**Kubernetes**

[Kubernetes](https://kubernetes.io/docs/concepts/overview/), often abbreviated as “K8s”, orchestrates [containerized](https://tanzu.vmware.com/containers) applications to run on a cluster of hosts. The K8s system automates the deployment and management of [cloud native](https://tanzu.vmware.com/cloud-native) applications using on-premises infrastructure or public cloud platforms. It distributes application workloads across a [Kubernetes cluster](https://www.vmware.com/topics/glossary/content/kubernetes-cluster.html" \t "/home/linuxdavi/Documents\\x/_blank) and automates dynamic container networking needs. Kubernetes also allocates storage and persistent volumes to running containers, provides automatic scaling, and works continuously to maintain the desired state of applications, providing resiliency.

**Features**

Kubernetes has many features that help orchestrate containers across multiple hosts, automate the management of K8s clusters, and maximize resource usage through better utilization of infrastructure. Important features include:

**\* Auto-scaling**. Automatically scale containerized applications and their resources up or down based on usage

**\* Lifecycle management**. Automate deployments and updates with the ability to:

\* Rollback to previous versions

\*Pause and continue a deployment

**\* Declarative model**. Declare the desired state, and K8s works in the background to maintain that state and recover from any failures

**\* Resilience and self-healing**. Auto placement, auto restart, auto replication and auto scaling provide application self-healing

**\* Persistent storage**. Ability to mount and add storage dynamically

**\* Load balancing**. Kubernetes supports a variety of internal and external load balancing options to address diverse needs

**\* DevSecOps support**. [DevSecOps](https://tanzu.vmware.com/devsecops" \t "/home/linuxdavi/Documents\\x/_blank) is an advanced approach to security that simplifies and automates container operations across clouds, integrates security throughout the container lifecycle, and enables teams to deliver secure, high-quality software more quickly. Combining DevSecOps practices and Kubernetes improves developer productivity.

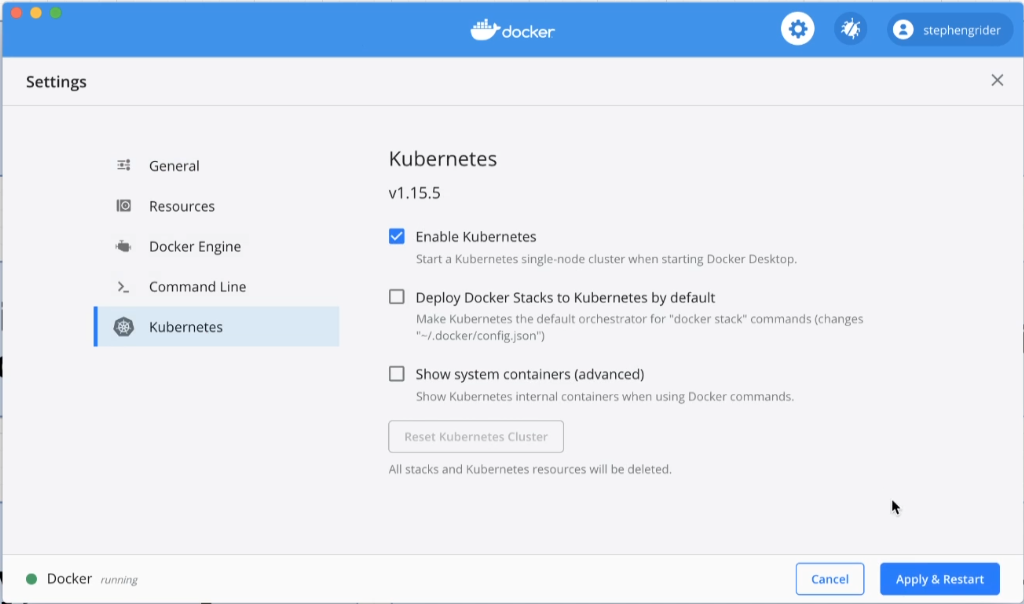
**Installing Kubernetes**

IMG_256 For windows

\* First install Docker Desktop Go to [Link](https://docs.docker.com/desktop/install/windows-install/)

\* Download and Start installing

\* Open Docker and Go to Settings -> Kubernetes -> check mark [ **Enable Kubernetes** ]

\* Click on **Apply & Restart**

IMG_256-- For Linux debian based like Ubuntu

\* To install kubernetes you need to install DOCKER First and minikube for kubernetes

\* To install Docker.io -> Follow Steps on **[link](https://www.simplilearn.com/tutorials/docker-tutorial/how-to-install-docker-on-ubuntu)**

\* Now Install **minikube** ( **x86-64, Stable, Debian Package**) **-->** if type differ GOTO **[link](https://minikube.sigs.k8s.io/docs/start/) or [link](https://computingforgeeks.com/how-to-install-minikube-on-ubuntu-debian-linux/)**

**\* curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube\_latest\_amd64.deb**

**\* sudo dpkg -i minikube\_latest\_amd64.deb**

Importance of minikube

\* Minikube helps to provide an environment with one master and workers node

**Accessing Kubernetes**

**Important Points**

**for Linux Debian Users (Details Click [Link](https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-22-04))**

\* minikube is running inside the docker container

Check it with --> **docker ps -a ,** You will see and container running minikube in it

**So**

1. Always login the docker in ubunutu

**--> docker login -u user\_name**

in return type **password**

1. Now check is you are loged-in **--> sudo cat ~/.docker/config.json**
2. If it contain **auth**:’**sfsafdfafsafsdfsdfsfs90iiklj18uijlskf**’ like this, you are ok to go
3. Now push images to docker hub --> **sudo docker push** user\_name/image\_name:tag
4. \*\*\*Pushing images to docker hub will help in future to provide images to **kubernetes cluster** or **minkube** to fetch images from hub in case not available in local machine \*\*\*

\* Kubernetes is also available inside minikube vm(VirtualMachine)

-> **sudo** **minikube kubectl version**

\* will return Client and Server Versions

\* To use Extenally kubernetes (kubectl)

--> GOTO [Link](https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/)

**What if we use kubernetes within minikube ?**

-> it will provide many facilities, GUI, and VMEnvironment for managing Kubernetes

-> minikube is detailed as "Local Kubernetes engine"

**What is the default driver minikube use ?**

-> docker-machine-driver-kvm2

**What if you want to use different drivers ? ( Click [Link](https://linuxhint.com/install-minikube-ubuntu/) )**

1. install the driver eg -> KVM2,VirtualBox,Docker etc
2. start minikube with driver --> **sudo** minikube start --version **driver\_name**
3. To it make default driver --> **minikube** config set driver **driver\_name**

**What if minikube already exists or stop in between while creating cluster ?**

\* Remove the .minikube from root directory it will remove minikube **binary** manually

--> **sudo** rm -r ~/.minikube

Now Start Cluster

Try with (Default) docker-machine-driver-kvm2

--> **sudo** minikube start

--> **sudo** minikube dashboard

**How to access kubectl via minikube ?**

--> **sudo** minikube kubectl -- version

**What if we not want to write (** minikube kubectl -- **) again and again ?**

--> Create **alias,** it will help you to provide variable for accessing features

--> **sudo** alias kubectl = **“**minikube kubectl --**”**

**OR**

Put directly into .bashrc located in root (~)

--> **sudo** nano ~/.bashrc

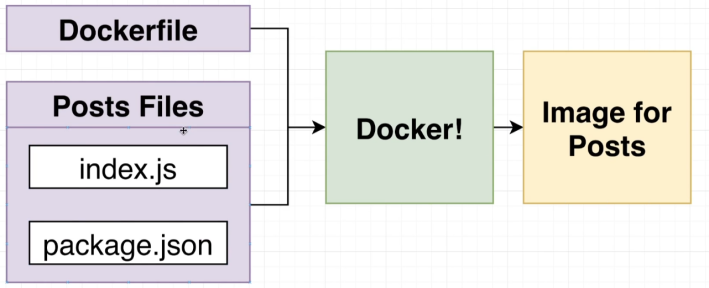
FIND line **add some ls aliases**

ADD line **alias k = ‘minikube kubectl --’**

**Understanding Kubernetes Terminology**

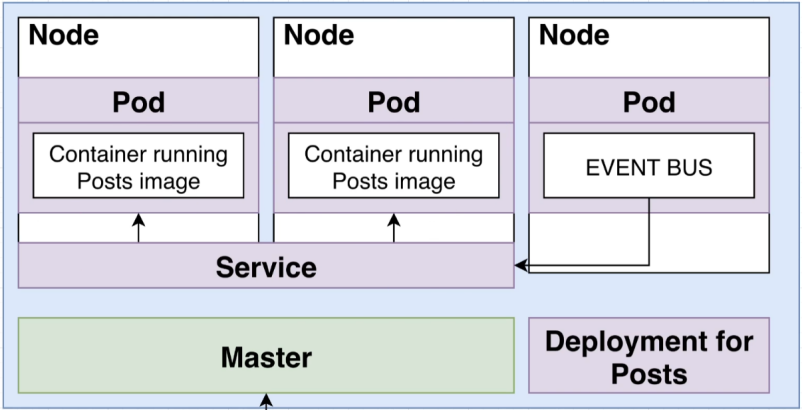
**What is an image ?**

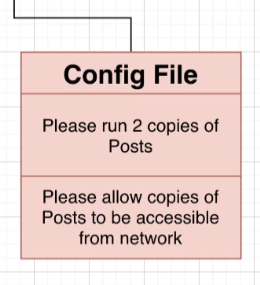
--> Images can be predefined or custom achitecture which can be just run by simple **run** command using docker

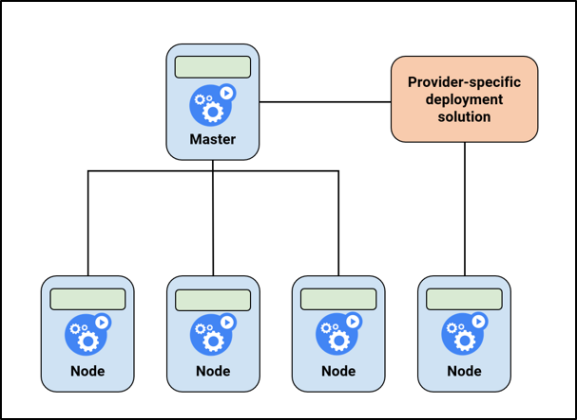


1. **What is the design of kubernetes to run these docker images ?**

**Kubernetes Cluster**







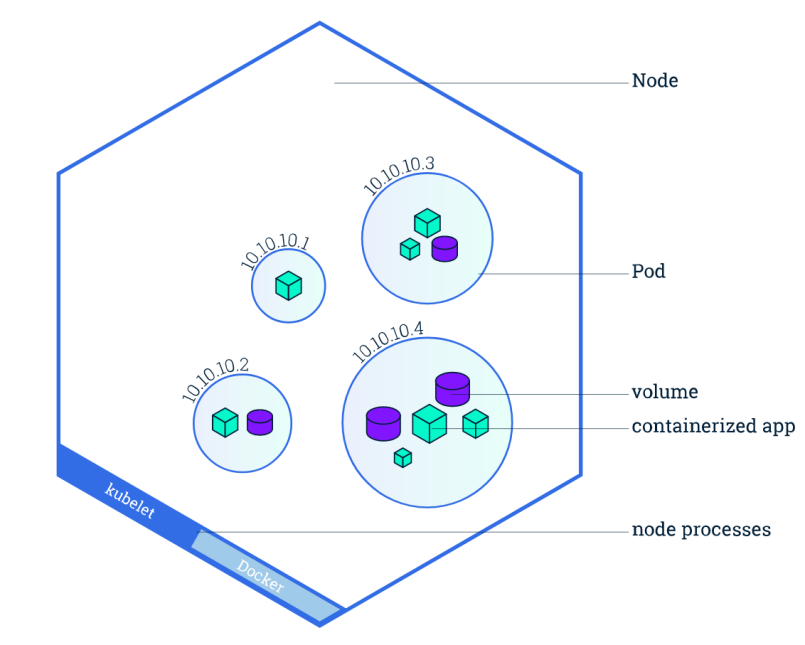
1. **Kubernetes Cluster, Node, Pod, Deployment, Service ?**

**Kubernetes Cluster**

A Kubernetes cluster is **a set of nodes that run containerized applications**. Containerizing applications packages an app with its dependences and some necessary services. They are more lightweight and flexible than virtual machines

Conclusion -> Each Cluster contains a **master** node and **worker** nodes By Default, can use Multiple **Master** inCase

Very Important-> **[Kubernetes Cluster vs Master Node](https://www.suse.com/c/kubernetes-cluster-vs-master-node/)**



**Node**

A Node is a **worker machine** in Kubernetes and may be either a virtual or a physical machine, depending on the cluster. Each Node is managed by the control plane. A Node can have multiple pods, and the Kubernetes control plane automatically handles scheduling the pods across the Nodes in the cluster

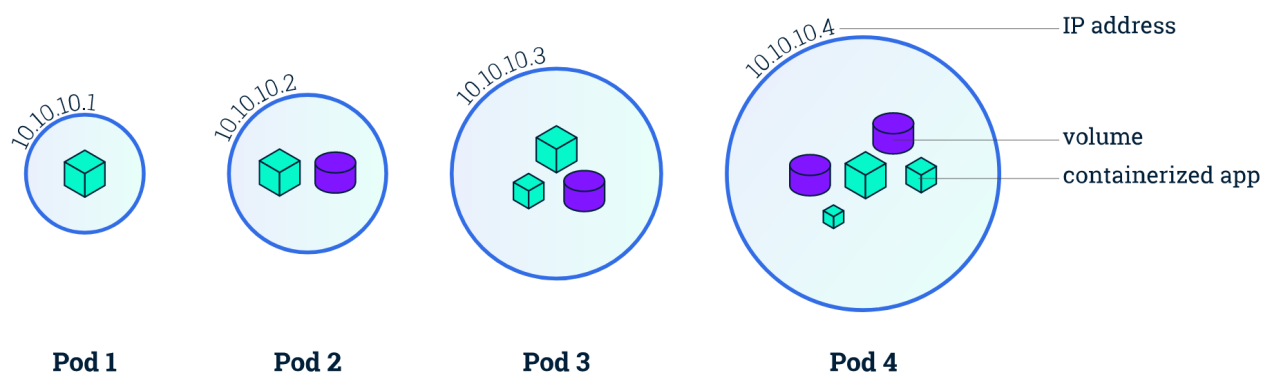
**Pod**

A Pod is a Kubernetes abstraction that represents a group of one or more **application containers** (such as Docker), and some shared resources for those containers. Those resources include:

\* Shared storage, as Volumes

\* Networking, as a unique cluster IP address

\* Information about how to run each container, such as the container image version or specific ports to use

A Pod models an application-specific "**logical host**" and can contain different application containers which are relatively tightly coupled. For example, **a Pod might include both the container with your Node.js app as well as a different container that feeds the data to be published by the Node.js webserver**. The containers in a Pod share an **IP Address** and **port space**, are always co-located and co-scheduled, and run in a shared context on the same Node.

Pods are the atomic unit on the Kubernetes platform. When we create a Deployment on Kubernetes, that Deployment creates Pods with containers inside them (as opposed to creating containers directly). Each Pod is tied to the Node where it is scheduled, and remains there until termination (according to restart policy) or deletion. In case of a Node failure, identical Pods are scheduled on other available Nodes in the cluster.

**Kubernetes Deployments**

A Kubernetes Deployment tells [Kubernetes](https://tanzu.vmware.com/kubernetes-vs-docker" \t "/home/linuxdavi/Documents\\x/_blank) how to create or modify instances of the pods that hold a [containerized application](https://tanzu.vmware.com/containers" \t "/home/linuxdavi/Documents\\x/_blank). Deployments can help to efficiently scale the number of replica pods, enable the rollout of updated code in a controlled manner, or roll back to an earlier deployment version if necessary. Kubernetes deployments are completed using kubectl, the command-line tool that can be installed on various platforms

**What are the benefits of using a Kubernetes Deployment?**

Kubernetes saves time and mitigates errors by automating the work and repetitive manual functions involved in deploying, scaling, and updating applications in production. Since the Kubernetes deployment controller continuously monitors the health of pods and nodes, it can make changes in real-time—like replacing a failed pod or bypassing down nodes—to ensure the continuity of critical applications.

Deployments automate the launching of pod instances and ensure they are running as defined across all the nodes in the Kubernetes cluster. More automation translates to faster deployments with fewer errors.

**What are Kubernetes Deployment Strategies?**

Will Discuss in Future (In Advance GOTO [link](https://www.vmware.com/topics/glossary/content/kubernetes-deployment.html))

**Service**

In Kubernetes, a Service is a method for exposing a network application that is running as one or more [Pods](https://kubernetes.io/docs/concepts/workloads/pods/" \o "" \t "/home/linuxdavi/Documents\\x/_blank) in your cluster.

A key aim of Services in Kubernetes is that you don't need to modify your existing application to use an unfamiliar service discovery mechanism. You can run code in Pods, whether this is a code designed for a cloud-native world, or an older app you've containerized. You use a Service to make that set of Pods available on the network so that clients can interact with it.

If you use a [Deployment](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/" \o "" \t "/home/linuxdavi/Documents\\x/_blank) to run your app, that Deployment can create and destroy Pods dynamically. From one moment to the next, you don't know how many of those Pods are working and healthy; you might not even know what those healthy Pods are named. Kubernetes [Pods](https://kubernetes.io/docs/concepts/workloads/pods/" \o "" \t "/home/linuxdavi/Documents\\x/_blank) are created and destroyed to match the desired state of your cluster. Pods are ephemeral resources (you should not expect that an individual Pod is reliable and durable).

Each Pod gets its own IP address (Kubernetes expects network plugins to ensure this). For a given Deployment in your cluster, the set of Pods running in one moment in time could be different from the set of Pods running that application a moment later.

This leads to a problem: if some set of Pods (call them "backends") provides functionality to other Pods (call them "frontends") inside your cluster, how do the frontends find out and keep track of which IP address to connect to, so that the frontend can use the backend part of the workload?

Enter Services

**Service In Kubernetes**

The **Service API**, part of Kubernetes, is an abstraction to help you **expose groups of Pods over a network**. Each Service object defines a logical set of **endpoints** (**usually these endpoints are Pods**) along with a policy about how to make those pods accessible.

In Future Will Descuse (For Advance GOTO [link1](https://www.vmware.com/topics/glossary/content/kubernetes-services.html) or [link2](https://kubernetes.io/docs/concepts/services-networking/service/))

End of Question

**\* How to create a pod?**

Ways to create pod

1. Pod can be created Directly by an Docker Image
2. Best Recomended way to create an pod is by creating YAML file

Recomended Way file Configration

create this file in **infra/k8s/**

**Simple YAML File (Indentation is Compelsory)**

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: Name\_of\_Pod

**spec**:

**containers**:

- **name**: Name\_of\_Container

**image**: Name\_of\_Docker\_Image

**Save This File --> podName.yaml or padName.yml**

**CMD For PODs and POD**

**Now Go to terminal to Create POD**

**--> sudo kubectl apply -f pod\_ConfigFile\_Name.yaml**

**\*Check Pod Status only**

**--> sudo kubectl get pods**

**\*\*Check Pod Status and details like ip Address ,wrapper Node etc**

**--> sudo kubectl get pods -o wide**

**\*\*\*Check Single Pod Description with Full Details**

**--> sudo kubectl describe pod pod\_Name**

**\*\*\*\*Execute in Pod**

**--> sudo kubectl exec -it [pod\_Name] [CMD]**

**\*\*\*\*\*Delete Pod**

**--> sudo kubectl delete pod [pod\_Name]**

**\*\*\*\*\*Delete Pod**

**--> sudo kubectl delete -f pod [pod\_ConfigFile\_Name.yaml]**

**CMD For NODES and NODE**

**\*Check Node Status only**

**--> sudo kubectl get nodes**

**\*\*Check Pod Status and details like ip Address ,wrapper Node etc**

**--> sudo kubectl get nodes -o wide**

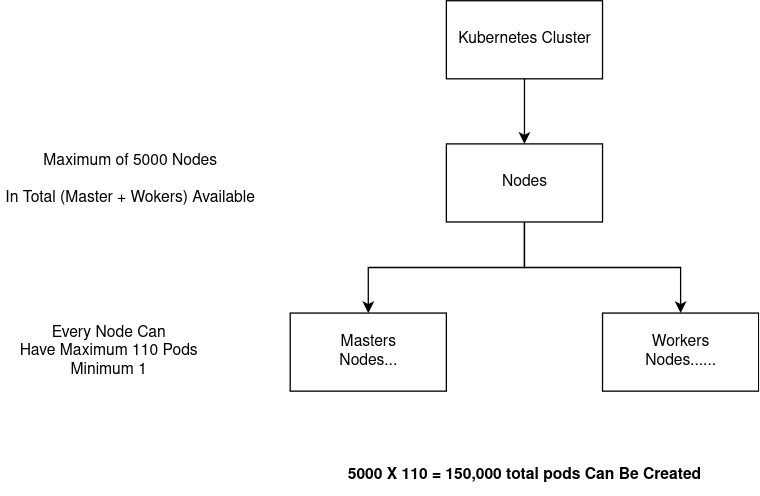
**\*\*\*Check Single Pod Description with Full Details**

**--> sudo kubectl describe node node\_Name**

**What is the limit number of pods in Kubernetes?**

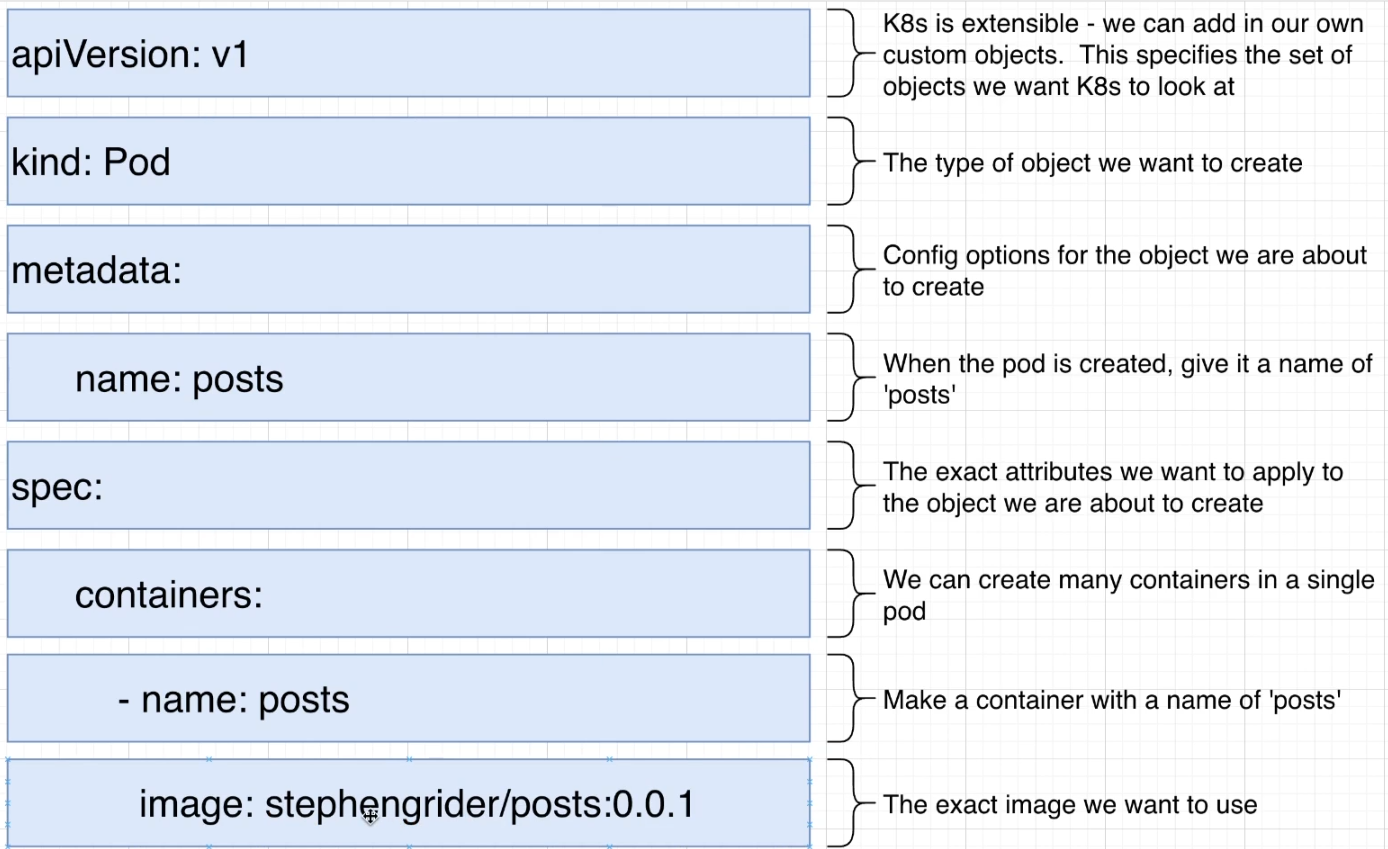
More specifically, Kubernetes is designed to accommodate configurations that meet all of the following criteria: **No more than 110 pods per node**. **No more than 5,000 nodes**. **No more than 150,000 total pods**.

**Conclusion**

****

**EXAMPLE OF COMPLEX POD** **YAML FILE**

**Detailed Description**

****

**Q. How to Create Deployment ? (For more details Goto Page-6 Kubernetes Deployments)**

**Remember Deployment act as an manager of pods in Node**

**For Ex -: There is 1 master node and 1 worker node (A user can create as much 5000 nodes in total till date)**

**in worker node :- we created 1 deployment of 2 pods and 1 more deployement of 3 pods**

**So in total a worker node contains 5 pods in total and there are 2 managers**

**Create An File in infra/k8s/DeploymentFile.yaml**

**DeployementFile.yaml**

**apiVersion: apps/v1**

**Remember :-**

selector will help deployment to recognize the responsibility pods by its labels

**matchLabels**

of Deployement (Selector)

must be same as **labels** of

pods (template)

**kind: Deployment**

**metadata:**

**name: Deployment\_Name**

**spec:**

**replicas: No\_Of\_Replicas**

**selector:**

**matchLabels:**

**app: label\_Need\_To\_Match\_Pod\_Label**

**template:**

**metadata:**

**labels:**

**app: label\_of\_Pod**

**spec:**

**containers:**

**- name: Pod\_Name**

**image: User\_Name/Image\_Name:Version**

**Simple Deployement Yaml File**

**Example**

**apiVersion:** apps/v1

**kind:** Deployment

**metadata:**

**name:** posts-depl

**spec:**

**replicas:** 1

**selector:**

Used by **Deployment**

For detecting pods based on **matchlabels**

Used For **Pod** Details

**matchLabels:**

**app:** posts

**template:**

**metadata:**

**labels:**

**app:** posts

**spec:**

**containers:**

**- name:** posts

**image:** davimehra/posts:0.0.1

**CMD For Deployments and Deployement**

**Now Go to terminal to Create Deployement**

**--> sudo kubectl apply -f deployement\_ConfigFile\_Name.yaml**

**\*Check Deployement Status only**

**--> sudo kubectl get deployements**

**\*\*Check Deployement Status and details like ip Address ,wrapper Node etc**

**--> sudo kubectl get deployements -o wide**

**\*\*\*Check Single Deployement Description with Full Details**

**--> sudo kubectl describe pod deployment\_Name**

**\*\*\*\*Execute in Deployement**

**--> sudo kubectl exec -it [pod\_Name] [CMD]**

**\*\*\*\*\*Delete Deployement**

**--> sudo kubectl delete deployement [deployment\_Name]**

**\*\*\*\*\*Delete Deployement**

**--> sudo kubectl delete -f deployement [deployment\_ConfigFile\_Name.yaml]**

**Details Example For Future Deployement Yaml File (Advance)**

**UPDATING DEPLOYEMENT**

**Method 1 (UnProffesional Way)**

Repeate Every Step from building image to ceating Deployement

1. Make Changes to server file
2. **Build Image --> sudo docker build -t username/imageName:newVersion .**
3. Change ImageVersion to newVersion in DeployementYaml file and Save
4. **Create Deployment -->** **sudo kubectl apply -f UpdatedDeploymentFile.yaml**
5. Deployement will be configured rather than created

**Method 2 (Proffesional Way) Recomended**

In This image version will always be latest and Deployment only need an restart

1. Make Changes to Server File
2. **Build Image --> sudo docker build -t username/imageName:latest .**
3. Note :- if Image Version is Null Docker will treate it as latest automatically
4. Check in DeployementYaml File ImageVersion must be Empty or latest
5. Note :- if Image Version is Empty, Kubernetes will treate it as latest automatically
6. **Restart Deployment -->** sudo **kubectl rollout restart deployment deplName.yaml**

**SERVICE**

**INTRODUCTION TO SERVICE**

**Lyaman Terms ->**Service in Kubernetes helps pods to expose to the outer network with the help of ports

**Defination**

A Kubernetes service is a logical abstraction for a deployed group of pods in a cluster (which all perform the same function).

Since pods are ephemeral, a service enables a group of pods, which provide specific functions (web services, image processing, etc.) to be assigned a name and unique IP address (clusterIP). As long as the service is running that IP address, it will not change. Services also define policies for their access.

**what is the difference between a service and a deployment?**

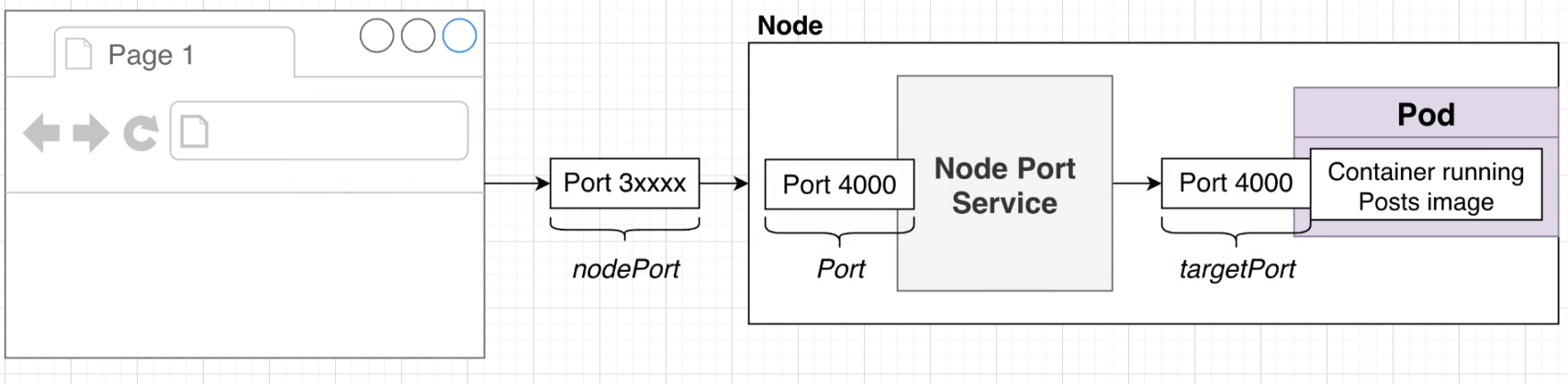
In Kubernetes a deployment is a method of launching a pod with [containerized applications](https://tanzu.vmware.com/containers" \t "/home/linuxdavi/Documents\\x/_blank) and ensuring that the necessary number of replicas is always running on the cluster.

On the other hand, a service is responsible for exposing an interface to those pods, which enables network access from either within the cluster or between external processes and the service.

**What are the types of Kubernetes services?**

* **ClusterIP.** Exposes a service which is only accessible from within the cluster.
* **NodePort.** Exposes a service via a static port on each node’s IP.
* **LoadBalancer.** Exposes the service via the cloud provider’s load balancer.
* **ExternalName.** Maps a service to a predefined externalName field by returning a value for the CNAME record.

**NodePort**



**NodePort** Help to expose an single pod **Port** to outer World,

this **Service Object** will act as an port forwarder between **Port of NodePort** and the **TargetPort of Pod**

In End Kubernetes will assign Port No. Starting from **3xxxxx** to the NodePort

This **3xxxx** Port No. can be accesed with **Localhost (Window or Mac)** or **minikube ip (Linux User)**

**Eg for Windows or Mac User-: <http://localhost:3xxxx>/PortName**

**Eg for Linux Users -->** run command for minikube ip **--> sudo minikube ip**

**Go-> <http://minikube_ip:3xxxx>/PortName**

**Remember:-** It is very dynamic in nature so try not to use NodePort as exposure to world,Because Every time an pod will **re-created** a new NodePort will be assigned, so all the promgrams dependent on the Prev PORT Number will FAIL

Simple Syntax for YAML File

Create an Seprate or Let it be in the Same YAML File of Deployment with (---) for New Line

**Demo Template**

**Example For Posts Server Service**

**apiVersion:** v1

**kind:** Service

**metadata:**

**name:** posts-srv

**spec:**

**type:** NodePort

**selector:**

**app:** posts

**ports:**

**- name:** posts

**port:** 4000

**protocol:** TCP

**targetPort:** 4000

**apiVersion:** v1

**kind:** Service

**metadata:**

**name:** ServiceName

**spec:**

**type:** NodePort

**selector:**

**app:** Name\_Of\_Pod\_To\_Apply

**ports:**

**- name:** PortName

**port:** 4000

**protocol:** TCP

**targetPort:** 4000

**ClusterIP**

This is the default type for service in Kubernetes.

As indicated by its name, this is just an address that can be used inside the cluster.

**Layman Term ->** **ClusterIP** is an Service Object which can be attached with pods so that other pods can communicate with each other by there own ClusterIP

**Q. Why ClusterIP , Why not NodePort ?**

**NodePort Service** is used when OuterWorld need to interact with the pods those ip address can be Your Virtual Machine Public Ip , Minikube Ip , localhos But Port No. will be generated randomly starting from 3xxxx.

**ClusterIP Service** is used when pods need to interact with each other because of these are called internal ip.

Full Example = **Deployment** + **ClusterIp Service**

apiVersion: apps/v1

kind: Deployment

metadata:

name: posts-depl

spec:

replicas: 1

selector:

matchLabels:

app: posts

template:

metadata:

labels:

app: posts

spec:

containers:

- name: posts

image: davimehra/posts

---

apiVersion: v1

kind: Service

metadata:

name: posts-clusterip-srv

spec:

selector:

app: posts

type: ClusterIP

ports:

- name: posts

port: 4000

targetPort: 4000

protocol: TCP

**Sample** Yaml Code

**apiVersion:** v1

**kind:** Service

**metadata:**

name: service\_name

**spec:**

selector:

app: name\_of\_pod

ports:

- name: port

protocol: TCP

port: port\_for\_Cluster\_Service

targetPort: port\_of\_Pod\_Listening\_to

**ClusterIp Service** Posts POD

**apiVersion:** v1

**kind:** Service

**metadata:**

name: posts-clusterip-srv

**spec:**

selector:

app: posts

type: ClusterIP

ports:

- name: posts

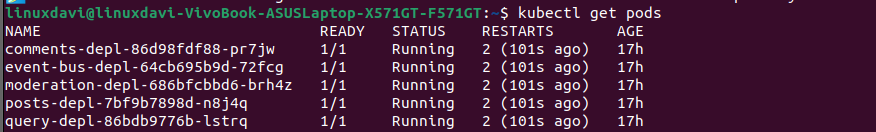
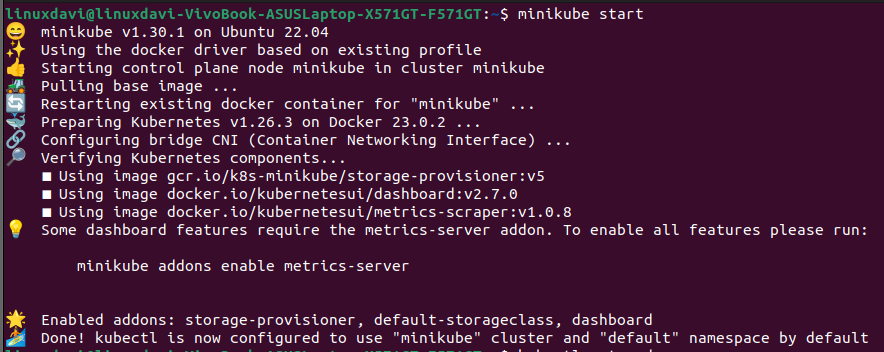
port: 4000

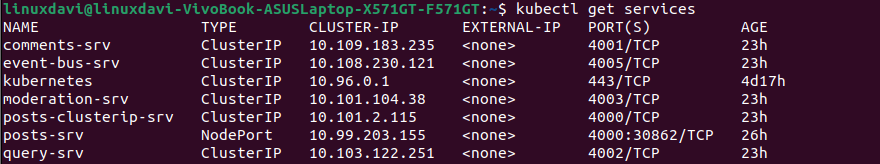
targetPort: 4000

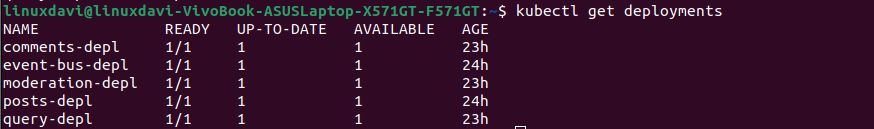
protocol: TCP

**What if we are in Project and want ClusterIP Service and communicate between pods ?**

1. We will take an example of Blog Project , you can see in this directory
2. First Create Yaml Files for Deployement and ClusterIp Service both, each for every Server POD
3. GO To blog/infra/k8s/ you will see (these file contains both Deployement + ClusterIP Service seperated by ---)
4. posts-depl**.yaml**, comments-depl**.yaml**, moderation-depl**.yaml**, query-depl**.yaml**, event-bus-depl**.yaml**
5. Now in this dir , apply with file (-f) in kubectl **--> sudo kubectl apply -f** name\_of\_file**.yaml**
6. Either do each file one by one or just do all at once **--> sudo kubectl apply -f .** (dot context for cwd)
7. Now For Outer World Request to POSTS pod we need 1 NODEPORT
8. Create File posts-srv**.yaml** for NODEPORT Service
9. Now Check for Running Services and Pods Created though Deployement
10. Run --> sudo minikube start --> sudo kubectl get pods --> sudo kubectl get services --> sudo kubectl get deployements







1. Now see precisely the service names in services
2. Now if some request require event-bus server response that request need to come to ClusterIP Service of **event-bus** which is **event-bus-srv ,** Now Change Every request url from **localhost** to **event-bus-srv**

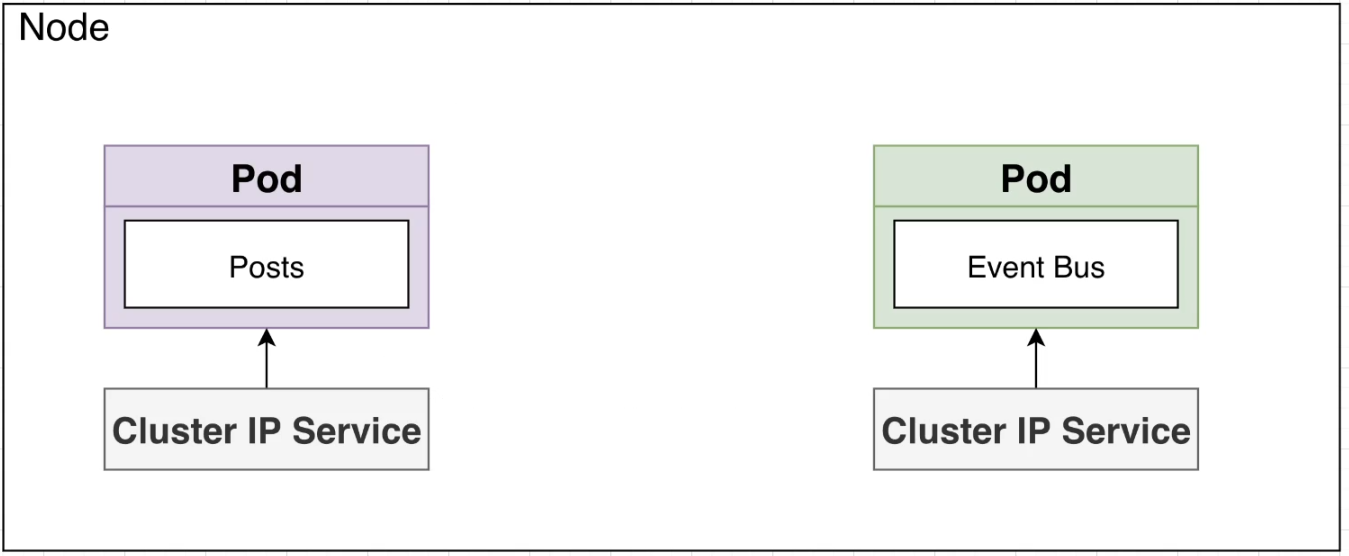
**Example ->** Go to Directory **blogs/posts/index.js**

All Request need to be updated from **localhost** to **event-bus-srv**

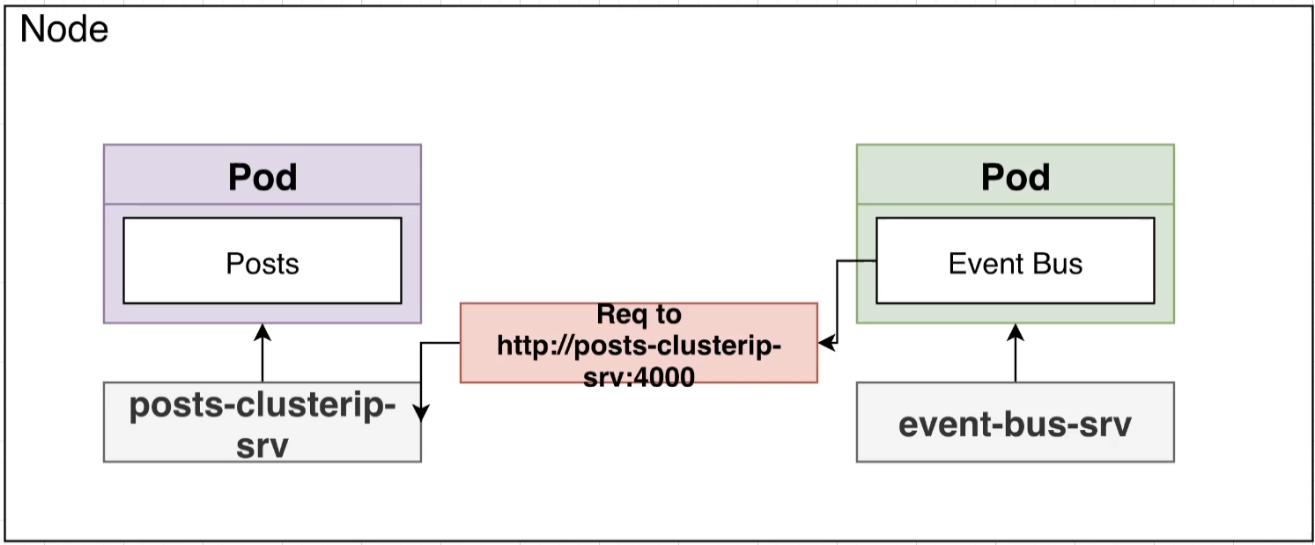
blogs/**event-bus**/index.js file also need to be **updated**

1. localhost:4000 -> posts-srv:4000
2. localhost:4001 -> comments-srv:4001
3. localhost:4002 -> query-srv:4002
4. localhost:4003 -> moderation:4003
5. Now Repeat this step for every **index.js file** in
6. blogs/**moderation**/index.js
7. blogs/**query**/index.js
8. blogs/**comments**/index.js
9. Now build every image again one by one
10. Push Every Image Again to Docker Hub One by one
11. Goto infra/k8s and rollout restart all deployements and services again
12. Do it one by one with **--> sudo kubectl rollout restart deployement deployoment\_Name**
13. Rollout Restart help deployment to restart deployement by pulling new image of pod from docker hub
14. Now Check with do and post Request at <http://minikube_ip:NodePortNo_posts/posts>
15. Request Data = {“title”:”POST1”} **-->** Now check logs in pods with **--> sudo kubectl logs** pods\_Name

Diagram Example of Posts ClusterIP communication with event-bus ClusterIP



Standard Diagram



Posts Internal

Request to

event-bus

and **Vise Versa**

Introduction to **Load Balancer** and **Ingress-Controller**

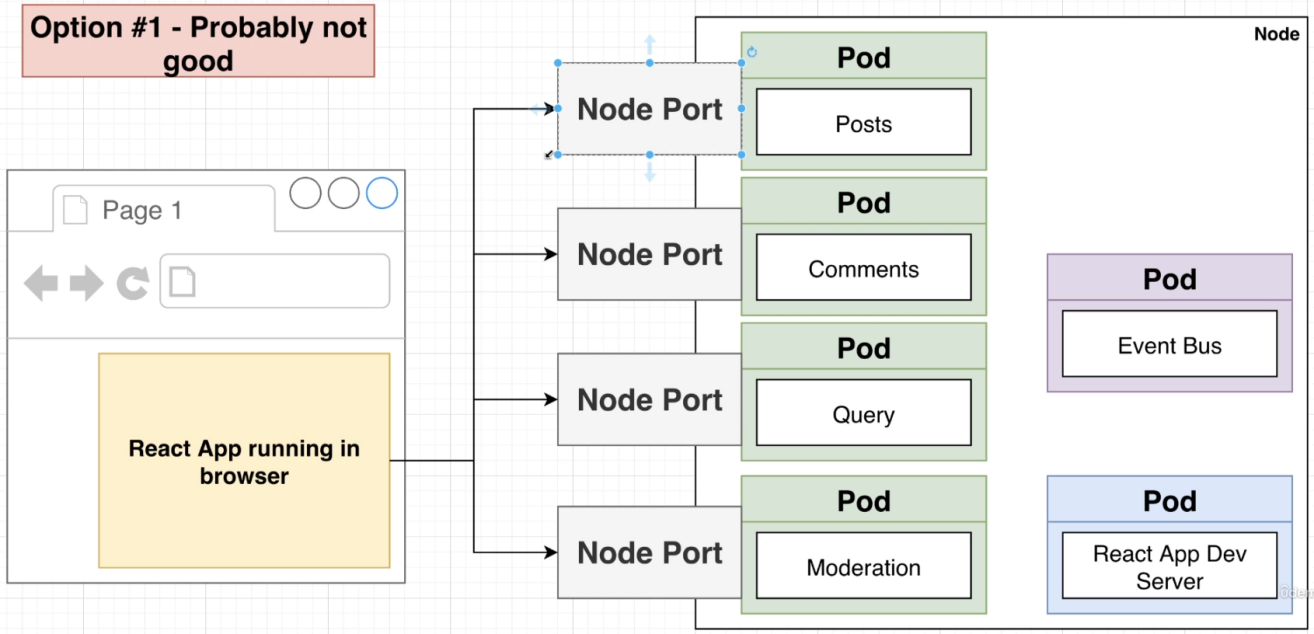
**What is the Problem ,why we need load balancer and ingess Controller ?**

\* Imagine we need to communicate with one of the POD service and then other pod service and then another, like wise many

There are two methods to deal with this

1. Either we can create a nodePort Service for every POD

So that other world can use these PODS (But it is considered as **BAD PRACTICE**)



NOT RECOMENDED

1. Efficient Way is to use **LOAD BALANCER** and **INGESS-CONTROLLER**

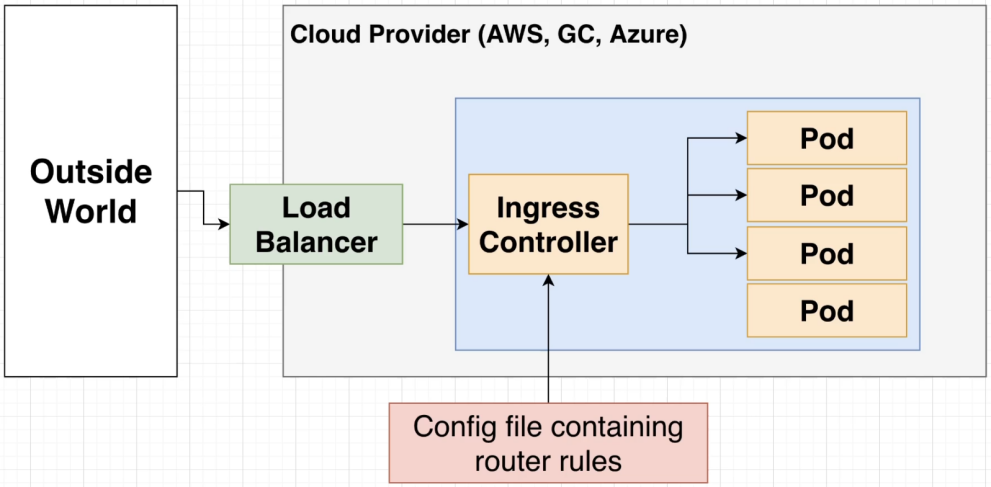
**Load Balancer Service**

It tells Kubernetes to reach out to its provider(Gcloud, Aws etc) and provision a load balancer.

This Load Balancer Service will tranfer all trafic to an single POD which is located in the cluster and that POD is called Ingress or Ingress Controller

**Ingress or Ingress Controller**

It is an pod with a set of routing rules to distribute traffic to other services (ClusterIP Service)

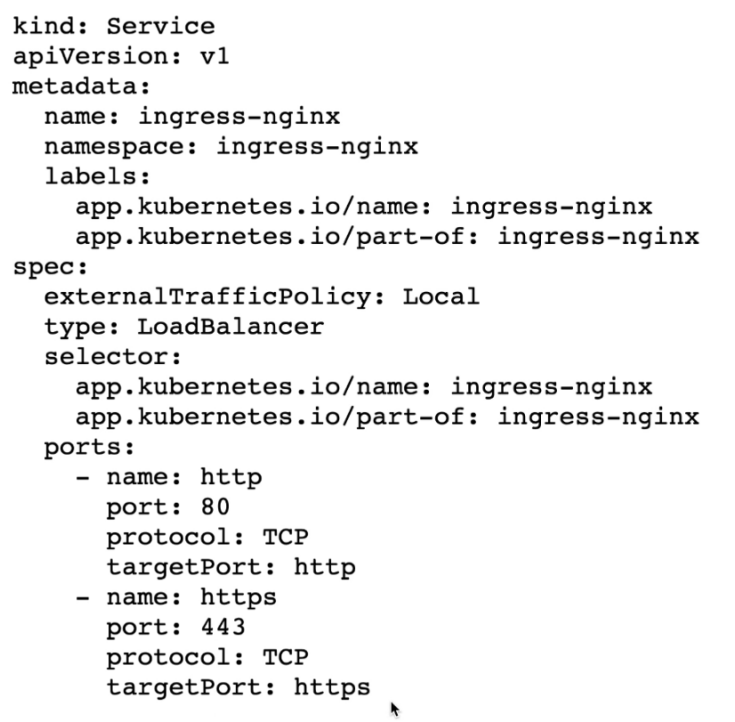


**How to implement the load Balancer and Ingress Controller into our Project ?**

1. Search for **ingress-nginx** or goto [link](https://kubernetes.github.io/ingress-nginx/deploy/) ( If readed from link you can skip following steps )
2. Now apply **ingress-controller pod** deployment with

**--> kubectl apply -f** <https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.7.0/deploy/static/provider/cloud/deploy.yaml>

1. Check if is Pod Working with Fligth Check **-->sudo kubectl get pods** --namespace=ingress-nginx
2. Now enable **LoadBalancer** **Service** By
3. For minikube **---> minikube addons enable ingress (don’t use sudo)**

****

For Knowledge see LoadBalancer **Yaml File**

Remember this is not for Entering in Code

this is just to know how

**minikube addons enable ingress**

Works

1. Now create the **Routing Rules ,** to create routing rules we will create an Ingress type Yaml File

Let File name be **ingress-srv.yaml --> COMPLETE EXAMPLE for just posts routing**

Old Version kubernetes version < 1.19x

**apiVersion:** networking.k8s.io/v1beta1

**kind:** Ingress

**metadata:**

**name:** ingress-srv

**annotations:**

**kubernetes.io/ingress.class:** nginx

**spec:**

**rules:**

**- host:** posts.com

**http:**

**paths:**

- **path:** /posts

**backend:**

**serviceName:** posts-clusterip-srv

**servicePort:** 4000

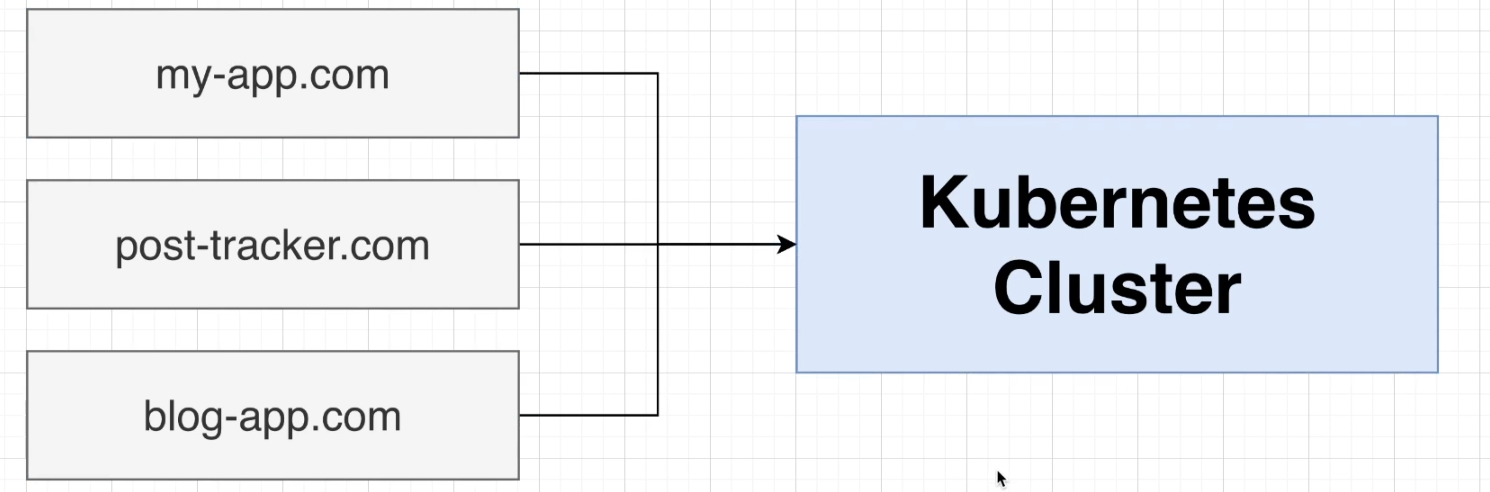
Nginx is configured to automatically discover all ingress (**Routing Rules**) with the

**kubernetes.io/ingress.class:** "nginx" annotation or

where **ingressClassName:** nginx is present. Please note that the ingress resource should be placed inside the same namespace of the backend resource.

**Routing Rules :-** These are simple rules for defining routes and directing the route path to specified backend service

For Version > 1.19x Kubernetes



**apiVersion:** networking.k8s.io/v1

**kind:** Ingress

**metadata:**

name: ingress-srv

**spec:**

**ingressClassName:** nginx

**rules:**

- **host:** posts.com

**http:**

**paths:**

- **path:** /posts

**pathType:** Prefix

**backend:**

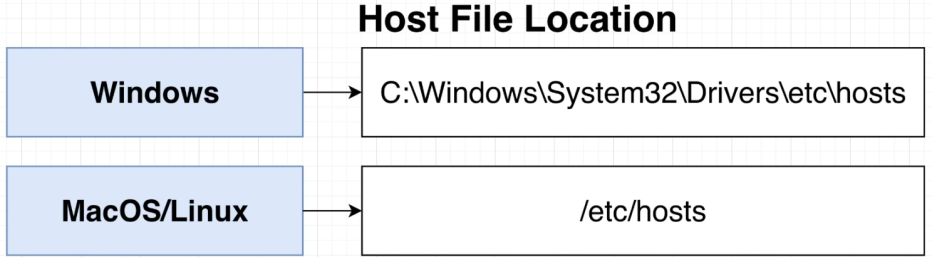
**service:**

**name:** posts-clusterip-srv

**port:**

**number:** 4000

1. This step is just for PROJECT DEVELOPMENT CYCLE -> in development Environment we use **localhost** as domain but we need some thing by which browser get tricked and feels as to be called by **posts.com,**  but just for development environment in production we will be calling these LoadBalancer from OuterWorld or from real [www.posts.com](http://www.posts.com)
2. Now see the host file Location



Open the **hosts** File and go in the bottom and just add line

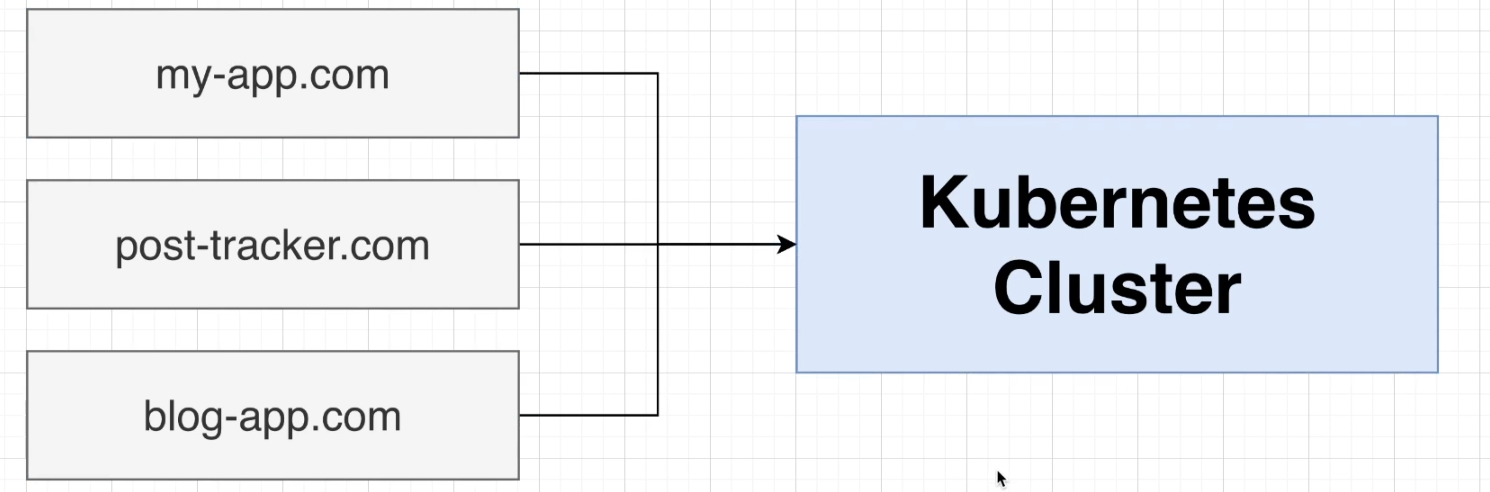
if on MacOs

**127.0.1.1** posts.com

if on minikube (Find minikube ip )

**minikube\_ip** posts.com

1. Save the file
2. go to http://posts.com/posts



**Important Note**

Kubernetes can hold

and can work with

different hosts at a

same time

**Limitations** of Ingress Controller

1. **Ingress Controller cannot diferenciate between request methods like (post,get,delete,patch,put etc).**

Problem For Eg :-

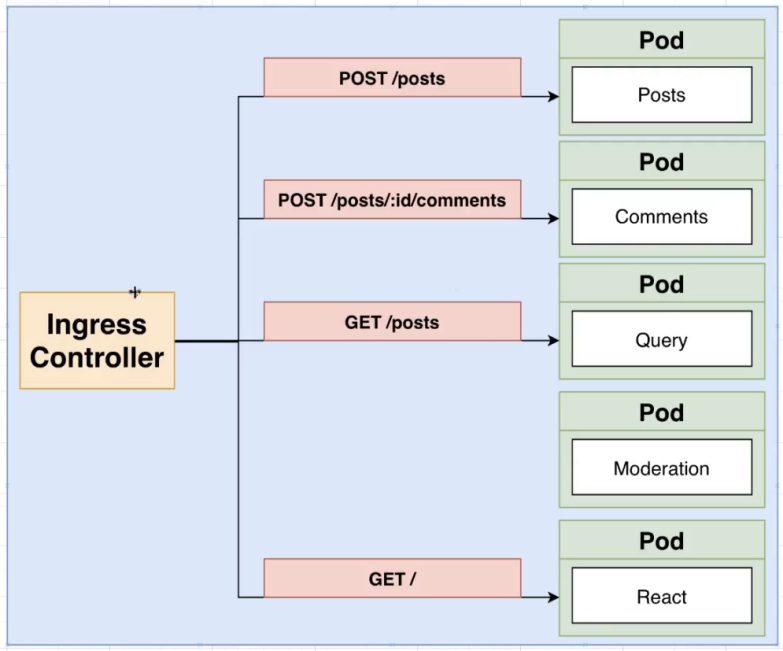
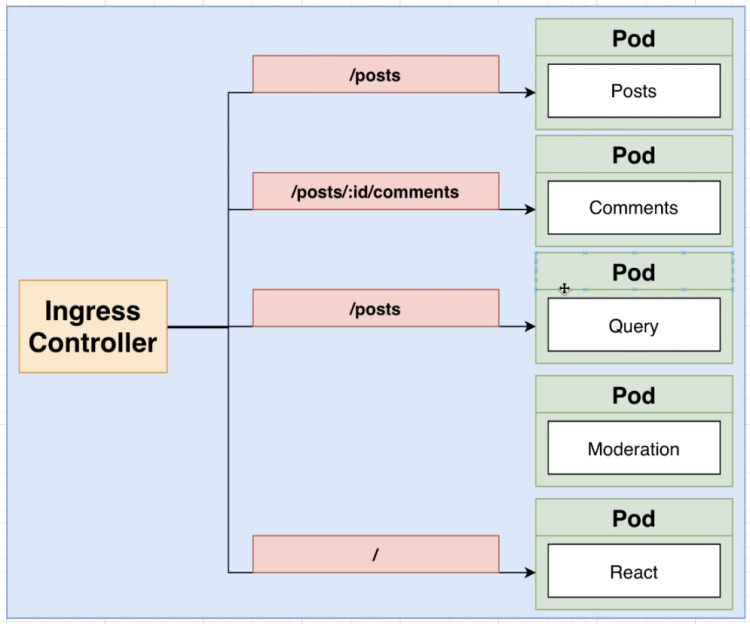
**method** **path**

delete /post

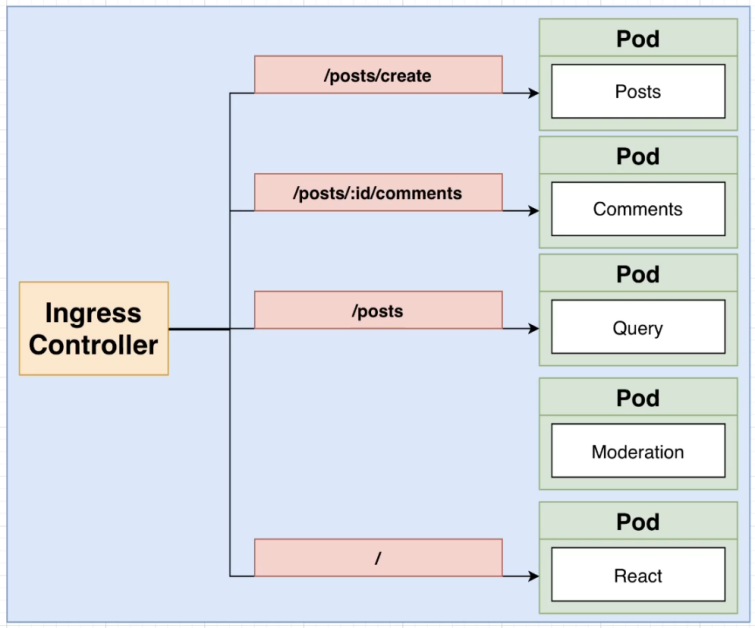
get /post

post /post

Here these both will cause conflict between path routing in ingress-controller routing rules.



**Solution :- Try to use diferent routes path**



**method** **path**

delete /post/delete

get /post/

post /post/create

1. **Ingress Controller cannot handle wildcard (:id) in Routes**

Problem --> **:id** --> here id will not be considered as wild card

Eg --> [http://www.getit.com/posts](http://www.getit.com/post/:id/comment)**[/:id](http://www.getit.com/post/:id/comment)**[/comment](http://www.getit.com/post/:id/comment)

Solution :- Use regex expression

regex Usage Tutorial

1. Define regex in annotations of **routing rules** and set it to **‘true’** (Remember it must be in string not boolean)
2. **annotations:**

**nginx.ingress.kubernetes.io/use-regex:** ‘true’

1. Now to use Regex start with ?( and end with )
2. Eg--> - path: /posts/?(.\*)/comments

**Topic : Working With Client App**

Client app Need to be in pod inside the cluster environment so that it can access all other services with other

pods ClusterIP

**update** client files with req localhost --> posts.com (ingnx regress routed for posts)

**create** image of client

**push** image to docker hub

**create** pod with deployment

**Topic: Routing Rules for ingress controller**

Create File ingress-srv**.yaml** (infra/k8s/ingress-srv.yaml)

Update File with

**Example**

**Explanation:** In this example all the request coming from outer world with domain (posts.com) will be redirected to services (**ClusterIP**) with respect to the routing path after --> **posts.com**/

Remember : **Backend Services** **First** and **Client Services** always at **last**

**Apply with -> kubectl** apply -f ingress-srv.yaml

**apiVersion:** networking.k8s.io/v1

**kind:** Ingress

**metadata:**

**name:** ingress-srv

**annotations:**

**nginx.ingress.kubernetes.io/use-regex:** "true"

**spec:**

ingressClassName: nginx

**rules:**

**- host:** posts.com

**http:**

**paths:**

**- path:** /posts/create

**pathType:** Prefix

**backend:**

**service:**

**name:** posts-clusterip-srv

**port:**

**number:** 4000

**- path:** /posts

**pathType:** Prefix

**backend:**

**service:**

**name:** query-srv

**port:**

**number:** 4002

**- path:** /posts/?(.\*)/comments

**pathType:** Prefix

**backend:**

**service:**

**name:** comments-srv

**port:**

**number:** 4001

**- path:** /?(.\*)

**pathType:** Prefix

**backend:**

**service:**

**name:** client-srv

**port:**

**number:** 3000

Here: if request came with **url**

http**://**posts.com**/** posts **/** create

This request of url will be d

**Directed To ->** post-srv (ClusterIP Service)

**ClusterIP Port ->** 4000

Here: if request came with **url**

http**://**posts.com**/**posts

This request of url will be

**Directed To ->** query-srv (ClusterIP Service)

**ClusterIP Port ->** 4002

Here: if request came with **url**

http**://**posts.com**/**posts**/** ?(.\*) **/**comments

This request of url will be

**Directed To**

comment-srv (ClusterIP Service)

**ClusterIP Port ->** 4001

Here: if request came with **url**

http**://**posts.com**/**posts**/** ?(.\*)

This request of url will be

**Directed To ->** client-srv (ClusterIP Service)

**ClusterIP Port ->** 3000

**Skaffold**

**What is Skaffold ?**

Skaffold is **a command line tool that facilitates continuous development for container based & Kubernetes applications**. Skaffold handles the workflow for building, pushing, and deploying your application, and provides building blocks for creating CI/CD pipelines

**Features**

* Automates many tasks in a Kubernetes dev environment
* Makes it really easy to update code in running pod
* Makes it really easy to create/delete all objects tied to a project at once
* **skoffold.dev**

**Installation [Link](https://skaffold.dev/docs/install/" \l "standalone-binary)**

**For linux x64-86 based ->**

**curl** -Lo skaffold https://storage.googleapis.com/skaffold/releases/latest/skaffold-linux-amd64

**sudo** install skaffold /usr/local/bin/

**Remember : Skaffold is an tool which runs outside the cluster , Steps to create skaffold.yaml File**

1. Go to parent folder
2. Eg -> blog is the parent folder
3. create file name any with extension .yaml
4. Now define apiVersion, kind, deploy, build
5. **apiVersion:** skaffold/v2alpha3
6. **kind:** Config
7. **deploy:**

**kubectl:**  (it is use to define type of cli used for the following files)

**manifests:**

- ./infra/k8s/\* (it will help skaffold to detect the kubectl executable yaml files)

1. **build:**

**local:** (it is used to define storage, and images to Docker Hub)

**push:** false (by Default it is true)

**artifacts:** (it will define the images context)

**- image:** davimehra/client (Name of image need to be Auto Managed)

**context:** client (folder need to watch for update,if update -> rebuild entire image)

**docker:**

**dockerfile:** Dockerfile (Define Docker File Location in context)

**sync:**  (Update the image in pod if the following src change)

**manual:**

**- src:** ‘ ./ src / \*\*/ \*.js ’ (Folder need to be watch manualy for change)

**dest:** . (destination path in the container where the files should be synced to.)

**- image:** davimehra/posts

**context:** posts

**docker:**

**dockerfile:** Dockerfile

**sync:**

**manual:**

**- src:** ‘ \*.js ’

**dest:** .

Can go to **[Link](https://skaffold.dev/docs/references/yaml/)**

**Full Example** in File --> Skaffold.yaml --> **sudo** skaffold **dev**

**apiVersion:** skaffold/v2alpha3

**kind:** Config

**deploy:**

**kubectl:**

**manifests:**

- ./infra/k8s/\*

**build:**

**local:**

**push:** false

**artifacts:**

**- image:** davimehra/client

**context:** client

**docker:**

**dockerfile:** Dockerfile

**sync:**

**manual:**

**- src:** 'src/\*\*/\*.js'

**dest:** .

**- image:** davimehra/comments

**context:** comments

**docker:**

**dockerfile:** Dockerfile

**sync:**

**manual:**

**- src:** '\*.js'

**dest:** .

**- image:** davimehra/moderation

**context:** moderation

**docker**:

**dockerfile:** Dockerfile

**sync:**

**manual:**

**- src:** '\*.js'

**dest:** .

**- image:** davimehra/posts

**context:** posts

**docker:**

**dockerfile:** Dockerfile

**sync:**

**manual:**

**- src:** '\*.js'

**dest:** .

**- image:** davimehra/event-bus

**context:** event-bus

**docker:**

**dockerfile:** Dockerfile

**sync:**

**manual:**

**- src:** '\*.js'

**dest:** .

**- image:** davimehra/query

**context:** query

**docker:**

**dockerfile:** Dockerfile

**sync:**

**manual:**

**- src:** '\*.js'

dest: .