Terraform with Docker & Kubernetes

Vishal Saini

P 2 V 2 C

Any application, web or otherwise, will need to be hosted on a system for it to function. These systems are often called as servers. for example, web applications have to be hosted on a powerful web server and made available over internet to serve web pages.

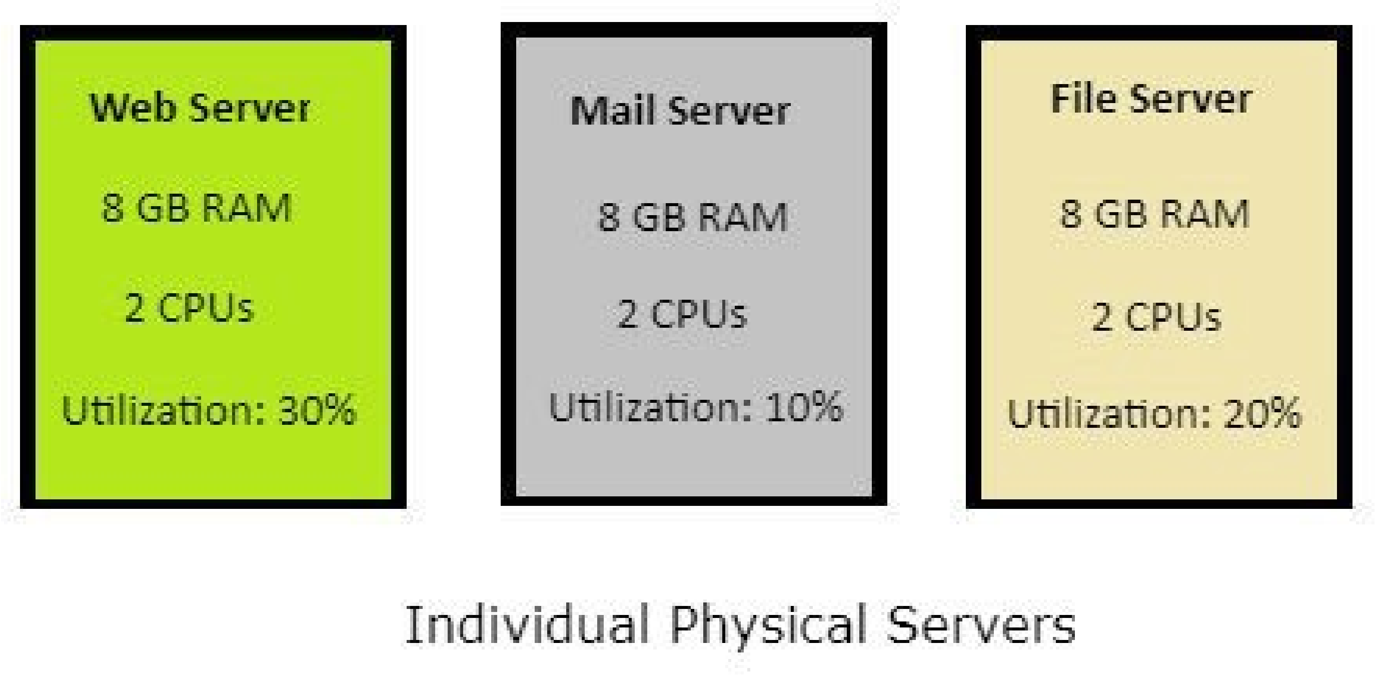
How these servers/systems have evolved over time and what physical systems, virtual machines and containers bring into the table?

Physical Systems / Servers:

Earlier days, Individual physical systems were used to host individual applications. Say, you need a web server to cater your web application, you will have to buy a dedicated physical machine with so and so configuration to power it up.

Again if you need a mail server to cater your mailboxes, you have to buy a new physical system. Your physical infrastructure keeps growing as your need grows.

Physical System



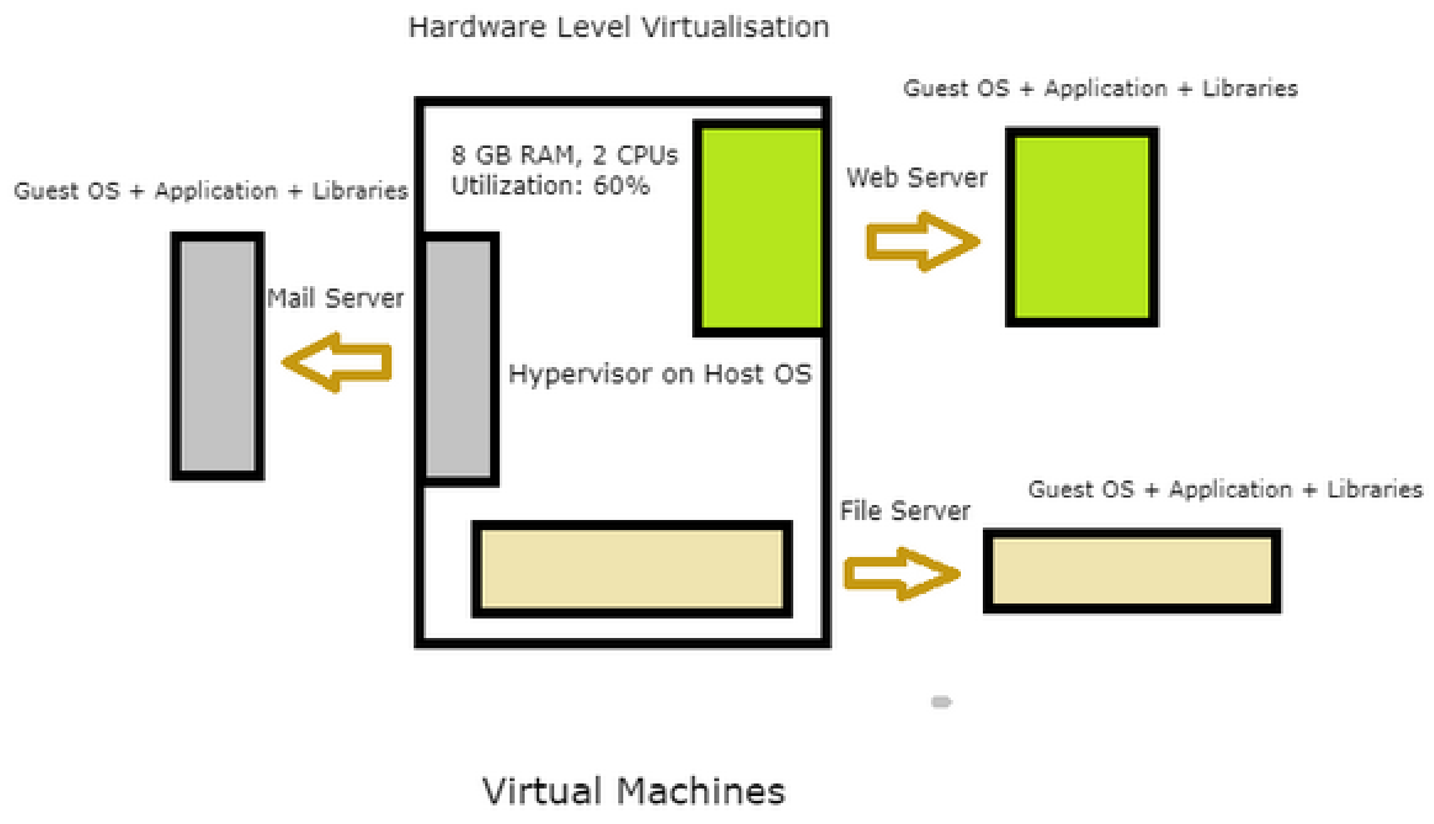
Virtual Machine ?

A Virtual machine provides a way to make a single physical system work as multiple isolated systems, resulting in higher infrastructure usage and reduced physical hardware infrastructure overhead.

How is it achieved?

Using hypervisor that runs on your host system, you can effectively split your underlying physical infrastructure into multiple smaller units that can run multiple isolated systems.

Virtual Machine

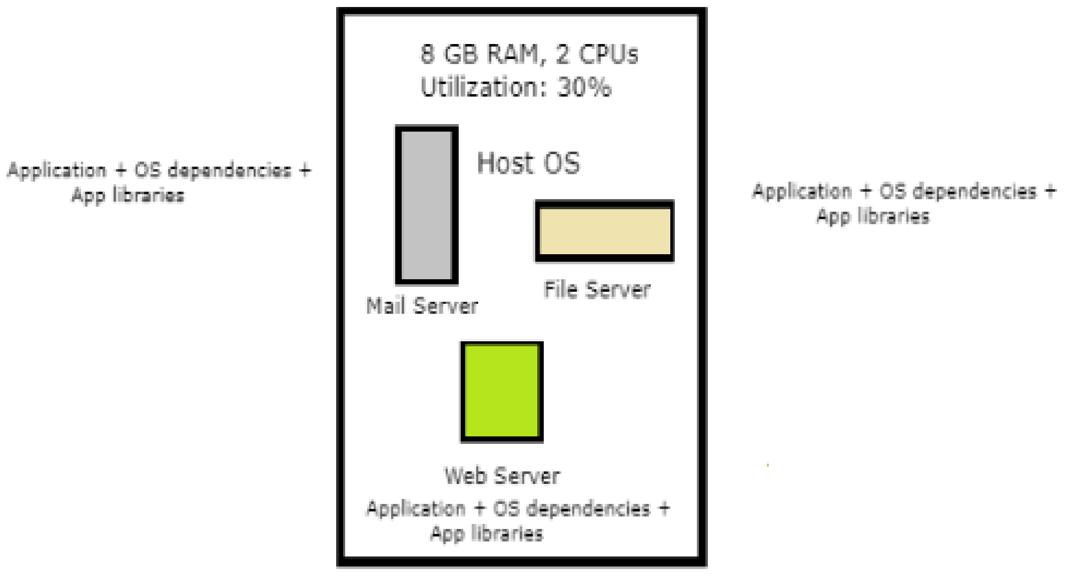

Hardware Level Virtualisation
Guest OS + Application + Libraries
8 GB RAM, 2 CPUs
Utilization: 60%
Web Server
Guest OS + Application + Libraries
Mail Server
Hypervisor on Host OS
File Server
Guest OS + Application + Libraries
Virtual Machines

Containers?

Containers enable virtualization in the sub-system level rather than hardware/OS level.

While hypervisors achieve virtualization by splitting hardware resources and running a separate set of OS on the virtualized hardware, containers utilize isolation at the subsystem level using namespaces and cgroups.

Subsystem level Virtualization


8 GB RAM, 2 CPUs
Utilization: 30%
Host OS
Mail Server
File Server
Web Server
Application + OS dependencies + App libraries
Application + OS dependencies + App libraries
Application + OS dependencies + App libraries

Containers

Advantages of Containers ?

> Portable

> Extremely small footprint

> Reduced IT management resources

> Quicker spinning of apps

What is Docker ?

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers.

Docker is currently the most popular container platform. Docker appeared on the market at the right time, and was open source from the beginning, which likely led to its current market domination.

The Need for Orchestration Systems

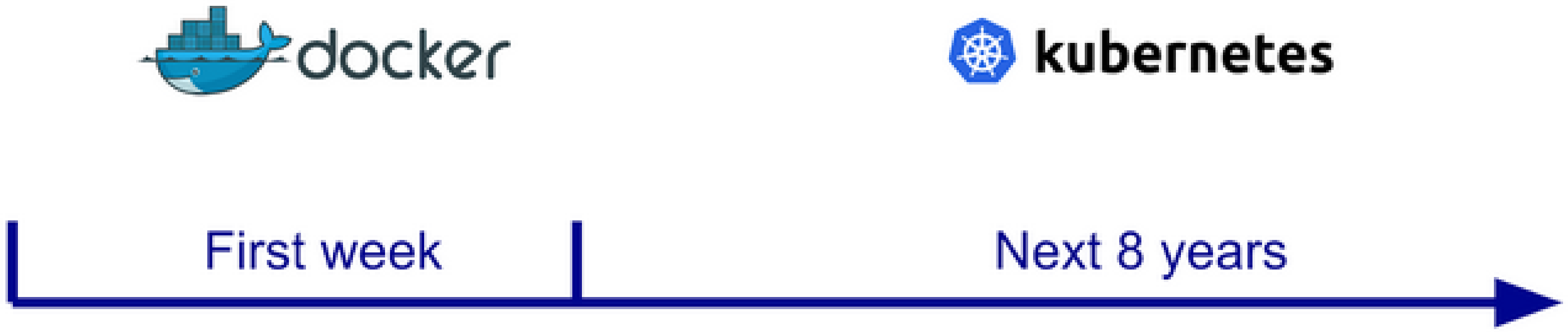
While Docker provided an open standard for packaging and distributing containerized applications, there arose a new problem.

> How would all of these containers be coordinated and scheduled?

>How do you seamlessly upgrade an application without any interruption of service?

> How do you monitor the health of an application, know when something goes wrong and seamlessly restart it?

Life of an application



Container Orchestration tools ?

docker SWARM

Apache MESOS™

kubernetes

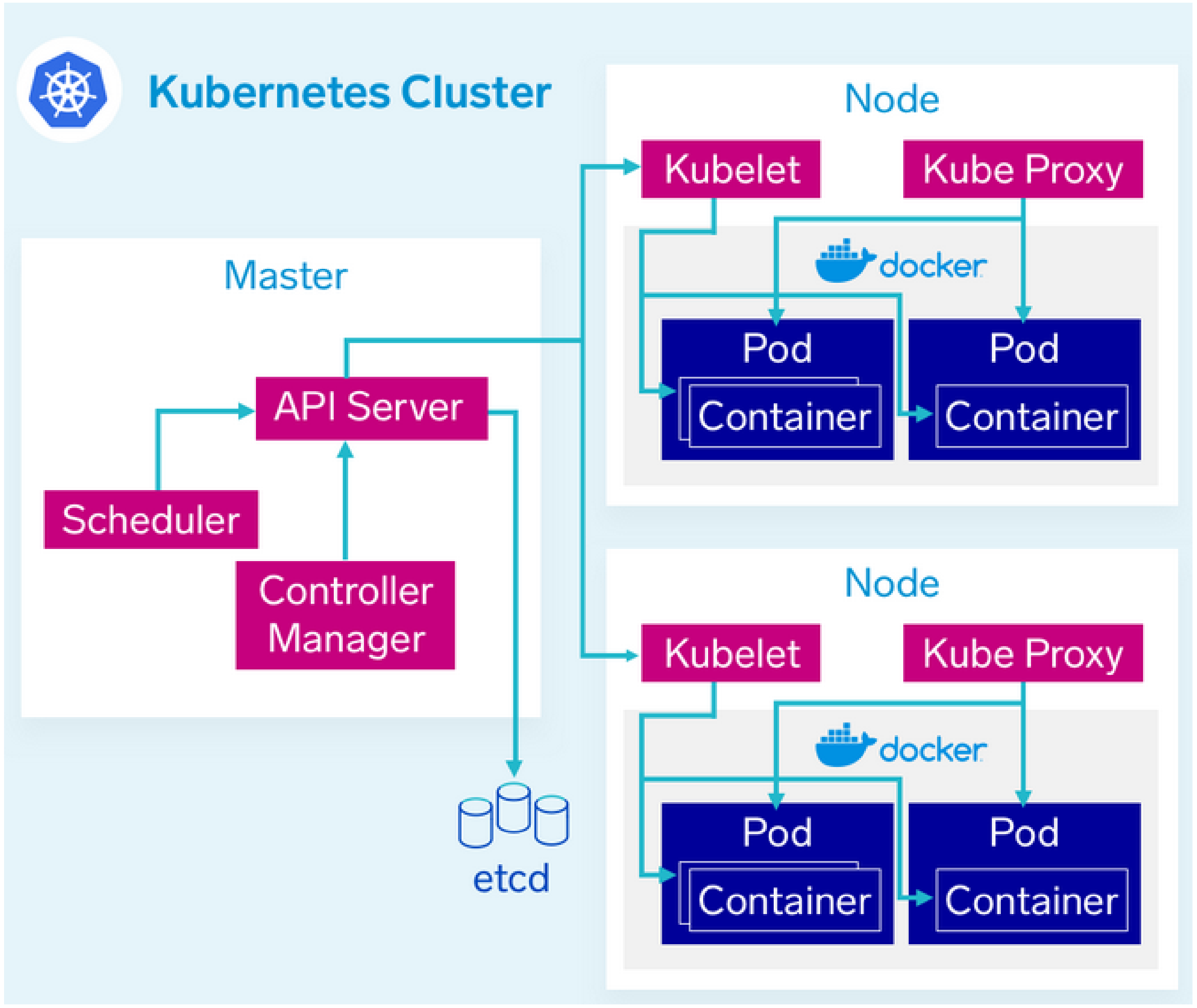
Kubernetes ?

Kubernetes is the container orchestrator that was developed at Google which has been donated to the CNCF and is now open source.

It has the advantage of leveraging Google's years of expertise in container management.

It is a comprehensive system for automating deployment, scheduling and scaling of containerized applications, and supports many containerization tools such as Docker.

Architecture of Kubernetes



Can you use Docker without Kubernetes?

Docker is commonly used without Kubernetes, in fact this is the norm. While Kubernetes offers many benefits, it is notoriously complex and there are many scenarios where the overhead of spinning up Kubernetes is unnecessary or unwanted.

In development environments it is common to use Docker without a container orchestrator like Kubernetes.

Can you use Kubernetes without Docker?

As Kubernetes is a container orchestrator, it needs a container runtime in order to orchestrate.

Kubernetes is most commonly used with Docker, but it can also be used with any container runtime.

RunC, cri-o, containerd are other container runtimes that you can deploy with Kubernetes.

Docker command vs terraform Template

```
# docker pull nginx:latest
```

To download the nginx image with latest tag from dockerhub

```
provider "docker" {}

resource "docker_image" "nginx" {
  name = "nginx:latest"
}
```

Template

# docker run –name=cont1 nginx

To run the container with downloaded nginx image

```
provider "docker" {}
resource "docker_image" "nginx" {
  name = "nginx:latest"
}
resource "docker_container" "dock1" {
 name = "cont1"
 image = "${docker_image.nginx.latest}"
}
```

Export port for container

```
# docker run -p 80:80 –name=cont1 nginx

resource "docker_container" "dock1" {
 name = "cont1"
 image = "${docker_image.nginx.latest}"
 ports {
    internal = 80
    external =8090
}
}
```

Docker network

```
#docker network create --driver bridge --subnet
192.168.0.0/25 net1


provider "docker" {}
resource "docker_network"  "network1" {
  name = "net1"
  driver = "bridge"
  ipam_config {
    subnet = "192.168.0.0/24"
}
}
```

Docker volume

```
# docker volume create vol1

provider "docker" {}

resource "docker_volume" "vol" {
  name = "vol1"
}
```

Kubernetes

We will be using minikube setup to do the practical of kubernetes.

Minikube is a tool that makes it easy to run Kubernetes locally.

Minikube runs a single-node Kubernetes cluster inside a Virtual Machine (VM) on your laptop for users looking to try out Kubernetes or develop with it day-to-day.

Kubernetes Namespace

```
provider "kubernetes" {
}

resource "kubernetes_namespace" "example" {
  metadata {
    name = "my-first-namespace"
  }
}
```

Kubernetes Pods

```
resource "kubernetes_pod" "ghost_alpine" {
  metadata {
    name = "ghost-alpine"
  }

  spec {
    host_network = "true"
    container {
      image = "ghost:alpine"
      name  = "ghost-alpine"
    }
  }
}
```

Pod and Service

```
provider "kubernetes" {
}
resource "kubernetes_service" "ghost_service" {
  metadata {
    name = "ghost-service"
  }
  spec {
    selector = {
      app = "${kubernetes_pod.ghost_alpine.metadata.0.labels.app}"
    }
    port {
      port = "2368"
      target_port = "2368"
      node_port = "8081"
    }
    type = "NodePort"
  }
}

resource "kubernetes_pod" "ghost_alpine" {
  metadata {
    name = "ghost-alpine"
    labels = {
      app = "ghost-blog"
    }
  }

  spec {
    container {
      image = "ghost:alpine"
      name  = "ghost-alpine"
      port  {
        container_port = "2368"
      }
    }
  }
}
```