**Kubernetes –Notes**

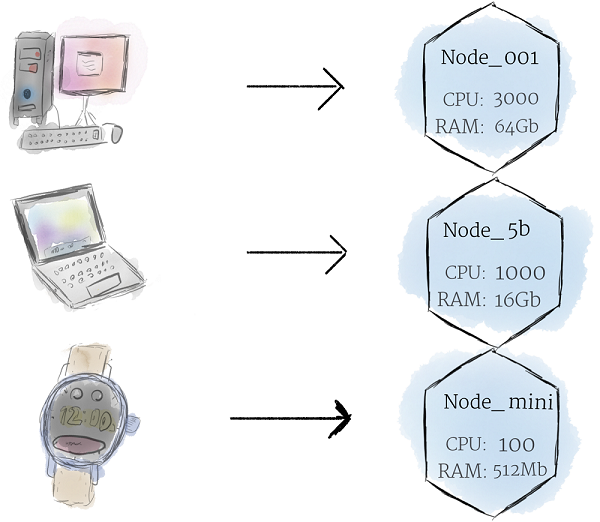
Kubernetes is a container management technology developed in Google lab to manage containerized applications in different kind of environments such as physical, virtual, and cloud infrastructure. It is an open source system which helps in creating and managing containerization of application.

**Kubernetes Hardware: - Nodes, Cluster, Persistent Volumes**

Kubernetes itself doesn’t need hardware, but the functioning system needs all the hardware.

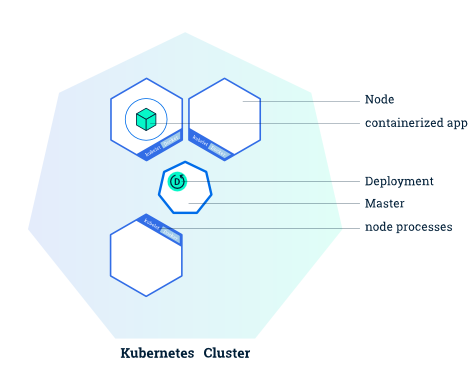
**Nodes**

What is a node in Kubernetes? One of the smallest units of computing in Kubernetes is known as a node. It is a singular machine and resides in a cluster. Node doesn’t necessarily need to be a physical machine or a part of the hardware. It is either a physical machine or a virtual machine. For the data center, a node is a physical machine. Similarly, for Google Cloud Platform, a node is a virtual machine



**Cluster**

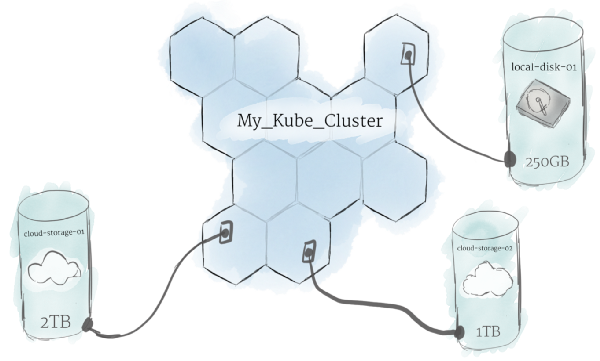
cluster is a collection of multiple nodes. All the nodes pool together their resources and together make a powerful machine



The cluster is intelligent. Do you know why? When the programs are deployed onto the cluster, it dynamically handles the distribution. In short, it assigns tasks to individual nodes. In between the process, if any node is added or removed, the cluster shifts the work as per the need. The programmer need not concentrate on such things like which individual machine is running the code, etc.

#### The Persistent Volumes

As mentioned, the programs run on the cluster and are powered by the nodes. But they don’t run on specific nodes. The programs run dynamically. Thus, there is a need for storing the information and it can’t be stored randomly in any file system. Why? For example, a program saves the data to a file. But later on, the program is relocated to another node. Next time when the program needs the file, it won’t be at the expected place. The location address is changed. To solve this problem, the traditional local storage related to each node is considered as a temporary cache for holding programs. But any locally saved data can’t be expected to persist.

So, who stores data permanently? Yes, the persistent volumes store it permanently. The cluster manages the CPU and RAM resources for all the nodes. But, the cluster is not responsible for storing data permanently in the persistent volume. The local drives and cloud drives can be attached to the cluster like a persistent volume. It is something like plugging an external drive to the cluster. The persistent volumes offer a file system. It can be mounted to the cluster without being associated to any specific node. 

**Kubernetes Software’s: - Containers, Pods, Controllers (Deployments, Replication sets, Services, Ingress, daemon sets, jobs, etc...)**

The overall concept of Kubernetes is based on software. So, this is the main part of Kubernetes.

#### Containers

In Kubernetes, Linux containers host the programs. These containers are globally accepted and already have pre-built images. The images can be deployed on Kubernetes. Do you know what is containerization? It allows you to create Linux execution environments.

The programs, as well as its dependencies, are packed in one single file and shared on the internet. So, anyone can download the container and deploy it on their infrastructure as per the requirement. Deployment is hassle-free with just a little setup. A container can be created with the help of a program. This enables the formation of effective CI and CD pipelines.

The containers are capable of handling multiple programs. But it is recommended to limit one process per container because it helps in troubleshooting. Updating the containers is easy and the deployment is easy if it is small. It is better to have many small containers, rather than a big one.

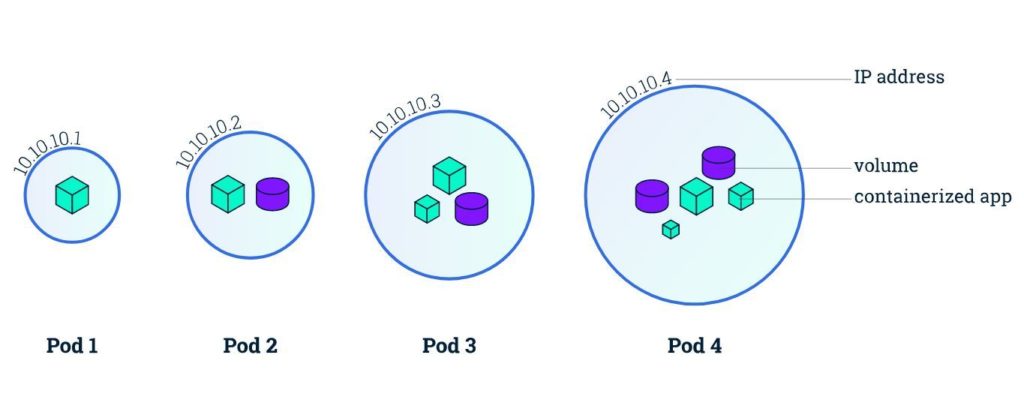
#### Pod

Kubernetes has some unique features and one of them is that it doesn’t run the containers directly. It rather wraps up one or more containers into a pod. The concept of a pod is that any containers within the same pod use the same resources and the same local network.

The benefit is that the containers can communicate with each other easily. They are isolated but are readily available for communication.

The pods can replicate in Kubernetes. For example, an application becomes popular and a single pod can’t sustain the load. At that moment, Kubernetes can be configured for deploying new replicas of the pod as per the requirement.

But it is not necessary that replication occurs only during heavy load. A pod can replicate during normal conditions as well. This helps in uniform load balancing and resisting failures.



Pods can hold multiple containers but one should limit one or two if possible. The reason is that the pods scale up and down as a single unit. The containers within the pod must also scale together with the pods. Their individual needs aren’t important at this stage. On the other side, this leads to wastage of resources and expensive bills.

To avoid all this, limit the pods to few containers. If you ever come across the term “side-cars”, it means helper containers. So, there is the main process container and there could be some helper containers

#### Deployment

#### Level Up The Kubernetes Bible for Beginners & Developers Article Image

#### If you notice, pods are the basic units in Kubernetes. But they aren’t launched directly on a cluster. They are managed by more than one layer of abstraction. This overall make “deployment”. The main purpose is to declare the number of replicas running at a time.

When the deployment is added, it spins up the number of pods and monitors them. Similarly, if the pod doesn’t exist anymore, it deployment re-creates it.

The fun part is that with deployment, there is no need to deal with pods. By declaring the state of the system, everything is managed automatically.

#### Services

Kubernetes [Pods](https://kubernetes.io/docs/concepts/workloads/pods/pod/) are mortal. They are born and when they die, they are not resurrected. [Replica Sets](https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/) create and destroy Pods dynamically (e.g. when scaling out or in). While each Pod gets its own IP address, even those IP addresses cannot be relied upon to be stable over time. This leads to a problem: if some set of Pods (let’s call them backends) provides functionality to other Pods (let’s call them frontends) inside the Kubernetes cluster, how do those frontends find out and keep track of which backends are in that set?

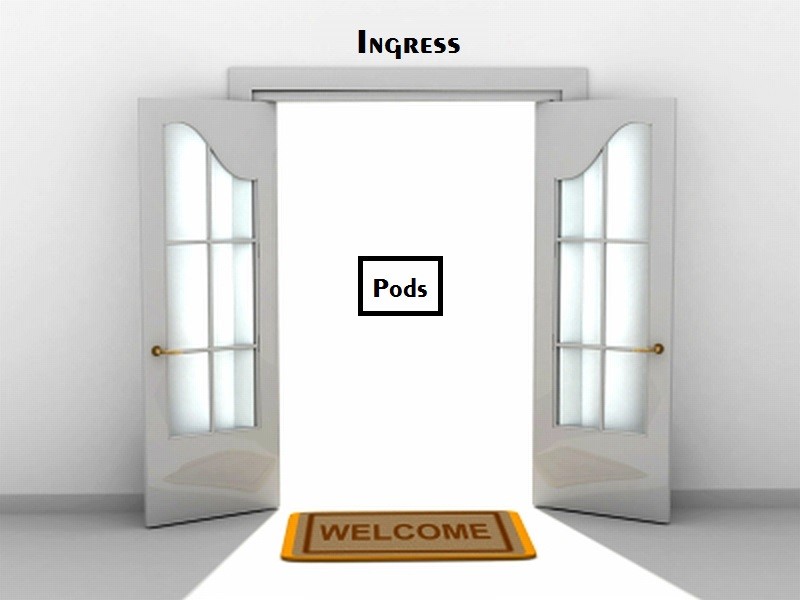
Enter Services.

A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them - sometimes called a micro-service. The set of Pods targeted by a Service is (usually) determined by a [Label Selector](https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#label-selectors).

**Types of Services: - ClusterIP, NodePort, LoadBalancer, External Name**

#### The march towards success with Ingress

#### We have discussed all the basic concepts of Kubernetes. Using them, you can create a cluster of nodes. Once the cluster is made, it is time to launch deployments of pods on the cluster. But how will you allow external traffic to your application? We haven’t discussed this yet.



#### As per the concept of Kubernetes, it offers isolation between pods and the outside world. To communicate with a service running within a pod, the outsider needs to open a channel. The channel is a medium of communication. It is known as “ingress”.

#### There are numerous ways for adding an ingress to the cluster. The most common being through an Ingress Controller or Load Balancer.

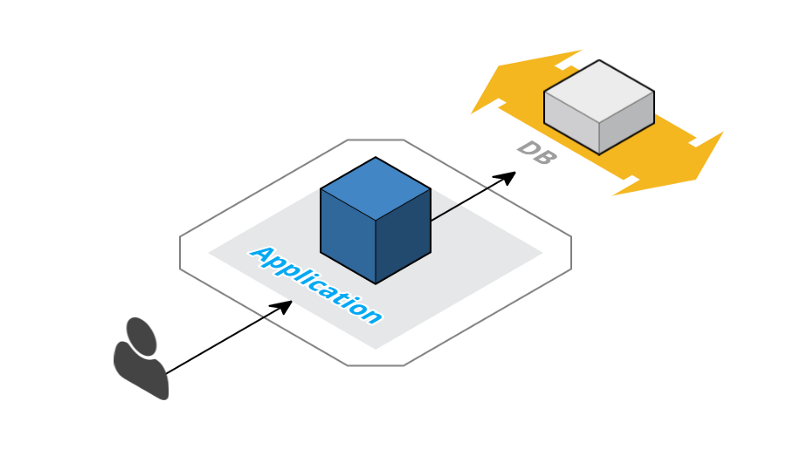
## **How Kubernetes Solves the Problem?**

After discussing the deployment part of Kubernetes, it is necessary to understand the importance of Kubernetes.

#### Container Orchestration & Kubernetes

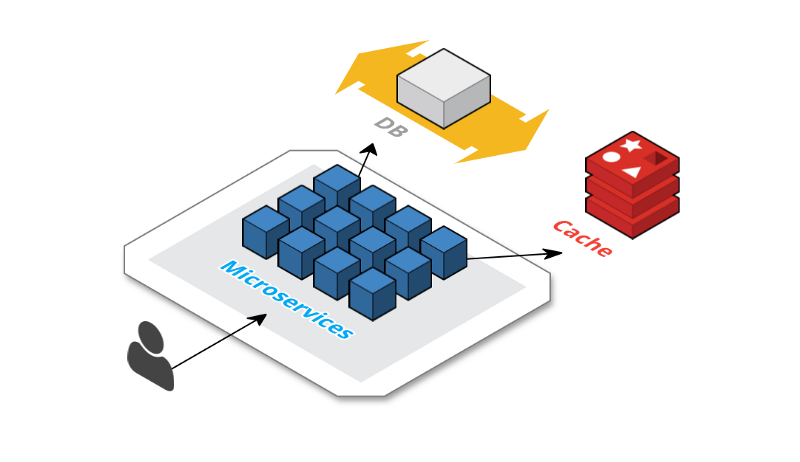
Containers are virtual machines. They are lightweight, scalable, and isolated. The containers are linked together for setting security policies, limiting resource utilization, etc. If your application infrastructure is like the image shared below, then container orchestration is necessary.

It might be Nginx/Apache + PHP/Python/Ruby/Node.js app running on a few containers, communicating with the replicated database. Container orchestration will help you manage everything by yourself.



Consider that your application keeps on growing. For example, you keep on adding more features/functionalities and at some point, in time, you realize that it has suddenly become a huge monolith.

Now, it is impossible to manage the vast application because it eats up your CPU and RAM. So you finally decide to split the application into smaller chunks. Each one of them with a specific task. Now, your infrastructure looks like this:



So, you need a caching layer with some queue system for a better asynchronous performance. Now, there are challenges like service discovery, load balancing, health checks, storage management, auto-scaling, etc.

Under all such hectic circumstances, who will come to your help? Yes, container orchestration will be your savior! The reason is that container orchestration is extremely powerful and solves most of the challenges.

# How Does the Kubernetes Networking Work?

#### The first part includes containers and pods. The second part includes service examination and the extraction layers. They allow the pods to be ephemeral. The third part includes ingress and accumulating traffic to the pods from outside the cluster.

#### Level Up How Does The Kubernetes Networking Pods Work? : Part 1 Article Image