Introduction to Cloud Native Principles

A cloud-native application is engineered to run on a platform and is designed for resiliency, agility, operability, and observability.

Resiliency, Agility, and Operability are the core pillars of the cloud. We then go on to learn a little bit of Microservices, Containers, VM and Kubernetes.

Now that you have an overview of a) what data center looks like b) have an understanding of the various pieces of AWS c) understand the reason on why your company needs to move to the cloud lets go into some of the Cloud Native Principles.

The reason that you need to deeply understand these principles is that ideally when an application is running on the cloud it needs to maximize the benefits of running on the cloud. When applications are not built keeping these principles in mind there are more chances of things failing.

Cloud Native Principles

A cloud native application is engineered to run on a platform and is designed for resiliency, agility, operability, and observability.

1. Resiliency embraces failures instead of trying to prevent them; it takes advantage of the dynamic nature of running on a platform.
2. Agility allows for fast deployments and quick iterations.
3. Operability adds control of application life cycles from inside the application instead of relying on external processes and monitors. Observability provides information to answer questions about application state. Think monitoring.

The Cloud Native Computing Foundation (CNCF) has defined cloud native as:

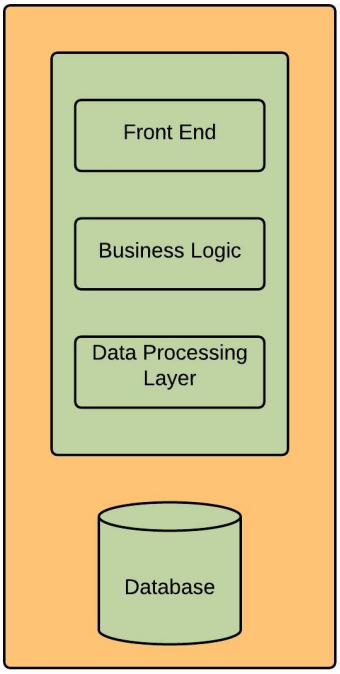
1. Micro-services based Architecture
2. Containerized
3. Distributed Management and Orchestration

Lets dive a little deeper and understand what this means.

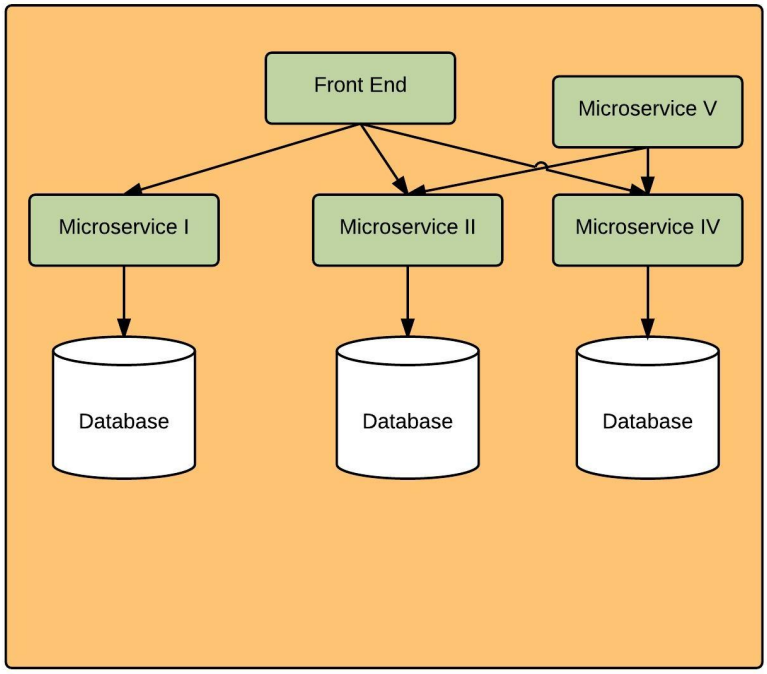
Micro-services Architecture

A micro-service architecture is a software architecture style where complex applications are composed of several small, independent processes communicating with each other using language-agnostic APIs. These application services are small, highly decoupled and focus on doing a small task. Microservices refers to smaller and more manageable services that serves a specific use case.

Microservices are generally deployed into containers. We will learn what is meant by containers in general and look into the most popular types of containers - docker containers.



Enterprise / Monolithic Architecture



Microservice Architecture

What is a container?

Container in short is a lightweight Virtual Machine. Containers are a way to package all that is needed for your application to work, be it operating system, libraries, config files or other applications all into one bundle.

There are 2 concepts in containers, Images & Running Containers.

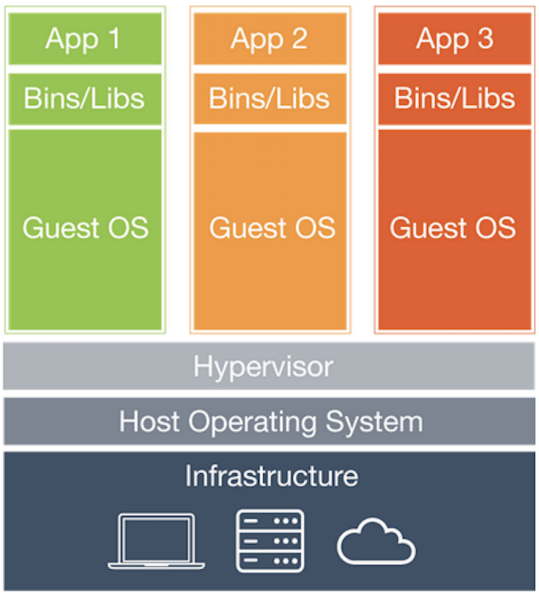
1. Image containers have images of the OS or the application itself, they are

static.

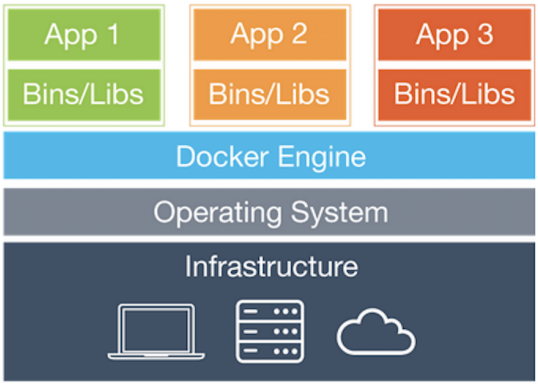
2. Running containers on the other hand are containers on a single host without visibility into each others' processes, files, network

Now, a container image includes an application & all its dependencies, using which you can create instances of Containers which can be run on any environment, be it developer, QA, support or in data-centers.

To understand the difference between VMs & containers, we should briefly look at VM architecture vs the Container Architecture.



Virtual Machines



Containers

Virtual Machines

VMs virtualize hardware aspect of a computer whereas containers virtualize at the operating system level. VMs need a Guest OS on top of Hypervisor, a virtualization software.

Hypervisors

Hypervisor manages the allocation of Hardware's processor, memory & resources to the VMs. On the flip side, Containers share the OS's compute, memory & resources and at the same time achieve the isolation provided by the VM architecture. Simply put, by eliminating the guest OS from a VM would give us Containers, which in itself reduces lot of overhead that comes along with VMs.

Containers

What are we trying to achieve using containers? Isolation, Co-existence of applications, virtualization of software, replication in diff environments.

Containerization is an OS feature where Kernel allows the existence of multiple, isolated user spaces known as Containers. These containers may appear as real computers to programs running in them as they have access to all the resources a real computer provides, like connected devices including network devices, compute power, memory, hard disk. However, a container is isolated from other containers & resources allocated to them.

If you need to replicate your application in multiple environments, you can package the application and all of its dependencies into a container image and all your developer, test, production & support teams will have the same image to start with. And once you have the container image, you can bring up the container in a matter of minutes or less.

Imagine a scenario in which a customer has an issue on the field and support folks try to replicate it in their own environments. How many times have they struggled with setting up the environments and not being able to reproduce it? Similar case repeats itself with developer & QA environments.

Now, just imagine a situation where the support folks are able to whip up a container and have exactly same environment as is in the field with the customer. Similarly, developers and test/QA personnel. I am sure you would have come across such inconsistencies in a lot of scenarios starting from developing a product to testing. This is just one of the issues that containers tackle.

What are docker containers ?

Docker is a platform for developing, shipping, running applications using container virtualization technology. Docker aids in separating your application from your infrastructure and helps in treating your Infrastructure like the way you would treat any managed application. Docker is one of the most commonly used Containerization tools.

Docker containers have demonstrated that containerization can drive the scalability and portability of applications. Developers and IT operations are

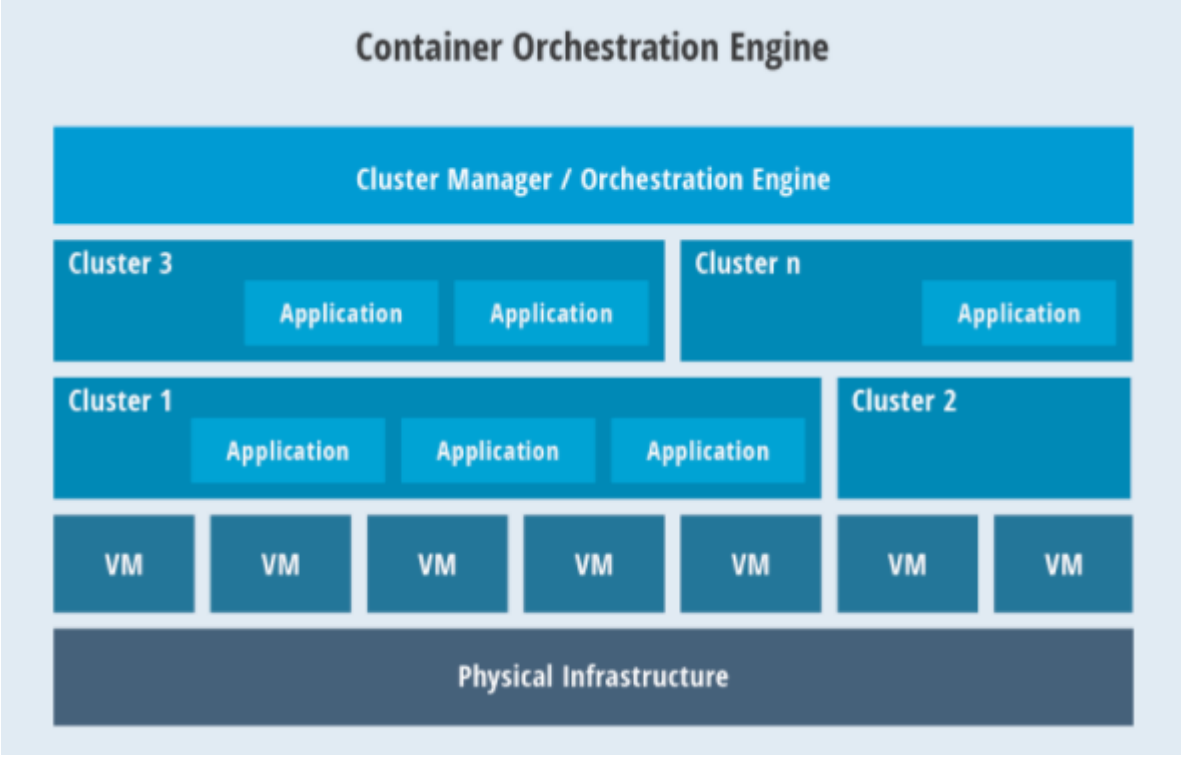
turning to containers for packaging code and dependencies written in a variety of languages. Containers are also playing a crucial role in DevOps processes. They have become an integral part of build automation and continuous integration and continuous deployment (CI/CD) pipelines.

Docker aims to provide a lightweight way to create containers to manage and deploy your applications with isolation and security where you can get more out of your hardware.

Kubernetes

Kubernetes is an open source container orchestration tool designed to automate deploying, scaling, and operating containerized applications. Why is this here ? It is here because “Distributed Management and Orchestration” is an integral part of making your application Cloud Native.

Kubernetes was born from Google’s 15-year experience running production workloads. It is designed to grow from tens, thousands, or even millions of containers. Kubernetes is container runtime agnostic.



Kubernetes’ features provide everything you need to deploy containerized applications. Here are the highlights:

Container Deployments & Rollout Control

Describe your containers and how many you want with a “Deployment.” Kubernetes will keep those containers running and handle deploying changes

Kubernetes will keep those containers running and handle deploying changes

(such as updating the image or changing environment variables) with a "rollout." You can pause, resume, and rollback changes as you like.

Resource Bin Packing

You can declare minimum and maximum compute resources (CPU & Memory) for your containers. Kubernetes will slot your containers into where ever they fit. This increases your compute efficiency and ultimately lowers costs.

Built-in Service Discovery & Autoscaling

Kubernetes can automatically expose your containers to the internet or other containers in the cluster. It automatically load-balances traffic across matching containers. Kubernetes supports service discovery via environment variables and DNS, out of the box. You can also configure CPU-based auto scaling for containers for increased resource utilization.

Heterogeneous Clusters

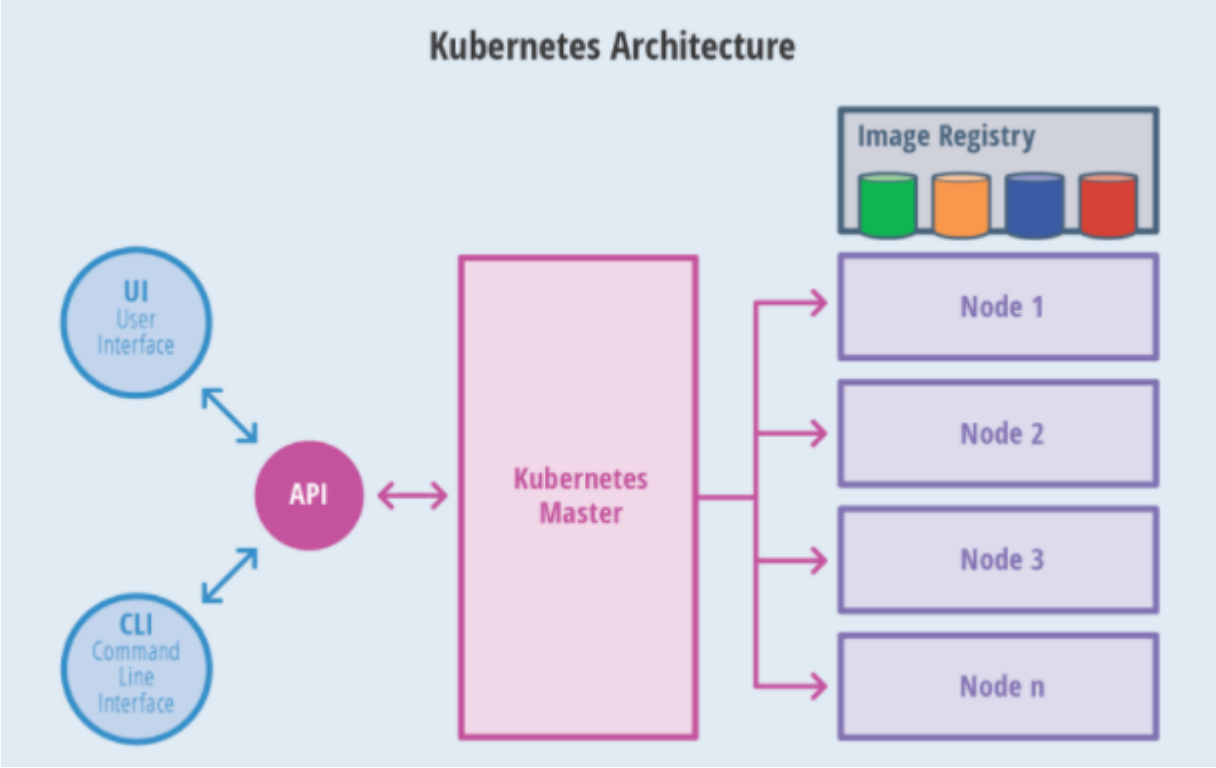
Kubernetes runs anywhere. You can build your Kubernetes cluster for a mix of virtual machines (VMs) running the cloud, on-prem, or bare metal in your datacenter. Simply choose the composition according to your requirements.

Persistent Storage

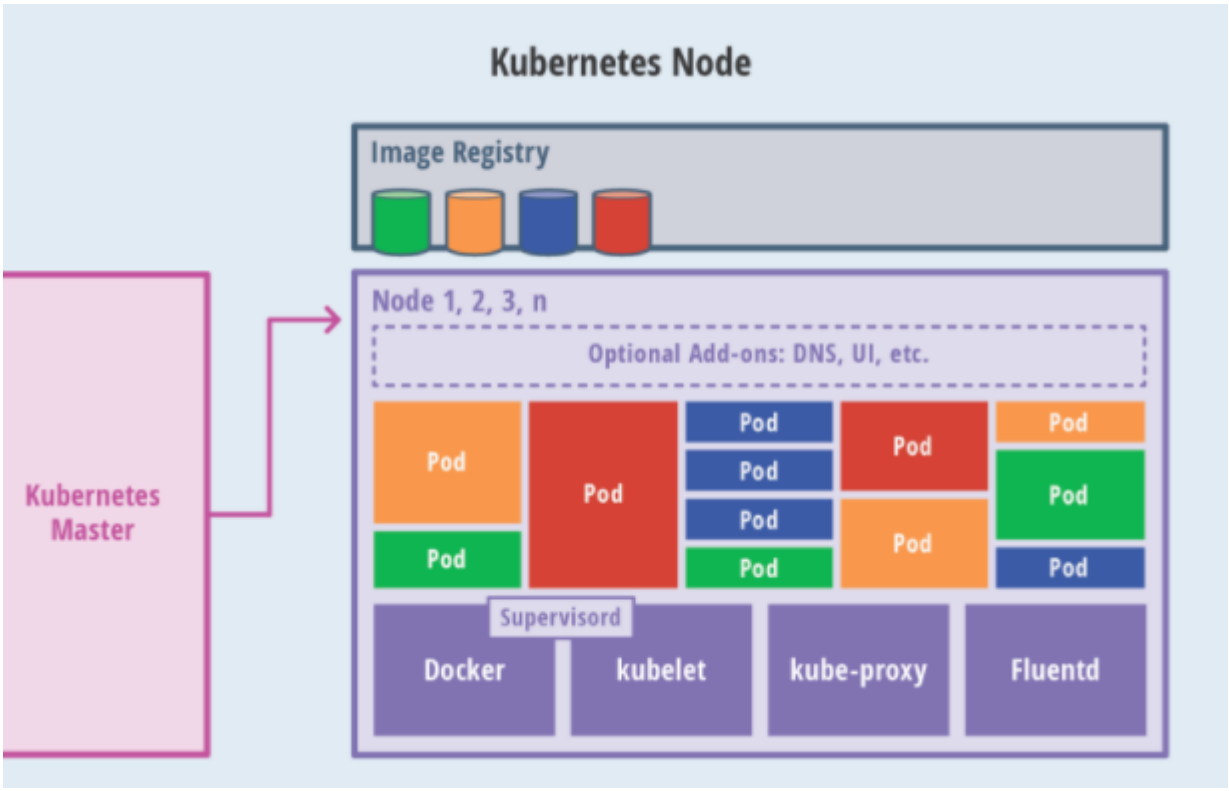
Kubernetes includes support for persistent storage connected to stateless application containers. There is support for Amazon Web Services EBS, Google Cloud Platform persistent disks, and many, many more.

High Availability Features.

Kubernetes is planet scale. This requires special attention to high availability features such as multi-master or cluster federation. Cluster federation allows linking clusters together so that if one cluster goes down containers can automatically move to another cluster.



These key features make Kubernetes well suited for running different application architectures from monolithic web applications, to highly distributed microservice applications, and even batch driven applications.



Kubernetes “clusters” are composed of “nodes.” The term “cluster” refers to “nodes” in the aggregate. “Cluster” refers to the entire running system. A “node” is a worker machine Kubernetes. A “node” may be a VM or physical machine. Each node has software configured to run containers managed by

machine. Each node has software configured to run containers managed by Kubernetes' control plane.

The control plane is the set of APIs and software that Kubernetes users interact with. The control plane services run on master nodes. Clusters may have multiple masters for high availability scenarios.