Introduction To Docker

Table of Contents

1. Docker containers
    - What is a Container?
    - Why Containers
    - Why should we use containers or what was the problem without containers?
        - Solution 1 with Virtual Machine
        - Solution 2 Docker containers
    - What is a containerization?
    - Benefit of Containers
2. Virtual Machine vs Docker Containers
3. Container Orchestration
    - What is Container Orchestration?
    - Tools of Container Orchestration
4. Containers Registry
    - Elastic Container Registry (ECR)
        - What is Elastic Container Registry (ECR)
        - Benefits of Elastic Container Registry (ECR)
    - Docker Hub
        - What is a Docker Registry?
        - Why should you use Docker Hub?

1. Docker containers

What is a Container?

A container is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to your application such as code, system libraries, system tools, runtime environment and settings.

Why Containers

Instead of virtualizing the hardware stack as with the virtual machines approach, containers virtualize at the operating system level, with multiple containers running on top the **OS kernel** directly. This means that containers are far more **lightweight** because they share the OS kernel, start **m uch faster**, and use a fraction of the **memory** compared to booting an entire OS. There are many container formats available. Docker is the most popular.

Why should we use containers or what was the problem without containers?

- In a normal IT work flow developers would develop a new application. Once the application development was completed, they would hand over that application over to the operations engineers, who were then supposed to install it on the production servers and get it running.
- If the operations engineers were lucky, they even got a somewhat accurate document with installation instructions from the developers. So far, so good, and life was easy.
- But things get a bit out of hand when, in an enterprise, there are many teams of developers that create quite different types of application, yet all of them need to be installed on the same production servers and kept running there.

- Usually, each application has some external dependencies, such as which framework it was built on, what libraries it uses, and so on. Sometimes, two applications use the same framework but in different versions that might or might not be compatible with each other.
- So, installing a new version of a certain application can be a complex project on its own, and often needed months of planning and testing
- But these days we have to release a patch, update very often so this development and testing cycle can be very risky to the business.

Solution 1 with Virtual Machine

The first solution was using **Virtual Machines**(VMs)

- Instead of running multiple applications, all on the same server, companies would package and **run a single application on each VM**
- With this, all the compatibility problems were gone and life seemed to be good again.
- But this comes with it’s own set of demerits where each VM needs a lot of resources where most is used by underlying system OS.

Solution 2 Docker containers

The ultimate solution to this problem was to provide something that is much more lightweight than VMs with is just **Docker container**

- Instead of virtualizing hardware, containers run on top of single Linux instance.
- Don't mistake Docker Engine as the equivalent of a hypervisor in more traditional VM, it is simply encapsulating process on the underlying system.
- With docker, developers would now package their application, dependent libraries, framework in a container and it can run everywhere that docker is installed.
- Operation teams can easily deploy the container without having to worry about configuring the application as these containers already contain an up and running application.

What is a containerization?

Let me explain this with an example. Usually, in the software development process, code developed on one machine might not work perfectly fine on any other machine because of the dependencies. This problem was solved by the containerization concept. So basically, an application that is being developed and deployed is bundled and wrapped together with all its configuration files and dependencies. This bundle is called a container. Now when you wish to run the application on another system, the container is deployed which will give a bug-free environment as all the dependencies and libraries are wrapped together. Most famous containerization environments are Docker and Kubernetes.

Benefit of Containers

Consistent Environment Containers give developers the ability to create predictable environments that are isolated from other applications. Containers can also include **software dependencies** needed by the application, such as specific versions of programming language runtimes and other software **libraries**. From the developer's perspective, all this is guaranteed to be ***consistent*** no matter where the application is ultimately deployed. All this translates to productivity. Developers and IT Ops teams spend less time debugging and diagnosing differences in environments. Therefore, they more time available to develop and shipp new functionality for users. And it means fewer bugs since developers can now make assumptions everything will work well in **dev, test and production environments**.

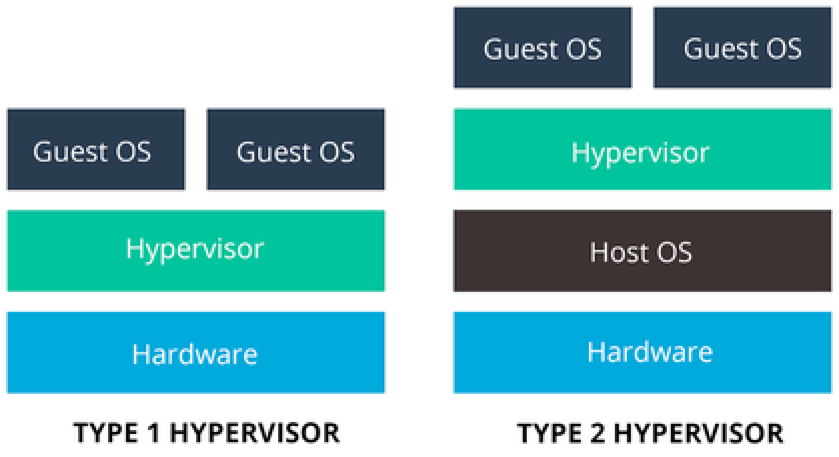
Run Anywhere Containers are able to run virtually anywhere, greatly easing development and deployment on Linux, Windows, and Mac operating systems, on virtual machines or bare metal, on a developer's machine, in data centers, on-premises and of course in the public cloud.

Isolation Containers virtualize CPU, memory, storage, and network resources at the OS-level, providing developers with a sandboxed view of the OS logically isolated from other applications.

2. Virtual Machine vs Docker Containers

The image should be self explanatory to understand the difference between **Docker and VMware** architecture.

- VM requires an **Hypervisor** which can be either installed on an operating system or directly on the hardware while a container can be deployed after installing docker.
- VM requires a **separating OS** to be installed to deploy your application while Docker containers share the **host operating system**, and that is why they are ***lightweight***
- Since Docker shares OS with host, the boot up time of docker container is **very less** while it is comparatively higher for VMs
- The docker containers share **Linux kernel** so it would be a good fit if you are planning to run multiple applications on the same Linux kernel but if you have applications that require different operating system then you will have to go for VM
- Since VM does not share the host OS it is comparatively more ***secure than Docker*** containers. An attacker may exploit all the containers if it gets access to the host or any single container.
- Since containers don't have OS they use comparatively very **less resources** to execute the application and you can utilize the underlying resources more effectively



3. Container Orchestration

What is Container Orchestration?

Container orchestration is a process that automates the deployment, management, scaling, networking, and availability of container-based applications. Also, it helps in monitoring the cluster and allow containers to communicate with one another.

Now that you are familiar with containers, next we need to learn about **container orchestration**. Just to summarise we have a docker container with certain applications running inside the container.

- It is possible that your application from container 1 is dependent on some other application from another container such as database, message, logging service in the production environment.
- You may also need the ability to **scale up** the number of containers during peak time, for example I am sure you must be familiar with Amazon sale during holidays when they have a bunch of extra offers on all products. In such case they need to scale up their resources for applications to be able to handle more number of users. Once the festive offer is finished then they would again need to **scale down** the amount of containers with applications.
- To enable this functionality we need an underlying platform with a set of resources and capabilities. The platform needs to **orchestrate** the connectivity between the containers and automatically scale up or down based on the load.
- This while process of deploying and managing containers is known as **container orchestration**
- **Kubernetes is thus a container orchestration technology** used to orchestrate the deployment and management of hundreds and thousands of containers in a cluster environment.
- There are multiple similar technologies available today, docker has it’s own orchestration software i.e. Docker Swarm, Kubernetes from Google and Mesos from Apache.

- Your application is now **highly available** as now we have multiple instances of your application across **multiple nodes**
- The user traffic is **load balanced** across various containers
- When demand increases, deploy more instances of the applications seamlessly and within a **matter of seconds** and we have the ability to do that at a service level when we run out of hardware resources then scale the number of underlying **nodes up and down** without taking down the application and this all can be done easily using a set of ***declarative object configuration file***.

Tools of Container Orchestration

What are some Container Orchestration tools available?

Kubernetes: An orchestration system for Docker containers. It handles scheduling and manages workloads based on user-defined parameters.

Amazon ECS: ECS supports Docker containers and lets you run applications on a managed cluster of Amazon EC2 instances.

Docker Swarm: Provides native clustering functionality for Docker containers, which turns a group of Docker engines into a single, virtual Docker engine..

Azure Kubernetes Service (AKS): It is a robust and cost-effective container orchestration service that helps you to deploy and manage containerized applications in seconds on Azure.

Amazon Elastic Kubernetes Service (Amazon EKS) gives you the flexibility to start, run, and scale Kubernetes applications in the AWS cloud.

AWS Fargate is a serverless compute engine for containers that works with both Amazon Elastic Container Service (ECS) and Amazon Elastic Kubernetes Service (EKS)

Amazon Elastic Container Service (Amazon ECS) is a highly scalable, fast container management service that makes it easy to run, stop, and manage containers on a cluster

4. Containers Registry

Elastic Container Registry (ECR)

- **Elastic Container Registry (ECR)** is a fully-managed Docker container registry that makes it easy for developers to store, manage, and deploy Docker container images.
- ECR hosts our images in a highly available and scalable architecture, allowing us to reliably deploy containers for our applications.
- Integration with AWS Identity and Access Management (IAM) provides resource-level control of each repository.
- With Amazon ECR, there are no upfront fees or commitments. We pay only for the amount of data you store in your repositories and data transferred to the Internet.

Benefits of Elastic Container Registry (ECR)

- Full managed
- Secure
- Highly Available
- Simplified Workflow

Docker Hub

What is a Docker Registry?

A Docker registry is a service that hosts and distributes Docker images.

In many cases, a registry will consist of **multiple repositories** which contain images related to a specific project. Within a given repository tags are used to differentiate between versions of an image (e.g. ubuntu/httpd:version2.4, ubuntu/httpd:version2.5, ubuntu/httpd:version2.6 where “version2.x” is the tag). Users can pull (download) images they want to use or push (upload) images they want to store in a registry.

What is a Docker Hub?

Docker Hub is Docker’s official cloud-based registry for Docker images.

Why should you use Docker Hub?

So, with all the different Docker registry options, why should you use Docker Hub? Here are a few of the key upsides of Docker Hub:

- **A large library of trusted images**- Docker Certified images, Verified Publisher images (which are Docker Certified and verified by the publisher), and Official Images published by Docker add a layer of trust for users. With millions — or in some cases billions — of downloads for many commonly used images, you can count on a reliable base image when you use Docker hub. While that’s great from the user perspective, it also benefits publishers as hosting an image in Docker Hub can give your project more exposure.
- **A free tier- Currently,** Docker’s free plan offers unlimited public repositories and 1 private repository with up to 3 collaborators.
- **Built-in security features** All accounts can benefit from local image vulnerability scans. “Team” accounts also gain access to audit-logs and multifactor authentication (MFA) to further secure repositories. **Integrations & features that enable CI/CD–** Docker Hub also supports GitHub & Bitbucket integrations, automated tests, build triggers, and webhooks to help automate development pipelines and enable CI/CD (continuous integration/continuous delivery).