Kubernetes

Multiple Vm Machines with Vagrant

Can you spin more than one server using vagrant?

```
  # For a complete reference, please see the online documentation
at
  # https://docs.vagrantup.com.
  # multi-machine config
  config.vm.define "master" do |master|
    master.vm.box = "ubuntu/trusty64"
  end

  config.vm.define "node_1" do |node_1|
    node_1.vm.box = "ubuntu/trusty64"
  end

  config.vm.define "node_2" do |node_2|
    node_2.vm.box = "ubuntu/trusty64"
  end
```

The path – vagrant folder (desktop)

PS C:\Users\Forex\OneDrive\Desktop\Worker_Node 1 -Josh>

```
$ vagrant destroy <VM-name>

$ vagrant up
```

Now SSH in to the guest machine with command:

```
$ vagrant ssh
```

Install Apache webserver in it. If the VM is Deb-based, run:

```
$ sudo apt install apache2
```

If it is RHEL-based system, run this:

```
$ sudo yum install httpd
```

Start Apache service:

```
$ sudo systemctl enable --now httpd
```

Revise Docker – 30 Minutes

Revise Docker-Compose – 30 Minutes

Kubernetes Concept

Why Kubernetes?

First let's understand, how our applications have evolved:

Monolithic Software Application

Before Kubernetes, company used to have what we call monolithic application.

Let take an example of Geico which has many applications.

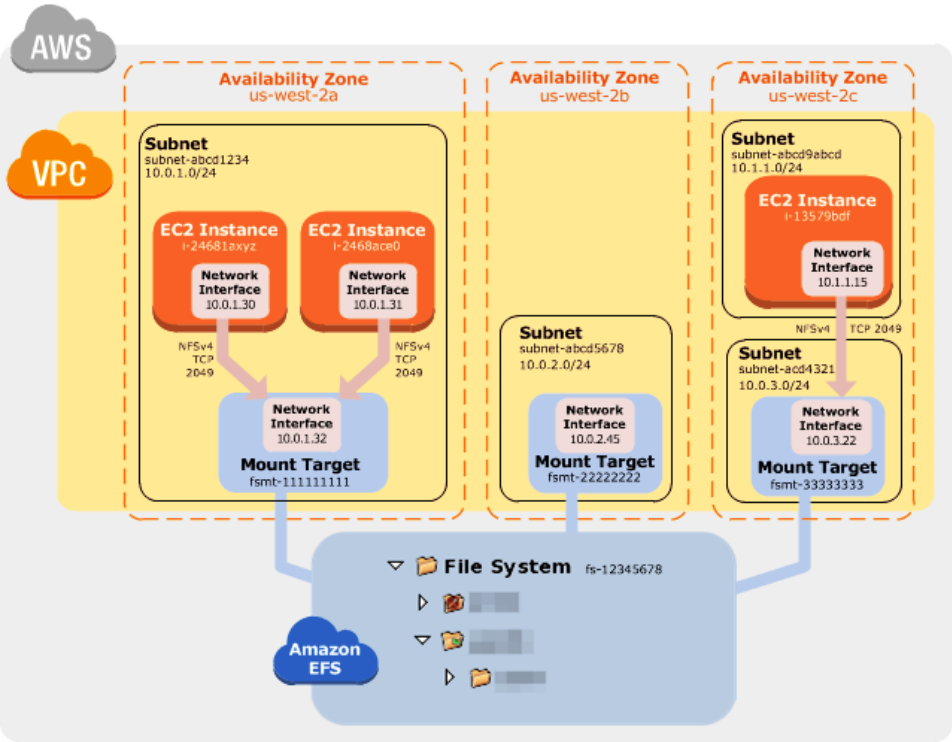
https://www.geico.com/

For instance, in the above, the SNS, Payments etc all were located in the same server.

If you wanted to make a change in any of the application, I have to work on the main code containing all the applications.

This was not effective because in most cases, while making the changes some functions will likely stop working.

No interaction was available among the applications. For instance in AWS, EC2 can interact using EFS while in different availability zone.



There was a huge dependency among the applications because they were all in the same one main code base.

How was this issue solved?

Why Kubernetes?

To counter the problems of Monolithic Application, we came up with a Microservices Architecture

This was solved using microservices.

This is the ability to deploy services separately and ensuring each service has its own code service.

Meaning each service is deployed in different **a server.**

Communication is done using Json, where each server sends files to the other and messages generated can be shared among the application.

The issue of dependency was completely solved.

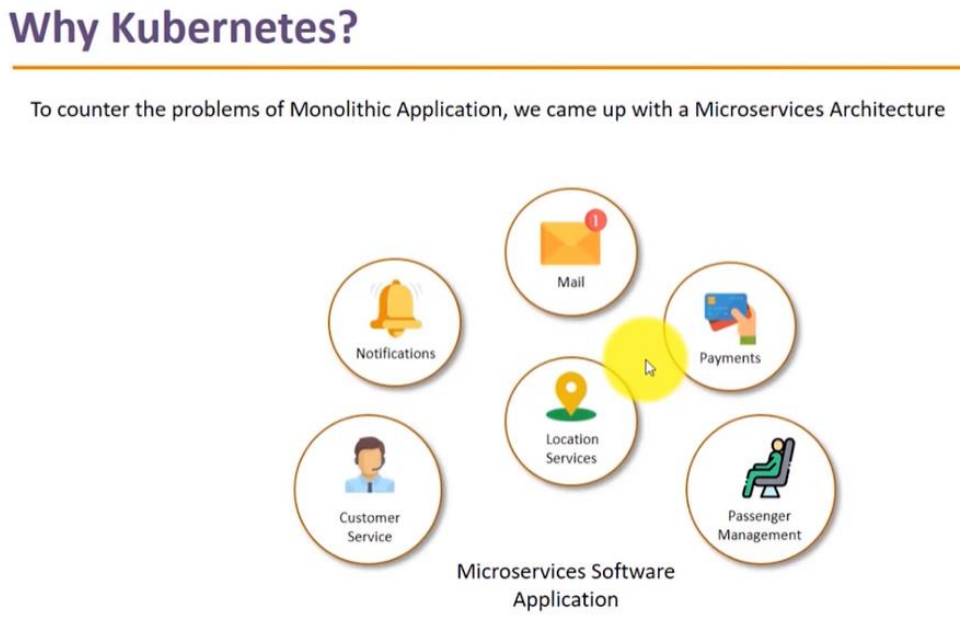
The issue of communication among the applications was also solved.

With microservices, it became easier to make the necessary changes without affecting the rest of the application. You can just correct where the issue is without interrupting with the entire code system.

https://www.geico.com/

Challenges of Microservices

Although microservices solved the dependency, functionality, and communication issues, it became obvious that deploying each service on a separate server was going to be a challenge.



We must deploy servers for the following microservices:

- Mail Server
- Payment Server
- Location Server
- Customer Service server
- Notifications Server
- Passenger Management Server

This is a great way to distribute the services.

However, scaling becomes an issue. This is because some services may not require the entire server to function while others may need more than one server.

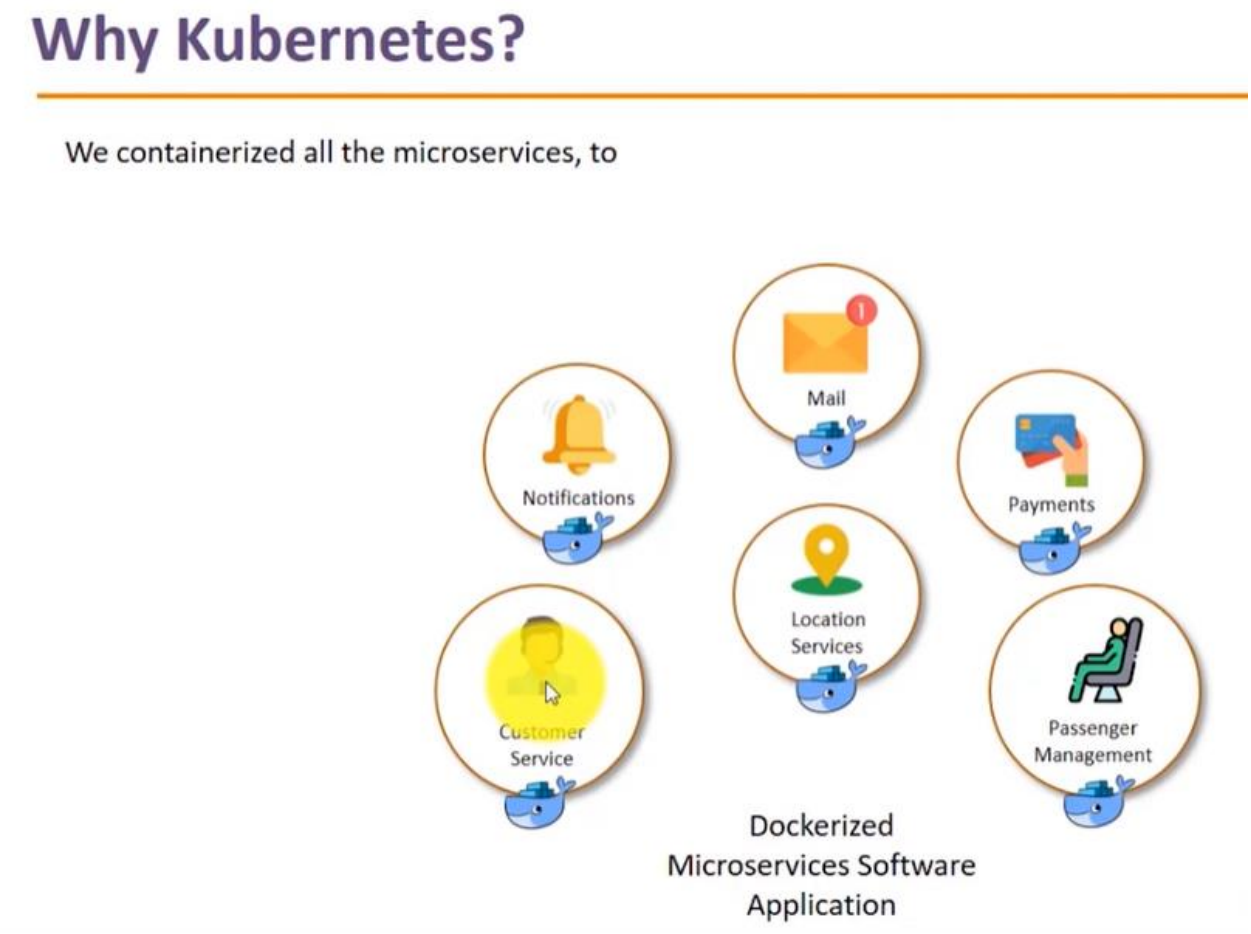
Again, it becomes impossible to deploy anything else in the server because each server is occupying the entire server.

The issues of underutilization of the servers are very frequent.

Downscaling becomes a problem for the servers that are being underutilized because at the end of the day, each service must occupy an entire server.

Upscaling was okay, because you could just add more servers, but downscaling became the main challenge.

Upscaling could be done with just adding another server. Even after adding the second server, the underutilization was still an on the added server/s.



This issue of underutilization of the server was solved using Docker.

Docker give each service a different environment.

These services could be dockerized to exist in one server and still be in their own standardized and unique environment.

This also brought another advantage because the only way the services could be down is if the entire server fails.

An example of a docker application structure

- Ubuntu – mail
- Centos6 – Notifications
- Centos7 – Payment
- Redhat – location services etc

All running on the same system/server.

The issue of server underutilization was solved

Upscaling is also not an issue because you could add containers if needed. (Docker-Compose)

Good job, we have containerized all the applications and they are now independent from each other, can scale, underutilization issue has been solved and we are good to go.

Another problem Arises

Many companies have many microservices

Google can have over 5000 microservices

Geico can have over 300 microservices

Amazon can have over 10,000 microservices

Containerizing 10,000 services is not an issue.

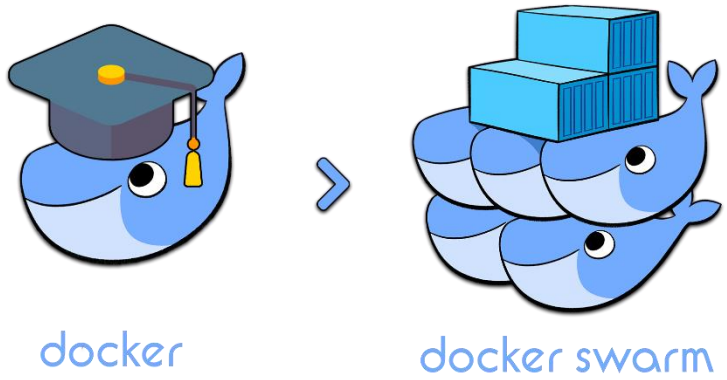
Who is going to manage 10,000 containers to ensure the following?

- Assessment of the containers to make sure they are all available.
- The health check of the container?
- If one container goes down, how do I monitor it?

- If one container goes down, how do I know which one it is?
- If I know which one it is, how do I re-deploy another one?

This is where Kubernetes orchestration tool came in.

You can also use Docker Swarm



However, Kubernetes has way more services than any other container orchestration tool that is out there.

Kubernetes allows a lot of options and flexibility.

Why Kubernetes?

- Each Service's Monitoring is difficult
- Scaling a Particular service based on load is not possible
- Too much manual intervention required for managing containers
- Managing containers on multiple servers become difficult

Without Kubernetes, you must manually go and scale the containers in accordance with the load size.

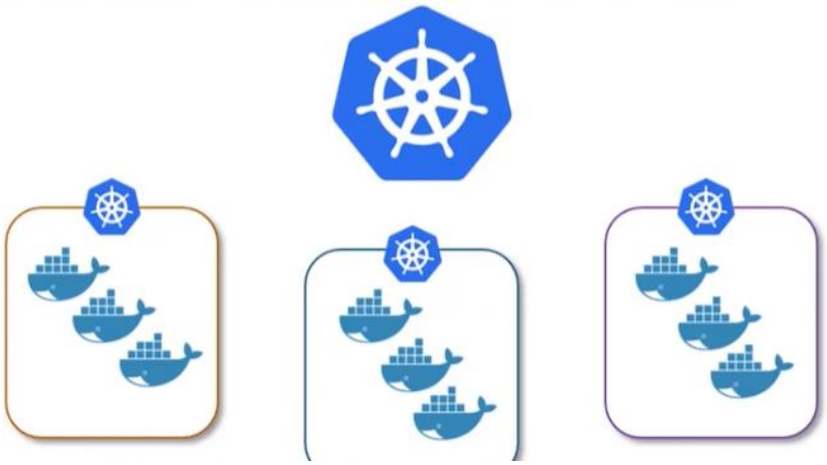
You also must manually remove or add containers

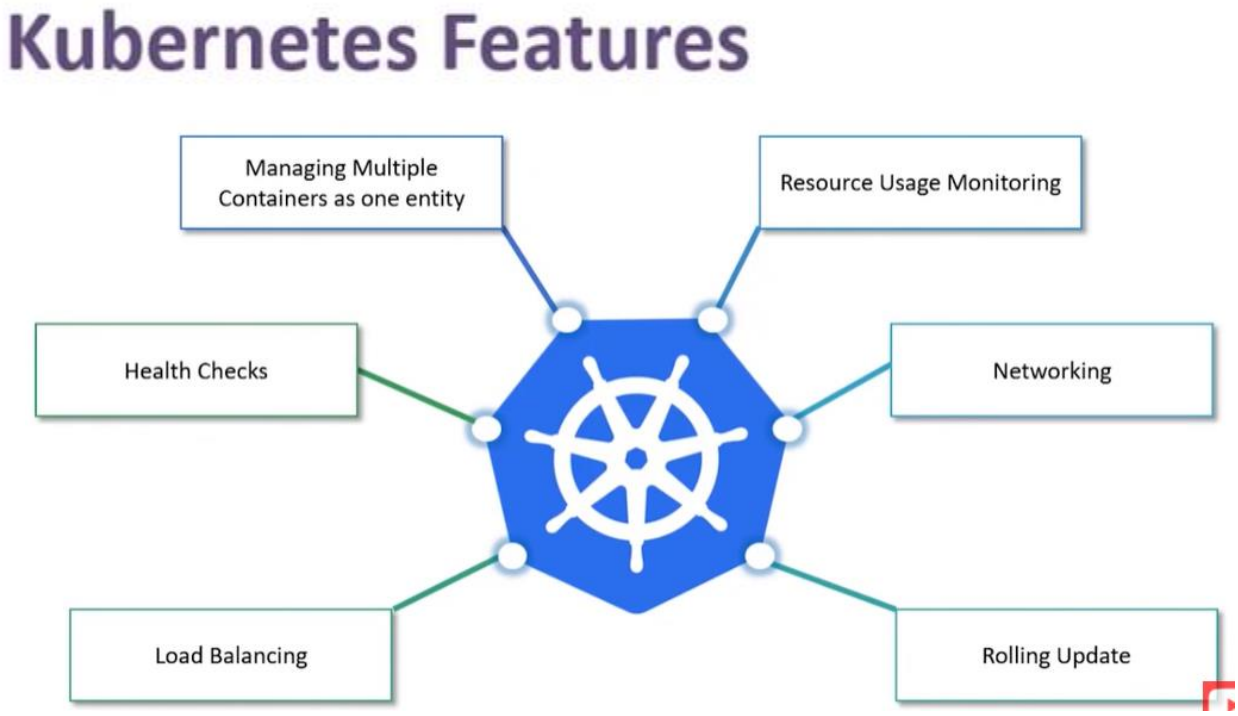
Somebody must monitor the containers performances manually.

It was very difficult to even know how many containers are running in each node.

What is Kubernetes?

Kubernetes being a container orchestration tool, is used when our application is distributed in multiple containers. It's job is to monitor, scale, restart containers automatically even if they are spread across multiple nodes





Kubernetes manages all containers – You do not have to worry about managing containers as Kubernetes will do it. Scaling will be taken care. You just launch the new deployment with number of containers specified and Kubernetes will take care of everything.

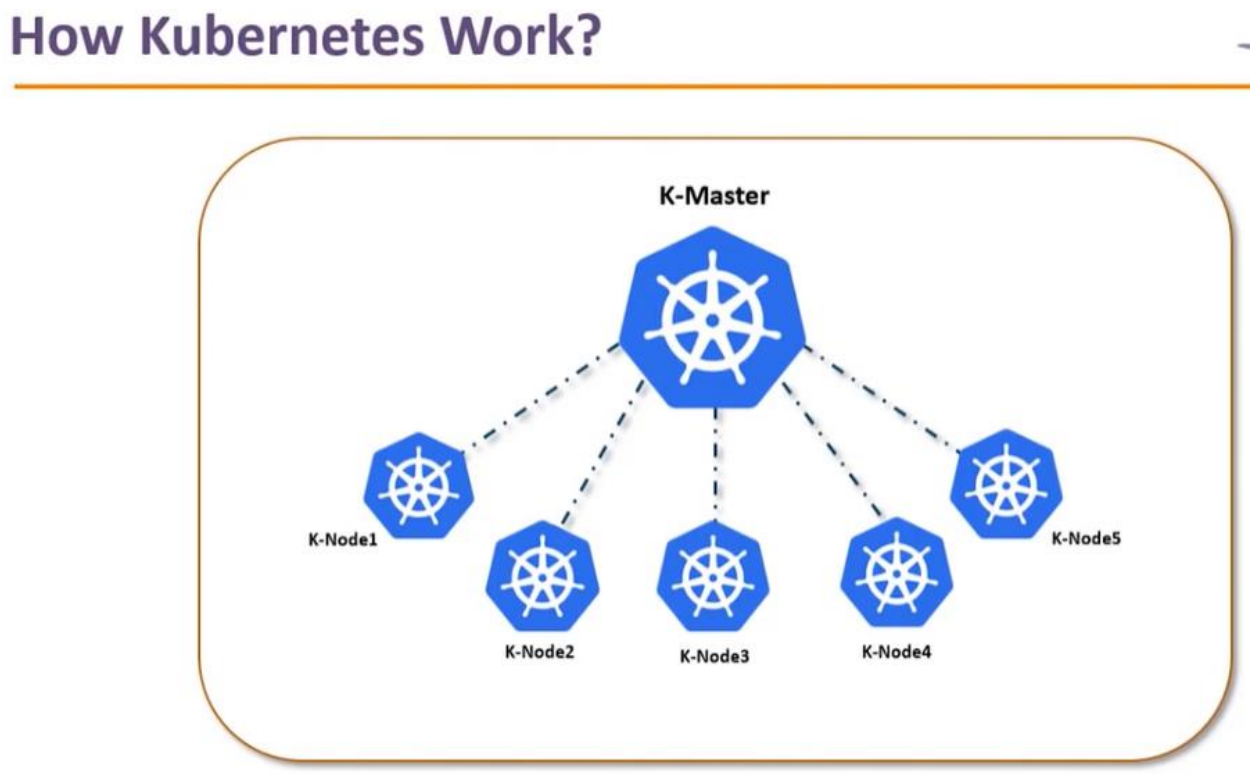
Resources usage was also taken care of. You can define how much resources each container should use and if it goes beyond the defined resources, Kubernetes will launch a new container.

Networking – All containers in Kubernetes cluster can interact as if they are in one network regardless of them being in 100 different nodes.

Rolling updates on the applications or containers is very easy. You can do that using blue-green features in Kubernetes.

Load balancing is another feature of Kubernetes – When having a replica of the containers inside the node, you can balance the load accordingly.

Health check – Kubernetes is self-healing. It checks the health of the containers and if one is not working accordingly, it will launch another healthy container and therefore there is never a downtime.



It is a cluster. You have a master and several nodes

Nodes are the workers

The master schedule deployment of containers, schedule the new containers, monitoring everything and keep track of logs (what needs to be done in the application), while the nodes do the processing of the work that is required by the application. Both the master and the nodes work together, and they act as if they are one system.

You can create a new worker(node) by just using a command through the master.

What is a Pod

A pod is a container.

A single pod has one container.

However, a pod can have more than one container. For instance, there are containers that cannot exist alone.

Like PhP and WordPress Containers.

You can also create a replica in your deployment and a pod can have like 2 to 3 replicas.

You can either deploy a pod or a deployment.

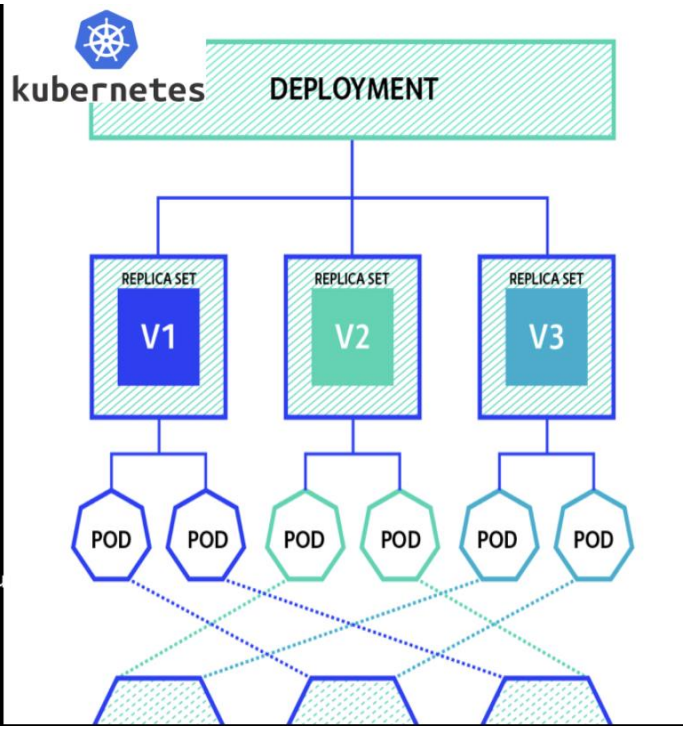
Deployment takes care of the number pods in the cluster.

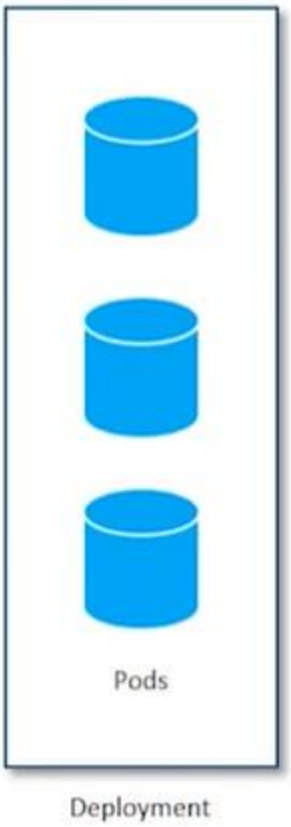
When you create a deployment, it automatically creates pods in the cluster.

While creating the deployment you specify the image to use, in the container and the pods automatically will be created.

Simply, deployment help us to deploy number of pods and the images to be inside the pods.

```
kind: Deployment

metadata:
  # Unique key of the Deployment instance
  name: demo-deployment

spec:
  # 3 Pods should exist at all times.
  replicas: 3

  selector:
    matchLabels:
      app: nginx

  template:
    metadata:
      labels:
        # Apply this label to pods and default
        # the Deployment label selector to this valu
        app: nginx
    spec:
      containers:
      - name: nginx
```





This is a simple deployment

Simply, in deployment you must have

- Number of pods
- Number of Replica
- The image to be in each pod

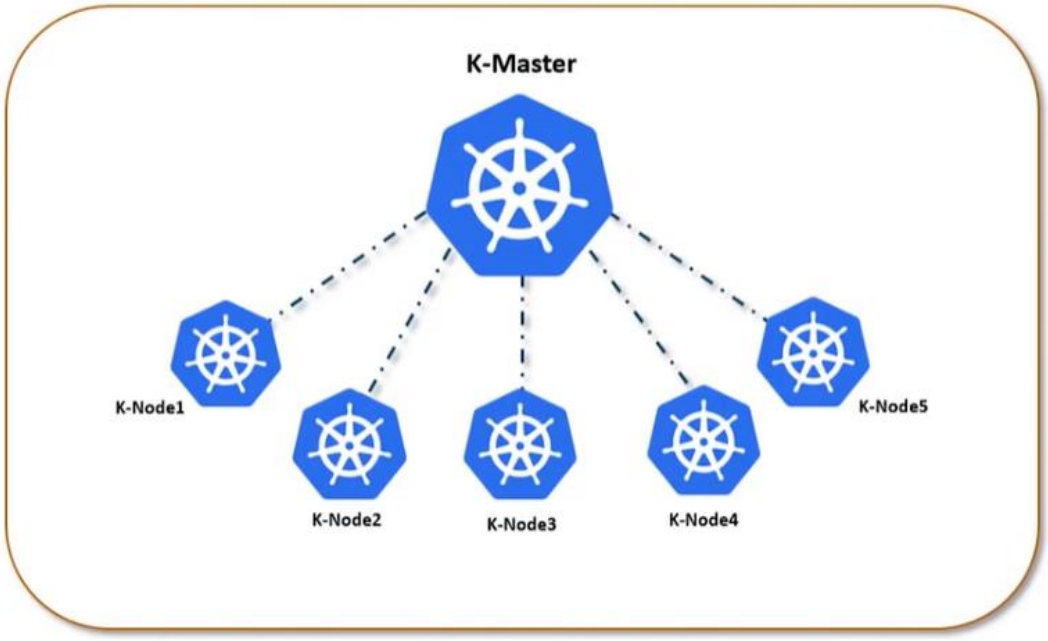
The Kubernetes is always in the server.

How will a user access an application that is inside the pod in Kubernetes cluster?

Let say geico is in pods in Kubernetes cluster

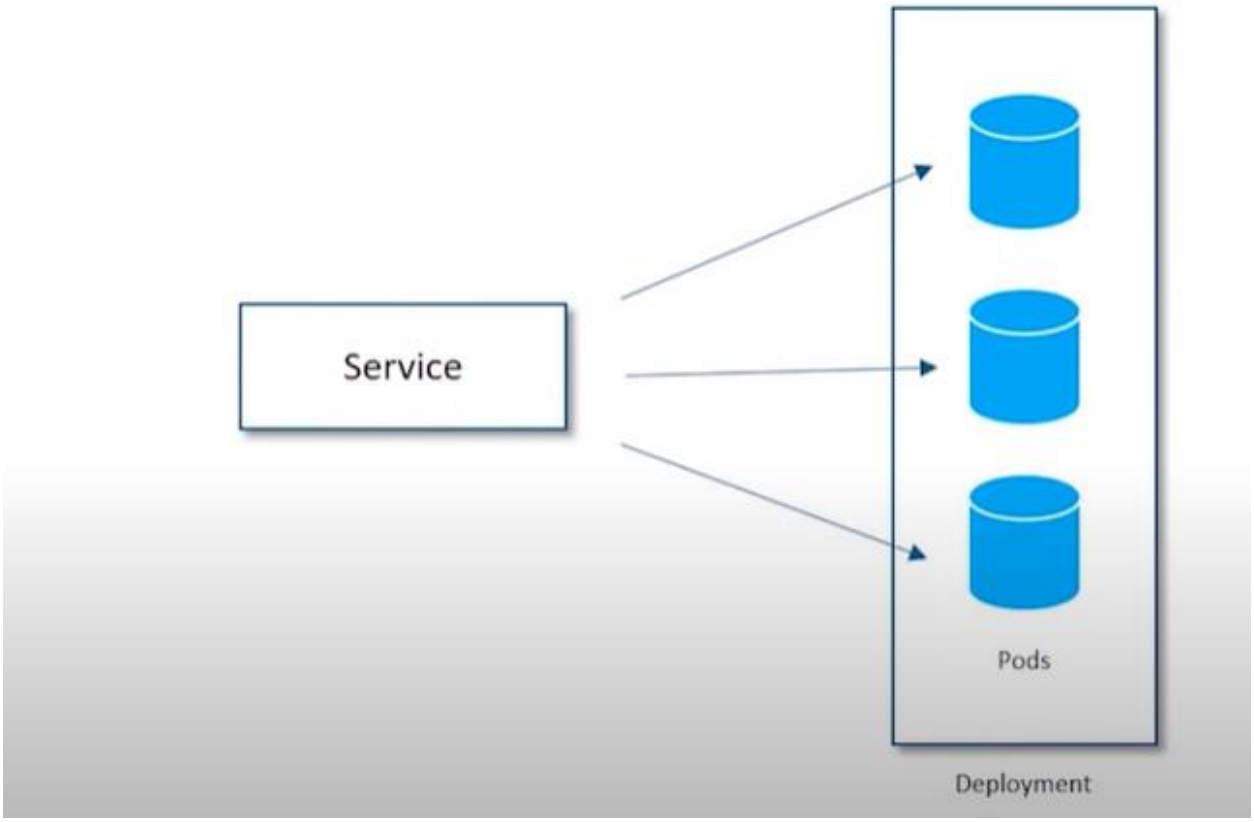
As a user, you will click geico.com on the web browser, which will be pointing to a server that will be having the application. However, the application will be inside a pod and inside the replica if specified in the deployment.

How Kubernetes Work?



The information a user requires can be accessed through any of the IP addresses available on the master or the nodes.

However, you cannot access the Kubernetes cluster (master or the pods) directly.



This is where the importance of the service comes in.

What is a service?

A service is an internal load balancer for all the pods that are running.

Types of Load Balancers

- Application Load Balancer
- Network Load Balancer
- Classic Load Balancer
- Gateway Load Balancers

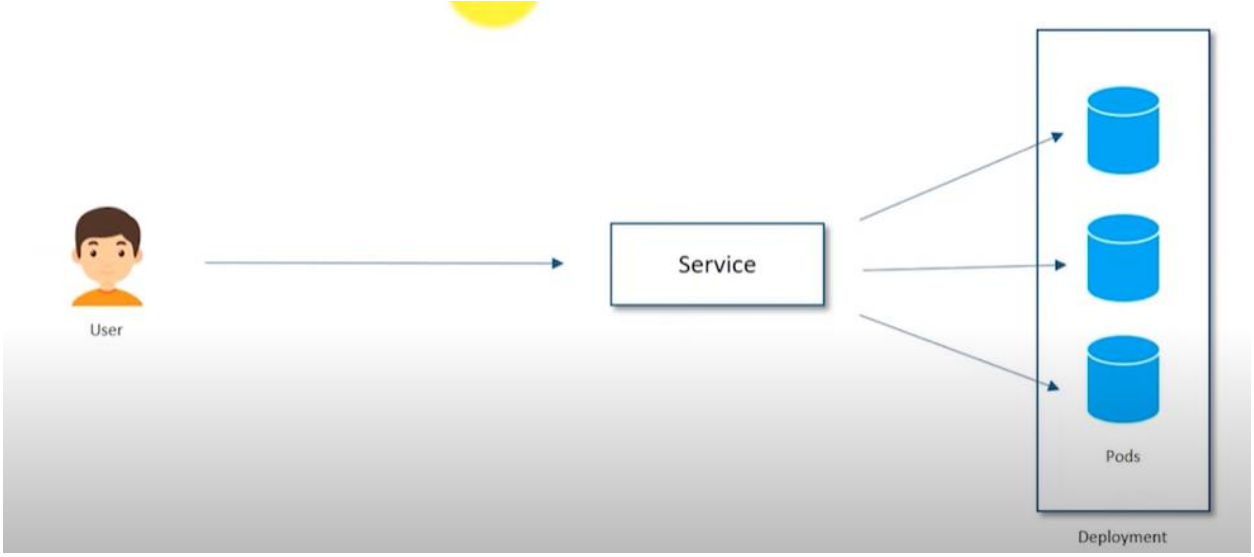
In each deployment there is a service. The service work is just to expose the pods to the users on the web browser.

A problem

Although all the pods **have the same application**, how does the SERVICE know which pod to send the request?

First time request will be distributed randomly.

However, as the requests increase, the service will choose which pods is less active and that where the request will be sent to, for load balancing purpose.



On the web-browser you send a request

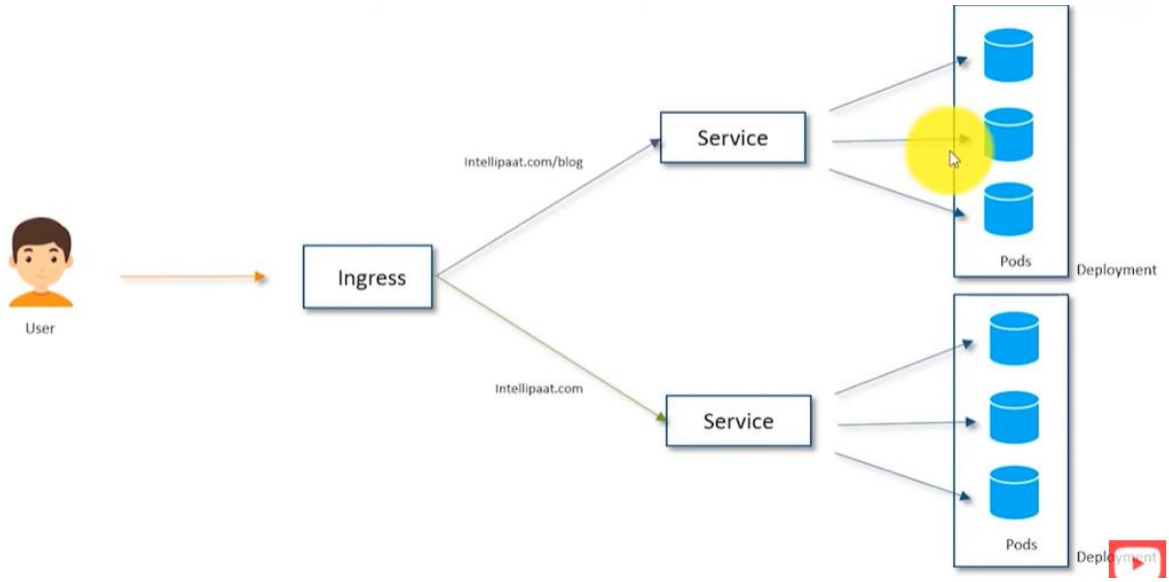
The domain name (special Ip Address) will be pointing to the service which will distribute the load to different pods.

A Problem

What if you want to access information from geico/car insurance and then from Geico/boat insurance?

These are two different services that will be contained in two different pods.

How does the service know which pod to send the request in the cluster?



Ingress is the solution.

It will direct the request to the correct service.

Geico.com/car insurance

Geico.com/boat insurance

Ingress will receive your request through the Url and determines which service to send the request to.

Some hands on

Kubernetes Installation

There are numerous ways to install Kubernetes, following are some of the popular ways:

- **Kubeadm** – Bare Metal Installation
- **Minikube** – Virtualized Environment for Kubernetes
- **Kops** – Kubernetes on AWS
- **Kubernetes on GCP** – Kubernetes running on Google Cloud Platform

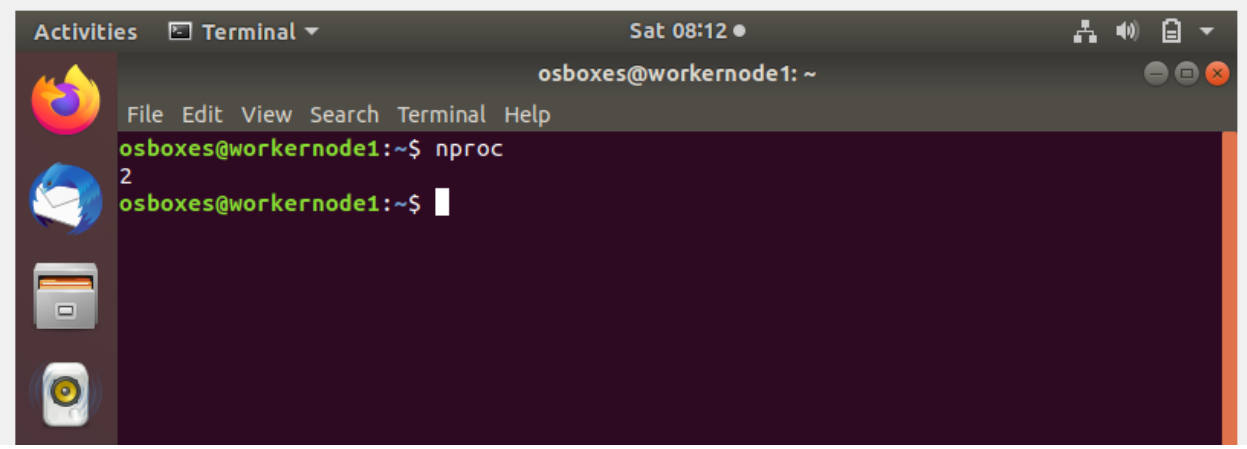
- Any bear metal servers – Any generic server – For instance, a server on Virtual machine-like centos, Ubuntu
- Minikube – virtualized environment – everything is done for you and is great to practice Kubernetes commands.
- Kops – shortcut for installing Kubernetes on aws but only limited to aws and do not have instances that fall under the free tier account.
- Kubernetes on GCP – This is the easiest way to practice Kubernetes. Remember Kubernetes is a product of google. It will work like a charm on GCP.

Install Minikube and Kubectl on Ubuntu 20.04

Increase the memory and number of cpus

1. Open Virtual Box.
2. Click the name of the virtual machine that you want to make your CPU available to, then click the "Settings" button at the top of the window.
3. Click the "System" heading on the left side of the Settings window.
4. Click the "Processor" tab at the top of the window.
5. Drag the slider next to "Processor(s)" to the right until the value matches the number of processors or processor cores installed in your computer.
6. Drag the slider next to "Execution Cap" to the right until the value reads "100." This allows VirtualBox to use all of your processor's resources.
7. Click "OK," then double-click the virtual machine to turn it on.

As you can see I have 2 CPUs



What you will need to install minikube

- 2 CPUs or more
- 2GB of free memory
- 20GB of free disk space
- Internet connection
- Container or virtual machine manager, such as: Docker, Hyperkit, Hyper-V, KVM, Parallels, Podman, VirtualBox, or VMware

Let install the minkube

https://www.youtube.com/watch?v=hg1OCbVhuNw

https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/#install-kubectl-binary-with-curl-on-linux