Kubernetes: An open source scheduler for Docker containers

Submitted by
Udyan Khurana 201202101
Veerendra Bidare 201405571
Sourav Sarangi 201301014
Rupinder Singh Khokhar 201201181

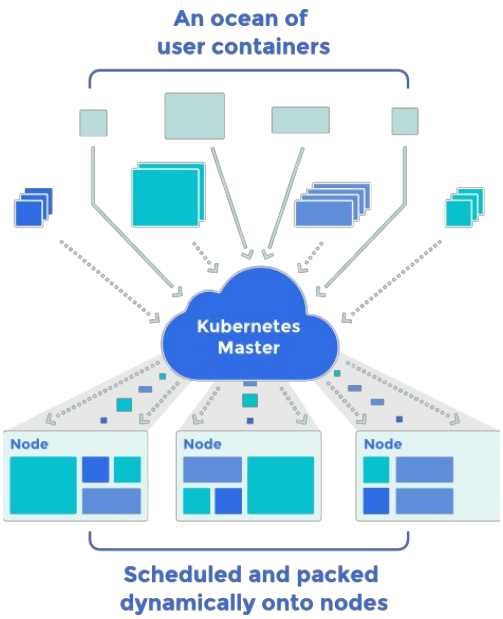
Introduction

Containers have been around for a while now. Docker is one of the companies that have made a big impact on how Container technology has evolved. Virtual Machines, had gained popularity in the past, as they could hand us a ready hardware of a given configuration. Docker containers add a whole new layer by making the application readily deployable, with a ready Virtual Machine. Some of the palpable advantages are,

1. Reduced load during deployment.
2. All dependencies being at one place.
3. Application being isolated from other applications from resource perspective.
4. Last but not the least, reducing 'That works on my machine' debate between developers and Testers!

What is Kubernetes?

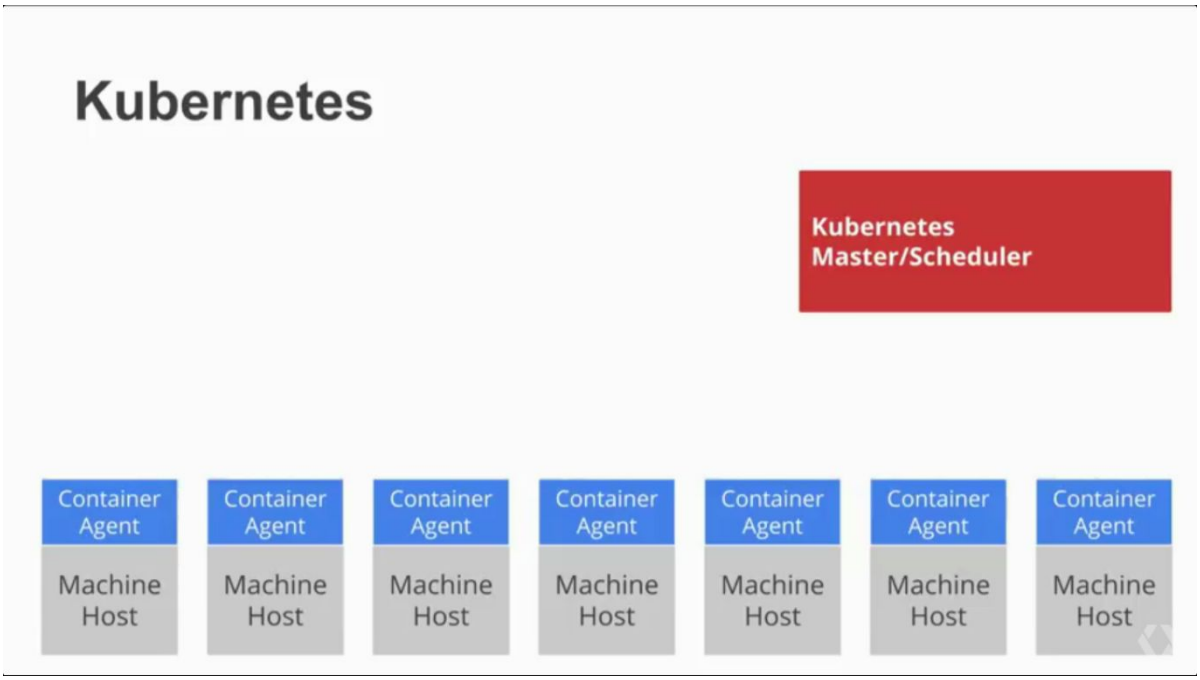
Kubernetes is an open-source orchestration platform for automating deployment, scaling, and operations of Docker containers across clusters of hosts. It was started by Google in 2014. It handles scheduling onto nodes in a compute cluster and actively manages workloads to ensure that their state matches the user's declared intentions. It groups the containers which make up an application into logical units for easy management and discovery. Kubernetes allows you to scale your applications easily, seamlessly roll out new features, and optimize use of your hardware by using only the resources you need. It is lightweight, portable,



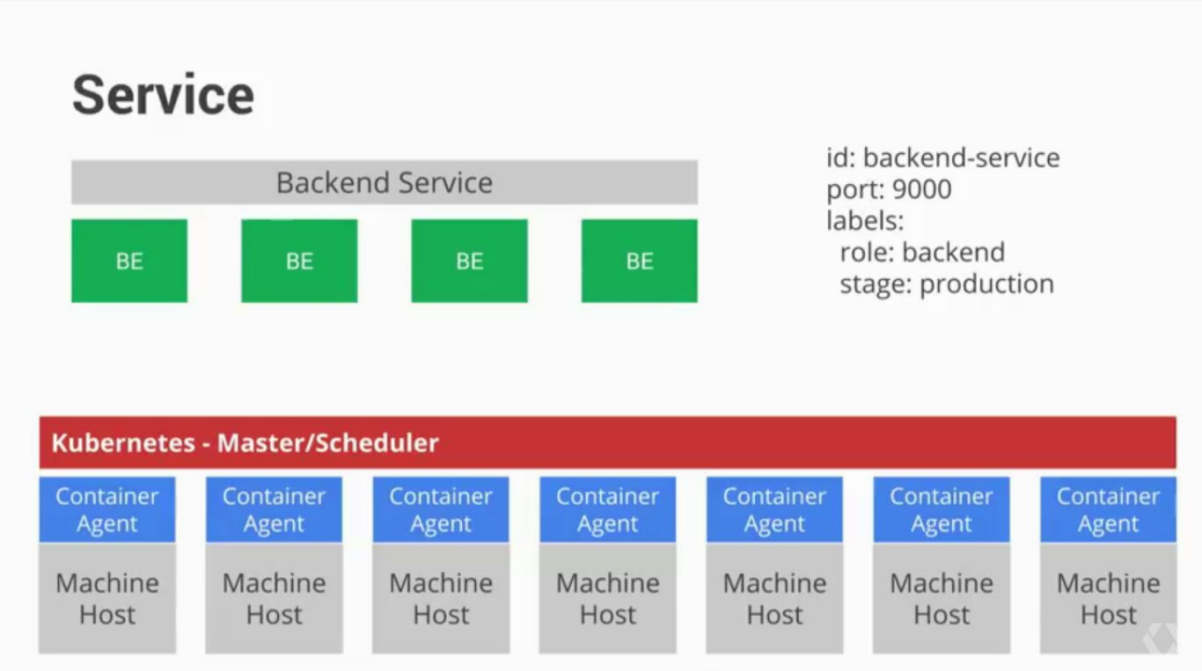
extensible and self-healing. Kubernetes also provides separation of build and deployment. Therefore, decoupling applications from infrastructure.

Architecture

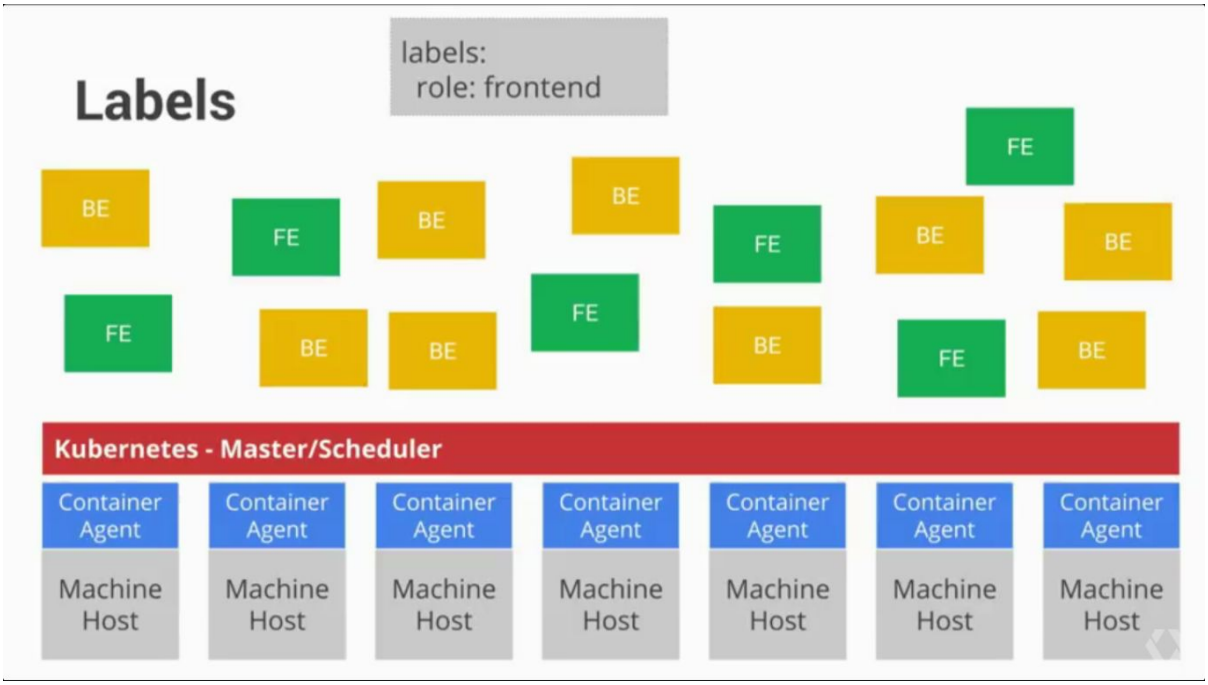
1. A running Kubernetes cluster contains node agents (kubelet) and master components (APIs, scheduler, etc), on top of a distributed storage solution.



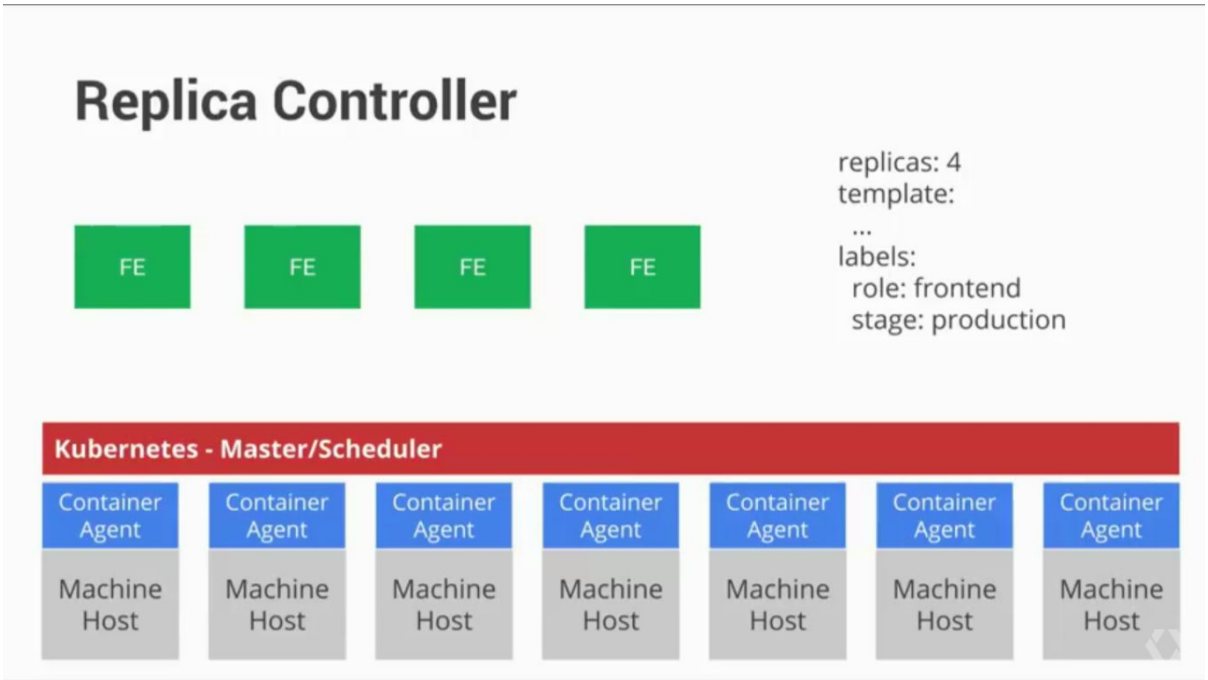
2. Architecture of the system consists of services that run on the worker node and services that compose the cluster-level control plane.



3. The Kubernetes node has the services necessary to run application containers and be managed from the master systems.
4. Each node runs Docker which takes care of the details of downloading images and running containers. The Kubelet manages pods and their containers, their images, their volumes, etc.
5. A *pod* corresponds to a colocated group of applications running with a shared context. Within that context, the applications may also have individual cgroup isolations applied. A pod models an application-specific "logical host" in a containerized environment.
6. Labels : Containers are labelled with their roles, stage it works in, the port it occupies, the service it is assigned to and so on, which helps in selecting the containers easily later.



7. Replica Controller : Dynamic resizing is easy with Kubernetes. An application can request the number of containers dynamically, and Replica Controller handles all that. The resulting application is robust.



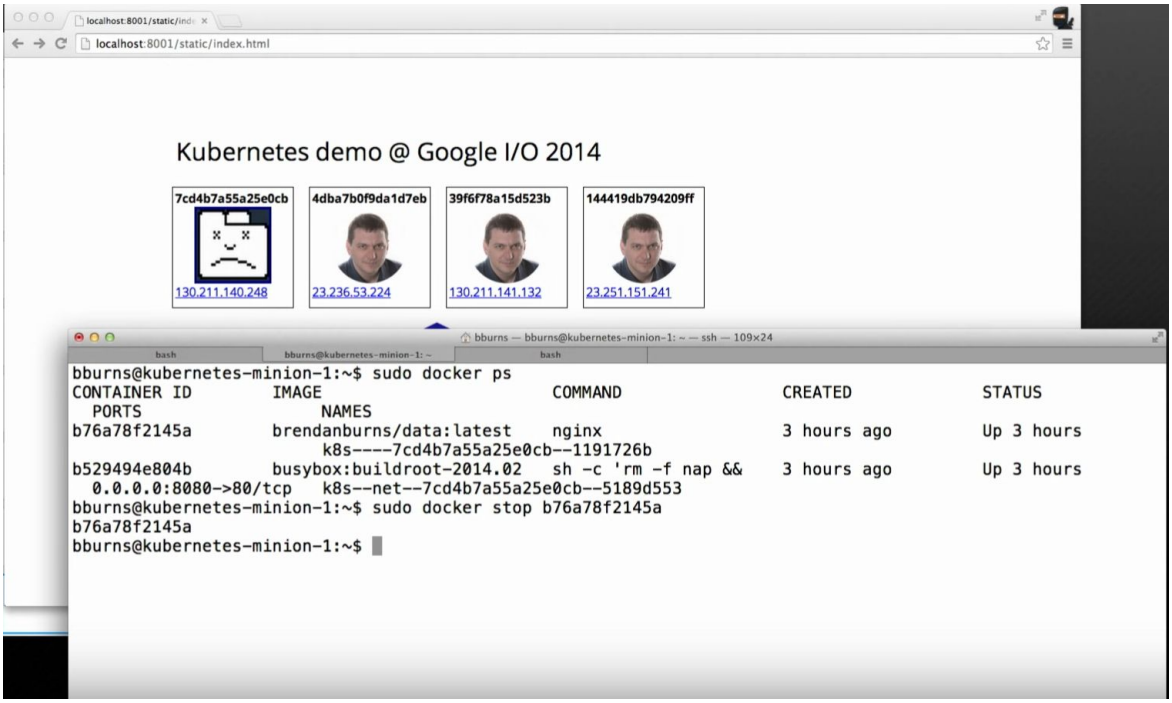
8. Health Management : If an application requests 4 containers and for some reason, one container stops working, Kubernetes makes sure, a new one is created and available.

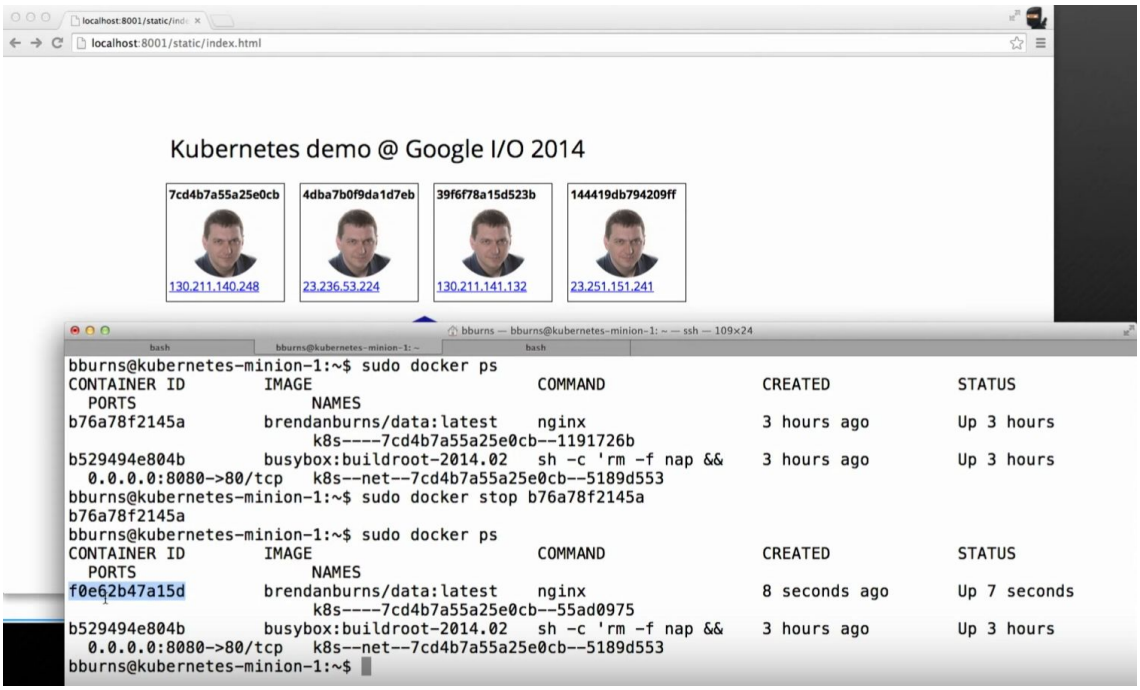
Proposed Approach and High Level Design

In this project, we propose to use Kubernetes for a small application and utilize all the features mentioned above. The proposed system will have the following capability.

1.Self Healing Property :

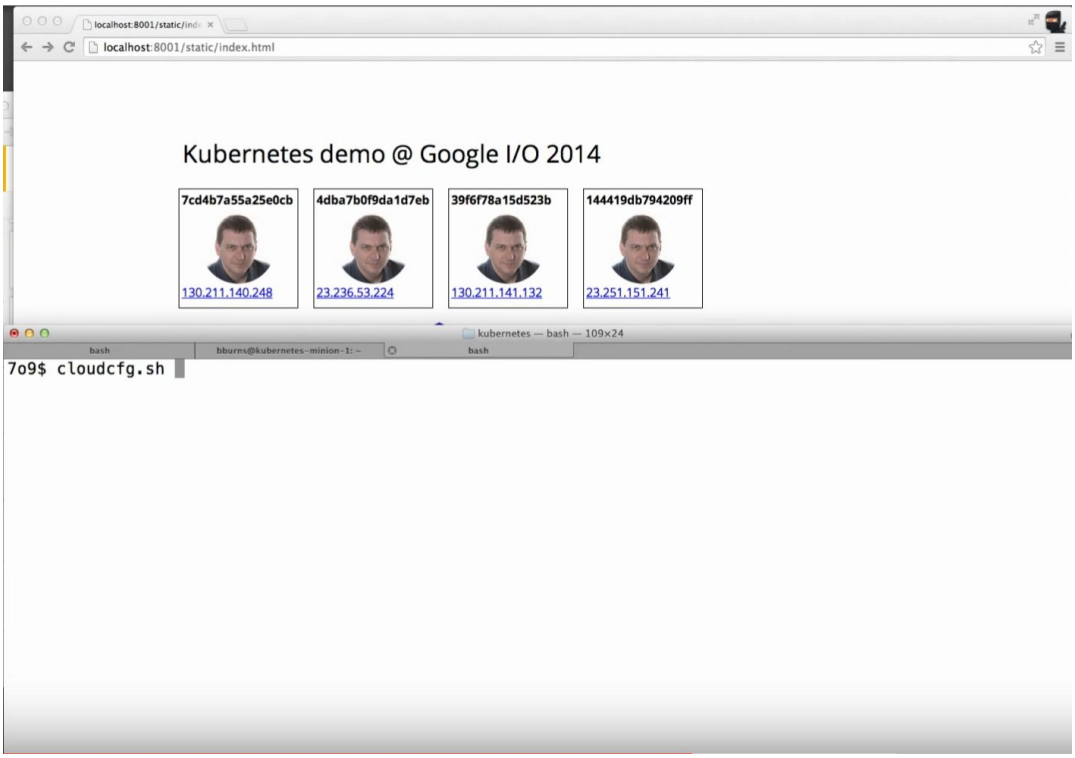
Kubernetes automatically health checks-in on the images and checks if they are being served or not. If any of the images are down, the agent that runs on that VM automatically notices it and restarts the container.

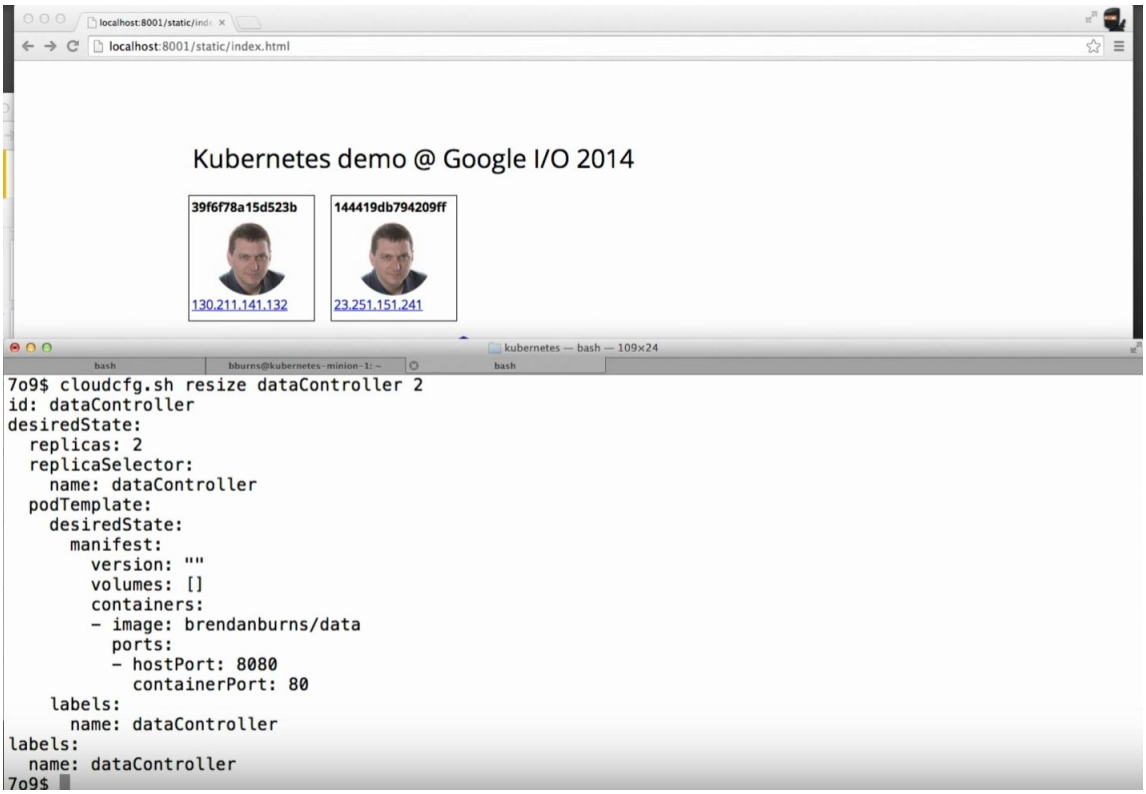




2.Dynamic Resizing

Kubernetes has a RESTful HTTP API that could be used to write arbitrary clients that communicate with this API to create new containers, create new services, create new replication controllers, resize application controllers. It ships with a command line interface that allows you to do these things and with which you can do any kind of orchestration management.





3.Editing and Updating a container

Workflow:

1.Edit the data in the image

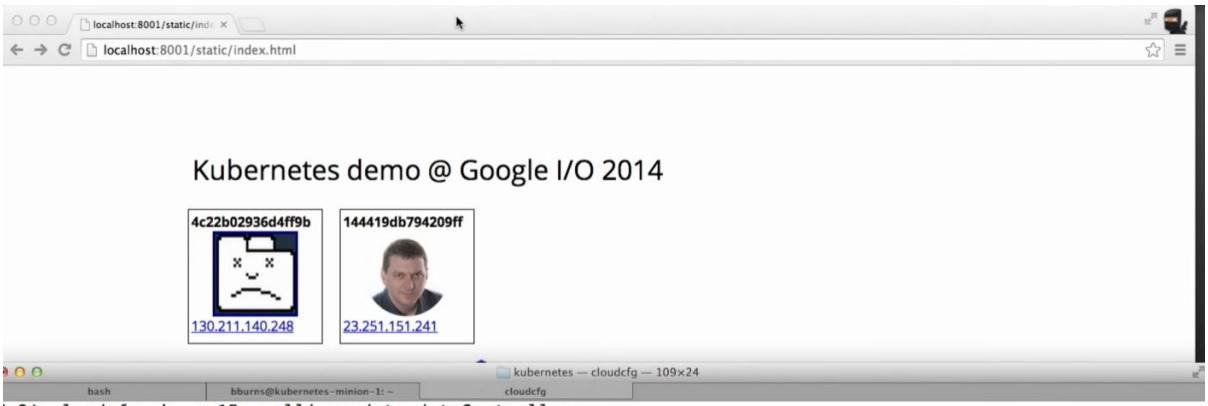
2.Build the image with docker

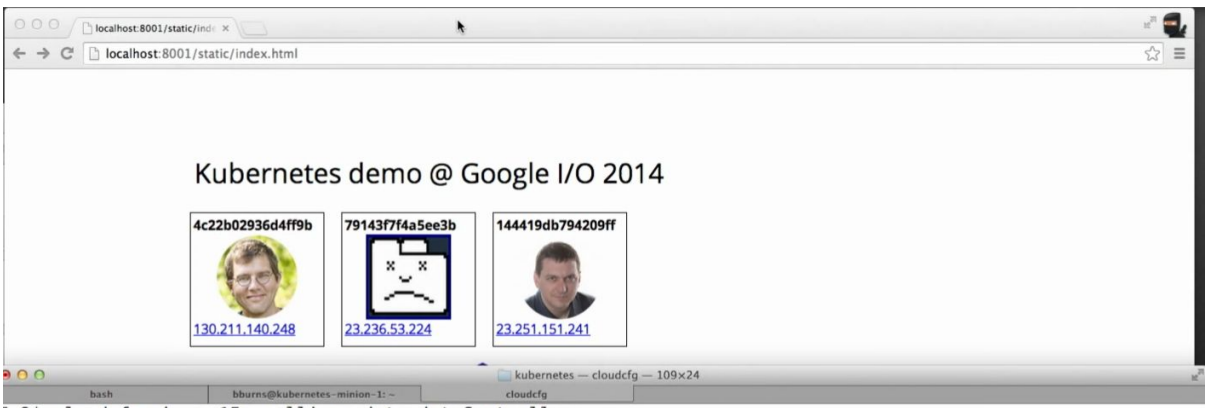
3. Next we push the image

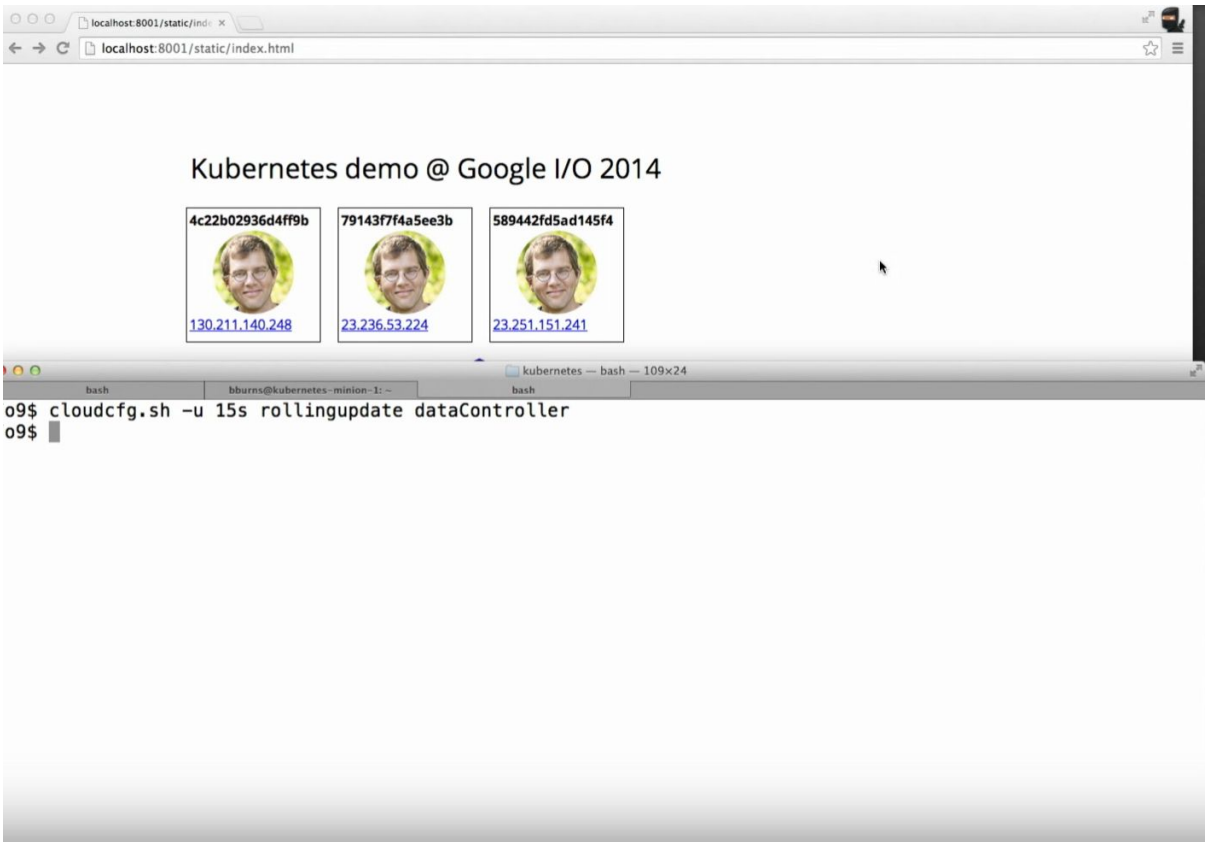
There is complete decoupling of container image from where it is actually get executed, so when we update a container it doesn't change the running service. Application update is separated from running service update. Application update is atomic entity which involves pushing an image somewhere, but service update is another atomic thing which involves running that application on my cluster. So there is much more liability in partitioning both because if application update fails it can be redone and if service update fails it's okay as it can be rolled back to previous version of the application.

4.Rolling Update

Rolling Update means one container at a time it will be brought down and then it is brought up again with update after rebooting it.


localhost:8001/static/index.html
Kubernetes demo @ Google I/O 2014
4c22b02936d4ff9b
130.211.140.248
144419db794209ff
23.251.151.241
kubernetes — cloudcfg — 109×24
bash
bburns@kubernetes-minion-1: ~
cloudcfg
7o9$ cloudcfg.sh -u 15s rollingupdate dataController


localhost:8001/static/index.html
Kubernetes demo @ Google I/O 2014
4c22b02936d4ff9b
130.211.140.248
79143f7f4a5ee3b
23.236.53.224
144419db794209ff
23.251.151.241
kubernetes — cloudcfg — 109×24
bash
bburns@kubernetes-minion-1: ~
cloudcfg
7o9$ cloudcfg.sh -u 15s rollingupdate dataController



Proposed Plan

We create a simple application like a web server, using a cluster of Docker Containers managed for us by Kubernetes. Various scenarios are simulated and Kubernetes features mentioned above are demonstrated.

Requirements

1. Environment - Docker, etcd and go programming language.
2. Web Server - Apache.
3. Back-end - MongoDB.

Expected Outcome

1. In the event of an unexpected unavailability of a Docker container, the Kubernetes should be create an exact Replica and the application should not experience any glitch.
2. If the load on the application is high, Kubernetes should be able to facilitate new containers to handle the extra load.
3. Rolling out updates on this system should be a smooth transition.