I have decided to build the guestbook web application using Kubernetes and Docker

Application will have the following components:

* Web front end
* Redis master for Storage (is used for write operations)
* Redis slave for offloading read operations

I’m using containers because all of the application’s code, libraries, and dependencies are packed together in the container as an immutable artifact.

And using Kubernetes as an container orchestration platform, it allows large numbers of containers to work together in harmony, reducing operational burden. Other aspects it helps in are:

1. Running containers across many different machines
2. Scaling up or down by adding or removing containers when demand changes
3. Keeping storage consistent with multiple instances of an application
4. Distributing load between the containers
5. Launching new containers on different machines if something fails

Steps performed to build the application running at:

http://35.226.72.24:3000

1. Created a Kubernetes Cluster using gcloud
2. Created the Redis master replication controller in the Kubernetes cluster

### Creating the Redis master service

### Creating the Redis slave pods (the Redis read slaves we are creating here are 'replicated' pods)

### Creating the Redis slave service (we want to have a service to proxy connections to the read slaves. the Redis slave service also provides transparent load balancing to clients)

### Create the guestbook pods

### Create the guestbook service (to group the guestbook pods and make the guestbook front end externally visible).

1. Run the guestbook application

**Lessons Learnt**

* They make it much easier to package an application with its required infrastructure. While Docker provide the actual containers, we also need tools to take care of things such as replication and failovers, as well as APIs to automate deployments to multiple machines.
* **Replication Controllers-** management component of Kubernetes. You give it a set of Pods, tell it for e.g. “I want three copies of this,” and it creates those copies on the cluster. It will do its best to keep those copies always running, so if one crashes it will start another.
* **Services**: A service is the single point of contact for a group of Pods. A Service splits the traffic to each of the multiple copies of a web server pod. They are “permanent” while the pods behind them can come and go.
* Pods can start anywhere where the scheduler sees fit in a potentially large cluster obliges administrators to think out their storage in a highly dynamic and replicated way.
* Kubernetes as any other redundant infrastructure, storage can survive a hardware crash and all applications will be able to restart gracefully on a same host in case of failure.