**DESCRIPTION**

**Contanerization**

**Reference**

* [History of Containerization](https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016)
* [Video Explanation from VMWare](https://www.youtube.com/watch?v=EnJ7qX9fkcU)
* Docker for the Virtualization Admin Book
* [IBM learning Containerization](https://www.ibm.com/cloud/learn/containerization)
* [Namespaces Explained from NDC by Michael Kerrisk](https://app.revature.com/admin-v2/https:/www.youtube.com/watch?v=0kJPa-1FuoI&list=RDCMUCTdw38Cw6jcm0atBPA39a0Q&start_radio=1)
* [Windows Containers](https://app.revature.com/admin-v2/https:/docs.microsoft.com/en-us/virtualization/windowscontainers/manage-containers/hyperv-container)
* [Windows Containers Under the hood](https://medium.com/@justen.walker/a-short-introduction-to-windows-containers-db5adc0db536)
* [Windows Containers a History](https://docs.microsoft.com/en-us/archive/msdn-magazine/2017/april/containers-bringing-docker-to-windows-developers-with-windows-server-containers)

**Introduction**

A **container** is a mechanism for packaging an application(or limited number of processes) with its dependencies so that it runs in its own isolated sandbox.

Containerization helps to ensure the application or set of processes can run reliably regardless of the host environment. The container shouldn't be able to modify or interact with it anything it doesn't need and, on the whole, changes in the container should not effect the Host or other containers (and vice versa).

The following diagram illustrates the structure of containers- an application in a container is isolated to just what it needs.

**Foundations**

**Linux**

LinuX Containers, which provide the foundation for most modern container systems (including Docker), began with two key kernel tools in Linux: cgroups and namespaces

* Control groups (cgroups) is a kernel feature that allows you to exercise control over the resources used for a particular process-- via monitoring and limiting
  + Things like limiting the amount of memory for a process, determining the priority of a process- how much CPU time, and stopping/starting that process or set of processes.
* Namespaces
  + method of encapsulating a global resource
  + Different types:
    - mount: set of mount points exposed (First namespace type introduced)
    - PID: set of process IDs exposed, form a hierarchy
    - Net (network): set of network resources exposed; routing tables; socket port member names etc.
    - UTS: Unix timesharing system- defines visible host and domainname-- not related to DNS
    - User: Isolates the user ID and group ID-- can make a user seem like the super user for its namespace, but in actuality the user has limited privileges in the context of the greater system
    - IPC
    - Cgroup
    - and more
  + Namespaces determine what is visible to a particular process. A process begins as part of a single instance of each particular namespace type.
    - can only see a certain set of other process IDs, network resources, mount points etc.
    - namespaces also provide a layer of virtualization-- a process might see itself as having PID 1. In its namespace this may be "true", but there may be a namespace further up the heirarchy that understands this same process as having PID 312

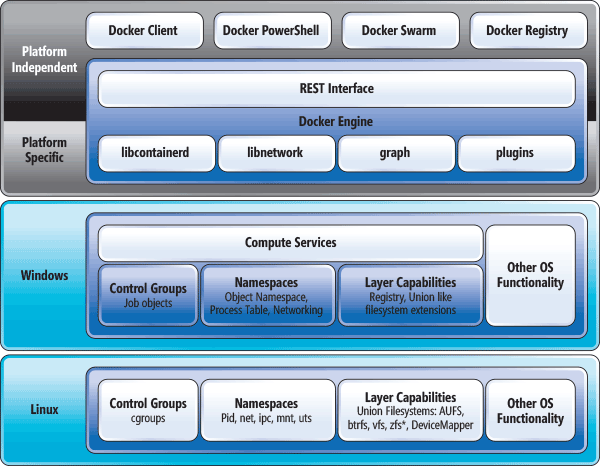
Take the cgroup management of resources + namespaces => the basics of a container

=> Isolation and Virtualization => Containerization

By packaging up an application in a container you ensure that it has all the needed dependencies and can modify/use only what it needs. You also ensure that the application is portable.

**Windows Containers**

As Linux containers gained popularity, Windows worked to keep up and find it's own solution to support containerization. While Windows met with some challenges since their operating system setup is a bit different from Linux, they were able to ultimately support the same functionality through tools that were analogous to cgroups and namespaces. Though, because of the manner in through which certain resources must be shared, they also came up with another version of containers that's based on the same technology used by virtual machines. As they got things working and were able to break down the problems they faced, in 2016, the question became how to make these features available to users of the Windows OS. The solution became to partner with Docker who by that point was already a huge driving force in the move toward containerization.

As the diagram illustrates the docker engine runs upon the platform specific supports for containers within either the Windows OS or Linux OS.

**Containers**

* Built from images (template for the container)
* Run on an engine (on the host OS) i.e. Docker Engine
* Ideally stateless
  + State needed to persist for an application should be stored in a way that is essentially "detachable" from the actual container- otherwise state only persists so long as a container is running- volumes-- solve this
* Virtualization
* Isolation

**Benefits**

* Secure
  + Isolation and Virtualization keep your containerized apps more secure
* Standardized and thus Portable
  + Think write once run anywhere
* Lightweight
  + shares the host operating system's kernel
* Flexible and Loosely Coupled
* Scalable
  + Easy to spin up and because of this lightweight ease they can be scaled up quickly