KUBERNETES

Why do we need Kubernetes?

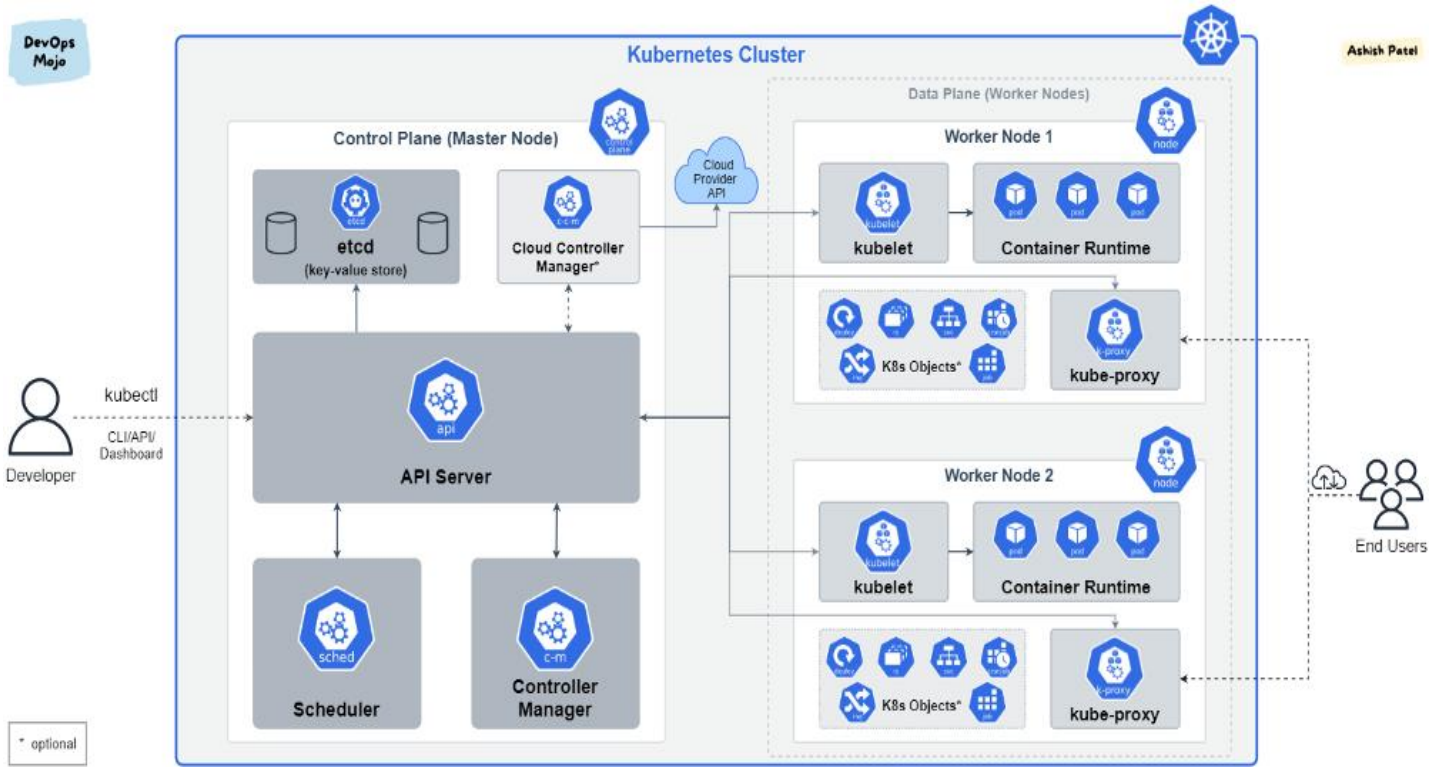
We are now in a container deployment era. Having the same set of configurations for an application in production might not be cost effective.
For Instance,
If we consider a shopping application.
A **BLACK FRIDAY** is the day where the traffic will be more than usual, maybe 10x or 20x than the usual; having the same servers with a limited capacity that can handle users on a usual day might break.
Hence, we will need to deploy replicas or we will need to spread the nodes across wrt geography so that the application can handle the traffic and also can make a recognizable profit.
Similarly,
A **JULY 4th** is the day where the traffic will be very less, maybe almost close to nil. Having the same set of servers of a usual day or a black Friday might not be cost effective as the traffic expected is at the least. So, intelligence is to dial down the servers and configurations as we now have “PAY AS YOU GO” platforms.

This sort of orchestrations can be done easily with Kubernetes. That’s the “why” for us.

ADVANTAGES OF K8s:

Service discovery and load balancing Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.
Storage orchestration Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
Automated rollouts and rollbacks You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.
Automatic bin packing You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks. You tell Kubernetes how much CPU and memory (RAM) each container needs. Kubernetes can fit containers onto your nodes to make the best use of your resources.
Self-healing Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.
Secret and configuration management Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

KUBERNETES COMPONENTS:



CONTROL PLACE (MASTER NODE)

API SERVER(kube-apiserver):

- This is the front end of the Kubernetes control plane (REST/Kubectl)
- Exposes the Kubernetes API.
- It tracks the state of all the cluster components and manage the interaction between them.
- Designed to scale horizontally (add more worker nodes – addition of physical/virtual worker nodes)
- It consumes YAML/JSON manifest files.
- Validates and processes the requests made by the APIs.

ETCD(key-value store)

- Consistent, distributed and highly available key-value store.
- Stateful, persistent storage that stores all of Kubernetes cluster data (cluster state and config)
- Source of truth for the cluster.
- Can be configured as a part of control place or configured externally.
- If we have multiple masters and worker nodes in a cluster, ETCD is responsible for implementing locks to ensure there are no conflicts between masters.

SCHEDULERS(kube-scheduler)

- Schedules pods to worker nodes.
- It watches api-server for newly created pods/containers with no assigned node and selects a healthy node for them to run on.
- If there are no suitable nodes, the pods are put in pending state until such a healthy node appears again.
- Watches API server for new work tasks.
- Responsible for distributing the work or containers across multiple nodes

Factors taken into account for scheduling decisions include:

1. Individual and collective resource requirements.
2. Hardware/software/policy constraints.
3. Affinity and anti-affinity specifications.
4. Data locality.
5. Inter-workload interference.
6. Deadlines and taints.

CONTROLLER MANAGER (kube-controller-manager)

- Controllers are the brain behind orchestration.
- It watches the desired state of the objects it manages and watches their current state via API server. It makes sure that the current state of objects is the desired state.
- It is the controller of controllers. It runs controller processes. Logically, each controller is a separate process but to reduce complexity, they are all compiled into single binary and run-in single process.
  Some of the types of controllers are:
  1. **Node Controller**: Responsible for noticing and responding when the nodes go down.
     >> provides CIDR block to a node's life (CIDR blocks are groups of addresses that share the same prefix and contain the same number of bits).
     >> keeps the list of nodes up to date with cloud providers list of available machines. Whenever a node is unhealthy, the node controller asks the cloud provider if the VM for that node is still available. If not, the node controller deletes the node from its list of nodes.
     >> The third is monitoring the nodes' health. The node controller is responsible for updating the NodeReady condition of NodeStatus to ConditionUnknown when a node becomes unreachable (i.e., the node controller stops receiving heartbeats for some reason, for example due to the node being down), and then later evicting all the pods from the node (using graceful termination) if the node continues to be unreachable. (The default timeouts are 40s to start reporting ConditionUnknown and 5m after that to start evicting pods.) The node controller checks the state of each node every --node-monitor period second.

2. **Job Controller**: Responsible for job objects that represent one off tasks, then creates pods to run those tasks to completion.
   >> Job controller is an example of controllers that come as part of Kubernetes itself ("built-in" controllers)
   >> When the Job controller sees a new job, it makes sure that, somewhere in your cluster, the kubelets on a set of Nodes are running the right number of Pods to get the work done. The Job controller does not run any Pods or containers itself. Instead, the Job controller tells the API server to create or remove Pods. Other components in the control plane act on the new information (there are new Pods to schedule and run), and eventually the work is done.
   >> After you create a new Job, the desired state is for that Job to be completed. The Job controller makes the current state for that Job be nearer to your desired state: creating Pods that do the work you wanted for that Job, so that the Job is closer to completion. Controllers also update the objects that configure them. For example: once the work is done for a Job, the Job controller updates that Job object to mark it Finished.

3. **Endpoint Controller**: populates the endpoints object (joins services and pods)

4. **Service Account and token controllers**: Create default accounts and API access tokens for new namespaces.

CLOUD-CONTROLLER MANAGER

- The cloud manager integrates with the underlying cloud technologies in the cluster when running in a cloud environment.
- This runs only controllers that are specific to cloud provider.
- This allows you to link your cluster into cloud provider's API and separates out the components that interact with the cloud platform from the components that only interact with the cluster.

  Node controllers: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding

  Route controllers: For setting up routes in the underlying cloud infrastructure.

  Service controllers: For creating, updating and deleting cloud provider load balancers.

NODE COMPONENTS:

CONTAINER RUNTIME:

It is the underlying software that's used to run containers. In our case, it is docker.

KUBELET:

It is the agent that runs on each node of the cluster. It makes sure that containers are running as expected in the pods.
The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.

KUBE-PROXY:

kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.
kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

CONTROL PLANE Vs NODE

CONTROL PLANE	NODE
	1) Node takes care of the containers/pods using container run time – Docker in our case
2) Kube-apiserver is what that makes that it a control plane	2) Similarly, the worker nodes have kubelets interacts with master to provide health information of worker node and carry out actions requested by master on worker nodes
3) All information gathered are stored in key-value store and is available in etcd.	
4) Controller is controller of controllers that is placed in control plane so that it takes care of the nodes, jobs, endpoints, service accounts and api access tokens	
5) Scheduler takes care of assignment of new pods/containers to nodes available.	

KUBERNETES SETUP

LOCAL ENVIRONMENTS:

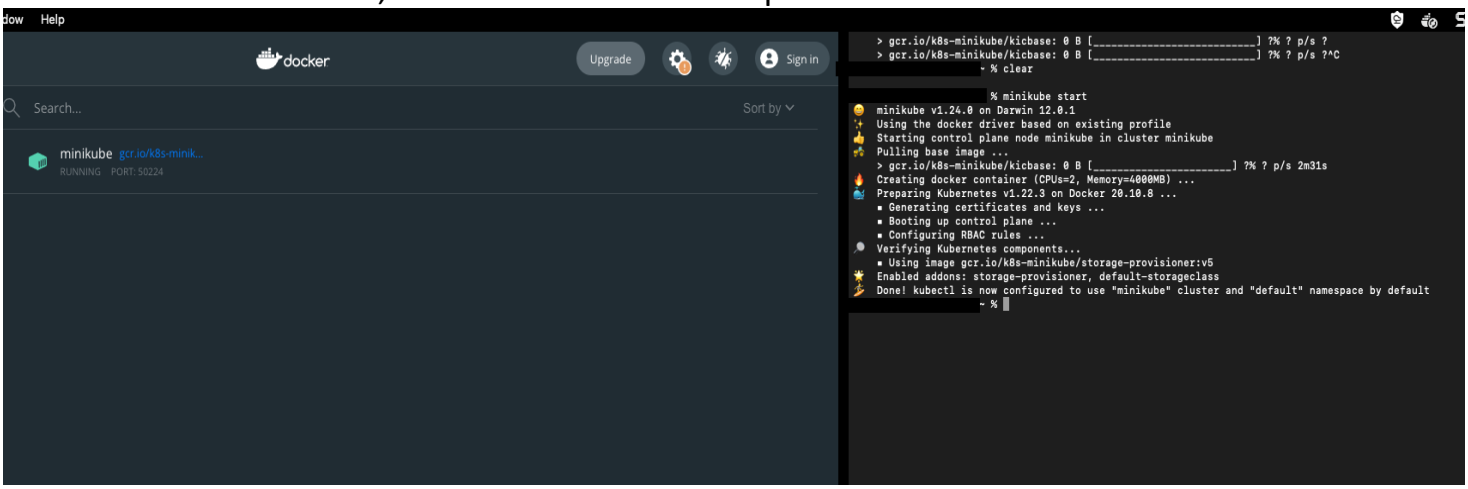
For getting started with Kubernetes local environments, we can use :

1. Minikube – single node local cluster.
2. Microk8s – simplest prod grade upstream k8s.
3. Kubeadm – multiple local clusters.

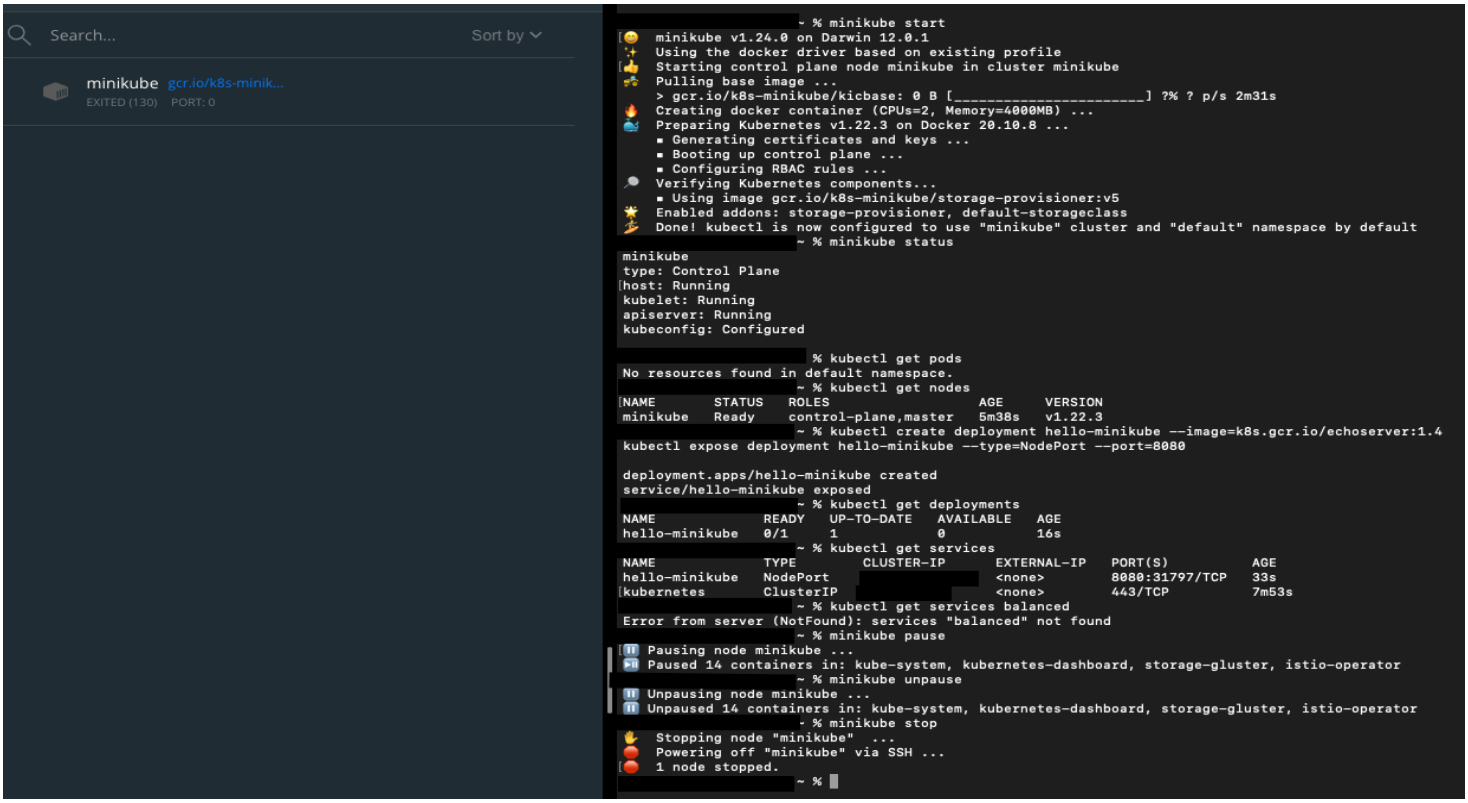
CLOUD PLATFORMS:

1. GCP
2. AWS
3. MICROSOFT AZURE

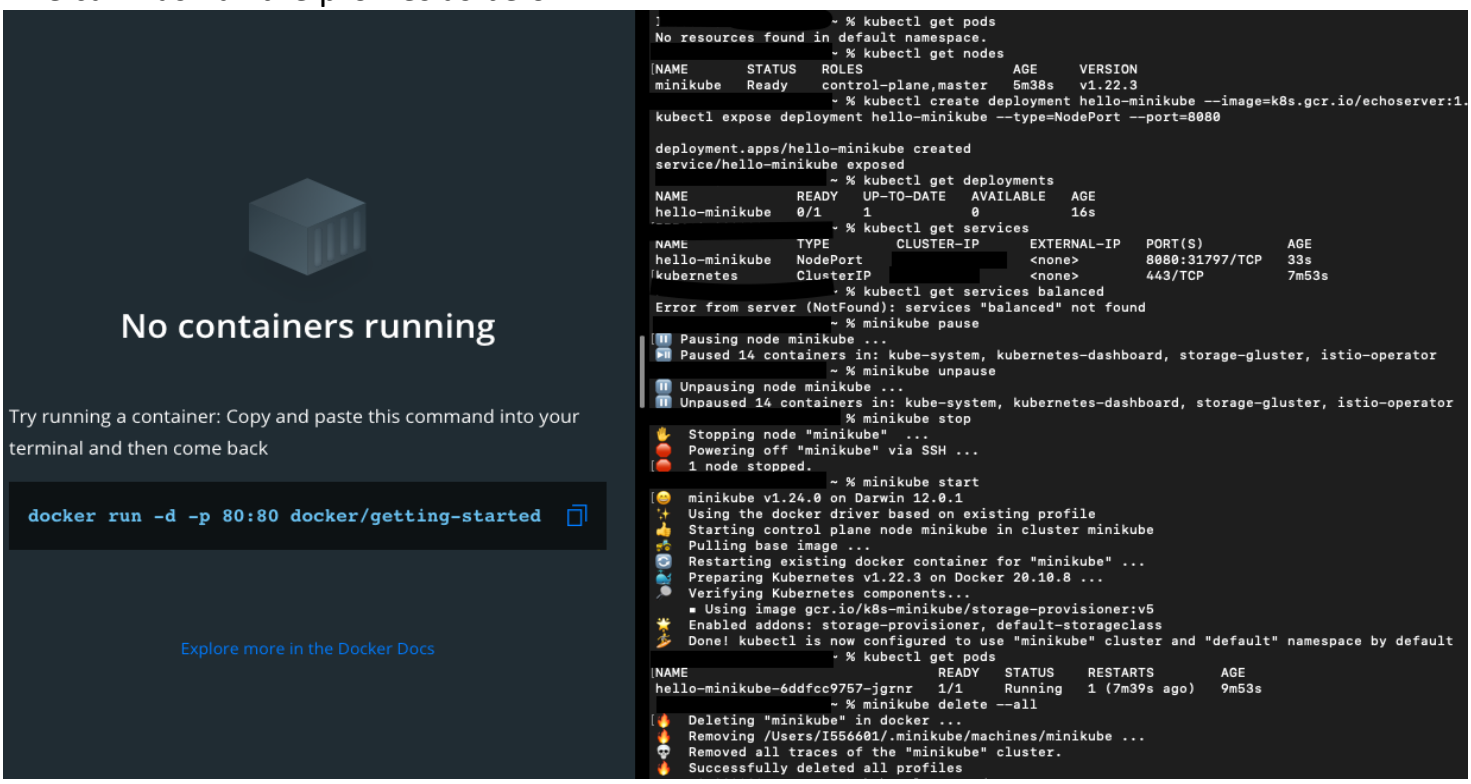
With Minikube + Kubectl, we can have a local setup as follows:



After this, we can have a primary test to test if Minikube and Kubectl are working fine in the machine.That is ,



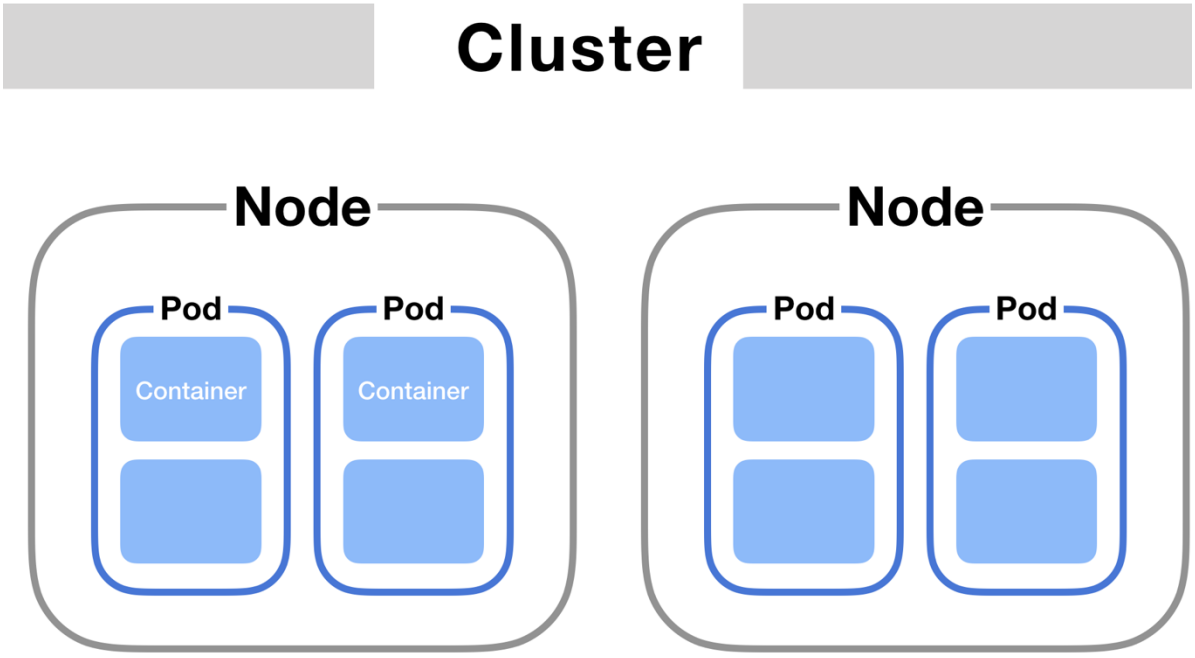
We can flush all the profiles as below:



KUBERNETES PODS:

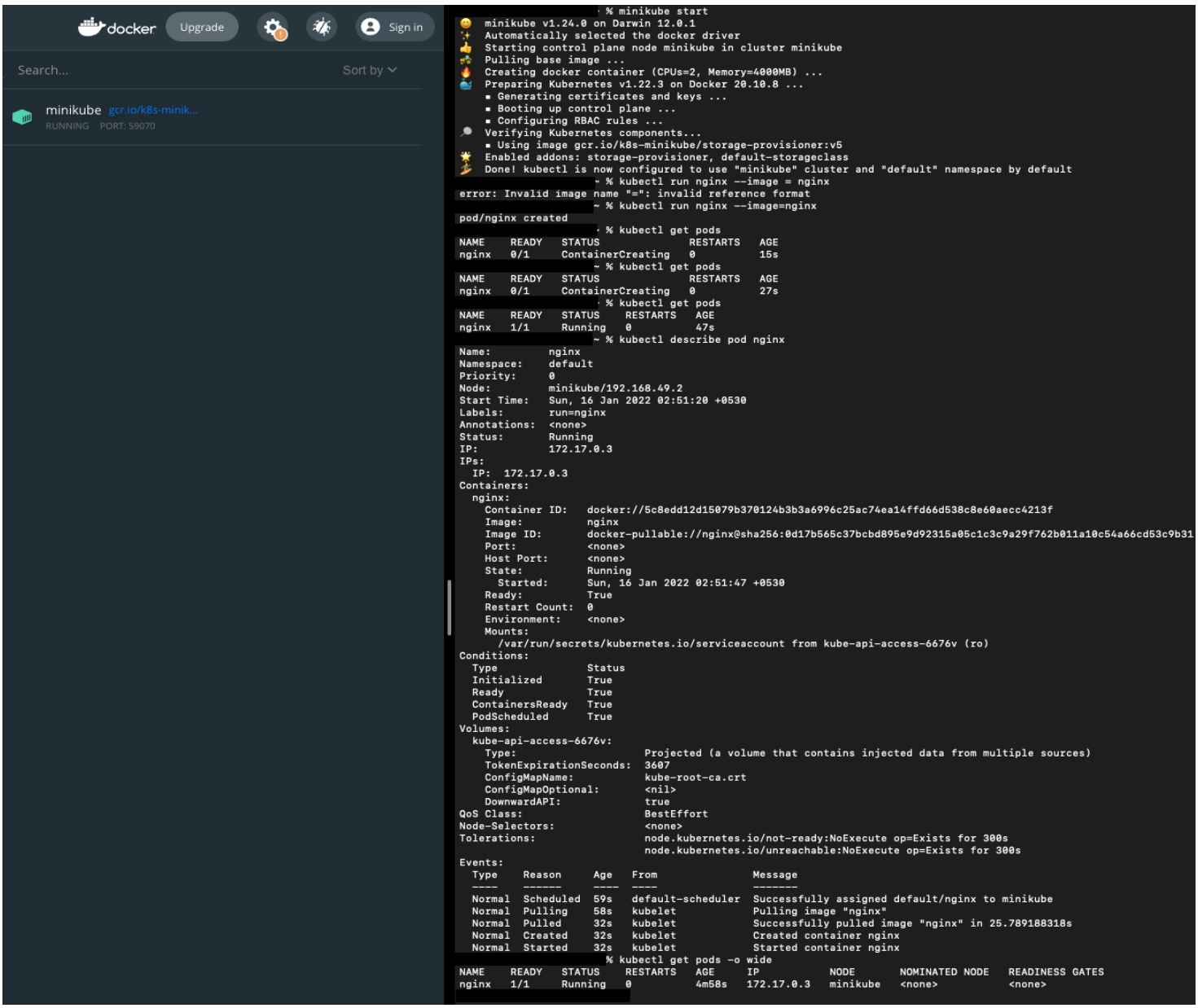
The pre-state is such that any application is containerized, and an image is already pushed into a docker registry such as docker hub. Similarly, we assume that a Kubernetes cluster is also set up by the organization. It can be either a single or a multi-node cluster.

Our ultimate goal is to deploy our application in the form of containers on a set of machines that are configured as nodes/minions in a cluster. However these containers are not directly deployed into these nodes.
The containers are encapsulated with/without other side-containers called as Pods.



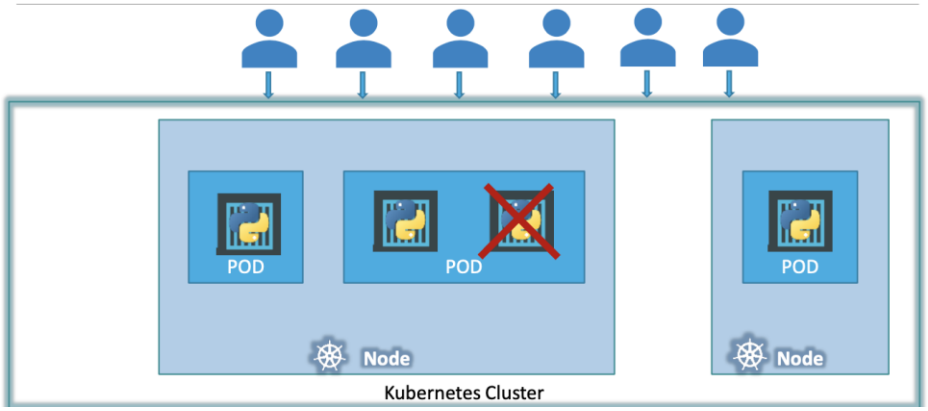
Pods are the smallest deployable units of computing that you can create and manage in Kubernetes.
A *Pod* (as in a pod of whales or pea pod) is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers. A Pod's contents are always co-located and co-scheduled, and run in a shared context. A Pod models an application-specific "logical host": it contains one or more application containers which are relatively tightly coupled. In non-cloud contexts, applications executed on the same physical or virtual machine are analogous to cloud applications executed on the same logical host.
For instance,
When we have a single instance of pod running in a node; if the traffic increases, we will need to bring in more instances. So, these instances are not directly deployed, they will be encapsulated into a pod and as the result, we will have multiple pods of the application running in a node. **So, container – pod is an 1-1 relationship. That is, the containers within a pod aren't of the same application/kind/app-purpose.**

Demo:



Here, if we notice, the IP of the node is 192.168.49.2. And that of the pod is 172.17.0.3

POD



The containers in a single pod may refer to each other as localhost as they share the same network space. They may share the same storage space as well.
In case of insufficient space in the node, the controller will bring up an extra node with similar instance of pod running inside of it.

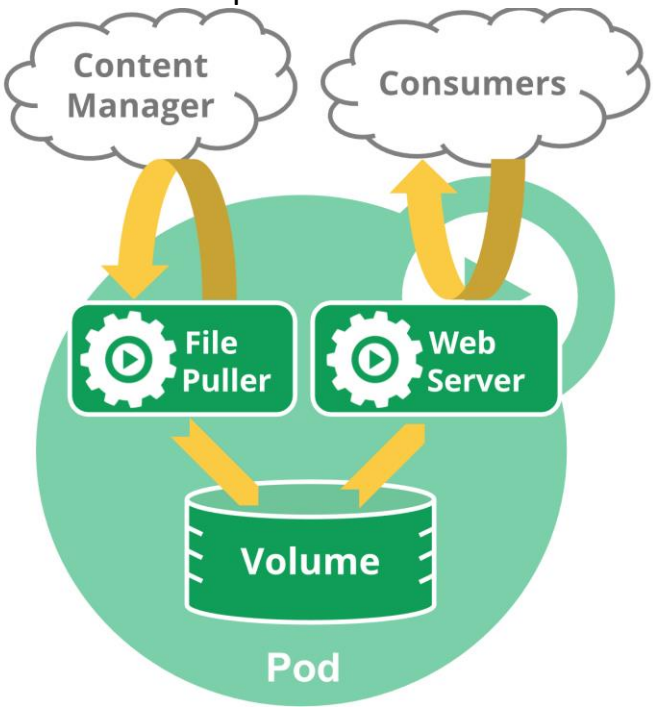
Using PODS:
Simple pod.yaml will be as follows:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

Usually you don't need to create Pods directly, even singleton Pods. Instead, create them using workload resources such as Deployment or Job. If your Pods need to track state, consider the StatefulSet resource.
Pods in a Kubernetes cluster are used in two main ways:

1. Pods that run a single container. The "one-container-per-Pod" model is the most common Kubernetes use case; in this case, you can think of a Pod as a wrapper around a single container; Kubernetes manages Pods rather than managing the containers directly.
2. Pods that run multiple containers that need to work together. A Pod can encapsulate an application composed of multiple co-located containers that are tightly coupled and need to share resources. These co-located containers form a single cohesive unit of service—for example, one container serving data stored in a shared volume to the public, while a separate sidecar container refreshes or updates those files. The Pod wraps these containers, storage resources, and an ephemeral network identity together as a single unit.

Each Pod is meant to run a single instance of a given application. If you want to scale your application horizontally (to provide more overall resources by running more instances in a node), you should use multiple Pods, one for each instance. In Kubernetes, this is typically referred to as replication.



Pods are designed to support multiple cooperating processes (as containers) that form a cohesive unit of service. The containers in a Pod are automatically co-located and co-scheduled on the same physical or virtual machine in the cluster. The containers can share resources and dependencies, communicate with one another, and coordinate when and how they are terminated.

When a Pod gets created (directly by you, or indirectly by a controller), the new Pod is scheduled to run on a Node in your cluster. The Pod remains on that node until

- The Pod finishes execution OR
- The Pod object is deleted OR
- The Pod is evicted for lack of resources OR
- The node fails.

PODS AND CONTROLLERS:

You can use workload resources to create and manage multiple Pods for you. **A controller for the resource handles replication and rollout and automatic healing in case of Pod failure**. For example, if a Node fails, a controller notices that Pods on that Node have stopped working and creates a replacement Pod. The scheduler places the replacement Pod onto a healthy Node.

Here are some examples of workload resources that manage one or more Pods:

- Deployment
- ReplicaSet
- StatefulSet
- DaemonSet

POD TEMPLATES:

Pod Templates are specifications for creating Pods, and are included in workload resources such as Deployments, Jobs, and DaemonSets. Controllers use this to create and manage pods on your behalf.

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: hello
spec:
  template:
    # This is the pod template
    spec:
      containers:
      - name: hello
        image: busybox
        command: ['sh', '-c', 'echo "Hello, Kubernetes!" && sleep 3600']
      restartPolicy: OnFailure
    # The pod template ends here
```

Modifying the pod template or switching to a new pod template has no direct effect on the Pods that already exist. However, if you change the pod template for a workload resource (Deployment/StatefulSet/DaemonSet), that resource needs to create replacement Pods that use the updated template.

For instance,

The StatefulSet controller ensures that the running Pods match the current pod template for each StatefulSet object. If you edit the StatefulSet to change its pod template, the StatefulSet starts to create new Pods based on the updated template. Eventually, all the old Pods are replaced with new Pods, and the update is complete.

POD UPDATE AND REPLACEMENT

When the pod template for a workload resource is changed, the controller creates new pods based on the updated template instead of updating or patching the existing pods.

Kubernetes, however still allows you to update few fields on the existing running pods. Though, most of the metadata of the pod is still immutable we can still edit spec.containers[*].image, **or** spec.initContainers[*].image, spec.activeDeadlineSeconds or spec.tolerations.

For spec.tolerations

STORAGE IN PODS:

A pod can specify shared set of storage volumes. All containers in the pods can access the shared volumes, allowing those containers to share data. Volumes allow persistent data in pods to survive in case one of the containers within needs to be restarted.

POD NETWORKING:

Each Pod is assigned a unique IP address for each address family. Every container in a Pod shares the network namespace, including the IP address and network ports. Within a Pod, containers share an IP address and port space and can find each other via localhost.

When containers in a Pod communicate with entities outside the Pod, they must coordinate how they use the shared network resources (such as ports

Containers in different Pods have distinct IP addresses and cannot communicate by IPC without special configuration.

STATIC PODS:

Static Pods are managed directly by the kubelet daemon on a specific node, without the API server observing them. Whereas most Pods are managed by the control plane (for example, a Deployment), for static Pods, the kubelet directly supervises each static Pod (and restarts it if it fails).

The main use for static Pods is to run a self-hosted control plane: in other words, using the kubelet to supervise the individual control plane components.

The kubelet automatically tries to create a mirror Pod on the Kubernetes API server for each static Pod. This means that the Pods running on a node are visible on the API server but cannot be controlled from there.

YAML:

⇒ Dictionary

⇒ Lists

⇒ Lists of Dictionaries

DICTIONARY:

```
Banana:
  Calories: 105
  Fat: 0.4g
  Carbs: 27
```

"="

```
Banana:
  Calories: 105
  Carbs: 27
  Fat: 0.4
```

Dictionary is an **unordered** list of items which is used to describe the characteristics of an object.

LISTS:

```
Fruits:
  - Apple
  - Banana
  - Orange
```

"!="

```
Fruits:
  - Orange
  - Banana
  - Apple
```

List is an ordered list of items which is used to describe the multiple options in the characteristics of an object that might be possible at once.

LISTS OF DICTIONARY:

Dictionary which has lists of dictionary as part of it.For instance,the employee yaml with list of dictionary will look like this:

```
Employee:
  Name: Jacob
  Sex: Male
  Age: 30
  Title: Systems Engineer
  Projects:
    - Automation
    - Support
  Payslips:
    - Month: June
      Wage: 4000
    - Month: July
      Wage: 4500
    - Month: August
      Wage: 4000
```

YAML IN KUBERNETES:

Yaml in Kubernetes need to have below root level properties as required fields:

Pod-definition.yml

```
apiVersion:
kind:
metadata:
```

```
spec:
```

- API VERSION

Depending on the kind, we need to specify the right apiVersion. This will be in the form of a string.
That is:

Kind	apiVersion
CertificateSigningRequest	certificates.k8s.io/v1beta1
ClusterRoleBinding	rbac.authorization.k8s.io/v1
ClusterRole	rbac.authorization.k8s.io/v1
ComponentStatus	v1
ConfigMap	v1
ControllerRevision	apps/v1
CronJob	batch/v1beta1
DaemonSet	extensions/v1beta1
Deployment	extensions/v1beta1
Endpoints	v1
Event	v1
HorizontalPodAutoscaler	autoscaling/v1
Ingress	extensions/v1beta1
Job	batch/v1
LimitRange	v1
Namespace	v1
NetworkPolicy	extensions/v1beta1
Node	v1
PersistentVolumeClaim	v1
PersistentVolume	v1
PodDisruptionBudget	policy/v1beta1
Pod	v1
PodSecurityPolicy	extensions/v1beta1
PodTemplate	v1
ReplicaSet	extensions/v1beta1
ReplicationController	v1
ResourceQuota	v1
RoleBinding	rbac.authorization.k8s.io/v1
Role	rbac.authorization.k8s.io/v1
Secret	v1
ServiceAccount	v1
Service	v1
StatefulSet	apps/v1

The api versions mean as follows:

alpha
API versions with ‘alpha’ in their name are early candidates for new functionality coming into Kubernetes. These may contain bugs and are not guaranteed to work in the future.

beta
‘beta’ in the API version name means that testing has progressed past alpha level, and that

the feature will eventually be included in Kubernetes. Although the way it works might change, and the way objects are defined may change completely, the feature itself is highly likely to make it into Kubernetes in some form.

stable
These do not contain ‘alpha’ or ‘beta’ in their name. They are safe to use.
v1
This was the first stable release of the Kubernetes API. It contains many core objects.

apps/v1
apps is the most common API group in Kubernetes, with many core objects being drawn from it and v1. It includes functionality related to running applications on Kubernetes, like Deployments, RollingUpdates, and ReplicaSets.

autoscaling/v1
This API version allows pods to be autoscaled based on different resource usage metrics. This stable version includes support for only CPU scaling, but future alpha and beta versions will allow you to scale based on memory usage and custom metrics.

batch/v1
The batch API group contains objects related to batch processing and job-like tasks (rather than application-like tasks like running a webserver indefinitely). This apiVersion is the first stable release of these API objects.

batch/v1beta1
A beta release of new functionality for batch objects in Kubernetes, notably including CronJobs that let you run Jobs at a specific time or periodicity.

certificates.k8s.io/v1beta1
This API release adds functionality to validate network certificates for secure communication in your cluster.

extensions/v1beta1
This version of the API includes many new, commonly used features of Kubernetes. Deployments, DaemonSets, ReplicaSets, and Ingresses all received significant changes in this release.
Note that in Kubernetes 1.6, some of these objects were relocated from extensions to specific API groups (e.g. apps). When these objects move out of beta, expect them to be in a specific API group like apps/v1. Using extensions/v1beta1 is becoming deprecated—try to use the specific API group where possible, depending on your Kubernetes cluster version.

policy/v1beta1
This apiVersion adds the ability to set a pod disruption budget and new rules around pod security.

rbac.authorization.k8s.io/v1
This apiVersion includes extra functionality for Kubernetes role-based access control. This helps you to secure your cluster.

- KIND

The kind refers to the type of object we are trying to create. This will be in the form of string.

- METADATA

Data that helps uniquely identify the object, including labels, name, UID and namespace(optional).
This will be in the form of a dictionary and may also consist of list/list of dictionaries.

- SPEC

This provides the state the object described should be in.
This will be in the form of a dictionary and will consist of list/list of dictionaries.

Once the pod-definition.yaml file is created, when

```
Kubectl create -f pod-definition.yaml
```

Is executed, we can see the pod created with the help of “Kubectl get pods” and to get detailed information about the pod created (for instance, considering the nginx pod created), we can use “Kubectl describe nginx” to see its characteristics such as pod network, container network, storage and other details.

Demo

Demo – 1

```
machine-name@machine-name ~ % kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          2d15h
machine-name@machine-name ~ % vim pod.yaml
machine-name@machine-name ~ % cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-1
  labels:
    app: nginx
    tier: frontend
spec:
  containers:
  - name: nginx
    image: nginx
machine-name@machine-name ~ % kubectl apply -f pod.yaml

pod/nginx-1 created
machine-name@machine-name ~ % kubectl get pods
NAME      READY   STATUS              RESTARTS   AGE
nginx     1/1     Running             0          2d15h
nginx-1   0/1     ContainerCreating   0          6s
machine-name@machine-name ~ % kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
nginx     1/1     Running   0          2d15h
nginx-1   1/1     Running   0          25s
machine-name@machine-name ~ % kubectl describe pod nginx-1
Name:         nginx-1
Namespace:    default
Priority:     0
Node:         minikube/192.168.49.2
Start Time:   Tue, 18 Jan 2022 18:04:36 +0530
Labels:       app=nginx
              tier=frontend
Annotations:  <none>
Status:       Running
IP:           172.17.0.4
IPs:
  IP:  172.17.0.4
Containers:
  nginx:
    Container ID:   docker://e7481c49026c42727d16b2ee315378758a888f57bffb4ead9508b8727c5b4e3a
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:0d17b565c37bcbd895e9d92315a05c1c3c9a29f762b011a10c54a66cd53c9b31
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Tue, 18 Jan 2022 18:04:42 +0530
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-t7d4g (ro)
Conditions:
```

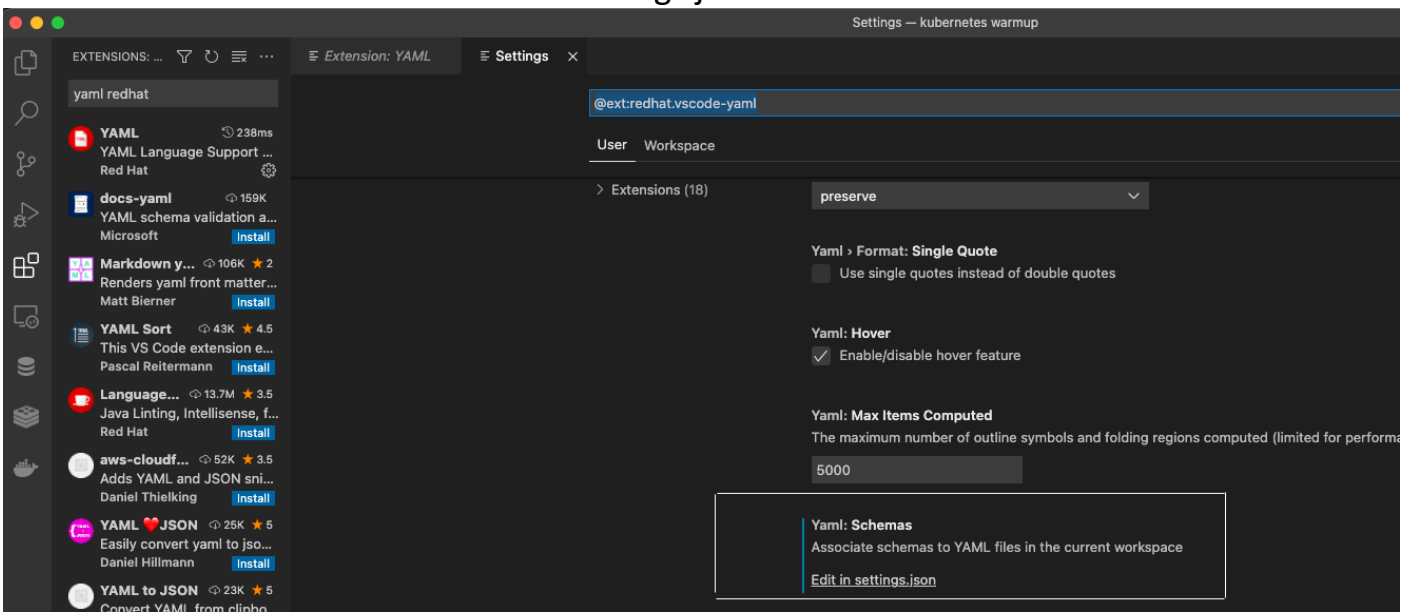
```
  Type           Status
  Initialized    True
  Ready          True
  ContainersReady   True
  PodScheduled   True
Volumes:
  kube-api-access-t7d4g:
    Type:                Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:       kube-root-ca.crt
    ConfigMapOptional:   <nil>
    DownwardAPI:         true
QoS Class:               BestEffort
Node-Selectors:          <none>
Tolerations:             node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                         node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  66s   default-scheduler  Successfully assigned default/nginx-1 to minikube
  Normal  Pulling    65s   kubelet            Pulling image "nginx"
  Normal  Pulled     60s   kubelet            Successfully pulled image "nginx" in 4.4960349s
  Normal  Created    60s   kubelet            Created container nginx
  Normal  Started    60s   kubelet            Started container nginx
machine-name@machine-name ~ %
```

YAML FORMATTING:

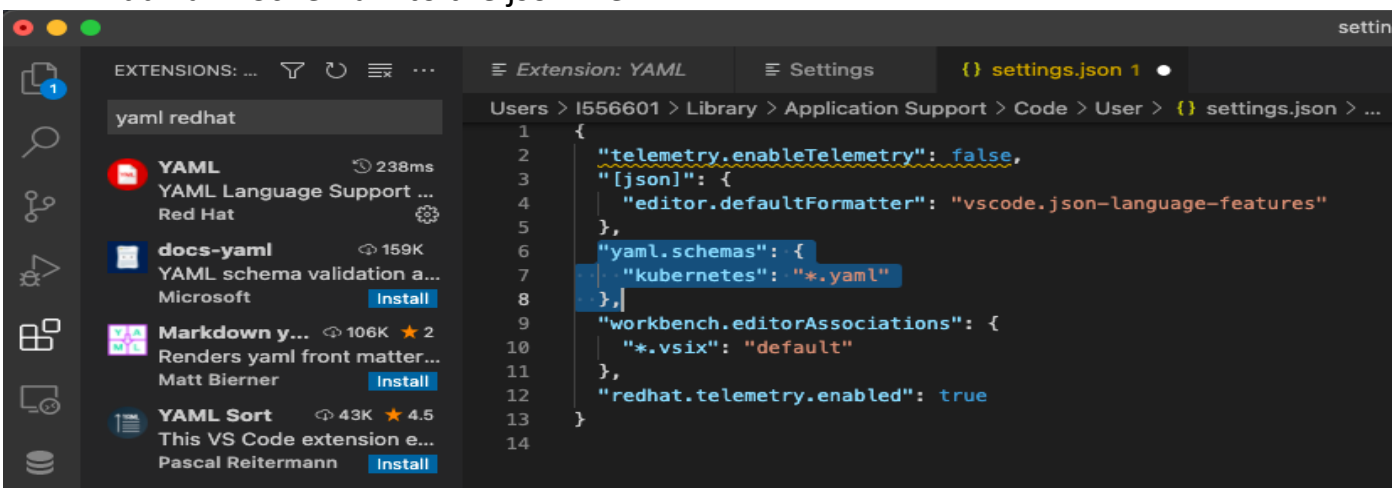
Many builds and deployments fail a lot of times due to incorrect indentation and formats of yaml files. To remediate this issue, we can make use of “YAML”- VS plugin which is very efficient and time saver.

As shown in screen shot below, we can set it up and format the documents.

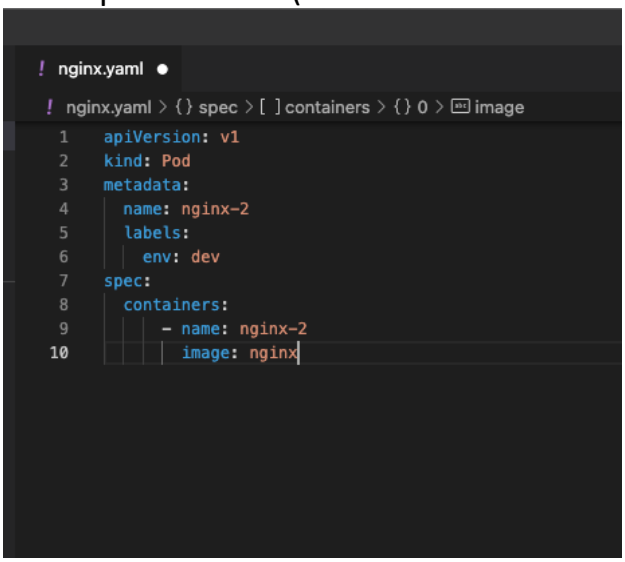
1. Go to Yaml:Schemas >> edit in settings.json

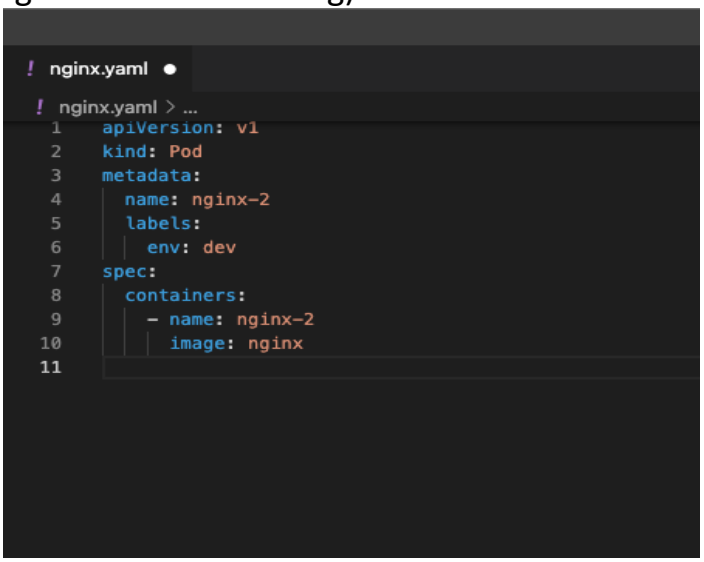


2. Add Yaml Schema into the json file



3. Once this setting is saved, we can select any yaml file, right click, select format document and YAML plugin will format the file for us according to Kubernetes preferences.(left -before formatting, right – after formatting)





Demo-2

machine kubernetes warmup % ls
nginx.yaml pod.yaml
machine kubernetes warmup % cat nginx.yaml
apiVersion: v1
kind: Pod
metadata:
name: nginx-2
labels:
env: dev
spec:
containers:
- name: nginx-2
image: nginx
machine kubernetes warmup % kubectl apply -f nginx.yaml
The connection to the server 127.0.0.1:59074 was refused - did you specify the right host or port?
machine kubernetes warmup % minikube start
😄 minikube v1.24.0 on Darwin 12.1

```
Using the docker driver based on existing profile
Starting control plane node minikube in cluster minikube
Pulling base image ...
Restarting existing docker container for "minikube" ...
Preparing Kubernetes v1.22.3 on Docker 20.10.8 ...
Verifying Kubernetes components...
  ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
Enabled addons: storage-provisioner, default-storageclass
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
machine kubernetes warmup % kubectl apply -f nginx.yaml
pod/nginx-2 created
machine kubernetes warmup % kubectl get pods
NAME      READY   STATUS    RESTARTS      AGE
nginx     1/1     Running   1 (10m ago)   3d20h
nginx-1   1/1     Running   1 (10m ago)   29h
nginx-2   1/1     Running   0             9s
machine kubernetes warmup % kubectl delete pods
error: resource(s) were provided, but no name was specified
machine kubernetes warmup % kubectl delete pod nginx
pod "nginx" deleted
machine kubernetes warmup % kubectl delete pod nginx nginx-1 nginx-2
pod "nginx-1" deleted
pod "nginx-2" deleted
Error from server (NotFound): pods "nginx" not found
machine kubernetes warmup % kubectl get pods
No resources found in default namespace.
machine kubernetes warmup %
```

REPLICATION CONTROLLERS AND REPLICATION SETS:

REPLICATION CONTROLLER:

A ReplicationController ensures that a specified number of pod replicas are running at any one time. In other words, a ReplicationController makes sure that a pod or a homogeneous set of pods is always up and available.
Even when a pod has a single instance of container running, in case of failure replication controller helps to bring up required number of containers no matter what.
There can be one or more replicas in a pod. Replication controllers also help in case of load balancing and scaling. It spans across multiple nodes in the cluster and hence helps balance load across paths on different nodes as well as scale the application when demand increases.
Replication Controllers are replaced by replication sets as its recommended.

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

Run the above replication.yaml file using :

```
kubectl apply -f https://k8s.io/examples/controllers/replication.yaml
```

or

```
kubectl create -f https://k8s.io/examples/controllers/replication.yaml
```

Now to check the status of this replication controller object, we can use the command below and output will look like:

```
kubectl describe replicationcontrollers/nginx
```

```
Name:         nginx
Namespace:    default
Selector:     app=nginx
Labels:       app=nginx
Annotations:    <none>
```

```
Replicas:    3 current / 3 desired
Pods Status: 0 Running / 3 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:       app=nginx
  Containers:
   nginx:
    Image:              nginx
    Port:               80/TCP
    Environment:        <none>
    Mounts:             <none>
  Volumes:              <none>
Events:
  FirstSeen       LastSeen     Count    From                    SubobjectPath    Type      Reason            Message
  ---------       --------     -----    ----                    -------------    ----      ------            -------
  20s             20s          1        {replication-controller }                Normal    SuccessfulCreate  Created pod:
nginx-qrm3m
  20s             20s          1        {replication-controller }                Normal    SuccessfulCreate  Created pod:
nginx-3ntk0
  20s             20s          1        {replication-controller }                Normal    SuccessfulCreate  Created pod:
nginx-4ok8v
```

WRITING A REPLICATIONCONTROLLER SPEC

- For ReplicationController needs apiVersion, kind and metadata fields. However, .spec.template is the only required filed of the .spec.
- The .spec.template is a pod template and it has exactly the same schema as a pod, except it is nested and doesn't have a apiVersion and kind.
  That is .spec.template.spec.
- For local container restarts, controller delegates to an agent, to kubelet or Docker.

LABELS ON REPLICATION CONTROLLER

The ReplicationController itself can have labels. That is, Metadata – labels similar to the parent yaml structure, we can find it in the spec of replicaController too.

POD SELECTOR

A replication controller manages all the pods with labels that matches with that of the selector.It doesn't distinguish between pods that it created or deleted or whether if it was created by other person/process.
This allows the ReplicationController to be replaced without affecting the pods that are running.
So .spec.template.metadata.labels must be equal to .spec.selector. or it will be rejected by API.
If .spec.selector is unspecified, it will be defaulted to .spec.template.metadata.labels.

Also you should not normally create any pods whose labels match this selector, either directly, with another ReplicationController, or with another controller such as Job. If you do so, the ReplicationController thinks that it created the other pods.

If you do end up with multiple controllers that have overlapping selectors, maintenance will be difficult and deletion too.

MULTIPLE REPLICAS

You can specify how many pods should run concurrently by setting .spec.replicas to the number of pods you would like to have running concurrently.

Deleting the Replication Controllers and its pods

To delete, we can use Kubectl delete. This will delete the pods and then deletes the replication controller itself. In case if just the controller wants to be deleted, we use –cascade=orphan option so that the pods will remain unscathed.
Once the original is deleted, you can create a new ReplicationController to replace it. As long as the old and new .spec.selector are the same, then the new one will adopt the old pods.

RESCHEDULING

a ReplicationController will ensure that the specified number of pods exists, even in the event of node failure or pod termination

SCALING

This enables scaling the number of replicas up or down, either manually or via auto-scaling control agent by updating replicas field.

ROLLING UPDATES

The approach is to

1. create another rolling controller along with the old one in a cluster.
2. update +1 pod to the new controller and -1 pod to the old controller.
3. Delete the old controller at the end so that it doesn't have any orphan pods.

MULTIPLE RELEASE TRACKS

We can have multiple ReplicationControllers targeting both canary and non-canary pods using labels.
For instance, you can have 10 replicated pods for prod or
Have:

1. 9 pods with 1 replication controller labeled tier=frontend, environment=prod, track=stable
2. 1 pod with another replicationController labeled tier=frontend, environment=prod, track=canary

DEMO

```
machinename@machinename kubernetes warmup % cat rc-deinition.yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: myapp-rc
  labels:
    app: myapp
    type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
        - name: nginx-container
          image: nginx
  replicas: 3
machinename@machinename kubernetes warmup % kubectl create -f rc-deinition.yaml
replicationcontroller/myapp-rc created
machinename@machinename kubernetes warmup % kubectl get replicationcontroller
NAME       DESIRED   CURRENT   READY   AGE
myapp-rc   3         3         2       12s
machinename@machinename kubernetes warmup % kubectl get pods
NAME             READY   STATUS    RESTARTS   AGE
myapp-rc-bpc2l   1/1     Running   0          20s
myapp-rc-c692r   1/1     Running   0          20s
myapp-rc-fxz5p   1/1     Running   0          20s
machinename@machinename kubernetes warmup %
```

REPLICATIONSET

A replicationSet is defined by:

1. Selector: that specifies how to identify the pods the resource can acquire.
2. Number of replicas: indicating how many pods the resource should be maintaining
3. Pod template: which specifies he data of new pods the resource should create.
4. ReplicaSet: This then create the number of replicas that's required to reach the number.

A ReplicaSet is bound to its pods using .metadata.ownerReferences field, which specifies what resource the current object is owned by.
A ReplicaSet identifies new Pods to acquire by using its selector. If there is a Pod that has no OwnerReference or the OwnerReference is not a Controller and it matches a ReplicaSet's selector, it will be immediately acquired by said ReplicaSet.

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # modify replicas according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

Demo:

```
machinename@machinename kubernetes warmup % kubectl create -f rc-deinition.yaml
```

```
replicationcontroller/myapp-rc created
machinename@machinename kubernetes warmup % kubectl get replicationcontroller
NAME       DESIRED   CURRENT   READY   AGE
myapp-rc   3         3         2       12s
machinename@machinename kubernetes warmup % kubectl get pods
NAME             READY   STATUS    RESTARTS   AGE
myapp-rc-bpc2l   1/1     Running   0          20s
myapp-rc-c692r   1/1     Running   0          20s
myapp-rc-fxz5p   1/1     Running   0          20s
machinename@machinename kubernetes warmup % kubectl apply -f replicasets.yaml
replicaset.apps/frontend created
machinename@machinename kubernetes warmup % kubectl get replicasets
NAME       DESIRED   CURRENT   READY   AGE
frontend   3         3         0       15s
machinename@machinename kubernetes warmup % kubectl get pods
NAME             READY   STATUS              RESTARTS   AGE
frontend-4594q   0/1     ContainerCreating   0          24s
frontend-rwn2f   0/1     ContainerCreating   0          24s
frontend-ztpt7   0/1     ContainerCreating   0          24s
myapp-rc-bpc2l   1/1     Running             0          125m
myapp-rc-c692r   1/1     Running             0          125m
myapp-rc-fxz5p   1/1     Running             0          125m
machinename@machinename kubernetes warmup % kubectl get all
NAME                 READY   STATUS              RESTARTS   AGE
pod/frontend-4594q   0/1     ContainerCreating   0          37s
pod/frontend-rwn2f   0/1     ContainerCreating   0          37s
pod/frontend-ztpt7   0/1     ContainerCreating   0          37s
pod/myapp-rc-bpc2l   1/1     Running             0          125m
pod/myapp-rc-c692r   1/1     Running             0          125m
pod/myapp-rc-fxz5p   1/1     Running             0          125m

NAME                           DESIRED   CURRENT   READY   AGE
replicationcontroller/myapp-rc   3         3         3       125m

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   8d

NAME                         DESIRED   CURRENT   READY   AGE
replicaset.apps/frontend     3         3         0       37s
machinename@machinename kubernetes warmup % cat replicasets.yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # modify replicas according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
```

```
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3%
machinename@machinename kubernetes warmup %
```

Pod Replicas (Green section above)

This section specifies how many pods should be run concurrently. If not explicitly given, defaults to 1.

Pod Selector (Yellow Section above)

This a label selector.All the pods or objects with the matching label will be considered as a potential match-pod to acquire.

Pod Template (blue section above)

This section is also required to have a label that helps the resource to match and aquire the object.

A replicasection can be auto scaled via HPA(Horizontal pod autoscalers).For example:

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-scaler
spec:
  scaleTargetRef:
    kind: ReplicaSet
    name: frontend
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

Alternatively, you can have this autoscaling done via command as well.

```
$ kubectl autoscale rs frontend --max=10 --min=3 --cpu-percent=50
```

- **The Major difference between replicationController and replicaset is the ".spec.selector" section.That is, along with the pods that were created by the resource, it can consider all the other pods too which has a label that's provided in this section.**
- **Also, if the pods aren't available, the replicaset will create them for you matching the minReplicas that need to be running at any given instance of time.**

Scaling

Suppose if we are not using HPA and need to scale using Kubectl commands, there are couple of ways to achieve it.

1. Kubectl replace -f replicationSet.yaml (where file is updated with 6 replicas instead of 3)
2. Kubectl scale –replicas=6 -f replicationSet.yaml
3. Kubectl scale –replicas=6 replicaset myapp-replicaset
   Where replicaset is the type and myapp-replicaset is the name of the type of the resource.

Demo

```
machinename@machinename kubernetes warmup % kubectl delete pod frontend-4594q
pod "frontend-4594q" deleted
machinename@machinename kubernetes warmup % kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
frontend-94pkn   1/1     Running   0          8s
frontend-rwn2f   1/1     Running   0          72m
frontend-ztpt7   1/1     Running   0          72m
myapp-rc-bpc2l   1/1     Running   0          3h17m
myapp-rc-c692r   1/1     Running   0          3h17m
myapp-rc-fxz5p   1/1     Running   0          3h17m
machinename@machinename kubernetes warmup % kubectl delete pod frontend-94pkn frontend-rwn2f
pod "frontend-94pkn" deleted
pod "frontend-rwn2f" deleted
machinename@machinename kubernetes warmup % kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
frontend-5b262   1/1     Running   0          5s
frontend-rftx7   1/1     Running   0          4s
frontend-ztpt7   1/1     Running   0          73m
myapp-rc-bpc2l   1/1     Running   0          3h18m
myapp-rc-c692r   1/1     Running   0          3h18m
myapp-rc-fxz5p   1/1     Running   0          3h18m
machinename@machinename kubernetes warmup % kubectl describe replicaset
Name:         frontend
Namespace:    default
Selector:     tier=frontend
Labels:       app=guestbook
              tier=frontend
Annotations:  <none>
Replicas:     3 current / 3 desired
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  tier=frontend
  Containers:
   php-redis:
    Image:        gcr.io/google_samples/gb-frontend:v3
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
```

```
  Volumes:        <none>
Events:
  Type    Reason            Age    From                   Message
  ----    ------            ----   ----                   -------
  Normal  SuccessfulCreate  2m34s  replicaset-controller  Created pod: frontend-94pkn
  Normal  SuccessfulCreate  110s   replicaset-controller  Created pod: frontend-5b262
  Normal  SuccessfulCreate  109s   replicaset-controller  Created pod: frontend-rftx7
machinename@machinename kubernetes warmup % kubectl edit replicaset frontend
replicaset.apps/frontend edited
machinename@machinename kubernetes warmup % kubectl get pods
NAME             READY   STATUS    RESTARTS   AGE
frontend-5b262   1/1     Running   0          9m54s
frontend-kt965   1/1     Running   0          12s
frontend-rftx7   1/1     Running   0          9m53s
frontend-ztpt7   1/1     Running   0          83m
myapp-rc-bpc2l   1/1     Running   0          3h28m
myapp-rc-c692r   1/1     Running   0          3h28m
myapp-rc-fxz5p   1/1     Running   0          3h28m
machinename@machinename kubernetes warmup % kubectl scale --replicas=10 replicaset frontend
replicaset.apps/frontend scaled
machinename@machinename kubernetes warmup % kubectl get pods
NAME             READY   STATUS              RESTARTS   AGE
frontend-5b262   1/1     Running             0          10m
frontend-826sn   0/1     ContainerCreating   0          4s
frontend-c7696   0/1     ContainerCreating   0          4s
frontend-cjv5x   0/1     ContainerCreating   0          4s
frontend-fbwqd   0/1     ContainerCreating   0          4s
frontend-hz6cj   0/1     ContainerCreating   0          4s
frontend-kt965   1/1     Running             0          58s
frontend-rftx7   1/1     Running             0          10m
frontend-v9kf4   0/1     ContainerCreating   0          4s
frontend-ztpt7   1/1     Running             0          84m
myapp-rc-bpc2l   1/1     Running             0          3h29m
myapp-rc-c692r   1/1     Running             0          3h29m
myapp-rc-fxz5p   1/1     Running             0          3h29m
machinename@machinename kubernetes warmup % kubectl scale --replicas=3 replicaset frontend
replicaset.apps/frontend scaled
machinename@machinename kubernetes warmup % kubectl get pods
NAME             READY   STATUS        RESTARTS   AGE
frontend-5b262   1/1     Running       0          10m
frontend-826sn   1/1     Terminating   0          22s
frontend-c7696   0/1     Terminating   0          22s
frontend-cjv5x   1/1     Terminating   0          22s
frontend-rftx7   1/1     Running       0          10m
frontend-v9kf4   1/1     Terminating   0          22s
frontend-ztpt7   1/1     Running       0          84m
myapp-rc-bpc2l   1/1     Running       0          3h29m
myapp-rc-c692r   1/1     Running       0          3h29m
myapp-rc-fxz5p   1/1     Running       0          3h29m
machinename@machinename kubernetes warmup % kubectl apply -f sample.yaml
deployment.apps/deploy created
machinename@machinename kubernetes warmup % kubectl get deployments
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
deploy   0/3     3            0           8s
```

```
machinename@machinename kubernetes warmup % kubectl get all
NAME                          READY   STATUS              RESTARTS   AGE
pod/deploy-6b9bcc5f46-gbpqk   1/1     Running             0          15s
pod/deploy-6b9bcc5f46-hv7gt   1/1     Running             0          15s
pod/deploy-6b9bcc5f46-pr5m5   0/1     ContainerCreating   0          15s
pod/frontend-5b262            1/1     Running             0          60m
pod/frontend-rftx7            1/1     Running             0          60m
pod/frontend-ztpt7            1/1     Running             0          133m
pod/myapp-rc-bpc2l            1/1     Running             0          4h18m
pod/myapp-rc-c692r            1/1     Running             0          4h18m
pod/myapp-rc-fxz5p            1/1     Running             0          4h18m

NAME                            DESIRED   CURRENT   READY   AGE
replicationcontroller/myapp-rc   3         3         3       4h18m

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   8d

NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/deploy   2/3     3            2           15s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/deploy-6b9bcc5f46   3         3         2       15s
replicaset.apps/frontend            3         3         3       133m
machinename@machinename kubernetes warmup % cat sample.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy
  labels:
    app: myapp-deploy
    tier: frontend
spec:
  replicas: 3
  template:
    metadata:
      name: deploy
      labels:
        app: myapp-deploy
    spec:
      containers:
        - name: nginx
          image: nginx
  selector:
    matchLabels:
      app: myapp-deploy
machinename@machinename kubernetes warmup % kubectl describe deployment deploy
Name:                   deploy
Namespace:              default
CreationTimestamp:      Mon, 24 Jan 2022 18:25:27 +0530
Labels:                 app=myapp-deploy
                        tier=frontend
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=myapp-deploy
```

```
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=myapp-deploy
  Containers:
   nginx:
    Image:        nginx
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   deploy-6b9bcc5f46 (3/3 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  43s   deployment-controller  Scaled up replica set deploy-6b9bcc5f46 to 3
```

UPDATE AND ROLLBACK

There are rollouts done during the revision of the applications and command used is :

Demo:

```
machinename@machinename kubernetes warmup % kubectl apply -f sample.yaml --record
Flag --record has been deprecated, --record will be removed in the future
deployment.apps/deploy created
machinename@machinename kubernetes warmup % kubectl rollout status deployment.apps/deploy
```

```
Waiting for deployment "deploy" rollout to finish: 0 of 6 updated replicas are available...
Waiting for deployment "deploy" rollout to finish: 1 of 6 updated replicas are available...
Waiting for deployment "deploy" rollout to finish: 2 of 6 updated replicas are available...
Waiting for deployment "deploy" rollout to finish: 3 of 6 updated replicas are available...
Waiting for deployment "deploy" rollout to finish: 4 of 6 updated replicas are available...
Waiting for deployment "deploy" rollout to finish: 5 of 6 updated replicas are available...
deployment "deploy" successfully rolled out
machinename@machinename kubernetes warmup % kubectl rollout history deployment.apps/deploy
deployment.apps/deploy
REVISION  CHANGE-CAUSE
1         kubectl apply --filename=sample.yaml --record=true
machinename@machinename kubernetes warmup % kubectl describe deployment deploy
Name:                   deploy
Namespace:              default
CreationTimestamp:      Mon, 24 Jan 2022 19:06:56 +0530
Labels:                 app=myapp-deploy
                        tier=frontend
Annotations:            deployment.kubernetes.io/revision: 1
                        kubernetes.io/change-cause: kubectl apply --filename=sample.yaml --record=true
Selector:               app=myapp-deploy
Replicas:               6 desired | 6 updated | 6 total | 6 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=myapp-deploy
  Containers:
   nginx:
    Image:        nginx
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   deploy-6b9bcc5f46 (6/6 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  112s  deployment-controller  Scaled up replica set deploy-6b9bcc5f46 to 6
machinename@machinename kubernetes warmup % kubectl edit -f sample.yaml --record
Flag --record has been deprecated, --record will be removed in the future
deployment.apps/deploy edited
machinename@machinename kubernetes warmup % kubectl rollout status deployment.apps/deploy
Waiting for deployment "deploy" rollout to finish: 3 out of 6 new replicas have been updated...
Waiting for deployment "deploy" rollout to finish: 3 out of 6 new replicas have been updated...
Waiting for deployment "deploy" rollout to finish: 3 out of 6 new replicas have been updated...
Waiting for deployment "deploy" rollout to finish: 3 out of 6 new replicas have been updated...
Waiting for deployment "deploy" rollout to finish: 3 out of 6 new replicas have been updated...
```

```
Waiting for deployment "deploy" rollout to finish: 4 out of 6 new replicas have been updated...
Waiting for deployment "deploy" rollout to finish: 4 out of 6 new replicas have been updated...
Waiting for deployment "deploy" rollout to finish: 4 out of 6 new replicas have been updated...
Waiting for deployment "deploy" rollout to finish: 4 out of 6 new replicas have been updated...
Waiting for deployment "deploy" rollout to finish: 5 out of 6 new replicas have been updated...
Waiting for deployment "deploy" rollout to finish: 5 out of 6 new replicas have been updated...
Waiting for deployment "deploy" rollout to finish: 5 out of 6 new replicas have been updated...
Waiting for deployment "deploy" rollout to finish: 5 out of 6 new replicas have been updated...
Waiting for deployment "deploy" rollout to finish: 2 old replicas are pending termination...
Waiting for deployment "deploy" rollout to finish: 2 old replicas are pending termination...
Waiting for deployment "deploy" rollout to finish: 2 old replicas are pending termination...
Waiting for deployment "deploy" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "deploy" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "deploy" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "deploy" rollout to finish: 5 of 6 updated replicas are available...
deployment "deploy" successfully rolled out
machinename@machinename kubernetes warmup % kubectl describe deployment deploy
Name:                   deploy
Namespace:              default
CreationTimestamp:      Mon, 24 Jan 2022 19:06:56 +0530
Labels:                 app=myapp-deploy
                        tier=frontend
Annotations:            deployment.kubernetes.io/revision: 2
                        kubernetes.io/change-cause: kubectl edit --filename=sample.yaml --record=true
Selector:               app=myapp-deploy
Replicas:               6 desired | 6 updated | 6 total | 6 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=myapp-deploy
  Containers:
   nginx:
    Image:        nginx:1.18
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   deploy-c8799765b (6/6 replicas created)
Events:
  Type    Reason             Age              From                   Message
  ----    ------             ----             ----                   -------
  Normal  ScalingReplicaSet  3m53s            deployment-controller  Scaled up replica set deploy-6b9bcc5f46
to 6
  Normal  ScalingReplicaSet  62s              deployment-controller  Scaled up replica set deploy-c8799765b to
2
```

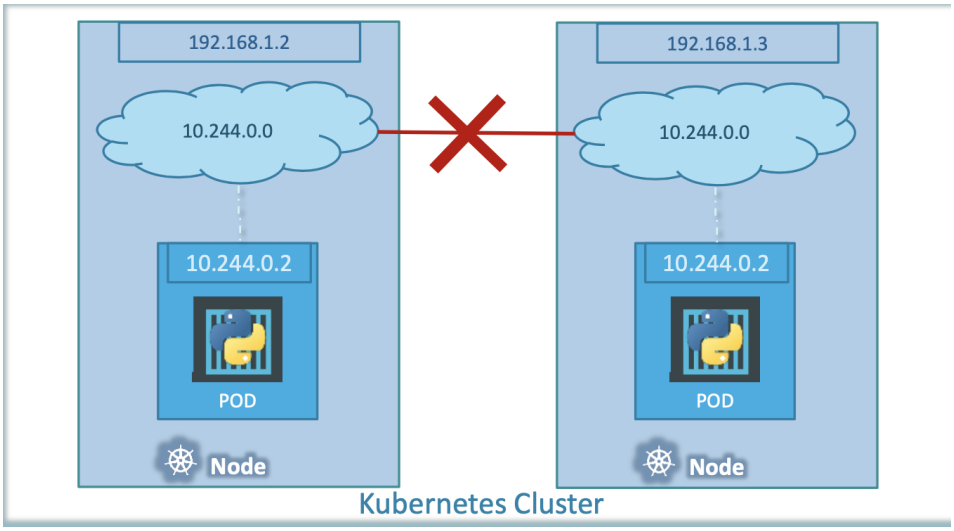
```
  Normal  ScalingReplicaSet  61s              deployment-controller  Scaled down replica set deploy-6b9bcc5f46
to 5
  Normal  ScalingReplicaSet  61s              deployment-controller  Scaled up replica set deploy-c8799765b to
3
  Normal  ScalingReplicaSet  38s              deployment-controller  Scaled down replica set deploy-6b9bcc5f46
to 4
  Normal  ScalingReplicaSet  38s              deployment-controller  Scaled up replica set deploy-c8799765b to
4
  Normal  ScalingReplicaSet  34s              deployment-controller  Scaled down replica set deploy-6b9bcc5f46
to 3
  Normal  ScalingReplicaSet  34s              deployment-controller  Scaled up replica set deploy-c8799765b to
5
  Normal  ScalingReplicaSet  30s              deployment-controller  Scaled down replica set deploy-6b9bcc5f46
to 2
  Normal  ScalingReplicaSet  21s (x3 over 30s)  deployment-controller  (combined from similar events): Scaled
down replica set deploy-6b9bcc5f46 to 0
machinename@machinename kubernetes warmup % kubectl set image deployment deploy nginx=mginx1.19
--record=true
Flag --record has been deprecated, --record will be removed in the future
deployment.apps/deploy image updated
machinename@machinename kubernetes warmup % kubectl rollout status deployment.apps/deploy
Waiting for deployment "deploy" rollout to finish: 3 out of 6 new replicas have been updated...
^C%
machinename@machinename kubernetes warmup % kubectl rollout undo deployment.apps/deploy
deployment.apps/deploy rolled back
machinename@machinename kubernetes warmup % kubectl rollout history deployment.apps/deploy
deployment.apps/deploy
REVISION  CHANGE-CAUSE
1         kubectl apply --filename=sample.yaml --record=true
3         kubectl set image deployment deploy nginx=mginx1.19 --record=true
4         kubectl edit --filename=sample.yaml --record=true

machinename@machinename kubernetes warmup % kubectl edit -f sample.yaml
deployment.apps/deploy edited
machinename@machinename kubernetes warmup % kubectl rollout status deployment.apps/deploy
//updating a wrong version of image
Waiting for deployment "deploy" rollout to finish: 3 out of 6 new replicas have been updated...
^C%           //gets stuck as its trying to download a image version that's wrong
machinename@machinename kubernetes warmup % kubectl get deployment deploy
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
deploy   5/6     3            5           12m
machinename@machinename kubernetes warmup % kubectl get pods
NAME                      READY   STATUS             RESTARTS   AGE
deploy-6864b458b-hvn9t    0/1     ImagePullBackOff   0          32s
deploy-6864b458b-pnnmz    0/1     ImagePullBackOff   0          32s
deploy-6864b458b-r2xcj    0/1     ImagePullBackOff   0          32s
deploy-c8799765b-bjwxj    1/1     Running            0          9m45s
deploy-c8799765b-j6qwx    1/1     Running            0          9m45s
deploy-c8799765b-lfhx9    1/1     Running            0          9m22s
deploy-c8799765b-lhnqt    1/1     Running            0          9m45s
deploy-c8799765b-s5lr7    1/1     Running            0          9m14s
frontend-5b262            1/1     Running            0          114m
frontend-rftx7            1/1     Running            0          114m
frontend-ztpt7            1/1     Running            0          3h7m
```

```
myapp-rc-bpc2l          1/1     Running           0          5h12m
myapp-rc-c692r          1/1     Running           0          5h12m
myapp-rc-fxz5p          1/1     Running           0          5h12m
machinename@machinename kubernetes warmup % kubectl rollout history deployment.apps/deploy
deployment.apps/deploy
REVISION  CHANGE-CAUSE
1         kubectl apply --filename=sample.yaml --record=true
3         kubectl set image deployment deploy nginx=mginx1.19 --record=true
4         kubectl edit --filename=sample.yaml --record=true
5         kubectl edit --filename=sample.yaml --record=true

machinename@machinename kubernetes warmup % kubectl rollout undo deployment.apps/deploy //rolled
back the worng image changes
deployment.apps/deploy rolled back
machinename@machinename kubernetes warmup % kubectl rollout history deployment.apps/deploy
deployment.apps/deploy
REVISION  CHANGE-CAUSE
1         kubectl apply --filename=sample.yaml --record=true
3         kubectl set image deployment deploy nginx=mginx1.19 --record=true
5         kubectl edit --filename=sample.yaml --record=true
6         kubectl edit --filename=sample.yaml --record=true

machinename@machinename kubernetes warmup % kubectl get pods
NAME                    READY   STATUS              RESTARTS   AGE
deploy-c8799765b-9brvl   0/1     ContainerCreating   0          9s
deploy-c8799765b-bjwxj   1/1     Running             0          10m
deploy-c8799765b-j6qwx   1/1     Running             0          10m
deploy-c8799765b-lfhx9   1/1     Running             0          10m
deploy-c8799765b-lhnqt   1/1     Running             0          10m
deploy-c8799765b-s5lr7   1/1     Running             0          9m58s
frontend-5b262          1/1     Running           0          115m
frontend-rftx7          1/1     Running           0          115m
frontend-ztpt7          1/1     Running           0          3h8m
myapp-rc-bpc2l          1/1     Running           0          5h13m
myapp-rc-c692r          1/1     Running           0          5h13m
myapp-rc-fxz5p          1/1     Running           0          5h13m
machinename@machinename kubernetes warmup % kubectl describe deployment deploy
Name:                   deploy
Namespace:              default
CreationTimestamp:      Mon, 24 Jan 2022 19:06:56 +0530
Labels:                 app=myapp-deploy
                        tier=frontend
Annotations:            deployment.kubernetes.io/revision: 6
                        kubernetes.io/change-cause: kubectl edit --filename=sample.yaml --record=true
Selector:               app=myapp-deploy
Replicas:               6 desired | 6 updated | 6 total | 6 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=myapp-deploy
  Containers:
   nginx:
    Image:        nginx:1.18
```

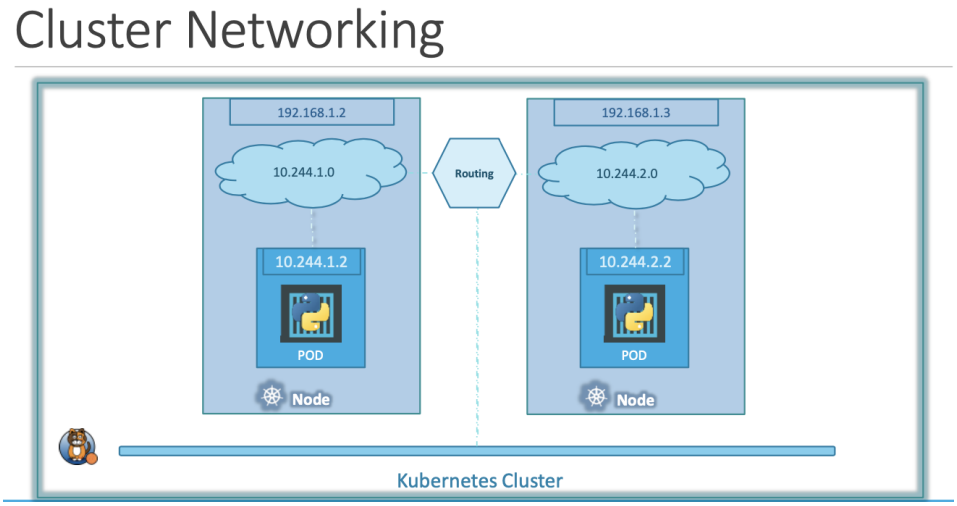
```
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type          Status  Reason
  ----          ------  ------
  Available     True    MinimumReplicasAvailable
  Progressing   True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   deploy-c8799765b (6/6 replicas created)
Events:
  Type    Reason             Age              From                   Message
  ----    ------             ----             ----                   -------
  Normal  ScalingReplicaSet  13m              deployment-controller  Scaled up replica set deploy-6b9bcc5f46
to 6
  Normal  ScalingReplicaSet  11m              deployment-controller  Scaled up replica set deploy-c8799765b to
2
  Normal  ScalingReplicaSet  11m              deployment-controller  Scaled down replica set deploy-
6b9bcc5f46 to 5
  Normal  ScalingReplicaSet  11m              deployment-controller  Scaled up replica set deploy-c8799765b to
3
  Normal  ScalingReplicaSet  10m              deployment-controller  Scaled down replica set deploy-
6b9bcc5f46 to 4
  Normal  ScalingReplicaSet  10m              deployment-controller  Scaled up replica set deploy-c8799765b to
4
  Normal  ScalingReplicaSet  10m              deployment-controller  Scaled down replica set deploy-
6b9bcc5f46 to 3
  Normal  ScalingReplicaSet  10m              deployment-controller  Scaled up replica set deploy-c8799765b to
5
  Normal  ScalingReplicaSet  10m              deployment-controller  Scaled down replica set deploy-
6b9bcc5f46 to 2
  Normal  ScalingReplicaSet  110s             deployment-controller  Scaled down replica set deploy-
c8799765b to 5
  Normal  ScalingReplicaSet  43s (x11 over 10m)  deployment-controller  (combined from similar events):
Scaled down replica set deploy-6864b458b to 0
  Normal  ScalingReplicaSet  43s              deployment-controller  Scaled up replica set deploy-c8799765b to
6
machinename@machinename kubernetes warmup %
```

NETWORKING IN KUBERNETES

- A node has an ip address attached to it.
- Like node, each pod within node also has an internal ip allotted to it.
- When a Kubernetes cluster is created at the beginning, it has a network set up. All the pods receive ip address from that network. And the pods internally communicate via this ip address that's allotted to each of them. But using these might not be a good idea as IPs change when a pod is destroyed and recreated again.
  If we consider 2 pods in different nodes, chances might be that both pods bear same IP address. Hence using these internal pod ip addresses can raise conflict.



- Kubernetes expects us to setup networks to meet certain fundamental criteria so that all nodes and pods/containers can communicate with each other. And we can set it up using VMware nsx, Kalikow or other solutions.
- With the help of these, we can make sure that each pod has a separate ip address assigned to it.



SERVICES

Kubernetes Services enable communication between various components within and outside of the application. Kubernetes Services helps us connect applications together with other applications or users. For example, our application has groups of PODs running various sections, such as a group for serving front-end load to users, another group running back-end processes, and a third group connecting to an external data source. It is Services that enable connectivity between these groups of PODs. Services enable the front-end application to be made available to users, it helps communication between back-end and front-end PODs, and

helps in establishing connectivity to an external data source. Thus services enable loose coupling between microservices in our application.

It's an abstract way to expose an application running on a set of Pods as a network service. With Kubernetes you don't need to modify your application to use an unfamiliar service discovery mechanism. Kubernetes gives Pods their own IP addresses and a single DNS name for a set of Pods, and can load-balance across them.

MOTIVATION

Kubernetes Pods are created and destroyed to match the state of your cluster. Pods are non-permanent resources. If you use a Deployment to run your app, it can create and destroy Pods dynamically.

Each Pod gets its own IP address, however in a Deployment, the set of Pods running in one moment in time could be different from the set of Pods running that application a moment later. The traffic might scale up or down the pods of your application.

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*.

A Service in Kubernetes is a REST object, similar to a Pod. Like all of the REST objects, you can POST a Service definition to the API server to create a new instance.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

- This specification creates a new Service object named "my-service", which targets TCP port 9376 on any Pod with the app=MyApp label.
- Kubernetes assigns this Service an IP address (sometimes called the "cluster IP"), which is used by the Service proxies (see Virtual IPs and service proxies below).
- The controller for the Service selector continuously scans for Pods that match its selector, "MyApp" and then POSTs any updates to an Endpoint object also named "my-service".
- The Targetport gives a lot of flexibility for deploying and evolving services.That is, for instance, the next set of services created can be exposed in a different port number without breaking other components.

SERVICES WITHOUT SELECTORS

There are instances or situations where the application you are building is unstable and going through a batch of testing.So the database you are required to connect might be in your cluster or outside. You might be expected to connect to a service which is residing in a complete different namespace or a cluster.

In such cases, you can ignore giving the selectors.Instead you can do a manual set up of connecting the service to an endpoint object.

That is :

You create a service without selector

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

The corresponding endpoint object is not created automatically. Hence, we have to set it up manually to the network address and port where its running.

That is:

```yaml
apiVersion: v1
kind: Endpoints
metadata:
  name: my-service
subsets:
  - addresses:
      - ip: 192.0.2.42
    ports:
      - port: 9376
```

In the example above, traffic is routed to the single endpoint defined in the YAML 192.0.2.42:9376 (TCP).

OVER CAPACITY ENDPOINTS:

If the Endpoints resource has more than 1000 endpoints then, Kubernetes annotates that endpoints with “endpoints.kubernetes.io/over-capacity: truncated”

ENDPOINT SLICES

EndpointSlices are alternatives to endpoint objects. This allow distributing network endpoints over multiple endpoints across multiple resources.

By default its considered full with 100 endpoints. Once it reaches that point, additional endpointSlices will be created.

APPLICATION PROTOCOL:

It provides a way to specify an application protocol for each service port. The value of this field is mirrored by corresponding endpoints/endpointSlice objects.

VIRTUAL IPs and service proxies

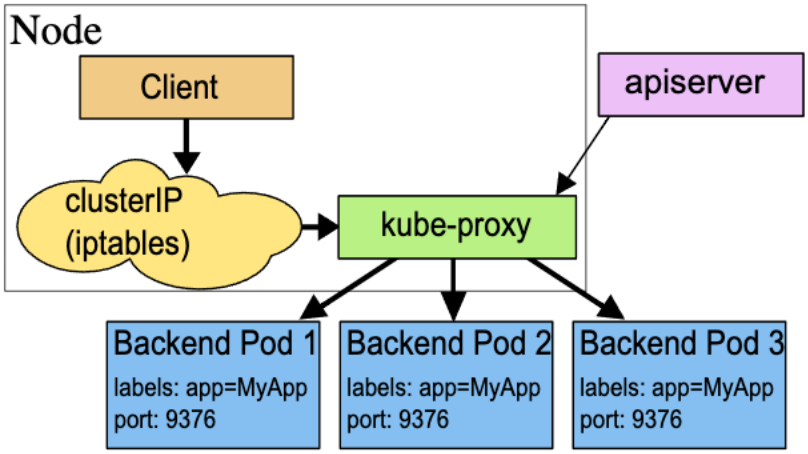
Kube-proxy

You should note that, when running kube-proxy, kernel level rules may be modified (for example, iptables rules might get created), which won't get cleaned up, in some cases until you reboot. Thus, running kube-proxy is something that should only be done by an administrator which understands the consequences of having a low level, privileged network proxying service on a computer.

CONFIGURATION

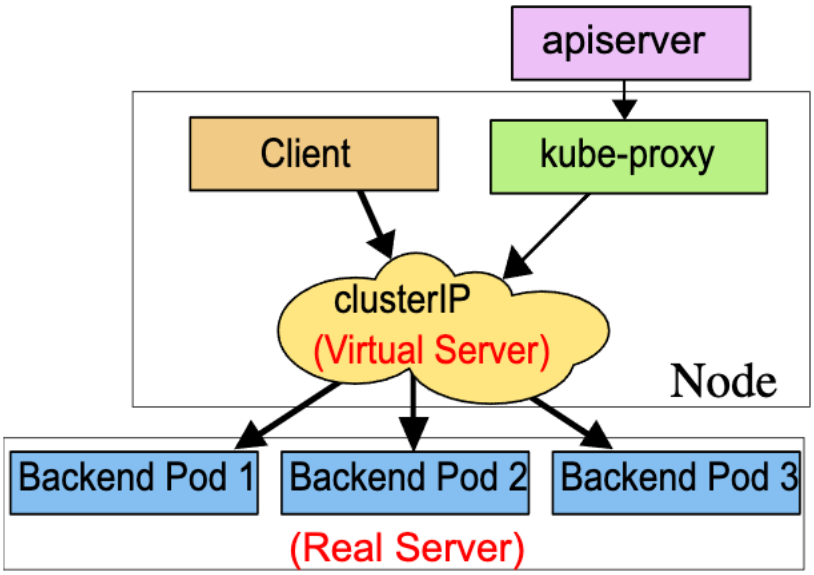
The kube-proxy configuration is done via config-map and this deprecates the behaviour for almost all the flags for kube-proxy.
It doesn’t allow live reloading of configuration if or when required.
The map parameters cannot all be validated at the startup.

USER SPACE PROXY MODE/IPTABLES PROXY MODE



- For each Service it opens a port (randomly chosen) on the local node. Any connections to this "proxy port" are proxied to one of the Service's backend Pods
- In Legacy: For each Service, it installs iptables rules, which capture traffic to the Service's clusterIP and port, and redirect that traffic to one of the Service's backend sets. For each Endpoint object, it installs iptables rules which select a backend Pod via round robin algorithm.
- In IPTables: For each Service, it installs iptables rules, which capture traffic to the Service's clusterIP and port, and redirect that traffic to one of the Service's backend sets. For each Endpoint object, it installs iptables rules which select a backend Pod randomly.

IPVS PROXY MODE



- In these proxy models, the traffic bound for the Service's IP: Port is proxied to an appropriate backend without the clients knowing anything about Kubernetes or Services or Pods.
- If you want to make sure that connections from a particular client are passed to the same Pod each time, you can select the session affinity based on the client's IP addresses by setting service. spec. sessionAffinity to "ClientIP" (the default is "None"). You can also set the maximum session sticky time by setting service. spec. sessionAffinityConfig. clientIP. timeoutSeconds appropriately.

MULTI-PORT SERVICES

For some services, you need more than one port.This can be configured in Kubernetes as below:

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 9376
    - name: https
      protocol: TCP
      port: 443
      targetPort: 9377
```

CHOOSING OWN IP ADDRESS

You can specify your own clusterIP address as part of Service creation. In case of an existing DNS, you can set it with .spec.clusterIP field

TRAFFIC POLICIES

External

You can specify how to control traffic from external sources with .spec.externalTrafficPolicy field.

The possible values for this are:

1. Cluster: This is to route external traffic to all Ready endpoints.
2. Local: This is to route only to ready node-local endpoints. If the policy is local and there are no node-local endpoints, kube-proxy won't allow the traffic to relevant services.

Internal

Similar to External policy, you can set the internal traffic policy with spec.internaltrafficPolicy. Here too, the possible values are Cluster and Local and act similarly.

DISCOVERING SERVICES

Environment variables:

When a Pod is run on a Node, the kubelet adds a set of environment variables for each active Service. It supports both Docker links compatible variables (see makeLinkVariables) and simpler {SERVICENAME}_SERVICE_HOST and {SERVICENAME}_SERVICE_PORT variables, where the Service name is upper-cased and dashes are converted to underscores.

For example, the Service redis-master which exposes TCP port 6379 and has been allocated cluster IP address 10.0.0.11, produces the following environment variables:

```
REDIS_MASTER_SERVICE_HOST=10.0.0.11
REDIS_MASTER_SERVICE_PORT=6379
REDIS_MASTER_PORT=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP_PROTO=tcp
REDIS_MASTER_PORT_6379_TCP_PORT=6379
REDIS_MASTER_PORT_6379_TCP_ADDR=10.0.0.11
```

DNS

You can (and almost always should) set up a DNS service for your Kubernetes cluster.

A cluster-aware DNS server, such as CoreDNS, watches the Kubernetes API for new Services and creates a set of DNS records for each one. If DNS has been enabled throughout your cluster then all Pods should automatically be able to resolve Services by their DNS name. For example,

1. if you have a Service called my-service in a Kubernetes namespace my-ns, the control plane and the DNS Service acting together create a DNS record for my-service.my-ns.Pods in the my-ns namespace should be able to find the service by doing a name lookup for my-service (my-service.my-ns would also work).
2. Pods in other namespaces must qualify the name as my-service.my-ns. These names will resolve to the cluster IP assigned for the Service.
3. Kubernetes also supports DNS SRV (Service) records for named ports. If the my-service.my-ns Service has a port named http with the protocol set to TCP, you can do a DNS SRV query for _http._tcp.my-service.my-ns to discover the port number for http, as well as the IP address.

Headless Services

Sometimes you don't need load-balancing and a single Service IP. In this case, you can create what are termed "headless" Services, by explicitly specifying "None" for the cluster IP (.spec.clusterIP).
You can use a headless Service to interface with other service discovery mechanisms, without being tied to Kubernetes' implementation.

For headless Services, a cluster IP is not allocated, kube-proxy does not handle these Services, and there is no load balancing or proxying done by the platform for them. How DNS is automatically configured depends on whether the Service

PUBLISHING SERVICES

Kubernetes ServiceTypes allow you to specify what kind of Service you want. The default is ClusterIP.

1. **ClusterIP**:
   Exposes the Service on a cluster-internal IP. Choosing this value makes the Service only reachable from within the cluster.

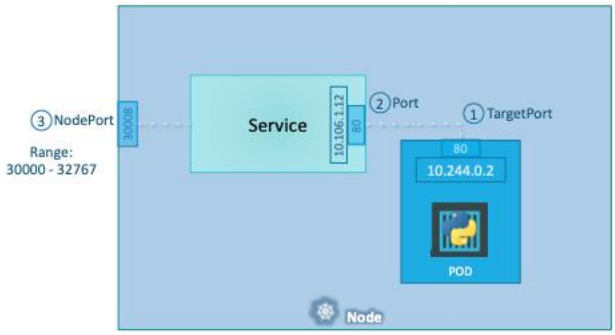
2. **NodePort**:
   Exposes the Service on each Node's IP at a static port (the NodePort).
   A ClusterIP Service, to which the NodePort Service routes, is automatically created.
   You'll be able to contact the NodePort Service, from outside the cluster, by requesting <NodeIP>:<NodePort>

3. **LoadBalancer**:
   Exposes the Service externally using a cloud provider's load balancer. NodePort and ClusterIP Services, to which the external load balancer routes, are automatically created.

4. **ExternalName**:
   Maps the Service to the contents of the externalName field (e.g. foo.bar.example.com), by returning a CNAME record with its value. No proxying of any kind is set up.

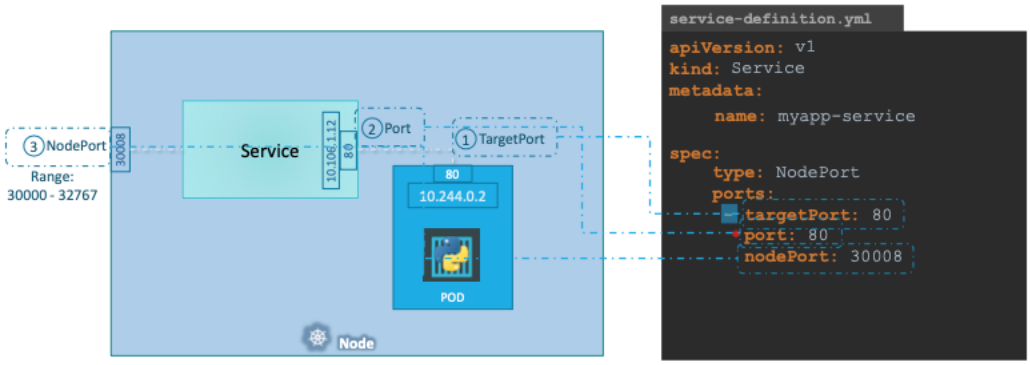
NODEPORT:

Service - NodePort



1. Let's take a closer look at the Service. If you look at it, there are 3 ports involved. The port on the POD where the actual web server is running is port 80. And it is referred to as the targetPort because that is where the service forwards the requests to.
2. The second port is the port on the service itself. It is simply referred to as the port. Remember, these terms are from the viewpoint of the service. The service is in fact like a virtual server inside the node. Inside the cluster it has its own IP address. And that IP address is called the Cluster-IP of the service.
3. And finally, we have the port on the Node itself which we use to access the web server externally. And that is known as the NodePort. As you can see it is 30008. That is because NodePorts can only be in a valid range which is from 30000 to 32767.

So the definition will look like:

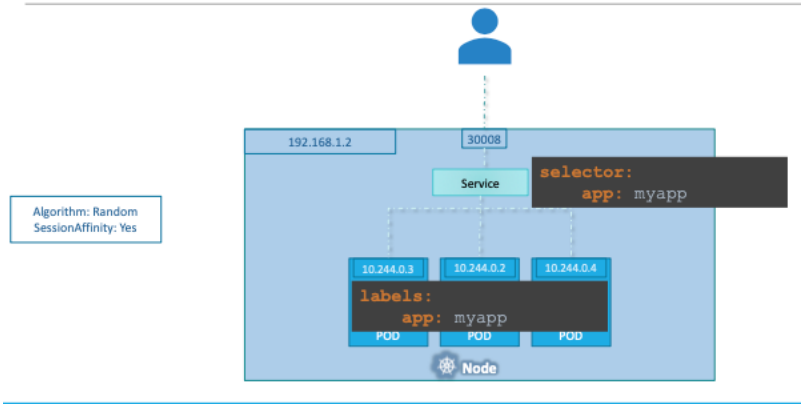



```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app: MyApp
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30007
```

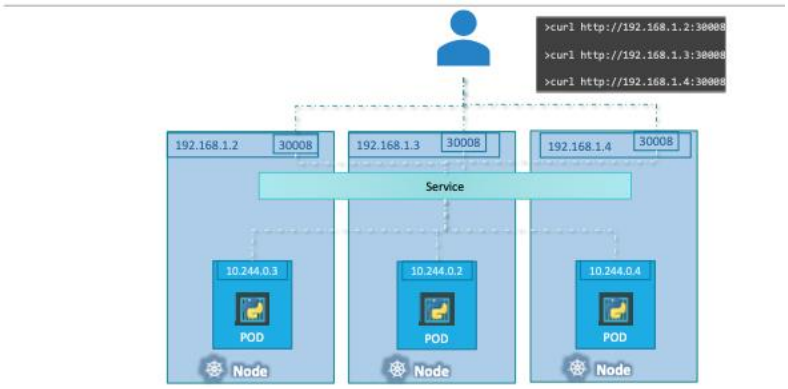
Demo:

```
machinename@machinename kubernetes warmup % kubectl create -f service-deinition.yaml
service/myapp-service created
machinename@machinename kubernetes warmup % kubectl get svc
NAME            TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP   10.96.0.1       <none>        443/TCP        8d
my-service      NodePort    10.100.215.84   <none>        80:30007/TCP   13m
myapp-service   NodePort    10.110.91.198   <none>        80:30004/TCP   7s
machinename@machinename kubernetes warmup % cat service-deinition.yaml
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: NodePort
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30004
  selector:
    name:  myapp-pod
machinename@machinename kubernetes warmup % minikube service myapp-service --url
http://10.110.91.198:30004
```

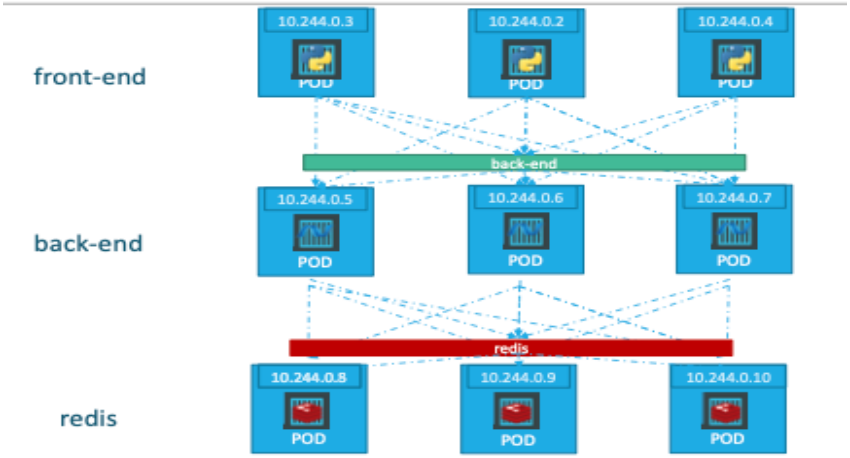
When the appis spread across different pods in a singlenode, a service is created across all such pods and these pods can be recognized with same selector label “myapp”



When the app is spanned across multiple nodes, the cluster creates a service that’s connected to all these nodes and this service acts as a loadbalancer.
Thus making application available on any such node with same port (and also with the help of the selector label as shown above). In our case : 30008



CLUSTERIP



For example,

A service created for the backend PODs will help group all the backend PODs together and provide a single interface for other PODs to access this service. The requests are forwarded to one of the PODs under the service randomly.

Similarly, create additional services for Redis and allow the backend PODs to access the redis system through this service. This enables us to easily and effectively deploy a microservices based application on kubernetes cluster.

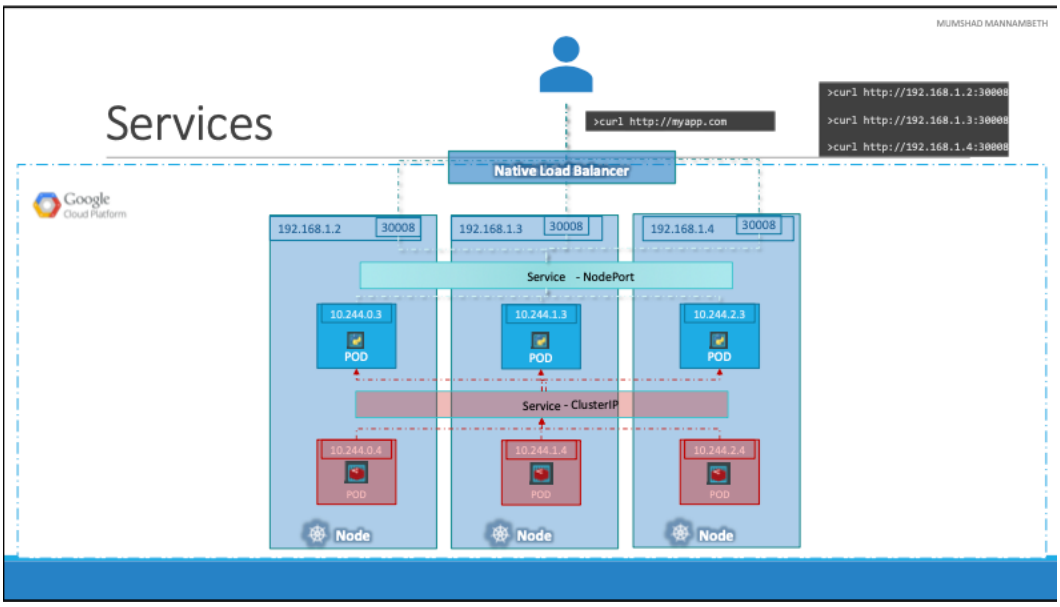
Each layer can now scale or move as required without impacting communication between the various services. Each service gets an IP and name assigned to it inside the cluster and that is the name that should be used by other PODs to access the service. This type of service is known as ClusterIP.

Demo:

```
machinename@machinename kubernetes warmup % kubectl create -f clusterIp.yaml
service/back-end created
machinename@machinename kubernetes warmup % kubectl get svc
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
back-end      ClusterIP   10.96.100.152   <none>        80/TCP         7s
kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP        8d
my-service    NodePort    10.100.215.84   <none>        80:30007/TCP   28m
myapp-service NodePort    10.110.91.198   <none>        80:30004/TCP   14m
machinename@machinename kubernetes warmup % cat clusterIp.yaml
apiVersion: v1
kind: Service
metadata:
```

```
  name:  back-end
spec:
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 80

  selector:
    name:   myapp-deploy%
machinename@machinename kubernetes warmup %
```

LOAD BALANCERS

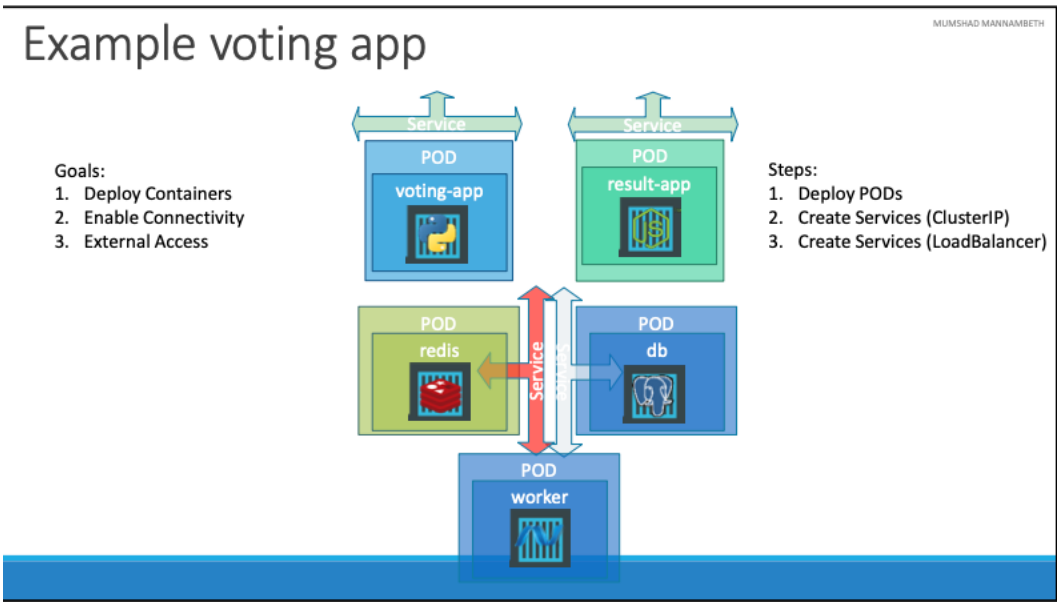


If we are on cloud like GCP, we can make use of the native load balancer provided there and can create an address through which we can connect to any ip:port in the cluster.

MICROSERVICES

Sample application – voting application

EndGoal is :



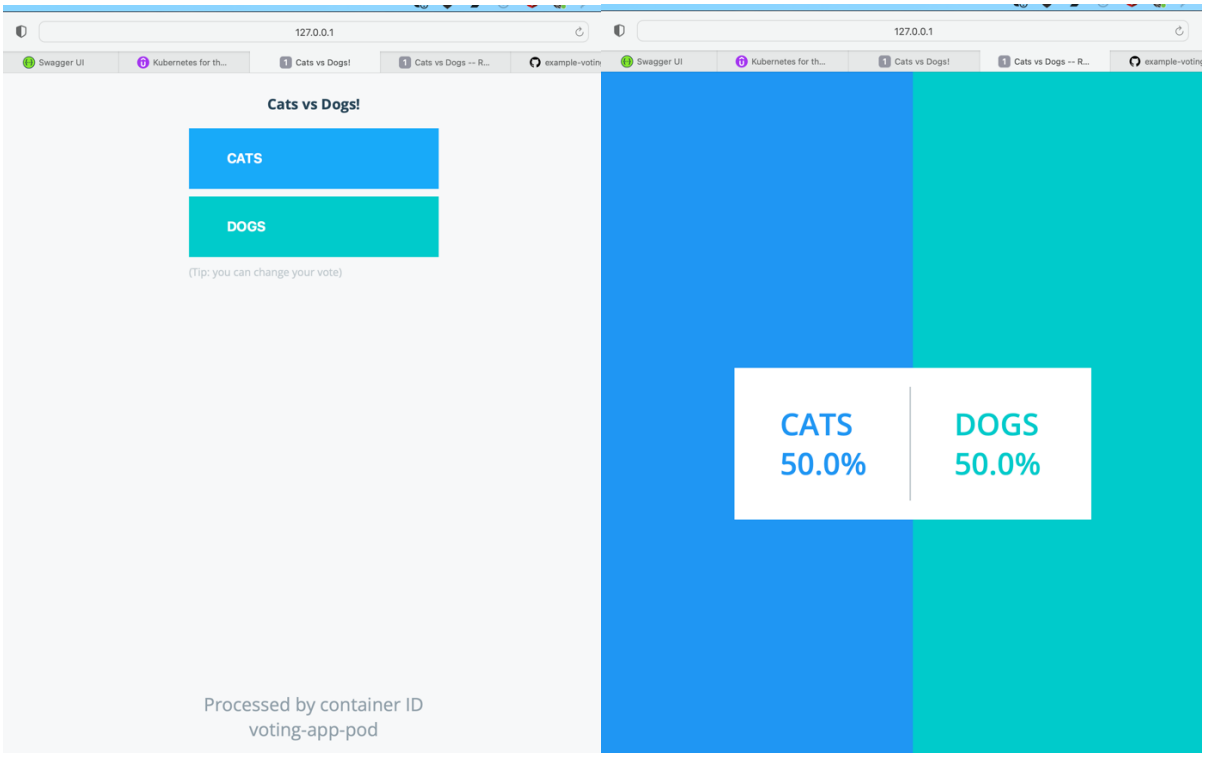
Demo:

```
machinename@machinename voting-app % kubectl apply voting-app-pod.yaml
error: must specify one of -f and -k
machinename@machinename voting-app % kubectl apply -f voting-app-pod.yaml
pod/voting-app-pod created
machinename@machinename voting-app % kubectl apply -f result-app-pod.yaml
pod/result-app-pod created
machinename@machinename voting-app % kubectl apply -f redis-pod.yaml
pod/redis-pod created
machinename@machinename voting-app % kubectl apply -f postgres-pod.yaml
pod/postgres-pod created
machinename@machinename voting-app % kubectl apply -f worker-pod.yaml
pod/worker-app-pod created
machinename@machinename voting-app % kubectl create -f voting-app-service.yaml
service/voting-service created
machinename@machinename voting-app % kubectl create -f result-app-service.yaml
```

```
service/result-service created
machinename@machinename voting-app % kubectl create -f redis-service.yaml
service/redis created
machinename@machinename voting-app % kubectl create -f postgres-service.yaml
service/db created
machinename@machinename voting-app % kubectl get svc
NAME             TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
db               ClusterIP   10.98.44.160     <none>        5432/TCP       7s
kubernetes       ClusterIP   10.96.0.1        <none>        443/TCP        9d
redis            ClusterIP   10.101.0.168     <none>        6379/TCP       16s
result-service   NodePort    10.101.89.75     <none>        80:30005/TCP   27s
voting-service   NodePort    10.111.185.151   <none>        80:30004/TCP   33s
machinename@machinename voting-app % minikube service voting-service
|-----------|----------------|-------------|-------------------------|
| NAMESPACE |      NAME      | TARGET PORT |           URL           |
|-----------|----------------|-------------|-------------------------|
| default   | voting-service |          80 | http://192.168.49.2:30004 |
|-----------|----------------|-------------|-------------------------|
🏃  Starting tunnel for service voting-service.
|-----------|----------------|-------------|------------------------|
| NAMESPACE |      NAME      | TARGET PORT |          URL           |
|-----------|----------------|-------------|------------------------|
| default   | voting-service |             | http://127.0.0.1:59492 |
|-----------|----------------|-------------|------------------------|
🎉  Opening service default/voting-service in default browser...
❗  Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
^C✋  Stopping tunnel for service voting-service.

❌  Exiting due to SVC_TUNNEL_STOP: stopping ssh tunnel: os: process already finished

╭──────────────────────────────────────────────────────────────────────────────────────────────╮
│                                                                                              │
│    😿  If the above advice does not help, please let us know:                                │
│    👉  https://github.com/kubernetes/minikube/issues/new/choose                             │
│                                                                                              │
│    Please run `minikube logs --file=logs.txt` and attach logs.txt to the GitHub issue.       │
│    Please also attach the following file to the GitHub issue:                                │
│    -
/var/folders/ll/nbr0044n48j59h4gl8lcwqs00000gn/T/minikube_service_cd07c5d9febe649e52b17ca2fa126b94db464
882_0.log    │
│                                                                                              │
╰──────────────────────────────────────────────────────────────────────────────────────────────╯

machinename@machinename voting-app % minikube service result-service
|-----------|----------------|-------------|-------------------------|
| NAMESPACE |      NAME      | TARGET PORT |           URL           |
|-----------|----------------|-------------|-------------------------|
| default   | result-service |          80 | http://192.168.49.2:30005 |
|-----------|----------------|-------------|-------------------------|
🏃  Starting tunnel for service result-service.
|-----------|----------------|-------------|------------------------|
| NAMESPACE |      NAME      | TARGET PORT |          URL           |
|-----------|----------------|-------------|------------------------|
```

```
| default   | result-service |             | http://127.0.0.1:59614 |
|-----------|----------------|-------------|------------------------|
🎉  Opening service default/result-service in default browser...
❗  Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
^C✋  Stopping tunnel for service result-service.

❌  Exiting due to SVC_TUNNEL_STOP: stopping ssh tunnel: os: process already finished

╭──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────╮
│                                                                                                                              │
│    😿  If the above advice does not help, please let us know:                                                                │
│    👉  https://github.com/kubernetes/minikube/issues/new/choose                                                             │
│                                                                                                                              │
│    Please run `minikube logs --file=logs.txt` and attach logs.txt to the GitHub issue.                                       │
│    Please also attach the following file to the GitHub issue:                                                                │
│    - /var/folders/ll/nbr0044n48j59h4gl8lcwqs00000gn/T/minikube_service_44e8d66436f50ffb5cc7f5caf98e1e9abac2c931_0.log        │
│                                                                                                                              │
╰──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────╯

machinename@machinename voting-app % kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
postgres-pod   1/1     Running   0          22m
redis-pod      1/1     Running   0          22m
result-app-pod 1/1     Running   0          22m
voting-app-pod 1/1     Running   0          22m
worker-app-pod 1/1     Running   0          6s
machinename@machinename voting-app % kubectl get svc
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
db             ClusterIP   10.98.44.160    <none>        5432/TCP       7s
kubernetes     ClusterIP   10.96.0.1       <none>        443/TCP        9d
redis          ClusterIP   10.101.0.168    <none>        6379/TCP       16s
result-service NodePort    10.101.89.75    <none>        80:30005/TCP   27s
voting-service NodePort    10.111.185.151  <none>        80:30004/TCP   33s
```

The UI will look as below:



Files are as follows:

Voting pod	apiVersion: v1
	kind: Pod
	metadata:
	  name: voting-app-pod
	  labels:
	  name: voting-app-pod
	  app: demo-voting-app
	spec:
	  containers:
	  - name: voting-app
	  image: kodekloud/examplevotingapp_vote:v1
	  ports:
	  - containerPort: 80

Voting service	apiVersion: v1
	kind: Service
	metadata:
	  name: voting-service
	  labels:
	  name: voting-service

	<pre>app: demo-voting-app
	spec:
	type: NodePort
	ports:
	- port: 80
	targetPort: 80
	nodePort: 30004
	selector:
	name: voting-app-pod
	app: demo-voting-app</pre>

Redis pod	<pre>apiVersion: v1
	kind: Pod
	metadata:
	name: redis-pod
	labels:
	name: redis-pod
	app: demo-voting-app
	spec:
	containers:
	- name: redis
	image: redis
	ports:
	- containerPort: 6379</pre>

Redis service	<pre>apiVersion: v1
	kind: Service
	metadata:
	name: redis
	labels:
	name: redis-service
	app: demo-voting-app
	spec:
	ports:
	- port: 6379
	targetPort: 6379
	selector:
	name: redis-pod
	app: demo-voting-app</pre>

Postgres pod	<pre>apiVersion: v1
	kind: Pod
	metadata:
	name: postgres-pod
	labels:
	name: postgres-pod
	app: demo-voting-app
	spec:
	containers:
	- name: postgres
	image: postgres
	ports:
	- containerPort: 5432
	env:
	- name: POSTGRES_USER
	value: "postgres"
	- name: POSTGRES_PASSWORD
	value: "postgres"</pre>

Postgres service	<pre>apiVersion: v1</pre>

	kind: Service
	metadata:
	  name: db
	  labels:
	  name: postgres-service
	  app: demo-voting-app
	spec:
	  ports:
	  - port: 5432
	  targetPort: 5432
	  selector:
	  name: postgres-pod
	  app: demo-voting-app

Worker pod	apiVersion: v1
	kind: Pod
	metadata:
	  name: worker-app-pod
	  labels:
	  name: worker-app-pod
	  app: demo-voting-app
	spec:
	  containers:
	  - name: worker-app
	  image: dockersamples/examplevotingapp_worker:latest

Result pod	apiVersion: v1
	kind: Pod
	metadata:
	  name: result-app-pod
	  labels:
	  name: result-app-pod
	  app: demo-voting-app
	spec:
	  containers:
	  - name: result-app
	  image: kodekloud/examplevotingapp_result:v1
	  ports:
	  - containerPort: 80

Result service	apiVersion: v1
	kind: Service
	metadata:
	  name: result-service
	  labels:
	  name: result-service
	  app: demo-voting-app
	spec:
	  type: NodePort
	  ports:
	  - port: 80
	  targetPort: 80
	  nodePort: 30005
	  selector:
	  name: result-app-pod
	  app: demo-voting-app

USING DEPLOYMENT AS WORKLOAD RESOURCE

Demo:

```
machinename@machinename voting-app % kubectl delete pods postgres-pod redis-pod voting-app-pod result-app-pod worker-app-pod
pod "postgres-pod" deleted
pod "redis-pod" deleted
pod "voting-app-pod" deleted
pod "result-app-pod" deleted
pod "worker-app-pod" deleted
machinename@machinename voting-app % kubectl get pods
No resources found in default namespace.
machinename@machinename voting-app % kubectl get svc
NAME             TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
db               ClusterIP   10.98.44.160     <none>        5432/TCP       58m
kubernetes       ClusterIP   10.96.0.1        <none>        443/TCP        9d
redis            ClusterIP   10.101.0.168     <none>        6379/TCP       59m
result-service   NodePort    10.101.89.75     <none>        80:30005/TCP   59m
voting-service   NodePort    10.111.185.151   <none>        80:30004/TCP   59m
machinename@machinename voting-app % kubectl delete svc db redis result-service voting-service
service "db" deleted
service "redis" deleted
service "result-service" deleted
```

```
service "voting-service" deleted
machinename@machinename voting-app % kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   9d
machinename@machinename voting-app % kubectl create -f voting-app-pod.yaml
pod/voting-app-pod created
machinename@machinename voting-app % kubectl create -f redis-pod.yaml
deployment.apps/redis-deploy created
machinename@machinename voting-app % kubectl create -f postgres-pod.yaml
deployment.apps/postgres-deploy created
machinename@machinename voting-app % kubectl create -f worker-pod.yaml
deployment.apps/worker-app-deploy created
machinename@machinename voting-app % kubectl create -f result-app-pod.yaml
pod/result-app-pod created
machinename@machinename voting-app % kubectl get pods
NAME                              READY   STATUS             RESTARTS      AGE
postgres-deploy-6f787b796b-2dvft  1/1     Running            0             24s
redis-deploy-5d7988b4bb-89cbw     1/1     Running            0             32s
result-app-pod                    1/1     Running            0             7s
voting-app-pod                    1/1     Running            0             46s
worker-app-deploy-799b5fb489-w29l9   0/1     CrashLoopBackOff   1 (14s ago)   17s
machinename@machinename voting-app % kubectl create -f voting-app-service.yaml
service/voting-service created
machinename@machinename voting-app % kubectl create -f redis-service.yaml
service/redis created
machinename@machinename voting-app % kubectl create -f postgres-service.yaml
service/db created
machinename@machinename voting-app % kubectl create -f result-app-service.yaml
service/result-service created
machinename@machinename voting-app % kubectl get svc
NAME             TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
db               ClusterIP      10.98.147.167    <none>        5432/TCP       19s
kubernetes       ClusterIP      10.96.0.1        <none>        443/TCP        9d
redis            ClusterIP      10.96.222.61     <none>        6379/TCP       28s
result-service   LoadBalancer   10.106.143.180   <pending>     80:32432/TCP   5s
voting-service   LoadBalancer   10.108.79.182    <pending>     80:30282/TCP   40s
machinename@machinename voting-app % minikube service voting-service
|-----------|----------------|-------------|-------------------------|
| NAMESPACE |      NAME      | TARGET PORT |           URL           |
|-----------|----------------|-------------|-------------------------|
| default   | voting-service |          80 | http://192.168.49.2:30282 |
|-----------|----------------|-------------|-------------------------|
🏃  Starting tunnel for service voting-service.
|-----------|----------------|-------------|------------------------|
| NAMESPACE |      NAME      | TARGET PORT |          URL           |
|-----------|----------------|-------------|------------------------|
| default   | voting-service |             | http://127.0.0.1:55694 |
|-----------|----------------|-------------|------------------------|
🎉  Opening service default/voting-service in default browser...
❗  Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
^C✋  Stopping tunnel for service voting-service.

❌  Exiting due to SVC_TUNNEL_STOP: stopping ssh tunnel: os: process already finished
```

```
│                                                                                │
│    If the above advice does not help, please let us know:                      │
│    https://github.com/kubernetes/minikube/issues/new/choose                    │
│                                                                                │
│    Please run `minikube logs --file=logs.txt` and attach logs.txt to the GitHub issue.  │
│    Please also attach the following file to the GitHub issue:                  │
│    -
/var/folders/ll/nbr0044n48j59h4gl8lcwqs00000gn/T/minikube_service_cd07c5d9febe649e52b17ca2fa126b94db464
882_0.log    │
│                                                                                │
╰────────────────────────────────────────────────────────────────────────────────
────────────────────────────────────────────────────────────────────────────────
──────────────────────────╯

machinename@machinename voting-app % minikube service result-service
|-----------|---------------|-------------|-------------------------|
| NAMESPACE |      NAME     | TARGET PORT |           URL           |
|-----------|---------------|-------------|-------------------------|
| default   | result-service |          80 | http://192.168.49.2:32432 |
|-----------|---------------|-------------|-------------------------|
  Starting tunnel for service result-service.
|-----------|---------------|-------------|-----------------------|
| NAMESPACE |      NAME     | TARGET PORT |          URL          |
|-----------|---------------|-------------|-----------------------|
| default   | result-service |             | http://127.0.0.1:55810 |
|-----------|---------------|-------------|-----------------------|
  Opening service default/result-service in default browser...
❗ Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```

Files will be as below:

Voting deployment	apiVersion: apps/v1
	kind: Deployment
	metadata:
	  name: voting-app-deploy
	  labels:
	    name: voting-app-deploy
	    app: demo-voting-app
	spec:
	  replicas: 1
	  selector:
	    matchLabels:
	      name: voting-app-pod
	      app: demo-voting-app
	
	  template:
	    metadata:
	      name: voting-app-pod
	      labels:

	<pre>name: voting-app-pod
	app: demo-voting-app
	spec:
	containers:
	- name: voting-app
	image: kodekloud/examplevotingapp_vote:v1
	ports:
	- containerPort: 80</pre>

Redis deployment	<pre>apiVersion: apps/v1
	kind: Deployment
	metadata:
	name: redis-deploy
	labels:
	name: redis-deploy
	app: demo-voting-app
	spec:
	replicas: 1
	selector:
	matchLabels:
	name: redis-pod
	app: demo-voting-app
	
	template:
	metadata:
	name: redis-pod
	labels:
	name: redis-pod
	app: demo-voting-app
	spec:
	containers:
	- name: redis
	image: redis
	ports:
	- containerPort: 6379</pre>

Postgres deployment	<pre>apiVersion: apps/v1
	kind: Deployment
	metadata:
	name: postgres-deploy
	labels:
	name: postgres-deploy
	app: demo-voting-app
	spec:
	replicas: 1
	selector:
	matchLabels:
	name: postgres-pod
	app: demo-voting-app
	
	template:
	metadata:
	name: postgres-pod
	labels:
	name: postgres-pod
	app: demo-voting-app</pre>

	<pre>spec:
	  containers:
	  - name: postgres
	  image: postgres
	  ports:
	  - containerPort: 5432
	  env:
	  - name: POSTGRES_USER
	  value: "postgres"
	  - name: POSTGRES_PASSWORD
	  value: "postgres"</pre>

Worker deployment	<pre>apiVersion: apps/v1
	kind: Deployment
	metadata:
	  name: worker-app-deploy
	  labels:
	  name: worker-app-deploy
	  app: demo-voting-app
	spec:
	  replicas: 1
	  selector:
	  matchLabels:
	  name: worker-app-pod
	  app: demo-voting-app
	
	  template:
	  metadata:
	  name: worker-app-pod
	  labels:
	  name: worker-app-pod
	  app: demo-voting-app
	  spec:
	  containers:
	  - name: worker-app
	  image: kodekloud/examplevotingapp_worker:v1</pre>

Result deployment	<pre>apiVersion: apps/v1
	kind: Deployment
	metadata:
	  name: result-app-deploy
	  labels:
	  name: result-app-deploy
	  app: demo-voting-app
	spec:
	  replicas: 1
	  selector:
	  matchLabels:
	  name: result-app-pod
	  app: demo-voting-app
	
	  template:
	  metadata:
	  name: result-app-pod
	  labels:
	  name: result-app-pod</pre>

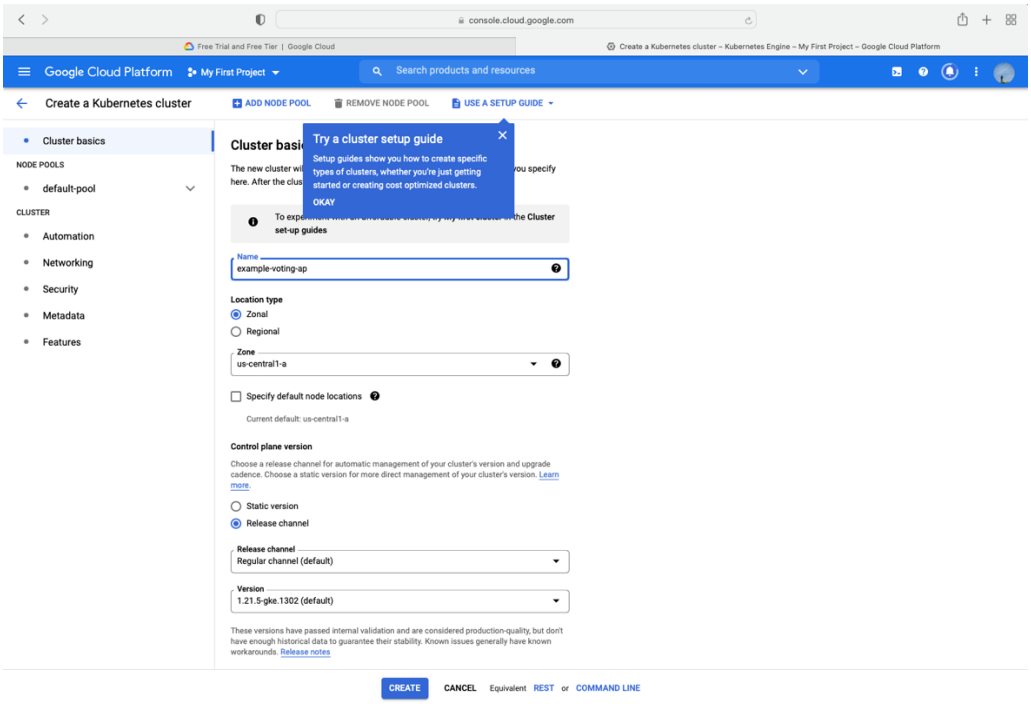
	app: demo-voting-app spec: containers: - name: result-app image: kodekloud/examplevotingapp_result:v1 ports: - containerPort: 80
Voting app loadbalancer	apiVersion: v1 kind: Service metadata: name: voting-service labels: name: voting-service app: demo-voting-app spec: type: LoadBalancer ports: - port: 80 targetPort: 80 selector: name: voting-app-pod app: demo-voting-app
Result app loadbalancer	apiVersion: v1 kind: Service metadata: name: result-service labels: name: result-service app: demo-voting-app spec: type: LoadBalancer ports: - port: 80 targetPort: 80 selector: name: result-app-pod app: demo-voting-app

KUBERNETES ON CLOUD

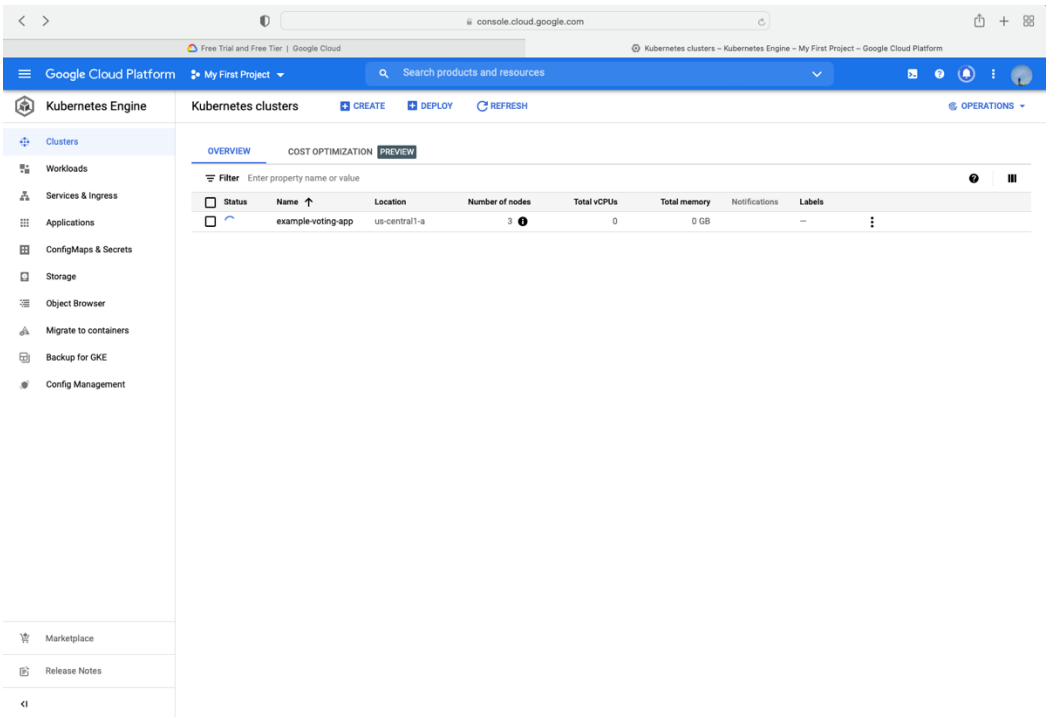
GCP/GKE:

Lets consider cloud for instance,

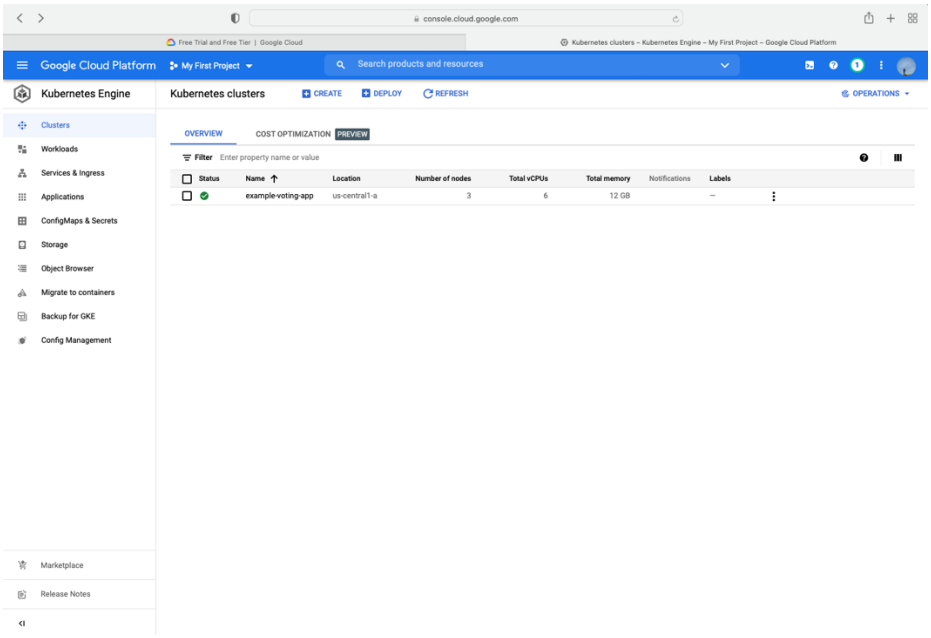
>> Open Google Cloud Console
>> Go to Kubernetes Engine
>> Create new cluster



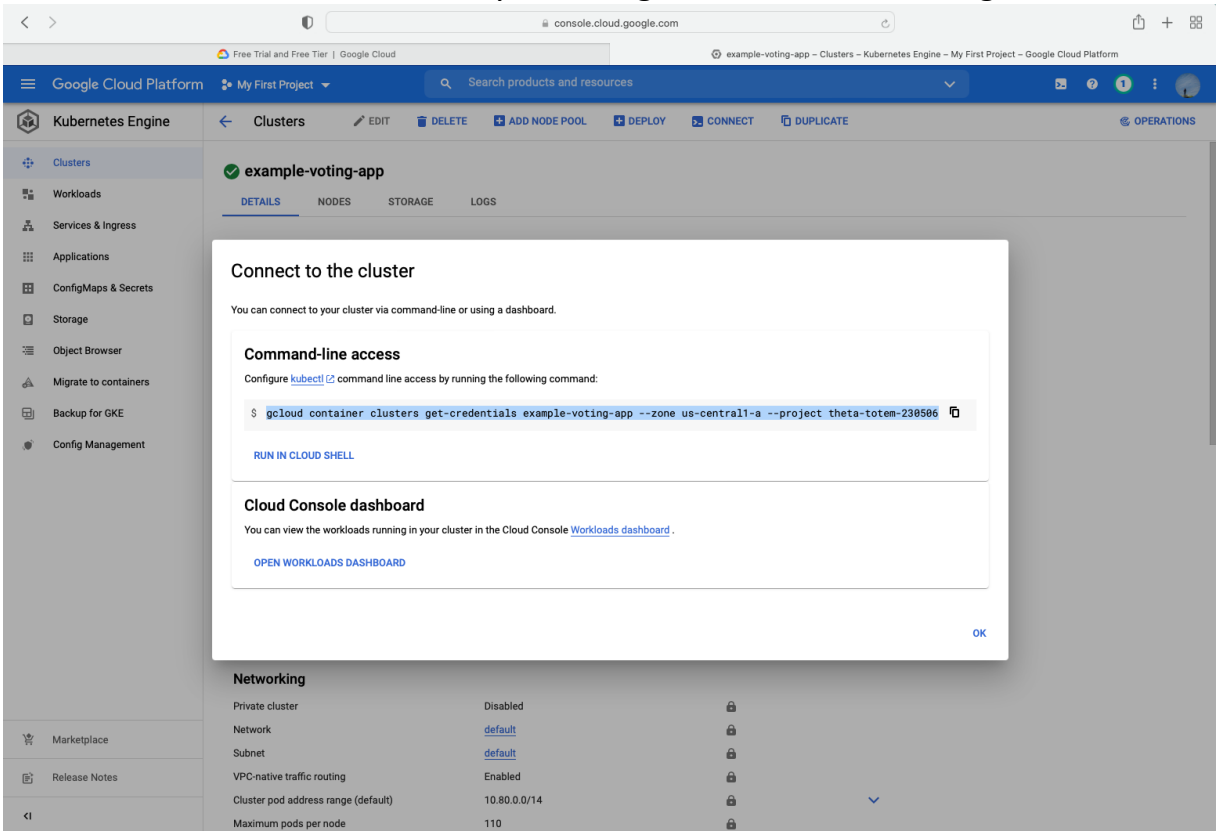
>> Select all defaults:



>> Wait till the status turns green



>> Then connect to the cluster by selecting the cluster and hitting on connect



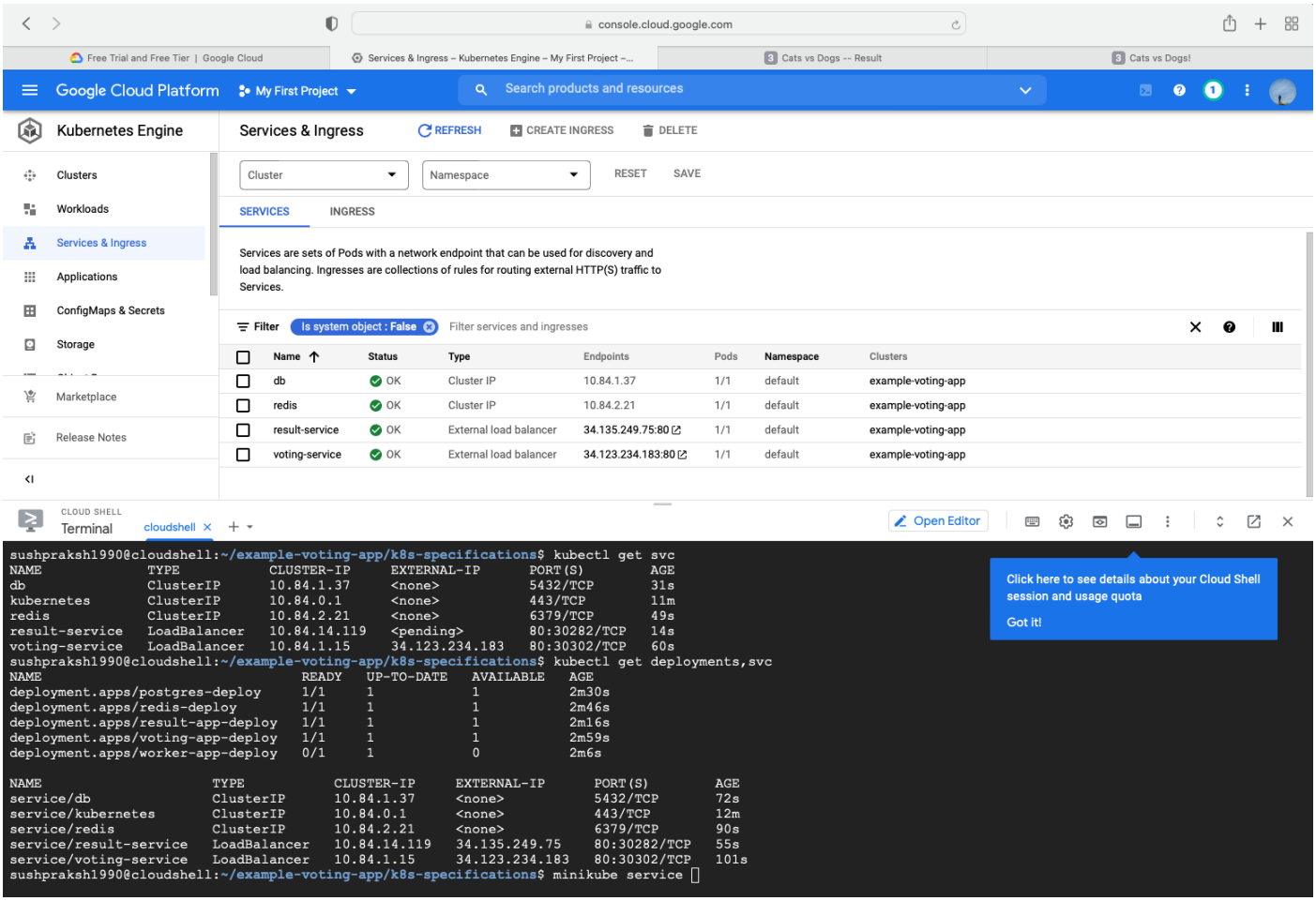
Hit on “run in cloud shell” and that shall open a terminal for you as follows:

Demo

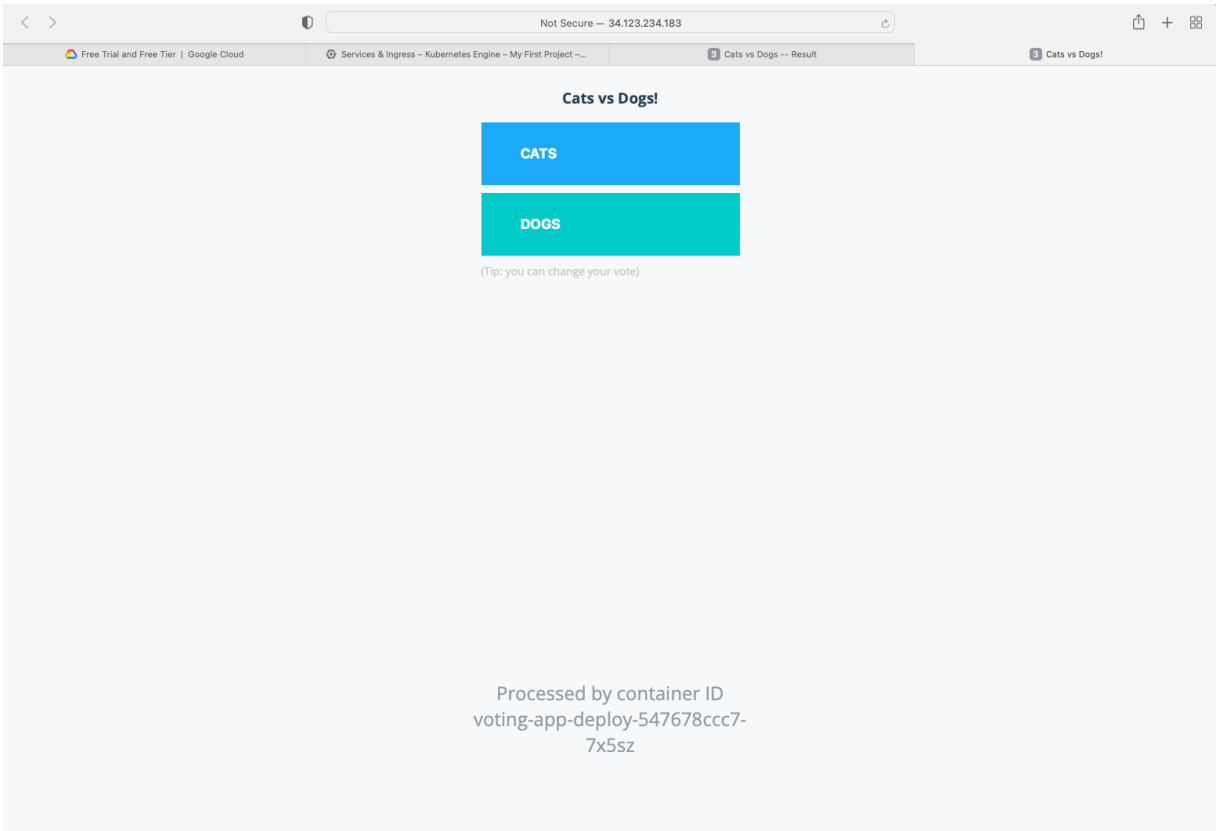
```
Welcome to Cloud Shell! Type "help" to get started.
To set your Cloud Platform project in this session use “gcloud config set project [PROJECT_ID]”
sushpraksh1990@cloudshell:~$ gcloud container clusters get-credentials example-voting-app --zone us-central1-a --project theta-totem-230506
Fetching cluster endpoint and auth data.
kubeconfig entry generated for example-voting-app.

sushpraksh1990@cloudshell:~$ kubectl get nodes
NAME                                          STATUS   ROLES    AGE     VERSION
gke-example-voting-app-default-pool-fb7460cf-7rfh   Ready    <none>   5m22s   v1.21.5-gke.1302
gke-example-voting-app-default-pool-fb7460cf-gnbf   Ready    <none>   5m22s   v1.21.5-gke.1302
gke-example-voting-app-default-pool-fb7460cf-gz79   Ready    <none>   5m25s   v1.21.5-gke.1302
sushpraksh1990@cloudshell:~$ git clone https://github.com/kodekloudhub/example-voting-app.git
Cloning into 'example-voting-app'...
remote: Enumerating objects: 872, done.
```

```
remote: Total 872 (delta 0), reused 0 (delta 0), pack-reused 872
Receiving objects: 100% (872/872), 957.04 KiB | 16.22 MiB/s, done.
Resolving deltas: 100% (310/310), done.
sushpraksh1990@cloudshell:~$ cd example-voting-app/k8s-specifications/
sushpraksh1990@cloudshell:~/example-voting-app/k8s-specifications$ ls

postgres-deploy.yaml  redis-deploy.yaml  result-app-deploy.yaml  voting-app-deploy.yaml  worker-app-deploy.yaml
postgres-service.yaml  redis-service.yaml  result-app-service.yaml  voting-app-service.yaml


postgres-service.yaml    redis-service.yaml      result-app-service.yaml  voting-app-service.yaml
sushpraksh1990@cloudshell:~/example-voting-app/k8s-specifications$ kubectl create -f voting-app-pod.yaml
error: the path "voting-app-pod.yaml" does not exist
sushpraksh1990@cloudshell:~/example-voting-app/k8s-specifications$ kubectl create -f voting-app-deploy.yaml
deployment.apps/voting-app-deploy created
sushpraksh1990@cloudshell:~/example-voting-app/k8s-specifications$ kubectl create -f redis-deploy.yaml
deployment.apps/redis-deploy created
sushpraksh1990@cloudshell:~/example-voting-app/k8s-specifications$ kubectl create -f postgres-deploy.yaml
deployment.apps/postgres-deploy created
sushpraksh1990@cloudshell:~/example-voting-app/k8s-specifications$ kubectl create -f result-app-deploy.yaml
deployment.apps/result-app-deploy created
sushpraksh1990@cloudshell:~/example-voting-app/k8s-specifications$ kubectl create -f worker-app-deploy.yaml
deployment.apps/worker-app-deploy created

sushpraksh1990@cloudshell:~/example-voting-app/k8s-specifications$ kubectl get pods
NAME                          READY   STATUS              RESTARTS   AGE
postgres-deploy-6f787b796b-h75mq    1/1     Running             0          30s
redis-deploy-5d7988b4bb-zz5rf       1/1     Running             0          46s
result-app-deploy-6cb79db456-8xftl  1/1     Running             0          16s
voting-app-deploy-547678ccc7-7x5sz  1/1     Running             0          59s
worker-app-deploy-799b5fb489-m2fc7  0/1     ContainerCreating   0          6s


sushpraksh1990@cloudshell:~/example-voting-app/k8s-specifications$ kubectl create -f voting-app-service.yaml
service/voting-service created
sushpraksh1990@cloudshell:~/example-voting-app/k8s-specifications$ kubectl create -f redis-service.yaml
service/redis created
sushpraksh1990@cloudshell:~/example-voting-app/k8s-specifications$ kubectl create -f postgres-service.yaml
service/db created
sushpraksh1990@cloudshell:~/example-voting-app/k8s-specifications$ kubectl create -f result-app-service.yaml
service/result-service created


sushpraksh1990@cloudshell:~/example-voting-app/k8s-specifications$ kubectl get deployments,svc
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/postgres-deploy    1/1     1            1           2m30s
deployment.apps/redis-deploy       1/1     1            1           2m46s
deployment.apps/result-app-deploy  1/1     1            1           2m16s
deployment.apps/voting-app-deploy  1/1     1            1           2m59s
deployment.apps/worker-app-deploy  0/1     1            0           2m6s

NAME                     TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)        AGE
service/db               ClusterIP      10.84.1.37     <none>           5432/TCP       72s
service/kubernetes       ClusterIP      10.84.0.1      <none>           443/TCP        12m
service/redis            ClusterIP      10.84.2.21     <none>           6379/TCP       90s
service/result-service   LoadBalancer   10.84.14.119   34.135.249.75    80:30282/TCP   55s
service/voting-service   LoadBalancer   10.84.1.15     34.123.234.183   80:30302/TCP   101s
```

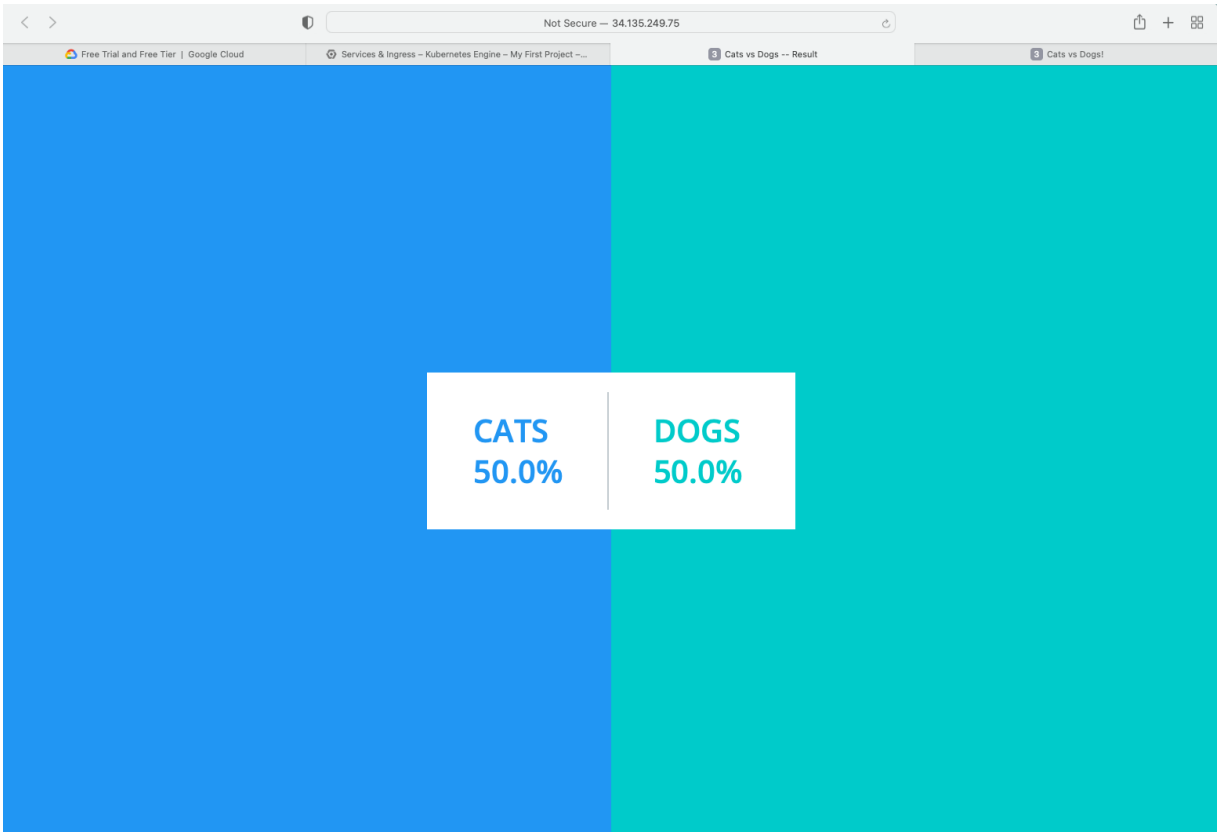
>> The services look like:



>> And the UI looks like:



And



Similar to GCP, all the cloud platforms provide similar functions a Cli/Kubectl terminal/cluster/load balancer/services with help of which we can deploy applications and also maintain without much hassle.