12/15/2017

Rao, Vivaswath Jagadish

Kagalvadi Chinnaswamy, Akhilesh Kumar

Lakhani, Manan

Tirupur Raamanarasar, Shivdevkumar

FINAL PROJECT REPORT

(CS6343.001: Cloud Computing, Group C3)

ACKNOWLEDGMENT

We would like to express our deepest appreciation to all those who provided us the possibility to complete this project.

We would like to thank Dr. I-Ling Yen for providing us an opportunity to do this project. Heartiest thanks to the TA, Nidhi Solanki, for her constant help and guidance.

We sincerely thank all the official and unofficial staff members of the University of Texas at Dallas who rendered their help during this project work. Finally, we would like to thank our colleagues for their invaluable advice.

Contents

[a. Table of Figures: 3](#_Toc501091182)

[1. INTRODUCTION 4](#_Toc501091183)

[GOAL: 4](#_Toc501091185)

[2. TECHNOLOGIES USED 5](#_Toc501091186)

[2.1 BOSH: 5](#_Toc501091187)

[2.2 Cloud Foundry: 5](#_Toc501091188)

[2.3 Robocode: 5](#_Toc501091189)

[3. SYSTEM OVERVIEW 6](#_Toc501091190)

[4. INITIAL INSTALLATION 7](#_Toc501091191)

[5. FINAL SYSTEM ARCHITECTURE 10](#_Toc501091192)

[6. USER MANAGEMENT AND ROLES 11](#_Toc501091193)

[7. CLOUD FOUNDRY APIs 13](#_Toc501091194)

[8. ACCESS CONTROL DESIGN & ENFORCEMENT 14](#_Toc501091195)

[8.1 Access Control Model: 14](#_Toc501091196)

[8.2 Enforcement Via UAA(Authentication): 14](#_Toc501091197)

[8.3 Enforcement Via Orgs and Spaces: 15](#_Toc501091198)

[8.4 Enforcement Via APIs and Robocode Application: 15](#_Toc501091199)

[9. CLOUD FOUNDRY WEB-UI 16](#_Toc501091200)

[10. DATABASE DESIGN 17](#_Toc501091201)

[11. MULTI-TENANCY SCENARIOS 18](#_Toc501091202)

[11.1 Inter-tenant Access within Teams: 18](#_Toc501091203)

[11.2 Cross-tenant Access within Competitions: 18](#_Toc501091204)

[11.3 Cross-tenant Access to Resources: 19](#_Toc501091205)

[12. ROBOCODE DEVELOPMENT 20](#_Toc501091206)

[13. PERFORMANCE STUDY 22](#_Toc501091207)

[14. INITIAL OBSERVATIONS 23](#_Toc501091208)

[15. ERRORS ENCOUNTERED 25](#_Toc501091209)

[16. CONCLUSION 26](#_Toc501091210)

WORK LOAD DISTRIBUTION…………………………………………………………………………………………………………………….27

REFERENCES……………………………………………………………………………………………………………………………………………28

# a. Table of Figures:

[Figure 1: System Overview 6](#_Toc496137785)

[Figure 2: Initial setup of cloud foundry 7](#_Toc496137785)

[Figure 3: System architecture 10](#_Toc496137785)

[Figure 4: Illustration of various roles 11](#_Toc496137785)

[Figure 5: Sub-division of Team Member role 12](#_Toc496137785)

[Figure 6: Role Based Access Control Model 14](#_Toc496137785)

[Figure 7: Enforcement and mapping of role based access control model 15](#_Toc496137785)

[Figure 8: Inter-tenant Access within a team (space level) 18](#_Toc496137785)

[Figure 9: Cross-tenant Access within a competition (org level) 19](#_Toc496137785)

[Figure 10: Login Scenario 21](#_Toc496137785)

# INTRODUCTION

### According to the National Institute of Standards and Technology, Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

Cloud can be divided into three services:

1. Infrastructure as a Service(IaaS): In this type of service, virtualized computing resources is provisioned over the Internet.
2. Platform as a Service(PaaS): In this type of service, a platform is provided over the internet allowing customers to develop, run and manage applications.
3. Software as a Service(SaaS): In this type of service, some third-party provider hosts applications and makes them available to customers over the Internet.

## GOAL:

In this project we will develop a SaaS application on a PaaS platform. The chosen PaaS platform is Cloud Foundry while the application to be deployed is the robocode application.

# TECHNOLOGIES USED

For the completion of this project various technologies are required. Below is a brief description of each requirement.

## 2.1 BOSH:

BOSH is an open source project that offers a tool chain for release engineering, deployment & life-cycle management of large scale distributed services. Namely, this tool chain is made of a server (the BOSH Director) and a command line tool. BOSH is typically used to package, deploy and manage cloud software. While BOSH was initially developed by VMware in 2010 to deploy Cloud Foundry PaaS, it can be used to deploy other software (such as Hadoop, RabbitMQ, or MySQL for instance). BOSH is particularly well-suited for managing the whole life cycle of large distributed systems.

## 2.2 Cloud Foundry:

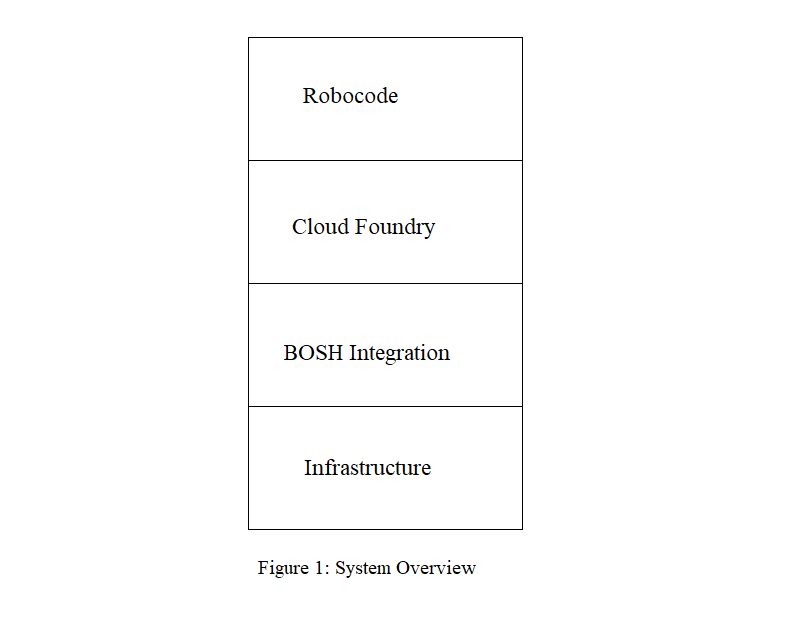
Cloud Foundry is an open source, multi cloud application platform as a service (PaaS). It consists of following components:

1. Diego: The container management system for Cloud Foundry. It consists of Diego Brain, Diego Bulletin Board System and Diego Cell.
2. Router: The router routes incoming traffic to the appropriate component, either a Cloud Controller component or a hosted application running on a Diego Cell.
3. Cloud Controller (CC): It manages the deployment of application. It also manages the orgs, spaces, user roles and services of the application.
4. Performance and fault tolerance: CC coordinate with nsync, BBS, and CellRep to assure performance.
5. Messaging: Consists of Consul and BBS.
6. Service broker: When a developer provisions and binds a service to an application, the service broker for that service is responsible for providing the service instance.

## 2.3 Robocode:

Robocode is a programming game, where the goal is to develop a robot battle tank to battle against other tanks in Java. The robot battles are running in real-time and on-screen.

# SYSTEM OVERVIEW



The system should consist of the following layers: -

1. Infrastructure: - A metal layer which provides required resources to the platform to support the service. It can be cluster or a single node machine. In our project we first try with OpenStack to create a cluster and then to switch to single machine installation.
2. BOSH integration: - Above our infrastructure we install BOSH which acts an integration tool to support cloud foundry.
3. Cloud foundry: -This acts as a platform to deploy our Robocode application.
4. Robocode: - The robocode is then provided as service to the users.

# INITIAL INSTALLATION

Initial installation was done using BOSH-LITE, a light weight installation of BOSH, on local node. Cloud foundry was then integrated on this and database services were then set up using docker method.

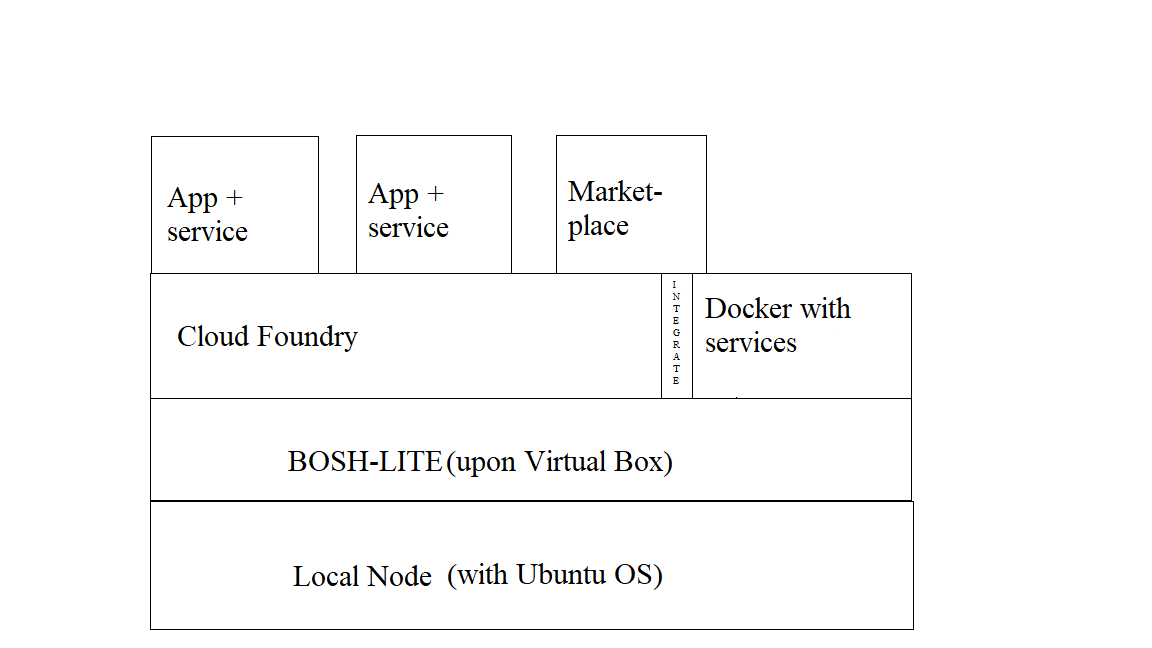


Figure 2: Initial setup of cloud foundry

The above setup was done using following steps

1. Configuration of local node:
2. A system was selected, and Ubuntu 16.04 was installed on it.
3. VirtualBox 5.1.22 was installed
4. Git was installed.

Issues: -

1. Initially latest version of VirtualBox was installed. However, in future steps due to a bug in the latest version of VB bosh installation could not be done. Hence VB version 5.1.22 was installed.
2. Installation of BOSH-LITE (BOSH 2 METHOD):
3. The BOSH CLI was installed.
4. We then bootstrap the BOSH-LITE director to our local VirtualBox installation.
5. We setup the BOSH-LITE environment.
6. We then create BOSH alias and setup environment variables.

Issues:

1. Initially BOSH-LITE was installed using vagrant based method. However, this was discontinued as vagrant based method does not support latest installation of cloud foundry.
2. VirtualBox 5.1.22 must be used as other versions contain a bug which hinders the installation process.
3. Deploying Cloud foundry:
4. For our deployment we will use the cf-deployment repository available in github (https://github.com/cloudfoundry/cf-deployment) which provides the latest version of cloud foundry.
5. Clone the above repository.
6. Upload the stemcell which is available in the repository.
7. Create and upload the cloud foundry release.
8. Finally, create a local route to access the cloud foundry environment.
9. Configure cloud foundry:
10. Install the cf CLI.
11. Login to cloud foundry.
12. Create org and space.
13. Target the org and space.
14. Push a sample application.
15. Creating marketplace for cloud foundry:
16. Clone the docket broker service repository which is available at github (https://github.com/cloudfoundry-community/docker-broker-deployment).
17. Create the environment and integrate it with cloud foundry by using BOSH deployment method.
18. Register the service broker.
19. Access the services from cloud foundry marketplace.
20. Linking service to application:
21. An instance of the service present in the marketplace is created.
22. This instance is then bound to the application.
23. The application is restarted.

Issues:

Initially the service could not be used by the application. Upon research it was found out that an application security group should be defined for the service.

This was done as follows:

1. First create a json file and add following parameters:

[

{

"ports": "REPLACE WITH MYSQL PORT NUMBER",

"protocol": "tcp",

"destination": "REPLACE WITH MYSQL HOSTNAME"

}

]

1. Use cf command to create a security group using above json file.
2. Bind the security group to the default security group of cloud foundry.
3. After this just follow above steps of linking service to application.

# FINAL SYSTEM ARCHITECTURE

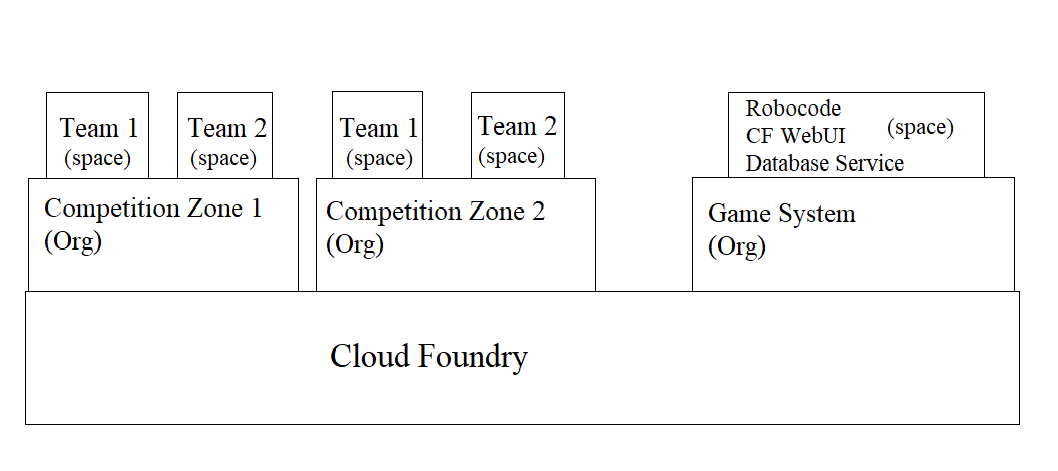


Figure 3: System architecture

Cloud foundry provides orgs and spaces for isolation, multi-tenancy and easy deployment and access of apps. We have built on this to create a “SaaS-ified” implementation of robocode. In our project, we implement the system as a gaming platform where robocode is given as a gaming service to various teams. This is further elaborated in the following points:

1. Initially, cloud foundry will be divided into multiple “Competition Zones”. These Zones can represent various areas in real world. For example, one competition can be held in the Texas region while another can be held in the California region. These zones are created over cloud foundry using the concept of orgs.
2. Within these competition zones, we will have multiple teams who will use the robocode application to build the robots. Then these teams will compete with each other by battling with their created robots. These teams will be created using the concept of spaces. The teams in our system will represent the tenants.
3. There will another separate org known as the “Game System” which will contain a space containing the actual Robocode application along with CF WebUI for managing users and a database service to provide support to the robocode application.

The various mechanisms and techniques used to implement the above architecture will be explained in detail in the following sections of the report.

# USER MANAGEMENT AND ROLES

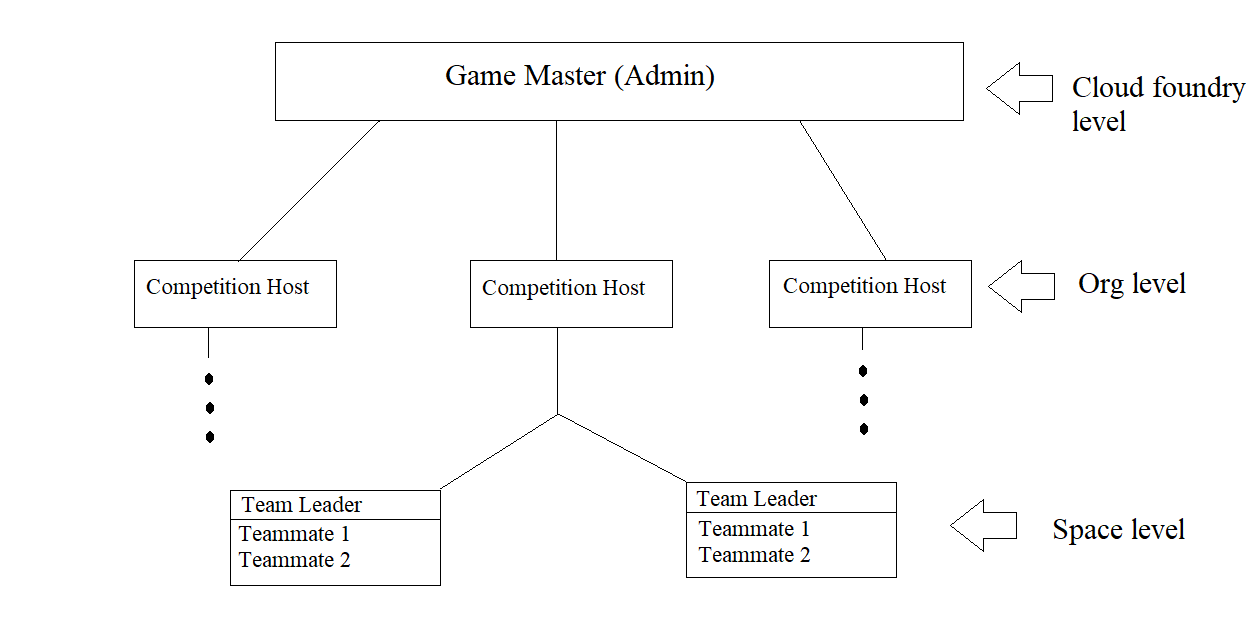


Figure 4: Illustration of various roles

Cloud foundry provides various default roles to assign to users. These roles are:

1. Admin: An admin user has permissions on all orgs and spaces.
2. Org Managers: Managers or other users who need to administer the org.
3. Space Managers: Managers or other users who administer a space within an org.
4. Space Developers: Application developers or other users who manage applications and services in a space.
5. Space Auditors: Users view but cannot edit the space.

We have extended these roles to create specially defined roles that act will late act as the foundation stones to our access control model. These roles are explained in detail below:

1. The Game Master: The main job of the Game Master is to create various competition zones across cloud foundry. The game master is also responsible for creating user id and passwords for all users of this gaming platform. The game master is also the only user who will access to the “Game System” org, so that they could manage the robocode and associated database services. This role is created using the admin role provided by cloud foundry.
2. Competition Host: The users who have been assigned this role oversee creating the respective spaces required to host a team. They assign various roles to the underlying team members of the space such as registering team members in the competition (assigning them a role in the Org) and appointing a Team Leader to lead the team. This role is mapped using the Org Manager role provided by cloud foundry.
3. Team member: These are the users that actually use the Robocode application. For efficient access, we will be splitting team members into three sub-roles. These are:
4. Team Leader: A team member is appointed to be the leader of their team by the Competition Host. This is done using the Space Manager role provided by cloud foundry. A team leader can they assign the role of a Builder or Player to their teammates.
5. Builder: Users with this role can build robots for their team. They can also view and play with the robots. This role is created using the Space Developer role provided by cloud foundry.
6. Player: Users with this role can only view and play with the robots. This role is created using the Space Auditor role provided by cloud foundry.

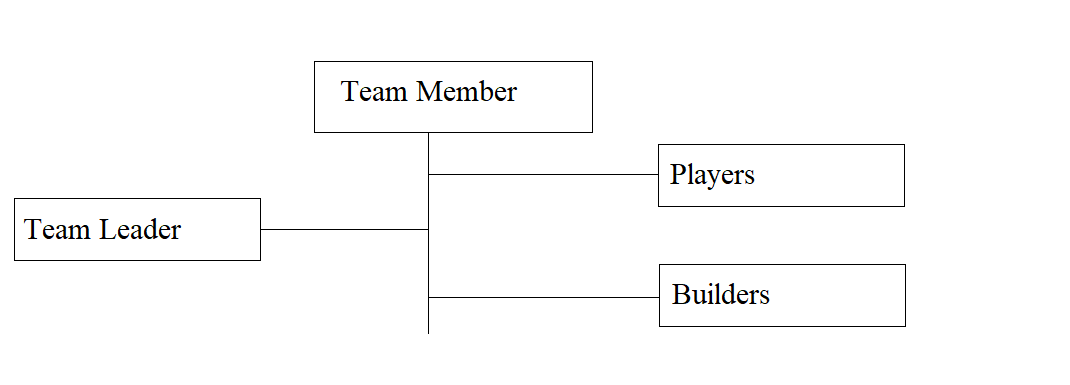


Figure 5: Sub-division of Team Member role

# CLOUD FOUNDRY APIs

To access the information regarding the user roles, space and org information, we will use the APIs provided by BOSH and cloud foundry. These API’s are accessed using the token obtained from UAA authentication methods (this will be explained in detail n following sections of this report).

The various APIs used are:

1. uaa.bosh-lite.com/oauth/token: This API is used to get the initial tokens required to provide secure environment for our app. The access token obtained will also be used as an Authorization token by the below APIs to access various information of the logged in user.
2. api.bosh-lite.com/v2/spaces: This API is used to get information about the space the user has access to. Using this API, we get a unique GUID associated to the space.
3. api.bosh-lite.com/v2/spaces/GUID/developers: This API returns a list of developers belonging to the space associated to the GUID.
4. api.bosh-lite.com/v2/spaces/GUID/managers: This API returns a list of managers belonging to the space associated to the GUID.
5. api.bosh-lite.com/v2/spaces/GUID/auditors: This API returns a list of auditors belonging to the space associated to the GUID.

These APIs are integrated in the robocode application and is used to generate necessary session variables and to obtain cloud foundry defined roles.

# ACCESS CONTROL DESIGN & ENFORCEMENT

## Access Control Model:

In our implementation, we have used the Role based access control model to design the system. The users are provided various rights based on the roles that they are assigned.

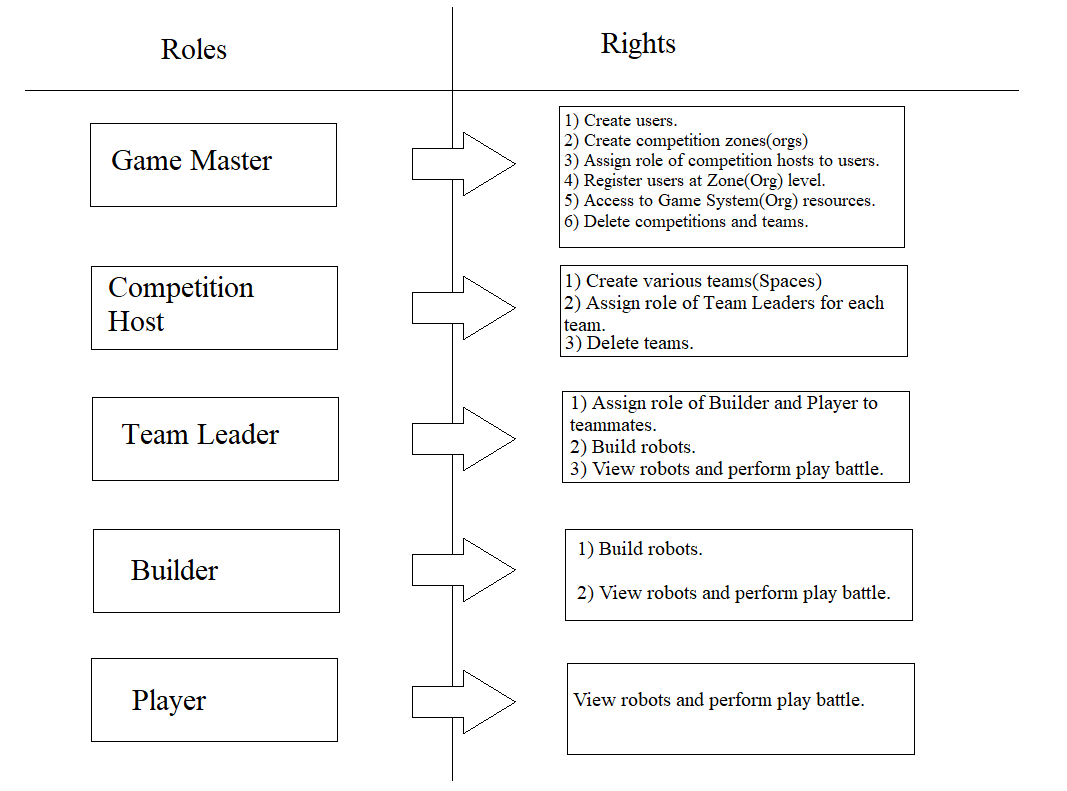


Figure 6: Role Based Access Control Model

## Enforcement Via UAA(Authentication):

We use the UAA hosted on BOSH-LITE to authenticate users. This UAA acts as an OAuth2 provider, issuing tokens for client applications to use. A valid credential will result in the production of access tokens which contain user information. We use the UAA login as the first stage of enforcement for our role based access control. This done by mapping the tokens to respective cloud foundry APIs to get user level information which will then be mapped into roles.

To use UAA login, we must do the following steps:

1. Install the uaac gem, a command line client to access UAA present on BOSH-LITE.
2. Register the robocode app on the uaac and provide necessary scopes and redirection URL.
3. Integrate the UAA login page in the robocode app with necessary parameters.

## Enforcement Via Orgs and Spaces:

Just the token is not enough for enforcement, we also need to map necessary users to cloud foundry provided roles. This was discussed in detail in the previous section of User Management and roles. Thus, when a user is mapped into a specific Org or Space role, the information will be present in the access token associated with the user.

## Enforcement Via APIs and Robocode Application:

The next stage involves the conversion of cloud foundry roles into our user defined roles according to the role based access control model. This is done at application level. We use the access token obtained from the UAA and map it to cloud foundry APIs to obtain user information and their cloud foundry roles. We then apply programming logic to convert these into user defined roles. Once user roles are obtained we then apply restrictions on them based on the role based access control model.

The entire process explained above is diagrammatically represented below:

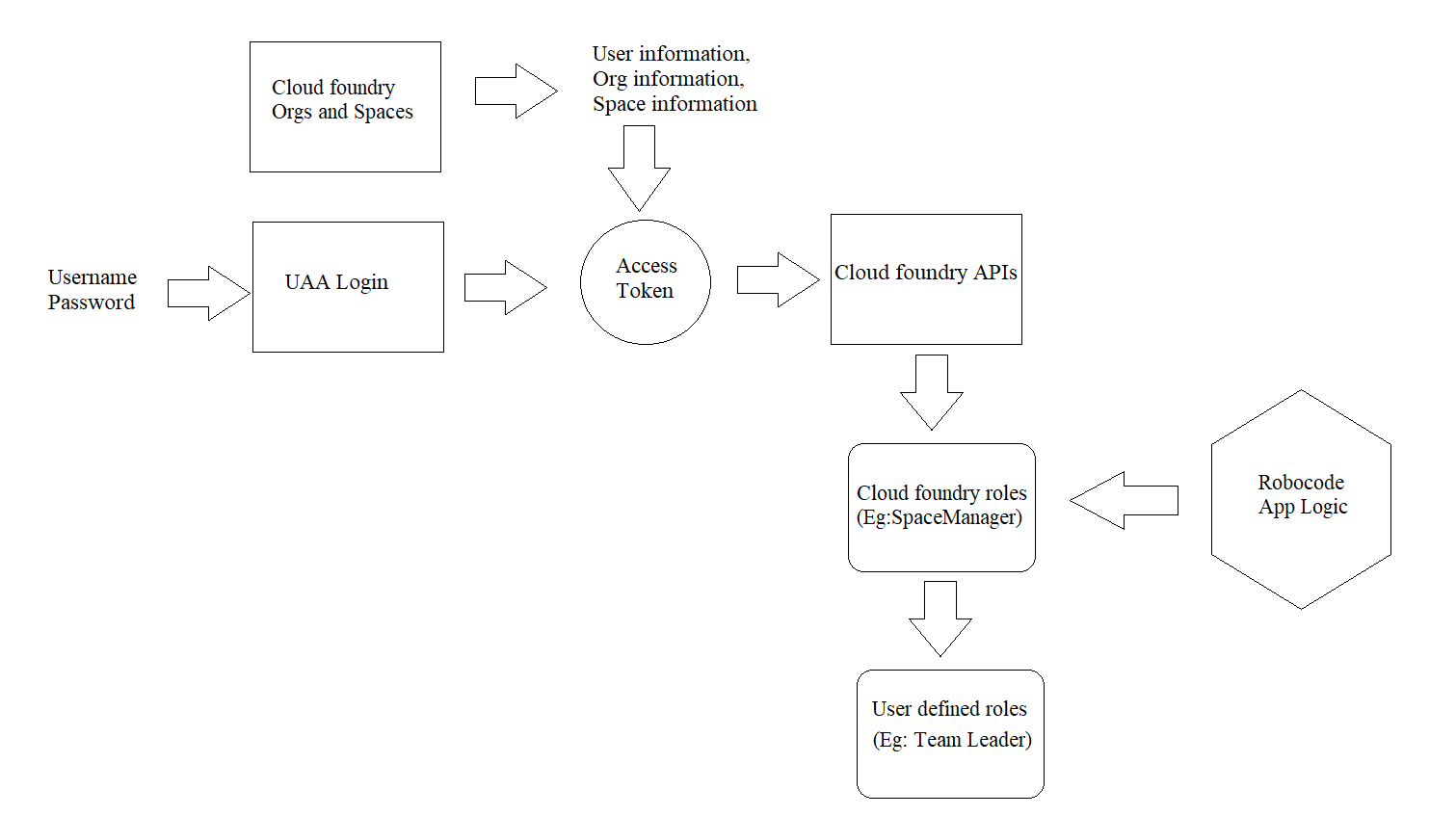


Figure 7: Enforcement and mapping of role based access control model

# CLOUD FOUNDRY WEB-UI

Instead of using terminal to implement enforcement via orgs and spaces, we can use cloud foundry web apps to create space and orgs as well as assign user roles. These tools are known as CF-WebUI. They provide a means for user to login and see their responsibilities and duties.

The CF-WebUI will behave differently for different users (the mappings mentioned below are all done using default cloud foundry roles):

1. If the game master logs in, they will be able to create competition zones(orgs) and the assign the role of competition host (Org Manager) to any of the users. They can also register all users at org levels. Also, the game master will have access to resources such as the actual robocode app and database services.
2. If the competition host logs in, they will be able to create teams(spaces) and assign the role of team leader to any team member (Space Manager) of the created team.
3. If the team leader logs in, they will be able to assign role of builder (Space Developer) or player (Space Auditor) to their teammates.
4. If a player or builder logs in, they will be able to only view details of their teams and the competition to which they belong.

Along with above feature, the CF-WebUI will also give users access to certain features specific to their default cloud foundry roles. For example, Game Master(admin) can view and edit features of the deployed robocode app.

We have experimented with two major CF-WebUI. They are explained in detail below:

1. Icclab WebUI: This WebUI is developed using AngularJS and Bootstrap. It provides basic features such as staging applications, managing user roles and monitoring features.
2. Stratos WebUI: This WebUI allows users to manage their applications and cloud foundry deployments. It does so by managing "endpoints", where each endpoint is a reference to a Cloud Foundry deployment. Stratos stores endpoint metadata in a relational database. All users can connect to these endpoints using their credentials, ensuring that they get the appropriate level of access when interacting with Cloud Foundry. This WebUI provides more features when compared to Icclab WebUI, even allowing admins of cloud foundry to directly deploy applications with the need of cloud foundry CLI.

In our project, we will be using the Stratos WebUI due to the benefits it provides to the user.

# DATABASE DESIGN

Database is deployed in cloud foundry via the docker method. We use the MySQL database to store our robots. We must provide necessary security groups to provide access to database. We have explained in detail the steps required to deploy and launch MySQL as a service in the previous section of Initial development.

Database in our system plays a vital part as it oversees multi-tenant access control. In our system we treat Team Name as the Tenant ID. This tenant ID will be used to control inter-tenant access. For cross tenancy we will use Competition Name as the restrictor. These multi-tenant scenarios are discussed in detail in the following section.

# MULTI-TENANCY SCENARIOS

In our system the team name (space name) will act as the Tenant ID, which will be stored in the database. We have implemented three types of multi-tenant access in our project:

## Inter-tenant Access within Teams:

In our project we have provided inter-tenant access by means of Players and builders within a team(space). This done at robocode application level. The process of access is explained below:

1. A Builder will create robots. These robots will be stored in the database along with the Tenant Id.
2. These robots are then accessed by other teammates belonging to the builder’s team. This access is controlled such a way that, Players can only view and play with the created robots while other builders can build as well as view and play with the created robots.

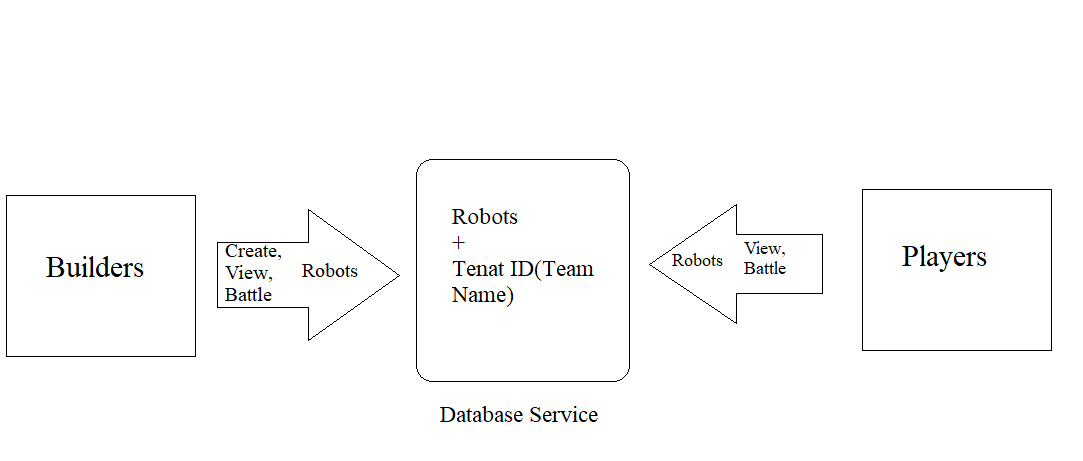


Figure 8: Inter-tenant Access within a team (space level)

## Cross-tenant Access within Competitions:

We provide cross-tenant access in terms of Play battle for a competition zone(Org). That is, any team within a competition can play with another team’s robots. However, they cannot view or edit these robots.

The process of access is explained below:

1. Teams create and store robots in the database along with their Tenant ID.
2. Teams can then play with these stored robots. However, they cannot view or edit the code of these robots. This restriction is enabled using Tenant ID.
3. Finally, teams of one competition should not be able to access another competition’s robots. This is done by maintaining a separate Org ID (Competition Name) in the database to restrict access.

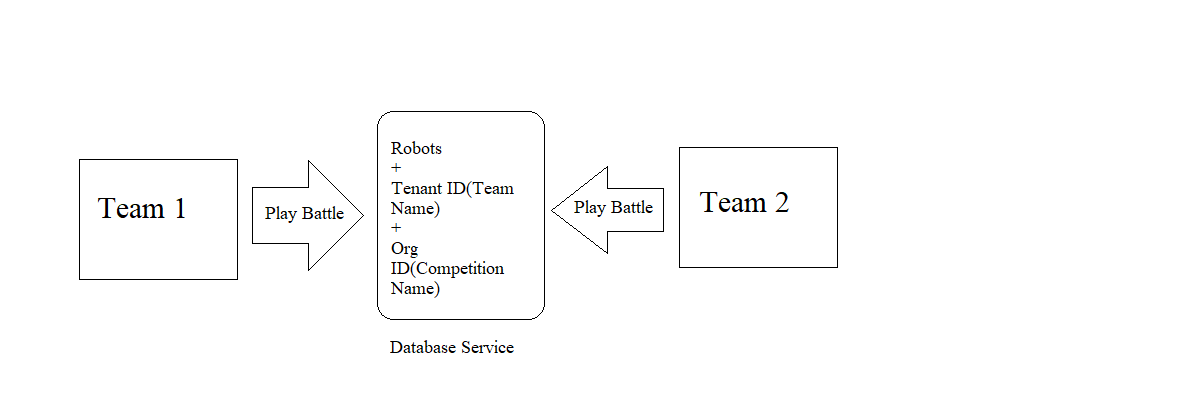


Figure 9: Cross-tenant Access within a competition (org level)

## Cross-tenant Access to Resources:

The second cross-tenant access provided in the system involves the access of robocode application, database service and CF WebUI to all teams across cloud foundry. The robocode application and the CF WebUI are accessed using the API endpoints while database service is accessed within the robocode application itself. The Robocode, WebUI and database services are deployed in a separate org (called Game System) and space (called resources). Only Game Master(admin) has access to this org and space.

# ROBOCODE DEVELOPMENT

Robocode is an application that help users to create robots to experiment their algorithm with other users. It enables user to not only write modify Robots (Java Classes) but also compile them and run them with other java files.

The following were the changes made to the existing the Robocode:

1. The Robocode did not support login mechanism, we modified the Robocode to use the cloud foundry UAA to provide login and creating users in the UAA.

a. It involved the configuration at the UAA side to create a sample client configuration(“testlogin”) to redirect to UAARedirectServlet on Robocode, once the app is authorized and token is generated at UAA. This configuration also sets the permission(scope) that this client would have (openid,cloud\_controller.read --authorized\_grant\_types 'authorization\_code,client\_credentials').

“uaac client update --name testlogin --scope openid,cloud\_controller.read --authorized\_grant\_types 'authorization\_code,client\_credentials' --access\_token\_validity 1209600 --authorities oauth.login --redirect\_uri 'http://robocode.bosh-lite.com/UAARedirectServlet'

b. In the Robocode, we made Oauth calls to Robocode utilizing Apache Oltu library to fetch the token from the UAAC, and once token is retrieved, it sent back to cloud foundry to get the user details. This user details are stored in session of the cloud foundry. In every page of the Robocode “header Page” is accessed and a check is made to see if the username is available at the time. If the username is available, then the Robocode will be shown, else if user is not available at any page, it is redirected to logout page. The above procedure is described in the diagram.

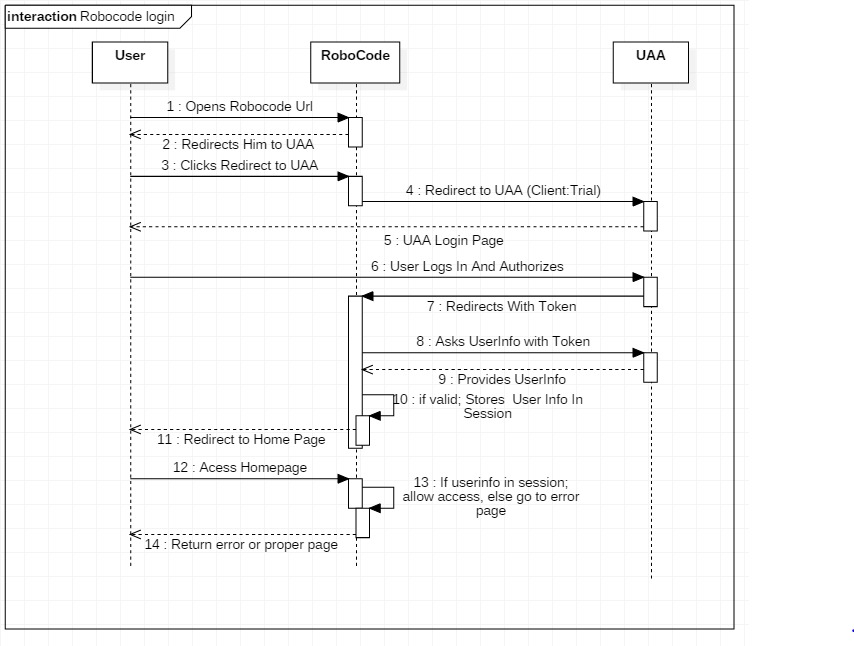


Figure 10: Login Scenario

2. The modified Robocode supports roles and these roles are defined in the Cloud Foundry. Since the cloud Foundry supports only few roles and no custom roles we consider following roles in the cloud foundry to map to Robocode role like “Manager” (Team leader), “Developer “(Builder) and “Auditor” (Player). These roles are used to decide the role based access in the Robocode. This role based access is defined in 8.1. In Robocode, to enable role based access the new queries were made, and the application logic is also adapted to support them.

3. The modified Robocode also supports Multi-tenancy for which the new column were added on the database like tenant id and org id which are again fetched from Cloud Foundry and stored in database and the application logic is also adapted to support them.

# PERFORMANCE STUDY

1) Deployment time:

We have measured the time taken to deploy our robocode by using a shell script that will take the difference between the start time and end time of deployment.

The shell script is as follows:

#! /bin/bash

START=$(date +%s)

cf push robocode -p robocode.war

END=$(date +%s)

DIFF=$(( $END -$START ))

echo “Time taken is $DIFF seconds”

Generally, it took around average of 120 seconds to deploy the robocode application.

2) Memory Limit:

Generally, a space can hold applications up to 4GB. If this is bypassed it may throw an error or crash. Also, multiple apps across spaces will also decrease the efficiency of cloud foundry.

# INITIAL OBSERVATIONS

In this section we will briefly discuss about our initial deployment on cluster using OpenStack and the issues faced by this method.

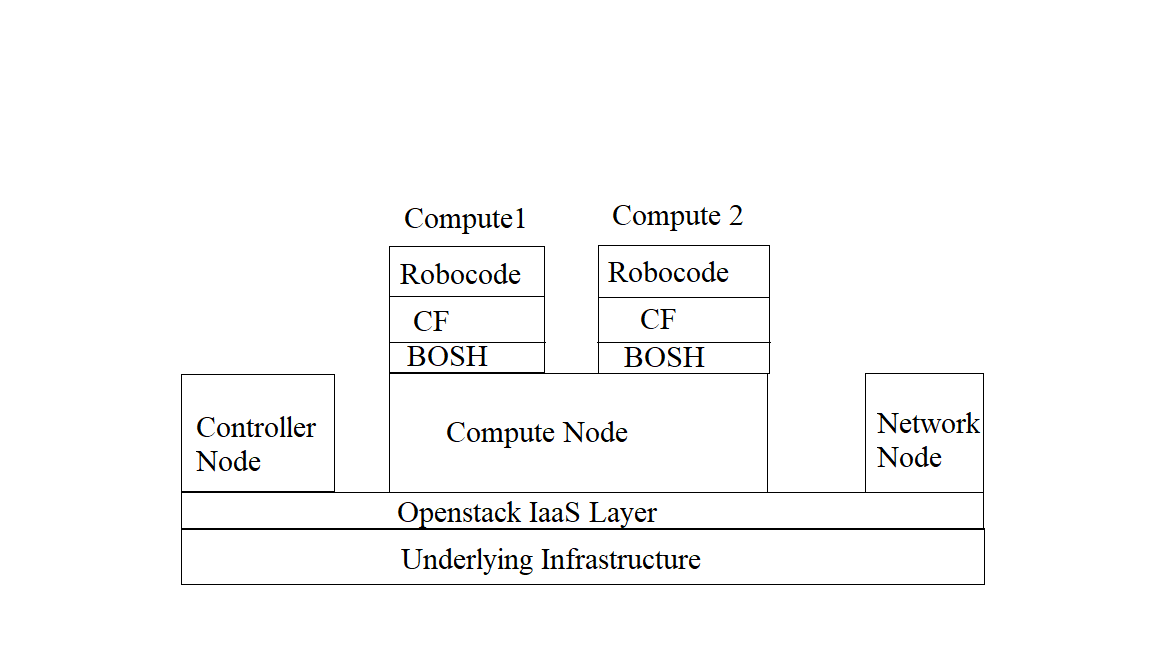


Figure 10: Installation on a Cluster

OpenStack was used to create a cluster of machines upon which cloud foundry was installed. However, BOSH integration was a failure.

Steps followed:

1. Creation of Cluster:
2. Three machines where chosen and NIC cards were installed in them. Then CentOS was installed as the base OS.
3. OpenStack (version: Newton) was installed using Packstack method as follows:
4. Name the three machines as controller, compute and network
5. Edit the hosts file of all three machines and add the IP addresses of individual machines.
6. Stop and disable firewall and network manager.
7. Disable SELinux.
8. Configure ssh from controller to compute and network nodes.
9. Set OpenStack Newton repository and install packstack utility.
10. Generate and update OpenStack answer file according to our infrastructure.
11. Start the installation.
12. BOSH Integration:
13. Prepare the OpenStack environment by setting up key pair, floating ips, networks and security groups.
14. Install the BOSH CLI.
15. Create the BOSH Environment.

Issues faced: -

1. OpenStack-
2. Error: No hosts available

Solution: Resolved by partitioning Centos root and home folder into separate partitions.

1. Bosh setup-
2. Error: Improper identity

Solution: Resolved by giving Key Stone access.

1. Error: Unable to connect to director

(Could not resolve this error)

# ERRORS ENCOUNTERED

Throughout the project various errors were encountered, some of which were resolved. We have provided information about these errors below,

1. Failure of installation on cluster: This has been thoroughly explained in the above section of Initial observations.
2. Failure of vagrant method of installation: Initially a vagrant method was used to install BOSH-LITE. However, this was abandoned as the latest version of cloud foundry was not compatible with this method. We then used the BOSH 2.0 method to deploy cloud foundry.
3. Reliance on VirtualBox 5.1.22: BOSH integration fails on VirtualBox versions above 5.1.22.
4. Sudden shutdown of machines due to power failure would result in cloud foundry crashing.
5. MySQL security access: A security access group should be created to access MySQL.
6. Open URL: Anyone can access URL of deployed app. To solve this use UAA and sessions.
7. SSL certificate error: To access UAA, proper certification is required to perform handshake.
8. Applications deployed using BOSH-LITE cannot be accessed from other machines.
9. Cloud foundry provides only two level of access control and less number of user roles to develop upon.
10. Cloud foundry slows down or may even crash if apps greater than 4GB is deployed.

# CONCLUSION

In this project we have thoroughly explored the Cloud foundry platform. We have performed various deployments methods: On cluster and local node. We have integrated services over cloud foundry and finally created an architecture to support the Saasification of robocode.

In this report we highlight the entire process of building the cloud foundry from the bottom up till deployment of robocode as a SaaS on top of it. We have also discussed the various roles necessary to implement the system. We then laid out the access control model needed to develop this system along with its enforcement mechanisms. We have shown how multi-tenancy works in our system also. Finally, we discuss about the performance factors as well as some common errors encountered throughout the process.

WORK LOAD DISTRIBUTION

The weekly load and work distribution was tracked using the word document, the link to this word document is provided below:

<https://docs.google.com/document/d/1CGOp6dWHNgEPl8o9TwN-3Wuzgyvt_Y9Oo4kHpE156L8/edit>

REFERENCES

[1]<https://www.cloudfoundry.org/>

[2]<https://www.openstack.org/>

[3]<https://bosh.io/>

[4]<https://www.linuxtechi.com/multinode-openstack-newton-installation-centos7/>

[5]<https://banck.net/2017/03/deploying-cloud-foundry-virtualbox-using-bosh-cli-v2/>

[6]<http://www.starkandwayne.com/blog/adding-docker-based-services-to-your-cloud-foundry-with-bosh2/>

[7] <https://docs.pivotal.io/p-mysql/1-7/app-security-groups.html>

[8] Stratos Web UI : <https://github.com/SUSE/stratos-ui>

[9] Postman to do get request: <https://www.getpostman.com/>

[10] cloud foundry Cloud Foundry API: <https://apidocs.cloudfoundry.org/280/>

[11] Json to Pojo converters : <http://www.freecodeformat.com/json2pojo.php>