything outside that subset from the container.

Containerization

- In essence containerization tries to do VM like behaviour but without the need for having a full entire OS as part of an image.
- As it would be in a VM type environment.
- It also tries to do away with the need for a VMM by patching containers through to the underlying OS in order to run the container and its associated programs.
- The aim here is that by removing the OS the CPU, memory, and space required to run a container should be less than that needed to run a VM.

Containerization/VMM architectures



Features of Containerization that are similar to Virtualisation

- Containerization like virtualisation is used for securely allocating hardware resources among a set of containers to ensure they don't use more resources than they are allocated.
- Also like virtualisation it is used to separate services and programs into different containers as a way of improving security.
- There is also migration of containers where a container can be paused and be transferred to another node and restarted as if nothing had changed.
  - Useful in a load balancing situation.

Features of Containerisation that are dissimilar to virtualisation

- Containers are only capable of running applications that are built for the same OS as the host.
- Containers cannot run an entirely different OS from the host either.

Docker

- Docker is an example of such containerization software.
- It is an implementation of this operating system level virtualisation approach.
- Has been around since 2013.
- Cloud services like AWS and Google Cloud have support for this.

Docker how it works

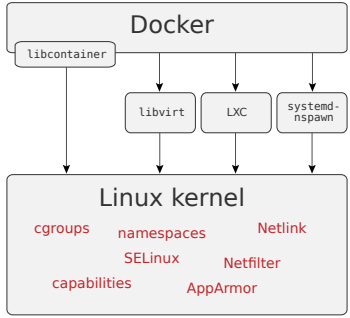
- Docker uses the features of a host OS to help run the containers.
- Thus a container is responsible for bundling together a set of applications, tools, libraries, and configuration files.
- This bundle will be stored in an image that can be used to make one or more containers.
- i.e. everything bar an OS that is required to run that application.

Docker how it works

- In order to provide support for containers and isolation between containers there are a number of features an OS must support in order for this to work.
- Control groups: used to limit, monitor, and isolate resource usage for a process or collection of processes.
- Kernel namespaces: such that a kernel can securely partition kernel resources between different sets of processes.
- A union-capable filesystem: a way of combining multiple directories into one that appears to contain their combined contents.
  - taken from "Union Mounts in 4.4BSD-Lite".

Docker how it works

- Using Linux as an example (Docker works on Windows and OSX too) the architecture diagram looks like the following (taken from wiki).



Docker how it works

- As you can see there are four different interfaces that docker can use to access the necessary OS features to provide isolation to containers.
- libvirt which is a virtualisation API that was originally built and used in the orchestration layer of hypervisors.
- LXC which is a containerisation method in and of itself that does a very similar job to Docker.

Docker how it works

- systemd-nspawn which is linux's system controller facilities for dealing with the creation and management of containers.
- libcontainer which is Docker's own library to get access to the virtualisation features of the kernel.

Docker libcontainer

- While there are multiple different mechanisms that can be used by docker to support containerization libcontainer is now the default.
- The reason for this is that the other three mechanisms that are mentioned are specific to the GNU/Linux OS.
- Thus if Docker was to be made available on other OSes one of these would have to be ported over to the other OSes as well.
- Thus it makes sense to write a single abstraction layer that will handle cross OS differences that Docker can use directly.

Docker libcontainer

- The single abstraction layer purpose is then to take every single piece of functionality that a container needs in order to run and translate it to the specific OS calls required for the OS that it is running on.
- It also standardises the way in which containers are stored, delivered, and how they are run.
- For docker to run on a different OS all that is needed is to port Docker itself and provide the necessary translations for libcontainer.
  - This is of course assuming that the OS supports all of the necessary features for OS level virtualisation.

Additional features that an OS must support in order to use docker

- There needs to be a mechanism to support inter process communication to enable communication between the individual containers. (linux uses the netlink interface for this)
- There needs to be a mechanism to filter or firewall traffic between containers and also to other machines. (linux uses the netfilter interface for this)

Additional features that an OS must support in order to use docker

- There needs to be a mechanism to provide fine grained control over superuser permissions so root/superuser usage can be avoided. (linux uses capabilities for this)
- Further to the previous point there needs to be a mechanism to restrict the capabilities of each individual program. (linux uses AppArmor for this)
- Finally there needs to be a mechanism to support access control security policies including the mandatory access controls model. (linux uses SELinux for this)

Additional features that an OS must support in order to use docker

- All these additional features are there to help facilitate communication between containers to enable them to synchronise and interact with each other.
- However a lot of the additional features are needed from the security perspective to lock down what a container can do should it be compromised.
- Similar to how VMs are locked down to prevent them from taking over a physical node of the cloud.

Additional features that an OS must support in order to use docker

- From a security perspective it a container is compromised the security features if implemented properly will ensure the following.
- By tightly filtering communication additional channels of communication can be blocked off.
- By giving each container the minimum set of capabilities it needs to do its job then the compromised container can only do the minimum possible damage.
  - Unless there are bugs in the docker system that can be used to perform privilege escalation.
- The mandatory access controls and access lists are designed to do a similar thing.

Docker components

- Docker is ecosystem composed of a number of components.
- The Docker daemon (dockerd) which is a service that manages and handles all docker containers.
  - It will listen out for any calls to the Docker Engine API or any requests from the docker command line tool.

Docker components

- A series of objects that are used to assemble Docker applications.
  - Containers: the standard encapsulated environment that applications run in.
  - Images: stores an application and associated libraries. these are read only and cannot be modified (security reasons for this).
  - Services: used for scaling containers across multiple nodes running docker, also known as a swarm.

Docker components

- Registries are used as a repository for docker images. These a modelled a little like Git in that images can be pushed and pulled from repositories.
  - Meaning they can be uploaded and downloaded respectively.
- One thing that Docker does not provide however is a mechanism to automatically scale and manage docker containers in response to demand.
  - Which is core functionality to the cloud.
- This is where something like Kubernetes is useful.

Docker Advantages

- Docker containers require much less resources to run than a full VM as an OS is not included as part of a container.
- As a result it is possible to run more containers on the same machine spec than you could run VMs.
  - As there is essentially one OS shared between all containers rather than each VM having an OS to itself.

Docker Disadvantages

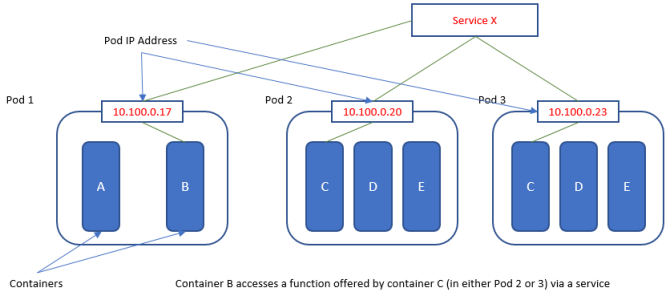
- You cannot run other OSes or other OS applications in concert like you could with VMs and a VMM.
- Applications have to be built for the OS that is running the instance of Docker.

Kubernetes

- Kubernetes is a container orchestration system that is designed to automatically manage, deploy and scale containerised applications.
- Much in the same way that AWS and Google Cloud or any cloud system has these facilities for dealing with VM based applications.
- It can work with a number of containerisation software but Docker was one of the first it used.
- The aim like cloud computing is to have self managing applications that scale with demand.

Kubernetes architecture

- This is what the general architecture of Kubernetes looks like.



Kubernetes Pods

- The basic unit of scheduling in Kubernetes is a pod which is a group of containers that are guaranteed to be located on the same physical host.
- Every pod is allocated a unique Pod-IP address however, these are not used directly in case pods have shutdown or restart.
  - Pod-IP addresses are allocated dynamically.
- Pods can also have volumes attached to them (storage either local or network) for the pod to access and manipulate.

Kubernetes Services

- Services are a collection of pods that are designed to work together.
- For this to work there needs to be a mechanism for translating service names to pod-ip addresses dynamically.
  - This is where the Kubernetes DNS comes in.
- Kubernetes will provide service discovery and will allocate pod-ips to pods and will also take a note of what service they provide.
  - Thus should a service be required it will be requested from the Kubernetes DNS which will provide the pod-ip along with the container that will provide the service in that pod.

Kubernetes Volumes

- Volumes are there to provide persistent storage that pods can write to.
- This is to get around the limitation of docker that images are read only.
- Volumes can be shared by multiple pods at the same time if necessary.
- Necessary for any application that needs to store and use data similar to cloud services.
  - i.e. basically everything bar the most trivial applications.