

# **Deployment as a Service on Hybrid Cloud**

A Project Report  
Presented to  
The Faculty of the College of  
Engineering  
San Jose State University  
In Partial Fulfillment  
Of the Requirements for the Degree  
**Master of Science in Software Engineering**

By  
**Arushi Gangrade**  
**Sakshi Gangrade**  
**Dhruv Kalaria**  
**Vivek Maheshwari**

December, 2016

Copyright © 2016  
**Arushi Gangrade**  
**Sakshi Gangrade**  
**Dhruv Kalaria**  
**Vivek Maheshwari**  
ALL RIGHTS RESERVED

**APPROVED**

DocuSigned by:  
*Rakesh Ranjan*  
A1B50AC7ED01466...

---

**Prof. Rakesh Ranjan**, Project Advisor

---

**Prof. Dan Harkey**, Director, MS Software Engineering

---

**Prof. Xiao Su**, Department Chair

## ABSTRACT

### **Deployment as a Service on Hybrid Cloud**

By

Arushi Gangrade, Sakshi Gangrade, Dhruv Kalaria, Vivek Maheshwari

Cloud computing has become one of the biggest trends in the software industry. It has promising potential of elasticity and efficient performance. According to a survey by Evans Data conducted in 2015 - 5 million developers around the globe use cloud computing services for developing products. It is important that the developer chooses the best cloud platform based on the product requirement. They should be able to seamlessly deploy their applications without worrying about the setup and the hassles involved.

Currently, the developer has to manually configure and deploy multiple modules individually which is time consuming and error prone on different cloud platforms. Also it is difficult to scale and automate such kind of deployment. Hence, a service is needed that can abstract the complexities from the developer and make application up and running within few minutes. Configuring servers and resolving the dependencies of environment for application to run takes up a lot of time while delivering solutions to the customer. Without automating this process, the time to market time increases which is not suitable for the agile development.

Deployment as a Service on Hybrid Cloud is a solution where the developer can leverage various cloud platforms for deployment of the application without knowing the deployment process. We are designing a platform which can deploy micro-services to multiple cloud platforms with Kubernetes as Container Orchestration and Docker as container engine. Our platform will help the enterprise to move from one cloud to another or private cloud to public cloud wherein they already have their infrastructure running with Kubernetes and Docker. The platform reads the current configuration in the current cloud and will replicate a similar environment on the destination cloud with custom modifications by the user. On top of service, we also provide network and storage management layer so that a user could plug-and-play NFS or portable storage to different clouds. For the scope of this project we plan to deploy our platform on AWS and help enterprise to migrate from Google Cloud to AWS. Once deployed the platform will help to manage individual applications and scale up and scale down.

### **Acknowledgments**

We would like to thank our project adviser Prof. Rakesh Ranjan for his constant support, coordination and encouragement during the course of the project. We are grateful to him for helping us in deciding on the project topic, experimenting with new technologies and thinking in a different way.

We also acknowledge the guidance of Prof. Dan Harkey, our project instructor, in helping us identify the mistakes that we made and providing us with valuable recommendations in rectifying them. We also extend our sincere thanks to Prof. Vicki Parrish and Dr. Robinson for their insights in technical writing, which helped us prepare the documentation for our project.

Finally, we are grateful to all the teaching and non-teaching staff members of Software Engineering and Computer Engineering Department and to our fellow classmates who directly or indirectly contributed for the success of our project.

## Table of Contents

<b>Chapter 1. Project Overview .....</b>	<b>1</b>
Introduction .....	1
Proposed Areas of Study and Academic Contribution .....	2
Current State of the Art .....	3
<b>Chapter 2. Project Architecture .....</b>	<b>4</b>
Introduction .....	7
System Flow .....	8
Architecture Subsystems .....	9
<b>Chapter 3. System Requirements and Analysis .....</b>	<b>12</b>
Business and Domain Requirements .....	12
Non Functional and Performance Requirements .....	13
Technology and Resource Requirements .....	14
Client Technologies .....	15
Middle-Tier Technologies .....	16
Data-Tier Technologies .....	19
<b>Chapter 4. Project Design .....</b>	<b>20</b>
Client Design .....	20
Middle-Tier Design .....	23
Data-Tier Design .....	27
<b>Chapter 5. Project Implementation.....</b>	<b>28</b>
Middle-Tier Implementation: .....	41
Data-Tier Implementation .....	46
System Implementation Summary .....	47
Tools and Technologies Used .....	49
<b>Chapter 6. Performance and Benchmarks .....</b>	<b>50</b>
<b>Chapter 7. Deployment, Operations, Maintenance .....</b>	<b>51</b>
<b>Chapter 8. Testing and Verification.....</b>	<b>52</b>
Test Environment Set Up .....	52
Testing Plan and Test Design .....	52
<b>Chapter 9. Summary, Conclusions, and Recommendations.....</b>	<b>67</b>
Summary .....	67
Conclusions .....	67
Recommendations for Further Research .....	68

**Glossary .....69**

**References .....70**

**Appendices .....71**

    Appendix A. ....71

## List of Figures

<b>Figure 1. Creating Kubernetes Cluster .....</b>	<b>4</b>
<b>Figure 2. App Deployment on Kubernetes Cluster .....</b>	<b>5</b>
<b>Figure 3. Understanding Pods .....</b>	<b>5</b>
<b>Figure 4. Exploring the Pods and Nodes .....</b>	<b>6</b>
<b>Figure 5. Exposing the apps using API .....</b>	<b>6</b>
<b>Figure 6. Project Architecture .....</b>	<b>8</b>
<b>Figure 7. Client Design – User Registration.....</b>	<b>20</b>
<b>Figure 8. Client Design - Migration from existing Kubernetes Cluster.....</b>	<b>21</b>
<b>Figure 9. Client Design –Migrating Dockers from Local Environment.....</b>	<b>22</b>
<b>Figure 10. Middle-Tier Design-Migration from existing Kubernetes Cluster ..</b>	<b>23</b>
<b>Figure 11. Sequence Dig Migration from existing cloud to another cloud .....</b>	<b>24</b>
<b>Figure 12. Middle-Tier Design- Migrating Dockers from Local Environment</b>	<b>25</b>
<b>Figure 13. Sequence Diagram-Migration to a cloud.....</b>	<b>26</b>
<b>Figure 14. Data-Tier Design .....</b>	<b>27</b>
<b>Figure 15. Client Architecture Component Diagram.....</b>	<b>28</b>
<b>Figure 16 – Sign Up Page.....</b>	<b>29</b>
<b>Figure 17 – Sign IN Page.....</b>	<b>30</b>
<b>Figure 18 –Kubernetes Dashboard without projects.....</b>	<b>31</b>
<b>Figure 19 –Kubernetes Cluster Visualization-I.....</b>	<b>32</b>
<b>Figure 20 –Kubernetes Cluster Visualization-II.....</b>	<b>32</b>
<b>Figure 21 –Kubernetes Cluster Visualization-III.....</b>	<b>33</b>



**Figure 22 – Add Project – Project general Information – 1st Project.....34**

**Figure 23 – Add Project – Kubernetes Infrastructure Information.....35**

**Figure 24 – Add Project – Cluster creation Status Information.....36**

**Figure 25 – Add Project – Download Private Key.....36**

**Figure 26 – Add Project – Deploy Application.....37**

**Figure 27 – Add Project – Deploy from existing cluster.....38**

**Figure 28 – Add Project – Confirm Deployment and Services.....38**

**Figure 29 – Add Project – Deploy app loading and app URL generation.....39**

**Figure 30 – Kubernetes Page.....40**

**Figure 31. Architecture Component Diagram.....41**

**Figure 32. Migrating Docker from Local Environment.....43**

**Figure 33. Migrating from Existing Kubernetes Cluster.....45**

**Figure 34. Data Tier Access Implementation.....46**

**Figure 35: Tools and Technologies Summarized.....49**

**List of Tables**

**Table 1. Domain Requirements.....13**

**Table 2. Non Functional Requirements.....14**

## **Chapter 1. Project Overview**

### **Introduction**

The way Enterprises are delivering software solutions is going through a wave of change as the requirements, operating environments, technologies and customer needs have become mercurial. With increase in the use of cloud computing in the industry for scalable and reliable solutions, it is important that developers can seamlessly deploy their applications on cloud. Different research works are being done to simplify the process of continuous integration and continuous deployment. There are many cloud operators in market which offer different benefits to name a few like performance, cost, ease of scalability, storage location, security etc. To leverage the use of all of these cloud vendors and deploy the software solutions in Hybrid Cloud benefits the organization.

To deliver products faster, the gap between development of product and deploying the product should be reduced. This bridge can be reduced by using deployment automation tools and cloud based resource providers. <sup>[1]</sup>

With the use of hybrid cloud come a great lot of advantages but there are also challenges which needs to be taken care of too. Some of the challenges are increased latency, cross cloud communication, automation orchestration, availability of the cloud, etc. But with the rise of micro service architecture we can move to hybrid cloud. We can develop more fault tolerant systems with this architecture with graceful degradation in case of any fault in any of the services.

Enterprises are facing issues such as expensive vendor integrations, or repairing together multiple solutions. There are certain problems associated with the current deployment model:

- With presence of various cloud vendors, their usability and offerings in terms of technology, scalability, security, capacity, low-cost or customer service, the user would like to leverage the most suitable cloud as per the applications requirement for its deployment. But, often the user loses the control over cloud environment and ends up following the tools and technologies dictated by the cloud providers.

- Software Engineers often have to spend hours specifying the complexities of a specific cloud environment.
- Configuring servers and resolving the dependencies of environment for the application to run generally takes up a lot of time while delivering solutions to the customer.
- Manually configuring and deploying whole application as well as any patches later becomes error prone and time consuming.
- In short, the enterprises are locked with a specific vendor.

Deployment as a Service on Hybrid Cloud is a solution where the developer can leverage various cloud platforms for deployment of the application without knowing the deployment process. The platform provides option for the enterprise to deploy or migrate to a public cloud. We are using Kubernetes as backend service to deploy Docker containers. The platform is capable of reading the existing Kubernetes configuration from the currently deployed environment and could replicate in a different destination cloud. Also, the platform provides capabilities to accept Docker configuration file and deploy through Kubernetes. The platform is scalable as it creates separate Kubernetes instance on for each project and we call it management server for that cluster.

Currently, a huge gap exists between cloud providers and we are trying to fill the gap using Kubernetes to help enterprise take advantage of multiple cloud platforms.

### **Proposed Areas of Study and Academic Contribution**

As a part of working on this project, we solving the cloud migration problem involving multiple clouds of an enterprise. We studied various tools which could help us in achieving our goal to deploy and migrate micro-services deployed in containers. Some of the tools we studied are Kubernetes and Docker Swarm. After some research, we went forward with Kubernetes as it is more widely accepted orchestration engine by enterprises and hence more enterprises could benefit from our platform.

## **Current State of the Art**

In recent years, containers have gained popularity amongst developers and IT organizations by providing a lightweight solution for dependencies and configurations while moving the application between different computing environments. They are used to package the application, dependencies and configuration file in an isolated space which eventually provides run time environment for the application. Docker is a technology that leverages these containers for wrapping a piece of software in a complete filesystem that contains everything needed to run the code regardless of the environment<sup>[2]</sup>. These docker containers help the enterprises achieve platform independence by running on any cloud and infrastructure.

Management of docker containers is very complex. This is where Kubernetes comes into play for managing docker containers across cluster of nodes<sup>[3]</sup>. Kubernetes is used to schedule deployments and scale applications. Kubernetes basically groups containers making an application into a logical unit which can be easily discovered and managed. Kubernetes is portable, self-healing and extensible.

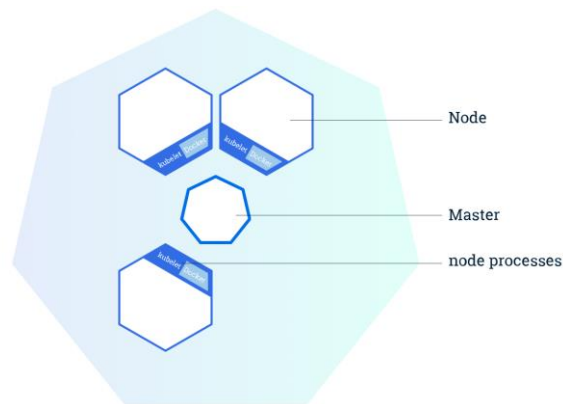
Fabric8 is an integration platform that provides java client for managing kubernetes clusters. This client allows management of different components of kubernetes and leveraging Kubernetes services. The GitHub page<sup>[4]</sup> provides all the information of this java client along with the usage details.

## Chapter 2. Project Architecture

### Kubernetes Basics

#### 1. Creating the cluster:

Kubernetes is known for efficiently managing the deployment of containerized application to a cluster. A Kubernetes cluster is a collection of nodes that has Kubernetes installed on them and it consists of node/nodes, master, and node processes.



*Figure 1. Creating the Kubernetes Cluster*

The node consists of either Docker/rkt to perform all container related functionalities. Jobs like communication with master and node management are carried out by Kubelet for each node. The process of deployment begins by starting the containers which run on nodes. Kubernetes APIs enable communication between the cluster nodes and master.

#### 2. App deployment using Kubernetes:

After setting up the Kubernetes cluster, the apps are deployed on it using Kubernetes Deployment.

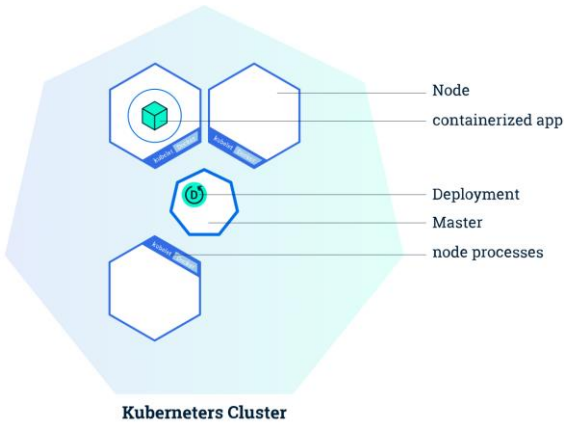


Figure 2. App Deployment on Kubernetes Cluster

For deploying the app, container name and number of times the server has to be replicated is needed which can be updated later.

3. Exploring Pods and Nodes:

After you deploy the app, the instance of this app is hosted on a Pod. Such pods consist of multiple containers that have same IP and port number.

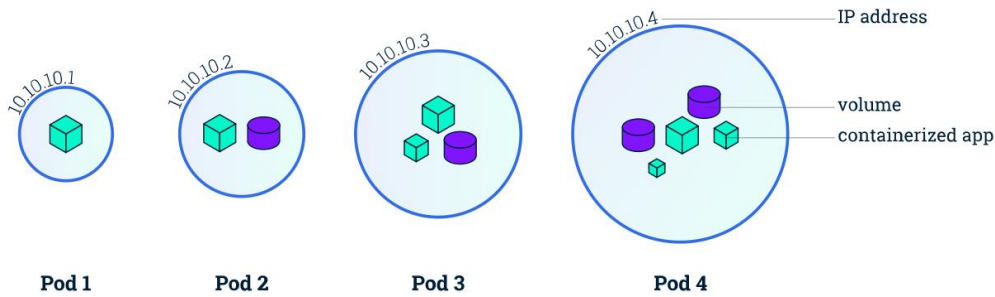


Figure 3. Understanding Pods

A Node consists of pods. Each of them has a running Kubelet along with a container runtime to manage communication with master and nodes and run the application.

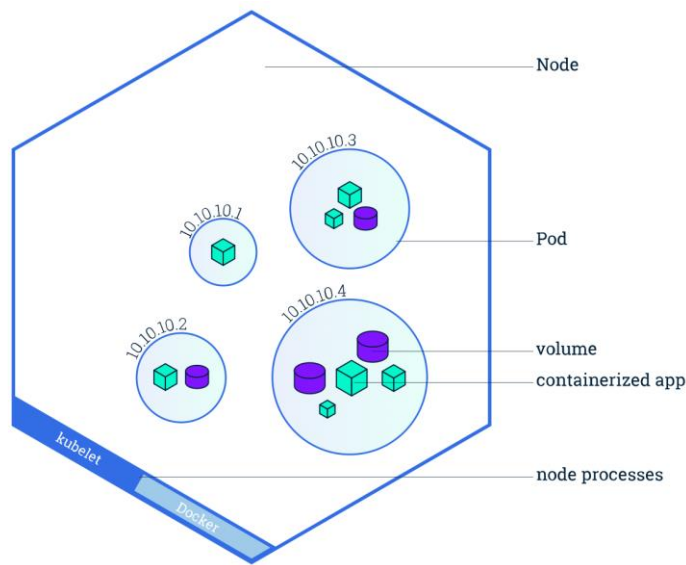


Figure 4. Exploring Pods and Nodes

4. Exposing the app using APIs:

The IP addresses of Pods cannot be accessed from outside the Kubernetes. Kubernetes Service comes into play for exposing these pods outside the cluster. These services when exposed can provide load balancing and Node port.

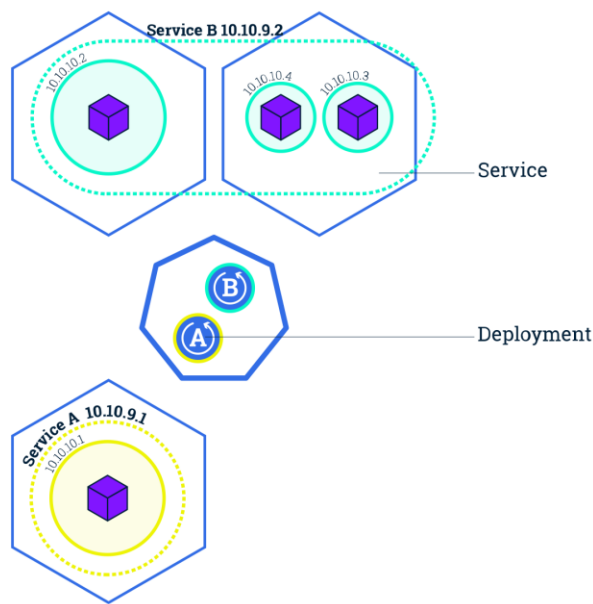


Figure 5. Exposing the apps using API



## **5. Scaling the app:**

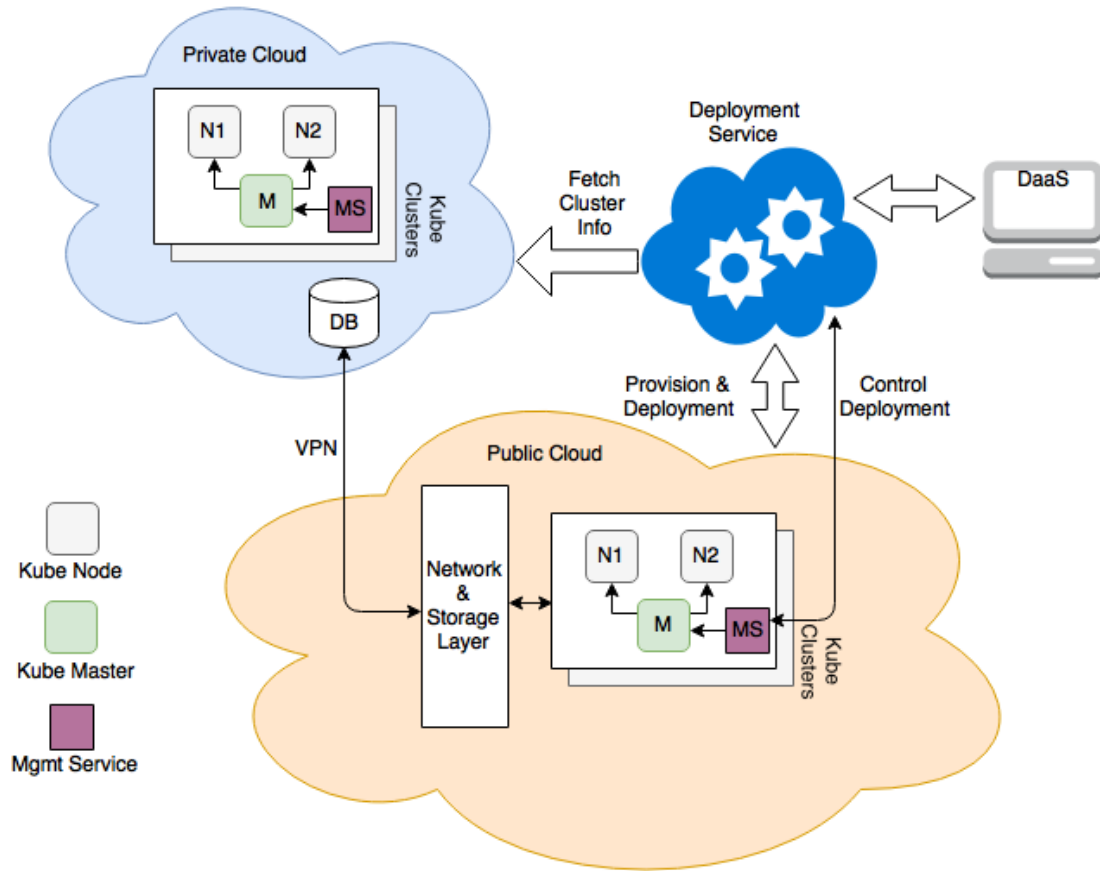
With an increase in number of hits on the application, scaling can be achieved by increasing the replication of clusters. While scaling up will result in increased Pods, scaling down will result in reduction. Using the load balancer that comes as part of Kubernetes Service, the incoming traffic can be divided amongst the Pods.

## **6. Updating the app:**

Kubernetes provides a way through which the application can be used by the user while the developer deploys the new features or updates. This is made possible through “Rolling Updates” where pod instances are updated one after the other. This helps achieve CI and CD, migration between environments, and switch between different versions.

## **Introduction**

DaaS is platform as a service solution for the enterprises, where any user could create a project, read the configuration of the existing kubernetes cluster or provide a docker image to deploy, select the destination cloud, create a kubernetes cluster and deploy the application. Once the application is deployed the user could manage and monitor the applications and scale out or scale in based on the requirements. Also, the platform provides monitoring of the application logs through ELK stack and will display real time information about the current events and health of the application.



*Figure 6. Project Architecture*

### System Flow

For the platform we are using kubernetes as the container orchestration engine and docker as the container technology. Following provides the happy flow path logical steps, the user will follow in order to migrate or create a brand new deployment on the selected cloud environment on top of kubernetes cluster.

- 1) User Registration
- 2) Provide Cloud Credentials of the target cloud environment.
- 3) Provide the master URL and credentials to read the cluster configuration on existing cloud or data center.

- 4) Edit the retrieved information and select the infrastructure on which the cluster needs to be created.
- 5) Confirm the selected resources and create cluster.
- 6) On successful cluster creation, deploy the application.

We are targeting to deploy micro-services and kubernetes as its orchestrator. The main advantage of using kubernetes is it provides fault tolerance based if any of the docker containers crash. Kubernetes will on its own restart the container service. Also, it is very easy to scale the application in case the load on the application increases and scale down back. Kubernetes is production ready distributed system and our platform gives an option for the enterprises to move their docker containers on top of kubernetes in the cloud of their choice.

As a part of the process, we are taking the cloud credentials from the user. Since these are very secure credentials we have made a design choice to never store the credentials anywhere on disk for the purpose of future retrieval. We will ask the user to provide the users to provide the secret keys in order to do any type of infrastructure update. These credentials are not required for docker scale up or scale down as they are deployed on top of kubernetes but are required only when the user does any cloud specific activities like increasing the VMs, creating a network, etc.

## **Architecture Subsystems**

### **1. Web-based UI – DaaS:**

The project provides a web based user interface (dashboard) which will allow an enterprise to migrate their containerized applications to a Kubernetes cluster on a public cloud. The applications can either be migrated from an enterprise own IT environment or from a public cloud provider used by the enterprise. The web app can also be used to get an overview of applications running on the cluster with some additional relevant information.

## **2. Deployment Service:**

The deployment service is basically the backend of our project. The service is supposed to get the configuration information about the cluster and the containerized application which is to be deployed on the public cloud either through the web user interface or an existing Kubernetes cluster. Once the service has the cluster information, it set-ups the same cluster on the public cloud environment and make sure the containerized applications are up and running as expected. Thus, the deployment service will be used to provision the migration of cloud environment to public cloud. Also, the deployment service will have a RESTful architecture to support api's for the web user interface.

## **3. Current Cloud Environment:**

The current cloud environment can either be a local IT environment or a cloud environment from which the enterprise applications are to be moved/migrated. For either case there are certain requirements which needs to be considered. If the cloud environment is local, then the Kubernetes cluster information will be provided via the web user interface. While for an existing cloud deployment, the deployment must be done via Kubernetes and necessary information about the cloud provider or the cluster must be provided. The deployment service will get the Kubernetes cluster information from this cloud environment. Further, the Kubernetes cluster will have a management service, a kube master node and can have many kube nodes.

## **4. Public Cloud Environment:**

The public cloud environment is where the containerized cluster application is to be finally deployed. The deployment service set-ups the required configuration for the Kubernetes cluster on this cloud environment. If the cloud migration is from local environment to cloud than the deployment service sets up the Kubernetes cluster with given configuration on this cloud. While for migration from existing cloud environment to public cloud, the Deployment service set-ups the same Kubernetes cluster as on

previous cloud with the user's consent. The kubernetes cluster can also be scaled up on the public cloud while migrating. Also, there's an additional Network and Storage layer for connection to database or internal apps which are still in the local cloud environment via VPN to the public cloud environment.

## Chapter 3. System Requirements and Analysis

### Business and Domain Requirements

This section briefs about the domain requirements of our project. These requirements typically drive the application architecture of a system. Thus, it focuses on the functioning of the system and its components. The table below (Table 1) describes these requirements -

Req. ID	Priority	Requirement	Description
1.	Essential	User Registration	The application should allow the user to register and create a profile.
2.	Essential	Project Dashboard	The application should allow its registered users to create and manage projects.
3.	Essential	Cloud Infrastructure and Configuration	The application should allow the users to attach the cloud related configurations for both cloud infrastructures.
4.	Essential	Resource Allocation on Cloud	The application should allocate the requested cluster resources by User to the requested Cloud provider.
5.	Essential	Deploy the Application	The application should deploy the user's application/project on the selected cloud with requested cluster configurations.
6.	Optional	Attach a NFS or Storage	The application supports the

		Infrastructure	feature of attaching a Network file system or a portable storage infrastructure to the deployed infrastructure.
--	--	----------------	---

Table 1. Domain Requirements

Non Functional and Performance Requirements

This section briefs about the non-functional requirements of the project. Non-functional requirements are basically the criteria which helps evaluate the operation of system as opposed to the specific behaviors (functional requirements). The table below (Table 2) briefs about the non-functional requirements for our project.

Req. ID	Non-functional Requirement	Description
1.	Availability	It is the degree to which the system is in a specified operable state. The system should therefore be available at any random time.
2.	Scalability	In general, it is the capability of the system to handle a growing amount of work. The system should be flexible enough to manage the users of the system.
3.	Security and Privacy	One of the most important non-functional requirement is Privacy and Security of the system. The system shall ensure the privacy of the user’s data while the system should be secure enough to protect user’s data from any harm. Also, certain measures like Backup can be used to recover any data in case of any harm to data.

4.	Performance, Throughput and Latency	The system should have high performance with minimum response time and high rate of processing work i.e. throughput with zero or minimum latency.
5.	Software Testability	It defines the degree to which the system supports testing in certain context. A test driven development helps improve the testability of software components.
6.	Maintainability and Reliability	The system should be reliable to use and should be easy to maintain. Also, a reliable system proves to be more cost-effective.
7.	Extensibility	The system should consider future growth into consideration. It can be through new features or through modification of existing ones.

**Table 2. Non-functional Requirements**

### **Technology and Resource Requirements**

The project, Deployment as a Service is a platform as a service and is a full-stack project, thereby requires certain technologies from every stack –

- Client Side Development (UI)
- Server Side Development (Systems)
- Databases
- Cloud Computing



## Client Technologies

For the platform we needed an extensive yet easy to use interface, to make the deployment process simpler for an individual or an enterprise. The technologies used for building the user interface are-

- **Angular JS** - Angular JS is a structural framework that is used to create dynamic web apps. It is used with HTML as the main template language and allows you to expand the HTML syntaxes in a way that would most aptly demonstrate the features of the web platform. The data-binding feature of AngularJS drastically reduces the code to be written to achieve the dependency. AngularJS also trains the browser to advanced syntaxes using constructs, which are referred as *derivatives*. We are using AngularJS as it is built around the idea that declarative code is more efficient than imperative code when developing UI and threading the software components along.



- **HTML5** – HTML is used to create the basic user interface. It basically devises the outlining of the webpages and using the tags, it represents the components of the webpage.
- **Bootstrap and CSS** - CSS is used for making the basic design and structure for the web pages. Bootstrap is a framework design, which provides design templates for various kinds of web components, web pages and applications. It is used alongside CSS and HTML and also provides a number of JavaScript extensions. It is only used for Client side development. Bootstrap is open source framework and

we are using it for interface development with responsive web pages, which behave consistently across all the devices.



- Kibana and D3 – Kibana will be used as a browser based analytics for the project dashboard in our application. Kibana is basically a search dashboard for Elasticsearch. Also, we'll be using D3 JS for creating interactive visualizations for relevant cluster data.

### **Middle-Tier Technologies**

The backend of our platform is build using Java. We are using Jersey JAX-RS client to develop the platform in a simpler yet efficient way. Also, we'll be using Java's AWS sdk for cloud provision and deployment and Elasticsearch, Logstash for searching an analyzing cluster data. Further, we'll be using Kubernetes and Fabric8.io java client to provision kubernetes cluster.

- Elasticsearch – It is an open source distributed and RESTful search engine which is based upon Apache Lucene. It is best suited to search all kinds of documents. Since it is a distributed platform, the indices can be sharded with zero or more replicas. It also provides near real time search and further supports multi-tenancy. Elasticsearch is a documented oriented search engine so it doesn't require a schema definition upfront and schema can be defined per type of customization of the indexing process.



Kubernetes – Kubernetes is a container cluster manager, which aims at providing scalable automated deployment platform across clusters. Kubernetes is based on small building blocks, termed as “*primitives*” which altogether support the mechanism of deployment, maintenance and scalability for the applications. The extensibility of Kubernetes is offered by the Kubernetes API, which can be used both internally and externally. The Kubernetes API uses RESTful interface hence allowing it to connect and use a large number of libraries readily. Some main components of Kubernetes are -

- Pod – Pods are the basic units of Kubernetes and they consist of one or more clusters located on a common host machine and share the resources.
- Labels and Selectors- Kubernetes offer the internal components and users to bind a key-value pair to the API objects, like nodes and pods called *Labels*. Hence labels and selectors are the resultant queries for matching labeled objects.
- Controller- A controller manages pods and leads the cluster from the actual state to the desired state and does so as a reconciliation loop.



Docker- Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries – anything that can be installed on a server. This guarantees that the software will always

run the same, regardless of its environment. Docker is also an open source project and it makes use of the resource isolation features enabled by Linux kernel hence allowing independent containers to run singularly on a Linux instance, rather than going through extra work of initializing, running and maintaining the virtual machines. Docker implements light weight containers and they are built over Linux kernels; they do not need a separate operating system or hypervisors.



Fabric8.io – Fabric8.io is an open source microservices Integration Platform which allows using a java client for handling and managing the Kubernetes clusters. Fabric8 eases up the process of deployment of a Java integration solution on multiple machines, processors, containers and JVMs. Each container maps to a Kubernetes pod which primarily contains a docker container. Also every pod of the Kubernetes gets a unique IP address. Also Fabric8 supports Git versioning for all the configurations and it implements a distributed git fabric hence no single point for configuration change loss or failures. Fabric8 also allows to re-utilize the features of docker by creating new containers using Docker container provider, which also enables the Docker Remote API to manage the containers.



**Data-Tier Technologies**

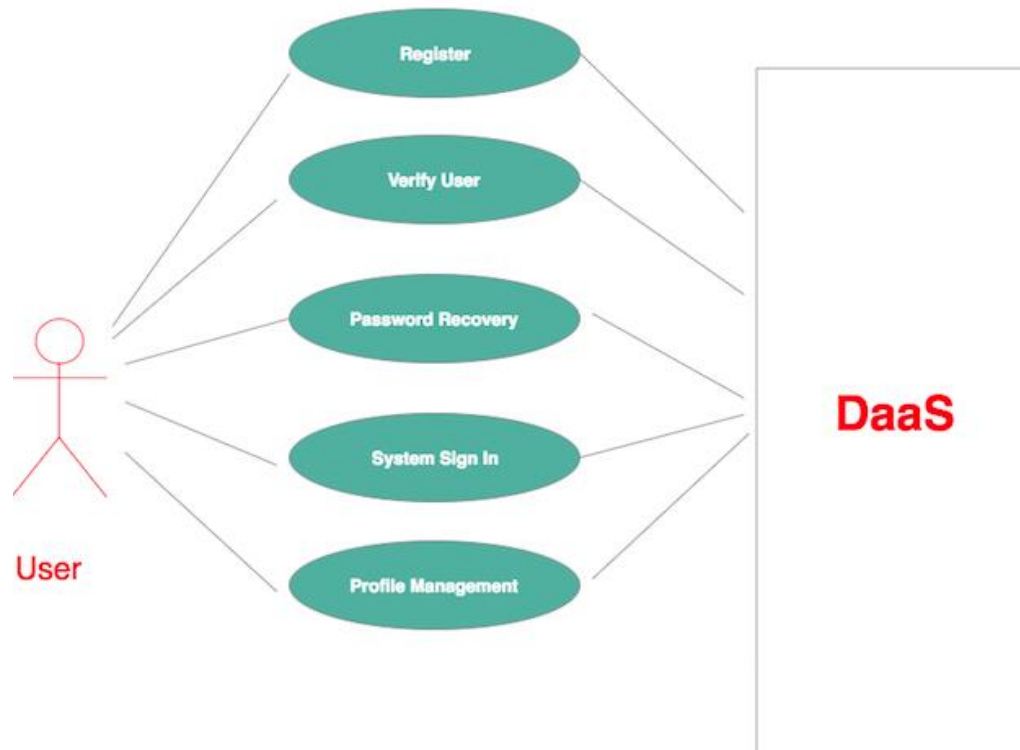
PostgreSQL- PostgreSQL is an openly available object-relational database system with traditional properties and standard SQL and also offers enhancements for the newly developed database system features. PostgreSQL can also be extensively used as it allows addition of new data types, functions and even operators. We are using PostgreSQL as it is one of the most flexible database systems which will be helpful in storing the various kind of information we would be collecting from our user.



## Chapter 4. Project Design

### Client Design

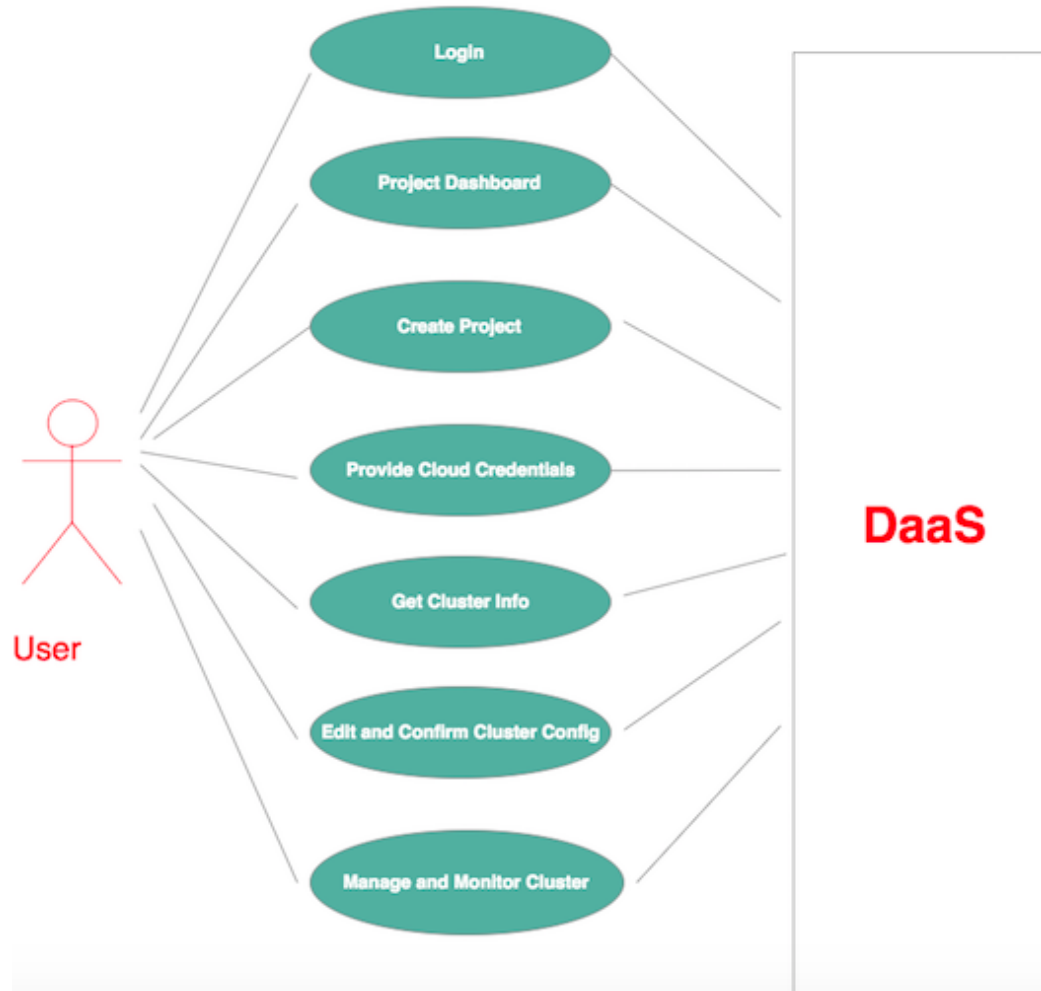
#### ✓ User Registration



*Figure 7. Client Design – User Registration*

The use case demonstrates the interaction between the system and the cloud service providers. As the initial step, cloud integration is done from system to the service provider. Once integration is done, infrastructure setup as needed by the user. Owing to the project files provided by the user, system does an environment set up on the cloud. Once both the setup steps are complete, the project (app) is deployed on the cloud service provider (s). As the last step system then gives the option of managing the deployment where the user can modify the needs if required.

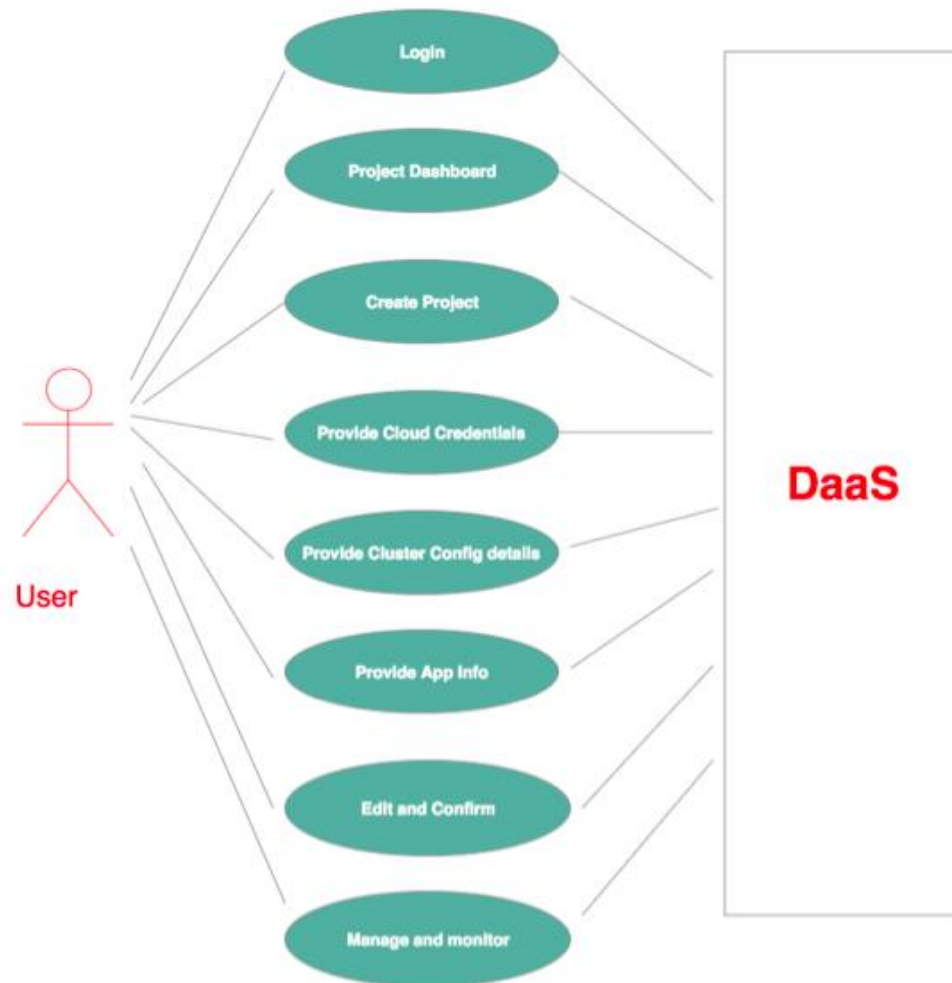
✓ Migration from existing Kubernetes Cluster



*Figure 8. Client Design – Migration from existing Kubernetes Cluster*

To migrate an existing Kubernetes cluster, the user has to login to the platform. On the project dashboard, the user will create a new project and provide their cloud credentials. Once DaaS has the cloud credentials, it will pull the Cluster info. DaaS will edit and confirm the cluster configuration and hence the migration will proceed. DaaS would be thereby used to manage and monitor the cluster.

✓ Migrating Dockers from Local Environment



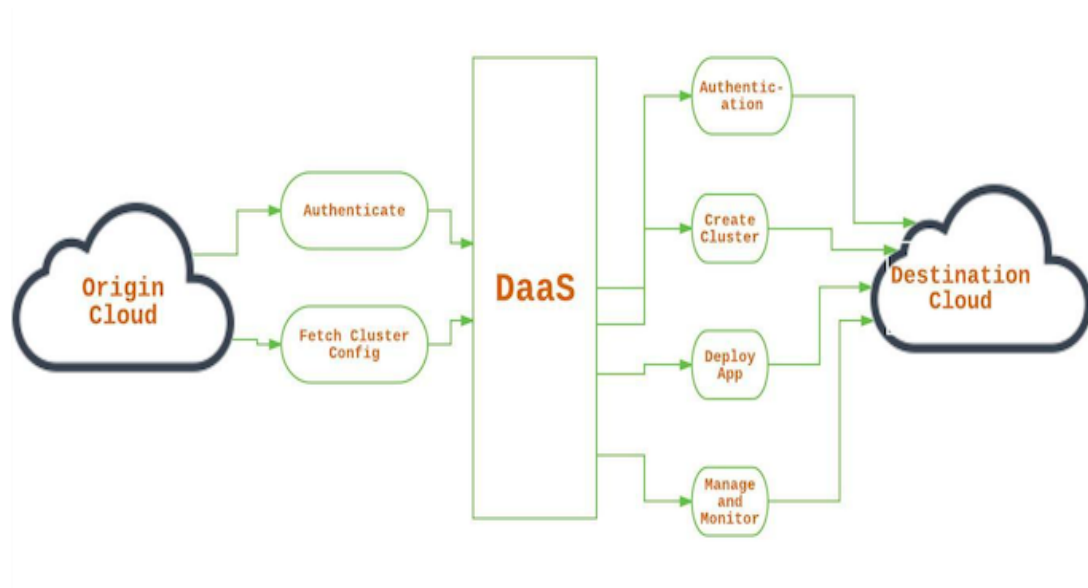
*Figure 9. Client Design – Migrating Dockers from Local Environment*

To migrate an existing Docker from the local environment the user has to login to the platform. On the project dashboard, the user will create a new project and provide their cloud credentials. The user would provide application information to DaaS and thus will edit and configure the docker container information. Once migration is done, DaaS would be thereby used to manage and monitor the cluster.



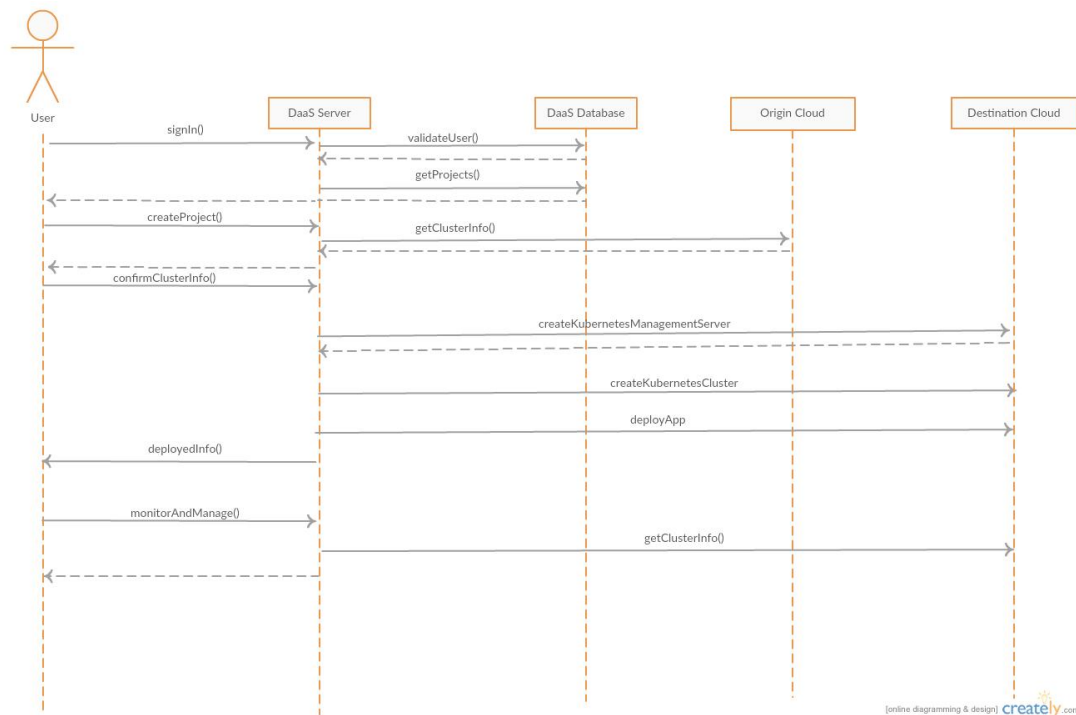
## Middle-Tier Design

- ✓ Migration from existing Kubernetes Cluster



*Figure 10. Middle-Tier Design – Migration from existing Kubernetes Cluster*

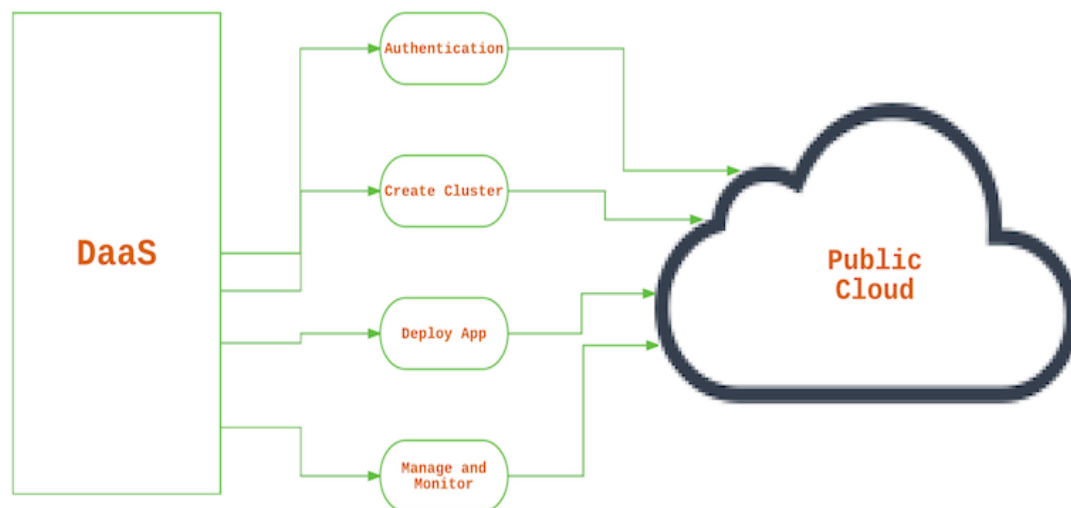
In the middle-tier level while migrating from existing Kubernetes cluster, the user needs to provide authentication and cloud credentials to DaaS. Once DaaS has the cluster information it would need the Destination Cloud credentials as well. After authentication on the destination cloud, DaaS creates a new cluster and deploys the app on the cloud. Post successful deployment, DaaS would be used to manage and monitor the Kubernetes cluster on the destination cloud.



**Figure 11. Sequence Diagram- Migration from existing cloud to another destination cloud**

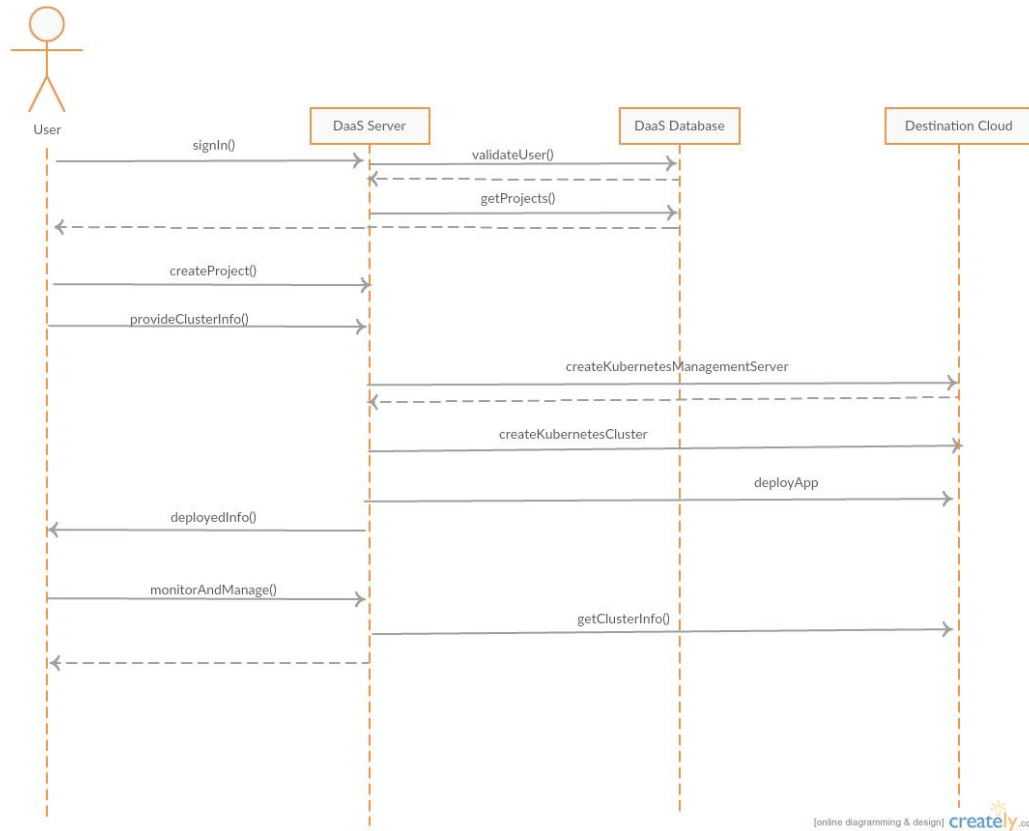
While migrating from an existing cloud to a different destination cloud, the user needs to sign in to the platform, and the DaaS server would validate the user. Once the user is validated, the user would create a project, and the DaaS server would get the cluster information from the origin cloud. After confirming the cluster details with the user, the DaaS server would create a Kubernetes Management server on the destination cloud. Post that the server would create a Kubernetes cluster, based on the information from origin cloud and hence deploy the app in the final step. Once the app is deployed, the server would send the information to the user, who can further go with monitoring and management of the application on the destination cloud.

✓ Migrating Dockers from Local Environment



*Figure 12. Middle-Tier Design – Migrating Dockers from Local Environment*

To migrate a cluster from a local environment to a cloud environment, DaaS would need cloud credentials to authenticate the user over the cloud. Post that, DaaS would create a cluster on the cloud and deploy the app on it. After successful deployment, DaaS would be used to manage and monitor the cluster.

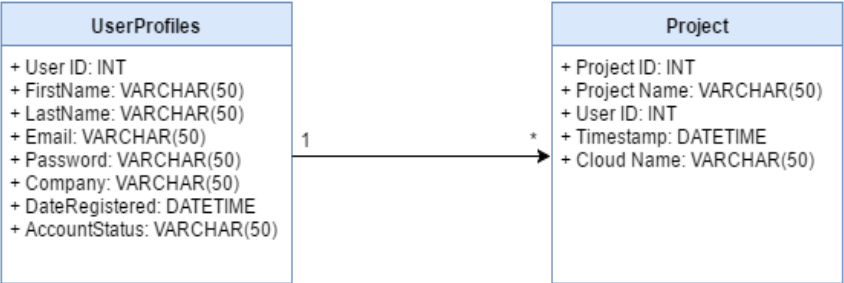


**Figure 13. Sequence Diagram- Migration from local cluster to the destination cloud**

While migrating to a destination cloud, the user needs to sign in to the platform, and the DaaS server would validate the user. Once the user is validated, the user would create a project, and provide the cluster information to the DaaS server. After which, the DaaS server would create a Kubernetes Management server on the destination cloud. Post that the server would create a Kubernetes cluster, based on the information provided by the user, and hence deploy the app in the final step. Once the app is deployed, the server would send the information to the user, for further monitoring and management of the application on the destination cloud.

**Data-Tier Design**

There are only 2 tables, User Profiles which contains information about the DaaS user and Project which contains information about the projects deployed. There's a one-to-many relation between user and project. Also, there will be non-structured data for cluster information which will be stored in Elasticsearch.

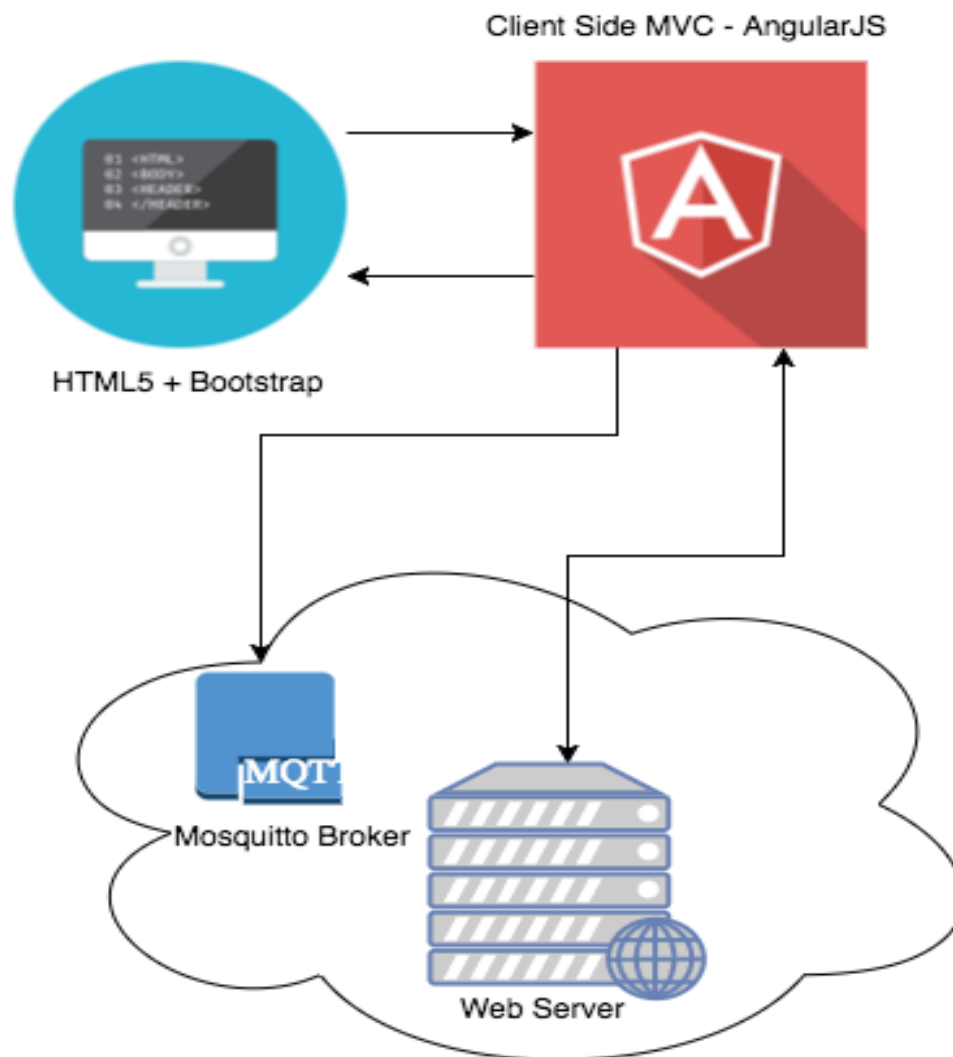


*Figure 14. Data-Tier Design*

## Chapter 5. Project Implementation

### -Client Implementation

The following architecture shows the client side implementation of the application.



*Figure 15 – Client Architecture Component Diagram*

The above figure represents the web client architecture. We have followed a clean design as per single responsibility principle and designed a separate HTML and JavaScript pages for each use cases. The various pages with their description are listed below:

### **I. Name: Sign Up Page**

**Description:** This page allows the user of the organization to sign up. Currently we support only a single user per application.

Sign up to DaaS

Enter your First Name

Enter your Last Name

Enter your Organization Name

Enter your Email Id

Enter your Password

By clicking on Sign up, you agree to [terms & conditions](#) and [privacy policy](#)

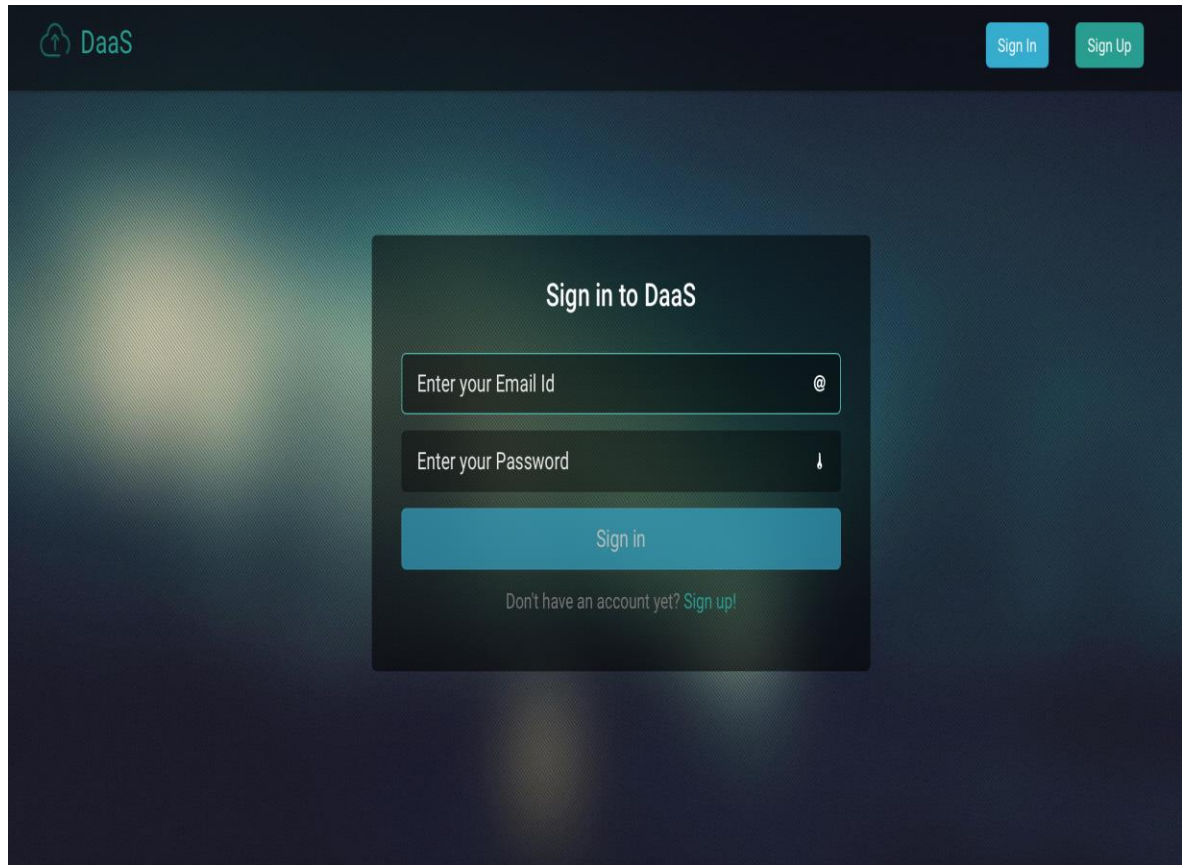
Sign up

Already have an account? [Log in now](#)

***Figure 16 – Sign Up Page***

## II. Name: Sign IN Page

**Description:** Allows the user to login to the application. A valid JWT token is generated and will be reused for subsequent request to access any page inside the application. The JWT token will be stored inside the cookies and the project details will be stored in the browser local storage.

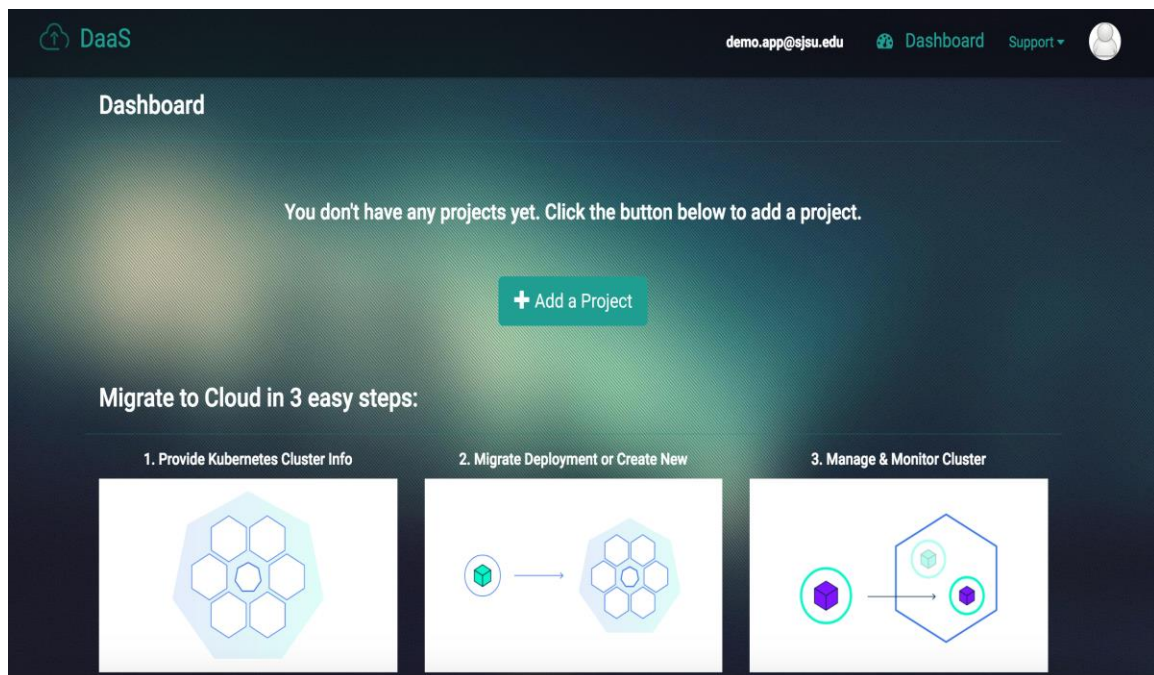


*Figure 17 – Sign IN Page*

### III(a) Name: Dashboard Page with no project

**Description:** The dashboard page displays all the projects which are created by the user. When there are no projects it shows a basic work flow of the application and Add project button. Access the add project button to create a new project. The project flow information is explained in the following sub points





*Figure 18 –Kubernetes Dashboard without projects*

### **III(b). Name: Dashboard Page with Projects**

**Description:** The dashboard page shows information about all the projects created by the user. The projects are listed in the descending order of their creation by default. There is a project search function as well where in user can search by project name. The project detail card contains details about the Cloud provider, Kubernetes Master, Kubernetes Node size and count and Volume attached to it. It also shows a graphical representation of the cluster in form of d3.js chart.

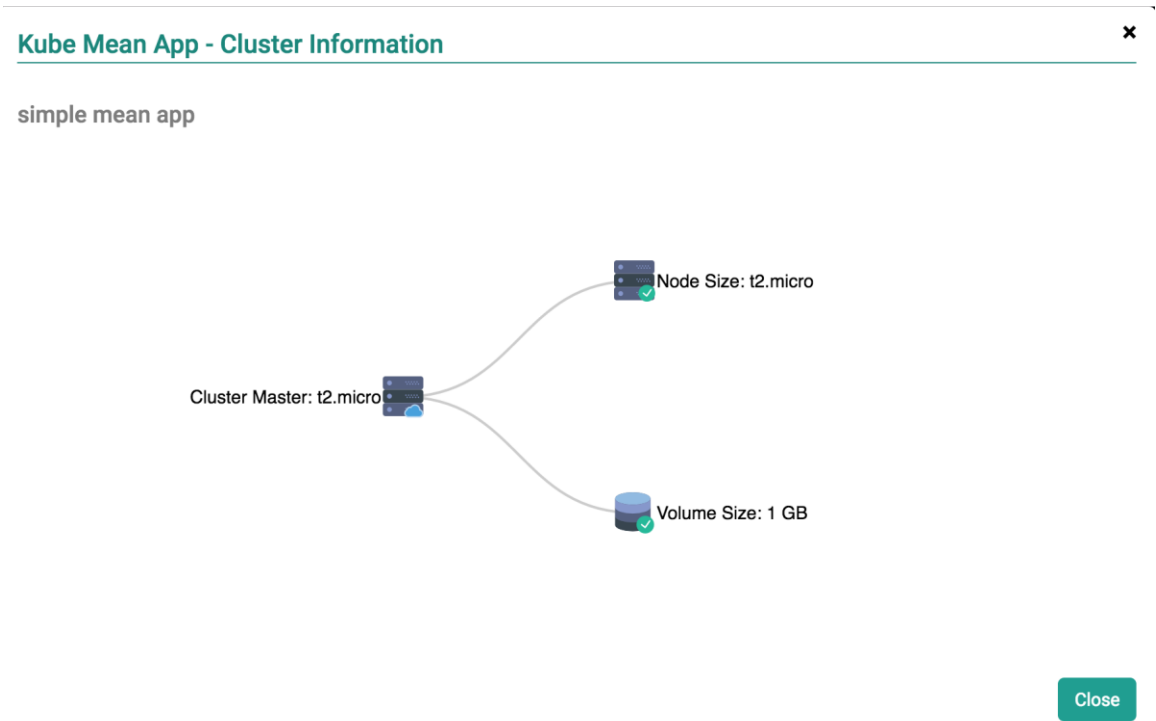


Figure 19 –Kubernetes Cluster Visualization-I

If the application is not deployed on the cluster, then a button to deploy the app from existing cluster is displayed on the project details card else the application URL is displayed. Also, on deleting the project whole Kubernetes cluster as well as the application are removed and the EC2 instances are freed up. We have also included various health check graphs of the cluster. We are showing the overall CPU and Memory utilization of the cluster.

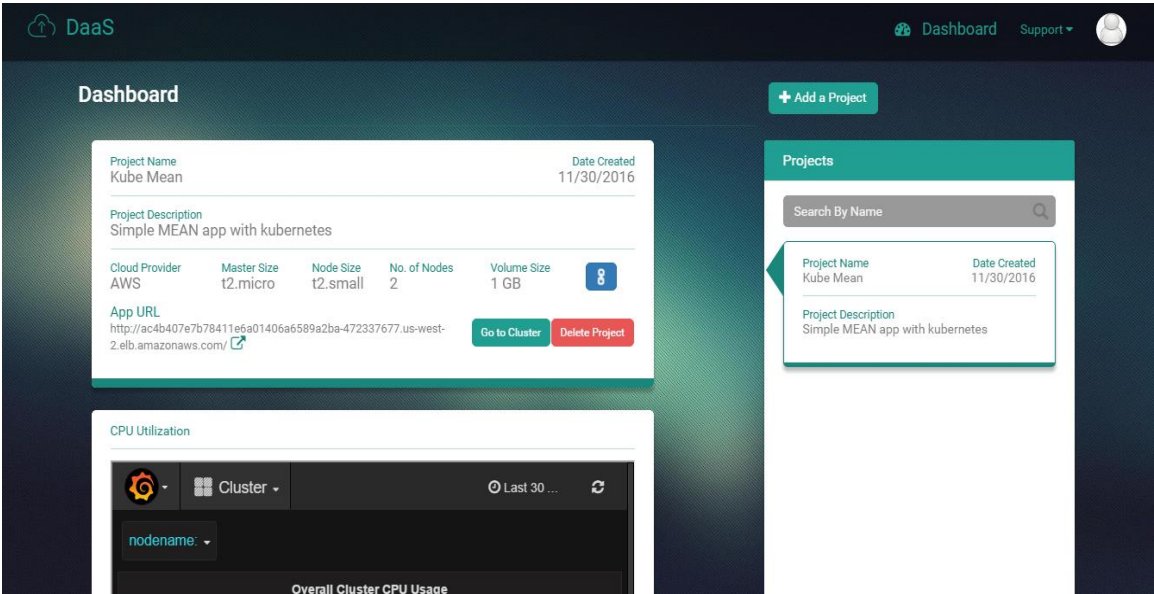


Figure 20 –Kubernetes Cluster Visualization-II

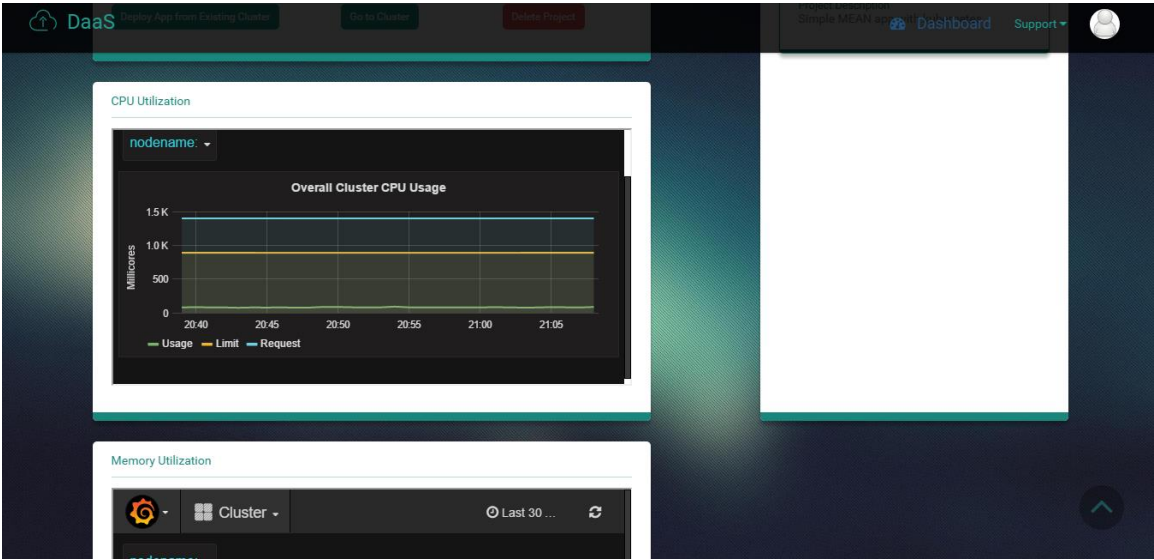
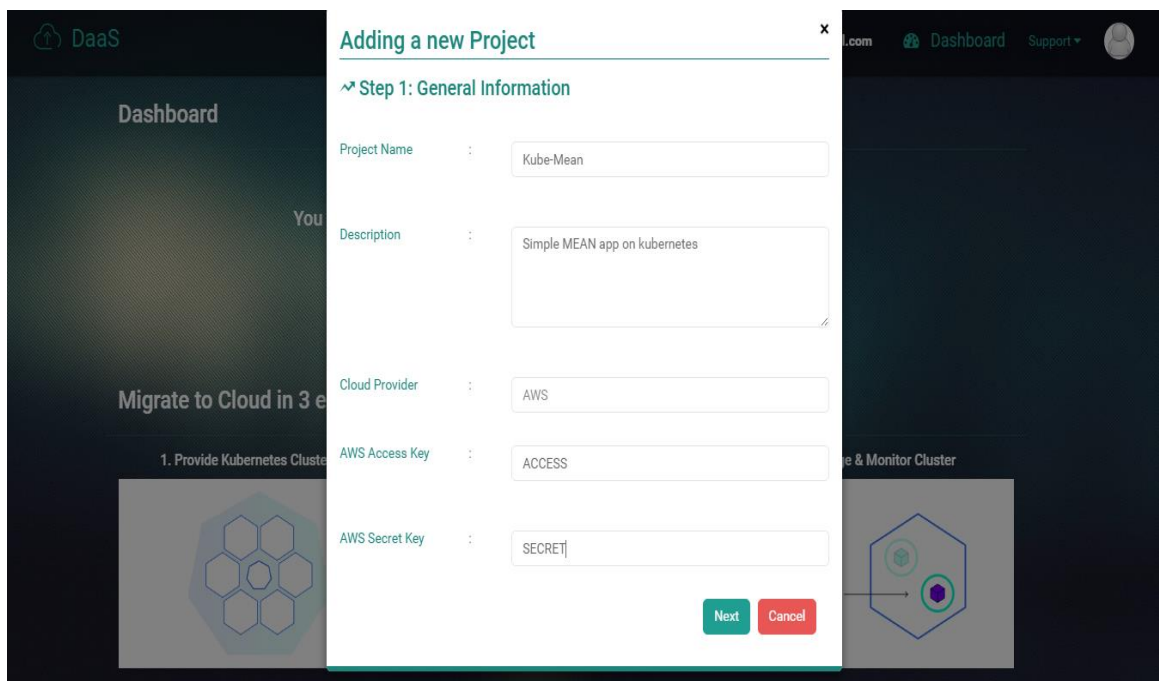


Figure 21 –Kubernetes Cluster Visualization-III

**IV(a). Name: Add a Project – General Project Information**

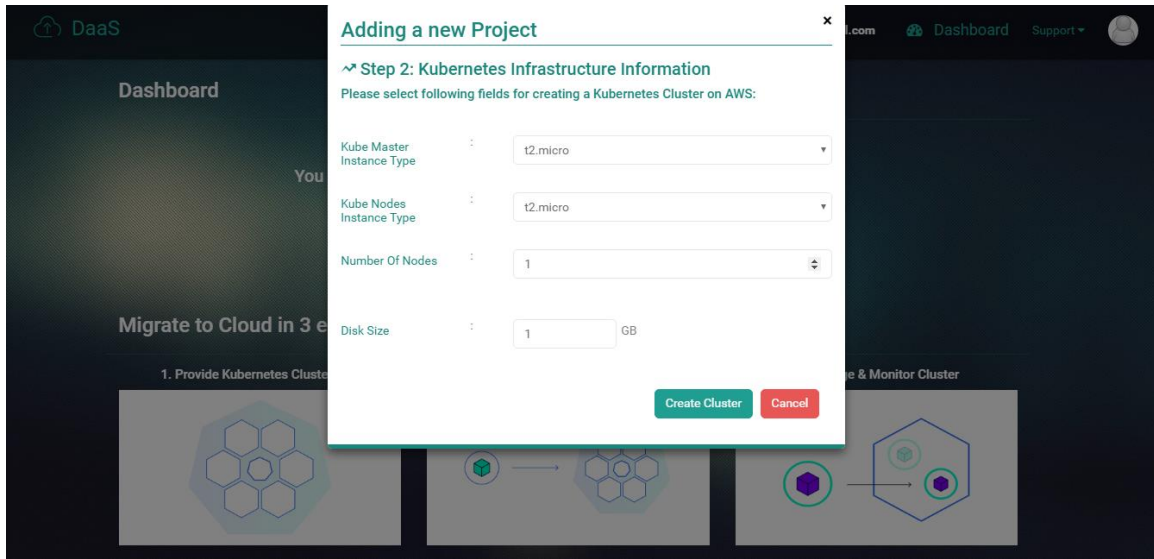
**Description:** The addition of the project is done through a series of modals which asks the user for the required information to create Kubernetes cluster. The first modal asks for the basic project information and destination cloud information. The Access Key provided by the user is stored and will be reused in case of any modification. If this is not the first project by user then, we will require the Private Key of the management server which was given to the user while creating the first project.

The image shows a web application interface with a dark theme. A modal window titled "Adding a new Project" is open, showing "Step 1: General Information". The modal has a white background and rounded corners. It contains several form fields: "Project Name" with the value "Kube-Mean", "Description" with the value "Simple MEAN app on kubernetes", "Cloud Provider" with the value "AWS", "AWS Access Key" with the value "ACCESS", and "AWS Secret Key" with the value "SECRET". At the bottom right of the modal are two buttons: "Next" (green) and "Cancel" (red). The background shows a dashboard with a "DaaS" logo, a "Dashboard" title, and a "Migrate to Cloud in 3 e" section. There is also a "1. Provide Kubernetes Cluste" section with a hexagonal icon.

*Figure 22 – Add Project – Project general Information – 1<sup>st</sup> Project*

**IV(b): Name: Add a Project – Kubernetes Infrastructure Information**

**Description:** On this modal the user provides the infrastructure information for the kubernetes cluster to be deployed. Since, we are using AWS as the Cloud provider here, we are asking for the server information in terms of EC2 instance size. All of master and nodes are deployed on different EC2 instances. Master size, Node size, Volume size and number of Nodes are the required information.



*Figure 23 – Add Project – Kubernetes Infrastructure Information*

**IV(c): Name:** Add a Project – Cluster Creation Status Information

**Description:** Once the cluster creation has started, we show the real time status about the component creation. We have used MQTT with Mosquitto broker to achieve this. The kubernetes cluster creation script was updated to publish the message to the broker and on this modal we subscribe on the topic. The cluster creation on an average takes about 40 mins. Once the cluster creation is done a Private Key for the management server will be returned to the user to be used for any maintenance activity by the user. We are not storing any keys on our backend for security reasons

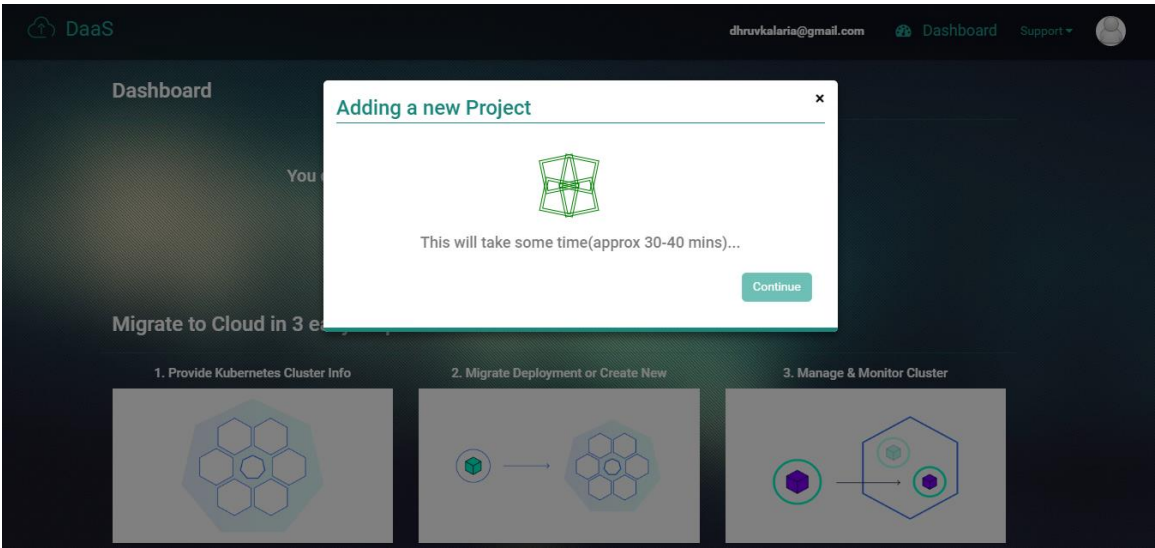


Figure 24 – Add Project – Cluster creation Status Information

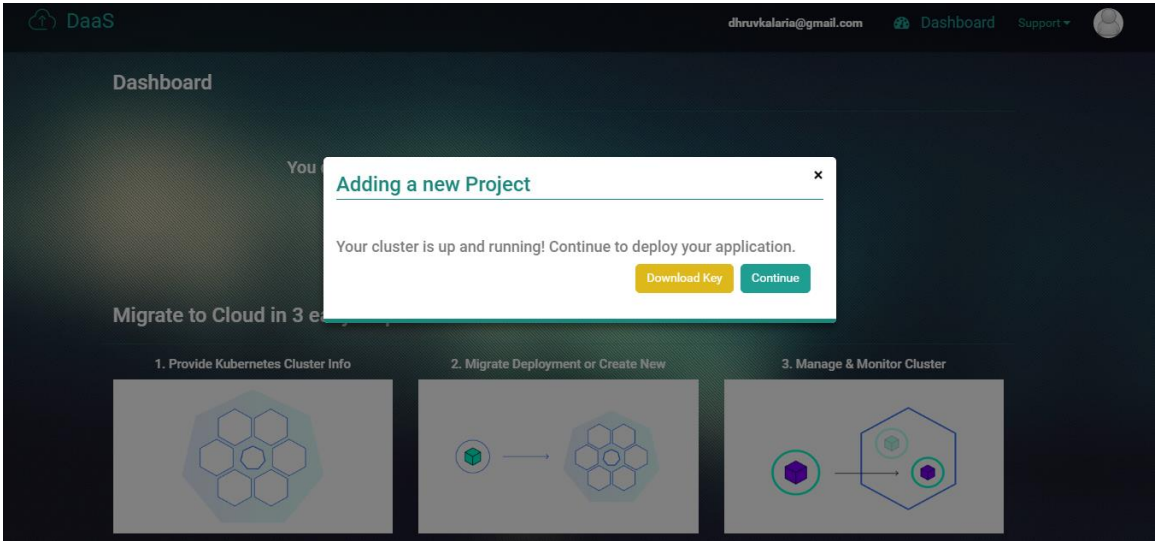
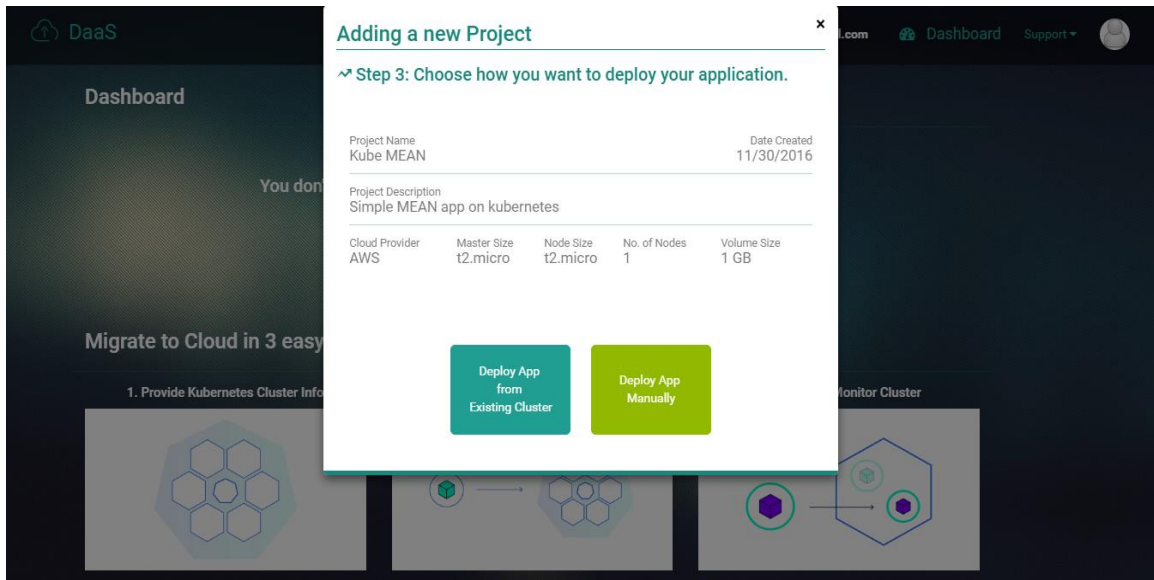


Figure 25 – Add Project – Download Private Key



**IV(d): Name: Add a Project – Deploy Application**

**Description:** Once the cluster creation is completed. The next step is to deploy the application on top of the kubernetes cluster. We have two options for the user – either to migrate from the existing kubernetes cluster or create an app from scratch.



*Figure 26 – Add Project – Deploy Application*

**IV(e): Name: Add a Project – Deploy from existing cluster**

**Description:** In order to migrate the application from existing cluster, we need to read the existing application information from kubernetes master URL with master access credentials. The backend will fetch the information with help of fabric8 APIs and will parse the actual deployments and services from default deployments and services

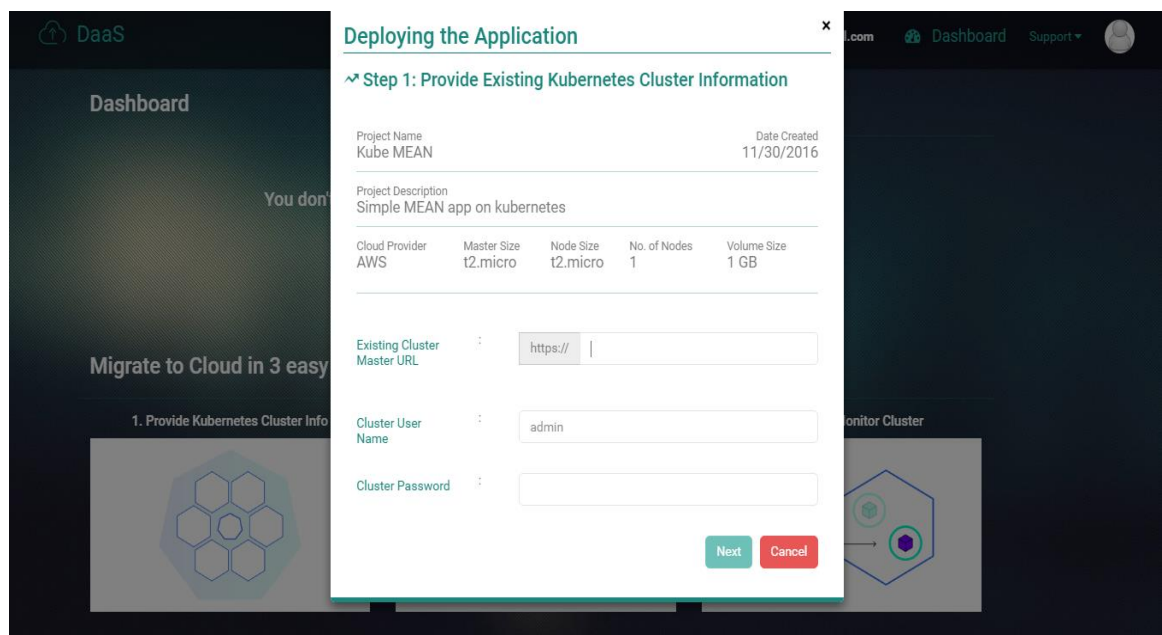


Figure 27 – Add Project – Deploy from existing cluster

**IV(f): Name: Add a Project – Confirm Deployments and Services**

Description: On this page, user can view existing services and deployments. Also, edit the deployment replication information and then confirm the deployment.

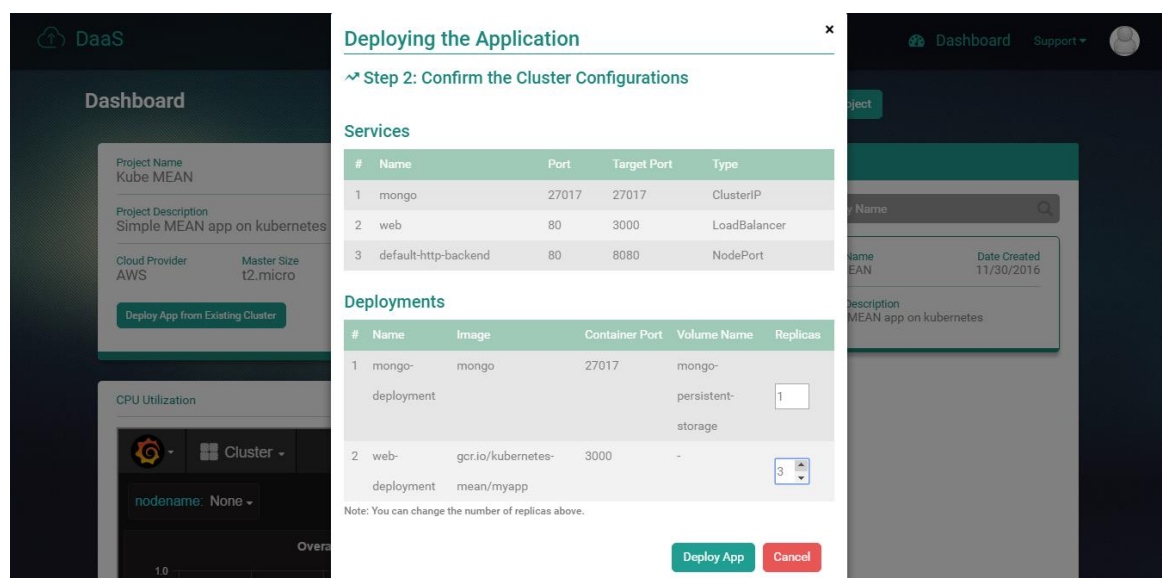
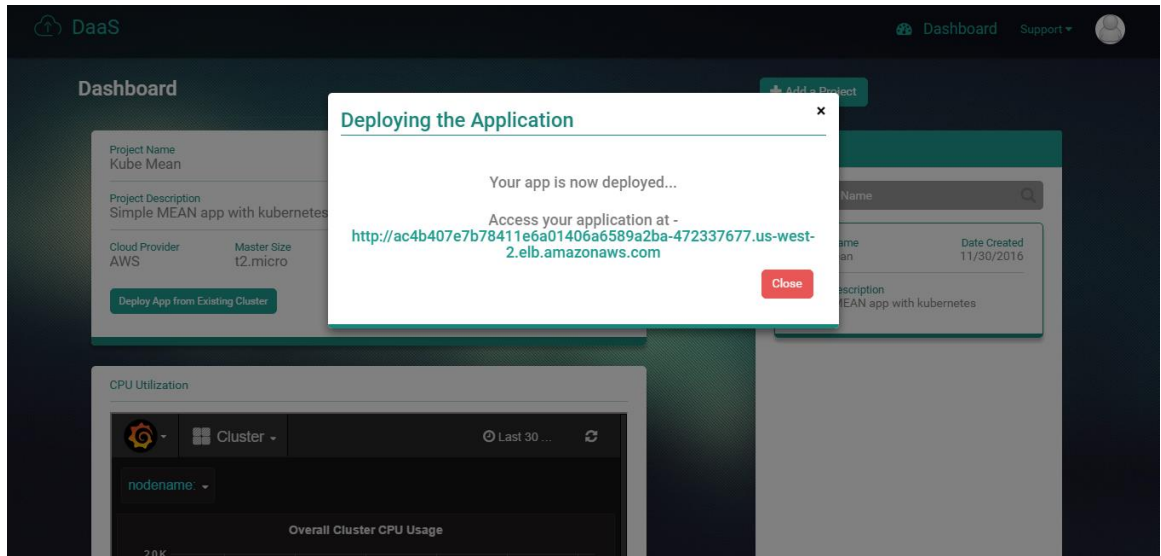


Figure 28 – Add Project – Confirm Deployment and Services



**IV(g): Name: Add a Project – Deploy app loading and app URL generation**

**Description:** Once the application is deployed successfully on the kubernetes cluster, the application URL is generated and shown to the user on this page. On closing this modal, the user is directed to the Kubernetes Dashboard page and can manage and monitor the cluster as well as deployments.



*Figure 29 – Add Project – Deploy app loading and app URL generation*

**V. Name: Kubernetes Cluster UI Page**

**Description:** This page lists all the projects created by the user and on clicking any individual project, the kubernetes UI for that project is dynamically loaded inside the iframe. The authentication on the UI is handled by injecting appropriate credentials automatically by angular.

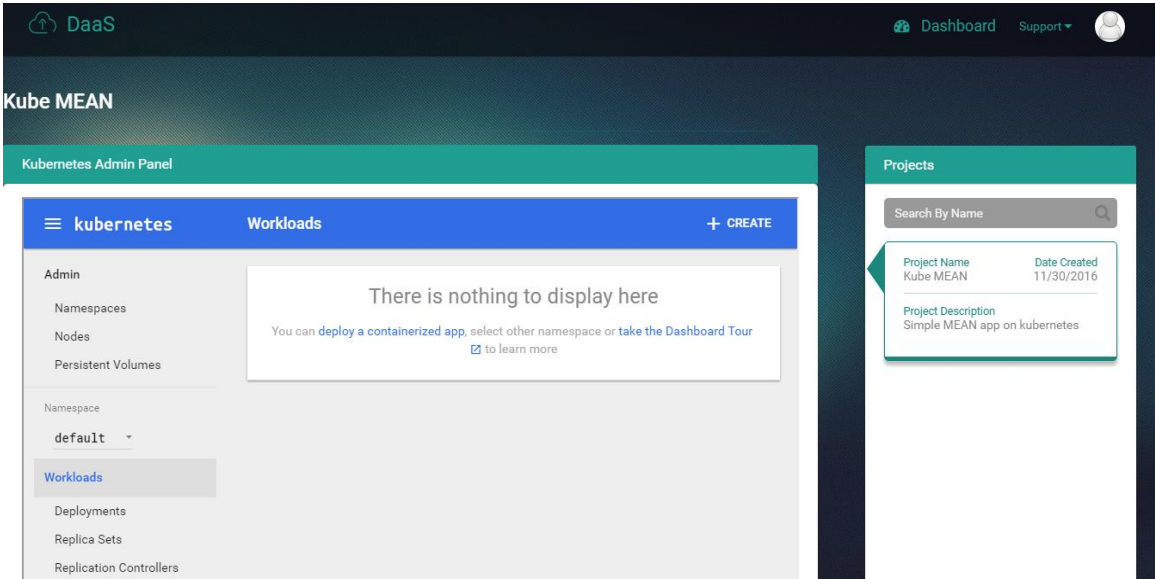
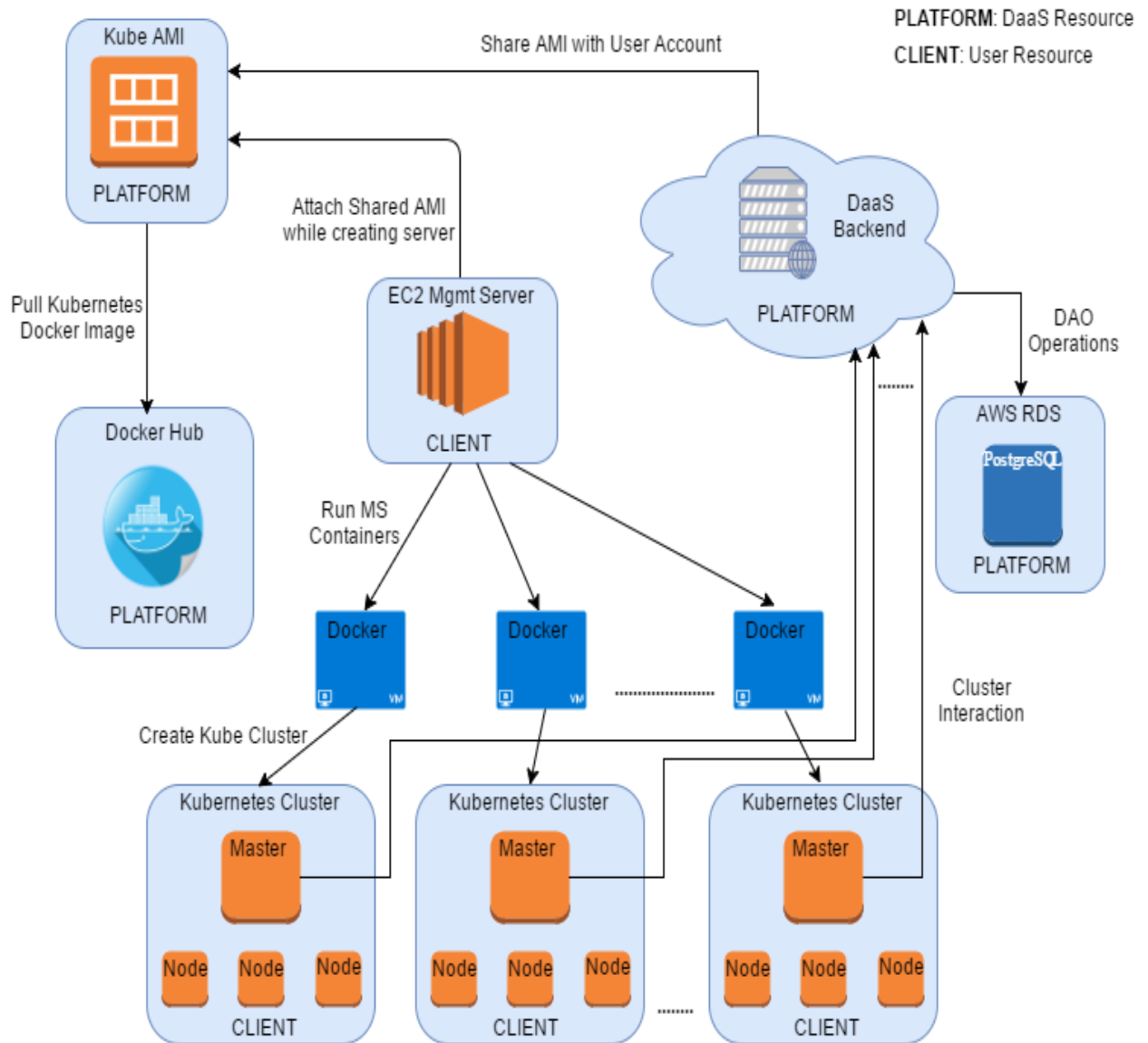


Figure 30 – Kubernetes Page

- ✓ Architecture Component Diagram



The above figure describes the broad brush design of our platform. It is a shared responsibility model, where the infrastructure responsibilities lies with the platform provider i.e. DaaS and the cloud provider i.e. AWS. The responsibility of deploying the application and scaling it as per needs rests with the enterprise who deploys the

application on top of our service. We provide a platform to create and manage the kubernetes cluster on client controlled public cloud infrastructure.

In the above figure, we have created two tags to show different cloud environments. The tag with 'Platform' means it is our cloud environment whereas the tag 'Client' means it is the cloud environment of the enterprise. Hence, all the costs related to deploying and running the backend infrastructure will still lie on the enterprise account. Our backend infrastructure is deployed on AWS and uses Elastic Bean Stalk and RDS services of the cloud.

To create multiple running kubernetes clusters for any enterprise, we create a EC2 management server. Inside management server we host the kubernetes installation inside a docker image pre-pulled from docker hub. For every new project a new container will initialize and create a cluster. We inject the user requirements for the cluster inside the kubernetes from a bash script.

While building platform we have taken the security to be number one priority. Since the users trusts us to provide the cloud credentials to us, we never store the credentials. The backend service keeps the credentials in memory and once the project is created the credentials are garbage collected by JVM. Hence, whenever the user needs to update anything related to his infrastructure, the AWS client ID and AWS Secret Key needs to be supplied again.

In order to reduce the cluster creation time and environment set up we have done some optimizations in the process to create the environment.

1. Shared AMI:

None of the existing AMIs which are available as public AMI's which can be used has docker support.

2. Pre-pulled Docker Image from Docker Hub:

Instead of pulling the docker image from docker hub every time we create a new project, we have pre-installed it inside the AMI. In case of any update the image will fetch itself from the docker hub.

3. Managing multiple clusters from Single EC2 instance:

Instead of creating a per project EC2 instance to manage the kubernetes cluster, we used docker to manage multiple kubernetes on same host by injecting the kubernetes specific environment variables runtime. This approach reduces the cost of operation for the user who has multiple kubernetes clusters running in his environment.

✓ USE CASE - 1: Migrating Docker from Local Environment

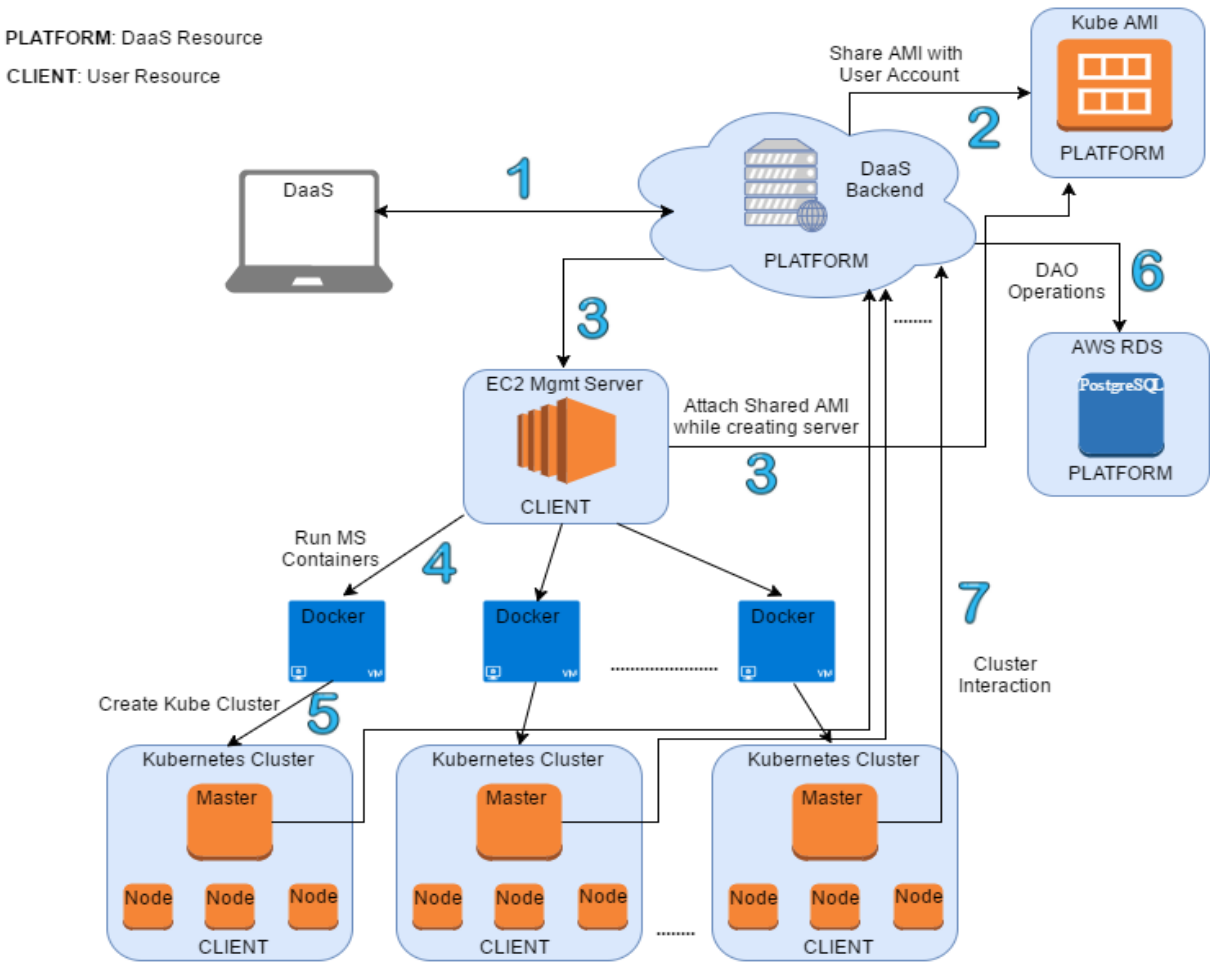


Figure 32: Migrating Docker from Local Environment

The above figure represents the data and control flow in order to create kubernetes cluster for a user in AWS with user defined configurations. In this use case, the user could create

a deployment in cloud from the scratch without any pre-requisites. The serial number in the following description maps to the number described in the above figure.

1. Passing in project creation details:

User will log in to the system and performs 'Create Project' action from Web Application. User will pass AWS CLIENT ID, AWS SECRET KEY, AWS ACCOUNT ID, Kubernetes Cluster Information (Size of Master, Size of Minions, Number of Minions, Cloud Provider) to create a project

2. Sharing the global AWS EC2 Image:

If the user is creating first project from the account, then in order to spin up a management server a global AWS AMI is required.

3. Creating Management Server:

Backend service will share the AMI with the user account and spin up a management server which manages the kubernetes clusters.

4. Instantiating Docker Container:

Once the management server is created, a docker container with pre-built kubernetes version is instantiated. While running the docker, all the user-defined kubernetes cluster configurations are injected into the docker to create the cluster

5. Spinning up Kubernetes Cluster:

Docker containers will spin up a kubernetes cluster on the provided cloud environment. Once the cluster is up and running, the master information and the credentials to access the kubernetes master will be passed back to the user

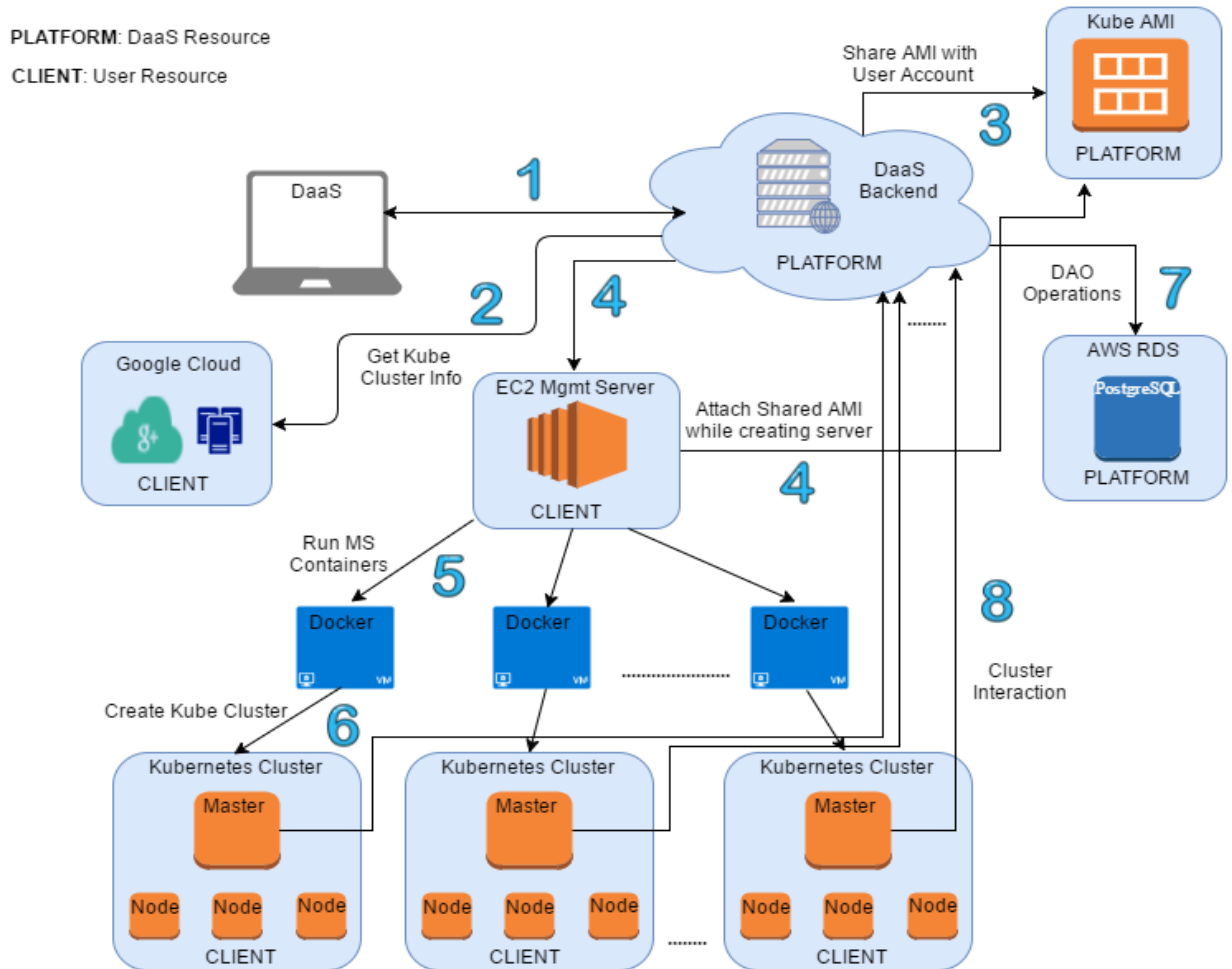
6. Storing the project info:

Once the cluster is up and running, the metadata information regarding the cluster will be stored in Amazon RDS database of the platform.

7. Managing the deployments:

After cluster management is completed, the user now will upload the yaml file from his machine with information about services and deployments to the master. The master will then proceed as per the configuration and deploy the application.

✓ USE CASE - 2: Migrating from existing Kubernetes Cluster



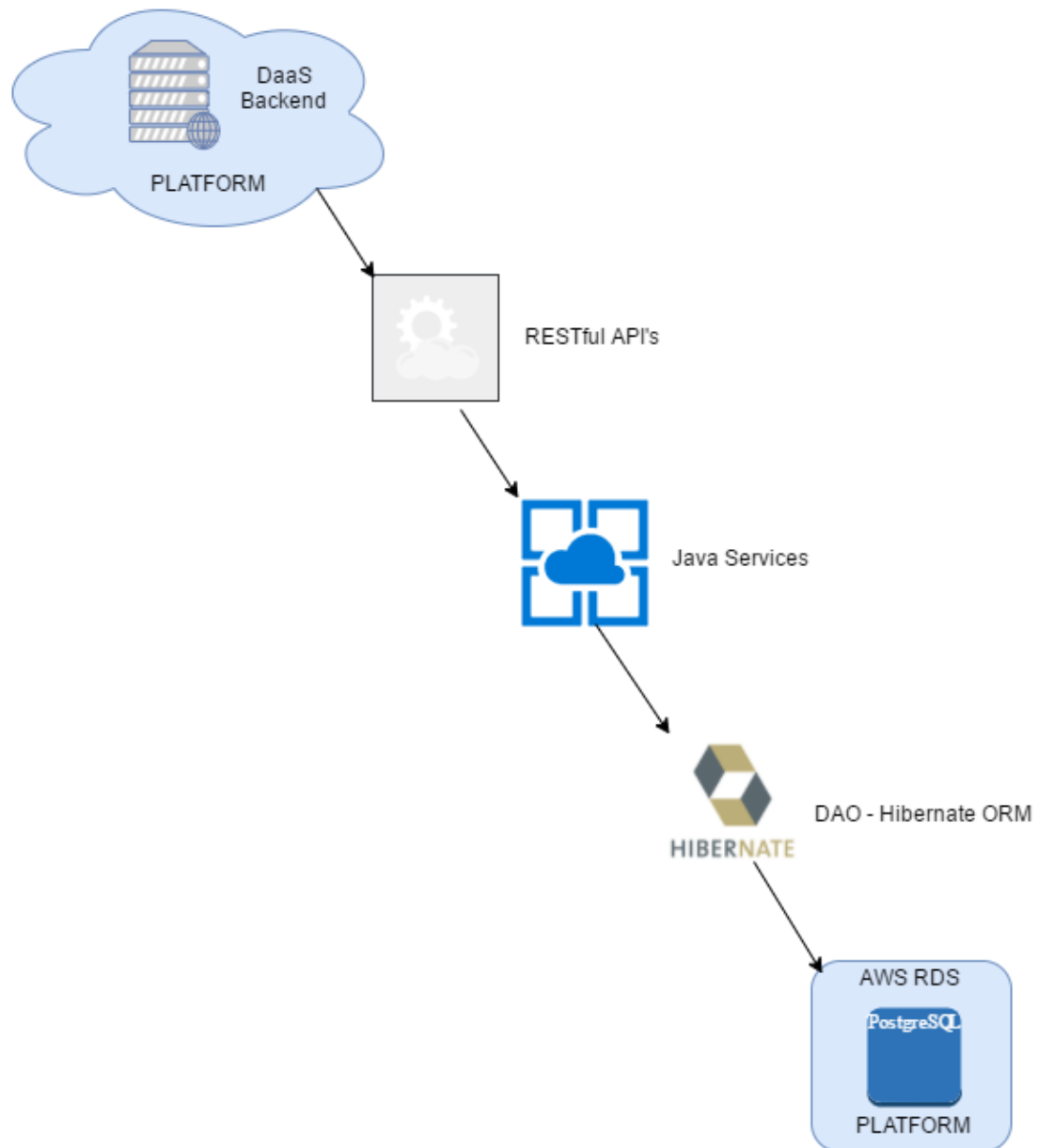
*Figure 33: Migrating from Existing Kubernetes Cluster*

The above figure represents the data and control flow in order to create kubernetes cluster for a user in AWS with user defined configurations. In this use case, the user reads the existing kubernetes configuration from the cloud and replicate into the destination cloud provider. The flow is same as described in the previous section only the change is in flow – 2 which is described below:

- Reading Existing Kubernetes Configuration:

User enters the URI of the existing Kubernetes Master and the credentials in order to access the information. The backend reads the information and asks the user to confirm the details.

### Data-Tier Implementation



*Figure 34: Data Tier Access Implementation*



The above figure represents the access implementation of the database by the backend service. The RESTful API will call the java controller which will in turn call the Hibernate DAO. Hibernate is configured to access the Postgres data store. The database is hosted on AWS RDS service. Hibernate connects to RDS via JDBC protocol and uses a connection pool to access the database services.

### **System Implementation Summary**

Construction Phase – The construction phase of our system, DaaS involves two things: building and testing the functional system which fulfills the design requirements. There are certain activities involved with the system implementation with a focused purpose, prerequisite's and deliverables. Below is the summary of these activities -

#### **Activity: Building the Software Package**

The software package for Deployment as a Service has certain prerequisite's and planned deliverables. To achieve the design requirements, the software is built with certain security measures keeping the user's secure information in mind. Below is the highlight of the important measures taken across building the software –

- ✓ All critical information like user's credentials for our system are hashed and stored.
- ✓ The backend uses JWT implementation to authenticate and authorize the functionality of the system.
- ✓ The tokens are stored using cookies and have an expiration time attached, asking the user to authenticate again upon expiration.
- ✓ No cloud credentials of the users are ever stored in our database and are just used in-memory. Thus, every time a new project is to be added, we ask for the credentials again.
- ✓ For the very first DaaS project, the system creates a Management Server and attached a key pair to it, which is provided to user via UI and can be downloaded.

The key pair is also not stored by us and upon creating next projects, we ask for the key pair to be uploaded via UI to assist the functionality of the system.

- ✓ While providing information about existing kubernetes cluster, the credentials for the cluster are also never stored and are used in-memory.

#### Activity: Write and Test the Software

The system is built keeping in mind to keep the software components independent and functional so that it's possible to reuse these software components. Also, certain components are developed using the test-driven development approach making the software more reliable and easy to build. Of course, there are certain integration requirements with this approach and it requires good integration testing of our system. The system components do have certain unit tests and integration tests to make sure it works as intended. Further, it also pushes towards system testing ensuring that the isolated components work properly when integrated into a whole system.

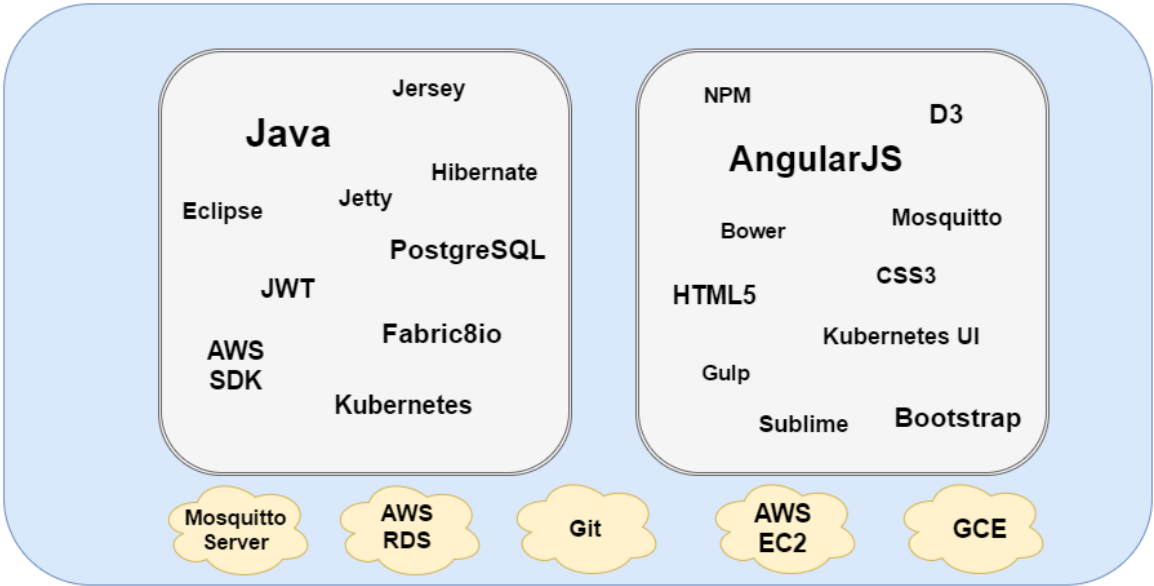
#### Activity: Build and Test Database

This starts with the database design requirements. The database is build using the system design specifications. The system, DaaS uses a relational database with User's and Projects table. The relationship between them is of course one-to-many. The database schema is reviewed keeping the requirements in mind. No secure information is ever stored directly to the database. Also, the database tables are tested with sample data to ensure the relationship to add, delete, modify, read records while maintain the data integrity checks.

Delivery Phase – This requires delivering the system into operation. Before delivering the system, we ensured that the software components of the system works as a whole. The database is installed and the connections are verified. Another important consideration is that the system is tested once deployed. The basic functionalities of the system work well and can be accessed by the end-users. The purpose of this phase is to make sure the end-users have a smooth experience while accessing the system.

**Tools and Technologies Used**

Below is the summarized figure showing the tools and technologies used -



*Figure 35: Tools and Technologies Summarized*

## **Chapter 6. Performance and Benchmarks**

Benchmarking helps in performance improvement of a software product and its development. We need benchmarks to monitor the continuous scalability of the platform and the throughput over the performance

Below are the benchmarks we intend to achieve in the project-

- Achieving all the aspects defined during the early stages
- Meeting high productivity during development
- Delivering on time.
- Software free of defects
- Implementing all the requirements in minimal budget.
- Adapt the change requests needed efficiently

## Chapter 7. Deployment, Operations, Maintenance

The backend application and frontend applications are deployed individually as different servers. The application servers are hosted on AWS EC2 infrastructure.

Frontend Deployment:

- Gulp server is used to deploy frontend application

Steps to deploy:

- **Do a Git clone from the repo** – “<https://github.com/arushigangrade/deployment-as-a-service-web>” **on origin/master branch**
- Run ‘npm install’ to resolve all the dependencies
- Run ‘gulp serve’ to start the application

Backend Deployment:

- Jetty Servlet Container server is used to host the backend application server
- For Database we are using AWS RDS database as a service

Steps to deploy:

- Do a Git clone from the repo – “<https://github.com/DhruvKalaria/deployment-as-a-service>” on origin/master branch
- Run ‘mvn package’ to create a war file
- Run the application as “Java jetty\_runner.jar app.war”

Operations and Maintenance:

- Any new patches to the software can be pulled from the existing repo and the above mentioned steps could be repeated to re deploy the application. In future, a CI/CD pipeline can be developed for automated deployments

## Chapter 8. Testing and Verification

### Test Environment Set Up

Following steps are needed to set the environment for test infrastructure:

- Deploy the app on cloud
- The RDS database should be up and running on cloud
- Server should be running for UI testing
- Latest browser versions should be installed in test the environment

### Testing Plan and Test Design

#### Testing Approach

Below are the types of testing covered as part of the testing phase of our product lifecycle:

- **Unit Testing:** Testing of each module for its UI, Database connectivity, and checking the API calls to the cloud provider.
- **Functional Testing:** This covers testing of all functional requirements prior to system integration.
- **Scenario Based Testing:** Comprehensive testing of different use-cases of the platform.
- **Non- Functional Testing:** To test security, performance, and scalability of the system.

#### Test Scope

Test Objectives	Test Scope
UI / UX	To test user login, dashboard, create project, cluster information, destination cloud, and monitoring project. Testing needs to be done using different browsers to check compatibility.
Unit Test	To test every module in the server side code
Component Testing	To test following components - Project Creation, Infrastructure Setup, Environment Setup, Automated Deployment, and Monitoring as individual applications.

System Integration Testing	Test Cases to check if all components are working properly together.
Performance Testing	Test Cases to test performance while infrastructure setup.
Security Testing	Implement test cases to test security aspects of e-vault storage.

## Test Design

### Unit Tests:

<b>Test Case ID</b>	DaaS01
<b>Description</b>	Unit Test for Security Group Creation
<b>Pre-condition</b>	Working AWS Account
<b>Input</b>	AWS credentials, EC2 config properties – ami ID, instance Type, security group name, keypair name, iamName
<b>Expected Output</b>	EC2 instance is created with provided security group

<b>Test Case ID</b>	DaaS02
<b>Description</b>	Share AMI with user account
<b>Pre-condition</b>	User account is created
<b>Input</b>	amID and user account ID
<b>Expected Output</b>	amID is shared with the user account

<b>Test Case ID</b>	DaaS03
<b>Description</b>	Check if IAM role exists
<b>Pre-condition</b>	AWS Account
<b>Input</b>	AWS Credentials

<b>Expected Output</b>	The IAM role which is not created should not exist
------------------------	--

<b>Test Case ID</b>	DaaS04
<b>Description</b>	Create IAM Role with Administrator Access
<b>Pre-condition</b>	AWS Account
<b>Input</b>	AWS Credentials and role name
<b>Expected Output</b>	IAM role with administrative rights should be created with the provided role name

<b>Test Case ID</b>	DaaS05
<b>Description</b>	Get AWS Account ID for User
<b>Pre-condition</b>	AWS Account for User
<b>Input</b>	AWS Credentials
<b>Expected Output</b>	User's Account ID

<b>Test Case ID</b>	DaaS06
<b>Description</b>	Create a DaaS User
<b>Pre-condition</b>	RDS database should be up
<b>Input</b>	User details for account creation
<b>Expected Output</b>	amID is shared with the user account



<b>Test Case ID</b>	DaaS07
<b>Description</b>	Create a new Kubernetes Deployment using YAML file
<b>Pre-condition</b>	Kubernetes client
<b>Input</b>	Kubernetes client, YAML file
<b>Expected Output</b>	Kubernetes deployment is created for master URL

<b>Test Case ID</b>	DaaS08
<b>Description</b>	Create Kubernetes Service
<b>Pre-condition</b>	Kubernetes client
<b>Input</b>	YAML file, Kubernetes client
<b>Expected Output</b>	Kubernetes service is created for master URL

<b>Test Case ID</b>	DaaS09
<b>Description</b>	Start EC2 instance
<b>Pre-condition</b>	AWS Account with an instance created
<b>Input</b>	Instance ID
<b>Expected Output</b>	Start Instance Request call is successfully made to aws and instance is started

<b>Test Case ID</b>	DaaS10
<b>Description</b>	Stop EC2 instance
<b>Pre-condition</b>	AWS Account with an instance created
<b>Input</b>	Instance ID

<b>Expected Output</b>	Stop Instance Request call is successfully made to aws and instance is stopped
------------------------	--

<b>Test Case ID</b>	DaaS11
<b>Description</b>	Fetch public IP of cluster
<b>Pre-condition</b>	EC2 Instance is created
<b>Input</b>	Instance ID
<b>Expected Output</b>	IP is retrieved using DescribeInstancesRequest call to AWS

<b>Test Case ID</b>	DaaS12
<b>Description</b>	Create a secret signing key
<b>Pre-condition</b>	-
<b>Input</b>	Encryption algorithm name
<b>Expected Output</b>	Secret signing key is generated

<b>Test Case ID</b>	DaaS13
<b>Description</b>	Hash password
<b>Pre-condition</b>	JBcrypt is added as maven dependency
<b>Input</b>	Plain text password
<b>Expected Output</b>	Password is hashed

**Functional Testing**

<b>Test Case ID</b>	DaaS14
<b>Description</b>	User Sign Up
<b>Pre-condition</b>	Test Environment is working
<b>Input</b>	Object of User Type with user_id, firstname, lastname , email, password, organization
<b>Expected Output</b>	User is created with a temporary ManagementEC2InstanceId

<b>Test Case ID</b>	DaaS15
<b>Description</b>	User Login with valid credentials
<b>Pre-condition</b>	Test Environment is working
<b>Input</b>	Object of User Type
<b>Expected Output</b>	User can login successfully by fetching JWT and cookie

<b>Test Case ID</b>	DaaS16
<b>Description</b>	User Login with invalid credentials
<b>Pre-condition</b>	Test Environment is working
<b>Input</b>	Object of User Type
<b>Expected Output</b>	User service should not validate user and login should not be successful

<b>Test Case ID</b>	DaaS17
<b>Description</b>	Fetch all projects for user

<b>Pre-condition</b>	User is created in Test Environment
<b>Input</b>	User ID, Object of Cookie type
<b>Expected Output</b>	Token is generated using JWT and User Service is able to get List of all projects by fetching User from user id.

<b>Test Case ID</b>	DaaS18
<b>Description</b>	User Login with valid credentials
<b>Pre-condition</b>	Test Environment is working
<b>Input</b>	Object of User Type
<b>Expected Output</b>	User can login successfully by fetching JWT and cookie

<b>Test Case ID</b>	DaaS19
<b>Description</b>	Delete User
<b>Pre-condition</b>	Test Environment is working
<b>Input</b>	Object of User Type
<b>Expected Output</b>	User is deleted after fetching valid token and all the projects associated with the user are also deleted

<b>Test Case ID</b>	DaaS20
<b>Description</b>	Check Kubernetes Connection for existing cluster
<b>Pre-condition</b>	Existing cluster information
<b>Input</b>	URI, masterusername, masterpassword
<b>Expected Output</b>	Kubernetes connection is created and details related to services and deployments for the client are retrieved

<b>Test Case ID</b>	DaaS21
<b>Description</b>	Create First Project
<b>Pre-condition</b>	User ID, AWS credentials
<b>Input</b>	Project id, node numbers, User ID, organization name
<b>Expected Output</b>	EC2 instance is created and kubernetes cluster is created on that instance.

<b>Test Case ID</b>	DaaS22
<b>Description</b>	Add Project
<b>Pre-condition</b>	User ID, AWS credentials
<b>Input</b>	Project id, node numbers, User ID, organization name, Management EC2 Instance ID
<b>Expected Output</b>	EC2 instance is created and kubernetes cluster is created on that instance.

<b>Test Case ID</b>	DaaS23
<b>Description</b>	Functional testing for mandatory fields
<b>Pre-condition</b>	The web platform is up and running.
<b>Input</b>	While registering for an account, on the signup page, one of the mandatory fields is left blank.
<b>Expected Output</b>	Error Message should be displayed, as the required field is blank.

<b>Test Case ID</b>	DaaS24
---------------------	--------

<b>Description</b>	LogOut validation
<b>Pre-condition</b>	The user has logged in to the platform
<b>Input</b>	User clicks on the Logout under the profile section
<b>Expected Output</b>	User is successfully logged out of the platform

Scenario based Testing:

<b>Test Case ID</b>	DaaS25
<b>Description</b>	Validating the error message
<b>Pre-condition</b>	The web platform is up and running.
<b>Input</b>	While signing in, provide wrong password associated with an account
<b>Expected Output</b>	Error message with “Invalid Credentials” should be displayed

<b>Test Case ID</b>	DaaS26
<b>Description</b>	Field validation
<b>Pre-condition</b>	The web platform is up and running
<b>Input</b>	Invalid text on the email field, of Sign In/Sign Up page
<b>Expected Output</b>	Error message with “Invalid email address” should be displayed

<b>Test Case ID</b>	DaaS27
<b>Description</b>	Links Verification
<b>Pre-condition</b>	The user has logged in to the platform

<b>Input</b>	All the hyperlinks and buttons are clicked one by one
<b>Expected Output</b>	Correct links and tabs are open respective to clicks made.

<b>Test Case ID</b>	DaaS28
<b>Description</b>	Validating the dashboard components
<b>Pre-condition</b>	The user has deployed few projects on the platform
<b>Input</b>	From the project tab, one of the deployed projects is clicked
<b>Expected Output</b>	On the dashboard page, all the correct and relevant information with the selected project should be displayed.

<b>Test Case ID</b>	DaaS29
<b>Description</b>	Create Project
<b>Pre-condition</b>	The user has logged in to the platform
<b>Input</b>	User clicks on “Add a Project” button.
<b>Expected Output</b>	The modal for adding a new project opens, with fields for user to provide input.

<b>Test Case ID</b>	DaaS30
<b>Description</b>	Create Project validation
<b>Pre-condition</b>	The user has logged in to the platform and is Adding a new Project
<b>Input</b>	On the General Information page, user leaves the mandatory fields blank
<b>Expected Output</b>	The “Next” button is disabled until all fields are filled correctly.

<b>Test Case ID</b>	DaaS31
<b>Description</b>	Create Project validation
<b>Pre-condition</b>	The user has logged in to the platform and is Adding a new Project
<b>Input</b>	On the Kubernetes Infrastructure Information page, user leaves one of the mandatory fields blank
<b>Expected Output</b>	The “Create Cluster” button is disabled until all fields are filled correctly.

<b>Test Case ID</b>	DaaS32
<b>Description</b>	Create Project validation
<b>Pre-condition</b>	The user has logged in to the platform and is Adding a new Project
<b>Input</b>	On the Kubernetes Infrastructure Information page, user adds a character under the Disk Size field
<b>Expected Output</b>	Error message to “Select the disk size” should be displayed.

<b>Test Case ID</b>	DaaS33
<b>Description</b>	Create Project validation
<b>Pre-condition</b>	The user has logged in to the platform and is Adding a new Project
<b>Input</b>	On the Kubernetes Infrastructure Information page, user has provided all the valid details, and has clicked on “Create Cluster” button
<b>Expected Output</b>	The loading message with the approximate time to create cluster should be displayed.



<b>Test Case ID</b>	DaaS34
<b>Description</b>	Create Project validation
<b>Pre-condition</b>	The user has logged in to the platform and is Adding a new Project
<b>Input</b>	The cluster is created.
<b>Expected Output</b>	After the cluster is created, the page should display the project information correctly and is disabled to make any changes.

<b>Test Case ID</b>	DaaS35
<b>Description</b>	Create Project validation
<b>Pre-condition</b>	The user has logged in to the platform and is Adding a new Project
<b>Input</b>	The cluster is created.
<b>Expected Output</b>	“Deploy App Manually” button should take the user to Kubernetes creation page.

<b>Test Case ID</b>	DaaS36
<b>Description</b>	Create Project validation
<b>Pre-condition</b>	The user has logged in to the platform and is Adding a new Project
<b>Input</b>	The cluster is created.
<b>Expected Output</b>	“Deploy from existing cluster” button should open the Deploying app page, where the user can provide necessary information.

<b>Test Case ID</b>	DaaS36
<b>Description</b>	Create Project validation
<b>Pre-condition</b>	The user has logged in to the platform and is Adding a new Project

<b>Input</b>	The cluster is created and the user is deploying app from existing cluster.
<b>Expected Output</b>	On the Deploying app page, user can add the existing url, and the username is disabled with default name as admin.

<b>Test Case ID</b>	DaaS37
<b>Description</b>	Create Project validation
<b>Pre-condition</b>	The user has logged in to the platform and is Adding a new Project
<b>Input</b>	The cluster is created and the user is deploying app from existing cluster. User has not filled all the mandatory fields
<b>Expected Output</b>	On the Deploying app page, the “Next” button is disabled.

<b>Test Case ID</b>	DaaS38
<b>Description</b>	Create Project validation
<b>Pre-condition</b>	The user has logged in to the platform and is Adding a new Project
<b>Input</b>	The cluster is created and the user is deploying app from existing cluster. User has filled all the information correctly
<b>Expected Output</b>	The “Next” button is enabled and clicking on it will take the user to Loading page.

<b>Test Case ID</b>	DaaS39
<b>Description</b>	Go to Cluster validation
<b>Pre-condition</b>	The user has logged in to the platform and has atleast one project deployed
<b>Input</b>	On the dashboard, under a project card user clicks on the “Go to Cluster” button

<b>Expected Output</b>	The user is taken to the Kube Mean App page, where the Kubernetes Admin panel is displayed
------------------------	--

<b>Test Case ID</b>	DaaS40
<b>Description</b>	Delete Project validation
<b>Pre-condition</b>	The user has logged in to the platform and has atleast one project deployed
<b>Input</b>	On the dashboard, under a project card user clicks on the Delete Project button
<b>Expected Output</b>	A warning message is displayed to user to confirm if the project needs to be deleted.

<b>Test Case ID</b>	DaaS41
<b>Description</b>	Delete Project validation
<b>Pre-condition</b>	The user has logged in to the platform and has atleast one project deployed
<b>Input</b>	On the dashboard, under a project card user has deleted a project
<b>Expected Output</b>	After deletion is confirmed, none of the details of the deleted project is available

<b>Test Case ID</b>	DaaS42
<b>Description</b>	Search Project Validation
<b>Pre-condition</b>	The user has logged in to the platform has few projects deployed on the platform
<b>Input</b>	On the Dashboard, under the projects tab, user enters a keyword in the search box

<b>Expected Output</b>	The search results should include project with the keyword searched for.
------------------------	--

**Non-Functional Testing:**

<b>Test Case ID</b>	DaaS43
<b>Description</b>	Test for SQL Injection threat while user login
<b>Pre-condition</b>	The web platform is up and running.
<b>Input</b>	Give Username as “admin” and password as “anything' OR 'x'='x”
<b>Expected Output</b>	Login should be unsuccessful

<b>Test Case ID</b>	DaaS44
<b>Description</b>	Test if passwords are encrypted
<b>Pre-condition</b>	RDS is up and running
<b>Input</b>	Query the database for user passwords
<b>Expected Output</b>	Passwords should be encrypted and not be in plain text

<b>Test Case ID</b>	DaaS45
<b>Description</b>	Check for cross site scripting
<b>Pre-condition</b>	Web platform is up and running
<b>Input</b>	Enter this URL: <a href="http://35.163.209.126:3000/home?&lt;script&gt;alert('hello')&lt;/script&gt;">http://35.163.209.126:3000/home?&lt;script&gt;alert('hello')&lt;/script&gt;</a>
<b>Expected Output</b>	The alert box should not be displayed

## **Chapter 9. Summary, Conclusions, and Recommendations**

### **Summary**

The goal behind the project was to build a platform which would help the developer and an enterprise to deliver products faster, and minimize the gap between development of product and deploying the product. We aimed at building a portal which would reduce the aforementioned bridge by using deployment automation tools and cloud based resource providers. DaaS provides an alternative to the enterprises to deploy or migrate to a public cloud. By using Kubernetes as backend service to deploy Docker containers, we are able to access and read the existing Kubernetes configuration from a currently deployed environment and hence can replicate in a different destination cloud.

We have used Docker containers, as they are lightweight solution for dependencies and configurations while moving the application between different computing environments. The use of Docker allows the enterprise to be platform independent by running on any cloud and infrastructure. Managing a large number of docker containers is a bit of an intricate process, and that is where Kubernetes makes things easier with cluster of nodes. We have used Kubernetes in order to schedule deployments and scale up the applications.

While working on the project we solved the issues an enterprise faces while doing cloud migration over multiple cloud service providers. We have minimized the time spent by a developer or an enterprise in configuring servers, and resolving the dependencies in the final hours of deployment. Our platform also helps in reducing manual configuration and deployment with patches and changes at a later deployment stage.

### **Conclusions**

In conclusion, we would like to thank our advisor for guiding us throughout the project, and helping us believe in the implementation of the initial idea. The project gave us an opportunity to learn and work together for building a platform to help the developers and enterprises. Implementing the project involved learning new technologies

and working on several platforms which help us learn the industry standards better. Working as a team, lead us to finally achieving the goal we initially started with and thus building a solution to minimize the hurdles.

### **Recommendations for Further Research**

The project covers certain scenarios and implementation covering those common use cases. The future scope of the project involves certain improvements and enhancements to the current implementation. To summarize the further scope, below are some of the features or enhancements which can be performed -

- ✓ Support multiple cloud provider(s).
- ✓ Run the application as HA cluster.
- ✓ Better utilization of user's resources.
- ✓ Support for multiple users for a single organization.
- ✓ Improved and smooth UX.
- ✓ Improved monitoring and alerts for the system.
- ✓ Add health monitoring and metrics to the system.
- ✓ More features/use-cases to support.

## **Glossary**

- [1] Deployment: To make software available for use.
- [2] Devops: Automation of infrastructure provisioning tools.
- [3] Git: Version control
- [4] Hybrid Cloud: A combination of more than one cloud providers.
- [5] Containers: A method of OS virtualization which allows to run an application and its dependencies in resource-isolated processes.
- [6] Docker: Automating the deployment of Linux applications inside containers.
- [7] Kubernetes: An open source container cluster management software.
- [8] Cluster: Collection of nodes that can be either a VM or a physical machine.
- [9] Rolling Updates: Where updates can be deployed with zero downtime.

## References

[1] Virmani, M. "Understanding DevOps & Bridging the gap from Continuous Integration to Continuous Delivery" 2015 Published by IEEE Explore. Available: <http://ieeexplore.ieee.org/document/7173368>

[2] D. Inc, "What is Docker?," Docker, 2016. [Online]. Available: <https://www.docker.com/what-docker>

[This reference provides high level overview of docker and its significance.]

[3] Inc. DigitalOcean™, "An introduction to Kubernetes," DigitalOcean, 2014. [Online]. Available: <https://www.digitalocean.com/community/tutorials/an-introduction-to-kubernetes>

[This article provides introduction to Kubernetes including details about different components in Kubernetes.]

[4] fabric 8 io and fusesource-ci, "Fabric8io/kubernetes-client," GitHub, 2016. [Online]. Available: <https://github.com/fabric8io/kubernetes-client>

[ This GitHub page explains the usage of fabric8]



## Appendices

### Appendix A.

#### Spinning Up Kubernetes Cluster:

- Select AWS cloud environment and create a EC2 instance
- SSH into the machine and set following environment variables as per the requirement:
  - KUBERNETES\_PROVIDER=aws
  - MASTER\_SIZE=t2.large
  - NODE\_SIZE=m4.large
  - NUM\_NODES=5
- Set KUBE\_AWS\_INSTANCE\_PREFIX variable if you require to deploy multiple Kubernetes cluster from the same machine
- Once the environment variables are set, navigate to /kubernetes/cluster and run `bash kube-up.sh`
- The cluster creation will take some time and once created, we can access the kubernetes UI from the Master URL.