**Docker to Kubernetes Migration**

**Docker** is a containerization tool which helps the developers to package the application along with it‘ s dependencies such as libraries and binaries etc., along with it.

**Kubernetes** is a container orchestration tool which takes care of the scaling requirements, failover, deployment patterns, and more.

**Advantages of Kubernetes:**

**1.** **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.

**2. 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.

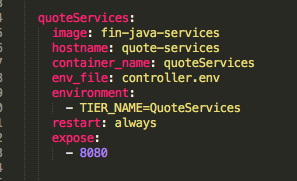
**3. Self-healing**  
Kubernetes restarts containers that fail, replaces containers, kills containers that don’t respond to your user-defined health check, and doesn’t advertise them to clients until they are ready to serve.

**Preparing for the Migration**

The first thing to do is to understand how Kubernetes is different from Docker. One of the most important differences is in container-to-container communication.

In a Docker Compose environment, the containers all run on a single host machine. Docker Compose creates a local network that the containers are all part of.

Take this snippet, for example:



This block will create a container called quoteServices with a hostname of quote-services and port 8080. With this definition, any container within the local Docker Compose network can access it using http://quote-services:8080. Anything outside of the local network would have to know the IP address of the container.

By comparison, Kubernetes usually runs on multiple servers called nodes, so it can’t simply create a local network for all the containers.

In Kubernetes, the same snipped mentioned above would be converted into two separate sections, a **Deployment** and a **Service**.

Let ‘s understand Deployment and Service in more detail.

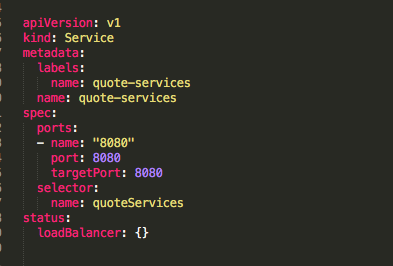
**Deployment Section:**

As its name implies, the deployment tells Kubernetes most of what it needs to know about how to deploy the containers. This information includes things like what to name the containers, where to pull the images from, how many containers to create, etc. The deployment for quoteServices is shown here:



**Service Section:**

The Service is what enables communication between containers. Here is the service definition for quoteServices:



This service definition tells Kubernetes to take the containers that have a name = quoteServices, as defined under selector, and to make them reachable using quote-services as hostname and port 8080.

So again, this service can be reached at **http://quote-services:8080** from within the Kubernetes application. The flexibility to define services this way allows us to keep our URLs intact within our application, so no changes are needed due to networking concerns.

So in order to migrate from Docker/Docker Compose to Kubernetes, we must covert the docker compose file to two files one to handle **deployment** and the other to handle **service**.