**CA674 Cloud Architecture MCM – Project**

Juhi Shrivastava

16212548

**Creation and Deployment of the Diet Share App on**

**PaaS OSS – Cloud Foundry, Openshift**

**Abstract:**

Cloud computing is where software applications, data storage and processing capacity are accessed over the internet. The 3 service models of cloud computing are Saas (Software as a service), Paas (Platform as a service) and Iaas (infrastructure as a service).

The purpose of this project is to carry out an extensive research on Paas Cloud Platform and exploration of open source software OSS.

1. A comparative analysis of the open source cloud computing PaaS - Cloud Foundry, Openshift, Stratos and Appscale addressing the technical concerns like the Performance, Scalability, Interoperability etc. and the architectural components is provided.
2. In addition, to demonstrate the platform functionality an app is developed on Laravel Framework for Php programming language – “Diet Share”. The main objective of the application is to facilitate people to add their diet plans for future references.
3. Furthermore, the same app is deployed on Cloud foundry and Openshift OSS.

**Contents:**

1. Introduction to PaaS:

1.1 Motivation – Why Paas:

1.2 What is Paas:

2. Characteristics of PaaS:

3. Major PaaS Cloud Providers:

4. Comparative Analysis of Cloud foundry, Openshift, Stratos and Appscale:

4.1 Architectural Comparison:

4.2 Performance, Interoperability and Scalability Comparison:

5. App Development:

6. Laravel and Screenshots

6.1 Installing Laravel

6.2 Screenshots

7. Application Deployment on Cloud Foundry:

8. Challenges while deploying the Application:

9. Cf essential commands:

10. Application Deployment on Openshift:

11. Challenges while deploying the Application:

12. Openshift essential commands

13. Appscale

13.1. Installation and Deployment

13.2 Limitations of using Appscale

14. Apache Stratos

14.1 Installation Prerequisites and setup

14.2 Deployment

14.3 Limitations of using Stratos

15. Conclusion:

16. References:

**1.Introduction to Paas:**

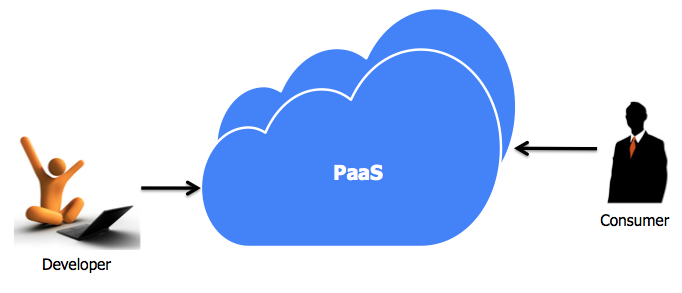
PaaS stands for Platform as a service, a layer of cloud computing services.

**1.1 Motivation – Why Paas ?**

Paas has become the future of cloud computing as with the evolution of Platform as a service we do not need to spend [1] long hours in downloading, compiling, installing, configuring, and connecting all sorts of components—and that is just to get an application up and running on a single virtual server instance. This is so unprogressive and takes away the time one would use to innovate and improve the application. There are many advantages of using Paas over “doing it yourself” in Iaas which includes agility, efficiency and quality etc.

* 1. **What is Pass?**

Platform-as-a-service (PaaS) is a type of cloud computing model that deliver applications as a platform over the internet. It allows the user to develop their own cloud applications using the tools and languages provided by the providers. [2] A PaaS provider hosts the hardware and software on its own infrastructure. The Service user is responsible for creation, updating and maintenance of the application. Paas offers services targeted to the developers that would render them to develop and deploy applications without thinking about the underlying Infrastructure.



Paas works on the pay-as-you go basis i.e when the users access the PaaS applications through a web browser, they get charged for that access. PaaS automates the configuration, deployment and ongoing management of applications in the cloud. It reduces the setup cost of the software development as they don’t have to invest in physical infrastructure because it already includes the operating system, hardware, database, server, and network which in turn increase the speed of the application deployment.

**2. Characteristics of Paas:**

There are several key traits that PaaS solutions have in common, some of them are:

* Multi-tenancy architecture: This facilitates concurrent use of the cloud computing resource by multiple users and common technical resources and code instance.
* Customizable/Programmable User Interface: It provides a highly flexible UI(user interface) via simple drag and drop which renders the creation of the components on the fly.
* Runtime Framework*:* [3] This is like the "Software stack". PaaS runtime framework executes the user code as per the Cloud Provider and Application owner policies
* Abstraction*:* PaaS provides the higher level of abstraction compared to IaaS. With PaaS, user can deploy its applications into the pool of computing resources, eliminating the infrastructure configuration and complexity of deployment
* Automation*:* PaaS runtime environment automates the application deployment and configuration. It also configures the load balancers and the databases
* Cloud Services: It provides various built-in cloud services and eliminates the integration requirement of these services

**3. Major PaaS Cloud Providers:**

There are various types of PaaS providers like Google app engine, Morpheus Data, Heroku etc. The open source cloud computing platforms which we have used while working on this report are:

* Pivotal Cloud Foundry*:* Cloud foundry is an open source cloud computing PaaS developed in Ruby by VMware under the terms of Apache License 2.0. It is open and extensible in nature thereby avoiding ***vendor lock-ins***. It supports the full lifecycle of an app starting from development, through all testing stages, to deployment.
* Redhat Openshift*:* Red Hat OpenShift depends on open source applications and offers a wide assortment of databases and segments. It is a free and open source cloud-based platform which allows developers to create, test and run their applications and deploy them to cloud.
* Apache Stratos*:* Apache Stratos is a platform which provides a framework that accommodates various elements such as the server support called Apache Tomcat, programming language such as PHP and DBMS system MySQL.
* Appscale*:* AppScale, is an open source implementation of Google App Engine provides runtime environments and API endpoints application development technologies such as Python, PHP, Go and Java. This easily removes the need for tasks related to operations and server maintenance and the focus can be completely curated to development alone.

**4. Comparative analysis of Cloud Foundry, Openshift, Stratos and Appscale:**

This section provides the relative study of the different PaaS cloud Platforms with respect to the architecture and the functionality of the Paas Platforms. Futher more a dissection on the technical aspects addressing the interoperability, scaling and performance is provided :

**4.1 Architectural Comparison:**

The table below shows the open Paas components comparison for all the 4 cloud platforms

selected:

**Openshift**

Openshift is a layered system designed to expose core Docker and Kubernetes concepts as precisely as possible, with easy composition of applications by a developer being the focus. The abstraction for packaging and creating Linux-based , lightweight containers is provided by Docker. Kubernetes provides the cluster management and orchestrates Docker containers on multiple hosts. Openshift adds source code management, builds and deployments for developers and also managing and promoting images at scale as they flow through the system. Application management at scale and also team and user tracking for organizing a large developer organization are also a vital feature that is added due to openshift.

Openshift utilizes a microservices based architecture of minor, decoupled units that work together. A reliable clustered key-value store with data about the objects stored in etcd can run on a Kubernetes cluster. These service can be divided into two functions namely REST APIs , which expose each of the core objects, and controllers, which read those APIs, apply changes to other objects and provide a status report.

To change the state of the system , users make calls to REST API and controllers use the API to read the user's desired state and then bring the other parts of the system into sync. The controller pattern indicates that most of the functionality in Openshift is extensible. How the images and deployments are handled can be customized separately depending on the way the builds are run and launched. From a system admin perspective, it means that the API can also be used to script common administrative actions on repeating schedule.

**CloudFoundry**

The application execution engine and automation engine form the core architectural components of Pivotal Cloud Foundry. The application execution engine administers the buildpack mechanism for adding frameworks and services where the automation engine essentially takes care of deployment and lifecycle management. Tasks such as container creation and destruction is automated internally which adds to the scalability of the deployed application. Additional components such as routers and service brokers provide a framework for network traffic routing and service binding.

**Stratos**

The architecture of apache stratos is made of three layers over a Infrastructure as a service layer. The Applications layer comprises of the applications that are deployed over the framework. The Cartridges layer provides a plug and play platform for software development frameworks such as spring and joomla. The stratos framework layer consists of abstract components for Load balancing, message brokering, api service providing and auto scaling.

**Appscale**

Appscale implements the design model of google app engine providing an API driven application development environment. The core functionalities of a deployable application is monitored and controlled by the App Controller. Appscale also provides message brokering and instance management. The AppServer component provides software framework for web application development.

[4]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Functionality** | **Cloud foundry** | **Openshift** | **Stratos** | **Appscale** |
| Core functionality | Cloud controller | Broker | Cloud Controller | AppController |
| Providing third party database services | Service Broker | Cartridge | Cartidge | Database Master |
| Routing of incoming traffic | Router | Rest API | Load Balancer | AppLoadBalancer |
| Querying the state of apps | Cloud controller | Broker |  | App Scale tools |
| Messaging | Message Bus | Broker |  | AppController |
| Application instance management | Droplet execution agent | Node |  | AppServer |
| Application state change | Health Manager | Broker |  | AppLoadBalancer |
| Containerization | Warden | Gear |  | Database Slave |
| Load balancing of user requests | Droplet Execution Manager | Broker |  | AppLoadBalancer |
| Framework provider | Blob store | Cartridge |  | AppServer |

Table below shows the Cloud Foundry, OpenShift, Stratos and Appscale PaaS supported languages (java, python, ruby), databases (MongoDB, MySQL, HBase) and frameworks (spring, rails, and flask). In OpenShift, languages and databases are supported in the form of cartridges. User defined cartridges are also allowed in OpenShift. Cloud Foundry provisions languages in the 60 Computer Science & Information Technology (CS & IT) form of build packs. Users can also pick to write their own build packs. Cloudify, Cloud Foundry and Openshift have extensible language support feature.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Cloud Foundry** | **Openshift** | **Stratos** | **Appscale** |
| **Languages** | Java, Ruby, Scala, Node.js, Groovy, grails, PHP, Go, Python | Java, PHP, Ruby, Python, Perl, JavaScript, Node.js | Java, PHP, .net | Python, Java, Go, PHP |
| **Databases** | MonogoDB, MySQL, PostgreSQL | PostgreSQL, MySQL, MongoDB | MySQL, PostgreSQL,  Cassandra,  mongoDB | Cassandra, HBase, Hypertable, MongoDB, SimpleDB, MySQL |
| **Frameworks** | Spring, Rails, Grails, Play, Sinatra | Rails, Flask, Django, Drupal, Vert.x | Spring, Struts,  Joomla | Django, Flask, Spring |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Features** | **Cloud Foundry** | **Openshift** | **Stratos** | **Appscale** |
| **Relational DB Support** | Yes | Yes | Yes | Yes |
| **NoSql DB support** | Yes | Yes | No | Yes |
| **Horizontal Scaling** | Yes | Yes | Yes | Yes |
| **Vertical Scaling** | Yes | Yes | No | No |
| **Autoscaling** | No | Yes | Yes | Yes |
| **Spring Framework Support** | Yes | No | No | Yes |

**4.2 Performance, Interoperability and Scalability Comparison:**

Interoperability is the capability of a system or a commodity to operate with other systems or commodities without specific effort on the portion of the customer.

Stratos is an open PaaS supports various Infrastructure as a Service (IaaS) environments, languages, multiple application platforms, and frameworks.  The Apache Stratos cartridge model and jCloud abstraction layer facilitates deployment on popular IaaS environments (Amazon AWS, vCloud, OpenStack), and units can consolidate their chosen application servers via cartridge extensions.

In providing IaaS interoperability through the DeltaCloud API, there has been some criticism that OpenShift’s future is tied to general adoption of DeltaCloud, however, this is not actually not true. DeltaCloud is a “wrapper layer” that outlines several APIs onto an individual API.  It may well be a lowest-common-denominator with regards to other Cloud APIs, but as far as that lowest common denominator is sufficient for OpenShift, then Red Hat has no dependency on extensive adoption of DeltaCloud since it can manage the mapping to and from other APIs itself.

CloudFoundry is created on the vSphere IaaS infrastructure (provided by VMware or others), though a fig leaf of portability to Amazon IaaS infrastructure is being rendered by RightScale. The two vendors inform regarding openness and the aptitude to run a wide range of languages, infrastructure, and application frameworks, it is also worth examining if the two platforms themselves are interoperable.

These platforms are traded to developers with the key message comprising that developers don’t need to be architects – the code magically falls on to the cloud and infrastructure is presented to it on request.  It may be confusing to non-developers that the interface to all of this is really a command-line. CLI is published by the “Launch Configuration” deployment tools that are installed in the Integrated Development Environment (IDE), which is Eclipse.  The sales-pitch to developers is around analyzing the formation of this command-line, and of automatically stocking the underlying architectural elements to that it deploys.

Apache Stratos is horizontally and auto scalable. Limited by resource (instance capacity) availability. Appscale also supports horizontal and automatical scaling. That is the applications can be scaled horizontally or let AppScale automatically scale the instances. Open shift can scale the applications vertically, horizontally or let OpenShift Online automatically scale the instances. Cloud Foundry can only scale the applications vertically and horizontally.

**5. App Development:**

Model view controller (MVC) is a software design pattern primarily for rapid application development. It divides a given software application into three main interconnected components. The model manages data, logic and has a direct connection to the database used by the application. A view is the generated html content, namely the controller accepts inputs and converts it to commands. Controller is the only point of connection for the model and the view thereby providing a multi tiered approach to software development. Laravel is a free, open source PHP web application framework, designed for the development of model-view-controller web applications.

Laravel framework poses the following server requirements:

**PHP >= 5.6.4**

* OpenSSL PHP Extension
* PDO PHP Extension
* Mbstring PHP Extension
* Tokenizer PHP Extension
* XML PHP Extension

**6. Laravel and screenshots:**

**6.1 Installing Laravel:**

Laravel [5] can be seamlessly installed using the composer command.

composer global require “laravel/installer”

**Description of the application:**

The application “Diet Share” is a platform for posting and sharing customised diet plans. The software stack of the application comprises of,

* PHP (Laravel 5.0 and above)
* MySQL (Version 15.1 and above
* Semantic-UI front end framework

**Database:**

There is a table ds\_plan that contains all the data plans that are posted to the database.

|  |  |
| --- | --- |
| **Field** | **Type** |
| id | Primary Key |
| planname | Varchar |
| Item1 | Varchar |
| quantity1 | Varchar |
| Item2 | Varchar |
| quantity2 | Varchar |
| Item3 | Varchar |
| quantity3 | Varchar |

**Models:**

ds\_plan:

This model serves as the primary gateway for common Create, Read, Update and Delete operations throughout the scope of the Application.

**Views:**

welcome.blade:

This view renders the html content of the application. On load, the controller returning the view fetches all the required data and serves into the view. Laravel comes with blade, a powerful templating engine which makes it possible for integration of control structures and looping logic over view variables.

**Controllers:**

PlanController:

This controller comprises of three methods that are modelled for fetching existing plans from data base, post a new plan to the database and delete selected plans.

**Routes:**

The following are the available url routes and mapped controllers for relevant logic execution.

|  |  |  |  |
| --- | --- | --- | --- |
| **Route Name** | **Http Method** | **Controller** | **Controller Method** |
| / | GET | PlanController | getPlans |
| /postplan | POST | PlanController | addPlan |
| /delete/{plan\_id} | POST | PlanController | deletePlan($plan\_id) |

Basic CLI commands for performing application level tasks:

Laravel ships with a CLI tool artisan, which makes its easy from creating migrations to serving apps a straightforward process.

The following table comprises of the most commonly used commands:

|  |  |
| --- | --- |
| **Command** | **Action** |
| php artisan make:migration <migration name> | Creates a database migration |
| php artisan migrate | Migrates the existing migrations |
| php artisan make:controller <controller name> | Creates a controller |
| php artisan make:model <model name> | Creates a database model |
| php artisan key:generate | Generates a private key for application level encryption |
| php artisan db:seed | Seeds the database with values |
| php artisan serve | Serves the application |

**6.2 Screenshots**

Snapshots for the application is attached below in Fig 1 :The main landing page of the application is as shown below. It consists of all the mean plans that are created and is pulled from the database. It also has an option to share it or delete the meal plan.

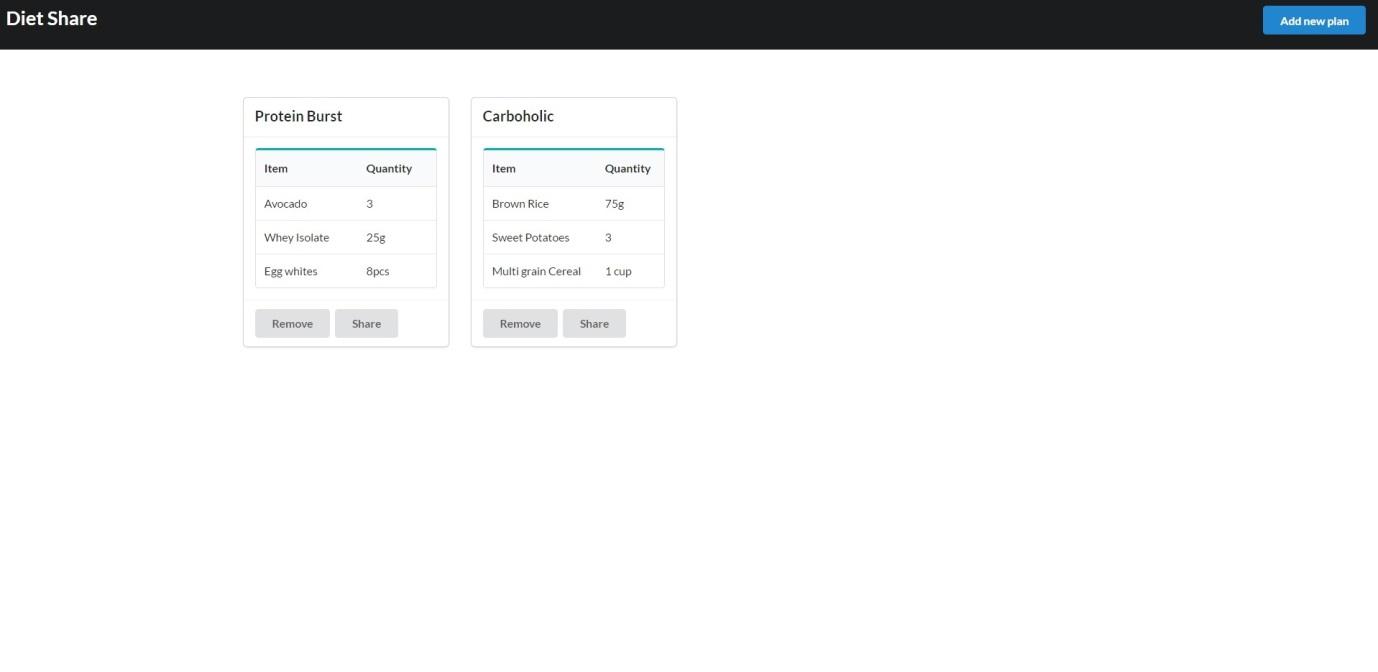


Fig 1 : Application landing page.

The screenshot below in Fig 2 shows the application interface when “Add a new plan” is clicked on. There are input fields provided on this page for Plan name, provisions to add three items with their corresponding quantities. Clicking on Looks Good adds the entire new plan with the items and quantities gets added to the database and is displayed on the main page at the start of the application

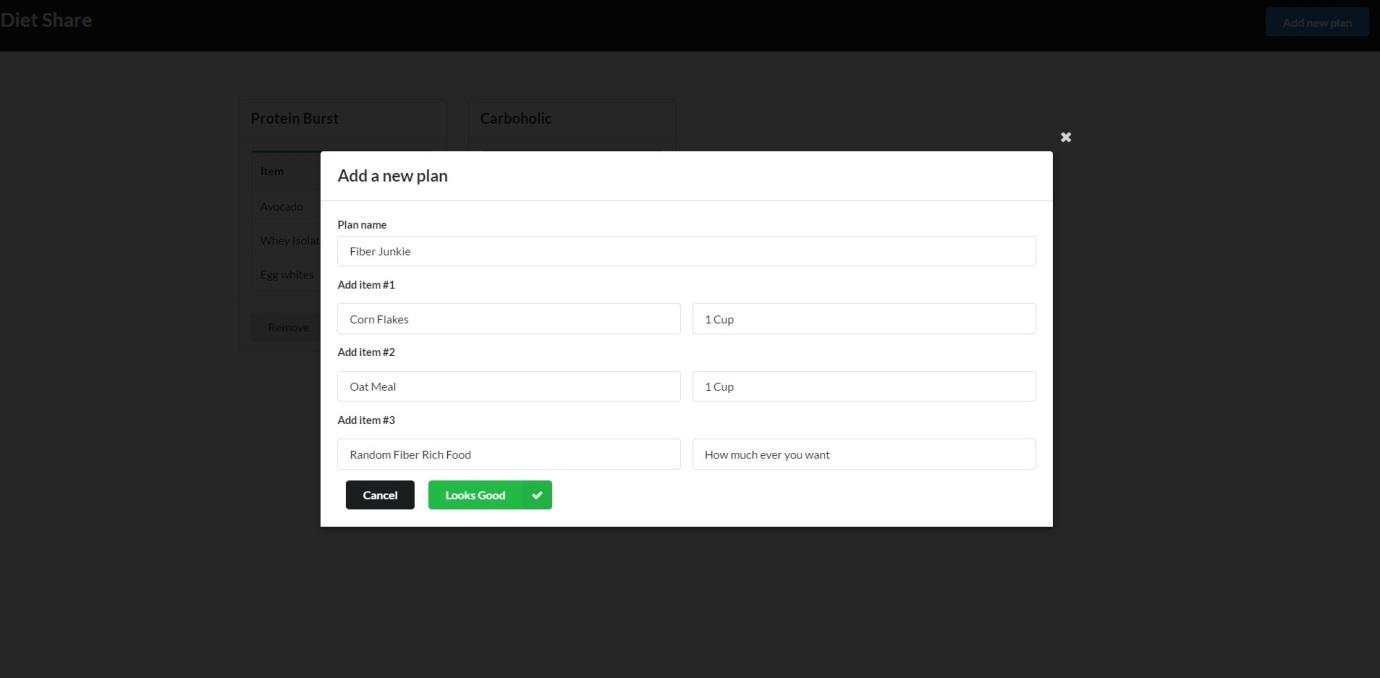


Fig 2 : Add a new plan page

The screenshot below in Fig 3 is the confirmation message screen which is displayed after the new plan is fed into the database successfully.

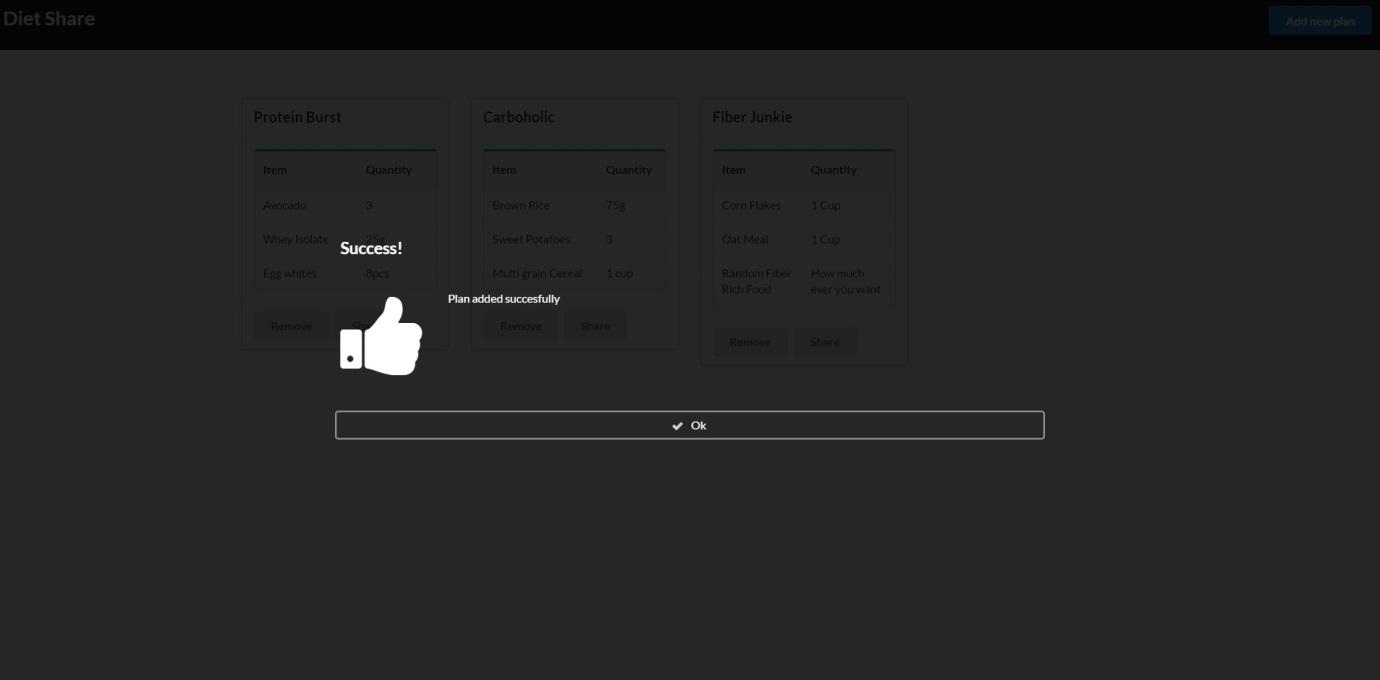


Fig 3 : Plan added confirmation page

The screenshot below in Fig 4 is the confirmation message screen which is displayed after an existing plan is deleted from the database successfully.

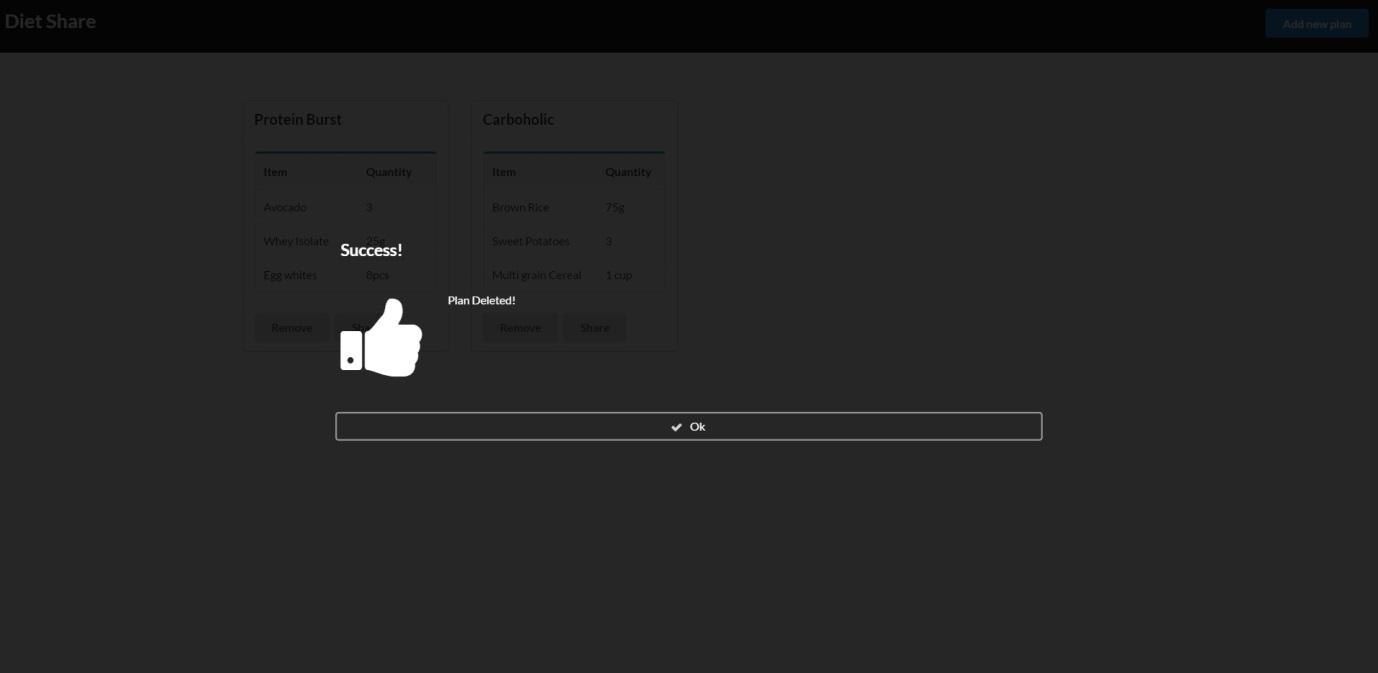


Fig 4 : Plan deleted confirmation page.

The screenshot below namely, Fig 5 shows the display screen once the share button is clicked which is present under each meal plan. It consists of social media icons namely facebook, twitter and google+.

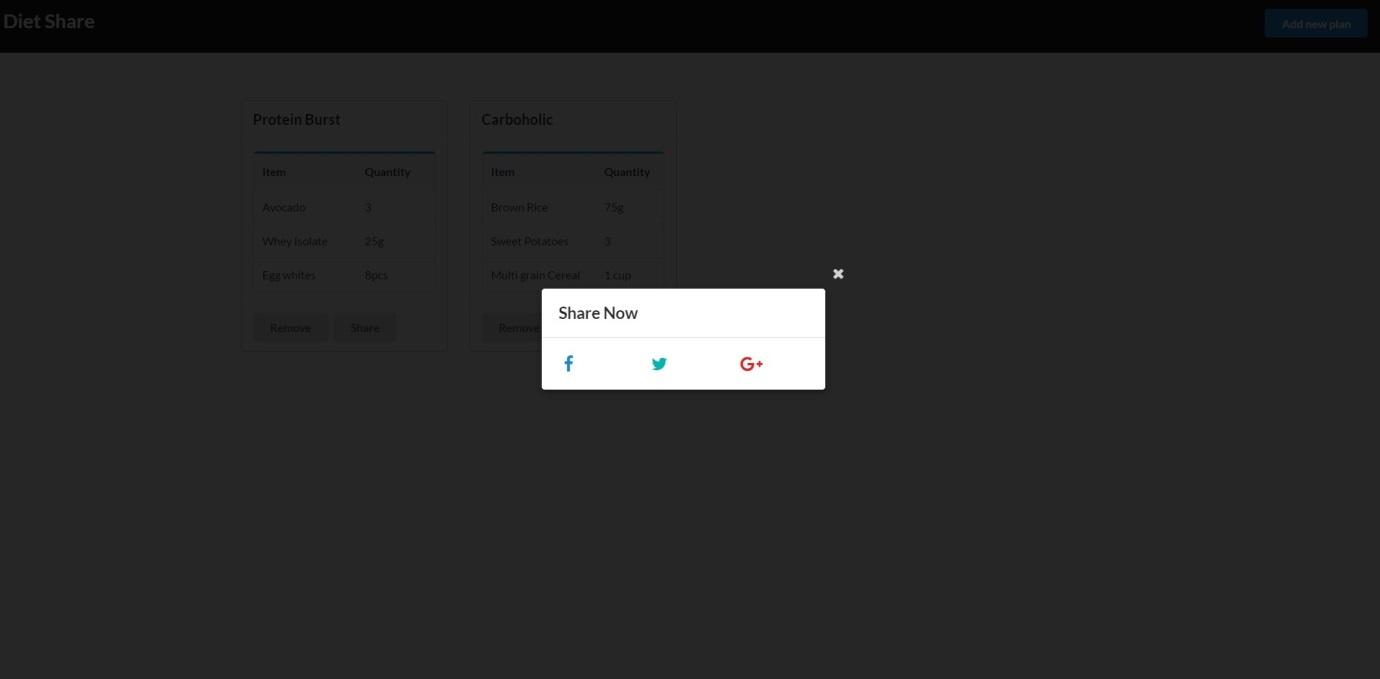


Fig 5 : Diet Share page

**7. Application deployment on Cloud Foundry:**

Cloud Platforms allows to deploy network applications and make them available quickly to the world. Unfortunately, not all cloud platforms are similar, some lack key app services, have limited language and framework support or restrict deployment to a single cloud. To address all these limitations a versatile open source platform - Cloud Foundry is used. The platform’s flexibility and extensibility renders its users enjoying a wide array of frameworks and languages to choose from and not being locked into a single cloud.

[6]Cloud Foundry is the one step solution for anybody interested in removing the cost and complexity of configuring infrastructure for their apps. The best feature of Cloud foundry is that it allows the developers to deploy their apps with zero modification to their code.

Below are the steps and the procedure undertaken to deploy the App created in the Step 5 on Laravel framework - “Diet Management system” in Cloud Foundry. But before starting to deploy the app in the cloud foundry there are some prerequisites required to be installed in the system:

**tep1** : Install the Cloud Foundry Command Line Interface - (cf CLI) from the Github repository by downloading the zip file for windows cli-installer as shown below in fig 6.

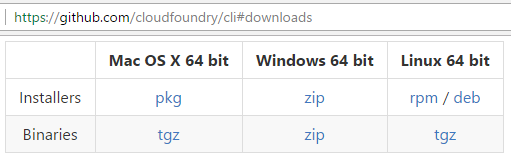


Fig 6 : Github repository page

**Step2** : After the download is complete, Unzip the installers folder and and run the cf\_installer.exe file as shown below in fig 7 , fig 8 and fig 9 :

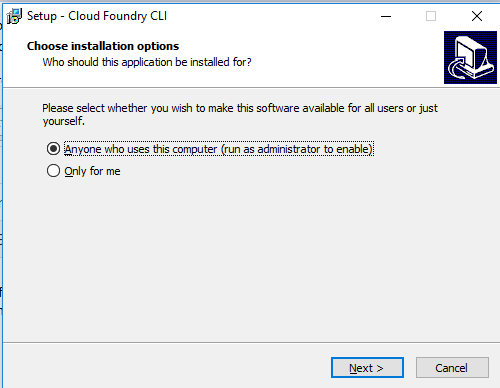


Fig 7 : Cloud Foundry setup page

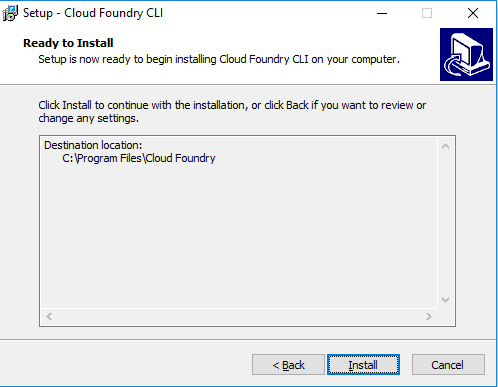


Fig 8 : Cloud Foundry Installer

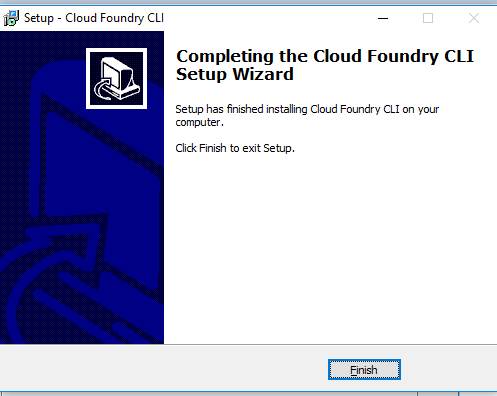


Fig 9 : Cloud Foundry Final setup Wizard

**Step3** : To Deploy the app in the cloud, PCF (Pivotal Cloud Foundry) is used which basically provides developers a production-ready application container runtime and fully automated service deployments. For this create an account in Pivotal Web Services (PWS) and login to go to the PWS as shown below in Fig 10 :

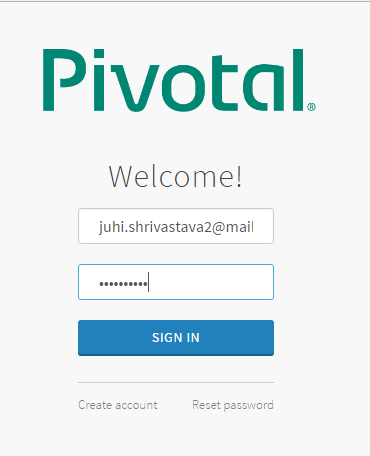
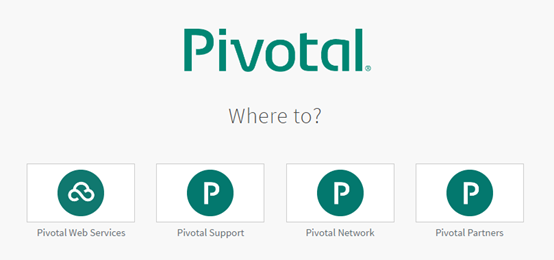
 

Fig 10 : PWS login (left) and start page (right)

**Step 4** : To organize user access to the cloud and to control resource use, a cloud operator defines Organizations and Spaces within an installation as shown below in fig 11 and assigns Roles such as admin, developer, or auditor to each user.

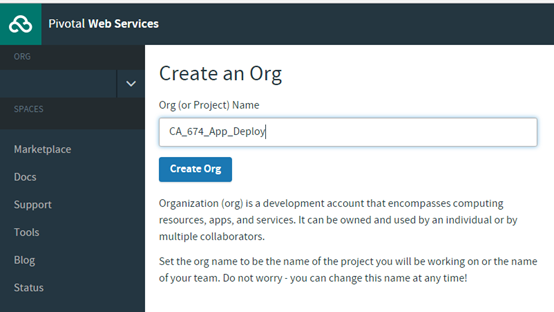
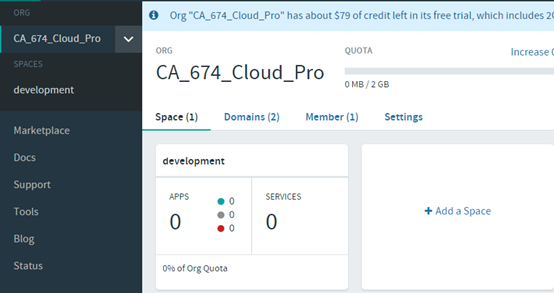
 

Fig 11 : Role assigning and organization page

**Step 5** : Now to verify the installation of the cf CLI, open a terminal window as shown below in Fig 12 and type cf. If your installation was successful, the cf CLI help listing appears.

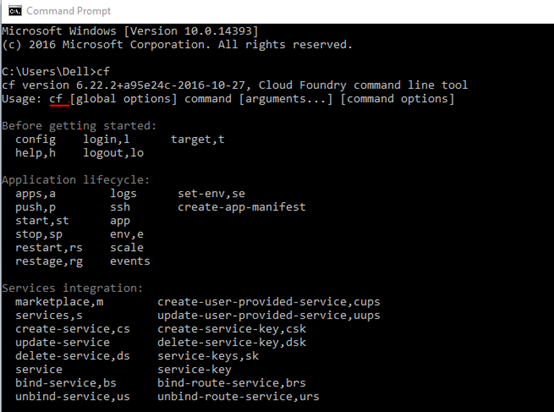


Fig 12 : Installation verification of CF CLI

**Step 6** : Now using the cf CLI login into the PWS using the same credentials which is used to create the account in PCF above. After Authentication the command prompt lists your organisations and prompts you to select one. The entire process is shown below in Fig 13

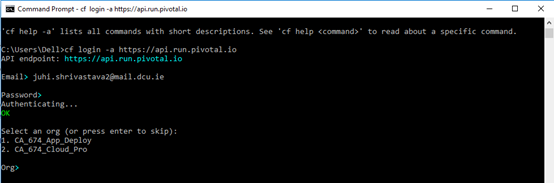


Fig 13 : CLI login into PWS and selection of organisation

**Step 7** : After the Organization is selected with the numeric response 1 or 2 , space is also shown by-default if there is only one space available, in our case its Development as shown below in Fig 14

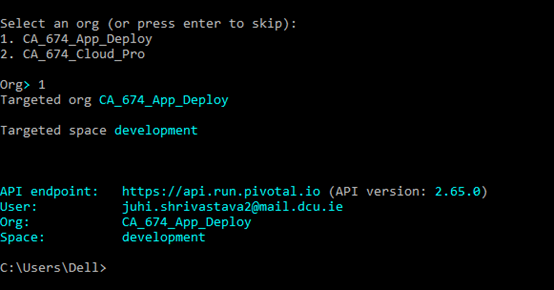


Fig 14 : Targeted space selection

**Step 8** : Get into the app folder on the system which contains all the files of the app created through Laravel and the manifest file created manually as shown below in fig 15..

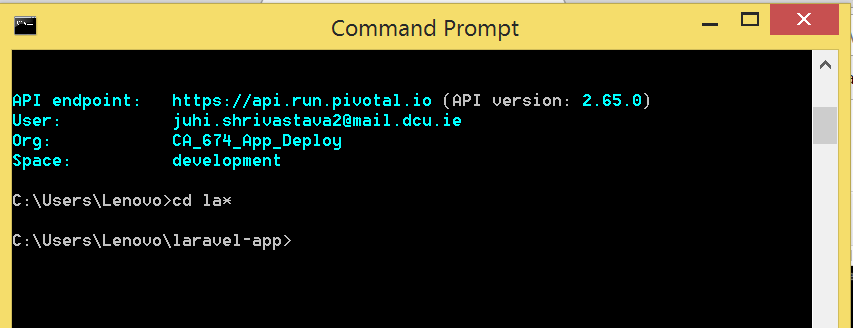


Fig 15 : Change directory to the app folder on the system

**Step 9** : Deploy the app into the PWS using command ‘cf push’ from the Command Line Interface (cf CLI). [7]Between the time that the cf push is run and the time that the app is available, Cloud Foundry performs the following tasks which is shown below in Fig 16:

* App files are uploaded and stored.
* App metadata is examined and stored
* A “droplet” (the Cloud Foundry unit of execution) for the app is created
* To run the droplet an appropriate Diego cell is selected
* App starts

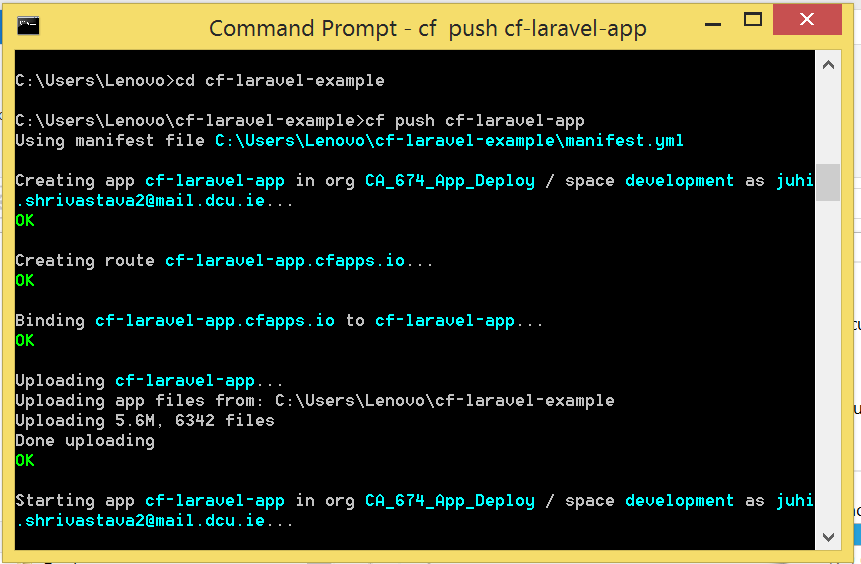


Fig 16 : App Deployment on CF and the tasks that follow

**Step10 :** Once the app is pushed into the cloud, The cf CLI will install all the Buildpacks required by itself and will create a container as shown below in Fig 17.

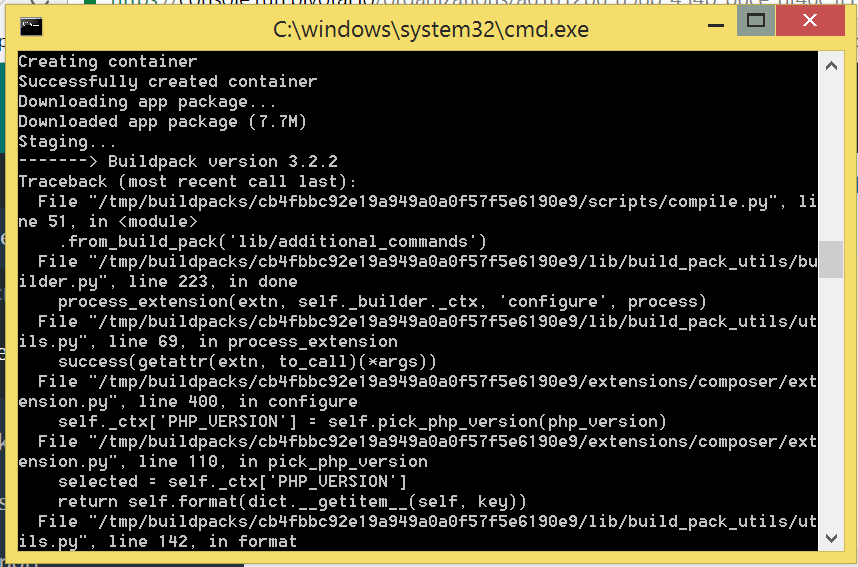


Fig 17 : Buildpacks followed by the Container creation

**Step11:** Once the buildpacks are installed the URL for the app is provided by Cf CLI as shown in fig 18:

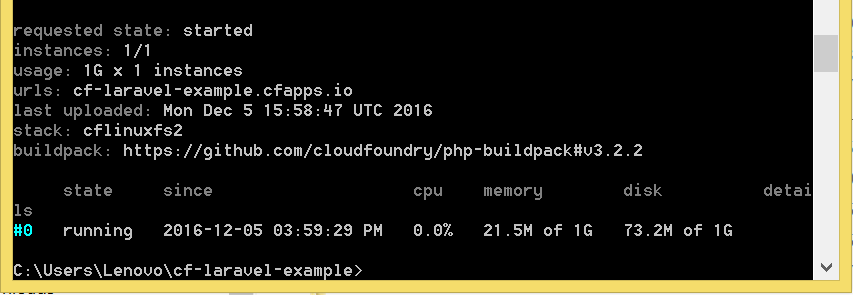


Fig 18 : URL for the app on CF CLI

**Step12:** Now the PWS will show the status of the App as running with the URL generated in the step above, as shown below in Fig 19 . This url will be used to access the application on the cloud :

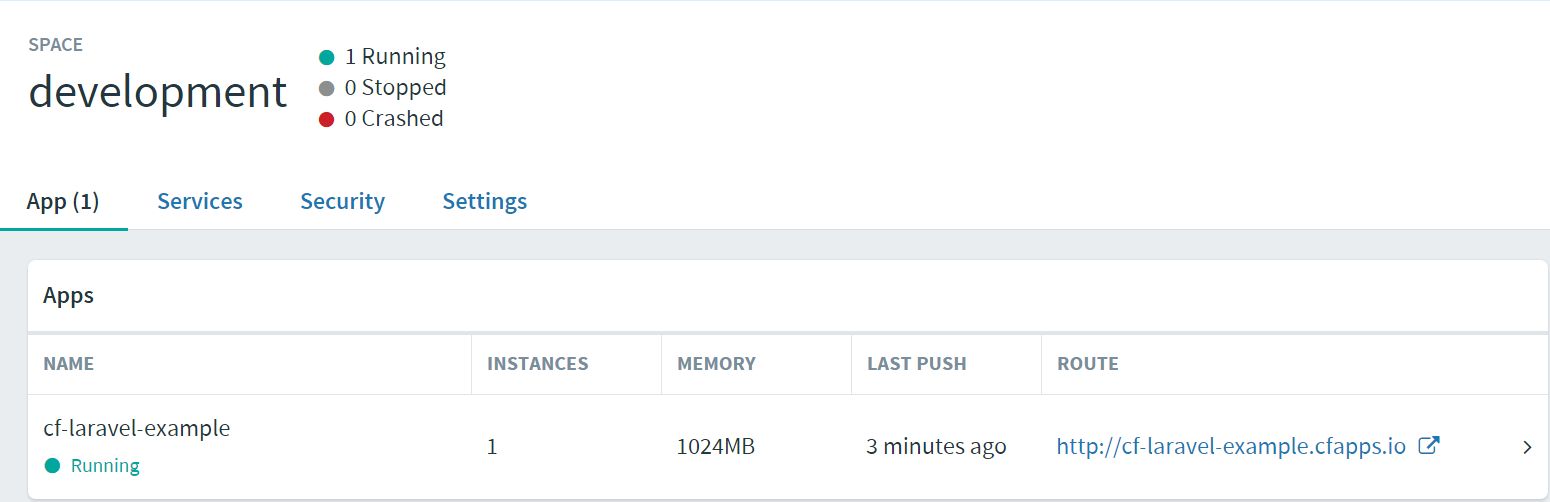


Fig 19 : PWS showing status of the App with running URL

**Step13:** The App created - Diet Share has now been successfully deployed as shown below in Fig 20

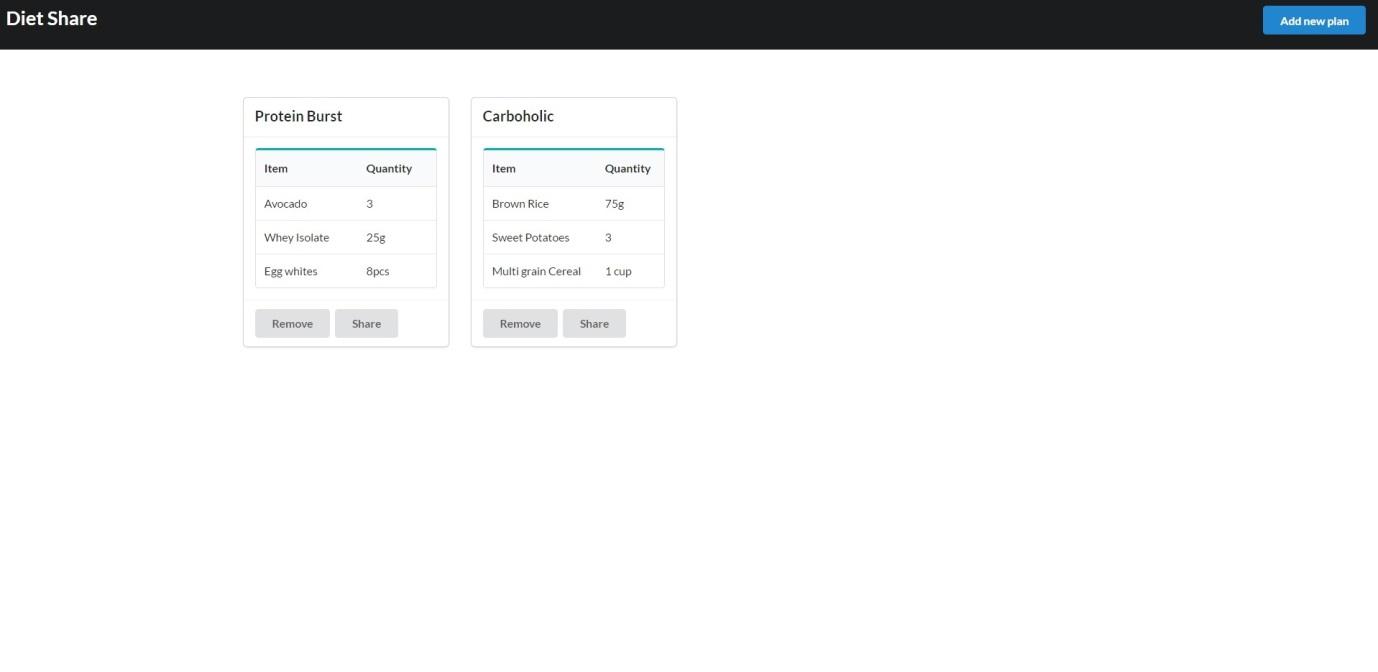


Fig 20 : App Deployed

**8. Challenges while deploying the Application:**

Some of the challenges encountered while deploying an app in Cloud Foundry are documented below for future references:

* In the option.json file created at the time of creation of the app in the Laravel framework changes were required to be done as cf Cli was unable to clone the Buildpacks from the GitHub repository. The new option.json file will have the following structure:

{

"PHP\_VERSION": "{PHP\_70\_LATEST}",

"WEBDIR": "public", "LIBDIR": "vendor", "COMPOSER\_VENDOR\_DIR": "vendor",

"PHP\_EXTENSIONS": [ "bz2", "zlib", "openssl", "fpm", "tokenizer", "curl",

"mcrypt", "mbstring", "pdo", "pdo\_mysql"]

}

* The app was unable to start after the latest buildpack updates and the cf push command kept failing. To address this issue in the cf CLI the app was pushed into the cloud with the following command which depreciated the buildpack version:

cf push -b https://github.com/cloudfoundry/php-buildpack.git#v3.2.2 “app name”

**9. Cf essential commands:**

Some Essential commands while deploying the app into the Cloud foundry are:

* To login into the PWS:

*cf login -a* [*https://api.run.pivotal.io*](https://api.run.pivotal.io)

* Push the app to PWS:

*cf push “app name”*

* View a snapshot of recent logs:

*cf logs “app name” --recent*

* Or, stream live logs:

*cf logs “app name”*

* Restart the app

*cf restart “app name”*

* Increase the number of app instances from one to two:

*cf scale “app name” -i 2*

* Check status of the app

*cf app “app name”*

* Increase the memory limit for each app instance:

*cf scale “app name” -m 1G*

* Increase the disk limit for each app instance

*cf scale “app name” -k 512M*

* Help command listing all other commands

*cf help*

* Check the buildpacks and their priority while staging of the app

*cf buildpacks*

**10. Application deployment on Openshift**

Openshift online 3 [8] is Red Hat’s application hosting platform which makes it convenient for developers to quickly build, launch and scale container-based web apps on a public cloud environment. Quickstarts is a basic example of any application that is executed on openshift online and it comes on a variety of languages and frameworks and are defined as a template which is built from a set of services, build and deployment configurations. The template can be customized by forking the repository and during the creation of an application from the template , the name of the forked repository should be substituted with the name of the default source repository. This results in the execution of the source code where code updation in the source repository can be done successfully followed by the launch of a new build to notice the changes reflected in the deployment.

[Laravel](http://laravel.com/) is a free, open source PHP web application framework, designed for the development of model-view-controller web applications. The laravel quickstart template facilitates the user with separate .env configuration file for local and remote access which is available at .env and .openshift/env respectively. MySQL is the open source database that was utilized in the setup.

Shown below in Fig 21 is the snapshot of the application page on Openshift online. Since there has been only one app deployed on the platform via the login, only that is seen.

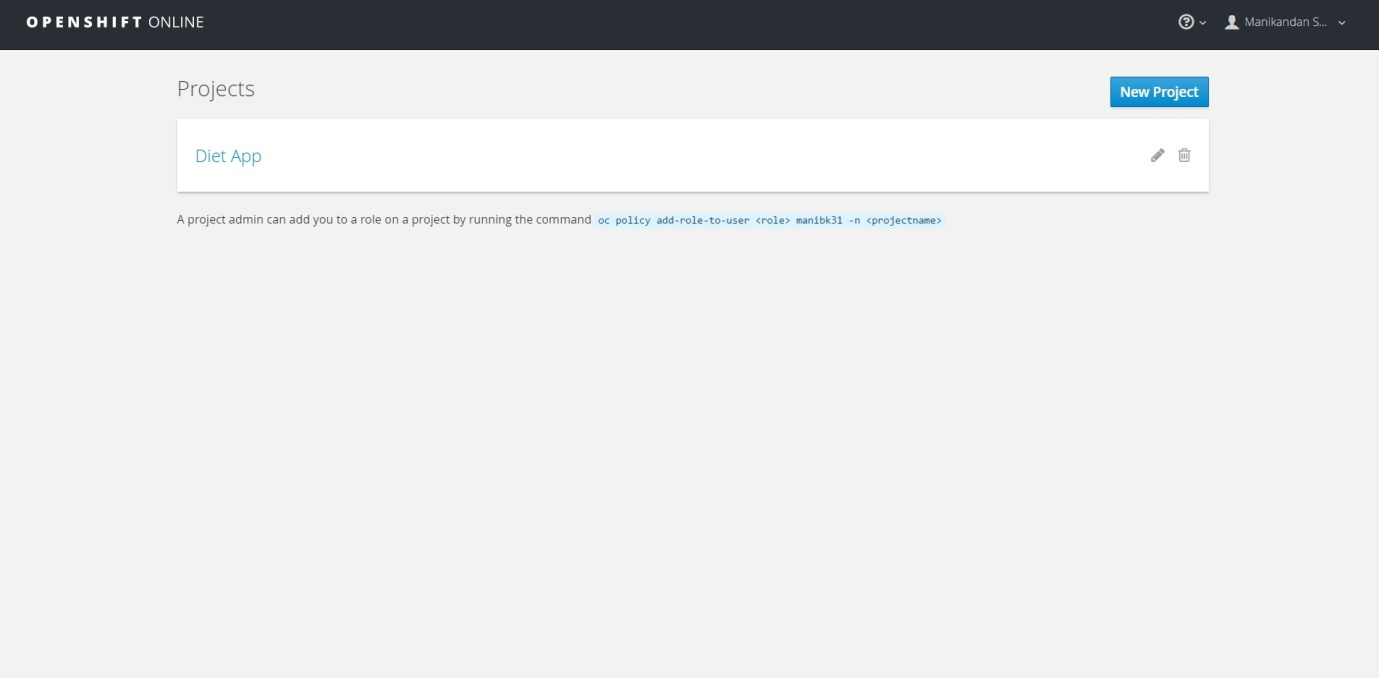


Fig 21 : Application page on Openshift

The snapshot shown below in Fig 22 displays the builds of the application. In software context, build refers to a version of a program. On this page, we can check the summary, configuration and environment of the application It also allows us to be aware of the number of builds we have initiated and the status of it which includes the time taken by it and when it was done.

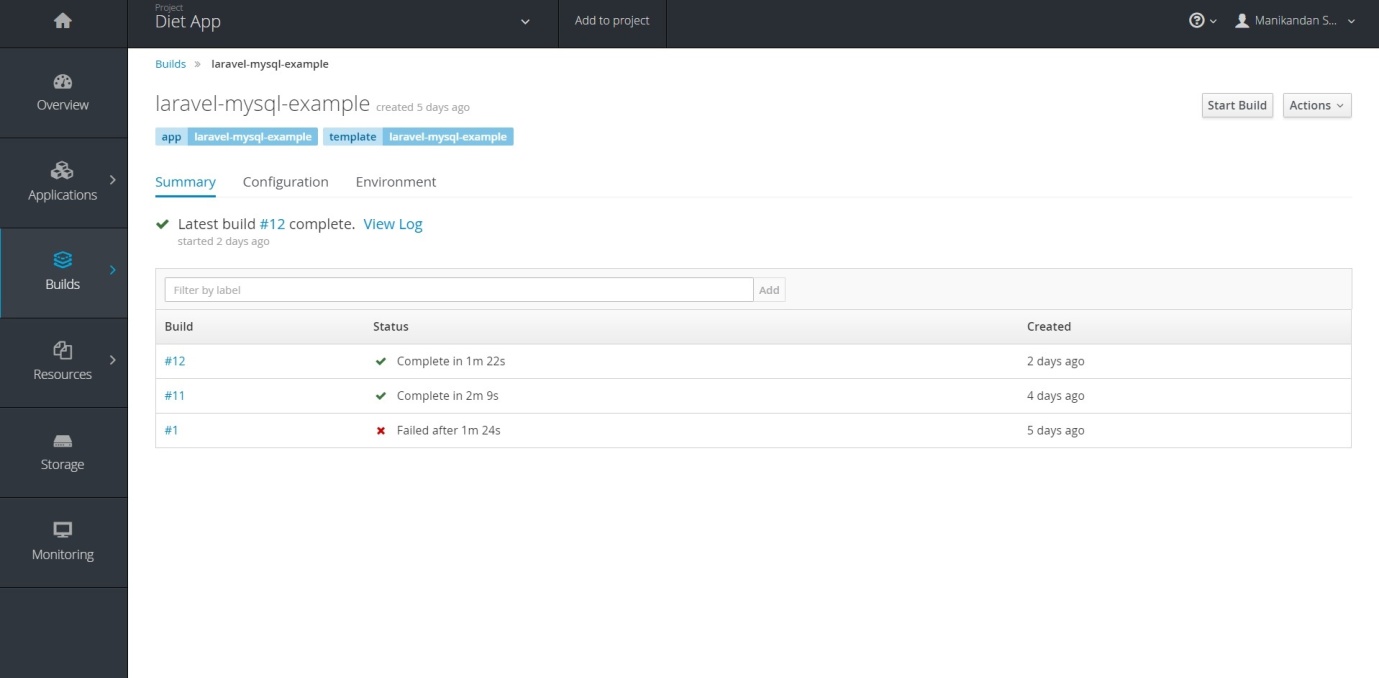


Fig 22 : Build, configuration and environment of the application

A snapshot of the build process underway is attached below in Fig 23. The process includes pulling in details from the composer file, libraries for the framework and finally pushing the application on to the platform.

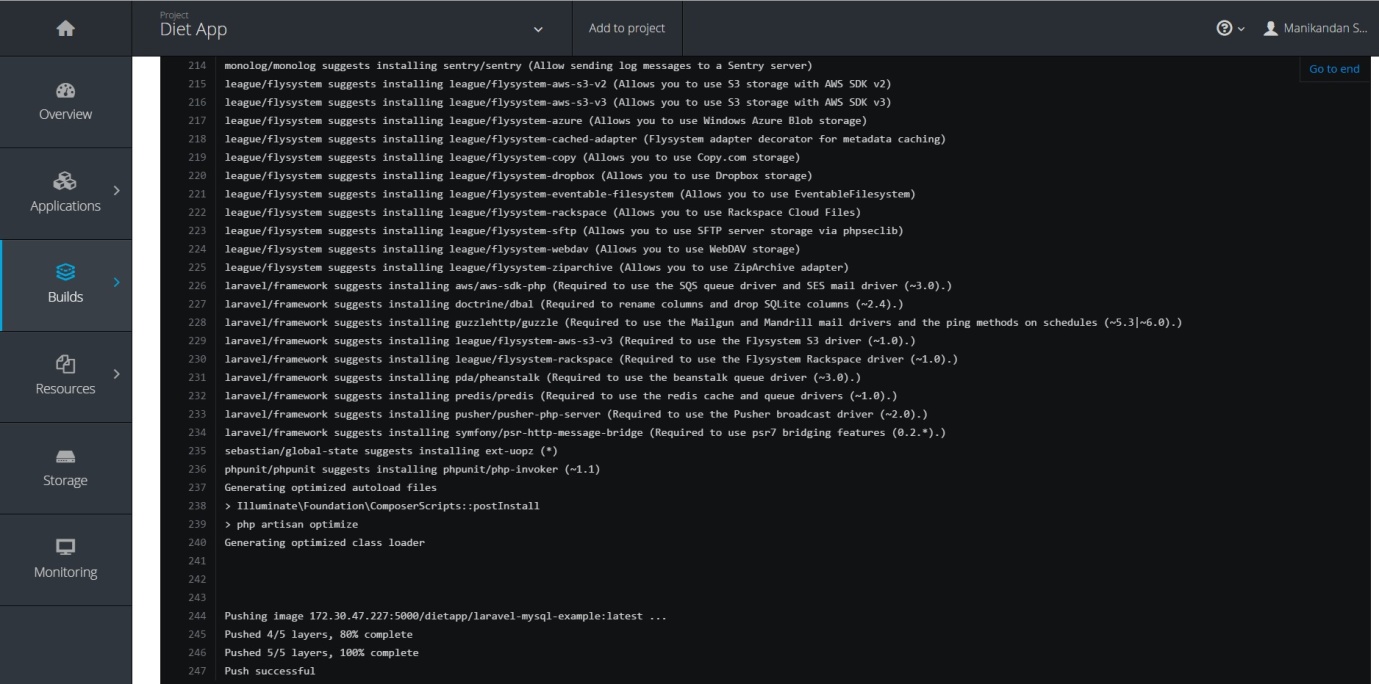


Fig 23 : Build process underway

The dashboard of the diet application which is deployed on openshift online is shown below in Fig 24. As the name indicates, dashboard consists of an overview of the applications deployed on the platform. This page also includes information on the image and port used.

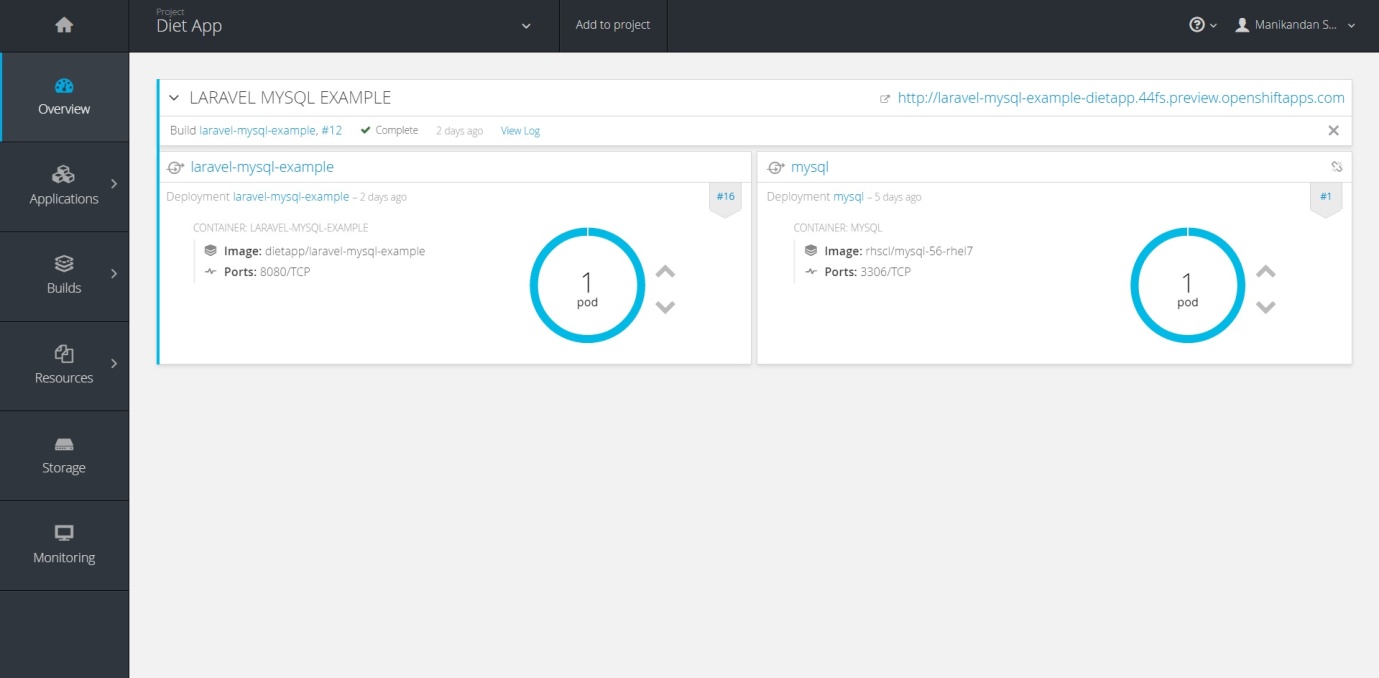


Fig 24 : Openshift Dashboard

Deployments of the application is shown below in Fig 25 as it is seen on the openshift online software. It displays all the names of the applications, when the build was last deployed , status of the application , when its build was initiated and what the trigger was.

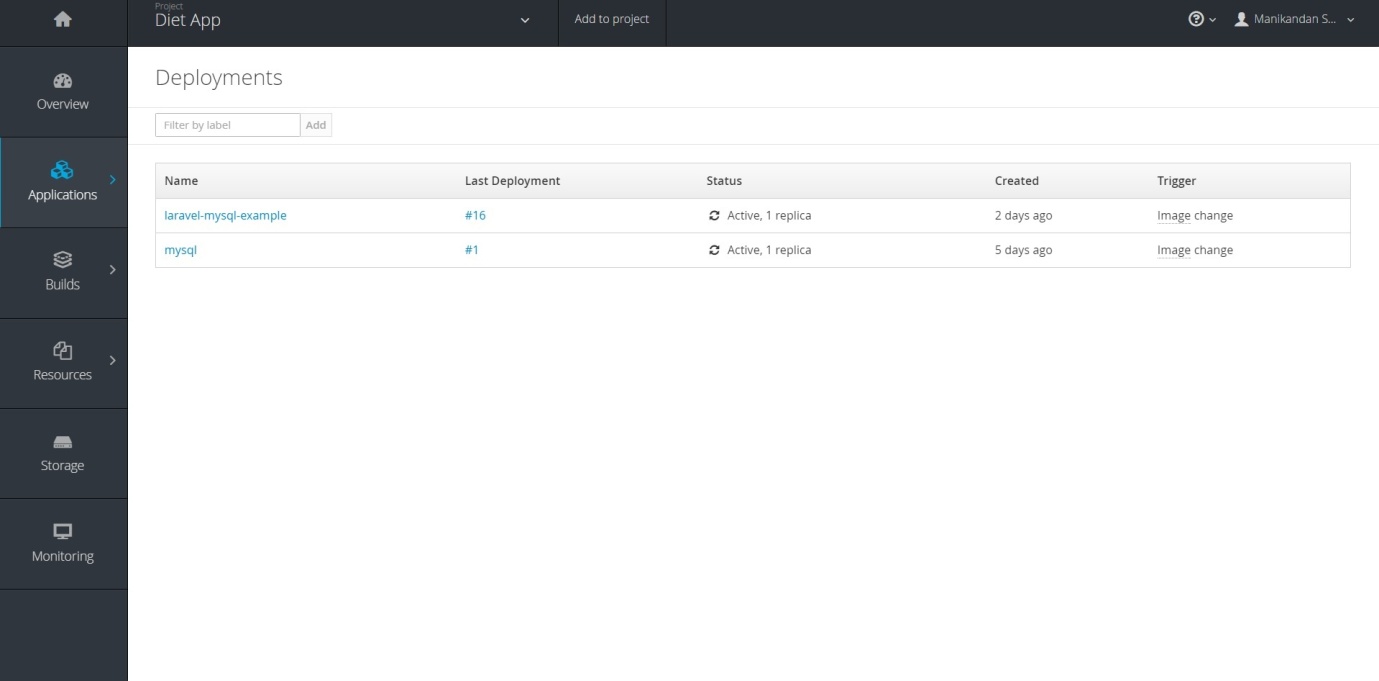


Fig 25 : Application Deployment

The templates available is listed below in the snapshot as shown in Fig 26. These template are the pre-written quickstart files for various languages as seen. Using these quickstarts makes the whole process less tedious.

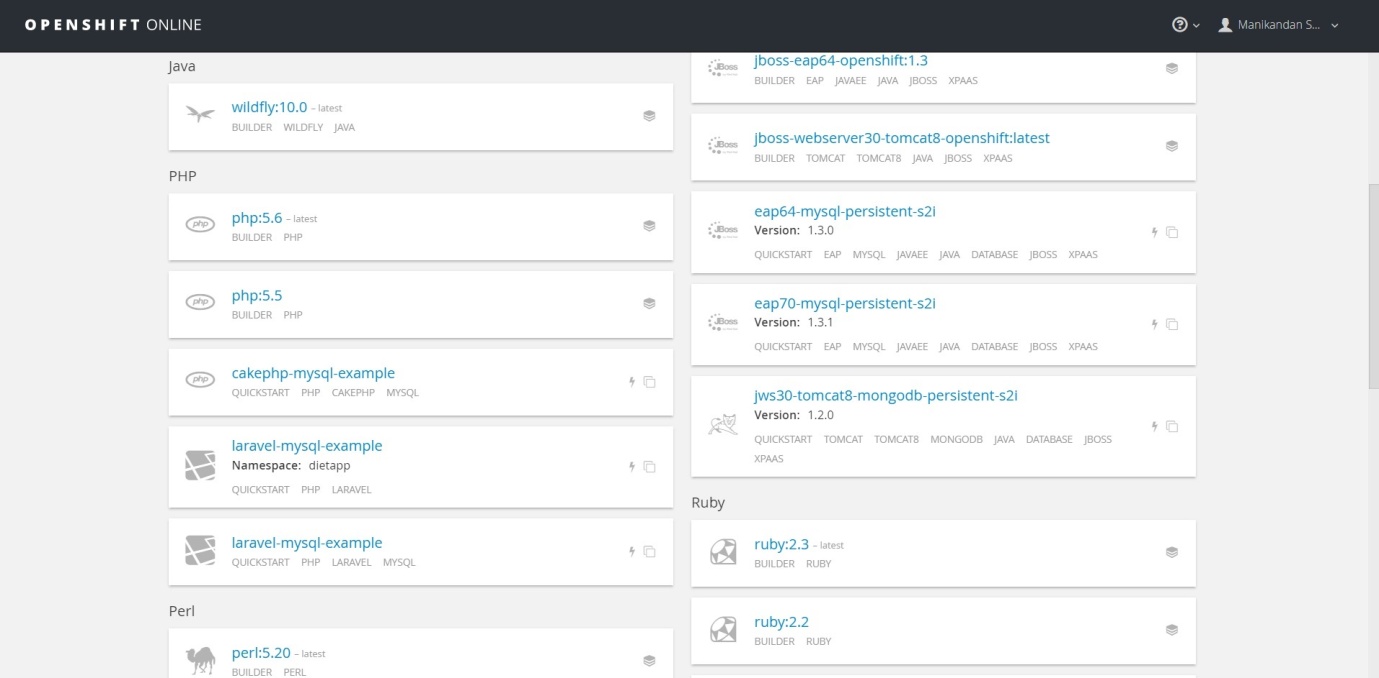


Fig 26 : Openshift Quickstart Templates

**11. Challenges while deploying the Application:**

* Frequent build process failure and lack of efficient debugging tools posed a challenge to make the app deploy ready
* Making the open source laravel template suitable for the developed application required a deeper study of the OpenShift online api’s and services

**12. Openshift essential commands:**

* Rhc app\_show -v
* Rhc app show –gears quota
* Rhc app show – gears ssh
* Rhc app-start
* Rhc app-stop
* Rhc app-force-stop
* Rhc app-restart
* Rhc app-tidy
* Rhc-tail

**13. Appscale**

AppScale is an open-source cloud computing platform that deploys and scales unchanged Google App Engine applications over public and private cloud systems and on-premise clusters. AppScale is created on the App Engine APIs and can support Python, Go, PHP and Java applications. AppScale is developed and managed by AppScale Systems, Inc., based in Santa Barbara, California, and Google.

The aim of AppScale is to equip developers with a speedy, API-driven development platform that can run applications on any cloud infrastructure. AppScale decouples app logic from its service ecosystem to give developers and cloud administrators power over app deployment, data storage, resource use, backup, and migration. It performs automatically over public clouds such as Google Compute Engine, Microsoft Azure, Alibaba Cloud, Amazon EC2, IBM SoftLayer, Rackspace, DigitalOcean, CloudStack, OpenStack, Eucalyptus, as well as KVM, Xen, VirtualBox, and VMWare. AppScale uses the standard three-tier web deployment model. The multi-tiered AppScale plan consists of a load balancer, multiple application servers, several services to maintain the different APIs, and a data store layer for persistent storage of cloud-wide data and state.

**13.1 INSTALLATION AND DEPLOYMENT**

* Making an AppScalefile : A configuration file is used in Appscale called AppScalefile [9]. This helps to specify how many machines need to be run over, to set up API services, and so on. To build an AppScalefile, run "appscale init cloud" to deploy over Amazon EC2 or Eucalyptus, or "appscale init cluster" for virtualized cluster. An AppScalefile is formed in current working directory, which can be edited to change how AppScale is deployed.

$ appscale init cluster

AppScalefile successfully created and can be customize for a particular cloud or cluster.

* Starting AppScale : Once AppScalefile is set up, a new AppScale deployment is started by running "appscale up"

$ appscale up

Starting AppScale 2.2.0 over a virtualized cluster.

Log in to your head node: ssh -i /Users/you/.appscale/appscale.key root@192.168.10.2

Copying over local copy of AppScale from ~/appscale

Head node successfully initialized at 192.168.10.2. It is now starting up cassandra.

Copying over deployment credentials

Starting AppController at 192.168.10.2

Please wait for the AppController to finish pre-processing tasks.

Please wait for AppScale to prepare your machines for use.

AppController just started

UserAppServer is at 192.168.10.2

Enter your desired admin e-mail address: test@example.com

Enter new password:

Confirm password:

Creating new user account test@example.com

Creating new user account test@192.168.10.2

Your XMPP username is test@192.168.10.2

Granting admin privileges to test@example.com

AppScale successfully started!

View status information about your AppScale deployment at http://192.168.10.2/status

* Deploying Apps to AppScale : Once AppScale is started, Google App Engine applications can be deployed to it. It should be ensured that the app has to exist at the path that is pointing to "deploy" at.

$ appscale deploy ~/sample-apps/python/guestbook

Enter your desired e-mail address: test@example.com

Uploading guestbook

We have reserved guestbook for your app

Creating remote directory to copy app into

Tarring application

Copying over application

Please wait for your app to start serving.

Your app can be reached at the following URL: <http://192.168.10.2:8080>

* Change App Host Port : Designated ports can be specified for the app. The application ID, the HTTP port, and the HTTPS port can be given.[10]
* Updating your Application : The same steps are followed as the initial deployment. This will immediately start serving the newer version of the app.

$ appscale deploy ~/baz

* Change the application port : Following example shows to run the application on port 80/443

$appscale relocate <app\_id> 80 443

* Seeing How AppScale is Doing : AppScale Tools can be used to query the status of AppScale deployment, by running "appscale status". This output is similar to the following:

$ appscale status

Status of node at 192.168.10.2:

Currently using 6.5 Percent CPU and 66.33 Percent Memory

Hard disk is 7 Percent full

Is currently: load\_balancer, shadow, db\_master, zookeeper, login, memcache, taskqueue\_master, appengine

Database is at 192.168.10.2

Is in cloud: cloud1

Current State: Preparing to run AppEngine apps if needed

Hosting the following apps: none

View status information about your AppScale deployment at <http://192.168.10.2/status>

**13.2 Limitations of using Appscale:**

* The maximum size of Blobstore 100MB. This value is configurable in the code and will support for the setting of this value in a configuration of the file.
* Datastore: AppScale does filtering of queries in-memory rather than index data. With the size of the Database getting larger, users may observe a decrease in performance for queries. Currently, work on indexing system is going on which can create indexes and still stay obedient with the ACID semantics required for transactions.
* Data persistence across AppScale deployments is also not supported. Rather, an extension is provided to the bulkloader that is part of Google App Engine with which user can download and upload data from their application or an AppScale deployment prior to shutting down and starting. The syntax and semantics of the bulkloader are similar to those of Google's implementation.
* Task Queue: Tasks that are awaiting processing are not fault-tolerant and AppScale does not handle delayed workers. Currently, refactoring the Task Queue support using an open source distributed queuing technology to fix this issue is being developed.
* Mail: Only the administrator is allowed to send mail, and reception of mail is not implemented.
* OAuth API is not implemented in AppScale.
* AppServer components are not elastic. The necessary technology for AppServer elasticity and plan to enable such functionality are currently being investigated.
* Ubuntu Karmic Koala is currently the only distribution and version that support the implementation of AppScale VM images. This is to define testing and allow for accelerated release cycles. AppScale is transferable to other distributions.
* Some datastores have no way of retrieving the complete table to run a query. A set of special keys needs to be used that track all keys in the table. These datastores, therefore, have the added burden of always accessing the special keys. These datastores are SimpleDB, Voldemort, and MemcacheDB.

**Apache Stratos**

Apache Stratos[11] is a Platform-as-a-Service (PaaS) framework that helps run composite applications Apache Tomcat, PHP, and MySQL applications and can be extended to assist many more environments on all major cloud infrastructures. For developers, Stratos gives a cloud-based environment for developing, testing, and running scalable applications. IT providers profit from high utilization automated resource management , rates and platform-wide insight including monitoring and billing. Stratos also supports the use of Docker in a PaaS by combining Kubernetes and CoreOS.

The reason for choosing Stratos is that it is currently incubating at Apache Software Foundation.

* Added significant capability of Apache Stratos is foretelling immediate future load based on current health statistics. All load balancers, cartridges, and health monitors publish health status into CEP (complex event processor) via a real-time event bus.
* Stratos PaaS is easy to get it up and working in less time. A developer can run and test PaaS framework on a single machine to try out.
* The implementation model for new cartridges is well explained and easy to practice.
* The programming model is simple, and the developer does not have to replace anything that he/she would do developing an on-premise application. Just develop the application and deploy onto Stratos, and will have a multi-tenant enabled SaaS application.

Stratos can be extended so that own flavours of PAAS like aPAAS (Application PAAS), iPAAS (Integration PAAS) etc. Stratos provides API’s and extensions. With Stratos it is easy to bring the applications to the cloud. Stratos is a single merchandise with multiple profiles. Leveraging WSO2 carbon multiple profile support. It has three profiles:

* Cloud Controller (cc)
* Stratos Manager (sm)
* Auto Scaler (AS)

**14.1 Installation Prerequisites and Setup**

* An Infrastructure-as-a-service (Iaas) provider
* Java 1.6
* Message broker with AMQP support
* Apache ActiveMQ/WSO2 MB
* MySQL
* Puppet- Puppet is a server automation tool and automated Cartridge Configuration. It is easy to manage several cartridges. All configurations are stored in the puppet master.

**Stratos Manager Console**

After starting all the requirements used by Apache Stratos, Stratos Management Console can be accessed. The login page is shown below in Fig 27

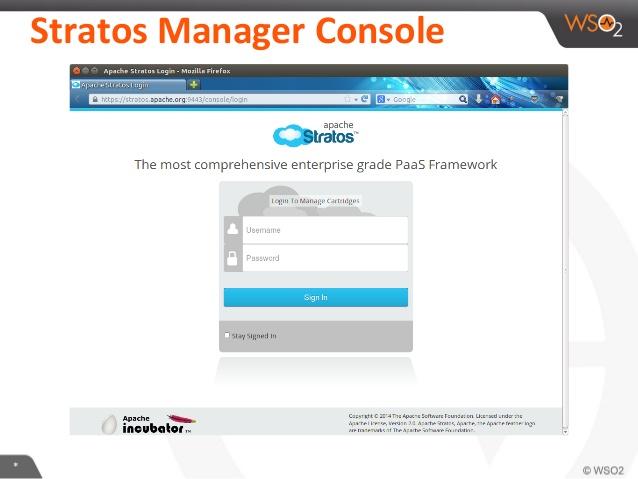


Fig 27 : Stratos console login page

\

**Stratos CLI**

The command line interface (CLI) is a strong tool that clients can use to interact with Stratos services. The CLI configuration only includes one jar file comprising all dependent java libraries. Once logged in as super-admin, it will be redirected to the My Cartridges page of Stratos UI. The page shows the Cartridge subscriptions that are made. Since no subscriptions are done yet, we would see a page like below in Fig 28

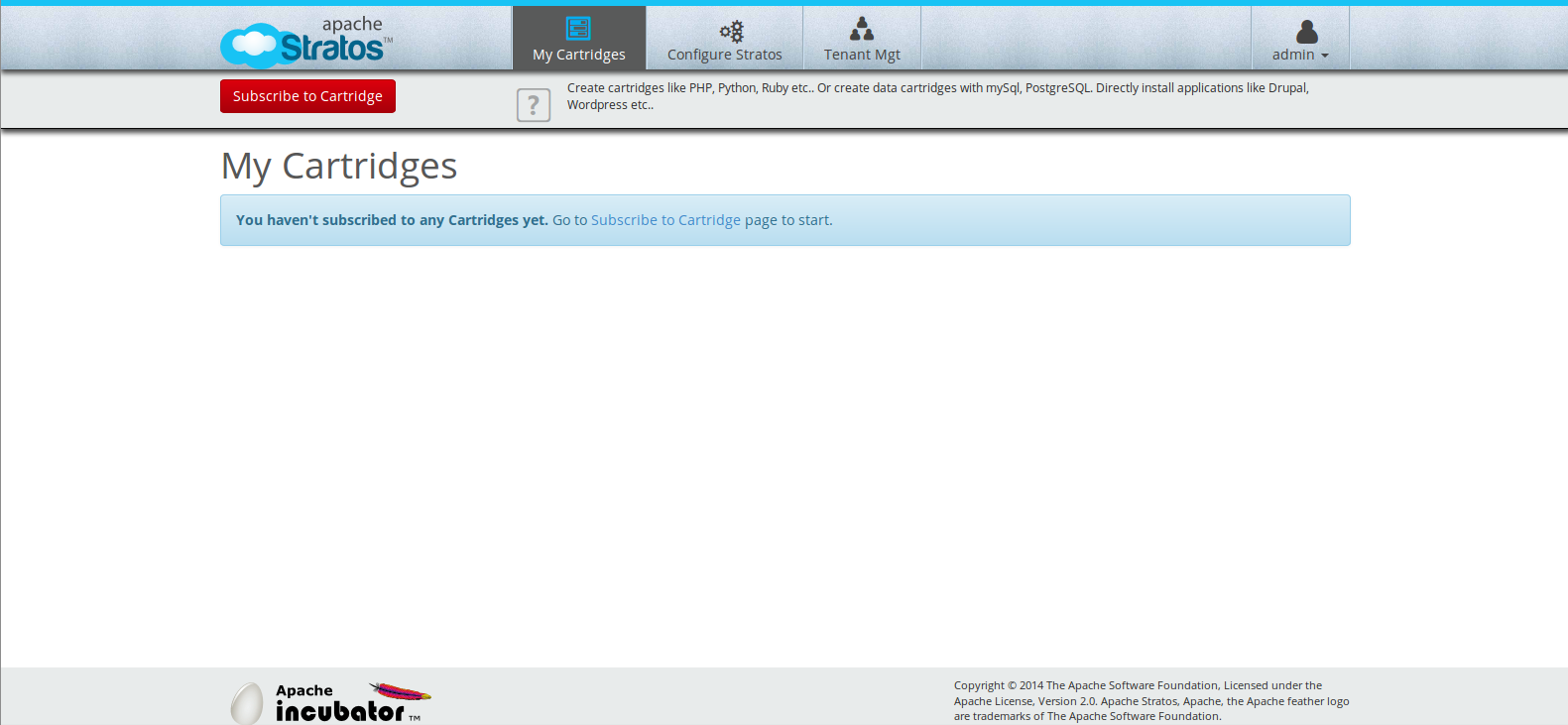


Fig 28 : Stratos Cartridge Subscription

**Configuring Stratos**

Navigate to the 'Configure Stratos' tab. This is the main entry point to configure the Apache Stratos PaaS Framework. The instructions walk through a set of well-defined steps and eventually support to configure Stratos. After clicking on 'Take the configuration wizard' button and the wizard setup begins. The first step of the wizard is the Partition Deployment.

**Partition Algorithms**

The available partition algorithms are as follows and an example is shown below in fig 29:

* One after the other
* Round Robin

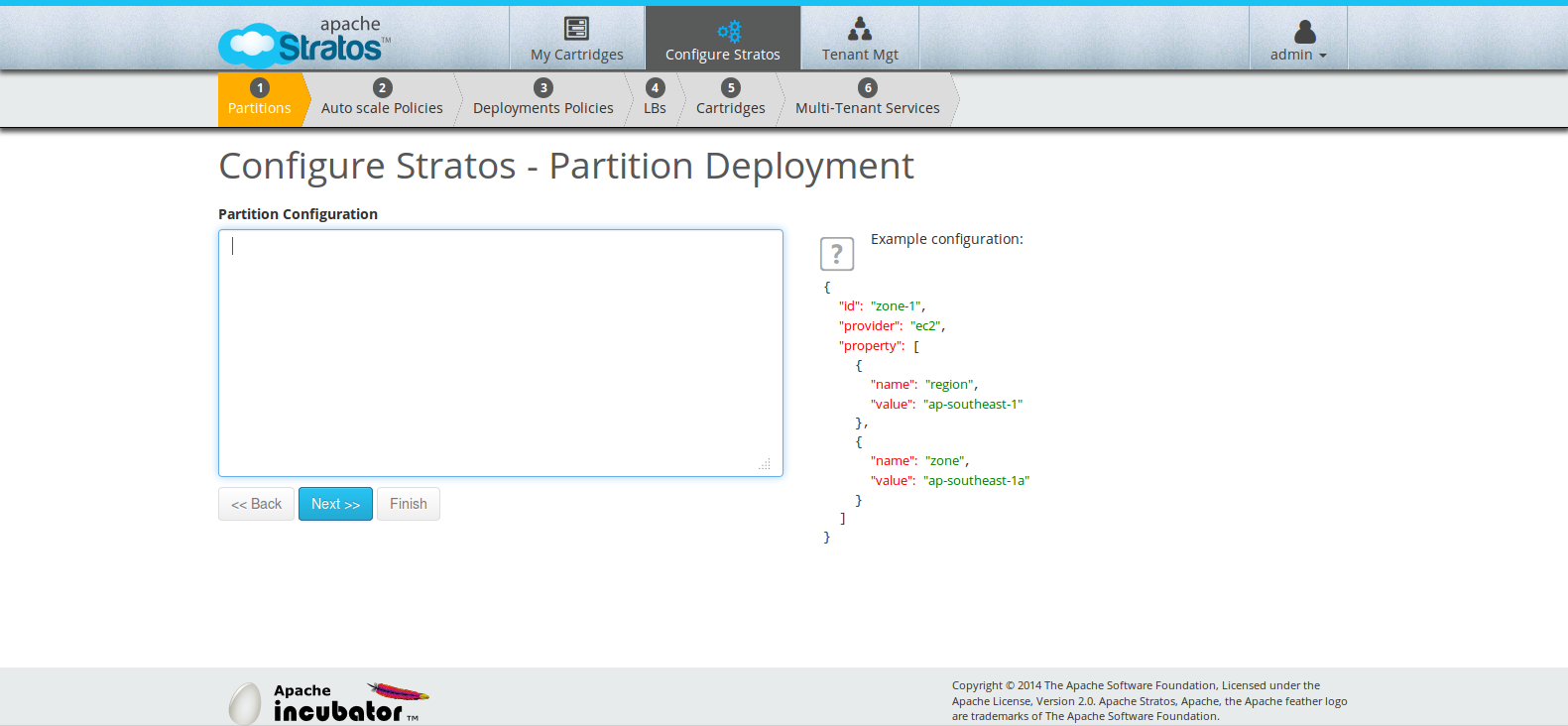


Fig 29 : Example partition algorithm in Stratos

**14.2 Deployment**

The steps after this will walk through and proceed to the next step of the configuration wizard as shown in Fig 30. On clicking 'Next', Stratos will validate the Partition configuration and then deploy it, if it is valid. Also a message is seen on top in yellow back-ground if it is successful and in case, the Partition is not valid, an error message is seen in a red back-ground.

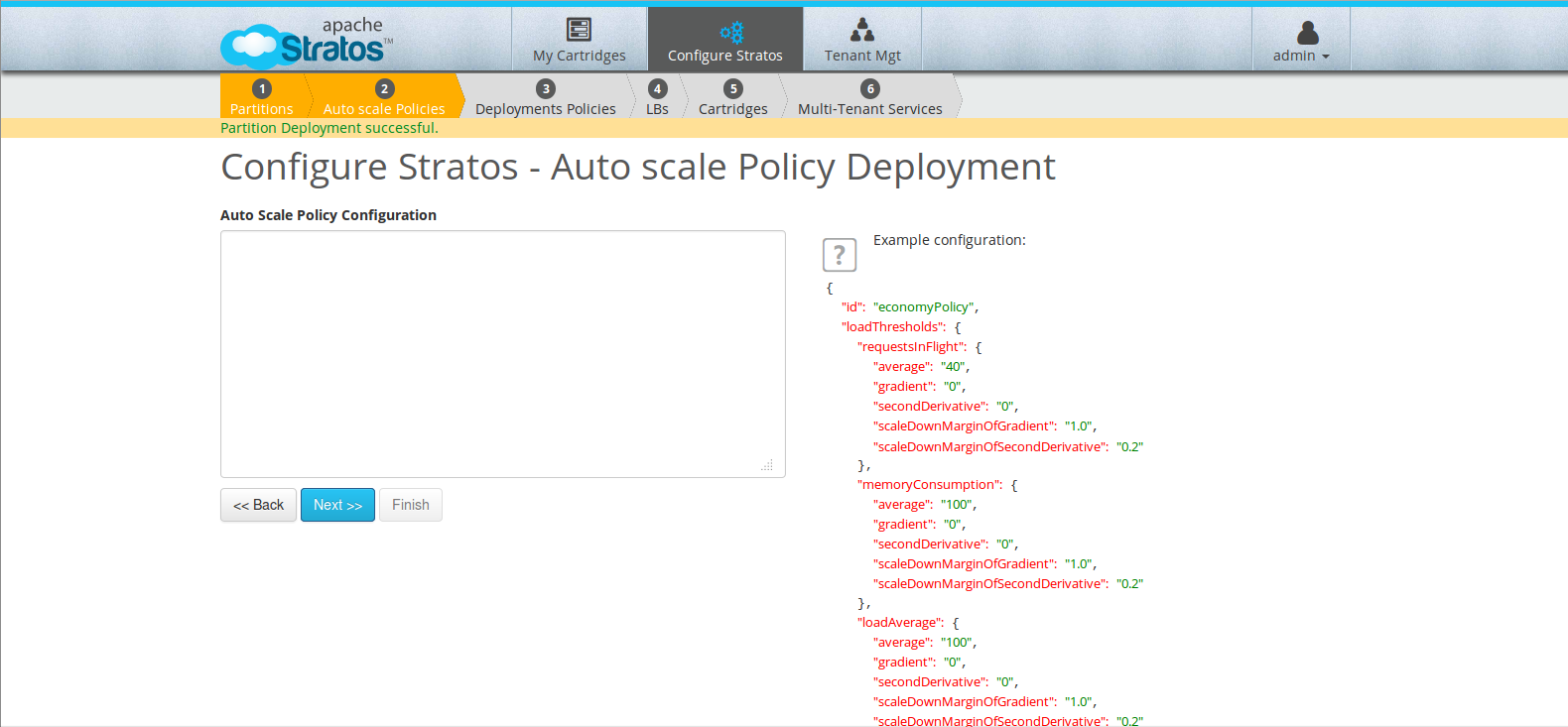


Fig 30 : Stratos Deployment

**14.3 Limitations of using Stratos**

Stratos being a newer and comparatively younger OSS solution developed lacks proper documentation and also requires a lot of prerequisites in terms of softwares to be installed before deploying an app.

**15. Conclusion**

This Report presents the exploratory research carried out on the Cloud Computing model - PaaS (Platform as a service) listing the advantages and the limitations of this layer wrt to the ease of deploying an app in cloud. The PaaS layer proves an organization and the business a complete platform as a package ready for the app to be deployed without worrying about the hassle of setting up the infrastructure for the app deployment. This paper also provides a comparative analysis on the basis of architecture and the performance, interoperability and scaling issues faced by the different Cloud providing  open source solutions like the Cloud foundry, Openshift, Appscale and Stratos. It is observed that the Cloud foundry and the Openshift are the better OSS platforms amongst the ones compared to deploy an app because of their easy to deploy environment and state of the art documentation. Cloud Foundry is emerging as new the  industry standard for the deployment of the App. Furthermore, an App named Diet Share which manages the diet plans of the individual is developed on Laravel Framework and has been deployed on Cloud Foundry and Open Shift successfully. It is also observed that Appscale requires a lot of prerequisites to be installed in the system before the app deployment can begin which is a major turn off and Stratos as being the younger OSS developed lacks proper documentation which renders users struggling with the procedure to be followed to set up the environment and deploy the app and has a lot of scope in Future.

**References**

[1] Engine Yard – Top 10 advantages of Platform as a service, available on - https://www.engineyard.com/whitepapers/top-10-advantages-of-platform-as-a-service/.

[2] Rouse, M. 2015. Platform as a Service (PaaS) definition [Online], January 2015. Availablefrom:http://searchcloudcomputing.techtarget.com/definition/Platform-as-a-Service-PaaS

[3] Sam. 2010. The Essential Characteristics of PaaS [Online], 22 February 2010. Available from: <http://cloudpulsestrat.com/posts/the-essential-characteristics-of-paas>

[4]Comparison of Open-source Paas Architectural Components by Mohan Krishna Verma Nandimandal and Eunmi Choi available from: <http://airccj.org/CSCP/vol6/csit64906.pdf>

[5]Laravel official documentation Available at - https://laravel.com/docs/5.3

[6]Pivotal cloud Foundry - Cloud Foundry overview Available from - https://docs.pivotal.io/pivotalcf/1-8/concepts/overview.html

[7]Pivotal Cloud Foundry Overview of deployment process. Available at - <https://docs.pivotal.io/pivotalcf/1-8/devguide/deploy-apps/deploy-app.html>

[8] Openshift Online official Documentation Available at - https://docs.openshift.com/online/welcome/index.html

[9] Appscale Documentation Available at - https://github.com/AppScale/appscale/wiki

[10] Appscale - https://github.com/AppScale/appscale/wiki/Managing-Apps,-Machines,-and-Logs

[11]Apache Stratos - <http://stratos.apache.org/about/why-apache-stratos.html>