kubernetes
June 2018

Introduction: (gcloud components install kubectl)

[Nice to read: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/]

Kubernetes is a platform to manage clusters (networks). Kubernetes can handle:

- Containers platform.
- Microservices platform.
- Portable cloud (Docker?).

With those abilities, Kubernetes enables portability across infrastructure providers.

Kubernetes operates at the container level rather than at the hardware level, it provides some generally applicable features common to PaaS (Platform as a Service) offerings, such as deployment, scaling, load balancing, logging, and monitoring. However, Kubernetes is not monolithic (we can change feature in Kubernetes), and these default solutions are optional and pluggable. Kubernetes provides the building blocks for building developer platforms, but preserves user choice and flexibility where it is important.

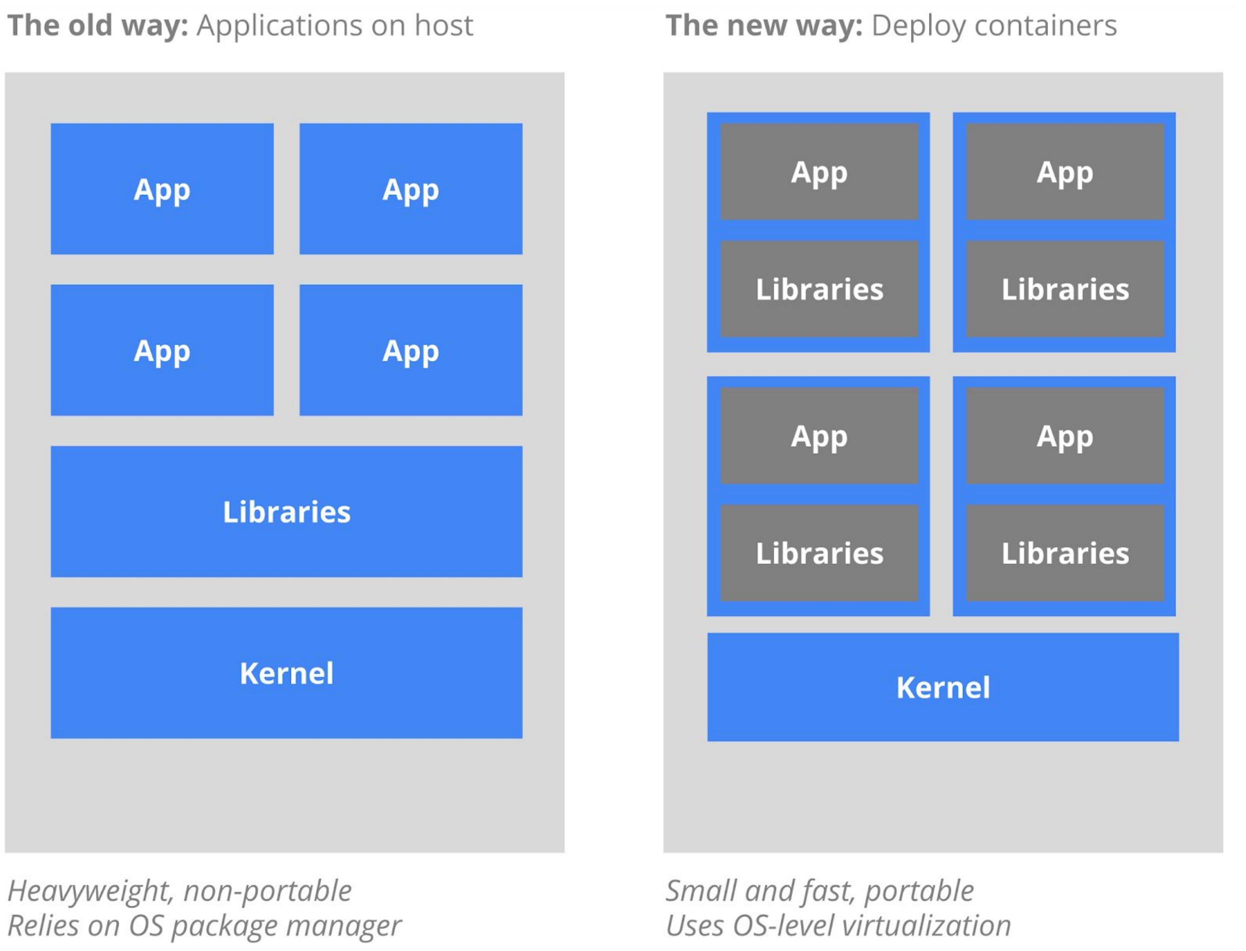
Kubernetes is comprised of a set of independent, composable control processes that continuously drive the current state towards the provided desired state. It shouldn't matter how you get from A to C.

Kubernetes is a container cluster orchestration system

Kubernetes gives us:

- Deployment.
- Scaling.
- Monitoring

Containers:



The old way: Applications on host
App
App
App
App
Libraries
Kernel
Heavyweight, non-portable
Relies on OS package manager
The new way: Deploy containers
App
Libraries
App
Libraries
App
Libraries
App
Libraries
Kernel
Small and fast, portable
Uses OS-level virtualization

Container is the app + the libs\bins for running the app. Containers has the same OS, CPU, memory. Container is running above a Docker.

Containers mean virtualization inside the operating-system layer.

Resources:

Pods - minimal running unit, include one container or more that can connect between them. Gets a unique IP in the cluster.

Deployment - with pods roles, but also add roles to the Kubernetes run:

- Amount of copies.
- When and how to upgrade a version (e.g. only replace 2 instances in a time, at least 10 copies...).
- Really good for stateless microservices.

DaemonSet - like deployment, but ensure us we will have at least one pod copy on every server. Use mainly for logs.

Service - The routing unit.

CronJob \ Job - like regular crontab.

ConfigMap & Secrets - Manage the secure info in the cluster.

Volumes - connect data to pods (we need it mainly because pod can run on different server every time so we need to connect the disk\data in a dynamic way).

Many Kubernetes solutions - https://github.com/kubernetes/helm

https://drone.io/

https://kubernetes.io/docs/tutorials/

Kubernetes Clusters

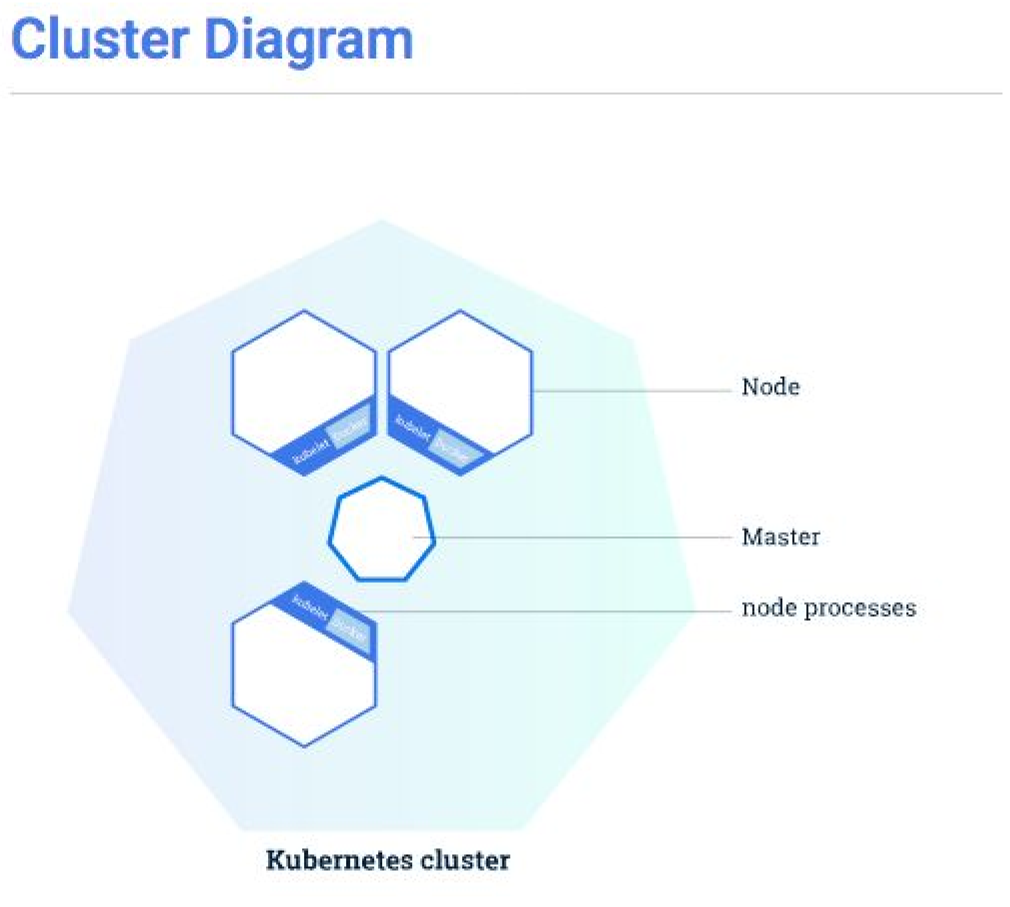
[https://kubernetes.io/docs/tutorials/kubernetes-basics/create-cluster/cluster-intro/]

Kubernetes coordinates a highly available cluster of computers that are connected to work as a single unit.

We can deploy a version to a cluster instead of to every compute engine. To make use of this new model of deployment, applications need to be packaged in a way that decouples them from individual hosts: they need to be containerized.

A Kubernetes cluster consists of two types of resources:

- The **Master** coordinates the cluster (like management).
- **Nodes** are the workers that run the app.



Each node has a Kubelet, which is an agent for managing the node and communicating with the Kubernetes master.

The node should have a tool for containers (like Docker).

How does it work?

When you deploy applications on Kubernetes, you tell the master to start the application containers. The master schedules the containers to run on the cluster's nodes. **The nodes communicate with the master using the Kubernetes API**, which the master exposes. End users can also use the Kubernetes API directly to interact with the cluster.

Minikube

Minikube is a lightweight Kubernetes implementation that creates a VM on your local machine and deploys a simple cluster containing only one node. Minikube is available for Linux, macOS, and Windows systems. The Minikube CLI provides basic bootstrapping operations for working with your cluster, including start, stop, status, and delete.

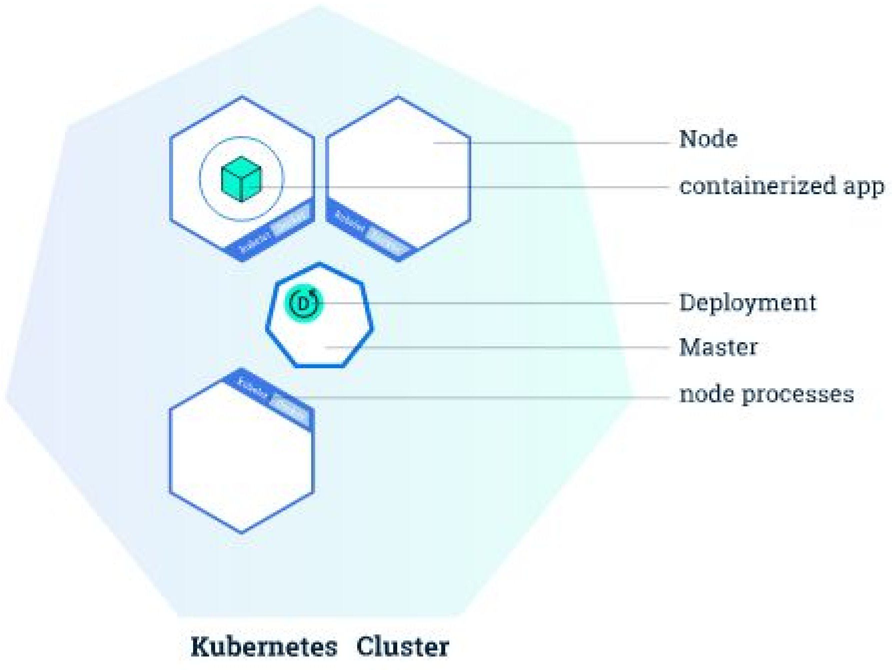
<u>How to use it?</u>

- minikube version - verify minikube is installed and we are ready to go.
- minikube start - started a virtual machine for you, and a Kubernetes cluster is now running in that VM.

Kubernetes Deployments

Kubernetes coordinates a highly available cluster of computers that are connected to work as a single unit.

The Deployment instructs Kubernetes how to create and update instances of your application. Once you've created a Deployment, the Kubernetes master schedules mentioned application instances onto individual Nodes in the cluster.



[Tutorial - https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-interactive/]

Kubectl

- kubectl cluster-info - cluster details.
- kubectl get nodes - view the nodes in the cluster.
- kubectl run - The run command creates a new deployment. We need to provide the deployment name and app image location (include the full repository url for images hosted outside Docker hub). Port can be added too. (e.g. kubectl run

kubernetes-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1 --port=8080).

- kubectl get deployments - list of our deployments.
- kubectl proxy - communicate with the cluster from outside (with curl commands for example: curl http://localhost:8001/version).
- Working with the proxy:
    - The API server will automatically create an endpoint for each pod, based on the pod name, that is also accessible through the proxy
    - First we need to get the Pod name, and we'll store in the environment variable POD_NAME:
        - export POD_NAME=$(kubectl get pods -o go-template --template '{{range .items}}{{.metadata.name}}{{"\n"}}{{end}}') echo Name of the Pod: $POD_NAME
    - Now we can make an HTTP request to the application running in that pod:
        - curl http://localhost:8001/api/v1/namespaces/default/pods/$POD_NAME/proxy/
- kubectl get pods|node|other - list resources
- kubectl describe pods|node|other - show detailed information about a resource
- kubectl logs $POD_NAME - print the logs from a container in a pod
- kubectl exec $POD_NAME <command, like env| -ti $POD_NAME bash |...>- execute a command on a container in a pod

Viewing Pods and Nodes

[https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/]

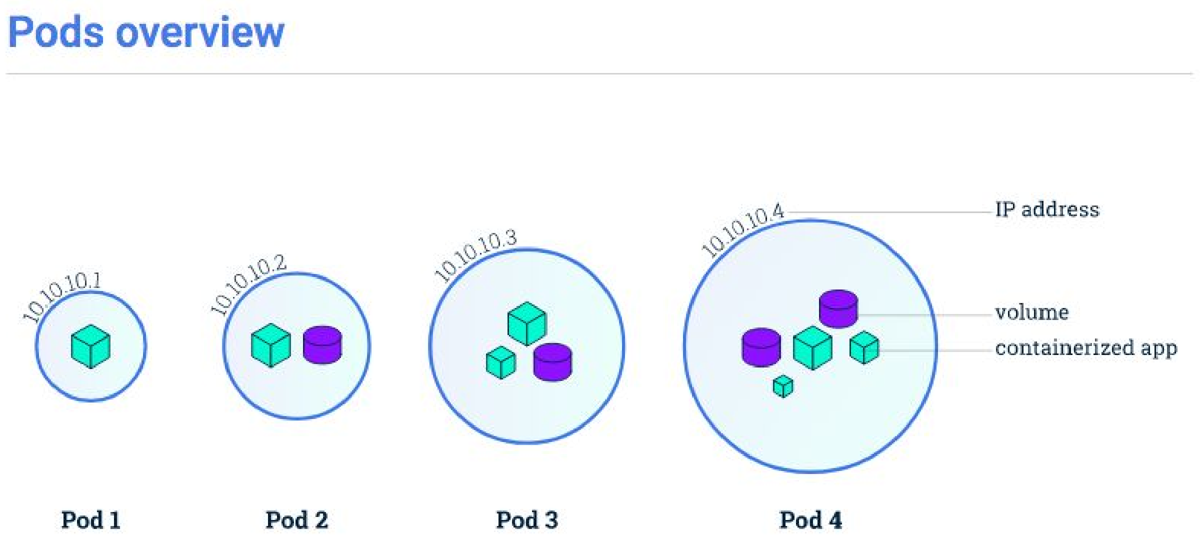
A Pod is a group of one or more application containers (such as Docker or rkt) and includes shared storage (volumes), IP address and information about how to run them.

Kubernetes Pod is a Kubernetes abstraction that represents a group of one or more application containers and some shared resources for those containers.

A Pod models an application-specific "logical host" and can contain different application containers which are relatively tightly coupled (For example, a Pod might include both the container with your app as well as a different container that feeds the data to be published by the app server). Pods are the atomic unit on the Kubernetes platform. When we create a Deployment on Kubernetes, that Deployment creates Pods with containers inside them.

Important things from the emulator:

- Pods are running in an isolated, private network - so we need to proxy access to them so we can debug and interact with them.
-



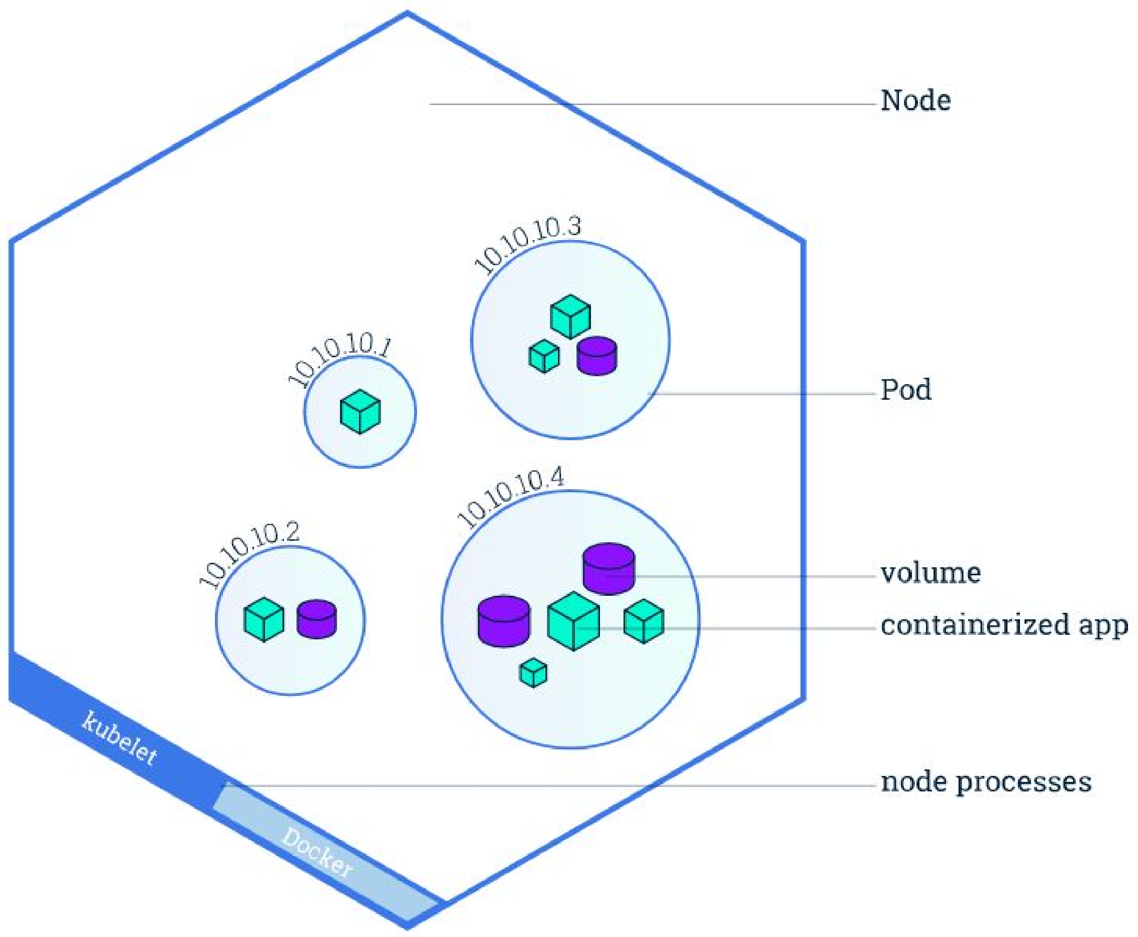
A node is a worker machine in Kubernetes and may be a VM or physical machine, depending on the cluster. Multiple Pods can run on one Node.

Nodes - A Pod always runs on a Node. A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine, depending on the cluster. Each Node is managed by the Master.

Every Kubernetes Node runs at least:

- Kubelet, a process responsible for communication between the Kubernetes Master and the Node; it manages the Pods and the containers running on a machine.
- A container runtime (like Docker, rkt) responsible for pulling the container image from a registry, unpacking the container and the running application.

Node overview



Using a Service to Expose Your App

A Kubernetes Service is an abstraction layer which defines a logical set of Pods and enables external traffic exposure, load balancing and service discovery for those Pods.

Kubernetes Services

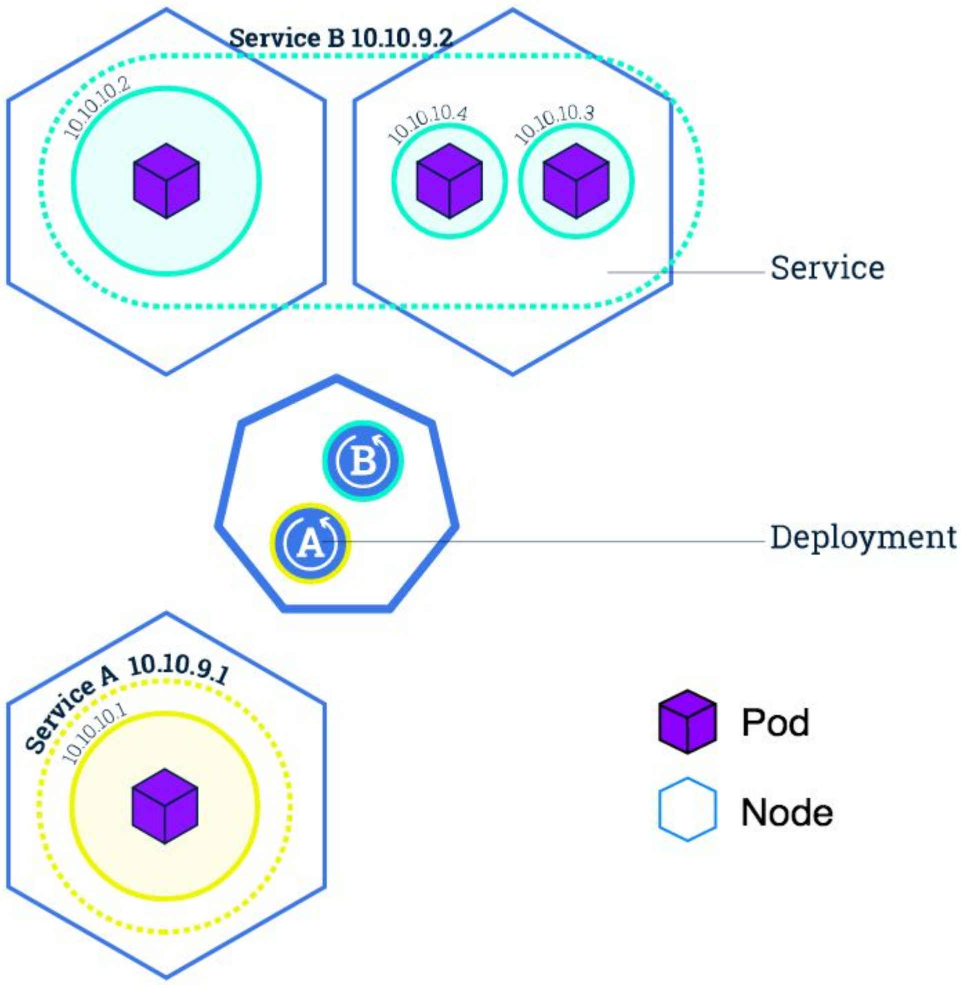
Kubernetes Pods are mortal. Pods in fact have a lifecycle. When a worker node dies, the Pods running on the Node are also lost. Each Pod in a Kubernetes cluster has a unique IP address, even Pods on the same Node, so there needs to be a way of automatically reconciling changes among Pods so that your applications continue to function.

A Service in Kubernetes is an abstraction which defines a logical set of Pods and a policy by which to access them. A Service is defined using YAML (preferred) or JSON, like all Kubernetes objects.

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 ServiceSpec:

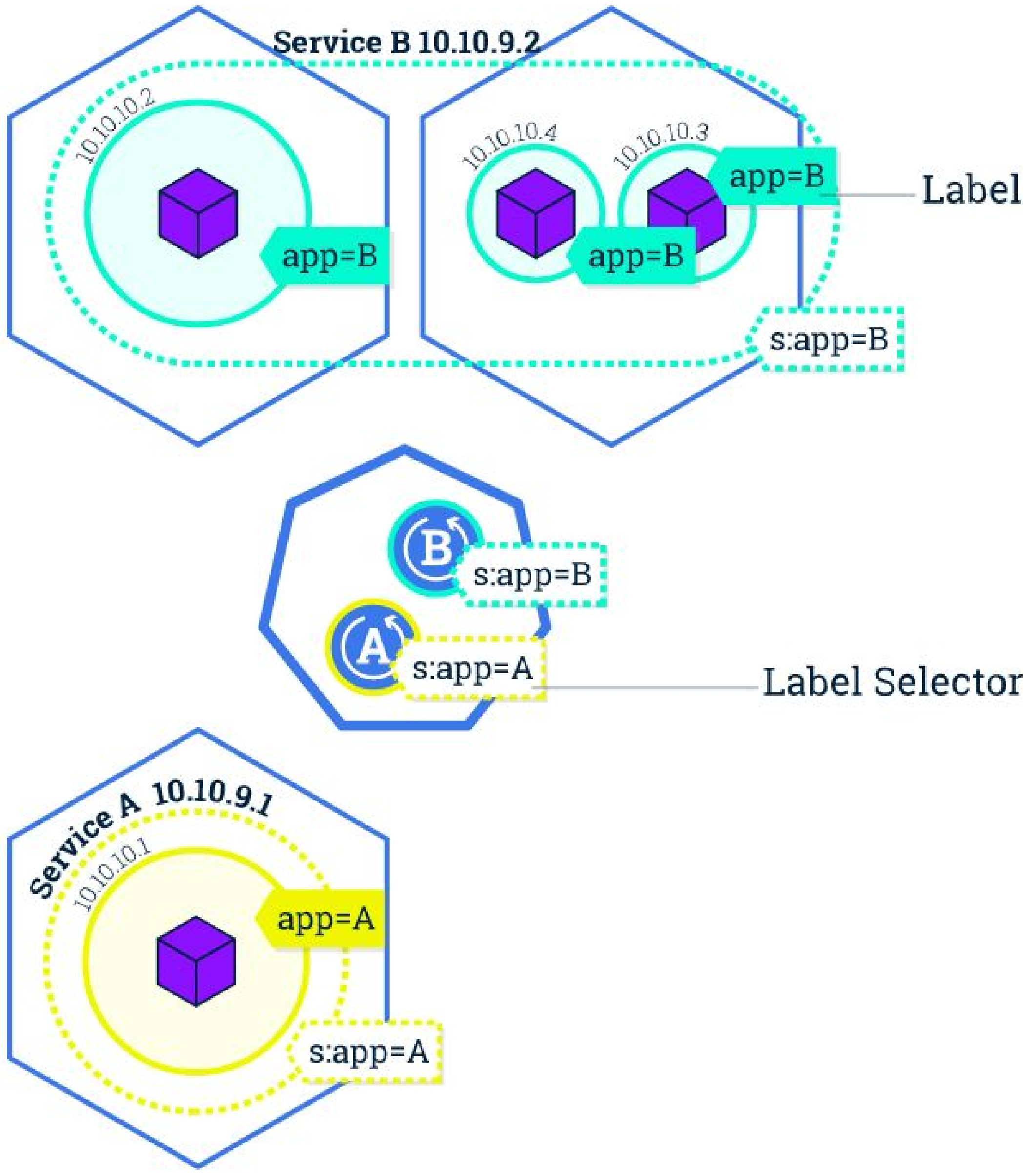
- *ClusterIP* (default) - Exposes the Service on an internal IP in the cluster. This type makes the Service only reachable from within the cluster.
- *NodePort* - Exposes the Service on the same port of each selected Node in the cluster using NAT. Makes a Service accessible from outside the cluster using <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.
- *ExternalName* - Exposes the Service using an arbitrary name (specified by externalName in the spec) by returning a CNAME record with the name (No proxy is used. This type requires v1.7 or higher of kube-dns).
- Useful - Connecting Applications with Services.

Services and Labels



A Service routes traffic across a set of Pods. **Services are the abstraction that allow pods to die and replicate in Kubernetes without impacting your application.** Services match a set of Pods using labels and selectors, a grouping primitive that allows logical operation on objects in Kubernetes.

Labels are dynamic and can be changed.



Useful commands:

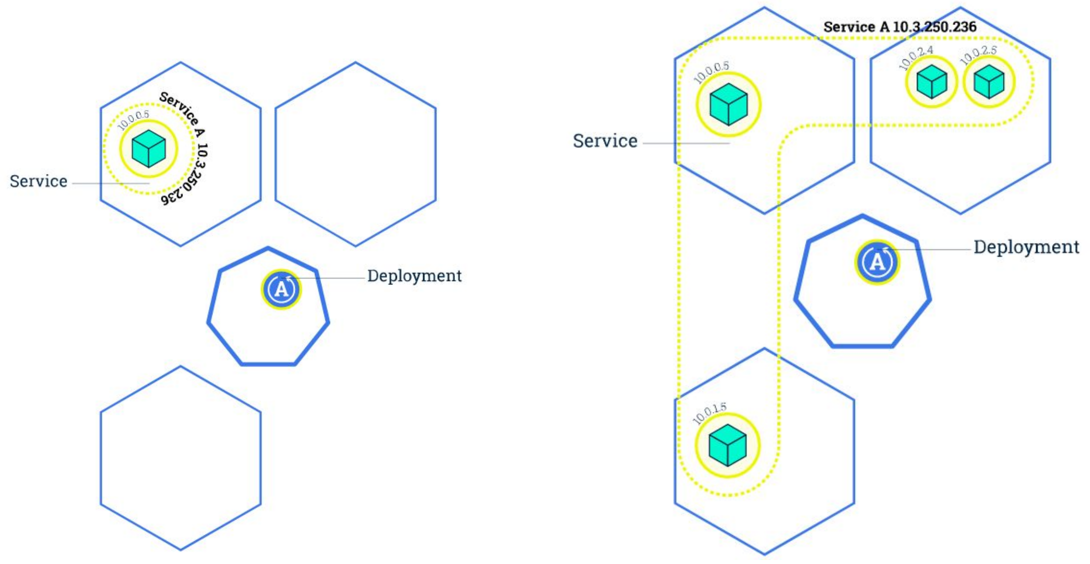
- kubectl expose <deployment>(deployment/kubernetes-bootcamp) --type="NodePort" --port 8080 - expose the service to external traffic.
- export NODE_PORT=$(kubectl get services/kubernetes-bootcamp -o go-template='{{(index .spec.ports 0).nodePort}}') - Create an environment variable

called NODE_PORT that has the value of the Node port expose. (curl $(minikube ip):$NODE_PORT to verify the app is expose outside).

- kubectl describe deployment - see the name of the label.
- kubectl label pod $POD_NAME app=v1 - Apply a new label we use the label command followed by the object type, object name and the new label.

Running Multiple Instances of Your App

Scaling in will reduce the number of Pods to the new desired state. Kubernetes also supports autoscaling of Pods.

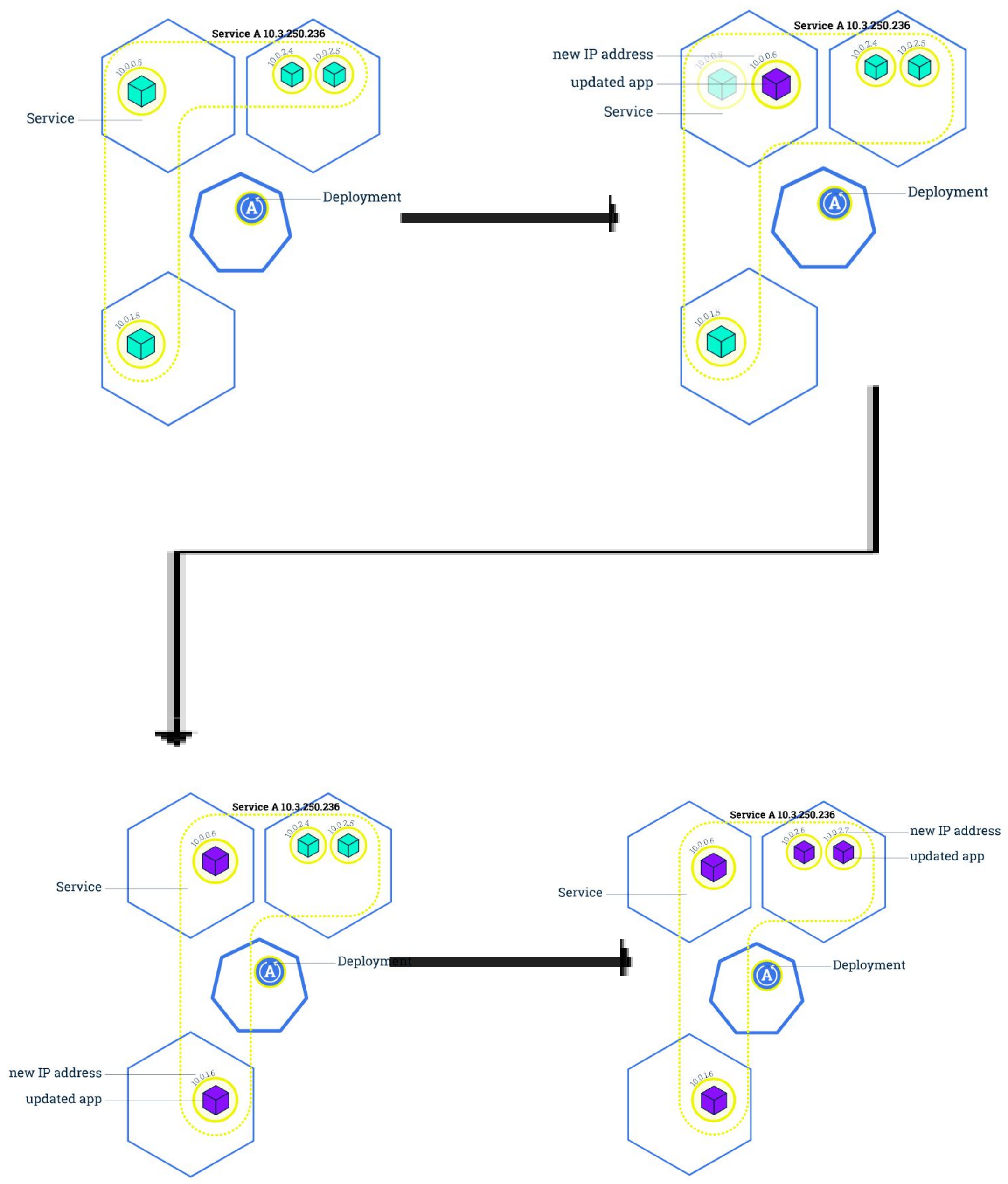


Useful commands:

- kubectl get deployments - The DESIRED state is showing the configured number of replicas, the CURRENT state show how many replicas are running now and the UP-TO-DATE is the number of replicas that were updated to match the desired (configured) state.
- kubectl scale <deployment type> <name> <number of instances> - deployments/kubernetes-bootcamp --replicas=4.
- kubectl get pods -o wide - verify pods number (should be 4 now).

Performing a Rolling Update

Rolling updates allow Deployments' update to take place with zero downtime by incrementally updating Pods instances with new ones. The new Pods will be scheduled on Nodes with available resources. The maximum number of Pods that can be unavailable during the update and the maximum number of new Pods that can be created, is one.



Useful commands:

- kubectl set image <deployment name> <new image version> - deploy new image.
- kubectl rollout status <deployment name> - confirmed update by running a rollout status command.
- kubectl rollout undo <deployment name> - The rollout command reverted the deployment to the previous known state.

Minikube

Finish minikube tutorial

https://kubernetes.io/docs/setup/minikube/

Basic installation steps:

- https://kubernetes.io/docs/tutorials/hello-minikube/

Docker - workshop

[https://github.com/alexryabtsev/docker-workshop#example-1-hello-world]

The dockerfile must have:

- **FROM** -- set base image
- **RUN** -- execute command in container
- **ENV** -- set environment variable
- **WORKDIR** -- set working directory
- **VOLUME** -- create mount-point for a volume
- **CMD** -- set executable for container

Nice to read:

Nodes structure:

- https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0
- https://medium.com/google-cloud/internal-load-balancing-for-kubernetes-services-on-google-cloud-f8aef11fb1c4

Cli:

- https://kubernetes.io/docs/reference/kubectl/cheatsheet/

Move to k8s:

- https://medium.com/google-cloud/kubernetes-101-pods-nodes-containers-and-clusters-c1509e409e16
- https://blog.alexellis.io/move-your-project-to-kubernetes/
- https://blog.openshift.com/dynamic-kubernetes-client-for-python-and-ansible/
- Scale: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

Logs (ELK):

- https://github.com/kubernetes/kubernetes/tree/master/cluster/addons/fluentd-elasticsearch

Minikube:

- https://medium.com/@wisegain/minikube-cheat-sheet-a273385e66c9
- https://gist.github.com/kevin-smets/b91a34cea662d0c523968472a81788f7