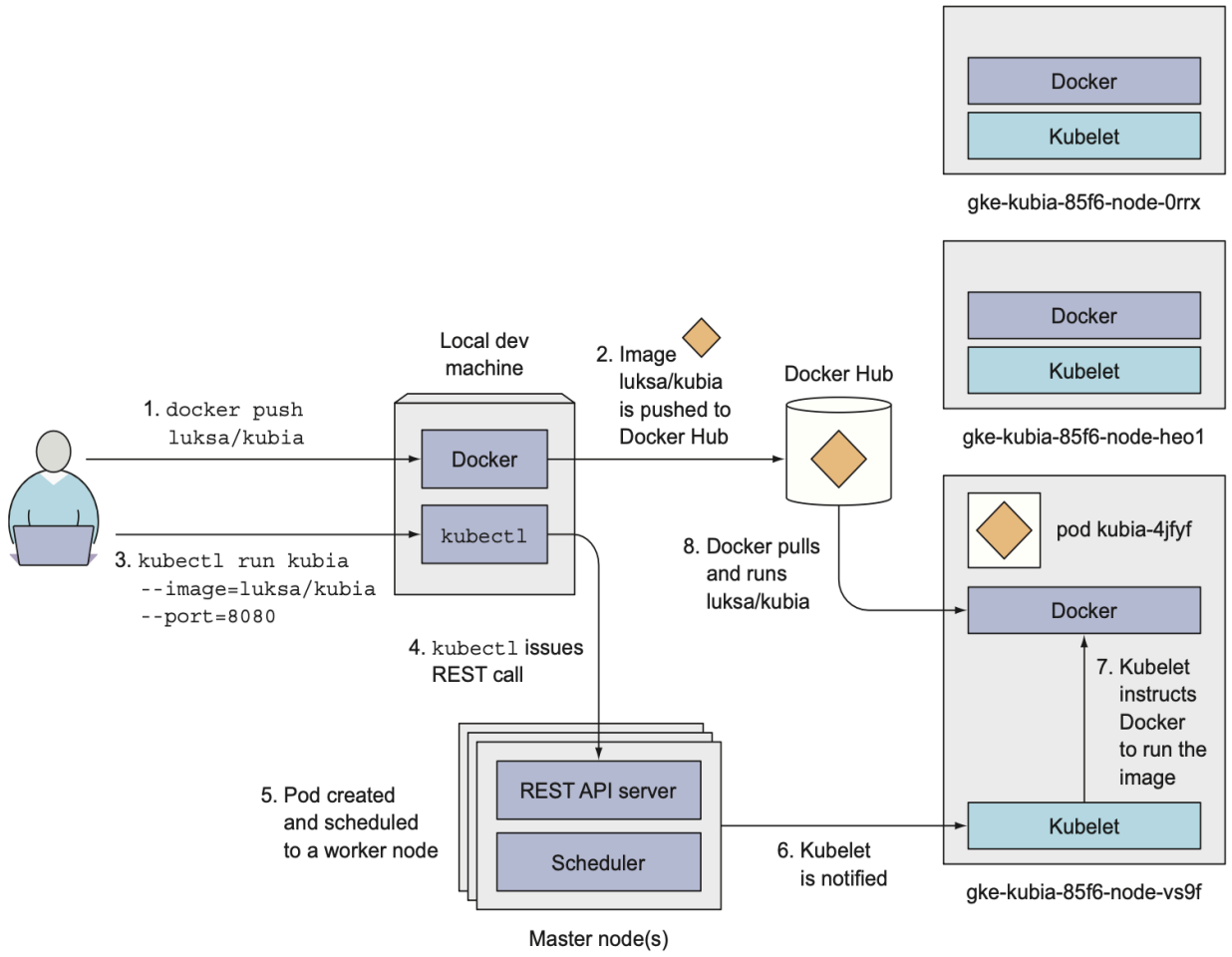
Hello Minikube

Running your first image/application on Minikube :

When you ran the kubectl command, it created a new ReplicationController object in the cluster by sending a REST HTTP request to the Kubernetes API server. The ReplicationController then created a new pod, which was then scheduled to one of the worker nodes by the Scheduler. The Kubelet on that node saw that the pod was scheduled to it and instructed Docker to pull the specified image from the registry because the image wasn't available locally. After downloading the image, Docker created and ran the container.



DEPLOYMENTS :

Once you have a running Kubernetes cluster / minikube , you can deploy your containerized applications on top of it. To do so, you create a Kubernetes Deployment. The Deployment instructs Kubernetes how to create and update instances of your application. Once you've created a Deployment, the Kubernetes control plane schedules the application instances included in that Deployment to run on individual Nodes in the cluster.
Once the application instances are created, a Kubernetes Deployment controller continuously monitors those instances. If the Node hosting an instance goes down or is deleted, the Deployment controller replaces the instance with an instance on another Node in the cluster. This provides a self-healing mechanism to address machine failure or maintenance

In a deployment, you can describe the desired state for your application and Kubernetes will constantly check if this state is matched. A deployment will create ReplicaSets which then ensures that the desired number of pods are running. If a pod goes down due to an interruption, the ReplicaSets controller will notice that the desired state does not match the actual state, and a new pod will be created.

<u>Here are some common use cases for deployments:</u>
Run stateless web servers, like the popular open-source Nginx. The deployment can request that a fixed number of pod replicas be instantiated, and Kubernetes will maintain that number of pods during the deployment.
Applications that require persistent storage, like a database instance, would use the StatefulSet type deployment and mount a persistent volume to ensure data integrity and longevity.
Deployments can automatically scale the number of replicas in the cluster as the workload increases. For example, they can automatically balance incoming requests between the replicas, create new replicas as demand increases, and terminate replicas as demand subsides.

Deploy an app :

Let's deploy our first app on Kubernetes with the kubectl create deployment command. We need to provide the deployment name and app image location (include the full repository url for images hosted outside Docker Hub).

kubectl create deployment himinikube --image=raghuramanbtech/nghw --replicas=1

Great! You just deployed your first application by creating a deployment. This performed a few things for you:

searched for a suitable node where an instance of the application could be run (we have only 1 available node)

scheduled the application to run on that Node

configured the cluster to reschedule the instance on a new Node when needed

To list your deployments use the kubectl get deployments command:

kubectl get deployments

We see that there is 1 deployment running a single instance of your app. The instance is running inside a container on your node.

```
raghuraman — -z

raghuraman@Raghus-Laptop ~ % kubectl create deployment himinikube --image=raghuramanbtech/nghw --replicas=1
deployment.apps/himinikube created
raghuraman@Raghus-Laptop ~ % kubectl get deployments
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
deplhwkb1         2/2     2            2           26d
hello-minikube    1/1     1            1           34d
himinikube        1/1     1            1           9s
mon15             1/1     1            1           20d
nghwdep           0/1     1            0           26d
nghwdephub        0/3     3            0           26d
nghwdephub1       1/1     1            1           26d
nginx             1/1     1            1           26d
testoday          1/1     1            1           23m
tue1              1/1     1            1           26d
wed1              0/1     1            0           26d
raghuraman@Raghus-Laptop ~ %
```

SERVICES :

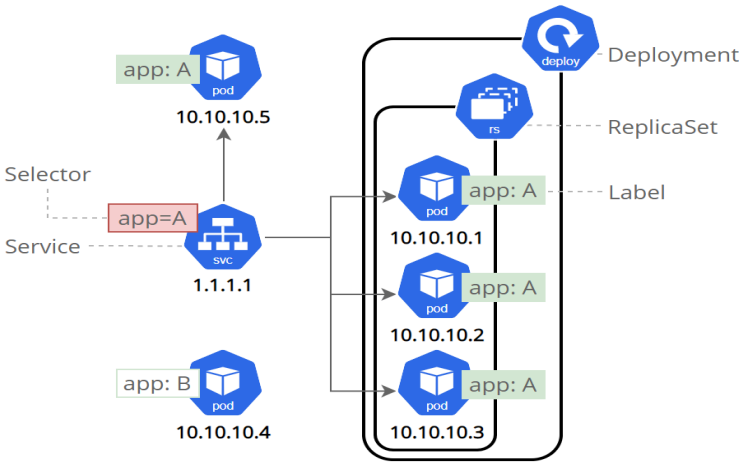
Although each Pod has a unique IP address, those IPs are not exposed outside the cluster without a Service. Services allow your applications to receive traffic. Services can be exposed in different ways by specifying a type in the spec of the Service:

- *ClusterIP* (default) - Exposes the Service on an internal IP in the cluster. This type makes the Service only reachable from within the cluster.
- *NodePort* - Exposes the Service on the same port of each selected Node in the cluster using NAT. Makes a Service accessible from outside the cluster using <NodeIP>:<NodePort>. Superset of ClusterIP.
- *LoadBalancer* - Creates an external load balancer in the current cloud (if supported) and assigns a fixed, external IP to the Service. Superset of NodePort.

Services and Labels

A Service routes traffic across a set of Pods. Services are the abstraction that allows pods to die and replicate in Kubernetes without impacting your application. Discovery and routing among dependent Pods (such as the frontend and backend components in an application) are handled by Kubernetes Services.

Services match a set of Pods using labels and selectors, a grouping primitive that allows logical operation on objects in Kubernetes. Labels are key/value pairs attached to objects .



Create a service :

A Service is a method for exposing a network application that is running as one or more Pods in your cluster. Pods are ephemeral resources (you should not expect that an individual Pod is reliable and durable).

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

To create a service , we need to expose our deployment to external traffic.
The —target port is your container's exposed port, the –port is the one you want to expose for your service.

kubectl expose deployment himinikube --type=NodePort --name=himiniservice --port=6663 --target-port=80

```
raghuraman@Raghus-Laptop ~ % kubectl expose deployment himinikube  --type=NodePort --name=himiniservice --port
service/himiniservice exposed
raghuraman@Raghus-Laptop ~ % kubectl get services
NAME             TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
deplhwkb1        NodePort       10.109.82.210    <none>        6666:31397/TCP   26d
depltoser1       NodePort       10.102.225.99    <none>        5554:31523/TCP   26d
hello-minikube   NodePort       10.101.42.206    <none>        8080:32743/TCP   34d
himiniservice    NodePort       10.110.232.101   <none>        6663:31444/TCP   10s
kubernetes       ClusterIP      10.96.0.1        <none>        443/TCP          35d
la               NodePort       10.101.42.226    <none>        6666:32232/TCP   42m
mon15            NodePort       10.102.109.16    <none>        6669:30756/TCP   20d
mon15-2          NodePort       10.109.248.21    <none>        6670:30982/TCP   20d
nghwdep          NodePort       10.107.112.172   <none>        6667:32536/TCP   26d
nghwdephub1      NodePort       10.98.62.235     <none>        6669:31353/TCP   26d
nginx            NodePort       10.111.105.31    <none>        7777:31700/TCP   26d
sername1         ClusterIP      10.99.117.235    <none>        5552/TCP         27d
sername2         NodePort       10.109.194.33    <none>        5553:31540/TCP   27d
testoday         ClusterIP      10.99.166.223    <none>        6669/TCP         44m
tue1             LoadBalancer   10.105.130.200   <pending>     8080:30191/TCP   26d
raghuraman@Raghus-Laptop ~ %
```

After exposing service , lets test our application,

Minikube service himinikube

```
raghuraman@Raghus-Laptop ~ % minikube service himiniservice
|-----------|---------------|-------------|---------------------------|
| NAMESPACE |     NAME      | TARGET PORT |            URL            |
|-----------|---------------|-------------|---------------------------|
| default   | himiniservice |        6663 | http://192.168.49.2:31444 |
|-----------|---------------|-------------|---------------------------|
🏃  Starting tunnel for service himiniservice.
|-----------|---------------|-------------|------------------------|
| NAMESPACE |     NAME      | TARGET PORT |          URL           |
|-----------|---------------|-------------|------------------------|
| default   | himiniservice |             | http://127.0.0.1:55447 |
|-----------|---------------|-------------|------------------------|
🎉  Opening service default/himiniservice in default browser...
❗  Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```

It will open the default Browser and we can see our application running .