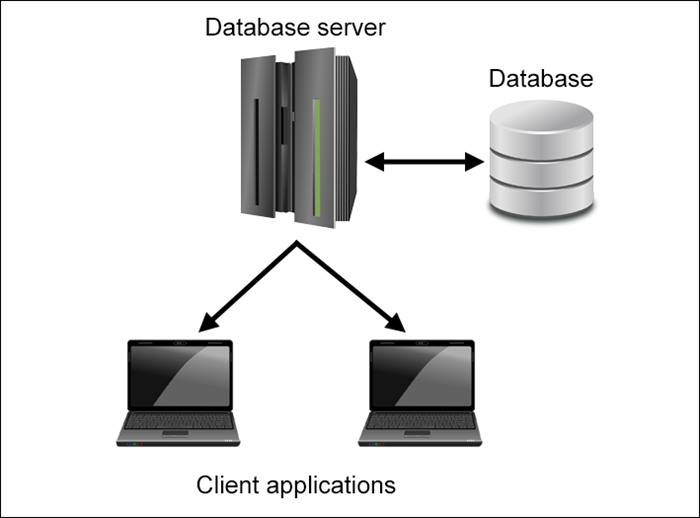
**The Beginner's Guide to Kubernetes: Understanding the Basics of Container Orchestration**

The technology has been progressing with ground breaking evolutions and every technology needs an infrastructure and application support to cater customer’s needs.

Have you ever wondered how the application is working and where it has been deployed? Well, earlier applications were being deployed on a few servers and these servers can be used for frontend or database etc. These servers can have Linux, Windows or Mac as an operating system to support our application.

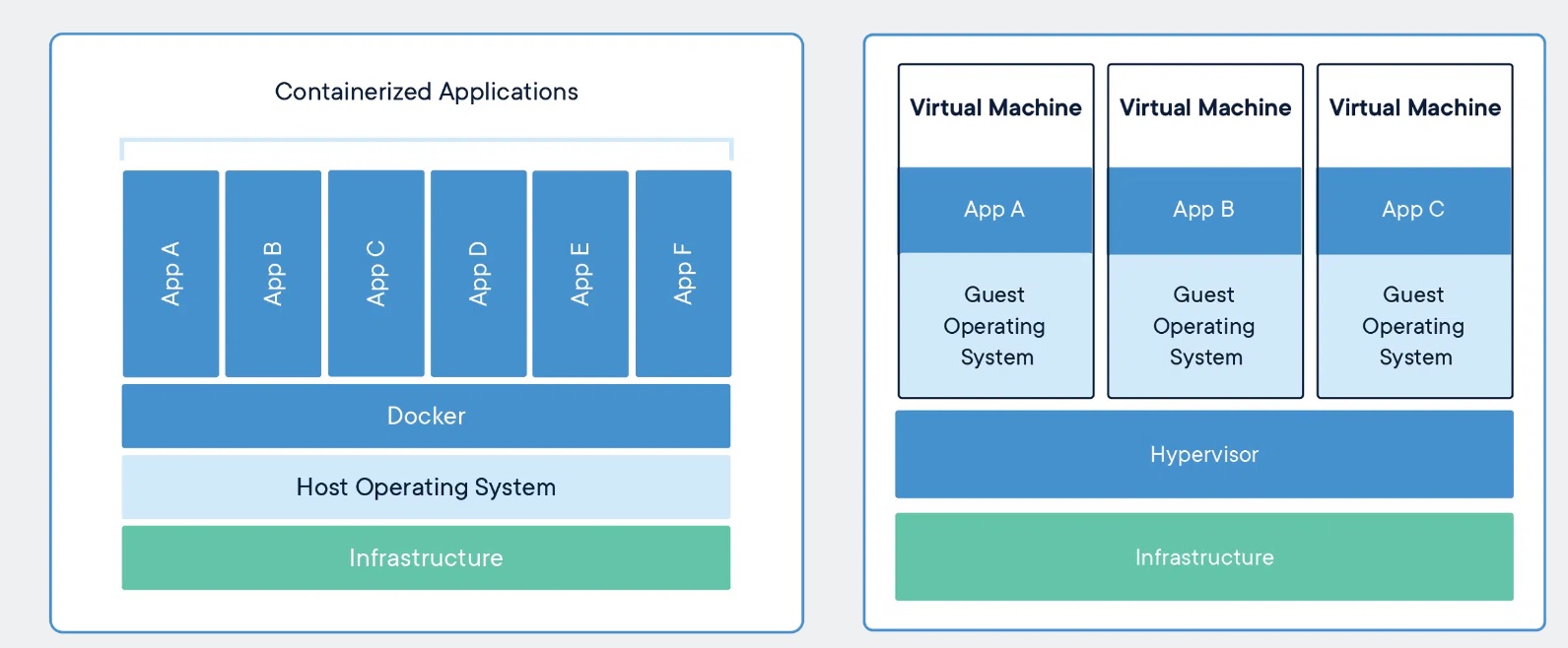


But why do we need one server for a dedicated application? Can we use one single server for multiple applications to save our resources and to meet customer’s expectations? Well we have “containers” to achieve the same.

**What is a container?**

Docker introduced containers in 2013 which are used to package the code and all the dependencies to run the application in different environments. Instead of deploying the application on a single dedicated server, now we have the lightweight containers which will be used to deploy the applications.

It leverages computing concepts like cgroups and namespaces which are used to assign the resources and isolation respectively. Containers virtualize the operating system which makes it a more portable and efficient solution. It can run on a physical server, virtual server or on a cloud.



Containers have multiple benefits over a virtual machine.

1. **Better resource pooling**: Containers can be used for temporary and expandable services to meet each client’s demands, hence it provides better resource pooling.
2. **Lightweight**: Containers are lightweight in nature and images are in megabytes rather than gigabytes.
3. **Faster Deployment**: Containers start in milliseconds hence resulting in faster deployment as compared to virtual machines.
4. **Cost Effective Solution:** A single system can have multiple containers which are using the resources in an efficient way and providing us a cost effective solution.
5. **Portable**: Containers are highly portable as the images can be easily pushed or pulled to different cloud providers or registries.

Still containers have many disadvantages:

1. **High availability**: Containers are running on a virtual machine or a bare metal device, if the machine is down then our containers are also terminated hence our application will not be working. So providing high availability is an issue as far as the container is concerned.
2. **Autoscaling**: As the traffic grows, containers will try to consume the resources as much as sufficient to support the incoming traffic. But it would be freezed or terminated if the traffic is out of control. So autoscaling can not be achieved using containers alone.
3. **Networking**: Docker is responsible to provide the IP addresses to the containers. If the container is restarted or created again, they will get a new free IP address which might create issues for us to access using a new IP address.
4. **Auto-Healing:** Containers are devoid of self healing feature, if the container is down then we have to start them again manually.
5. **Complexity**: Containers are simple but managing containers at scale will become more complex.

Now we have understood how containers were being used to run an application but with many limitations. To overcome these limitations, we need an orchestration tool to manage all the containers running in a cluster.

There are many orchestration tools available in the market like Docker Swarm, Mesos and Kubernetes. Out of them, Kubernetes is very famous and widely used by many companies in their production environment.

As mentioned above, we have other orchestration tools which are:

1. **Docker Swarm**: it is a container orchestration tool which is a part of Docker family. It is a simple tool which can be used for small and medium sized organisations. It offers the features of service discovery and load balancing. But, as far as scalability is concerned it becomes difficult for a large organisation. It lacks the robust features, community support which we can get in Kubernetes.
2. **Apache Mesos**: THis is another orchestration tool which is highly scalable and well suited for a large scale organisation. It is mostly used where machine learning and big data processing is being implemented. As compared to Docker swarm, it provides advanced features like task management, automatic resource allocation and fault tolerance. It requires a deep understanding of distributed systems and it also has limited support from community and third party tools.
3. **Nomad**: Just like Docker Swarm, this tool provides a lightweight and flexible platform to manage containers. Hence it is simple and easy to use too. It is also used for small and medium sized organisations. It lacks robustness and scalability hence, it is difficult to handle a large organisation having complex applications. Community and third party tool support is also not provided.
4. **Amazon ECS**: This tool is integrated with AWS cloud platform which is beneficial for the organisations using the AWS services already. It is quite simple to set up and easy to use hence, it can be used for small and medium sized organisations. However, it is not recommended for large organisations as it lacks scalability and does not have the community and third party tool support. Organisations need a deep understanding of AWS in order to use this service efficiently.
5. **Kubernetes :** Kubernetes is an orchestration tool which was started by Google in 2014 and it has been used by Google in their production environment for many years. It is an open source tool for managing and deploying containers in a cluster. It has become more popular in the world of Devops and cloud computing. Kubernetes provides a system to manage containers which allows us to deploy,manage and scale complex applications.

**Why Kubernetes:**

Kubernetes has become the most popular orchestration tool nowadays and it offers a number of benefits which helps the organisations to manage the complex applications. Here are few reasons which makes it more powerful:

1. **Scalability**: It is one of the best features of Kubernetes because organisations expect to scale their application based on demand. Pods can be added or removed dynamically to handle the traffic spikes or any fluctuations in demand.
2. **Automation**: Nowadays, organisations are moving towards automation and Kubernetes is a best fit in this arena. Kubernetes automates the tasks like deploying a pod, scaling the application, monitoring the health of the pods etc.
3. **Resiliency**: Kubernetes has features like rolling updates, self healing, automatic failover which helps the organisations to keep their application live all the time.
4. **Portability**: Kubernetes can work on different environments or different cloud platforms like AWS, GCP, Azure etc. this makes it easier to move the applications from one environment to another.
5. **Community Support**: kubernetes gets support from a community of developers and users who contribute for bug fixes, new features or any improvement to the platform. So the main goal of this community is to improve and evolve this platform to meet user’s expectation.

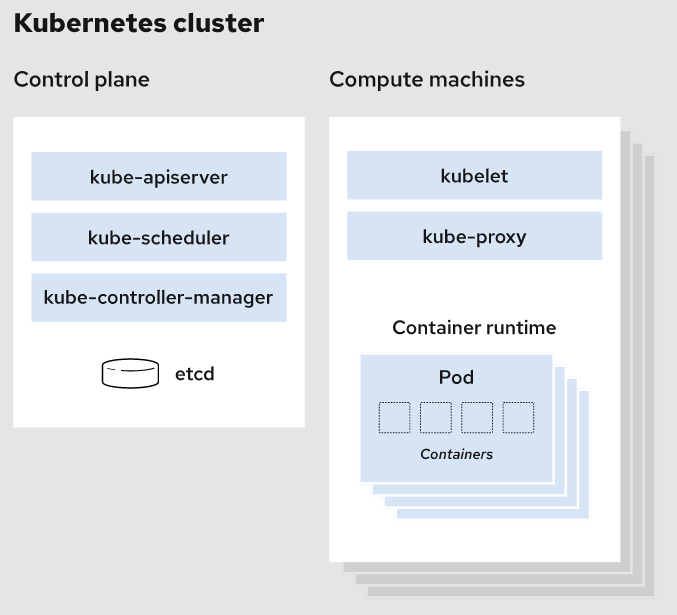
**How does Kubernetes work?**

Kubernetes manages the containerized application in a cluster of nodes. Every node runs a container runtime like Docker and the nodes are managed by Kubernetes control plane. This control plane has management components which are responsible for monitoring the cluster, deploying the pods on a specific node and scaling the pods. Basically it is orchestrating the pods running within a cluster.

Kubernetes supports the declarative approach to manage the cluster’s state. Kubernetes always makes sure that the actual state of the cluster matches with our desired state. Kubernetes makes it easy to manage a complex application with a cluster.

**Kubernetes Architecture:**

Kubernetes has several management components which are used to manage and orchestrate the pods running within a cluster. Let’s find out more about these components below:



1. **API server**: This is the main component that exposes the Kubernetes API and responsible for validating and processing API requests, and for communicating with other components in the control plane.
2. **ETCD**: It is a distributed key-value store which is used to store the information in key-value form. It stores the actual state of the cluster and only the Kube API server communicates with ETCD to get the information.
3. **Kube-scheduler**: this component is responsible for scheduling pods on nodes in the cluster. It actually decides where the pod should be scheduled but not actually schedules it. It takes that decision on the basis of some factors like resource availability, quality of service requirements and affinity rules.
4. **Kube controller Manager**: This component is basically a collection of controllers which monitor the state of the cluster and make the adjustment as required. One of the controllers is a node controller which is responsible for monitoring the state of the nodes and keeping a check on the health of the nodes. It makes sure that the desired number of nodes are available.
5. **Kubelet**: It is an agent which runs on each node and is responsible for managing the state of pods. It communicates with the Kube API server for the instructions to start a pod on a specific node and report back to the API server about the state of pods and nodes.
6. Container Runtime: This is the software which every node needs to start a container on each node. It takes the instructions from Kubelet to create a container. We have Docker, containerd, CRIO but Docker is the most commonly used container runtime with Kubernetes.

So all the above components work together to orchestrate pods in the cluster and makes it easy to deploy, scale, delete or monitor your containerized applications.