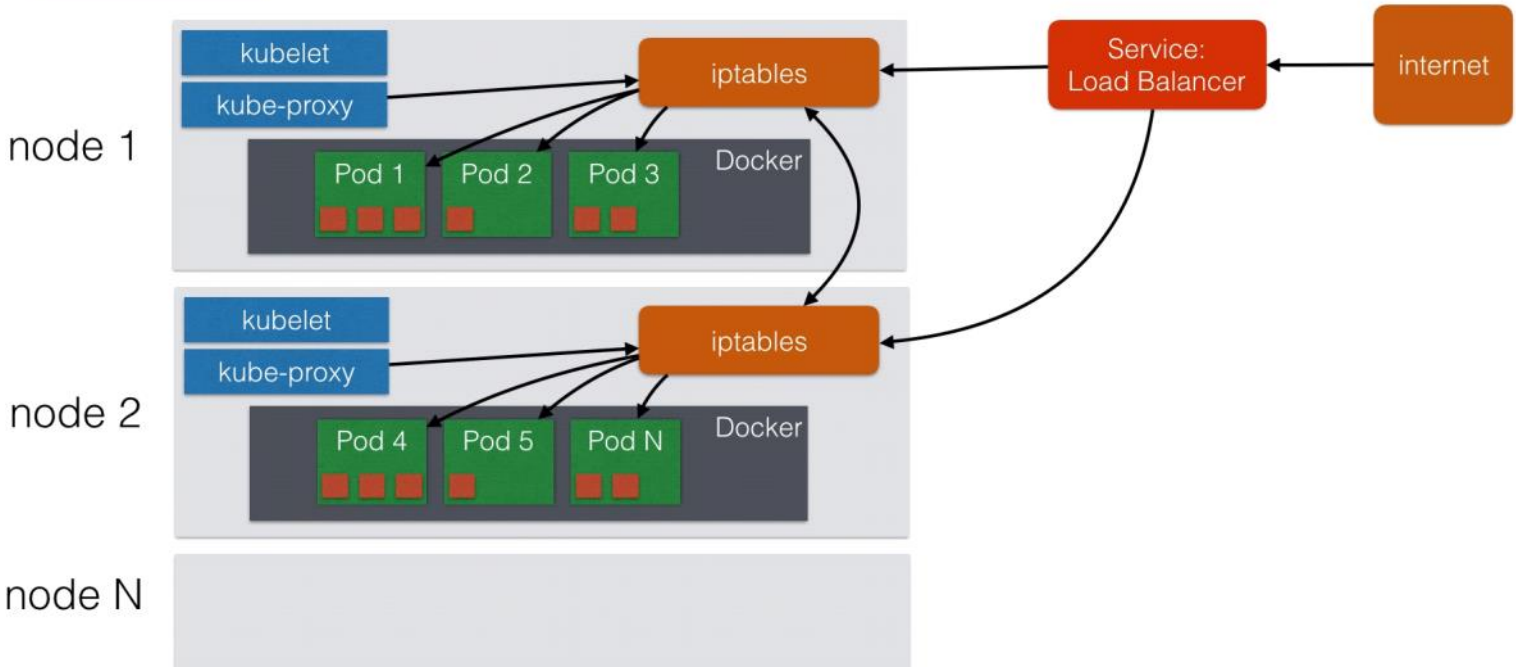
Saturday, October 5, 2019 7:06 AM

Kubernetes Node Architecture



Scaling

- If your application is **stateless** you can horizontally scale it
  - Stateless = your application does't have a **state**, it doesn't **write** any **local files** / keeps local sessions
  - All traditional databases (MySQL, Postgres) are **stateful**, they have database files that can't be split over multiple instances
- Most **web applications** can be made stateless:
  - **Session management** needs to be done outside the container

- Scaling in Kubernetes can be done using the **Replication Controller**
- The replication controller will **ensure** a specified number of **pod replicas** will run at all time
- A pods created with the replica controller will **automatically** be **replaced** if they fail, get deleted, or are terminated
- Using the replication controller is also **recommended** if you just want to make sure **1 pod** is always running, even after reboots
    - You can then run a replication controller with just **1 replica**

Deployment

- A deployment declaration in Kubernetes allows you to do app **deployments** and **updates**
- When using the deployment object, you define the **state** of your application
    - Kubernetes will then make sure the clusters matches your **desired** state
- Just using the **replication controller** or **replication set** might be **cumbersome** to deploy apps
    - The **Deployment Object** is easier to use and gives you more possibilities

- With a deployment object you can:
    - **Create** a deployment (e.g. deploying an app)
    - **Update** a deployment (e.g. deploying a new version)
    - Do **rolling updates** (zero downtime deployments)
    - **Roll back** to a previous version
    - **Pause / Resume** a deployment (e.g. to roll-out to only a certain percentage)

SERVICES

- **Pods** are very **dynamic**, they come and go on the Kubernetes cluster

Services

- **Pods** are very **dynamic**, they come and go on the Kubernetes cluster
  - When using a **Replication Controller**, pods are **terminated** and created during scaling operations
  - When using **Deployments**, when **updating** the image version, pods are **terminated** and new pods take the place of older pods
- That's why Pods should never be accessed directly, but always through a **Service**
- A service is the **logical bridge** between the "mortal" pods and other **services** or **end-users**
- When using the "kubectl expose" command earlier, you created a new Service for your pod, so it could be accessed externally
- Creating a service will create an endpoint for your pod(s):
  - a **ClusterIP**: a virtual IP address only reachable from within the cluster *(this is the default)*
  - a **NodePort**: a port that is the same on each node that is also reachable externally
  - a **LoadBalancer**: a LoadBalancer created by the **cloud provider** that will route external traffic to every node on the NodePort (ELB on AWS)