CSC 547/ ECE 547 Semester Lab Project Spring 2018

Final Project Report: Container as a Service

Team 0

Bhushan Thakur (bvthakur)

Gokul Yelchuru (gyelchu)

Harshini Kadakol (hkadako)

Kashish Aggarwal (kaggarw2)

Pavithra Iyer (piyer3)

Ragavi Swarnalatha Raman(rraman2)

Computer Science Department

NC State University

Title: Cloud Options for Container as a Service

Abstract:

In recent times , many organizations have started to migrate to containers for their computing and storage needs. Containers are fast, lightweight and allows users to install a full stack service locally with complete isolation. Containers are also preferred by organizations as it speeds up the deployment cycle and reduces resource overhead. This project aims to build a prototype Container as a Service system that satisfies the on-demand computational needs of the customer to additionally support a company’s existing IaaS cloud option. The system has been designed to have a user friendly interface, a consistent backend setup, and a modular approach towards its features. A comparison with Amazon ECS (Elastic Container Service) is made in terms of performance.

Table of Contents:

Figure Index:

Table Index:

1. Introduction:

VM hypervisors like KVM or Xen, emulate the hardware and hence are heavy in terms of system requirements. But, containers on the other hand use a shared OS and hence rest on top of a Linux instance. This makes it easier to bundle and run a container. Using containers effectively we can provision four to six times the number of server application instances we run using Xen or KVM on the same piece of hardware. We can say that containers give the application instant portability. Hence we provide Container as a service to our customers thereby enabling us, the developers to isolate code into a single container. We use the Docker platform since it is made possible to run more applications on the same server. It also enables us to easily pack, ship and run applications as a lightweight, portable containers which can be run virtually anywhere.

i. Problem Statement:

To present a cost effective Container as a Service (CaaS) service by building a prototype service application and compare it with Amazon ECS, a commercial public cloud provider and a leader in the industry.

Our company currently provides Infrastructure as a Service (IaaS) cloud options for different projects in the company. As the cloud architecture team, we are required to investigate the CaaS option for our company and test out its implications for various projects. To achieve this, we develop a prototype Container as a Service (CaaS) model, deploy and test it. The end goal is to compare the performance of the following models:

- Private CaaS implementation
- AWS container services (ECS)

ii. Motivation

The containerization technology has taken up the industry by storm. It has proved to be a lighter alternative to virtual machines that are bulky, slow and difficult to deploy and scale. Containers have heavily influenced the cloud industry and many commercial products that offer containers as a service on demand are already in the market. A company that decides to use Containers internally for their teams' computation and storage needs can either Choose from a whole list of commercial vendors or try building their own system that enables customers to reserve and use containers as and when they need them. Hence the basic motivation behind the project is to investigate and compare various design options, functional, non-functional features that a customer would need, cost to setup and run the system, labor and the pricing component that goes into building a CaaS. Another major motivation of the project is to measure and compare the performance of the team's CaaS system with a commercial provider against various metrics.

iii. Issues Encountered

a) **Docker Registry setup issues -** When we were setting up the registry we were not able to pull/push images to the local registry running on 5000 port. We had to add the registry url in a daemon.json file to make it work. [2]

b) **iptables -F** - When we were researching about the iptable rules that had to be added on the management node to redirect traffic to compute nodes, we noticed that docker pull was not able to connect to the registry set up on the management node. Then we noticed that docker itself adds some iptable rules and on flush they get removed. We were able to revert back the state of the node by restarting the docker daemon on the management node.

c) **Database timestamp -** On doing reservations in the digitalocean instance, we realized that the mysql NOW() command fetched the server timestamp which was not in some remote region. To disallow mismatch of timings with database, we have set a timestamp parameter in the mariadb configuration file. Also, in the reservations page we print EST to indicate that we accept container reservations only in EST timezone.

d) **Nodejs -** An advantage of nodejs is that it enables asynchronous tasks so that the requests can be handled in parallel and the response time from the server is less. This was one of the reasons to choose nodejs for implementing our CaaS solution. However, asynchronous tasks can create a havoc if not implemented right! Some tasks have to implemented sequentially, for example, you can insert a row in the container table until your container creation in compute node is successful. We learnt the concept of creating a promise to make the asynchronous task wait for a return status of the previous task and take action accordingly.

iv. Environment

The environment requirements of the project are as follows:

Management node required software:

a. MariaDB

b. Docker

c. Node

Compute Nodes:

a. Docker

v. References

For the design and implementation of this project, we took advantage of a number of resources. Some of these references are learnt from the course, while others were found through the team's research. The main references are listed at the end of the document.

2. Requirements:

The fundamental requirements for our project are divided into

(1) Functional requirements which are the software capabilities that should be present so that the user can perform the services provided by the feature

(2) Non-functional requirements which specifies the standards and qualities that the system developed must have/ comply with.

2.1. Functional requirements:

The following functional requirements consists of the services/ capabilities associated with the features that should be present in CaaS environment.

(F1)User authentication:

Authentication identifies various users and provides different levels of access depending on their role. Our project has two types of users:

(i) Guest User - The guest user has access to reserve containers for a particular duration, view current reservations and delete reservations.

(ii)Administrator - The administrator has additional privileges to monitor the container systems so as to view the number of containers running, view the cpu/memory utilization of the compute nodes along with the ability to manage the database tables in the management node.

(F1.1) User SignUp: A user registers with a username and password. The registered user details gets stored in the user table of the database and is fetched whenever authentication is necessary.

(F1.2) User Login: The admin or the guest user can login using his/her registered credentials. The details entered gets authenticated with the user database and the user is allowed to view his/her admin dashboard.

(F2) Scheduling and Load Balancing:

(F2.1) Balance between user needs and System Availability: The scheduling component built in the application determines which resource to allocate the reservations. A check on the available resources in the system is made against the user requirements and the compute node is chosen accordingly.

(F2.2) Load Balancing: A key feature of the system that balances reservation requests that arrive between the available two compute nodes based on ratio of the total RAM available to the RAM necessary for a docker container in a particular reservation. This module avoids over-provisioning of resources which is an important check to be kept in a cloud provider's setup.

(F3) Image Reservation on Demand:

(F3.1) New Reservations : The reservations form the core component of the CaaS system. The user selected image based on its resource requirements need to be spun up as a fully functional docker container inside a compute node chosen based on the scheduling and load balancing algorithm. The user gets a public ip address and a port number through which the container can be accessed.

(F4) Manage Reservations:

(F4.1) View Current Reservations : The current reservations page gives the user, details about the reservations that he owns in a particular time period and the duration of the reservation

(F4.2) Delete Reservations : The user / admin can delete reservations that are in the running state if needed using the Delete Reservation. On deletion, all data,configuration and storage managed by the container gets deleted.

(F5) Monitor the cloud setup:

(F5.1) Do health checks as Admin User: The administrator has special privileges to monitor the performance of the CaaS cloud setup and do periodic health checks that can be used to generate reports on the usage of the system as a whole.

(F6) Customer Profiles:

(F6.1) Choose between customer or cloud provider's location for storage: The reservation dashboard gets a profile (C1/C2/C3) from the user based on which storage is allocated to the user. The profile options vary between choosing the local storage offered by the docker container on creation or a dedicated storage block that the user can mount into his container.

ii. Non-functional requirements:

(NF1) Cost : Provide a billing section in the customer's dashboard that gives the cost details of the containers based on the resources that the user has consumed.

(NF2) Usability : The user interface / Dashboard to which the user interacts is made in a consistent manner using a modular approach ie) the enter UI does not break if certain components fail. It caters to the needs of a universal group of users who might or might not be an expertise with the system.

(NF3) Security : The CaaS cloud system is only accessible via a single public IP address ie) via the management node. The compute nodes are given private IP addresses hence making them accessible only via the management node's public ip address and a specific port number for each container. Iptables in the management node do NAT functionality to redirect the traffic to the containers.

(NF4) Performance

(NF4.1) REST API Based Application : REST Based applications are light-weight ,offers interoperability and is not tightly coupled to the server. These aspects offer it better performance for our use case when compared to other applications like SOAP.

3. System Environment:

Hardware:

● Management node
CPU Info:
Quad Processor - Two cores each
Intel Core Q6600 @2.40 GHz X 4
153.2GB Storage
3.8 GB RAM

● Compute Node 1
CPU info:
Intel(R) core E5620 @2.40GHz X 16
957.9 GB Storage
23GB RAM

● Compute Node 2
Compute Node 1
CPU info:
Intel(R) core E5620 @2.40GHz X 16
957.9 GB Storage
23GB RAM

● Storage Node
CPU Info:
Quad Processor - Two cores each
Intel Core Q6600 @2.40 GHz X 4
153.2GB Storage
3.8 GB RAM

Software:

Management node required software:

a. MariaDB

b. Docker

c. Nodejs

Compute Nodes required software:

a. Docker

4. ENVIRONMENT/LAB SETUP:

i. Networking:

Our CaaS systems is group of compute nodes which is controlled and managed by a management node. In our prototype lab environment, we use the two powerful T610 Dell Machines as our compute nodes as it has a more RAM and storage. We have chosen this approach so as to accommodate multiple containers which are expected to run on our setup. One of the remaining 2 machines was made the management node and the other was used to research on mounting remote storage node for customer profile 2 (for example customer 1 in problem statement). The management node and compute nodes are connected through a private network while few ports of management node and storage nodes are configured in a separate VLAN so as to expose it to the Internet.

Customers essentially interact with the web UI and are assigned a management IP and port upon reservation. Once the customer chooses the type of service he requires, he connects to the particular service through the management node. The management node then executes the IP table rules that forward traffic for specific ports to the respective container. The Netfilter hooks are used to achieve the port forwarding which is described in more detail in the Firewall design section.

ii. Installation:

Management node required software:

MariaDB

1. Docker
2. Nodejs

Compute Nodes -

1. Docker

iii. Key-Pair Setup: Add the public key of the management node to the compute node's authorized key file for the root user.[9] This is required to setup passwordless login to compute nodes from management node during the application runtime.

iv. Docker Registry Setup: We have set up the docker registry on management node on 5000 port.[1] In the current release, we are using official images from Docker Hub, and pushing these to our local registry. We fetched the information about the ports exposed from the docker hub website and added it to our database. These ports are mapped to available port on compute node at run time. By default, SSH is not enabled in these images. In the future release, if we want to give our customers SSH access to the images, we will have to expose 22 port by building a new image using Dockerfile.[3]

v. Repository - We have maintained a GitHub repository for the development of our CaaS solution. The project was cloned inside the management node in the Desktop directory. We used npm package manager to install the dependencies. You can start the web application by running -

```
node app.js
```

This starts the application on 3000 port.

vi. Database -

MariaDB Server is one of the most popular database servers in the world. We used MariaDB because it is fast, scalable and robust, with a rich ecosystem of storage engines, plugins and many other tools.

After installing MariaDB, we have to do the following configurations:-

1. We have to set the timezone of the database in the configuration file to EST. We can take reservations only in the EST timezone in our CaaS solution. Database has to be restarted so that the new configurations are loaded.
2. Database user is created.
3. Create database
4. Create tables in the database.
5. Preload the computer,image, image_ports, nat_ports, computer_ports in the table. The database administrator has to ensure that all the information about the images pushed to the registry are also added to the images table to make it available to the user. The nat_ports table should be loaded with all the ports available in management node that can be utilized in the NAT rules. All the ports exposed by images should go into the image_ports table. All the ports available in the compute nodes should be entered in the computer_ports table.

6. Ensure that management node gets comid 1 in the computer table. Only the management node gets public IP. The public IP column for the compute nodes should be left NULL.
7. Admin details have to be manually added to the database by the database administrator. For security reasons, we do not allow sign up of admin from the UI.

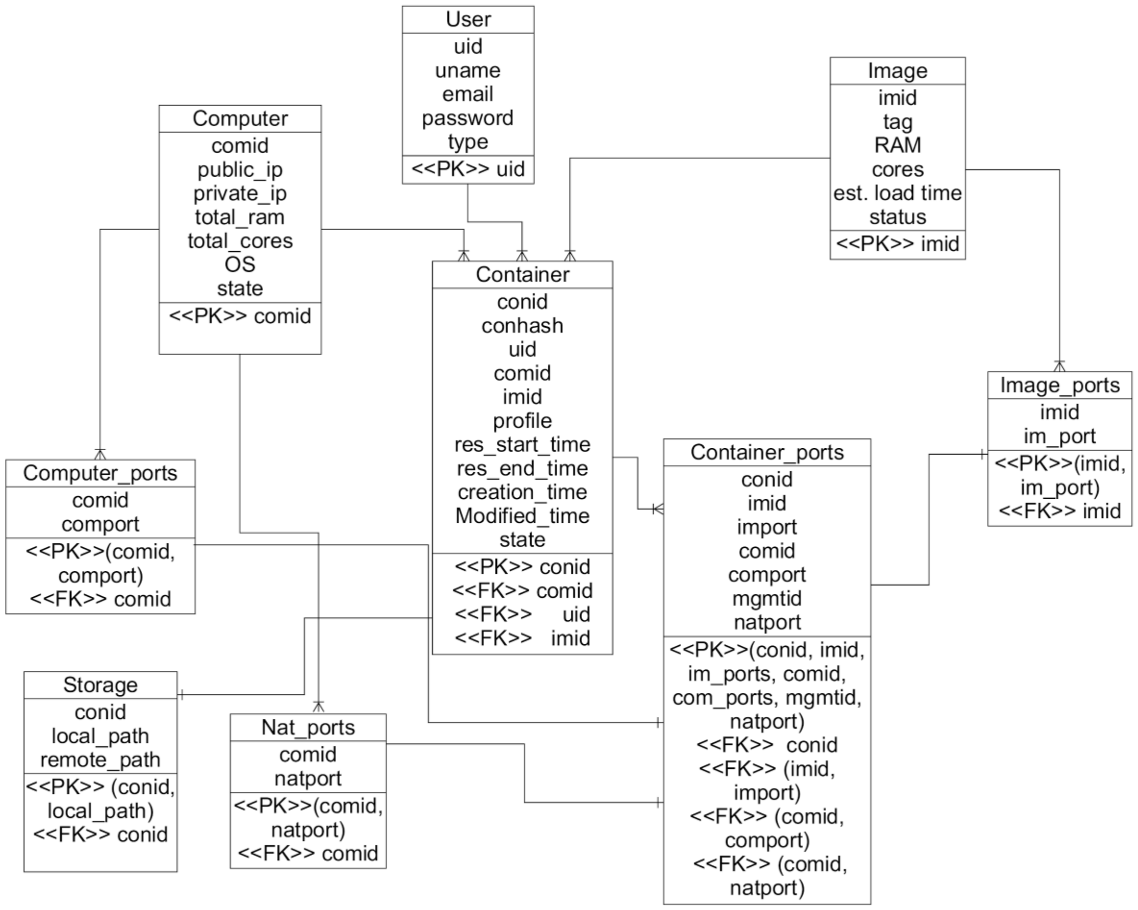


Fig 4.1 Database Schema

5. Design,Architecture And Results:

5.1. System Architecture:

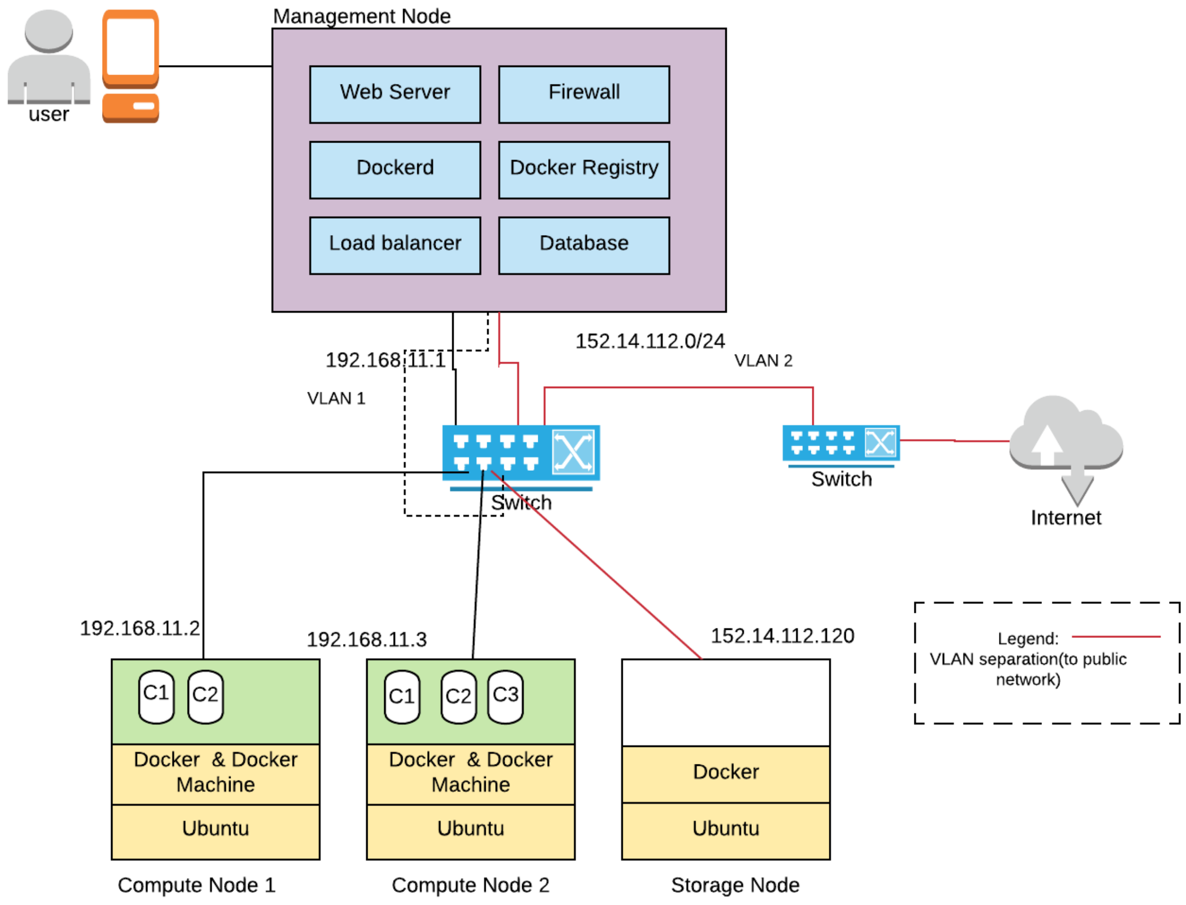


Fig 5.1 System Topology

5.2. Implementation

1. <u>Login/Signup:</u>

Authentication is considered an important component when creating a dynamic web application.

For signup we run an insertion query to add new users to database. Signup takes in 3 parameters: username, password and email. Unique username is enforced through the database. Hence, the registration is successful only if the username is unique. Passwords should not be stored in plaintext format in the database as it acts as a security threat. We have used bcrypt nodejs module to encrypt the password which hashes the password before inserting it into the database.

Ideally, passwords show not go in plaintext format over the network when user tries to login to prevent man in the middle attack. Nowadays, we have several ways to enforce security like single sign on and two-factor authentications. Since we are just building a prototype, we did not add these to our CaaS solutions. Security features will be added in the future release.

Login: We validate a user by checking if the username exists in database. We then validate the password for allowing successful login.

2.<u>Billing:</u>
We charge the customer $0.50/hr per core. A SQL query is executed to obtain the charges owed by the customer based on the usage. We have implemented a single query to fetch the usage for a customer from the database and calculate

ceil(end timestamp - start timestamp)*0.50*cores required by image/service

This feature can be enhanced later by generating monthly bills and keeping track of payments.

3. <u>Reservation:</u>

Creating a Reservation:
The reservation module is responsible for maintaining the image reservation details. The user is provided the ability to pick an image that is available in the local docker registry from a drop down menu and choose start_time, duration and customer profile (used to choose between storage options).

Following steps are completed in the given order to complete a reservation request:

- Call the scheduling module to determine the compute node where it needs to spin the container.
- Depending on chosen customer profile, configure storage options - local or remote.
- Map the ports exposed by the image to port available in the compute node. This mapping is done at runtime to ensure that different containers of same image are mapped to different ports on the same compute node. Eg - nginx image exposes port 80. The port mapping enables to redirect traffic to appropriate container based on the port which receives the tcp connection.
- Spin the container on the compute node and get the container hash value.

- Update the database with the new container information.
- Add iptable rules in management node. Each image port as specified above is mapped to a compute node port which again has to be mapped to a management port which serves as the NAT port. TCP traffic that hits the management node is redirected to the correct compute node and port through the iptable rule added in the nat table in the PREROUTING chain. Update database with the porting information.
- Provide the user with a public ip of management node and port numbers using which he can access his service/container.

The concept of port mapping and scheduling are discussed in detail in this report later.

Show/Delete Reservation:
The customer has a button in the dashboard to view his reservation. The image name, start time, end time, connection details(IP,port) and delete button is showed to the user. A simple SQL query is written to fetch these details from the database. The user can end his reservation before the reservation expires by clicking on the delete button.

4. Scheduling and Load Balancing:
The reservation module calls our scheduler which is responsible for selecting the appropriate compute node to run our docker instance. One of the very important characteristics of our scheduler is the load balancing. The aim is to have a uniform distribution of resource usage across machines i.e. ram, cores etc. as well as distribute the incoming network traffic to the management node uniformly to our backend nodes.
As part of our project we investigated the following load balancing algorithms and came up with the MINIMUM RAM RATIO technique that best suites our scenario.
Load balancing algorithm designs:

1. Round Robin

The container requests coming to the management node are delegated to compute nodes one by one in an ordered fashion. All the compute nodes in the system are configured to offer identical services. All are configured to use the same internet domain name but all of them have a unique ip.
Pros:

- Simple to implement
- The network traffic due to container requests is also distributed uniformly between the management node and the compute nodes.

Cons:

- Doesn’t take into account hardware configuration of the compute nodes as some powerful compute nodes can handle more containers than others.
- Additional information like a counter that know the last compute node used needs to be kept into account.

- Adding new compute nodes to the system, it won’t be fair to existing resources if the counter is till beginning of its round robin cycle.
- In run-time, when containers are deleted randomly, some compute nodes will be overloaded with containers whereas the other compute nodes will be free. Round robin algorithm cannot enforce equal utilization of resources.

2. Weighted Response Time

The server keeps track of the response time of different compute nodes to requests relayed by the management node. The compute node with fastest response time is chosen to deploy the next container. This ensures that nodes heavily loaded will respond more slowly and will not be sent new container requests. This allows the load to even out on the available node pool over time.

Pros:

- Uses most up to date stats of the system to determine the node that is most fit for container deployment.
- Automatically takes into account if there are any resource failures in the node as that will have a direct impact on the response time and our chances to select that node.

Cons:

- Really complex to implement
- If the latest response time is considered then even a small spike in network can impact our decision whereas if we take average response time, it will take us more time to realise a node failure.

3. Minimum RAM Ratio (**OUR IMPLEMENTATION**)

The management node checks which node has the minimum ram ratio:

RAM Ratio = RAM_in_use / Total_RAM

If there is an idle node, it will be chosen for container deployment else the node with minimum proportionate RAM in use will be chosen.

Pros:

- Simple to implement
- No extra data needed to be stored as the system already stores the hardware configuration of all the nodes and the resource demands of different container images.

Cons:

- Doesn’t take into account other resources like cores, processor speed etc which might exhausted first.
- Doesn’t take into account, the number of requests for each container, as a container using less RAM might be heavily loaded with requests and the node will be too busy taking care of those.
- Node failure is not detected.

Our scheduler works on the minimum RAM ratio algorithm. When a create reservation request is issued from the UI by the user, the scheduling algorithm calculates the ram ratio for that reservation interval and tells that reservation module that chosen host for container deployment.

5. Storage:

Our CaaS system offers two types of storage options:

1. Local repository
2. Remote cloud Provider's location

Volumes are a prefered mechanism used by container to store persistent data that is generated and used by them. Volumes are totally managed by docker.

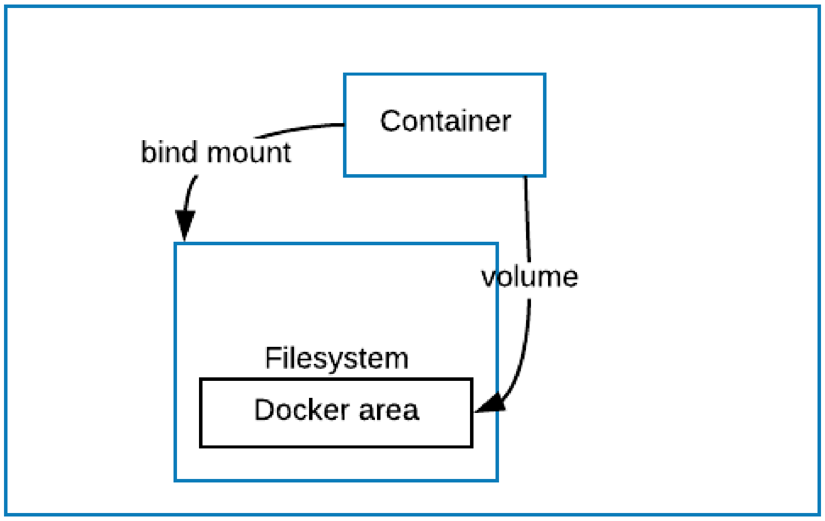


Fig 5.2 Storage Design

There are other mechanisms that could have been used but Volumes have the following advantages:

1. Volumes are easier to back up
2. Volume can be used by docker CLI commands
3. Volumes can be used on both windows and linux environments.
4. Volumes can be shared among different containers.
5. Volume drivers allow you to store data on remote hosts, or cloud providers.
6. Data present in a volume can be pre populated into a container.

Using a volume and the data that goes into it are kept outside a container's lifecycle so increase in data size doesn't increase the size of the container.

Local Repository

- A directory is created in the compute node (the host).
- While creating a new container, the directory is mounted in the container using '-v' tag.
- Any data that is stored or updated in the mounted directory will reflect in the host directory.
- The data will persist even after the container is destroyed and can be mounted on a different container.

We used the following commands to mount directories from a machine to our local host, using **sshfs.**

```
# docker-machine mount dev:/home/docker/foo foo
```

where dev is the remote machine and /home/docker/foo is the local directory

```
# docker run -v /root/local:/root/remote <hello-world>
```

where /root/data1 is local and /root/data is remote

To unmount use:

```
# docker-machine mount -u vm:/root/data1 /root/data
```

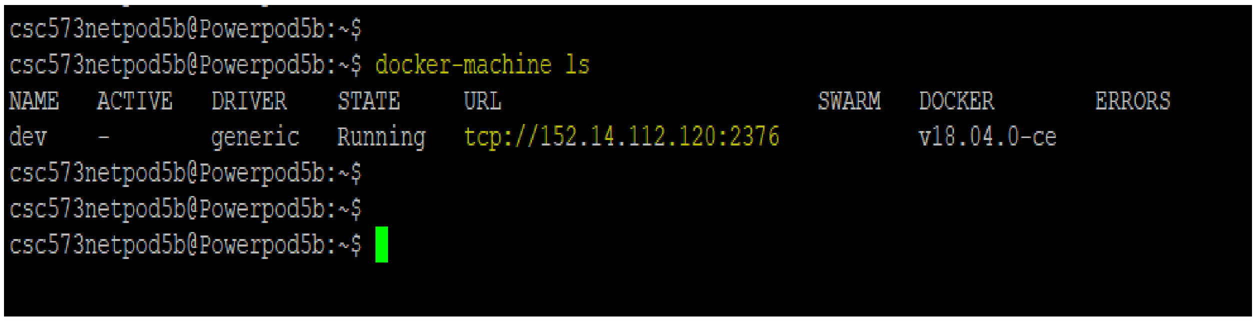
Remote Storage

Docker-machine is a tool that lets you use a remote machine as a docker host to run containers. In our case we our using it to use our dedicated storage node as remote storage. It can be used to seamlessly add any number of remote nodes and provision storage on the same.

- Register the storage node with the compute nodes as a docker host via docker-machine create command.
- Every time a user needs a remote storage option, get the chosen host from scheduler.
- Create a directory on the storage node via ssh and mount it on the compute node.
- Now use this mount point, and mount it inside the container using docker volume option.
- While deleting the container, unmount the remote storage.

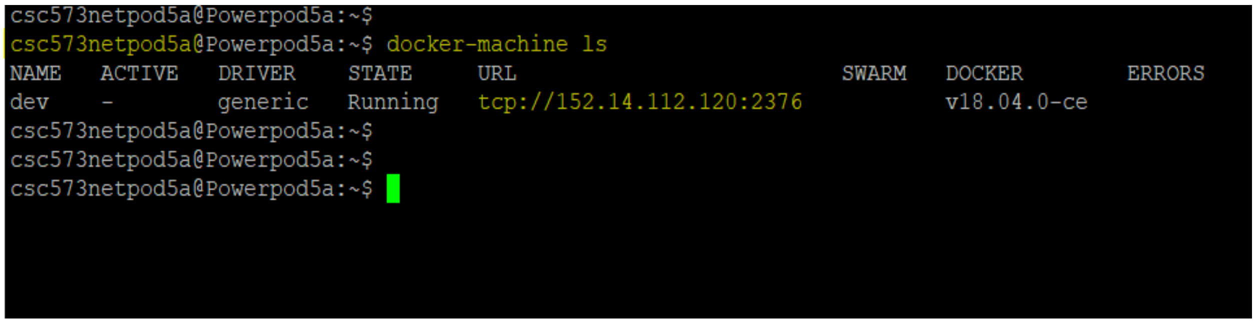
In our setup, all the containers requested by Customer 1 are provided with a local storage and Customer 3 are given a remote storage by default.

As part of future scope, docker-machine gives us a choice of using AWS S3 or storage nodes on digital ocean as an option too.



```
csc573netpod5b@Powerpod5b:~$
csc573netpod5b@Powerpod5b:~$ docker-machine ls
NAME   ACTIVE   DRIVER    STATE     URL                         SWARM   DOCKER        ERRORS
dev    -        generic   Running   tcp://152.14.112.120:2376           v18.04.0-ce
csc573netpod5b@Powerpod5b:~$
csc573netpod5b@Powerpod5b:~$
csc573netpod5b@Powerpod5b:~$
```

Fig 5.3 Docker machine on node1



```
csc573netpod5a@Powerpod5a:~$
csc573netpod5a@Powerpod5a:~$ docker-machine ls
NAME   ACTIVE   DRIVER    STATE     URL                         SWARM   DOCKER        ERRORS
dev    -        generic   Running   tcp://152.14.112.120:2376           v18.04.0-ce
csc573netpod5a@Powerpod5a:~$
csc573netpod5a@Powerpod5a:~$
csc573netpod5a@Powerpod5a:~$
```

Fig 5.4 Docker machine on node2

6. Monitoring:

- **Container Health Check:** The admin is provided a functionality to check the status of the containers in the compute node. All the containers expected to be running are inspected one by one and the status of the container is printed in the admin dashboard. The health of the container is determined by running a simple command[4]:

  ```
  docker inspect <container uuid>
  ```

  If docker service goes down in any compute node, it is an indication that the compute nodes are overloaded with containers. Admin might consider horizontal scaling(add more computers in his cloud) or cloud bursting (where he can redirect some image reservation traffic to AWS).
- **Compute Node Health Check:** The admin can see free memory, total memory, uptime and average load of all the compute nodes in our cloud. These run-time values can help the admin to solve real-time issues.

7. Firewall and Security Design:

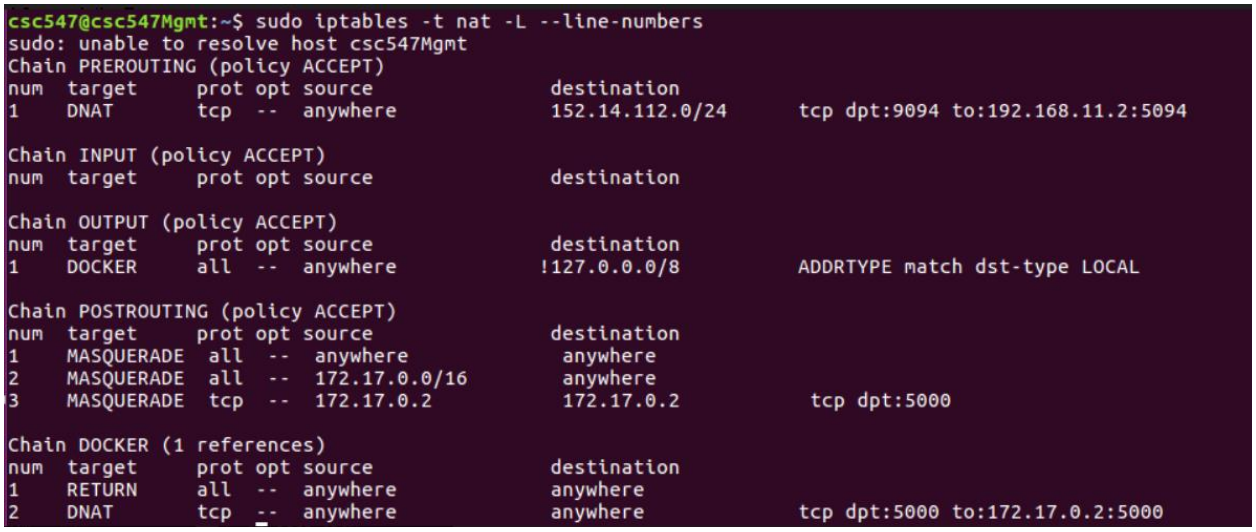
Netfilter is a framework implemented in the Linux kernel offering hooks to register call back function with the kernel networking stack. We use IPtable rules to implement our firewall design. It consists of five hooks (INPUT, OUTPUT, PREROUTING, POSTROUTING, FORWARD) out of which we have implemented the rule in FORWARD chain as we have to forward the request coming from the client to the compute node. It will keep track of the TCP states.

The following IP table rules were implemented:

```
iptables -t nat -I PREROUTING -p tcp -d <mgmtip> --dport <natport(mgmtnode)> -j DNAT --to-destination <compute node private ip>:<compute node port>
```

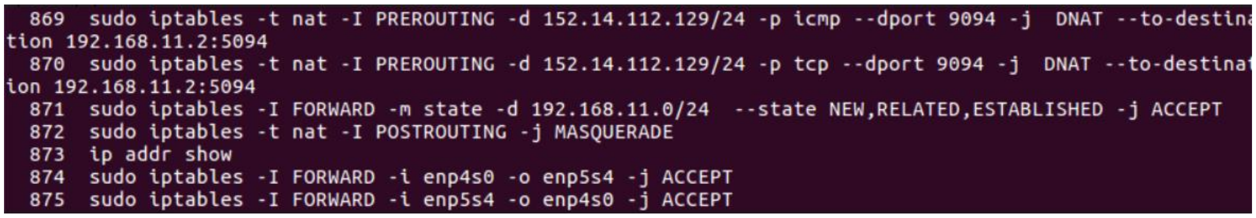
```
iptables -I FORWARD -m state -d <Compute Node Network Addr> --state NEW,RELATED,ESTABLISHED -j ACCEPT
```

Further, we have added a NAT rule on the PREROUTING hook to route the traffic before passing it to the management node's routing table.



```
csc547@csc547Mgmt:~$ sudo iptables -t nat -L --line-numbers
sudo: unable to resolve host csc547Mgmt
Chain PREROUTING (policy ACCEPT)
num  target     prot opt source               destination
1    DNAT       tcp  --  anywhere             152.14.112.0/24      tcp dpt:9094 to:192.168.11.2:5094

Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source               destination
1    DOCKER     all  --  anywhere            !127.0.0.0/8          ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT)
num  target     prot opt source               destination
1    MASQUERADE  all  --  anywhere             anywhere
2    MASQUERADE  all  --  172.17.0.0/16        anywhere
3    MASQUERADE  tcp  --  172.17.0.2           172.17.0.2           tcp dpt:5000

Chain DOCKER (1 references)
num  target     prot opt source               destination
1    RETURN     all  --  anywhere             anywhere
2    DNAT       tcp  --  anywhere             anywhere             tcp dpt:5000 to:172.17.0.2:5000
```

Fig 5.5 NAT rules



```
  869  sudo iptables -t nat -I PREROUTING -d 152.14.112.129/24 -p icmp --dport 9094 -j  DNAT --to-destina
tion 192.168.11.2:5094
  870  sudo iptables -t nat -I PREROUTING -d 152.14.112.129/24 -p tcp --dport 9094 -j  DNAT --to-destinat
ion 192.168.11.2:5094
  871  sudo iptables -I FORWARD -m state -d 192.168.11.0/24  --state NEW,RELATED,ESTABLISHED -j ACCEPT
  872  sudo iptables -t nat -I POSTROUTING -j MASQUERADE
  873  ip addr show
  874  sudo iptables -I FORWARD -i enp4s0 -o enp5s4 -j ACCEPT
  875  sudo iptables -I FORWARD -i enp5s4 -o enp4s0 -j ACCEPT
```

Fig 5.6 NAT Rules

5.3. Services and Applications:

The main motive of our CaaS system is to provide users with container services and applications. Currently the services supported by the private CaaS system are all web-based. All the images currently are based on public docker images from Docker Hub.

Our CaaS system currently supports the following 5 Services:

- Redis
- Jenkins
- nginx
- mysql
- mongo

5.4. Reasons for choosing Docker:

Even though with Docker, all containers must use the same operating system and kernel, this platform is useful if we want to provision the most server application instances running on a minimum amount of hardware. Through this we can save the cost incurred in hardware and power by a huge margin.

Since Docker and AWS have teamed up to make it easier to deploy Containers as a Service on an underlying AWS IaaS service, it was a choice of choosing the best comparison metric easier for the comparison of the company's CaaS. Docker was chosen as the first choice to be used in our company's provision of Containers as a Service.
Why choose Docker over other container technologies.

Docker is built on top of LXC. On top of this low-level foundation of kernel features, Docker offers a high-level tool with several powerful functionalities. Docker abstracts teh operating system kernel and provides a lightweight virtualization solution to run processes in isolation. It can provide a higher level of capabilities:

1.Portability: Ability to create a single object containing all the bundled applications. This object can then be transferred and quickly installed onto any other Docker-enabled Linux host. LXC has the sandbox capability as well but if we have an application installed with custom lxc configurations, it will not be possible to run on a different machine the same way it ran in the original machine since it is tied to the original machine's specifications. Docker provides the ability of abstraction to run unchanged on different machines with different configurations.

2.Versioning:Docker includes git-like capabilities for tracking successive versions of a container, inspecting the diff between versions, committing new versions, rolling back etc.

3.Reuse: Docker allows creating multiple packages which are already created. For example, we can create a base image having the image files of two or more applications and build it as one if we need to create several machines that require the same applications.

4.Continuous Integration/Continuous Deployment (CI/CD) : A DevOps methodology which Docker uses to integrate the code. Docker has the capability of automated build as well. The developers are free to use make, maven, chef, puppet, salt, debian packages, rpms, source tarballs, or any combination of the above, regardless of the configuration of the machines.

5.Sharing: Docker has access to a open-source public registry (https://registry.hub.docker.com/) where thousands of people have uploaded useful containers.

6. Results:

We as a CSC547 cloud provider successfully deployed the Containers as a Service(CaaS) in agreement with the functional and nonfunctional requirements. The customers are able to register and create reservations based on their specific requirements.
The following tasks requirements have been successfully achieved:
1. Users can signup, login and and reserve, view, delete containers.
2. Users are offered a user-friendly UI in the form of a dashboard.
4. Users are provided with five different Containers services.
The verification of these results are listed under the verification and validation section of this report.

7. Verification and validation:

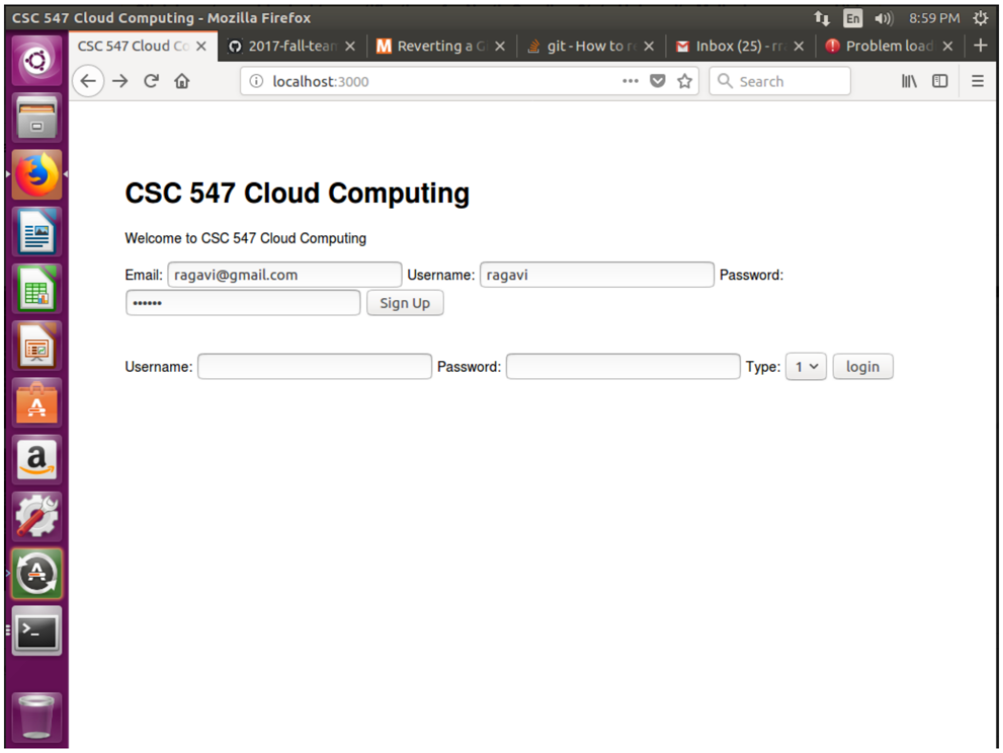
Requirement (F/NF)#	**TestCase**	**Expected Observation**	**Observed Results**
F1.1	User enters registration details and submits	Registration must be successful and addition of user details in user table	Addition of user details in user table takes place
F1.2	User logs in with registered credentials	Login should be Successful and redirected to Dashboard	Authentication happens and user is successfully redirected to the Dashboard.
F3.1	User gives valid image,start_time,duration that do not overlap and profile and hits reservation Button in the Dashboard	container reservation must be successful, user gets IP address and port number	container reservation successful, user gets IP address and port number
F3.2	All compute nodes down,User creates reservation request and hits reserve in the Dashboard	User should receive “no resources available” message	Reservation fails and user receives “no resources available” message in his dashboard
F2.1 and F2.2	Assign custom loads on different compute nodes for a create reservation request	The correct host should be allocated for a container request	The correct host is indeed allocated for a container request
F4.1	User needs to see his current reservations by clicking the “Manage Reservations” tab	User should be able to view current reservations and the duration of each container reservation	Dashboard page gets loaded successfully and user is able to see current reservations along with options to manage it like Delete button
F4.2	Reservation time for	Container should	Container stopped,

	the user ends	stop as soon as the end time reaches, the database entries and needs have to be removed	all configurations/data removed
F4.2	User wants to cancel his reservation in between	Container should stop as soon as the delete module executes, the database entries and needs have to be removed	Container stopped, all configurations/data removed and displays "Deletion successful" message in Dashboard
F5.1	Admin uses Dashboard to monitor/ do health checks on the containers	Admin should be able to see information about containers and compute nodes' health	Admin is able to see which containers are running, resources used by each compute node etc
F6.1	User wants to create reservation based on user profile to select storage requirements	The corresponding storage option should be selected and the new container should have appropriate amount of storage locally or mounted remotely	The storage option gets chosen correctly, and the new container is available for the user with storage allocated appropriately
NF3	The compute node is reachable only via the management ip address and port	Prerouting iptable rules for redirecting the traffic from management node to compute node should be run when user accesses the container in his browser and the webpage has to be seen	DNAT and MASQUERADE takes place and user is able to view the chosen service's homepage in his browser

Table 1. Verification and validation table

7.1 Test Analysis and Screenshots:

F1.1:SignUp of customer:



CSC 547 Cloud Computing

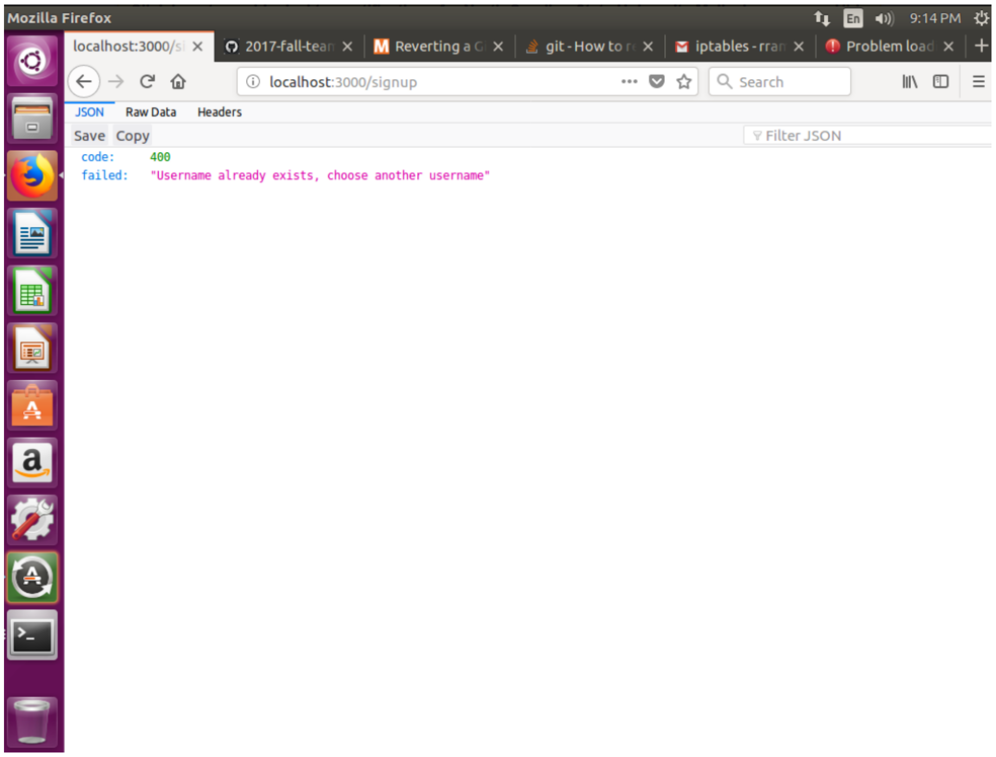
Welcome to CSC 547 Cloud Computing

Email: ragavi@gmail.com Username: ragavi Password: •••••• Sign Up

Username: Password: Type: 1 login

Fig 7.1 Signup of customer

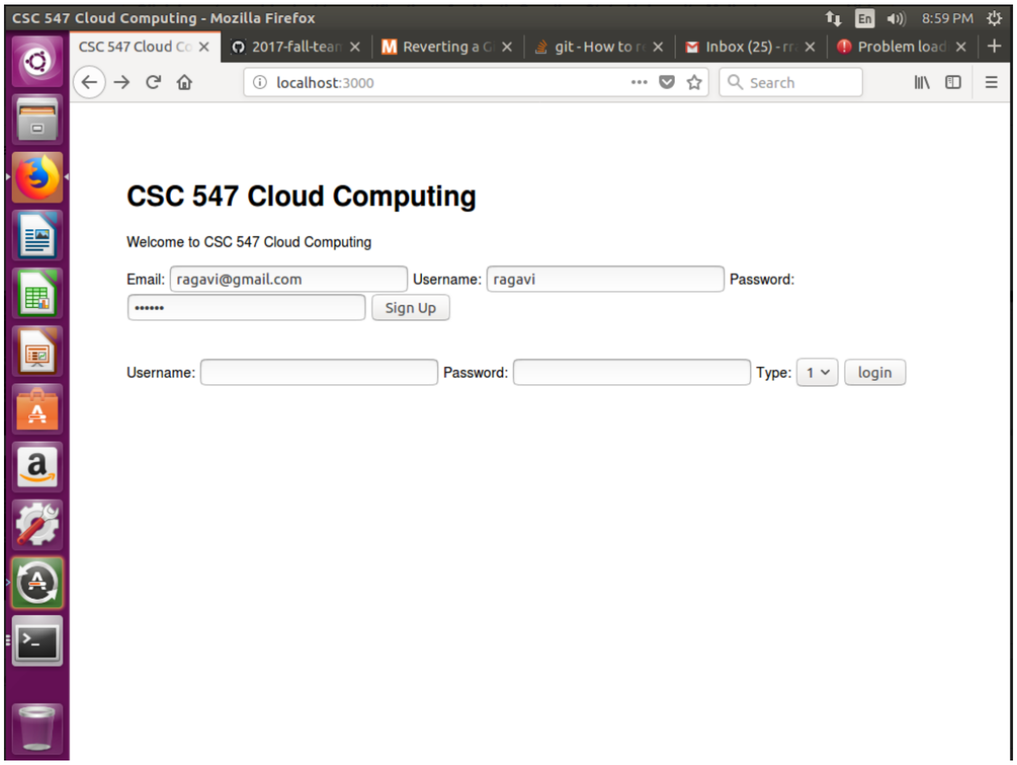
Below we can see that the user signup has failed for using a username already present



```
code:    400
failed:  "Username already exists, choose another username"
```

Fig 7.2 Signup failure

F1.2: User logs in



CSC 547 Cloud Computing

Welcome to CSC 547 Cloud Computing

Email: ragavi@gmail.com Username: ragavi Password: ••••••

Username: Password: Type: 1

Fig 7.3 User login page

F3.1:The User Reservation Page:

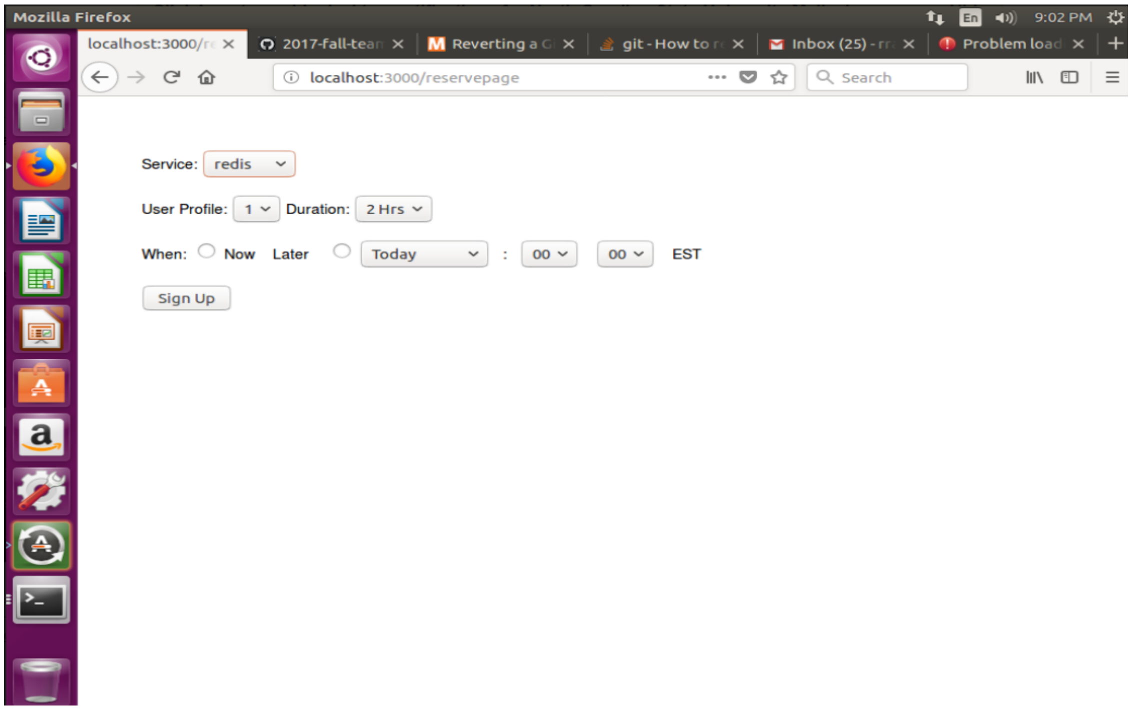
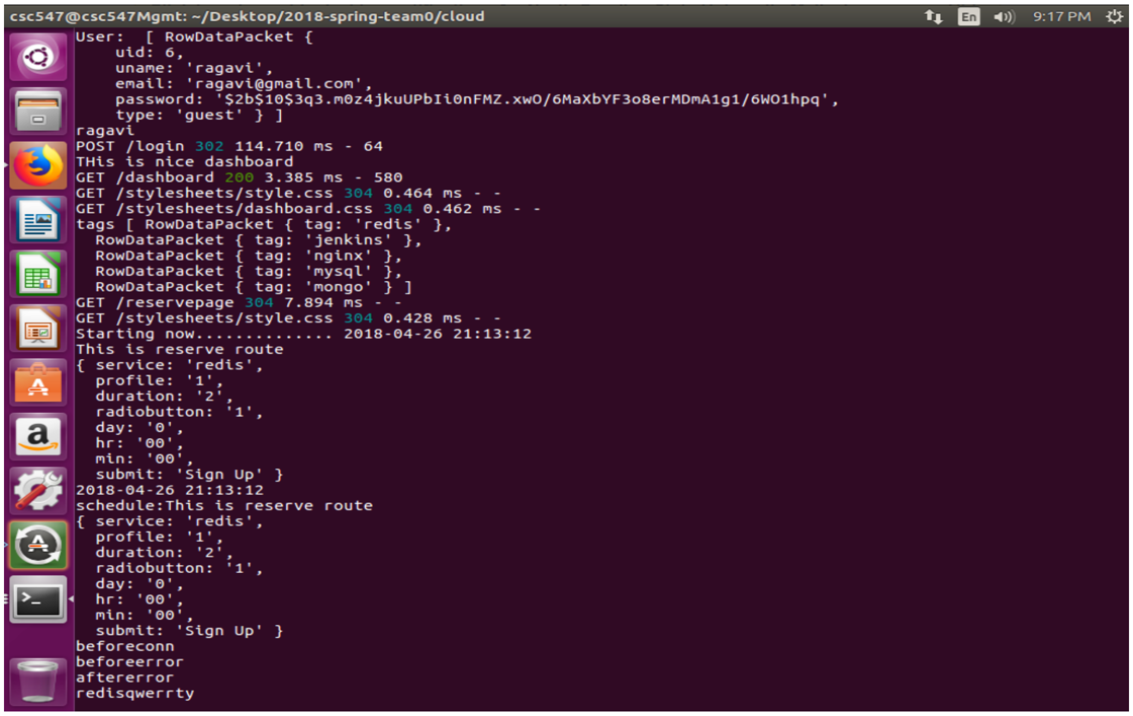


Fig 7.4 User Reservation Page

F3: Reservation Backend

The node.js log showing the app.js module picking up input values as JSON and passing it to Schedule and Reservation Page:



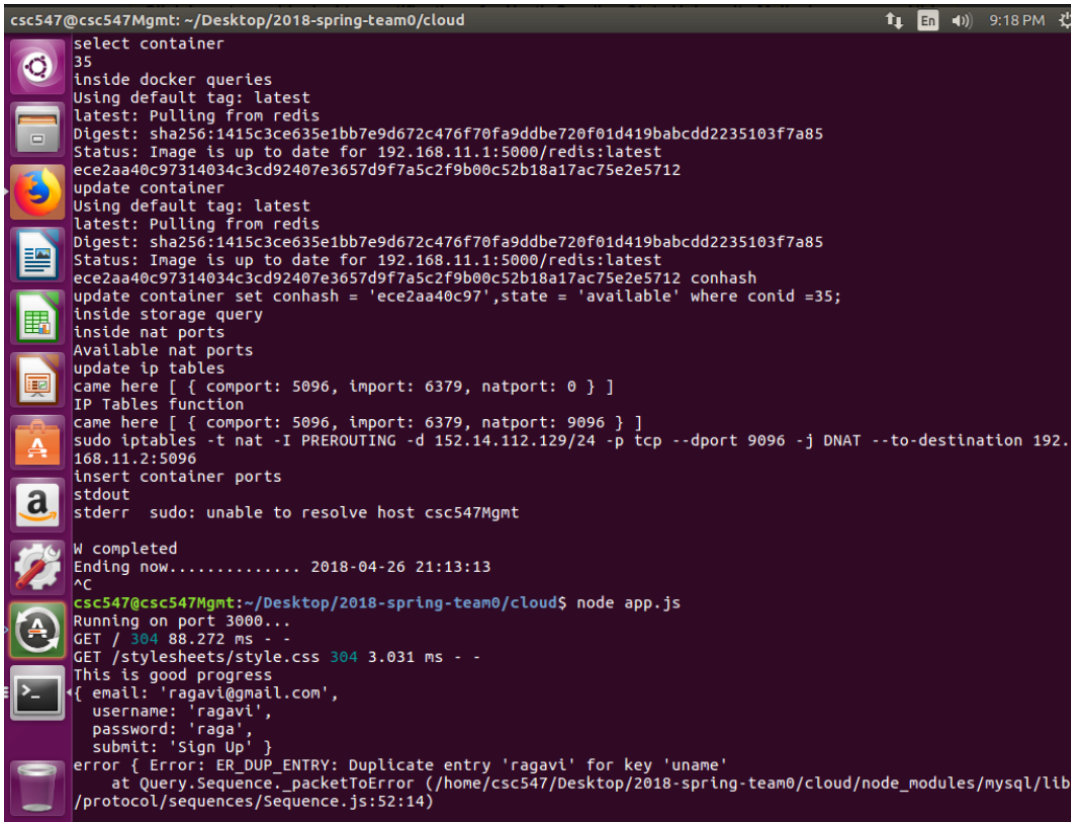
```
csc547@csc547Mgmt: ~/Desktop/2018-spring-team0/cloud
User:  [ RowDataPacket {
    uid: 6,
    uname: 'ragavi',
    email: 'ragavi@gmail.com',
    password: '$2b$10$3q3.m0z4jkuUPbIi0nFMZ.xwO/6MaXbYF3o8erMDmA1g1/6WO1hpq',
    type: 'guest' } ]
ragavi
POST /login 302 114.710 ms - 64
THis is nice dashboard
GET /dashboard 200 3.385 ms - 580
GET /stylesheets/style.css 304 0.464 ms - -
GET /stylesheets/dashboard.css 304 0.462 ms - -
tags [ RowDataPacket { tag: 'redis' },
  RowDataPacket { tag: 'jenkins' },
  RowDataPacket { tag: 'nginx' },
  RowDataPacket { tag: 'mysql' },
  RowDataPacket { tag: 'mongo' } ]
GET /reservepage 304 7.894 ms - -
GET /stylesheets/style.css 304 0.428 ms - -
Starting now.............. 2018-04-26 21:13:12
This is reserve route
{ service: 'redis',
  profile: '1',
  duration: '2',
  radiobutton: '1',
  day: '0',
  hr: '00',
  min: '00',
  submit: 'Sign Up' }
2018-04-26 21:13:12
schedule:This is reserve route
{ service: 'redis',
  profile: '1',
  duration: '2',
  radiobutton: '1',
  day: '0',
  hr: '00',
  min: '00',
  submit: 'Sign Up' }
beforeconn
beforeerror
aftererror
redisqwerrty
```

Fig 7.5 Reservation Page

F3.1, F6 and NF3 : Reservation and Security Backend

(v) The backend nodejs log showing

- **(a) Reservation success**
- **(b) The image being pulled from local registry**
- **(c) Container being created**
- **(d) Port Mapping (among image , management and compute ports)**
- **(e) IPtables executing when a client logs in the container**



```
select container
35
inside docker queries
Using default tag: latest
latest: Pulling from redis
Digest: sha256:1415c3ce635e1bb7e9d672c476f70fa9ddbe720f01d419babcdd2235103f7a85
Status: Image is up to date for 192.168.11.1:5000/redis:latest
ece2aa40c97314034c3cd92407e3657d9f7a5c2f9b00c52b18a17ac75e2e5712
update container
Using default tag: latest
latest: Pulling from redis
Digest: sha256:1415c3ce635e1bb7e9d672c476f70fa9ddbe720f01d419babcdd2235103f7a85
Status: Image is up to date for 192.168.11.1:5000/redis:latest
ece2aa40c97314034c3cd92407e3657d9f7a5c2f9b00c52b18a17ac75e2e5712 conhash
update container set conhash = 'ece2aa40c97',state = 'available' where conid =35;
inside storage query
inside nat ports
Available nat ports
update ip tables
came here [ { comport: 5096, import: 6379, natport: 0 } ]
IP Tables function
came here [ { comport: 5096, import: 6379, natport: 9096 } ]
sudo iptables -t nat -I PREROUTING -d 152.14.112.129/24 -p tcp --dport 9096 -j DNAT --to-destination 192.
168.11.2:5096
insert container ports
stdout
stderr  sudo: unable to resolve host csc547Mgmt

W completed
Ending now.............. 2018-04-26 21:13:13
^C
csc547@csc547Mgmt:~/Desktop/2018-spring-team0/cloud$ node app.js
Running on port 3000...
GET / 304 88.272 ms - -
GET /stylesheets/style.css 304 3.031 ms - -
This is good progress
{ email: 'ragavi@gmail.com',
  username: 'ragavi',
  password: 'raga',
  submit: 'Sign Up' }
error { Error: ER_DUP_ENTRY: Duplicate entry 'ragavi' for key 'uname'
    at Query.Sequence._packetToError (/home/csc547/Desktop/2018-spring-team0/cloud/node_modules/mysql/lib
/protocol/sequences/Sequence.js:52:14)
```

Fig 7.6 Reservation and Security Backend

(F4) User Manage/Delete Reservations:

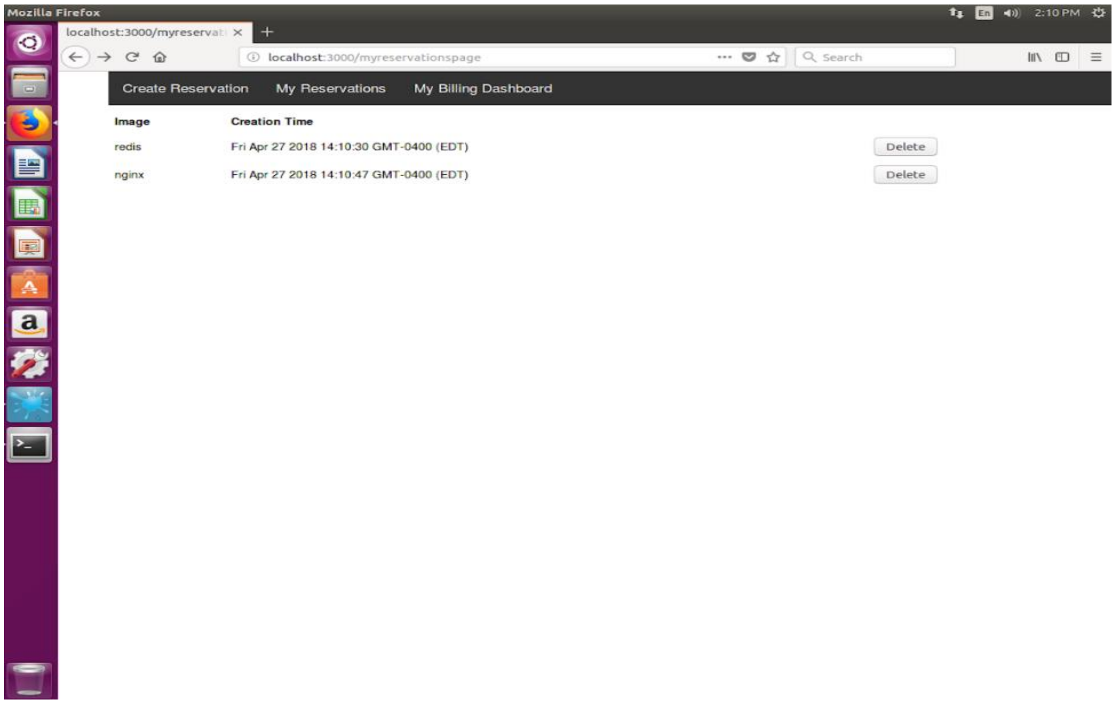
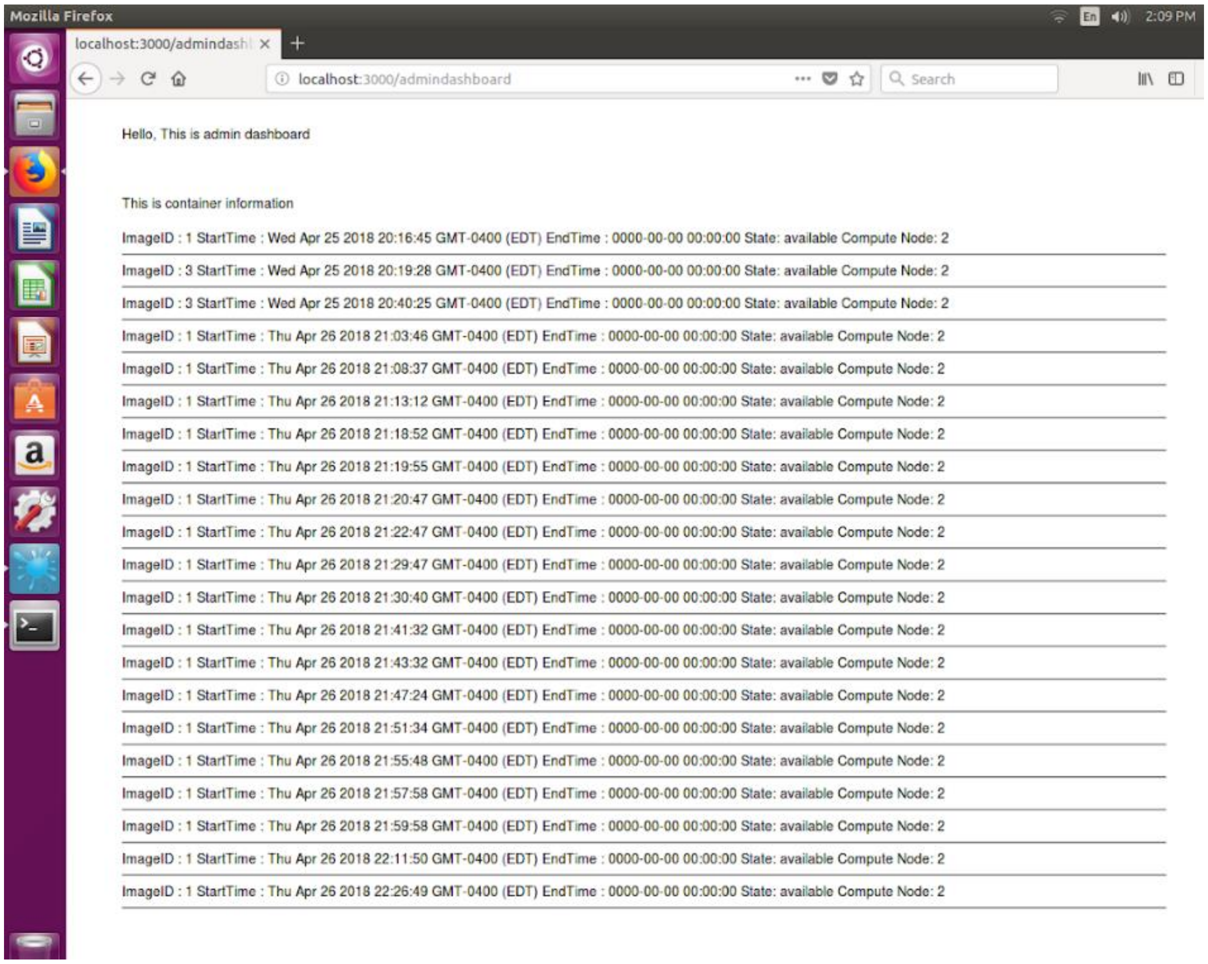


Image	Creation Time	
redis	Fri Apr 27 2018 14:10:30 GMT-0400 (EDT)	Delete
nginx	Fri Apr 27 2018 14:10:47 GMT-0400 (EDT)	Delete

Fig 7.7 User Manage Reservation Page

(F5.1) Do health checks as Admin User:



Hello, This is admin dashboard

This is container information

```
ImageID : 1 StartTime : Wed Apr 25 2018 20:16:45 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 3 StartTime : Wed Apr 25 2018 20:19:28 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 3 StartTime : Wed Apr 25 2018 20:40:25 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 1 StartTime : Thu Apr 26 2018 21:03:46 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 1 StartTime : Thu Apr 26 2018 21:08:37 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 1 StartTime : Thu Apr 26 2018 21:13:12 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 1 StartTime : Thu Apr 26 2018 21:18:52 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 1 StartTime : Thu Apr 26 2018 21:19:55 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 1 StartTime : Thu Apr 26 2018 21:20:47 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 1 StartTime : Thu Apr 26 2018 21:22:47 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 1 StartTime : Thu Apr 26 2018 21:29:47 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 1 StartTime : Thu Apr 26 2018 21:30:40 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 1 StartTime : Thu Apr 26 2018 21:41:32 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 1 StartTime : Thu Apr 26 2018 21:43:32 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 1 StartTime : Thu Apr 26 2018 21:47:24 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 1 StartTime : Thu Apr 26 2018 21:51:34 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 1 StartTime : Thu Apr 26 2018 21:55:48 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 1 StartTime : Thu Apr 26 2018 21:57:58 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 1 StartTime : Thu Apr 26 2018 21:59:58 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 1 StartTime : Thu Apr 26 2018 22:11:50 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
ImageID : 1 StartTime : Thu Apr 26 2018 22:26:49 GMT-0400 (EDT) EndTime : 0000-00-00 00:00:00 State: available Compute Node: 2
```

Fig 7.8 Admin Monitoring Page

8. Comparisons:

CaaS vs AWS CaaS vs VCL IaaS

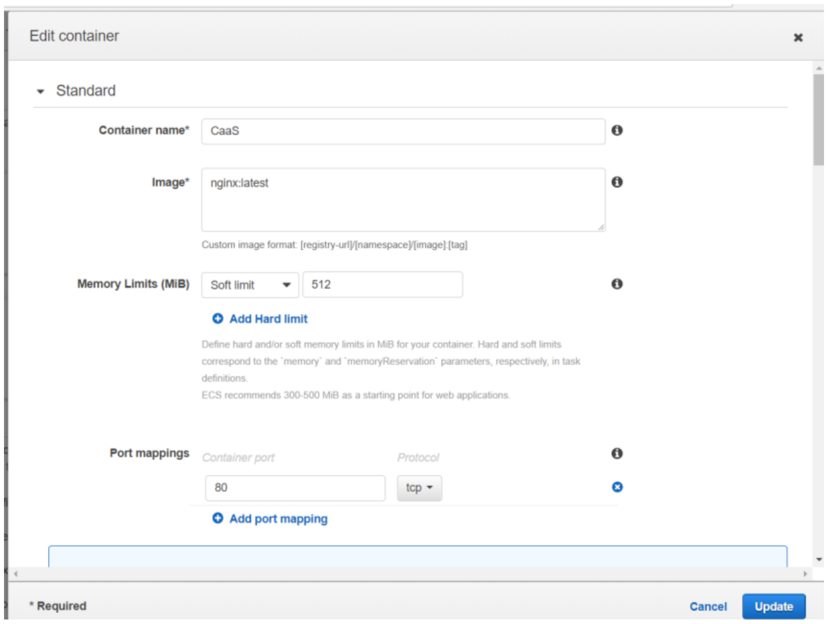
We have compared our CaaS implementation against AWS ECS service on the the basis of important structure in a cloud system. We provide the comparison of our Containers as a Service with AWS Containers as a Service. AWS Elastic Container Service(ECS) has two instance types: Fargate and EC2. Fargate is a new instance currently available in the us-east-1 (N.Virginia) Region. Fargate allows us to run tasks and services on infrastructure managed by AWS.

We have also compared how the current VCL system incorporates these subsystems.

i. Size and memory requirements:

CaaS Solution: Our CaaS Implementation, provides the customer a container size based on the size of the image/service they opt for. The customer is not given the privilege of choosing a customizable container size since CSC547 Cloud Computing Company is a startup and the developers have chosen to specify a static memory associated with each service being offered.

AWS ECS: AWS ECS Fargate instance allows us to customize the size for our container. The Docker daemon reserves a minimum of 4 MiB of memory for a container, so we should not specify fewer than 4 MiB of memory for our containers. The restriction which AWS specifies is that the total amount of memory reserved for all containers within a task should be lower than the task memory value. We can specify a hard limit or a soft limit(memory limit) and the container will request upto the limit specified. The container is killed if hard limit is chosen and exceeded.


Edit container

Standard

Container name* CaaS

Image* nginx:latest

Custom image format: [registry-url]/[namespace]/[image]:[tag]

Memory Limits (MiB) Soft limit 512

Add Hard limit

Define hard and/or soft memory limits in MiB for your container. Hard and soft limits correspond to the `memory` and `memoryReservation` parameters, respectively, in task definitions.
ECS recommends 300-500 MiB as a starting point for web applications.

Port mappings Container port 80 Protocol tcp

Add port mapping

* Required

Fig 8.1 Container being configured on AWS fargate

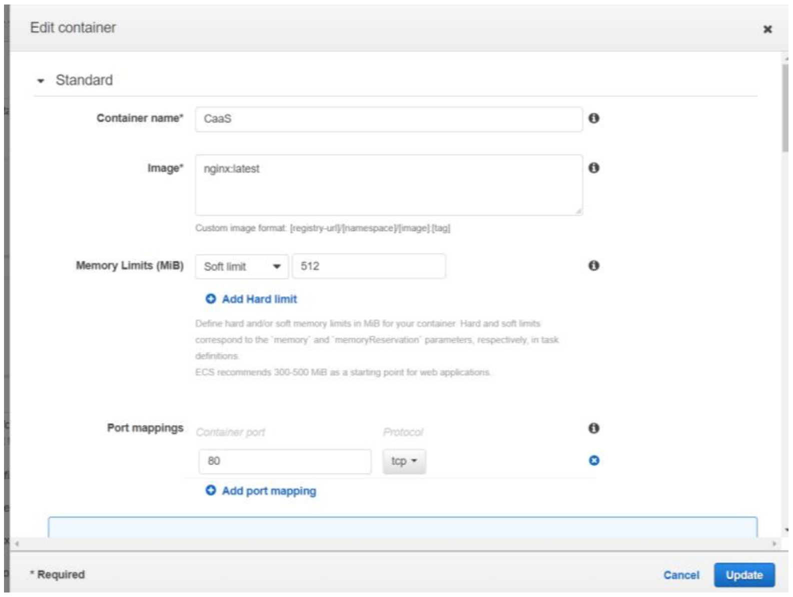
VCL IaaS: Size is not customizable by the end user in VCL implementation. The customer is allocated a VM based on the image he/she wants to reserve. The user does not get an option to configure the CPU and memory for the allocated VM.

Hence it is better to choose AWS ECS if customizable container size is of importance.

ii. Setup and Initialization Time

CaaS Solution: The management node configures the requested container
service and provides the client an IP and a port number to the client. The user can request the reservation for a certain duration. Our setup and implementation in the lab is in a LAN environment. Hence the containers are spawned within 20 to 30 seconds. Once the reservation is complete, the client can see the reservation on his/her portal

AWS ECS: When we have the AWS Fargate type instance, the boot up time is scheduled to be less than 10 minutes. We have noticed the time it takes to create resources for our service is less than 2 minutes.
Steps:
Client needs to select a task within the service. A task corresponds to an application. Memory, CPU and network and other parameters are configurable. The client can also specify number of instances that need to be spawned for that particular task (application).
We used the Nginx image, allocated it a vcpu of 0.25,
memory of 0.5 GB, VPC network for isolation and default cluster.



Edit container

Standard

Container name*: CaaS

Image*: nginx:latest

Custom image format: [registry-url]/[namespace]/[image]:[tag]

Memory Limits (MiB): Soft limit 512

Add Hard limit

Define hard and/or soft memory limits in MiB for your container. Hard and soft limits correspond to the `memory` and `memoryReservation` parameters, respectively, in task definitions.
ECS recommends 300-500 MiB as a starting point for web applications.

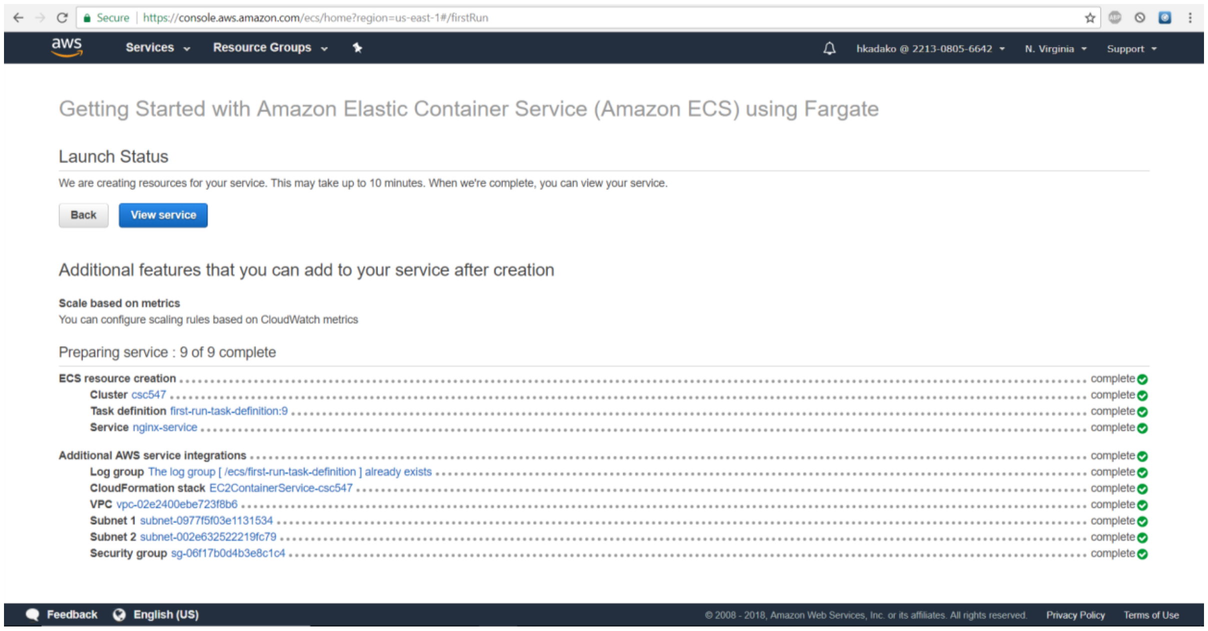
Port mappings

Container port	Protocol
80	tcp

Add port mapping

* Required

Fig 8.2 Container configs setup



Secure | https://console.aws.amazon.com/ecs/home?region=us-east-1#/firstRun
aws
Services
Resource Groups
hkadako @ 2213-0805-6642
N. Virginia
Support

Getting Started with Amazon Elastic Container Service (Amazon ECS) using Fargate

Launch Status

We are creating resources for your service. This may take up to 10 minutes. When we're complete, you can view your service.

Additional features that you can add to your service after creation

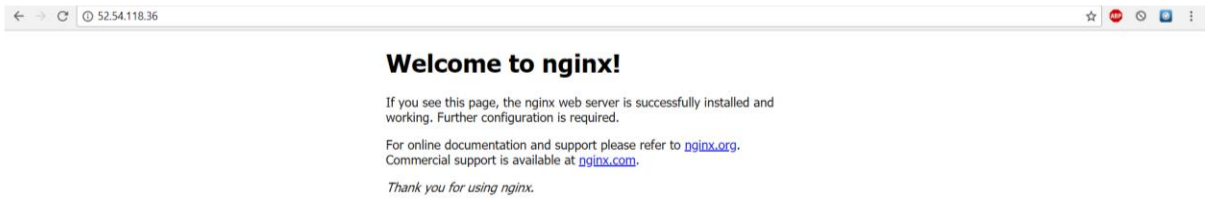
Scale based on metrics
You can configure scaling rules based on CloudWatch metrics

Preparing service : 9 of 9 complete

```
ECS resource creation ........................................ complete
    Cluster csc547 ........................................... complete
    Task definition first-run-task-definition:9 .............. complete
    Service nginx-service .................................... complete

Additional AWS service integrations .......................... complete
    Log group The log group [ /ecs/first-run-task-definition ] already exists ... complete
    CloudFormation stack EC2ContainerService-csc547 .......... complete
    VPC vpc-02e2400ebe723f8b6 ................................ complete
    Subnet 1 subnet-0977f5f03e1131534 ........................ complete
    Subnet 2 subnet-002e632522219fc79 ........................ complete
    Security group sg-06f17b0d4b3e8c1c4 ...................... complete
```

Feedback
English (US)
© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

Fig 8.3 After containers are spawned, it takes time until we can launch the services within the cluster


52.54.118.36

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Fig 8.4 Welcome page displayed after logging into the container's public IP

The EC2 launch type allows us to run containerized applications on a cluster of Amazon EC2 instances that we manage. We have not explored this feature for managing containers.

VCL IaaS:This implementation allows the user to select a base image, duration and time for the reservation.
The setup/initialization time in the case of VM reservations is around 3 minutes.

Using containers is advantageous as will take much less time.

iii. Web User Interface:
CaaS Solution: We provide a basic user interface for the customer to reserve and view his/her reservations.a container reservation and also view the status.

AWS ECS: AWS ECS has a very user friendly portal which has numerous options to configure the container service.

VCL IaaS: The client portal is not as developed as AWS. It provides the user a console to make reservation for the required VM.
Our CaaS implementation is a prototype and it provides limited number of features when compared to AWS.

iv. Usage Versatility

CaaS Solution: Our solution restricts the services given to a customer. This implementation has limited flexibility since the user cannot re-provision or stop and restart the containers when required. The number of services our solution currently offers is limited to 5 but can be extended to include more images using the registry in the future.

AWS ECS: Amazon ECS allows the flexibility to provide services through a number of different methods like AWS CLI, AWS SDK or the AWS website. Since they have a lot of instances to choose from, the user has numerous options. The client has an option to either choose AMI or to use any other Linux distribution as required. Container images can be pulled from docker
hub or Amazon Elastic Container Registry (ECR) that is a fully-managed Docker container registry that makes it easy for developers to store, manage, and deploy Docker container images.

VCL IaaS: VCL restricts the reservation request only through the portal. However, access to the VM is possible through CLI or RDP. VCL also provides the flexibility to add custom images to be used by the clients.

v. Registry

CaaS Solution: We use a local docker registry on our management node. The registry stores the images as required by the organization

AWS ECS: They use Elastic Container Registry (ECR) which is a fully-managed Docker container registry that makes it easy for developers to store, manage, and deploy Docker container images. The container images can also be pulled from other repositories like dockerhub.

VCL IaaS: Provides custom images that can be created using the imaging
reservation and pushed in the VCL registry.[5]

vi. Storage needs

CaaS Solution: We provide a remote as well as local storage facility to our customers. Customers intending to use a remote storage facility are currently given as profile 2.

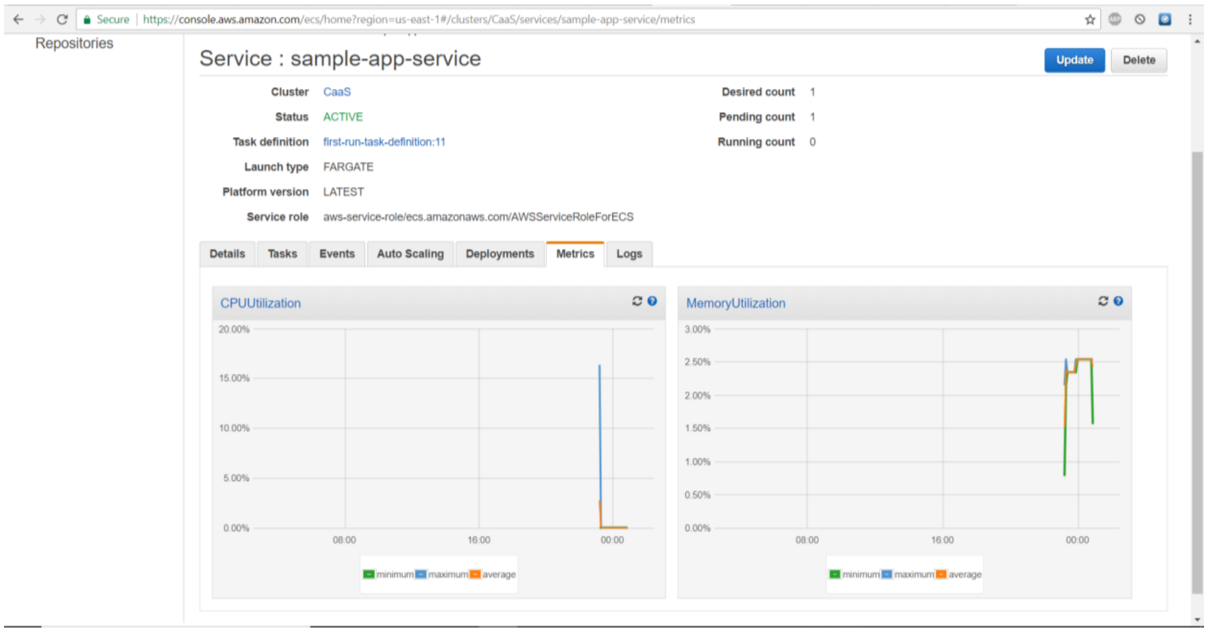
AWS ECS: Optimized AMIs from version 2015.09.d and later launch with an 8-GiB volume for the operating system that is attached at /dev/xvda and mounted as the root of the file system. There is an additional 22-GiB volume that is attached at /dev/xvdcz that Docker uses for image and metadata storage. Clients can add storage to the volume group that Docker uses by attaching a new EBS volume. [6]

VCL IaaS: Supports the storage using the Andrew File system [5]

vii. Monitoring and Logging

CaaS Solution: The admin is provided with the ability to monitor the health of the containers. We are keeping a record of the status by monitoring using docker health check commands. Container health is also monitored by running a service on the compute nodes. We can also extend it to use the top command to check the health of the underlying infrastructure.

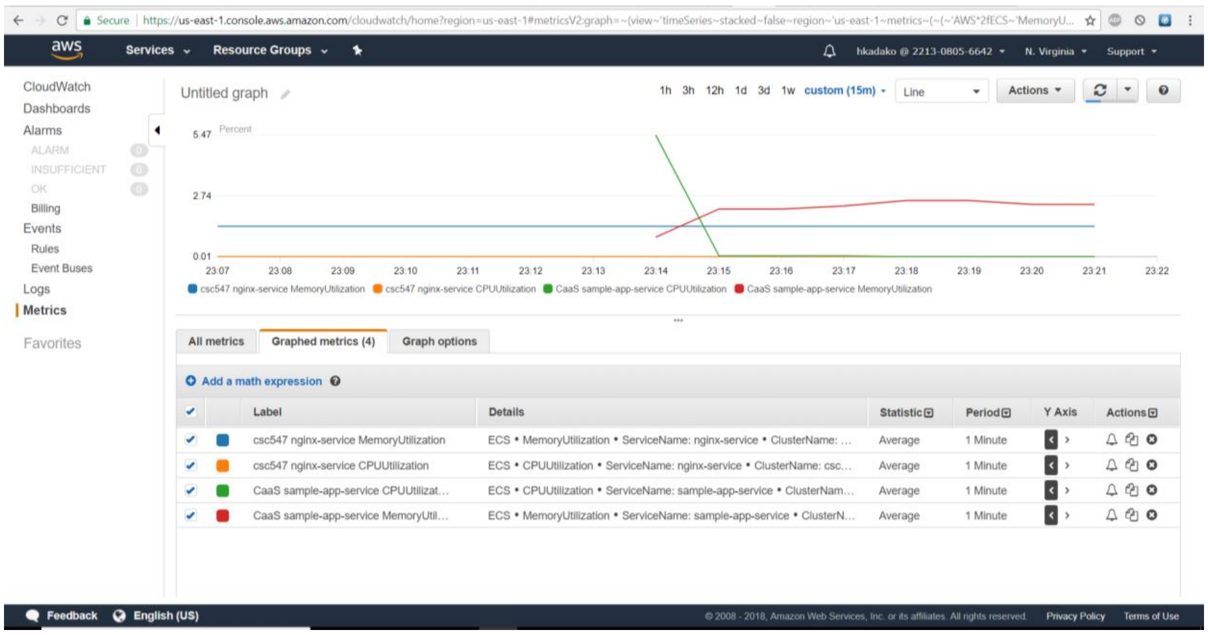
AWS ECS: The metric data is sent to CloudWatch at 1-minute periods and recorded for a period of two weeks. The available metrics for monitoring in Fargate are CPU Reservation, Memory reservation, CPU Utilization and Memory Utilization. We can monitor using the Clustername and Servicename. The values we see as a result are the percentage of CPU or memory utilization in a cluster or a service.
The user has the flexibility of using custom dashboards on cloudwatch. The user can monitor additional metrics by using cloudwatch scripts and can also set cloudwatch alarms and take actions if it reaches certain thresholds.
Containers can be configured to send logs to CloudWatch module and it also saves disk space on the container instances.



Service : sample-app-service

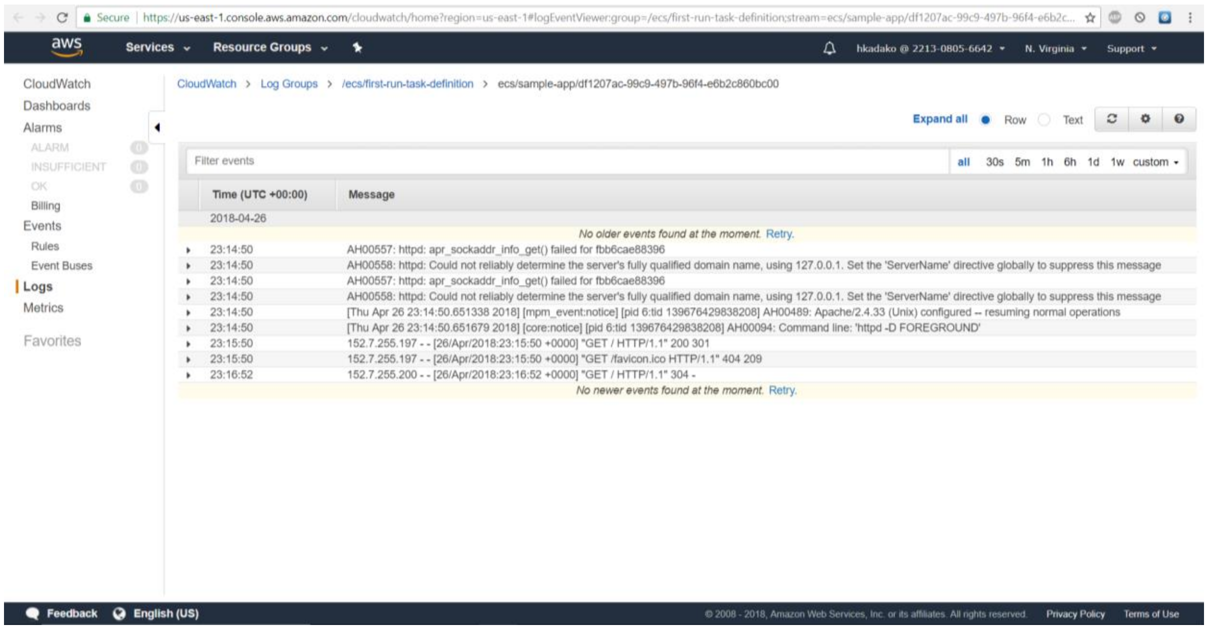
Cluster	CaaS	Desired count	1
Status	ACTIVE	Pending count	1
Task definition	first-run-task-definition:11	Running count	0
Launch type	FARGATE		
Platform version	LATEST		
Service role	aws-service-role/ecs.amazonaws.com/AWSServiceRoleForECS		

Fig 8.5 Monitoring CPU and memory utilization on ECS



✓		Label	Details	Statistic	Period	Y Axis	Actions
✓	🟦	csc547 nginx-service MemoryUtilization	ECS • MemoryUtilization • ServiceName: nginx-service • ClusterName: …	Average	1 Minute		
✓	🟧	csc547 nginx-service CPUUtilization	ECS • CPUUtilization • ServiceName: nginx-service • ClusterName: csc…	Average	1 Minute		
✓	🟩	CaaS sample-app-service CPUUtilizat…	ECS • CPUUtilization • ServiceName: sample-app-service • ClusterNam…	Average	1 Minute		
✓	🟥	CaaS sample-app-service MemoryUtil…	ECS • MemoryUtilization • ServiceName: sample-app-service • ClusterN…	Average	1 Minute		

Fig 8.6 Monitoring CPU and memory utilization on ECS using cloudwatch for two different services



CloudWatch > Log Groups > /ecs/first-run-task-definition > ecs/sample-app/df1207ac-99c9-497b-96f4-e6b2c860bc00

```
Time (UTC +00:00)   Message
2018-04-26
                    No older events found at the moment. Retry.
23:14:50            AH00557: httpd: apr_sockaddr_info_get() failed for fbb6cae88396
23:14:50            AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 127.0.0.1. Set the 'ServerName' directive globally to suppress this message
23:14:50            AH00557: httpd: apr_sockaddr_info_get() failed for fbb6cae88396
23:14:50            AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 127.0.0.1. Set the 'ServerName' directive globally to suppress this message
23:14:50            [Thu Apr 26 23:14:50.651338 2018] [mpm_event:notice] [pid 6:tid 139676429838208] AH00489: Apache/2.4.33 (Unix) configured -- resuming normal operations
23:14:50            [Thu Apr 26 23:14:50.651679 2018] [core:notice] [pid 6:tid 139676429838208] AH00094: Command line: 'httpd -D FOREGROUND'
23:15:50            152.7.255.197 - - [26/Apr/2018:23:15:50 +0000] "GET / HTTP/1.1" 200 301
23:15:50            152.7.255.197 - - [26/Apr/2018:23:15:50 +0000] "GET /favicon.ico HTTP/1.1" 404 209
23:16:52            152.7.255.200 - - [26/Apr/2018:23:16:52 +0000] "GET / HTTP/1.1" 304 -
                    No newer events found at the moment. Retry.
```

Fig 8.7 Logging on AWS ECS

VCL IaaS: In VCL, logging is provided through a combination of system logs, image usage statistics and operational profiles. For example, IBM Tivoli
Monitoring software can be used to assess image performance needs and thus make appropriate virtualization decisions. [7]

9. Future work

1.Provide SSH access to containers

Since by default, SSH is not enabled in the images we are providing currently, our future scope would be to provide the capability of SSH access to the images to our customers. For this, we will have to expose 22 port by building a new image using Dockerfile.[3]

2. Provide customizable container size to the customer during the time of reservation.

Currently we are not allowing the customer to choose his choice of size or CPU. In the future release, we can provide configurable specifications like AWS.

3. Provide more services to the customer

In the future, we plan to add more services which are available to the customer

10. Schedule:

TASK	START DATE	END DATE
Research about Docker and understand the working and installation	3/17/2018	3/23/2018
Initial UI design plan	3/23/2018	3/27/2018
Determine the system requirements and the cloud network setup	3/27/2018	4/1/2018
Install Ubuntu 16.04 and Docker on lab machines	4/2/2018	4/7/2018
Set up network functionalities on lab machines	4/7/2018	4/8/2018
Setting up database and docker registry on digital ocean	4/2/2018	4/9/2018
Decide the services offered to the customer	4/7/2018	4/11/2018
User container reservations	4/13/2018	4/18/2018
User login and signup	4/17/2018	4/19/2018

Show and delete reservations	4/19/2018	4/22/2018
Container health check (monitoring)	4/19/2018	4/22/2018
Network and scheduling components	4/15/2018	4/22/2018
User Billing	4/22/2018	4/23/2018
storage requirements implementation	4/13/2018	4/24/2018
Verification and validation	4/22/2018	4/25/2018
Performance comparison with AWS and VCL	4/22/2018	4/26/2018
Migration to lab machine	4/22/2018	4/25/2018
Documentation	4/20/2018	4/26/2018

Table 2. Schedule table

11. Tasks and Responsibilities:

TASKS	OWNER	RESPONSIBILITIES
UI design	Bhushan	All members
Determine the system requirements and the cloud network setup	Ragavi	All members
Install Ubuntu 16.04 and Docker on lab machines	Gokul	All members
Set up network functionalities on lab machines	Gokul	All members
Setting up docker registry on lab machines	Pavithra	All members
Decide the 5 services offered	Harshini	All members
Set up database schemas	Kashish	All members
User container reservations	Pavithra	All members
User login and signup	Harshini	All members

Show and delete reservations	Pavithra	All members
User and admin dashboard functionalities	Bhushan	All members
Monitoring	Bhushan	All members
Network components	Gokul	All members
Scheduling components	Ragavi	All members
Load Balancing Strategy	Kashish	
Research/Development for storage components	Kashish	All members
User Billing	Pavithra	All members
Verification and validation	Ragavi	All members
Migration to lab machine	Pavithra	All members
Performance comparison with AWS and VCL	Harshini	All members

Table 3. Tasks and responsibility

12. References

[1]https://github.com/docker/docker.github.io/blob/master/registry/deploying.md
[2]https://stackoverflow.com/questions/38695515/can-not-pull-push-images-after-update-docker-to-1-12
[3]https://docs.docker.com/engine/examples/running_ssh_service/
[4]https://docs.docker.com/engine/reference/commandline/inspect/
[5]NCSU, “File and Web Services”, EOS, as available on www.eos.ncsu.edu/services/
[6]https://github.com/awsdocs/amazon-ecs-developer-guide/blob/master/doc_source/ecs-ami-storage-config.md
[7]Sam Averitt, Michael Bugaev, Aaron Peeler et al, “Virtual Computing Laboratory (VCL)”, International Conference on Virtual Computing Initiative, May 8th 2007
[8]https://docs.docker.com/
[9]https://websiteforstudents.com/setup-ssh-server-key-authentication-ubuntu-17-04-17-10/
[10]Amazon, “What is Cloud Computing?”, AWS, as available on aws.amazon.com/what -is -cloud -computing/
[11]Ofir Nachmani, “5 Key Benefits of Docker”, DZone, April 29th 2015, as available on dzone.com/articles/5-key-benefits-docker-ci
[12]Docker, “What is a Container”, as available on www.docker.com/what-container

Appendices

1. **Registry setup command -**
```
docker run -d -p 5000:5000 --restart=always --name registry
registry:2
```

2. **Adding images to registry -**
```
sudo docker pull <image name>:latest
sudo docker tag <image name>:latest 192.168.11.1:5000/<image
name>
sudo docker push 192.168.11.1:5000/<image name>
sudo docker image remove 192.168.11.1:5000/<image name>
```

3. **Pull docker image from the registry in compute node -**
```
sudo docker pull 192.168.11.1:5000/nginx
```
4. **Run docker container -**
```
sudo docker run -d -p <compute port>:<image port> <image name>
```
5. **Container health check -**
```
sudo docker inspect <container uuid>
```
6. **Create public-private key pair -**
```
ssh-keygen -t rsa
```
7. **Connect MariaDB**
```
mysql -u <username(caas)> -p <database name(csc547caas)>
```
8. **Iptables rules -**
```
iptables -t nat -I PREROUTING -p tcp -d <mgmtip> --dport
<natport(mgmtnode)> -j DNAT --to-destination <compute
node private ip>:<compute node port>

iptables -I FORWARD -m state -d <Compute Node Network
Addr> --state NEW,RELATED,ESTABLISHED -j ACCEPT
```
9. **Storage -**
```
docker-machine mount dev:/home/docker/foo foo
docker run -v /root/local:/root/remote <hello-world>
docker-machine mount -u vm:/root/data1 /root/data
```