HYBRID CLOUD TECHNOLOGIES: DOCKERS, CONTAINERS AND KUBERNETES

Eshwari H M[1], Prof. Rekha B S[2], Dr. G N Srinivasan[3]

[1]Student, Department of Information Science and Engineering, RV College of Engineering, Bengaluru, India.
[2]Research Scholar, Bharathiar University, Coimbatore, Tamil Nadu, India.
[3]Professor Department of Information Science and Engineering, RV College of Engineering, Bengaluru, India.

ABSTRACT: *Dockers, Containers and Kubernetes has transformed the era of modern software development techniques using cloud technologies or DevOps techniques. Advancements in the cloud technology has transferred the concept of virtualization by providing an alternative for the hypervisors. Docker is used to create, deploy and run any application in any environment by creating an image of the application. Containers are running instances of image that are isolated and have their own set of processes. Kubernetes is an open-source platform that provides a dashboard to manage the deployed containers and also to scale up the resources if required. This paper provides an overview of the modern cloud technologies like docker, containers and Kubernetes and also explains how these techniques can be used to containerize an application.*

Keywords: Dockers, Containers, Kubernetes, Helm Charts, Containerization and Virtualization.

1. INTRODUCTION

Any Software development process involves setting up an End-to-End stack which includes various services like a web server, a database, a messaging system and an orchestration tool like ansible. There will be a lot of issues while integrating these different components. The developer should ensure that all these components have the compatibility with the underlying Operating Systems and also the compatibility of service libraries and the dependencies with the OS. A new developer may find difficulty in setting up the development environment. We also have developers who use different OS in which they are comfortable with. So, there is no guarantee that an application which is built in one environment (development/Test) would work in other environments.

With dockers we can run all these services in different containers with its own libraries and dependencies all on the same Virtual Machines and Operating System but in different containers reducing the overhead caused by virtualization. Containers are isolated environments. They can have their own processes or services, their own networking interfaces like virtual machines except that they all share the same OS kernel.

Kubernetes is nothing more than a container-management framework. It may be docker containers or other alternative containers. Kubernetes orchestrates, administers and forms a communication line between these containers.

2. VIRTUALIZATION vs CONTAINERIZATION

Virtualization lets you run multiple operating systems on a single physical server's hardware, while containerization lets you install multiple applications on a single virtual machine or server using the same operating system.

In case of virtual machines, the services such as node-js, mongo DB, Redis or even another operating system needs a fixed allocated memory. The Hypervisors (software used to create virtual machine) are installed above existing operating system and for each required component a virtual environment will be created using hypervisor. Each virtual machine will have its own operating system inside it. On top of this the required libraries and the dependencies would be running. This overhead causes the higher utilization of the underlying resources and also consume higher disk space as each VM is heavy and is usually in giga bytes (GB) in size. So usually these machine takes more time to boot as it needs the entire OS to boot.

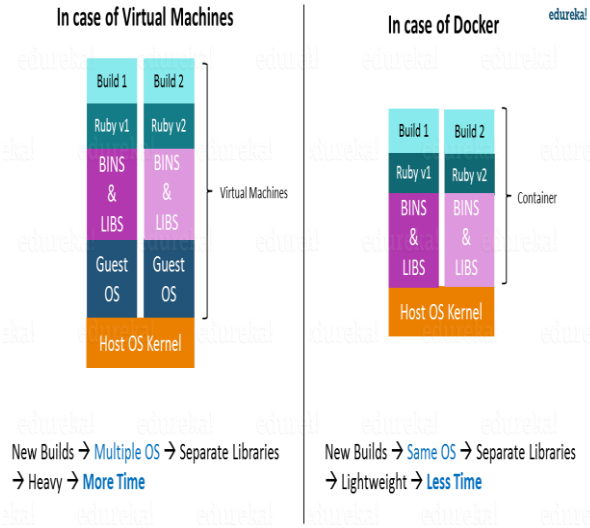

Figure 1: Virtualization vs Containerization

In case of Containerization we have the dockers installed on the Operating system and since the containers which contains the required services such as node-js, mongo DB, Redis or even another operating system runs on top of Dockers there is no need to create a separate VM's for each component. The container of a particular component will have its own libraries and dependencies. This reduces the requirement for RAM and CPU resources and reduces the wastage of memory. Docker Containers are usually light weight and are usually in megabytes (MB) in size. Because of this reason Dockers take less time to boot up.

3. RELATED WORKS

Maciej Gawel and Krzysztof Zielinski et.al [1] this paper discusses elements of the MANO specification with basic mechanisms of Kubernetes containers orchestration. It assesses the degree to which Containerized Network Functions (CNFs) can be handled using Kubernetes platform. To validate the study, an outstanding virtual IP Multimedia Network is used to perform a series of stress and chaos tests on the Kubernetes testbed. The authors concluded that in the area of performance and fault management, Kubernetes can natively meet performance constraints and fundamental requirements imposed by MANO standard. The experiments conducted confirmed that Kubernetes offers important mechanisms for ability to auto scale and self-heal.

Sachchidanand Singh and Nirmala Singh et.al [2] this paper highlights Container based virtualization and Docker's role in shaping Cloud technology's future. The authors says that Container adoption will continue to grow and in the future, the majority of Microservice applications will be based on the containers.

Marcel Großmann and Andreas Eiermann et.al [3] In this paper, a vitality-convincing strategy has been conveyed, Hypriot Cluster Lab (HCL), which changes Dockers moves to keep running on single board PCs fuelled by ARM. In this, the authors demonstrate how HCL works through techniques for large area systems with plenitude and replication between a couple has on different locations. Similarly, they use a job assigned to virtual private LANs to communicate between coursed with encoded correspondence over the Internet. Result shows that tolerant, strong and stable changes are opposed in order to combine two or three self-decisions with HCL to satisfy the QoS needs.

Devki Nandan Jha, Saurabh Garg, Prem Prakash Jayaraman, Rajkumar Buyya, Zheng Li, Rajiv Ranjan et.al [4] this paper presented the experimental research on performance evaluation of Docker containers running a heterogeneous collection of microservices simultaneously. Following CEEM (Cloud Evaluation Experiment Methodology) they conducted a detailed series of experiments to test the interference between containers running either competing or independent microservices. Authors also looked at the consequences of limiting a container's resources by specifically defining the cgroups.

4. DOCKER AND CONTAINER

Docker is a collection of (PaaS) products which use virtualization at the OS level to deliver software in packages called containers. Containers are separated from each other and they contain their own applications, libraries and configuration files; they can interact through well-defined channels with each other. All containers are operated by a single operating system kernel, and are therefore lighter than virtual machines. Docker Engine is the program which hosts the containers. Docker can package an application in a virtual container that can run on any Linux server. This helps to provide flexibility and portability that allows the application to run at different locations, whether on-site, in a public cloud or in a private cloud.

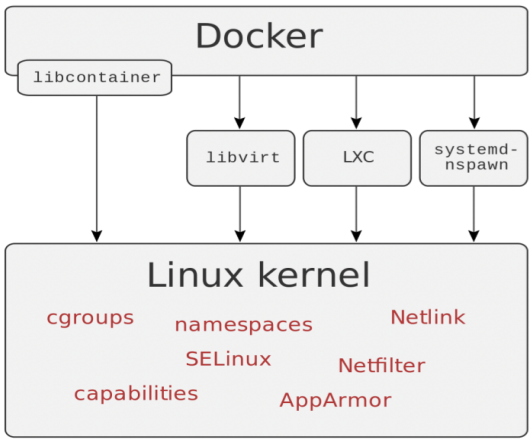

Figure 2: Docker with Interfaces Using Linux Kernel

Docker uses the Linux kernel resource isolation features (such as cgroups and kernel namespaces) and a union-capable file system (such as OverlayFS) to enable containers to run within a single Linux instance, avoiding the overhead of virtual machine start up and maintenance. Because the Docker containers are lightweight, several containers can be run simultaneously by a single server or virtual machine. Support for namespaces by the Linux kernel mostly isolates the operating environment view of an application, including process trees, network, user IDs and mounted file systems, while memory and CPU resource limits are provided by kernel groups. Figure 2. Shows Docker interfaces using Linux kernel.

Docker uses images to create containers. An image is a package or template which is used to create one or more containers, containers are running instances of image that are isolated and have their own set of processes. Containers are fast, reliable and makes use of less space. For example, if an application requires latest version of ubuntu, then we can pull the image from the docker hub using the command "docker pull ubuntu" which will fetch the latest ubuntu from the docker hub if the image is not present in the local repository. Figure 3 shows the working of docker. Developer issues the docker commands using the command line. If the required image is already loaded, then docker uses the local version else the image will be downloaded from the hub.

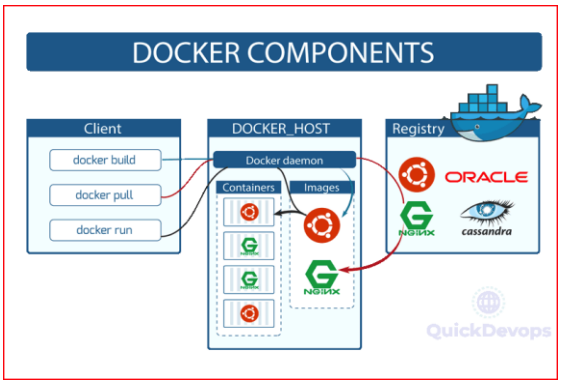

Figure 3: Working of Docker.

In order to create a container of an image we issue the "docker run" command. To list the available images in the local repository issue "docker images" command. Figure 4 shows all the available images. To know the list of running containers execute "docker ps" command. Figure 5 shows all the running containers.

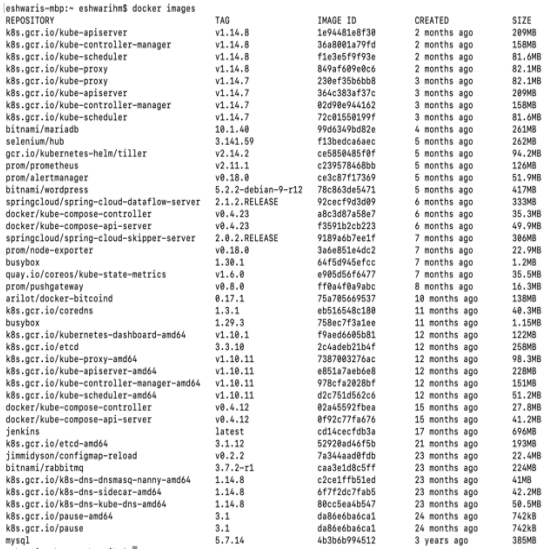

Figure 4: Available images

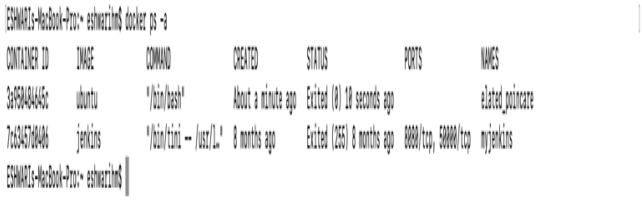

Figure 5: All Running Containers

Basic Docker Commands:

- docker ps
- docker run ImageName

- docker start ContainerName/ID
- docker stop ContainerName/ID
- docker pause ContainerName/ID
- docker unpause ContainerName/ID
- docker stats ContainerName/ID
- docker kill ContainerName/ID
- docker rm ContainerName/ID
- docker history ImageName/ID

To create an image of an application Docker Files are used. It is a text file with instructions to build an image Automation of Docker Image Creation. The "docker build" command builds the image and the "docker push" command pushes the created image to the docker hub.

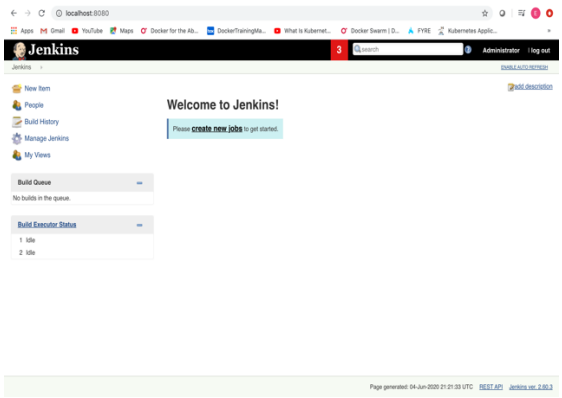

Figure 6: Jenkins Page

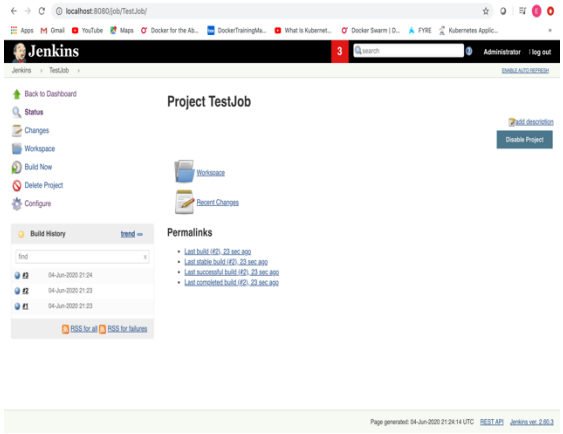

Figure 7: A Jenkins Job

Figure 6 and figure 7 shows the Jenkins application that is running on a container as Administrator. Jenkins application is used to continuously test and build the software projects which makes it easy for the developer to integrate the changes in the project. By using dockers and containers the web application could be started easily using the image available in the hub.

5. KUBERNETES

Kubernetes is a scalable open-source, extensible framework for managing containerized workloads and services that enables both declarative configuration and automation. It has a broad ecosystem, rising rapidly. Services, support and software offered by Kubernetes are widely accessible. Containers are a good way to get your applications bundled and run. You need to manage the containers running the applications in a production environment, and ensure that there is no downtime. This is provided by Kubernetes. It takes care of scaling and failover, as the framework offers templates for deployment, and more. Kubernetes for instance can easily manage a system canary deployment.

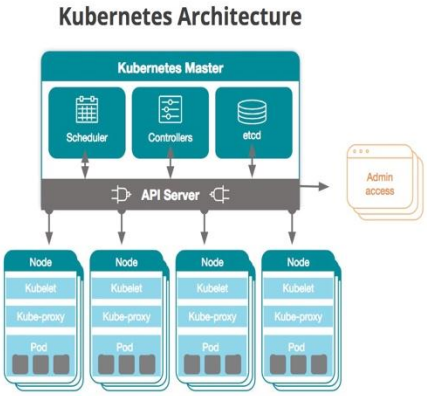

Figure 8: Kubernetes Orchestration System Architecture

Figure 8 shows the Kubernetes architecture.

Node: A node is also a worker machine in which containers are deployed by Kubernetes. Since the application that is running on a node may fail, we need to have more than one node.

Cluster: A cluster is a set of nodes grouped together. So if one node fails, still the application will be accessible from other nodes. Moreover, having multiple nodes helps in sharing load of the website.

Kubernetes Master: Master node is a machine (physical/virtual) in which Kubernetes is installed and configured as master. The master watches over the cluster of nodes and is responsible for actual orchestration of worker node in the cluster. They help in managing the cluster. When a node fails, the workload of the failed node is moved to other worker nodes using master.

API-Server: API server acts as a front end for Kubernetes. The users, management devices, command line interfaces all talk to the API to interact with the Kubernetes cluster.

ETCD Service: ETCD is a distributed, reliable key-value store used by Kubernetes to store all data used to manage the cluster. They are responsible for implementing the locks on the cluster to ensure that there are no conflicts within the masters.

Kubelet Service: Kubelet is an agent that runs on each node in the cluster. The agents are responsible to make sure that the containers are running on the cluster as expected.

Controller: The Controllers are the brain behind the Orchestration. They are responsible for noticing and responding when nodes, containers, or an end-point goes down. They make decisions to bring up new containers in such cases.

Scheduler: The scheduler is responsible for distributing the work on containers across multiple nodes. It looks for newly created containers and assigns them to nodes.

Pod: Kubernetes does not deploy the container directly on a worker node. The containers are encapsulated into a Kubernetes object known as Pods. A pod is a single instance of an application and is the smallest object that we can create in Kubernetes.

Minikube is a miniature version of Kubernetes that can be installed on the client to access Kubernetes. The "minikube start" command starts the kubernetes cluster. Figure 9 shows the result of the command. Kubernetes uses yaml files as input for creating objects such as pods, services deployments.

```
eshwaris-mbp:~ eshwarihm$ minikube start
⚠  There is a newer version of minikube available (v1.6.1).  Download it here:
https://github.com/kubernetes/minikube/releases/tag/v1.6.1

To disable this notification, run the following:
minikube config set WantUpdateNotification false

😄  minikube v1.2.0 on darwin (amd64)
💡  Tip: Use 'minikube start -p <name>' to create a new cluster, or 'minikube delete' to delete this one.
🏃  Re-using the currently running virtualbox VM for "minikube" ...
⌛  Waiting for SSH access ...
📶  Configuring environment for Kubernetes v1.15.0 on Docker 18.09.6
🔄  Relaunching Kubernetes v1.15.0 using kubeadm ...
⌛  Verifying: apiserver proxy etcd scheduler controller dns
🏄  Done! kubectl is now configured to use "minikube"
eshwaris-mbp:~ eshwarihm$
```

Figure 9: Result of minikube start

To get all the kubernetes objects such as pods, services, deployments etc. that is running on a cluster, use "kubectl get all" command. Figure 10 shows the output of the command.

```
eshwaris-mbp:Kubernetes eshwarihm$ kubectl get all
NAME                                                    READY   STATUS              RESTARTS   AGE
pod/loitering-jackal-ebl-chart-deployment-7779b77c96-trtrd   1/1     Running             0          8d
pod/loitering-jackal-mongodb-58ccd569b4-b4w77           1/1     Running             0          8d
pod/measly-abalone-77cc7997c5-ktgjp                     1/1     Running             0          7d23h
pod/myapp-deployment-776757f64f-flr66                   0/1     ContainerCreating   0          17s
pod/myapp-deployment-776757f64f-j7zkg                   0/1     ContainerCreating   0          17s
pod/myapp-deployment-776757f64f-ppkvp                   0/1     ContainerCreating   0          17s

NAME                                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/kubernetes                      ClusterIP   10.96.0.1       <none>        443/TCP          8d
service/loitering-jackal-ebl-chart-service   NodePort    10.110.219.4    <none>        8080:31167/TCP   8d
service/loitering-jackal-mongodb        ClusterIP   10.110.161.232  <none>        27017/TCP        8d
service/measly-abalone                  NodePort    10.108.83.51    <none>        80:30254/TCP     7d23h

NAME                                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/loitering-jackal-ebl-chart-deployment   1/1     1            1           8d
deployment.apps/loitering-jackal-mongodb            1/1     1            1           8d
deployment.apps/measly-abalone                      1/1     1            1           7d23h
deployment.apps/myapp-deployment                    0/3     3            0           17s

NAME                                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/loitering-jackal-ebl-chart-deployment-7779b77c96   1         1         1       8d
replicaset.apps/loitering-jackal-mongodb-58ccd569b4           1         1         1       8d
replicaset.apps/measly-abalone-77cc7997c5                     1         1         1       7d23h
replicaset.apps/myapp-deployment-776757f64f                   3         3         0       17s
```

Figure 10: The kubernetes objects.

Figure 11 shows the dashboard visualization of the kubernetes objects running on a container.

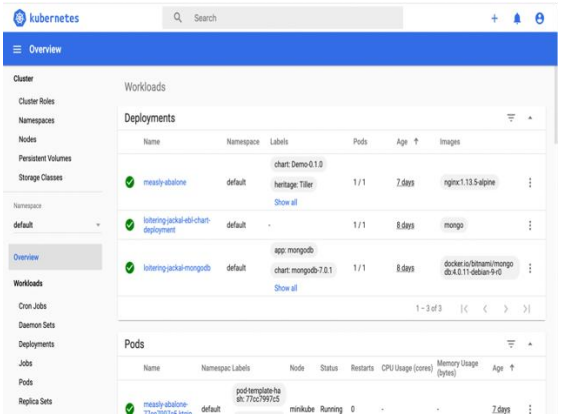

Figure 11: GUI to manage clusters.

Figure 12 shows the yaml file to deploy Jenkins on the kubernetes cluster. It is of kind deployment. Figure 13 is the yaml file that deploys a service.

Services allow interaction between different components within and outside the application.

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: jenkins
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: jenkins
    spec:
      containers:
        - name: jenkins
          image: <dockerhub_user>/jenkins-master
          env:
            - name: JAVA_OPTS
              value: -Djenkins.install.runSetupWizard=false
          ports:
            - name: http-port
              containerPort: 8080
            - name: jnlp-port
              containerPort: 50000
          volumeMounts:
            - name: jenkins-home
              mountPath: /var/jenkins_home
      volumes:
        - name: jenkins-home
          emptyDir: {}
```

Figure 12: YAML file

```yaml
apiVersion: v1
kind: Service
metadata:
  name: jenkins
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 8080
  selector:
    app: jenkins

---

apiVersion: v1
kind: Service
metadata:
  name: jenkins-jnlp
spec:
  type: ClusterIP
  ports:
    - port: 50000
      targetPort: 50000
  selector:
    app: jenkins
```

Figure 13: YAML file for Services

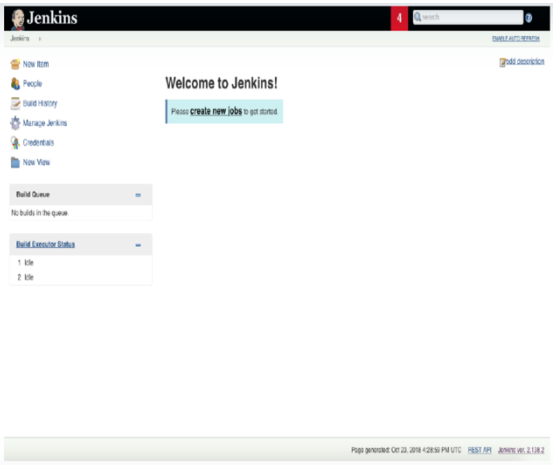

Figure 14: Jenkins running on Kubernetes

6. CONCLUSION

Dockers, Containers and Kubernetes work at different levels [11]. Docker uses containerization technology to create various containers. Kubernetes provides a user interface to manage these containers. Using Kubernetes, we can scale up or scale down the resources at a very fast rate. This paper gives an overview of modern cloud technologies and also shows some of the outputs obtained by using dockers, containers and Kubernetes. This paper also talks about virtualization and containerization and gives an overview of the basic docker, container and Kubernetes commands that can be used to containerize the application.

7. REFERENCES

1) Maciej Gawel and Krzysztof Zielinski "Analysis and Evaluation of Kubernetes based NFV management and orchestration", 12th International Conference on Cloud Computing (CLOUD), IEEE, 2019.

2) Sachchidanand Singh and Nirmala Singh "Containers & Docker: Emerging Roles & Future of Cloud Technology", IEEE, 2016.

3) Marcel Großmann and Andreas Eiermann, "Automated Establishment of a Secured Network for Providing a Distributed Container Cluster", 2016 28th International Teletraffic Congress - The First International Conference in Networking Science & Practice.

4) Devki Nandan Jha, Saurabh Garg, Prem Prakash Jayaraman, Rajkumar Buyya, Zheng Li, Rajiv Ranjan, "A Holistic Evaluation of Docker Containers for Interfering Microservices", International Conference on Services Computing, IEEE, 2018.

5) M. Fazio, A. Celesti, R. Ranjan, C. Liu, L. Chen, and M. Villari, "Open Issues in Scheduling Microservices in the Cloud," IEEE Cloud Computing, vol. 3, no. 5, pp. 81–88, 2016.

6) Douglas Bourgeois, David Kelly, Thomas Henry, "Cloud Native Applications- The Intersection of Agile Development and Cloud Platforms", Deloitte Touche Tohmatsu Limited, 2016.

7) Ms. Shalini Joshi, Dr. Uma Kumari, "Load Balancing in Cloud Computing: Challenges&Issues",2016 2nd International Conferenceon Contemporary Computing and Informatics (ic3i).

8) Mahfooz Alam and Zaki Ahmad Khan, "Issues and Challenges of Load Balancing Algorithm in Cloud Computing Environment", Indian Journal of Science and

Technology",Vol10(25), DOI:10.17485/ijst/2017/v10i25/105688, July 2017.

9) Charanjeet Singh and Amandeep Kaur, "A review on different approaches of load balancing in cloud computing", International Journal of Science and Research (IJSR), v5, 2013.

10) ZHANG Yan-huaa, Feng Leia, Yang Zhia, "Optimization of Cloud Database Route Scheduling Based on Combination of Genetic Algorithm and Ant Colony Algorithm", Science direct, Procedia Engineering 15 (2011), pp. 3341 – 3345.

11) Lecture slides of "Cloud Computing for Network Engineers" by Dr. Scott Kingsley, SMU, 2018.

12) "Docker." [Online]. Available: https://www.docker.com.

13) Lecture slides of "Cloud Computing for Network Engineers" by Dr. Scott Kingsley, SMU, 2018.