**Private Cloud Deployment with Docker and Kubernetes**

**by**

**Sean Patrick Sanders**

# Project submitted in partial fulfillment of the requirements for the

# degree of Master of Science in Networking and System Administration

**Rochester Institute of Technology**

**B. Thomas Golisano College**

**of**

**Computing and Information Sciences**

**Department of Information Sciences and Technologies**

# 5/10/16

**~ MS Project Committee ~**

Name Signature Date

Charles Border

Chair

Garret Arcoraci

Committee Member

**Abstract:**

Private cloud computing is important to many businesses because of the need for increased security and data governance. The need for automation, orchestration, security, and ease of deployment is becoming increasingly important when creating a private cloud computing infrastructure. Many open source tools can aid in private cloud infrastructure development, deployment, orchestration, and automation. There are two tools that aid in the process of creating a private cloud based containerized and virtualized environment called Docker and Kubernetes. Docker creates what is known as a containerized environment. Containers enable a user to deploy a single application service inside the container. Kubernetes allows you to automate, orchestrate, and deploy the environment on demand. This project will demonstrate how to create a private based infrastructure using Docker, Kubenretes, and VirtalBox. Kubernetes also includes a Google communications API. This is necessary for the system to properly configure, deploy, and monitor the private cloud infrastructure.

**Introduction:**

According to Forbes, Amazon increased its cloud based revenue by 69% this year up to $7.88 billion. The spending on the cloud computing infrastructure is estimated to grow from $38 billion to $173 billion by 2026. AWS Cloud computing is estimated to be expensive to new businesses and sometimes results in the business resorting to a private cloud based solutions. This results in the cloud computing becoming too expensive for small businesses and a different solution is then required. The greatest cost to the business is usually due to the way that the cloud computing company charges its customers. For example, Amazon charges customers on a per hour bases and by the resources that are used. They perform costs analysis mainly on an on demand cost model. Kubernetes solves the issue through the use of open source containers and virtualization accomplished via Docker or their cloud based infrastructure. The cloud based infrastructure is costly and results in many resorting to the Docker implementation. Docker and Kubernetes solves this issue by allowing the use of open source images and containers thus creating a private cloud infrastructure locally on the network.

Many businesses want to have their own private cloud based infrastructure and services. Docker with Kubernetes allows this by giving the tools to accomplish such a task. Docker allows the business to create a containerized environment. Containers are a place where an application can be stored while virtual machines have an image/OS and can run multiple applications. Docker utilizes VirtualBox which can create a virtualized environment consisting of one or many virtual machines. A business might want to utilize the best of two worlds and use a solution such as Kubernetes which combines the two into one solution as shown in Figure 1.

Below are the important things that a business might want to do with Kubernetes:

* Host a local WordPress blog
* Run a high performance http server (nginx)
* Run a form for data analytics
* Allow resiliency of applications through the replication controller process command
* Ensure every application and container is properly running through the use of kubectl checks command

Kubernetes when joined with Docker can offer a scalable environment that adapts effortlessly to the needs of the business or individual using Docker’s containerized environment. A containerized environment is an environment that allows you to run an operating system like an application and this reduces the overhead on your own personal computer or server. This results in the containers being more efficient than virtual machines. Docker is used most commonly to create a containerized environment. Kubernetes readily allows you to manage the containerized environment and scale it either larger or smaller, depending on the needs of the business or individual. Figure 1 below explains how Kubernetes Works.

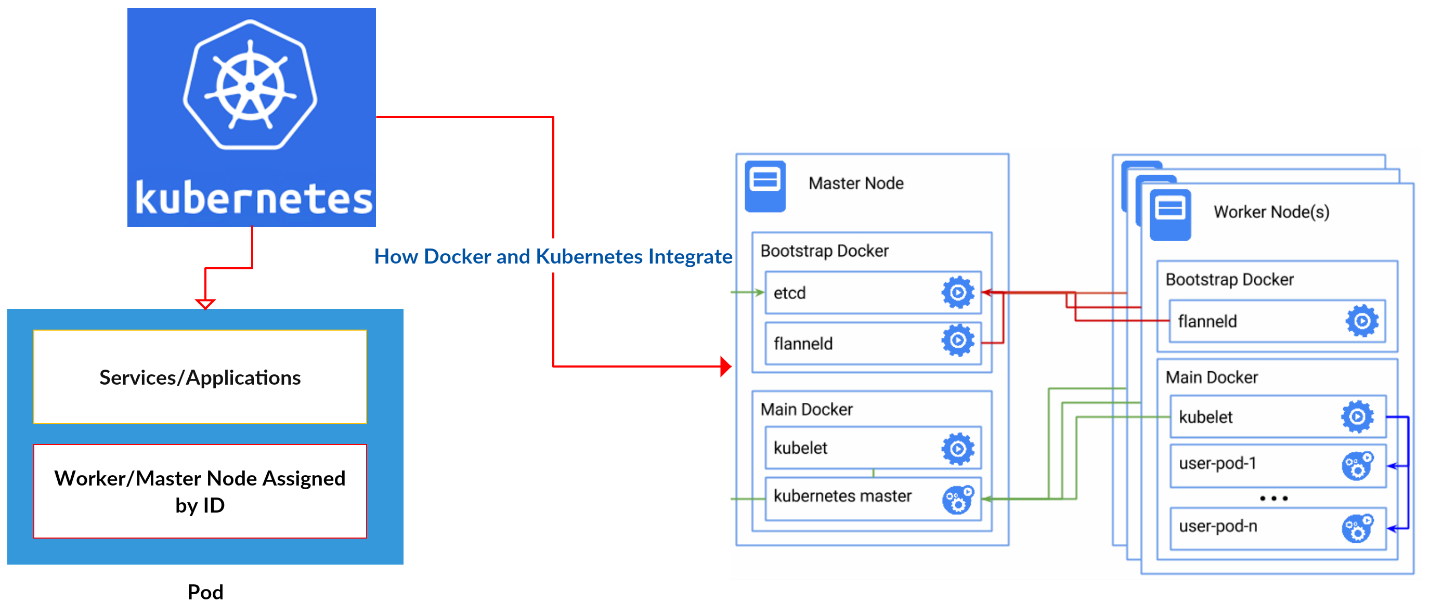


Figure 1: How Kubernetes Works

Project plan

The project plan entails use of my personal laptop and a Windows server to run Docker. With Docker I will use a Kubernetes image that will be a master node and then run multiple API services in order to have Kubernetes work. Master nodes allow you to control the other nodes (other nodes are referred to as minions or worker nodes). The pods and nodes need to be scaled on demand so I will use Kubernetes to do this through the master node (Note: Pods are used for services, applications, and can contain nodes or master nodes). In addition, I will make sure that minimal performance loss occurs. This requires me to use a tool or application to check the containers' performance on both my laptop and the server.

Docker has a built in stats command, but I am not sure if the stats show the actual state of the container. This requires me to search for and use another tool to test against the Docker stats feature. Next, I will perform common tasks as an administrator to test the functionality of Kubernetes; this step requires performance monitoring. One goal is to decrease the load on the containers when performing the steps described. Figure 2 provides a layout of what I plan to do from a networking perspective. Figure 3 provides the configuration commands to setup the environment.

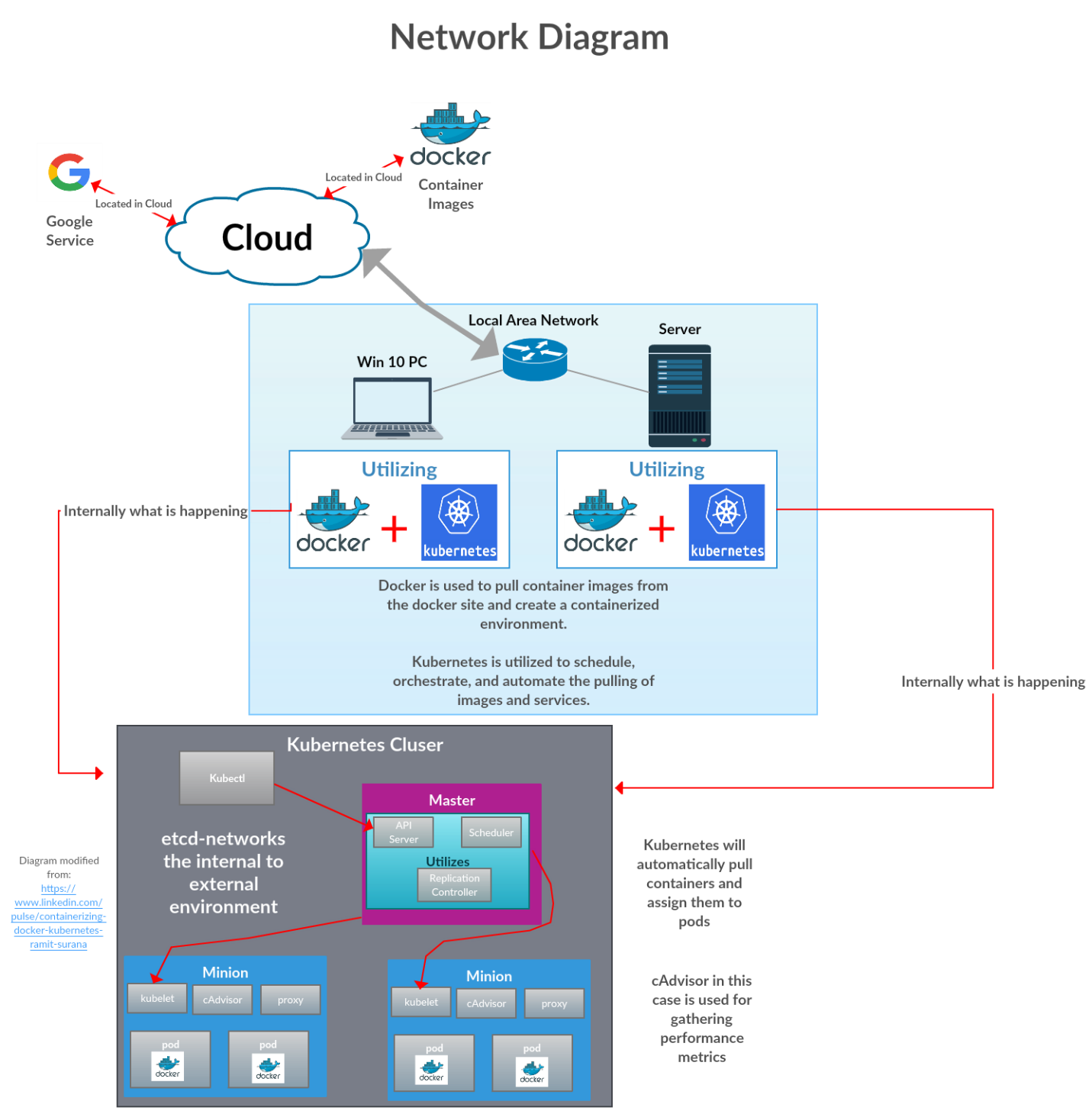


Figure 2: Network Infrastructure Diagram

After creating the environment, the next phase of the project is to test the scheduler mechanism using the kubectl command. This will require me to use a YAML file to properly initiate the use of the scheduling service. Afterward, I will set the minimum requirements for both containers and applications to target the important aspect of automation and scheduling. For example, this is a command that could begin the autoscaling of the pod: **kubectl autoscale rc foo --min=2 --max=5 --cpu-percent=80.** By setting up the minimum and maximum limits you are requiring Kubernetes to constantly monitor the pod/pods for any changes. Although this could create some performance issues, this usually is not the case because the CPU usage for the pod is limited to only eighty percent, which is usually a good limit for most pods. All of the previously mentioned steps will also be replicated on the server and laptop, first on the laptop and then the server.

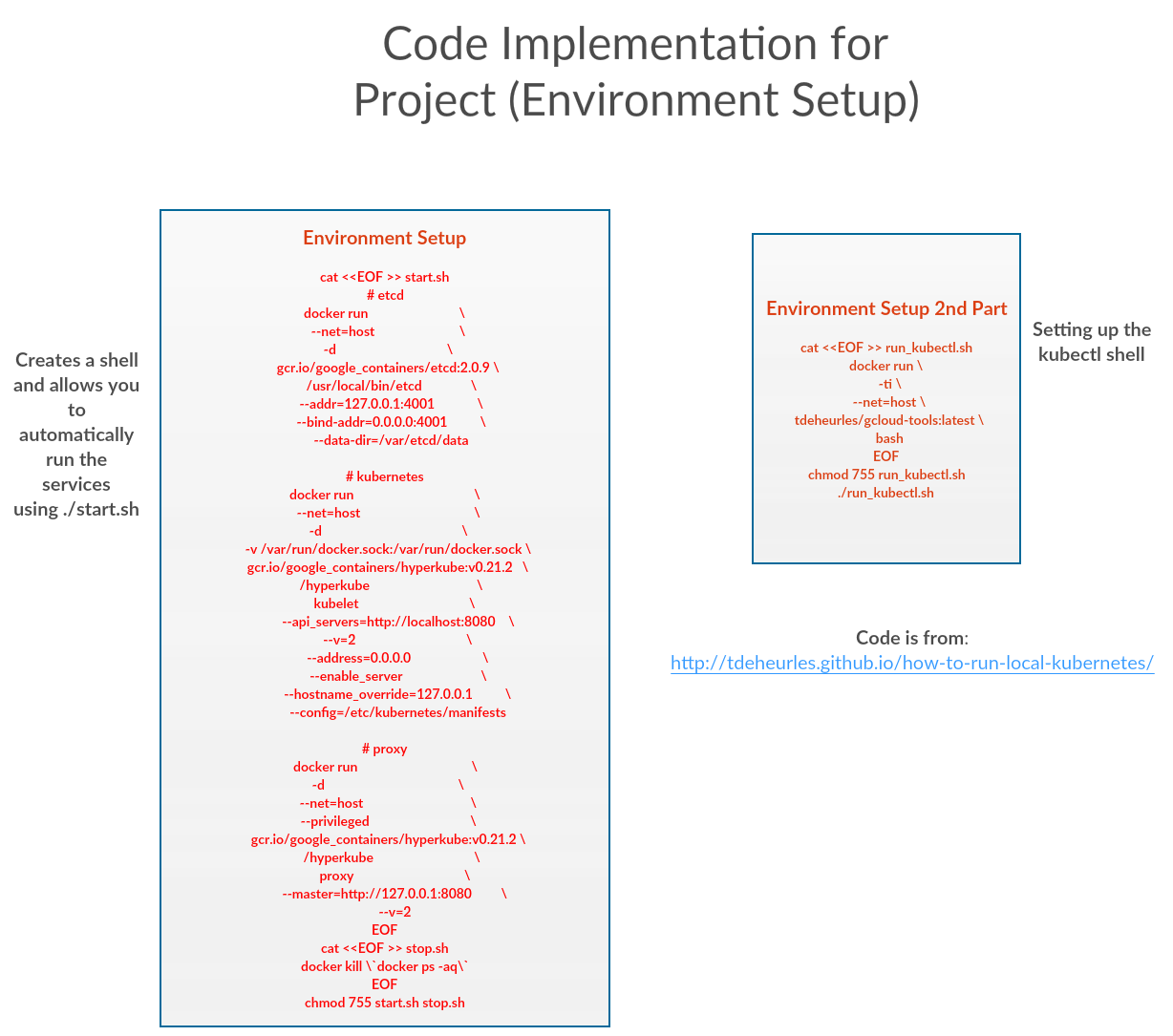


Figure 3: Environment Setup

Project objective

The reason for pursuing the project is to show how to automate, orchestrate, and create a private based cloud infrastructure using Docker and Kubernetes. The creation of a private cloud based infrastructure will ensure the business has full control of the data and rights to the data. The other important aspect is security which the business now can control compared to handing it off to a cloud provider. Dev ops professionals particularly can benefit from a tool such as Kubernetes. Dev ops can expand the number of containers or have a service up and running on demand without using the amount of resources that would be necessary with a virtual machine. For example, in my project I can easily create a nginx service, nginx virtual machine, and have it simultaneously run rapidly while scaling it. The nginx server is used to provide http service and is considered a high performance server.

From a networking perspective, another benefit of Kubernetes is the integration of flannel to allow control of the environment locally. Flannel uses etcd to map subnets to real IPs, which creates a mesh network and is an alternative to software defined networking. Flannel, created by Kubernetes, is currently being used in other projects and has grown in popularity. And since Kubernetes allows for scheduling and automation, network administrators are free to divert their attention towards the management of other sections of the network. This usually results in increased efficiency of network management, for Kubernetes creates a sort of self-healing containerized environment.

**METHODOLOGY**

Steps taken in the project

What I plan on doing for this project is to use four services that will run in Docker. These include the API service, scheduler, controller-manager, and kubelet. Each of these services will need to communicate with the Kubernetes service. I will use a container that has gcloud sdk installed, enabling me to use the kubectl command. After this step, I will install a nginx container and expose the service. Finally, I will need to perform other tasks commonly done by a network administrator; namely, expanding the number of containers through a replication process.

Testing the orchestration process will also require me to set the following limits: the minimum number of pods to be run, minimum applications, and the metrics to be used. The minimum requirements will in turn also hit on the important point of automation; it automatically will spin up a pod if the number of pods falls below the minimum limit. This will also ensure the service is never lost, leading to resiliency of the service. For scheduling I will have to use the kubectl to enable a YAML file to run. The YAML file allows for the running of scheduled tasks. Figure 4 shows an example of how a YAML file is scripted. In addition, the metrics of each of these containers need to be checked; this requires the use of Docker stats and another tool to compare the stats. Preferably, I will utilize cAdvisor for metrics gathering.

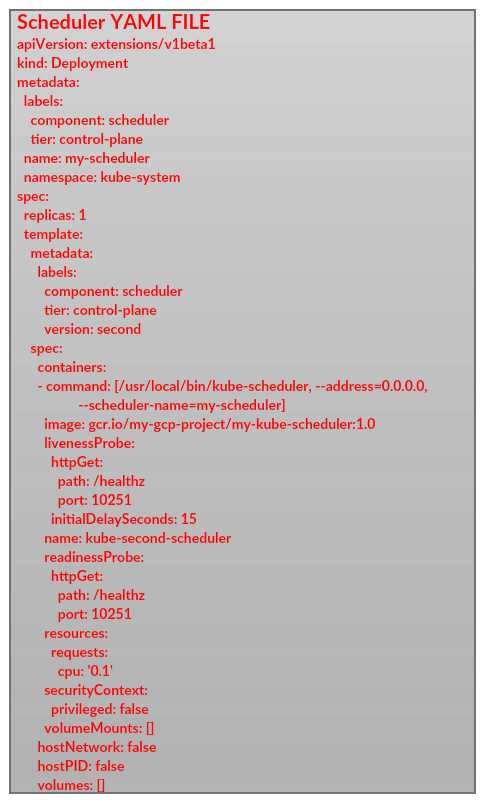


Figure 4: YAML FIle for Scheduler

**LIT REVIEW**

Similar projects that relate to the current one

The scholarly paper titled *A Reference Architecture for Real-Time Microservice API Consumption* mentions a Docker implementation that uses an API micro service for creating a containerized close based architecture. The use of a NoSQL database was utilized to create a ‘livequery’ technique to notify web-based clients of new updates. The NoSQL database was utilized in the authors’ research to scale the database, while REST APIs were used to allow clients to receive responses from the REST ‘microservices.’ Based on the subscription of the client the JSON data was returned through the microservice. The project explained in this article proves the importance of the need for private services and containerized environments on demand for application testing.

Similarly, *Executing Informal Processes* mentions how the automation of the research, decision, development, and maintenance processes are becoming critical for many businesses. This paper proposed the automation of the resource-centric informal process in two steps. The first step was integrating the resources of informal processes and the second step was to execute them. The researchers describe how to refine the Business Process Management Life Cycle through the automation process. The InProXec Method was introduced and includes four phases:

“The first phase of our method contributes to the creation of functional modeling and execution environments of resource-centric informal processes. This phase is followed by a modeling phase, in which business experts create informal process models. Hereafter, the third phase aims for producing executable informal process models. These models are then initialized in the fourth phase.” [11]

During the informal process the automation of services is necessary and for this the automation of a MediaWiki could be used. MediaWiki is a PHP based open source package used for collaboration and sharing. For phase two the importance of modeling and information gathering is emphasized. The third and fourth phases are essentially summed up as the automated execution of the informal process. The Docker component comes into play when talking about ‘Apification’ and how Docker containers can be used for API implementations. The automation of the PHP based service could have also been used in the Docker and Kubernetes environment, providing an on demand containerized environment. The importance of this results closer to the idea of private cloud based infrastructures.

In the article *Borg, Omega, and Kubernetes* a discussion was provided about how Kubernetes came into existence and why. The original infrastructure was Borg, which was the first content management system. The main goal of this infrastructure was to manage the long-run services and batch processing tasks in the containerized environment. The increase in source utilization was created by sharing machines between applications, which resulted in decreased costs.

Afterward, Omega was developed to improve the software engineering aspect of Borg. The article explains: “Omega stored the state of the cluster in a centralized Paxos-based transaction-oriented store that was accessed by the different parts of the cluster control plane (such as schedulers), using optimistic concurrency control to handle the occasional conflicts” [10]. Next was the development of Kubernetes, an open source internal system not specific to Google. At the core Kubernetes has a persistent store that scans the containerized environment for changes to objects. Kubernetes was geared primarily towards developers writing applications that run in a clustered environment, which shows its relevance to this project. The cost aspect was defined in this article in regards to cloud based solutions.

The article *Docker: Lightweight Linux Containers for Consistent Development and Deployment* describes the issue of programming and why a prepackaged on-demand infrastructure is necessary. The isolation between the host and container is another concept discussed in the article that relates to Docker. I feel that this is part of the reason why the containerized environment does not affect the performance on a personal computer as much as it does on a server environment. Containers virtualize at the operating system level, which leads to more efficient use of the container as compared to the virtualization of operating systems occurring at the hardware level. As many programmers know, hardware tends to be less efficient than software. The on demand model discussed in the article shows that images from the Docker server can be pulled at any time you want, as long as an internet connection exists.

In *Docker Containers across Multiple Clouds and Data Centers* the topics of scheduling and mobility were discussed. The limitation of Docker is that most applications are geared towards managing containers on a single site. The use of CometCloud was discussed as a solution to allocate and relocate resources. This article shows that the deployment model can be used with multiple providers, but the author mentions this might not be the best strategy.

The single provider deployment model is dangerous because you can be locked in and no longer have the ability to test other vendors. When choosing multiple vendors, you have to consider resource discovery/coordination, container scheduling/placement, dynamic adaptation/optimization, and the discovery/coordination can be problematic because you need to determine the main goal for the containerized environment. For example, if you are trying to solve the issue of privacy and legality, scheduling and placement may be problematic, especially with multiple vendors. Decisions about how to migrate the containerized environment as well as the life cycle of the containers are of high importance. The adaptation and optimization of containerized environments can be tricky, but the authors proposed using C-Ports because the integration of multiple cloud providers that containers reside in can be minimized.

**DELIVERABLES**

Presentation of results

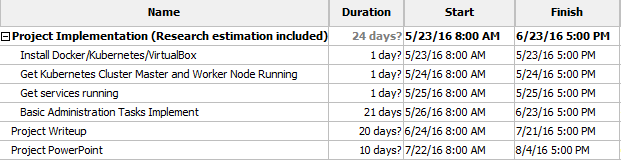
After completing this project, I plan on producing a project report in conjunction with a PowerPoint presentation. The PowerPoint presentation will supply an in-depth and visual analysis of the project, including a demo. The demo will show how to use Kubernetes with Docker, along with the steps necessary to run Kubernetes in a containerized environment to create a private cloud based infrastructure. I will also show how the master node can administrate the network and worker nodes. In addition, I will highlight the important concepts of automation, scheduling, and the orchestration of containers through the combined use of Kubernetes and Docker. Lastly, the demonstration and presentation will explain how the monitoring commands of Kubernetes allows dev ops and network administrators to control their containerized environment.

**PROJECT PLAN**

Meta-analysis of the project, including the timeline and perceived level of difficulty for each step

|  |  |  |
| --- | --- | --- |
| Docker Containers Monitoring | Hard | * Determining the most effective monitoring tool and learning how to use it is necessary |
| Kubernetes Implementation | Moderate | * Familiarity with the kubectl commands. * Scaling the environment is required environment * Scheduling and Orchestration is required * Automation using YAML file |
| Docker and Kubernetes setup | Easy | * One to two days necessary to install everything on laptop * Duplication of laptop to server should be easy |

Schedule



**Sources**

1. Abdelbaky, M., J. Diaz-Montes, M. Parashar, M. Unuvar, and M. Steinder. “Docker Containers across Multiple Clouds and Data Centers.” In 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), 368–71, 2015. doi:10.1109/UCC.2015.58.
2. “An Intro to Google’s Kubernetes and How to Use It - CenturyLink Cloud Developer Center.” Accessed May 5, 2016. https://www.ctl.io/developers/blog/post/what-is-kubernetes-and-how-to-use-it/.
3. “Kubernetes - Kubectl Scale.” Accessed May 5, 2016. http://kubernetes.io/docs/user-guide/kubectl/kubectl\_scale/.
4. “Kubernetes - What Is Kubernetes?” Accessed May 5, 2016. http://kubernetes.io/docs/whatisk8s/.
5. Lee, Bryan. “Using cAdvisor to Monitor Docker Containers.” Tutum Blog, August 7, 2014. https://blog.tutum.co/2014/08/07/using-cadvisor-to-monitor-docker-containers/.
6. Rouse, Margaret. “What Is Container-Based Virtualization (operating System-Level Virtualization)? - Definition from WhatIs.com.” SearchServerVirtualization. Accessed May 5, 2016. http://searchservervirtualization.techtarget.com/definition/container-based-virtualization-operating-system-level-virtualization.
7. Surana, Ramit. “Containerizing Docker on Kubernetes !!” LinkedIn Pulse, September 16, 2015. https://www.linkedin.com/pulse/containerizing-docker-kubernetes-ramit-surana.
8. Vaughan-Nichols, Steven J. “What Is Docker and Why Is It so Darn Popular?” ZDNet. Accessed May 5, 2016. http://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/.
9. “What Is Docker?” Docker, May 14, 2015. https://www.docker.com/what-docker.
10. Burns, Brendan, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. “Borg, Omega, and Kubernetes - ACM Queue.” Accessed May 14, 2016. http://queue.acm.org/detail.cfm?id=2898444.
11. Sungur, C. Timurhan, Uwe Breitenbücher, Frank Leymann, and Johannes Wettinger. “Executing Informal Processes,” 1–10. ACM Press, 2015. doi:10.1145/2837185.2837225.