

ENDPOINT SOFTWARE/APPLICATION MANAGEMENT PORTAL

A Project Report submitted in partial fulfillment of the requirements for award of the
degree

BACHELOR OF ENGINEERING

In

COMPUTER ENGINEERING

Of

SAVITRIBAI PHULE PUNE UNIVERSITY

By

TUSHAR DAHIBHATE

Exam No: B120234243

SUSHANT DHARMADHIKARI

Exam No: B120234259

KRISH GAMBHIR

Exam No: B120234272

SHRUTI KOTHARI

Exam No: B120234321

Under The Guidance of

Prof. A.R.JOSHI



Sinhgad Institutes

DEPARTMENT OF COMPUTER ENGINEERING

**SINHGAD COLLEGE OF ENGINEERING
VADGAON(BK), PUNE-41**

2015-16



Sinhgad Institutes

**SINHGAD COLLEGE OF ENGINEERING
DEPARTMENT OF COMPUTER ENGINEERING**

CERTIFICATE

This is to certify that the Project Report Entitled

Endpoint Software/Application Management Portal

Submitted by

Tushar Dahibhate

Exam No: B120234243

Sushant Dharmadhikari

Exam No: B120234259

Krish Gambhir

Exam No: B120234272

Shruti Kothari

Exam No: B120234321

Have successfully completed their Project under the supervision of Prof. A.R.Joshi for the partial fulfillment of Bachelor of engineering in Computer Engineering of Savitribai Phule Pune University. This work has not been submitted elsewhere for any degree.

Prof. A.R. Joshi
Internal Guide
Dept. of Computer Engg.

Dr. P.R.Futane
H.O.D
Dept. of Computer Engg.

External Examiner

Prof. S. D. Lokhande
Principal
Sinhgad college of Engineering

SPONSORSHIP LETTER

2/20/2016

Gmail - Sponsorship Letter for BE Project 2015/16



Tushar Dahibhate <tushar.h.dahibhate@gmail.com>

Sponsorship Letter for BE Project 2015/16

Siddharth Pendse <siddharth_pendse@persistent.com>

Fri, Sep 18, 2015 at 7:52 AM

To: "tushar.h.dahibhate@gmail.com" <tushar.h.dahibhate@gmail.com>



September 18, 2015

To Whomsoever it May Concern

This is to certify that following students from your college are undergoing their final year B.E. project at Persistent Systems Ltd. for academic year 2015-16 under group number SCOE(201) under title App Developments and Web.

Name of Students:

- i. Tushar Dahibhate**
- ii. Sushant Dharmadhikari**
- iii. Krish Gambhir**

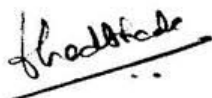
SPONSORSHIP LETTER

2/20/2016

Gmail - Sponsorship Letter for BE Project 2015/16

iv. Shruti Kothari

For Persistent Systems Ltd.

A handwritten signature in black ink, appearing to read 'K Bhadbhade', with a horizontal line drawn underneath it.

Kaustubh Bhadbhade

Senior Manager - Human Resource

DISCLAIMER ===== This e-mail may contain privileged and confidential information which is the property of Persistent Systems Ltd. It is intended only for the use of the individual or entity to which it is addressed. If you are not the intended recipient, you are not authorized to read, retain, copy, print, distribute or use this message. If you have received this communication in error, please notify the sender and delete all copies of this message. Persistent Systems Ltd. does not accept any liability for virus infected mails.

Abstract

The installation of any software requires prior knowledge of the software dependencies which only a person with some technical background will have. The installation process could be tedious and time consuming at times. Endpoint Software/Application management portal will manage software and applications on any endpoint. It will be used to automate the process of software installation to reduce the manual efforts required during installation of a software such as- extraction, installation, post-configurations, etc. and will also be time saving. The extraction and installation of the software will be done by the chef tool. Providing the development environment will be done using Docker tool.

Acknowledgments

It gives us great pleasure in presenting the preliminary project report on ‘Endpoint Software/Application Management Portal’.

*We would like to take this opportunity to thank our internal guide **Prof. A.R.Joshi** for giving us all the help and guidance we needed. We are really grateful to them for their kind support. Their valuable suggestions were very helpful.*

*We are also grateful to **Prof. P.R.Futane**, Head of Computer Engineering Department, Sinhgad College Of Engineering for his indispensable support, suggestions.*

Tushar Dahibhate
Sushant Dharmadhikari
Krish Gambhir
Shruti Kothari
(B.E. Computer Engg.)

Contents

1	INTRODUCTION	1
1.1	Background and basics	1
1.2	Literature Study	1
1.2.1	Docker	1
1.2.2	Chef	1
1.2.3	Existing alternatives	2
1.3	Project Undertaken	3
1.3.1	Problem definition	3
1.3.2	Scope	3
1.4	Outline	3
2	SOFTWARE REQUIREMENTS SPECIFICATION	5
2.1	Introduction	5
2.1.1	Purpose	5
2.1.2	Document Conventions	5
2.2	Overall Description	5
2.2.1	Product Perspective	5
2.2.2	Product Functions	6
2.2.3	Operating Environment	7
2.2.4	User Documentation	7
2.2.5	Dependencies	7
2.3	External Interface Requirements	7
2.3.1	User Interfaces	7
2.3.2	Software Interfaces	8
2.3.3	Communication Interfaces	8

2.4	System Features	9
2.4.1	UI Features	9
2.4.2	Software Installation	9
2.4.3	Application Environment Provision	9
2.4.4	Chef Client	9
2.4.5	Docker Functionality	9
2.4.6	Chef Workstation	9
2.4.7	Display Information	10
2.5	Other Non-Functional Requirements	10
2.5.1	Safety Requirements	10
2.5.2	Security Requirements	10
2.6	Project Process Modeling	10
2.7	Project Scheduling	11
2.7.1	Project task set	11
2.7.2	Timeline Chart	13
3	ANALYSIS DESIGN	14
3.1	Architectural Design	14
3.2	Usecase Diagram	15
3.3	Class Diagram	16
3.4	Activity Diagram	17
3.5	Sequence Diagram	19
3.6	Deployment Diagram	22
4	IMPLEMENTATION CODING	23
4.1	Chef Scripts	23
4.2	Docker Module	27
5	TESTING	40
5.1	Acceptance Testing	40
5.2	Unit Testing	40
5.2.1	GUI Testing Table	40
5.2.2	Communication Module	41
5.2.3	Chef Module	41

5.2.4	Docker Module	42
6	RESULTS DISCUSSION	43
6.1	Docker Results	43
6.2	Chef Results	44
7	CONCLUSION	47
8	REFERENCES	48
8.1	Review of Conference/Journal Papers supporting Project idea	48
9	APPENDIX	49

List of Figures

2.1	Waterfall model	10
2.2	Timeline Chart	13
3.1	Architecture Diagram	14
3.2	Use case-Chef	15
3.3	Use case-Docker	16
3.4	Class Diagram-Chef	16
3.5	Class Diagram-Docker	17
3.6	Activity Diagram-Chef	18
3.7	Activity Diagram-Docker	19
3.8	Sequence Diagram-Chef	20
3.9	Sequence Diagram-Docker	21
3.10	Deployment Diagram	22
6.1	Input Form	43
6.2	Output page	44
6.3	Deployed application	44
6.4	Adding new software	45
6.5	Software list	45
6.6	Providing endpoint details	46
6.7	Install Software	46

List of Tables

1	Acronyms	VIII
2.1	Time Estimation	12
5.1	Acceptance Testing Table	40

Acronyms

Acronym	Full Form
SRS	Software Requirement Specification
API	Application Programming Interface
UML	Unified Modeling Language

Table 1: Acronyms

INTRODUCTION

1.1 Background and basics

Endpoint Software/Application management portal will work as a platform to manage software and applications on any endpoint. It will be used to automate the process of software installation. It will reduce the manual efforts required during installation of a software such as- extraction, installation, post-configurations, etc. and will also be time saving. It also provides the development environment for deploying various applications inside isolated containers in a platform independent manner.

1.2 Literature Study

1.2.1 Docker

Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux. Docker uses the resource isolation features of the Linux kernel to allow independent "containers" to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines. Docker is a tool that can package an application and its dependencies in a virtual container that can run on any Linux server. This helps enable flexibility and portability on where the application can run.

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which manages Docker containers. Both the Docker client and the daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate via sockets or through a RESTful API.

1.2.2 Chef

Chef is a powerful automation platform that transforms complex infrastructure into code, bringing your servers and services to life. Whether youre operating in the cloud, on-

premises, or a hybrid, Chef automates how applications are configured, deployed, and managed across your network, no matter its size.

Chef is built around simple concepts: achieving desired state, centralized modeling of IT infrastructure, and resource primitives that serve as building blocks. These concepts enable you to quickly manage any infrastructure with Chef. These very same concepts allow Chef to handle the most difficult infrastructure challenges on the planet. Anything that can run the chef-client can be managed by Chef.

1.2.3 Existing alternatives

- Ninite:

Ninite is a software offering that lets users automatically install popular applications for their Windows operating system. It allows users to make a selection from a list of applications and bundles the selection into a single installer package. It is available only for Windows OS. There is no provision for adding new softwares dynamically.

Endpoint Software/Application Management Portal is available for Linux based Operating System. It provides the facility to add new softwares as per the users need.

- Vagrant:

Vagrant is an open-source product released in 2010, is best described as a VM manager. It allows you to script and package the VM config and the provisioning setup. It is designed to run on top of almost any VM tool VirtualBox, VMWare, AWS, etc. It provides a reproducible way to create fully virtualized machines using Oracle's VirtualBox.

Docker provides a way to create a snapshot of the Operating system and the applications that we want to run into a common image and then easily deploy these images on Docker hosts. These images are light-weight as compared to a full Virtual Machine.

- Puppet:

Puppet is an open source configuration management tool. It runs on many unix-like systems as well as on Windows and includes its own declarative language to

describe system configuration.

Chef in comparison to Puppet is more developer friendly with favourites like Chef-Knife plugin architecture and the Chef-Developer kit relegated mostly to developer use. Chef also provides an additional layer of security with Chef-Vault that enables easier management of encrypted Data bags.

1.3 Project Undertaken

1.3.1 Problem definition

To develop a solution which will work as a platform to provide software on any endpoint and to provide a development environment by packaging all the dependencies required to deploy an application.

1.3.2 Scope

Endpoint Software/Application Management Portal will work as a platform to manage software and applications on any endpoint. It will be used to automate the process of software installation. It will reduce the manual efforts required during installation of a software such as- extraction, installation, post-configurations, etc. and will also be time saving. Using this portal we can deploy web applications by installing and packaging all the dependencies inside an isolated container. Endpoint Software/Application Management Portal, being a web based tool, availability of internet is a must. Also softwares, specific to the Linux Operating system (Fedora 19 onwards) can be installed.

1.4 Outline

This project report is segregated into seven chapters.

- The first chapter consists of background, scope and purpose of the project.
- The second chapter is Software Requirement Specification which covers system features, external interfaces required and non functional requirements of the system.
- Architecture and Component level design and also a Data design and UML Diagrams comprise the third chapter.
- Fourth chapter presents an overview of implementation of project modules with coding.

- Chapter five deals with test plan, user acceptance testing and integration testing.
- The sixth chapter discusses the result of the project.
- Seventh chapter ends with the conclusion.
- References and research papers referred are attached at the end of this project report.

SOFTWARE REQUIREMENTS SPECIFICATION

2.1 Introduction

2.1.1 Purpose

To automate the tedious and the conventional process of software installation and to facilitate machine independent deployment of applications along with its dependencies. Endpoint Software/Application Management Portal will work as a platform to manage softwares and applications on any endpoint accessible.

2.1.2 Document Conventions

The format of this SRS is simple. Bold face and indentation is used on general topics and or specific points of interest.

2.2 Overall Description

2.2.1 Product Perspective

Installing software can sometimes be difficult and time consuming as well. It is also not possible to keep tabs on the installation process and to keep pressing next occasionally until the whole process is complete. Also in some installations we often need to change the Environment variables in order to conform the environment to the desired settings for the software to smoothly install and run. People from non-IT professions often need to take the assistance of an IT expert in order to install particular software.

2.2.2 Product Functions

2.2.2.1 Software installation

- Step 1: Accepting the user input.
 - * Name of the Software to be installed
 - * Operating System of the endpoint
 - * Version of the Software
 - * User Credentials(Root password)
 - * IP Address of the endpoint
- Step 2: Installation of Chef client.
- Step 3: Searching if the software is present in the database.
- Step 4: Creation of recipe corresponding to the software.
- Step 5: Running the chef recipe for installation of software.

2.2.2.2 Application Deployment

- Step 1: Accepting the user input.
 - * Application environment needed for the application(Example-Tomcat Server,NodeJS)
 - * Version required
 - * Path of the file to be deployed(tar.gz archive)
 - * User Credentials(Username and Root password)
 - * IP Address of the endpoint
- Step 2: Searching the Dockerfile corresponding to the development environment.
- Step 3: Building the Dockerfile to create a Docker Image.
- Step 4: Creating and Starting docker container using the Docker image built before.
- Step 5: Copying the file into the container.
- Step 6: Providing the details of the container to the user.

2.2.3 Operating Environment

- Client machine: The client or user machine on which the software gets installed
- Chef server: The server contains the repository of the software that have already been installed by some nodes and manage them in the database. It is a hosted Chef server.
- Chef Workstation :(Version 11.2) The Workstation helps in managing the nodes and deployment of the software on multiple nodes.
- Chef Client :(Version 12.8.1) Chef Client will take the responsibility to install the software automatically by running the corresponding recipe.
- Docker Engine:(Version 1.8) Docker engine will be present on the client machine.
- Docker Remote API: (Version 1.20) Docker Remote API must be enabled on the client machine where docker engine is present.

2.2.4 User Documentation

Online help how to use software will be provided to user.

2.2.5 Dependencies

- Database is needed for storing available Softwares.
- Eclipse IDE is needed on the server to host the Web application.
- Docker engine needs to be pre-installed on the Client side.

2.3 External Interface Requirements

2.3.1 User Interfaces

The user interface includes the web based GUI designed for the user to get the requirements as input and install the required software. For Software installation, the user will have to select his operating system and enter the software

name and IP address along with the user credentials and click install. For provision of application environment, the user will have to select the environment and version. User will then have to input the path of the file that he wants to deploy along with the IP address and user credentials. The User Interface will be developed using HTML and CSS.

2.3.2 Software Interfaces

- Java servlet is used for server-side scripting and processing.
- Docker Remote API is a ReSTful API provided to access Docker remotely.
- JSch library for secure copy of file from client to server.
- Java-JSON library for parsing JSON data
- Apache commons collection for multi-maps

2.3.3 Communication Interfaces

- Java Servlet: The software details are taken from the UI. These are then forwarded to chef and docker for further processing.
- Hyper Text Transfer Protocol : HTTP POST,GET and PUT methods are used for firing Docker requests. Docker Remote API is a RESTful API which uses these HTTP methods.
- System Interfaces:
 - * Chef client-server interaction: The client on which the software is to be installed gets bootstrapped to the server through chef workstation and simultaneously chef client gets installed. The software specified by the user gets added to runlist of the client and then the software gets installed. This process is automated by the chef itself and no external commands need to be carried out.
 - * Docker interaction: Dockerfile corresponding to the environment and the version gets built to form a Docker image. Container gets spawned for the Docker image created before. The file that user wants to deploy gets copied into the container. Container has the environment required to run the application which user wants to deploy.

2.4 System Features

2.4.1 UI Features

Our system will be based on Client-server architecture. We will employ a powerful commodity server. The server will be responsible for accepting client requests and giving them results accordingly.

2.4.2 Software Installation

- This feature will provide the option for the selection of the OS at the endpoint.
- Depending on the OS selected the user then will provide the software name, IP address and other credentials.

2.4.3 Application Environment Provision

- User will provide the environment and the version along with the path of the file to be deployed and the user credentials.

2.4.4 Chef Client

The chef client will take the responsibility to install the software automatically by running the corresponding recipe.

2.4.5 Docker Functionality

Docker will spawn containers which will package the application and its dependencies in an isolated manner.

2.4.6 Chef Workstation

The chef-workstation will be responsible to manage the nodes in the network. Chef workstation is available for Ubuntu 14.04 onwards.

The steps to install Chef Workstation are as follows:

Step 1: Get chef_dk(Development Kit) from the website chef.io

Step 2: Install chef_dk.

Step 3: Get the starter kit from the server.

Step 4: After extracting the starter kit, the directory chef-repo will be created.

Step 5: Execute the Knife commands.

2.4.7 Display Information

Once the software is installed, the display message should be shown for successful installation. If there occurs any exception then the respective message should be displayed to handle the exception.

2.5 Other Non-Functional Requirements

2.5.1 Safety Requirements

Installation through this tool should not cause any undesired settings or changes.

2.5.2 Security Requirements

Only the administrative privileges for the chef client should be provided by the endpoint.

2.6 Project Process Modeling

We have adopted the linear sequential model for software development.

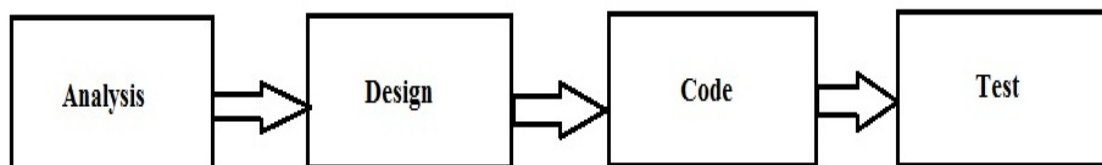


Figure 2.1: Waterfall model

Sometimes called the classic life cycle or the waterfall model, the linear sequential model suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing, and support. The above figure illustrates the linear sequential model for software engineering.

System/information engineering and modeling: Because software is always part of a larger system (or business), work begins by establishing requirements for all system elements and then allocating some subset of these requirements to software.

This system view is essential when software must interact with other elements such as hardware, people, and databases. System engineering and analysis encompass requirements gathering at the system level with a small amount of top level design and analysis. Information engineering encompasses requirements gathering at the strategic business level and at the business area level.

2.7 Project Scheduling

2.7.1 Project task set

Major Tasks in the Project stages are:

- **Software requirements analysis:** The requirements gathering process is intensified and focused specifically on software. To understand the nature of the program(s) to be built, the software engineer ("analyst") must understand the information domain for the software, as well as required function, behavior, performance, and interface. Requirements for both the system and the software are documented and reviewed with the customer.
- **Design:** Software design is a multistep process that focuses on four distinct attributes of a program: data structure, software architecture, interface representations, and procedural (algorithmic) detail. The design process translates requirements into a representation of the software that can be assessed for quality before coding begins. Like requirements, the design is documented and becomes part of the software configuration.
- **Planning:** Planning activity guides the team in the development of the project. The software project plan defines the software engineering work by describing the tech-

nical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.

- **Code generation:** The design must be translated into a machine-readable form. The code generation step performs this task. If design is performed in a detailed manner, code generation can be accomplished mechanistically.
- **Testing:** Once code has been generated, program testing begins. The testing process focuses on the logical internals of the software, ensuring that all statements have been tested, and on the functional externals; that is, conducting tests to uncover errors and ensure that defined input will produce actual results that agree with required results.
- **Deployment:** The software (as a complete entity or as a partially completed increment) is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

2.7.1.1 Time Estimates

Months	Activities
June	Selecting the domain
July	Searching the relevant topic from domain, Short listing and finalizing the topic, Review-1
August	Literature Survey, Identifying the scope, increasing the scope
September	Identify the suitable algorithm, Created UML Diagrams, Mathematical Model, PPT preparation, Review-2
October	Documentation ,Preliminary report, Idea Matrix, SRS, Parag Kulkarni Chapter 7, Feasibility Analysis, Revised Mathematical Model
November	Learning of Chef,Docker,Java Script,Java Servlet
December	In-Depth exploration of Chef and Docker
January	Module planning
February	Implementation of the modules
March	Implemented connectivity between each module, prepared report

Table 2.1: Time Estimation

2.7.2 Timeline Chart

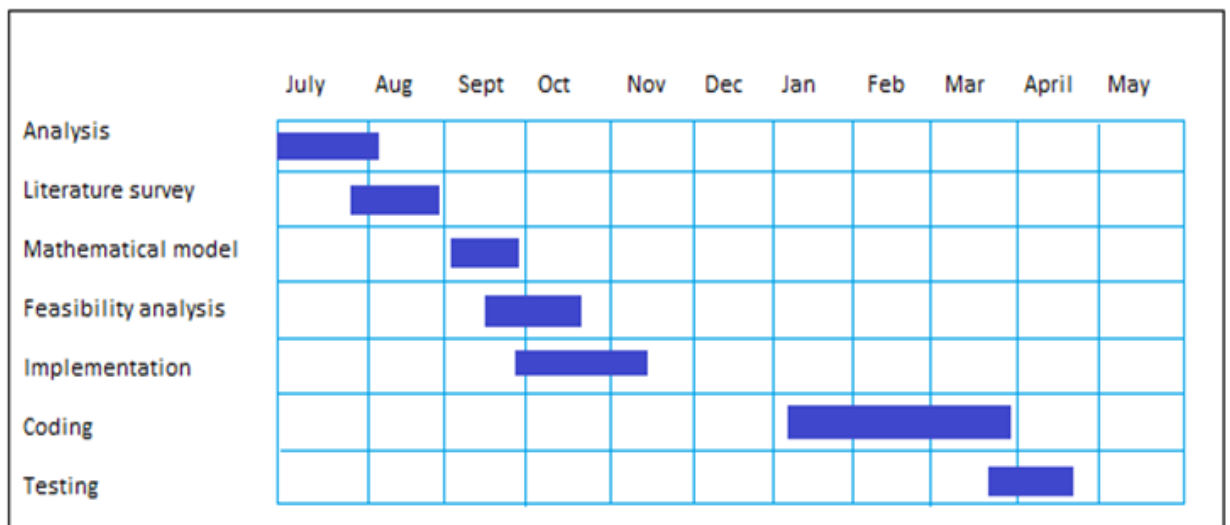


Figure 2.2: Timeline Chart

ANALYSIS DESIGN

3.1 Architectural Design

A description of the program architecture is presented. Subsystem design or Block diagram, Package Diagram, Deployment diagram with description is to be presented.

Below figure shows the architectural design of Endpoint Software Application Management Portal. The blocks which are shown in architecture diagram are as follows:

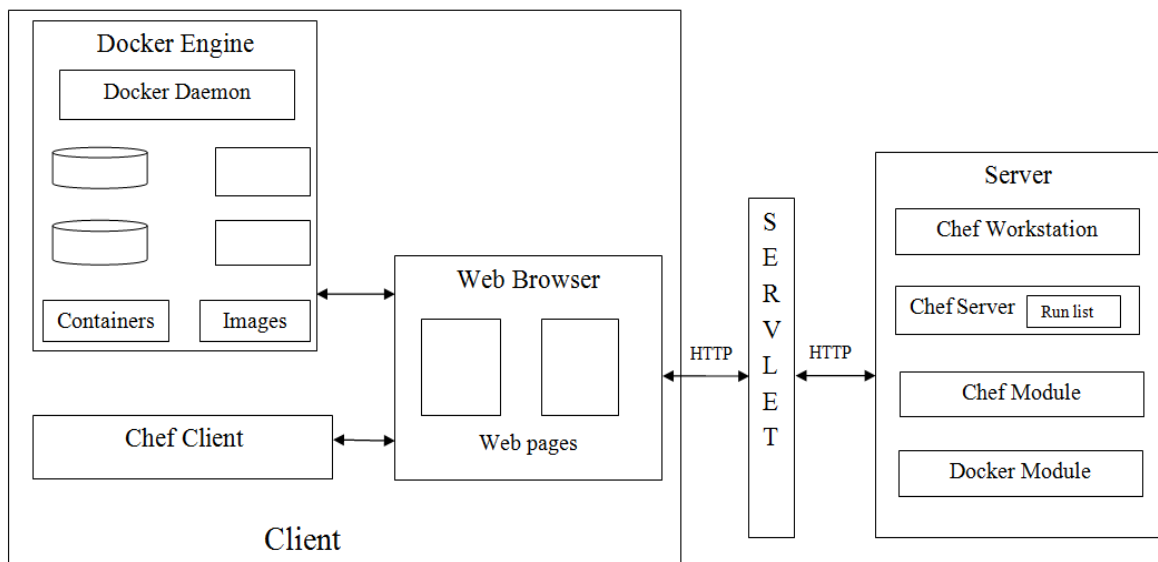


Figure 3.1: Architecture Diagram

- Chef client-The chef client will take the responsibility to install the software automatically by running the corresponding recipe.
- Docker Engine- Docker engine must be present on client machine. It is responsible to create docker containers. Docker containers are light weight containers which wrap up a piece of software in a complete file system along with every file system that a software needs to run.
- Docker image is a software which we load into the container. It is created using a

Docker file. A Docker file is a text file which contains all the commands that a user could call on the command line to assemble an image.

- Docker Daemon Docker daemon communicates with the Docker engine and manages the Docker containers.
- Web Browser
- Servlet
- Server-The server works as a store to keep the recipes for the software and the client lists and information of the client.

3.2 Usecase Diagram

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

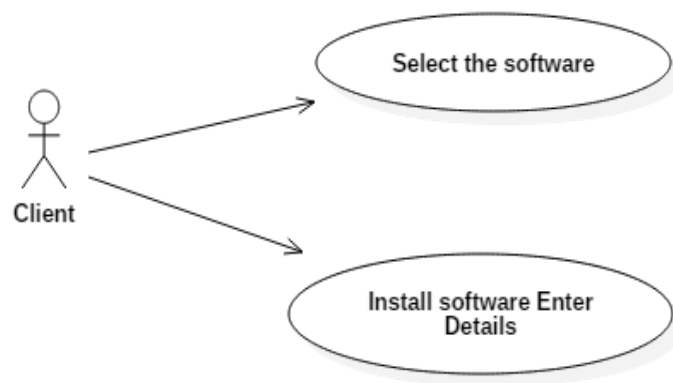


Figure 3.2: Use case-Chef

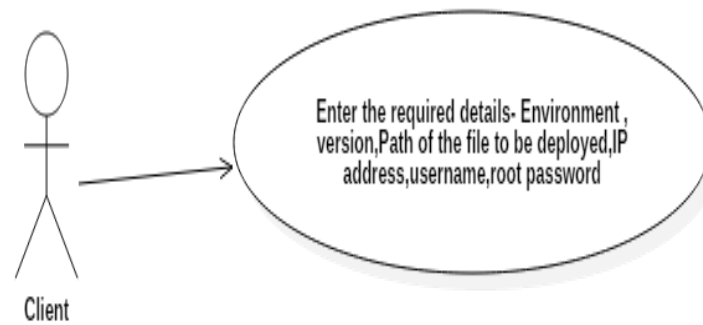


Figure 3.3: Use case-Docker

3.3 Class Diagram

The class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing and documenting different aspects of a system but also for constructing executable code of the software application.

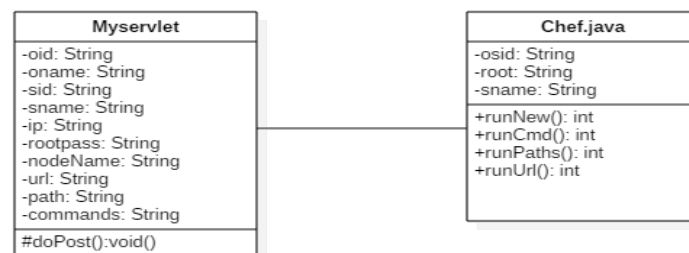


Figure 3.4: Class Diagram-Chef

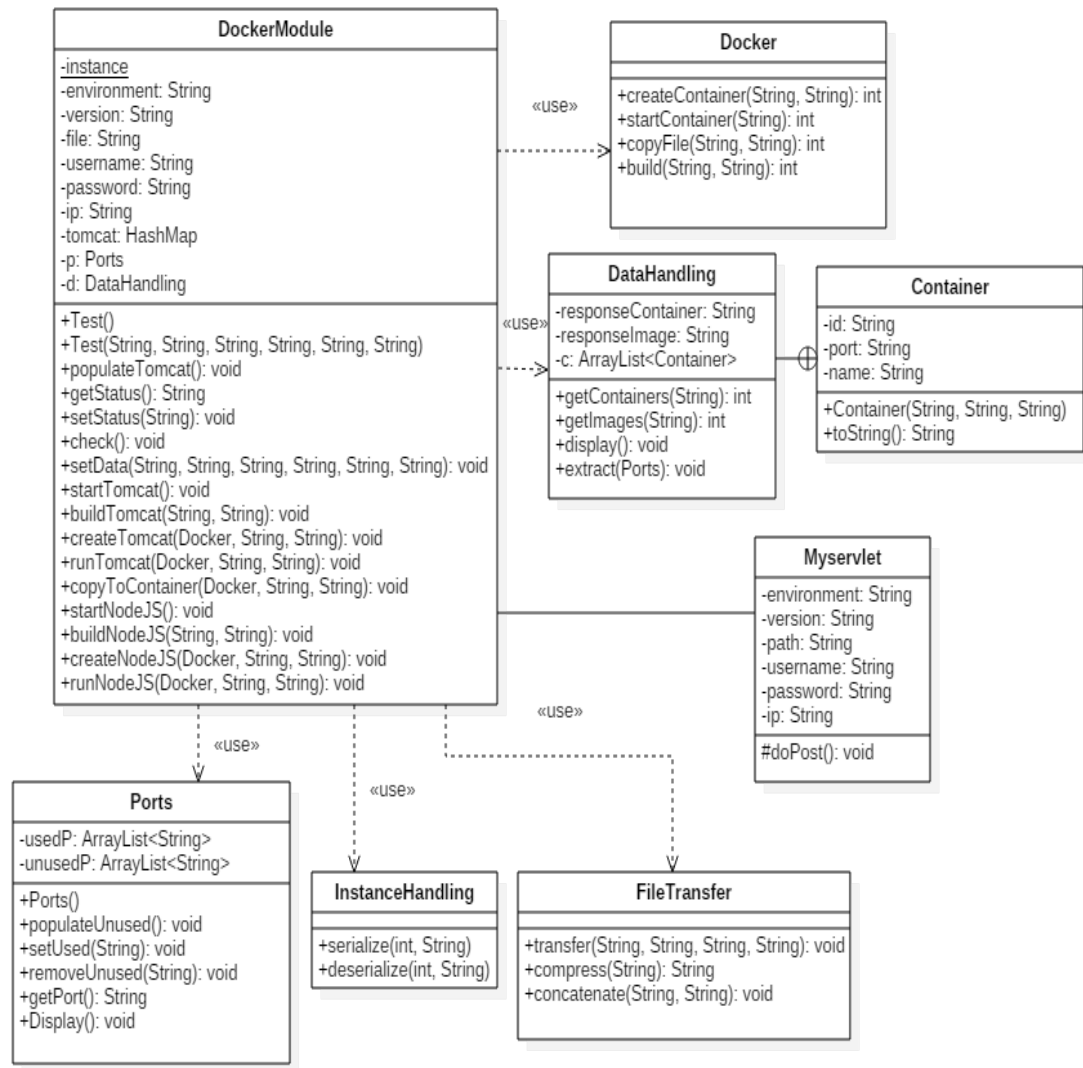


Figure 3.5: Class Diagram-Docker

3.4 Activity Diagram

An activity diagram refers to the flow of activity involved in triggering any event. The prerequisite for the activity diagram are scope statement and Use Case diagram. An activity is ongoing non atomic execution within a state machine. Activity ultimately results in those events which are made up of executable automatic computation that results in change of state of system. Activity diagram commonly contains activities, states, action states, transition, object notes etc.

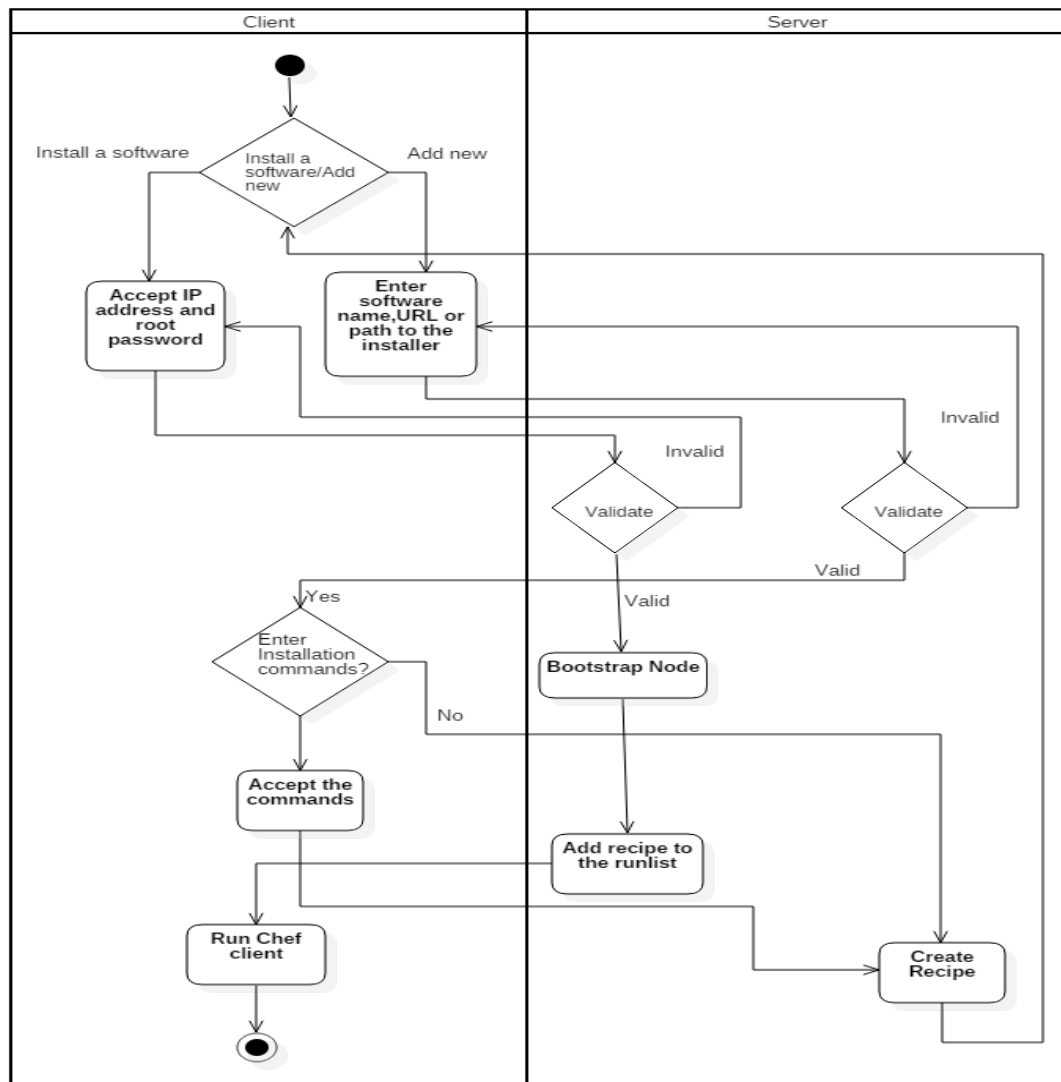


Figure 3.6: Activity Diagram-Chef

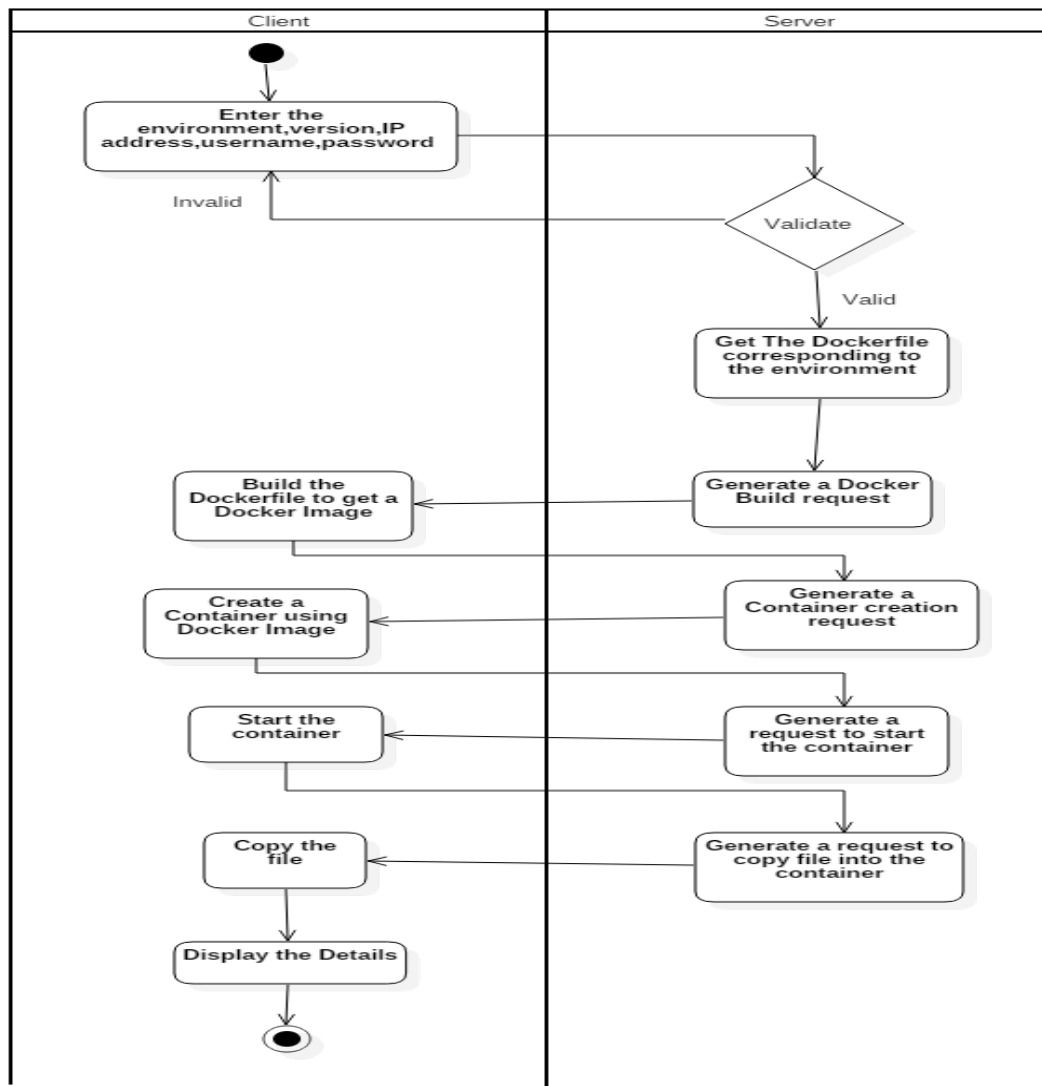


Figure 3.7: Activity Diagram-Docker

3.5 Sequence Diagram

A Sequence diagram is an interaction diagram that shows how processes operate with one another and in what order. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

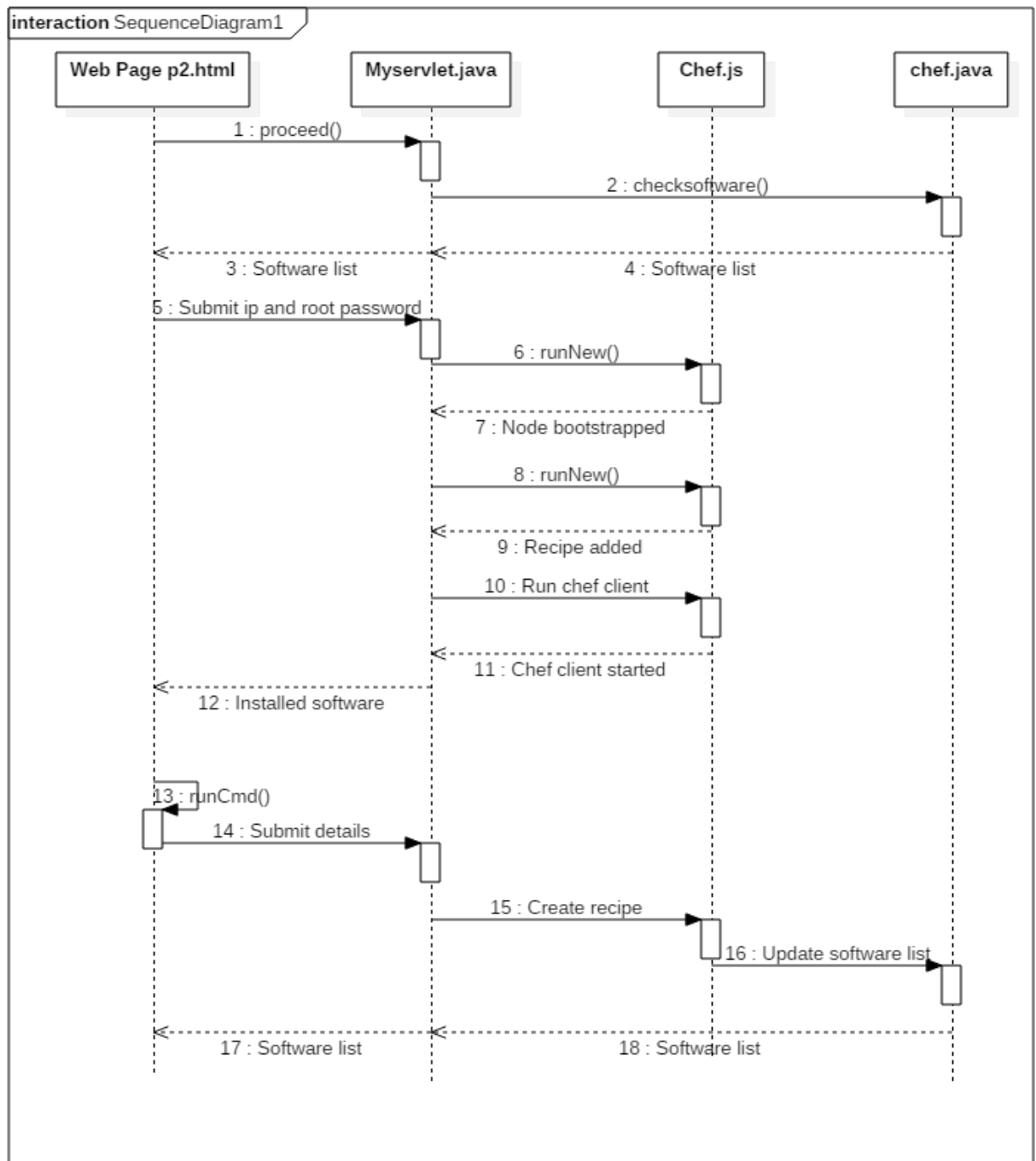


Figure 3.8: Sequence Diagram-Chef

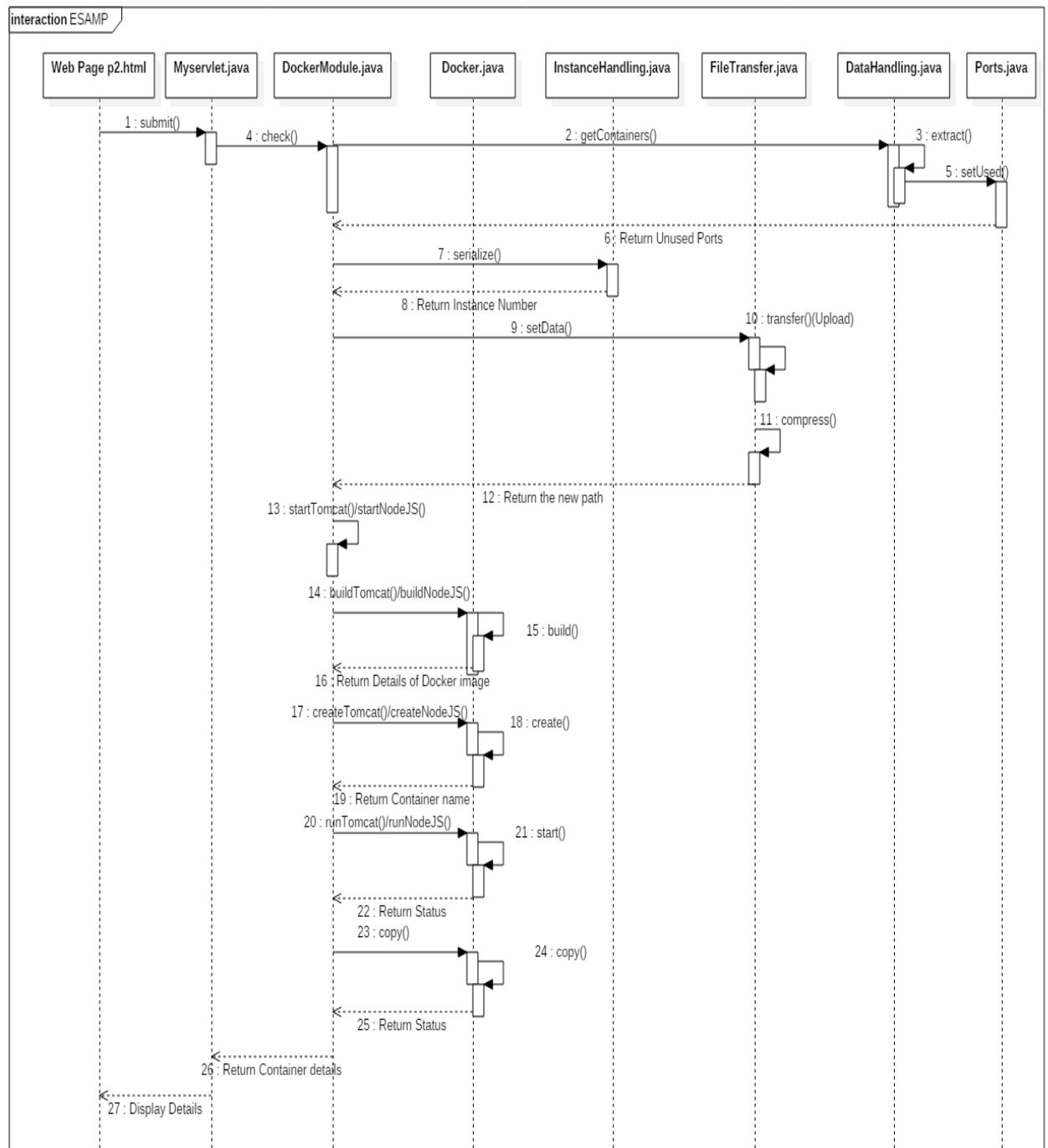


Figure 3.9: Sequence Diagram-Docker

3.6 Deployment Diagram

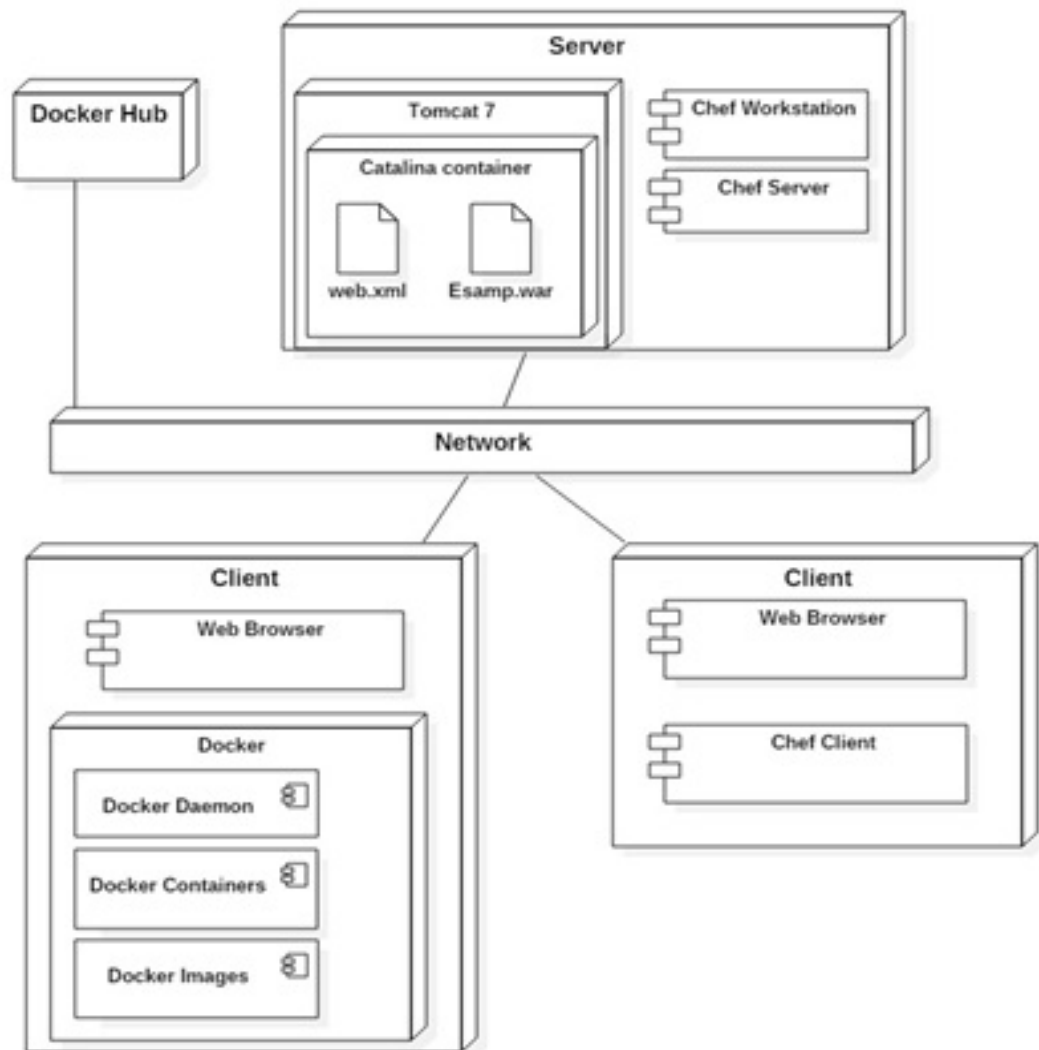


Figure 3.10: Deployment Diagram

IMPLEMENTATION CODING

4.1 Chef Scripts

MYSCRIPT.SH

```
#!/bin/bash
cd /home/esamp/chef-repo
echo "client list"
knife client list
echo "bootstrapping"
knife bootstrap $1 --ssh-user root --ssh-password $2 --sudo --use-sudo-password -
echo "DONE!"
echo $4
knife node run_list add $3 'recipe['$4']'
echo "recipe added successfully!"
knife ssh $1 'chef-client' --manual-list --ssh-user root --ssh-password $2
echo "Running chef-client!"
knife node run_list remove $3 'recipe['$4']'
```

MYDYNAMIC.SH

```
#!/bin/bash
echo "Creating recipe"
cd /home/esamp/chef-repo
echo '>default.rb'
cat template.rb>default.rb
grep -rl default default.rb | xargs sed -i "s/default/$1/g"
```

```
echo "Creating cookbook"
cd /home/esamp/chef-repo/
knife cookbook create $1
cd /home/esamp/chef-repo/cookbooks/$1/recipes
cat /home/esamp/chef-repo/default.rb>>default.rb
echo "Uploading cookbook"
knife cookbook upload $1
echo "Success!"
```

PATH.SH

```
#!/bin/bash
echo $1 #path
echo "Creating recipe"
echo ''>default_1.rb
echo ''>/home/esamp/chef-repo/cookbooks/default_1/recipes/default.rb
cat template_1.rb>default_1.rb
grep -rl path default_1.rb | xargs sed -i "s#path#$1#g"
echo "Creating cookbook"
#cd /home/esamp/chef-repo/
#knife cookbook create default_1
cd /home/esamp/chef-repo/cookbooks/default_1/recipes
cat /home/esamp/chef-repo/default_1.rb>>default.rb
echo "Uploading cookbook"
knife cookbook upload default_1
echo "Success!"
```

USER_COMMANDS.SH

```
#!/bin/bash
echo "User commands"
```

```
cd /home/esamp/chef-repo
echo ''>cmd.rb
echo ''>/home/esamp/chef-repo/cookbooks/usr_cmd/recipes/default.rb
cat template_cmd.rb>cmd.rb
grep -rl path cmd.rb | xargs sed -i "s#path#$3#g"
#knife ssh $1 '$3' --manual-list --ssh-user root --ssh-password $2
#knife cookbook create usr_cmd
#knife cookbook create $1
cd /home/esamp/chef-repo/cookbooks/usr_cmd/recipes
cat /home/esamp/chef-repo/cmd.rb>>default.rb
echo "Uploading cookbook"
knife cookbook upload usr_cmd
echo "Success!"
knife node run_list add N1.1.5.22 'recipe[usr_cmd]'
echo "recipe added successfully!"
knife ssh $1 'chef-client' --manual-list --ssh-user root --ssh-password $2
echo "Running chef-client!"
echo "Running commands!"
knife node run_list remove N1.1.5.22 'recipe[usr_cmd]'
```

URL.SH

```
#!/bin/bash
echo $1 #path
echo "Creating recipe"
echo ''>default_url.rb
echo ''>/home/esamp/chef-repo/cookbooks/default_url/recipes/default.rb
cat template_url.rb>default_url.rb
grep -rl url default_url.rb | xargs sed -i "s#url#$1#g"
echo "Creating cookbook"
#cd /home/esamp/chef-repo/
```

```
knife cookbook create default_url
cd /home/esamp/chef-repo/cookbooks/default_url/recipes
cat /home/esamp/chef-repo/default_url.rb>>default.rb
echo "Uploading cookbook"
knife cookbook upload default_url
echo "Success!"
```

TEMPLATE.RB

```
package 'default' do
  action :install
end
service 'default' do
  action [ :enable, :start ]
end
```

TEMPLATE_CMD.RB

```
execute 'user_cmds' do
  command 'path'
end
```

TEMPLATE_URL.RB

```
execute 'user_http' do
  command 'yum install -y url'
end
```

4.2 Docker Module

```
package org.Final_BE_Project.test;

import java.io.File;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class DockerModule
{
    static int instance;
    String environment,version,file,username,password,ip;
    String status;

    Map<String,String> tomcatDockerfilePaths=new HashMap<String,String>();
    Map<String,String> nodeJSDockerfilePaths=new HashMap<String,String>();

    Ports p=new Ports();
    DataHandling d=new DataHandling();

    DockerModule()
    {
        this.populateTomcat();
        this.populateNodeJS();
    }

    public DockerModule(String environment, String version,
        String file, String username, String password,String ip)
    {
        super();
        this.environment = environment;
        this.version = version;
        this.file = file;
```

```
this.username = username;
this.password = password;
this.ip=ip;

}

public void populateTomcat()
{
tomcatDockerfilePaths.put("7jre7", "/home/esamp/workspace/
Final_BE_Project/Tomcat/Tomcat7_7.tar.gz");
tomcatDockerfilePaths.put("7jre8", "/home/esamp/workspace/
Final_BE_Project/Tomcat/Tomcat7_8.tar.gz");
tomcatDockerfilePaths.put("8jre7", "/home/esamp/workspace/
Final_BE_Project/Tomcat/Tomcat8_7.tar.gz");
tomcatDockerfilePaths.put("8jre8", "/home/esamp/workspace/
Final_BE_Project/Tomcat/Tomcat8_8.tar.gz");
}

public void populateNodeJS()
{
nodeJSDockerfilePaths.put("Argon", "/home/esamp/workspace
/Final_BE_Project/NodeJS/NodeJS_Argon.tar.gz");
}

public void setStatus(String status)
{
this.status=status;
}

public String getStatus()
{
```



```
return status;
}

public void getData()
{
    Scanner sc=new Scanner(System.in);

    System.out.println("\n Enter the Environment");
    environment=sc.next();
    System.out.println("\n Enter the Version");
    version=sc.next();
    System.out.println("\n Enter the Path");
    file=sc.next();
    System.out.println("\n Enter the username");
    username=sc.next();
    System.out.println("\n Enter the password");
    password=sc.next();
    System.out.println("\nEnter ip");
    ip=sc.next();

}

public void setData(String environment,String version,
String file,String username,String password,String ip)
{
    this.environment=environment;
    this.version=version;
    this.file=file;
    this.username=username;
    this.password=password;
    this.ip=ip;
}
```

```
}

public int check()
{
String url_c="http://" + ip + ":2375/containers/json?all=1";
d.getContainers(url_c);
d.extract(p);

String file=ip+".ser";
InstanceHandling i=new InstanceHandling();

if(new File(file).exists())
{
instance=i.deserialize(instance,ip);
instance++;
i.serialize(instance, ip);
}
else
{
i.serialize(instance,ip);
}

if(environment.equals("Tomcat"))
{
startTomcat();

i.serialize(instance,ip);
}
else if(environment.equals("NodeJS"))
{
startNodeJS();
```

```
i.serialize(instance, ip);
}

status=getStatus();

if(!status.contains("Error"))
{
return 200;
}
else
{
return 0;
}

}

public void startTomcat()
{
String PATH;

if (tomcatDockerfilePaths.containsKey(version))
{
PATH=tomcatDockerfilePaths.get(version);
buildTomcat(PATH,ip);
}
else
{
System.out.println("Error. Please enter a correct version");
}
}
```

```
public void buildTomcat(String PATH,String ip)
{
String imageName="tomcat:"+version;
String urlBuild="http://"+ip+":2375/build?t="+imageName;
String containerName;

    Docker dockerObject=new Docker();

    if(dockerObject.build(urlBuild,PATH)==200)
    {
        System.out.println("Build Successful");
        containerName=createTomcat(dockerObject, imageName,ip);
        if(containerName.equals(null))
        {
            System.out.println("Error");
        }
        else
        {
            System.out.println("Creation Successful");
            runTomcat(dockerObject,containerName,ip);
        }

    }
    else
    {
        System.out.println("Build Unsuccessful");
    }

}

public String createTomcat(Docker dockerObject,String
    imageName,String ip)
```

```
{
String containerName=version+instance;

String publicPort=p.getPort();
String urlCreate="http://"+ip+":2375/containers/create?name=
"+containerName;
String data="{\"Image\": \""+imageName+"\", \"PortBindings\":
{ \"8080/tcp\": [{ \"HostPort\": \""+publicPort+"\" } ]},
  \"PublishAllPorts\": true}";

if(dockerObject.createContainer(urlCreate,data)==201)
{
instance++;
System.out.println(publicPort);

setStatus("Container Name: "+containerName+":
Port Name :"+publicPort);
System.out.println("Container created");
return containerName;
}
else if(dockerObject.createContainer(urlCreate,data)==404)
{
System.out.println("Creation failed");
}
else if(dockerObject.createContainer(urlCreate,data)==500)
{
System.out.println("Server error");
}
return "Error";

}
```

```
public void runTomcat(Docker d,String containerName,String ip)
{

String urlRun="http://"+ip+":2375/containers/"+containerName+"/start";
if(d.startContainer(urlRun)==204)
{
System.out.println("Container Started");

if(file.equals(null))
{
System.out.println("No file given !!");
}
else
{
copyToContainer(d,containerName,ip);
}
}
else if(d.startContainer(urlRun)==304)
{
System.out.println("Container already started");
}
else if(d.startContainer(urlRun)==404)
{
System.out.println("No such container");
}
else if(d.startContainer(urlRun)==500)
{
System.out.println("Server Error");
}

}
```

```
public void copyToContainer(Docker dockerObject,String containerName,
String ip)
{
String compressedFile;

FileTransfer f=new FileTransfer();
f.transfer(file, username, password,ip);
compressedFile=f.compress(file);

String url_a="http://" +ip+":2375/containers/"+containerName+
"/archive?path=/usr/local/tomcat/webapps/";
String path_a="/home/esamp/workspace/Final_BE_Project/
"+compressedFile;

if(dockerObject.copyFile(url_a, path_a)==200)
{
System.out.println("Copying successful");
}
else if(dockerObject.copyFile(url_a, path_a)==400)
{
System.out.println("Bad parameters");
}
else if(dockerObject.copyFile(url_a, path_a)==404)
{
System.out.println("Container not found");
}
else if(dockerObject.copyFile(url_a, path_a)==500)
{
System.out.println("Server error");
}
```

```
}

//-----

public void startNodeJS()
{

    String PATH = null;

    if (nodeJSDockerfilePaths.containsKey(version))
    {
        PATH=nodeJSDockerfilePaths.get(version);

        FileTransfer f=new FileTransfer();
        f.transfer(file, username, password, ip);
        f.concatenate(PATH,file);

        buildNodeJS(PATH, ip);
    }
    else
    {
        System.out.println("Error. Please enter a correct version");
    }

}

public void buildNodeJS(String PATH,String ip)
{
    String imageName="node:"+version;
    String urlBuild="http://"+ip+":2375/build?t="+imageName;
    String containerName;

    Docker dockerObject=new Docker();
```



```
if(dockerObject.build(urlBuild, PATH)==200)
{
System.out.println("Build Successful");
containerName=createNodeJS(dockerObject, imageName,ip);
if(containerName.equals(null))
{
System.out.println("Error");
}
else
{
System.out.println("Creation successful");
runNodeJS(dockerObject,containerName,ip);
}
}
else
{
System.out.println("Build unsuccessful");
}

}

public String createNodeJS(Docker dockerObject,String imageName,String ip)
{
String containerName=version+instance;

String publicPort=p.getPort();
String urlCreate="http://"+ip+":2375/containers/create?name
="+containerName;
String data="{\"Image\": \""+imageName+"\", \"PortBindings\":
{ \"8080/tcp\": [{ \"HostPort\": \""+publicPort+"\" }]},
  \"PublishAllPorts\": true}";
```

```
if(dockerObject.createContainer(urlCreate,data)==201)
{
instance++;
System.out.println(publicPort);
setStatus("Container Name: "+containerName+":
    Port Name :"+publicPort);
return containerName;
}
else if(dockerObject.createContainer(urlCreate,data)==404)
{
System.out.println("Creation failed");
}
else if(dockerObject.createContainer(urlCreate,data)==500)
{
System.out.println("Server error");
}

return "Error";

}

public void runNodeJS(Docker dockerObject,String containerName,
String ip)
{
if(containerName.isEmpty())
{
System.out.println("Container Not created");
}
else
{

```

```
String urlRun="http://"+ip+":2375/containers/"+containerName+"/start";

if(dockerObject.startContainer(urlRun)==204)
{
System.out.println("Container Started");
}
else if(dockerObject.startContainer(urlRun)==304)
{
System.out.println("Container already started");
}
else if(dockerObject.startContainer(urlRun)==404)
{
System.out.println("No such container");
}
else if(dockerObject.startContainer(urlRun)==500)
{
System.out.println("Server Error");
}
else
{
System.out.println("Fail");
status="NULL";
}

}
}
```

TESTING

5.1 Acceptance Testing

Table 5.1: Acceptance Testing Table

Test Case Id	Test Case	Input	Pass Criteria
1	Validate all fields of the forms	Login form	Proprly formatted data.
2	Authentication	Username, Password	Both should be correct.
3	Database connection	Database credentials	Validity of credentials as well as permissible data.

5.2 Unit Testing**5.2.1 GUI Testing Table**

Test Case Id	Test Case	Input	Pass Criteria
1	Database connection	Database credentials	Validity of credentials as well as permissible data.
2	Authentication	Username, Password	Both should be correct.
3	Validate all fields of the forms	Login form	Properly formatted data.

5.2.2 Communication Module

Test Case Id	Test Case	Input	Pass Criteria
1	Data arrived from from software Installation UI	OS, Software name along with its version, Client IP Address, User credentials	Data is forwarded to chef module.
2	Data arrived from application deployment	Environment name, Version, Path to the file, IP address, Username, password	Data is forwarded to docker module.

5.2.3 Chef Module

Test Case Id	Test Case	Input	Pass Criteria
1	Client Bootstrap	IP Address and root password	Chef client installed on client and an entry for the client on the chef server.
2	Package existence for the software	Valid software name	A valid cookbook is created and uploaded on the server
3	Adding recipe to runlist	Name of the cookbook	Cookbook name is added in the runlist of the client.
4	Execute chef-client on the client side	IP address of the client	The runlist gets executed and the software starts installing.

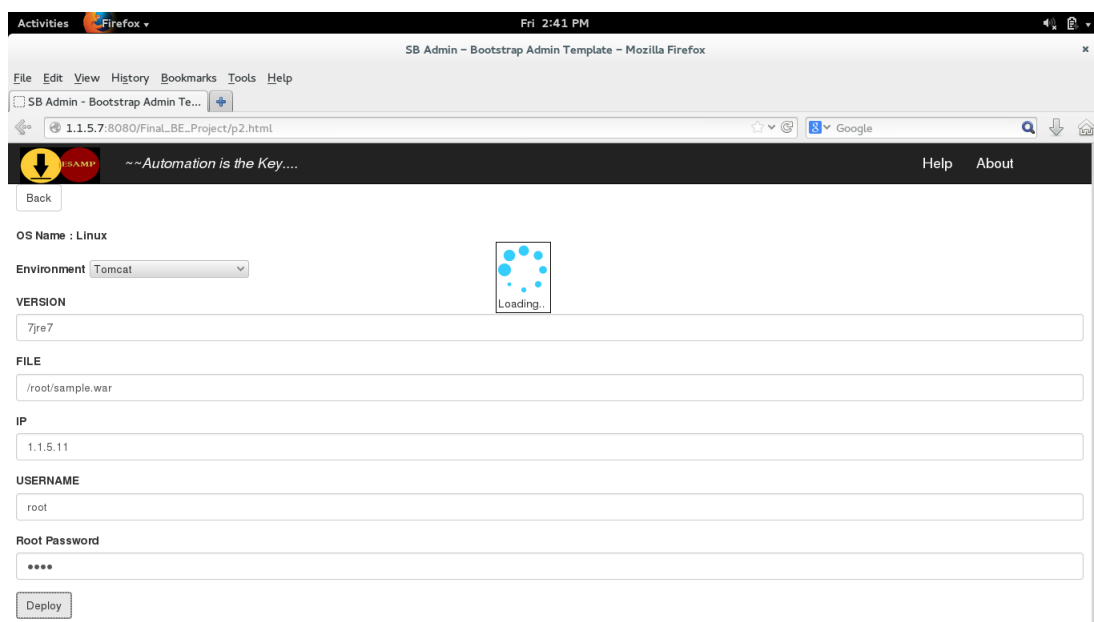
5.2.4 Docker Module

Test Case Id	Test Case	Input	Pass Criteria
1	Build the docker file	Docker file	Return code 200
2	Creation of container	Container name, Docker image	Return code 201
3	Run the container	Container name, available port	Return code 204
4	Copy a file into the container	Container name, path of the file	Return code 200.
5	File transfer	Path of the source, destination, user name, password	The file is present in the destination folder.

RESULTS DISCUSSION

6.1 Docker Results

After selecting Application Deployment on the Home page, user will be redirected to a form where the details of the required environment are to be entered. Figure 6.1 shows the input form.



The screenshot shows a web browser window with the title "SB Admin - Bootstrap Admin Template - Mozilla Firefox". The address bar shows the URL "1.1.5.7:8080/Final_BE_Project/p2.html". The page has a dark header with a logo and the text "Automation is the Key...". Below the header, there is a "Back" button. The form contains the following fields:

- OS Name :** Linux
- Environment :** Tomcat
- VERSION :** 7pre7
- FILE :** /root/sample.war
- IP :** 1.1.5.11
- USERNAME :** root
- Root Password :** ****

At the bottom of the form is a "Deploy" button. A small "Loading..." icon is visible next to the "Environment" dropdown.

Figure 6.1: Input Form

After clicking Deploy, a Docker container will be created containing all the dependencies required for a given web application to execute. A port will be allocated to the container. Using this port the user can access the deployed application from the web browser. Figure 6.2 shows an alert box having name of the container and its exposed port. This alert will be displayed after the container creation is successful.

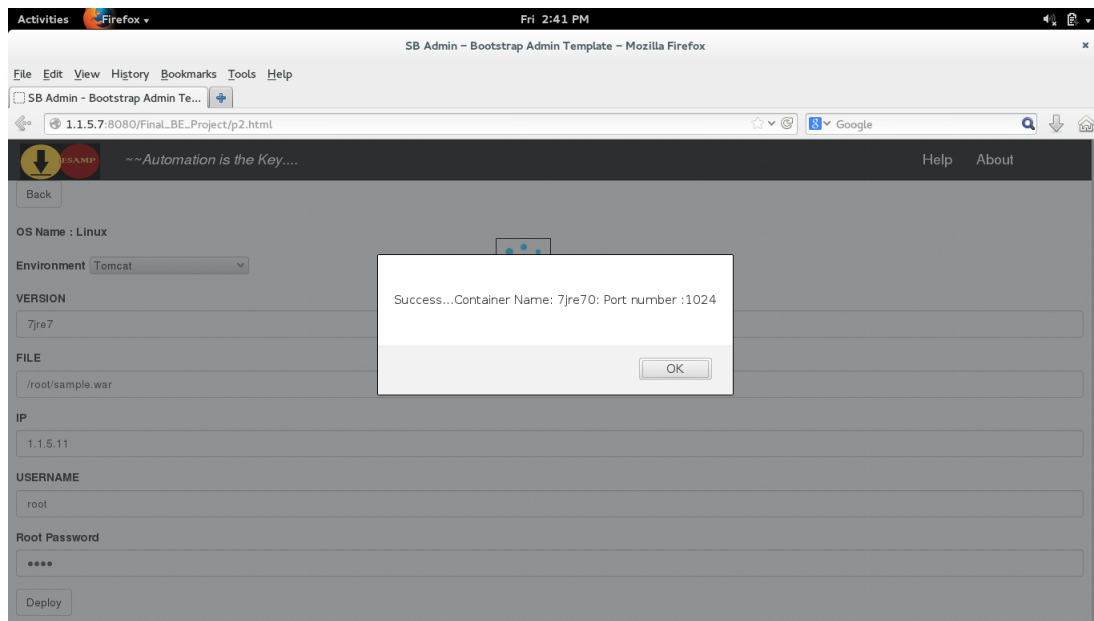


Figure 6.2: Output page

Figure 6.3 shows the web application which the user wanted to deploy. User can access the web application from the browser.

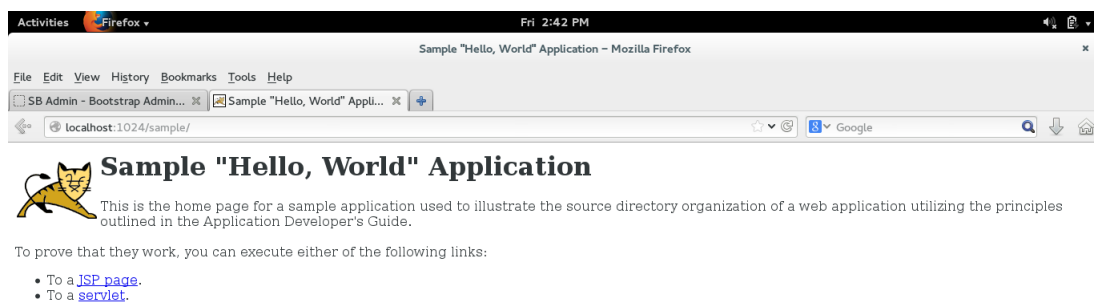


Figure 6.3: Deployed application

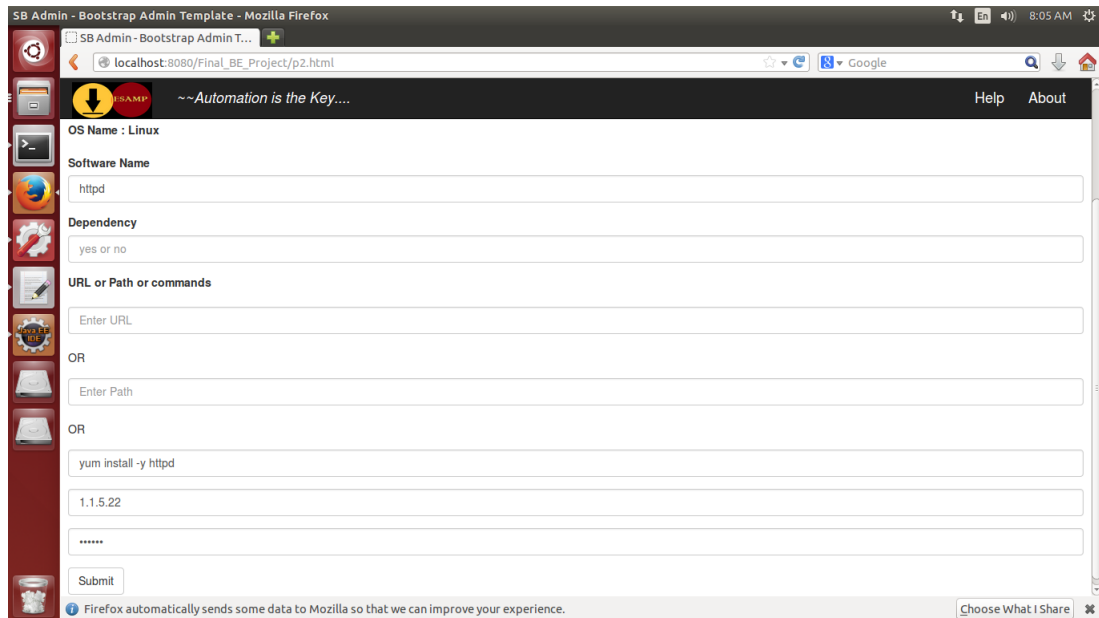
6.2 Chef Results

User can add a new software in 3 ways:

- By providing software name dependency

- By providing path/URL of the installer
- By providing installation commands manually

Figure 6.4 shows the process of adding new software by providing installation commands manually



The screenshot shows the 'SB Admin - Bootstrap Admin Template' in a Mozilla Firefox browser. The page is titled 'SB Admin - Bootstrap Admin T...' and the URL is 'localhost:8080/Final_BE_Project/p2.html'. The main content area is a form for adding new software. It includes a sidebar with icons for various functions. The form fields are: 'OS Name : Linux', 'Software Name' (with 'httpd' entered), 'Dependency' (with 'yes or no' entered), 'URL or Path or commands' (with 'Enter URL', 'OR', 'Enter Path', 'OR', and 'yum install -y httpd' entered), and a version field '1.1.5.22'. There is a 'Submit' button at the bottom.

Figure 6.4: Adding new software

After adding the new software ,it will be reflected dynamically in the software list. Figure 6.5 shows the software list with the newly added software.

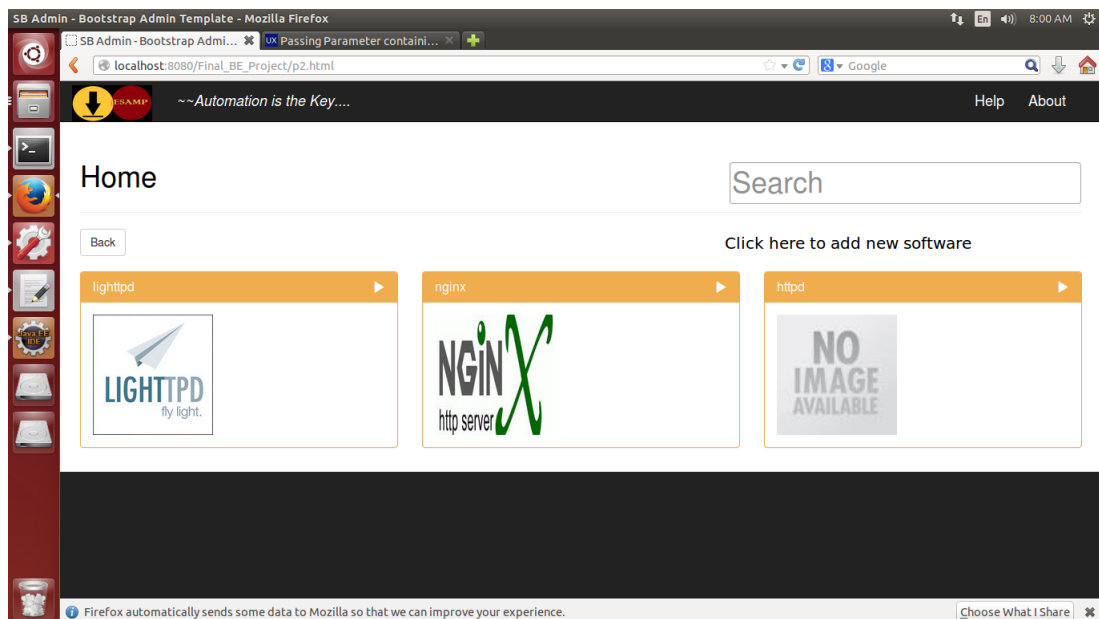



Figure 6.5: Software list

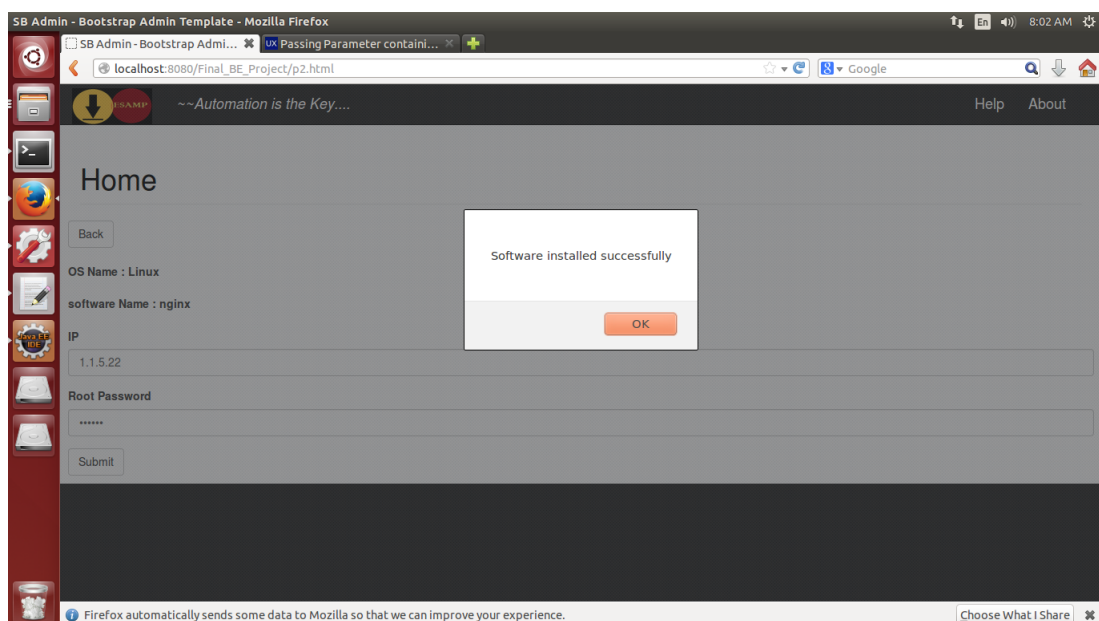
Figure 6.6 shows the form for accepting the details required for installation of a software. Ex.Nginx



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/Final_BE_Project/p2.html'. The page title is 'SB Admin - Bootstrap Admin Template - Mozilla Firefox'. The page content includes a sidebar with various icons, a main area with the heading 'Home', and a form for providing endpoint details. The form fields are: OS Name (Linux), software Name (nginx), IP (1.1.5.22), and Root Password (masked with asterisks). A 'Submit' button is located below the form. A 'Loading...' spinner is visible next to the software Name field. The footer of the page contains a message from Firefox and a 'Choose What I Share' link.

Figure 6.6: Providing endpoint details

After accepting the details, the chef client executes the cook book corresponding to the given software, thus installing it and providing a success message. Figure 6.7 shows the success message.



The screenshot shows the same web browser window as Figure 6.6, but with a success message displayed in a modal dialog box. The message reads 'Software installed successfully' and has an 'OK' button. The form fields and 'Submit' button are still visible in the background, but they are dimmed. The footer of the page remains the same.

Figure 6.7: Install Software

CONCLUSION

Endpoint Software/Application Management Portal has been successfully designed and tested. Endpoint Software/Application Management Portal, a web based system, is useful for both the novice as well as professional users. It will reduce the time as well as the efforts involved in installation of softwares as this process is automated.

The user can get the software or the application environment installed using Chef and Docker tools respectively, thus making the system independent and automated. The required software or application can be easily deployed on the client machine and can be made ready to use without any human intervention.

REFERENCES

8.1 Review of Conference/Journal Papers supporting Project idea

Refer article ,content on following link :

- <https://docs.docker.com>
- <http://www.wikipedia.com>
- <http://learn.chef.io>
- <http://www.youtube.com>
- [http://Steven c.Markey REST in the cloud www.ibm.com](http://Steven.c.Markey.REST.in.the.cloud.www.ibm.com)
- <http://https://www.digitalocean.com/community/tags/docker?type=tutorials>
- <http://http://blog.flux7.com/blogs/docker/docker-tutorial-series-part-1-an-introduction>

APPENDIX

Project paper acceptance by IJIR:

End Point Software/Application Management Portal

Tushar Dahibhate¹, Krish Gambhir², Shruti Kothari³ & Sushant Dharmadhikari⁴

Department of Computer Engineering, Sinhgad College of Engineering, Vadgaon(Bk), Pune

Abstract: The installation of any software requires prior knowledge of the software dependencies which only a person with some technical background will have. The installation process could be tedious and time consuming at times. Endpoint Software/Application management portal will work to manage software and applications on any endpoint. It will be used to automate the process of software installation to reduce the manual efforts required during installation of a software such as- extraction, installation, post-configurations, etc. and will also be time saving.

The software will consist of a Web UI that would be used to take the requirements from the user such as-name of the software, version, and end-point operating system. The extraction and installation of the software will be done by the chef tool. Installation of applications would be done by the docker tool. Rest API will act as an interface between the front end and the back end.

Keywords: Chef Tool, Docker Tool.

1. Introduction :

Endpoint Software/Application management portal will work as a platform to manage software and applications on any endpoint. It will be used to automate the process of software installation. It will reduce the manual efforts required during installation of a software such as- extraction, installation, post-configurations, etc. and will also. The basic components for the tool are as below.

Components:

- Chef client
- Docker Engine- Docker engine must be present on client machine. It is responsible to create docker containers. Docker containers are light weight containers which wrap up a piece of software in a complete file system along with every file system that a software needs to run.
- Docker image is a software which we load into the container. It is created using a Docker file.

be time saving. It also provides the development environment for deploying various applications inside isolated containers in a platform independent manner.

The extraction and installation of the software will be done by the chef tool. Providing the development environment will be done using Docker tool.

2. Literature survey :

There exists some tools for automation of the installation of software. Nineteen is a tool supported only for the Windows operating system which provides various softwares to install and automate the installation process. There is no such tool that supports the linux operating system.

Docker is a tool that can package an application and its dependencies in a virtual container that can run on any Linux server. This helps enable flexibility and portability on where the application can run. Docker uses a client-server architecture.

Chef is a powerful automation platform that transforms complex infrastructure into code. Chef is built around simple concepts: achieving desired state, centralized modeling of IT infrastructure, and resource primitives that serve as building blocks. These concepts enable you to quickly manage any infrastructure with Chef.

3. Block Diagram :

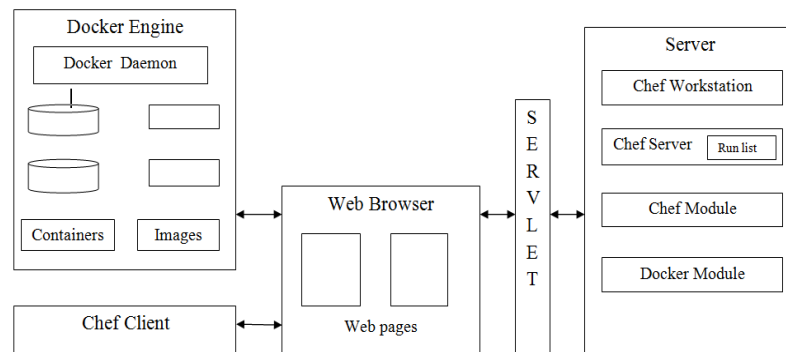
A Docker file is a text file which contains all the commands that a user could call on the command line to assemble an image.

- Docker Daemon – Docker daemon communicates with the Docker engine and manages the Docker containers.
- Web Browser
- Servlet
- Server

The server works as a store to keep the recipes for the software and the client lists and information of the client. The requests are transferred from the client to the server

through the servlet and executed by the chef

client/ docker engine at the client side.



3.1 Architecture

Working of the tool:

- **Software Installation :**

Step 1: Accepting the user input.

- Software Name
- Operating System
- Version
- Credentials
- IP Address

Step 2: Installation of Chef client.

Step 3: Searching if the software is present in the database.

Step 4: Creation of recipe corresponding to the software.

Step 5: Running the chef recipe for installation of software.

- **Application Deployment:**

Step 1: Accepting the user input.

- Application environment
 - Version
 - Path of the file to be executed (file/files must be provided in the form of a tar archive)
 - Credentials
 - IP Address
- c) Direct installer path for the software

Step 2: Searching the Dockerfile corresponding to the development environment.

Step 3: Building the Dockerfile to create a Docker Image.

Step 4: Creating and Starting docker container using the Docker image built before.

Step 5: Copying the file into the container.

Step 6: Providing the details of the container to the user.

4. System Features :

1. UI features- Interface for providing installation details

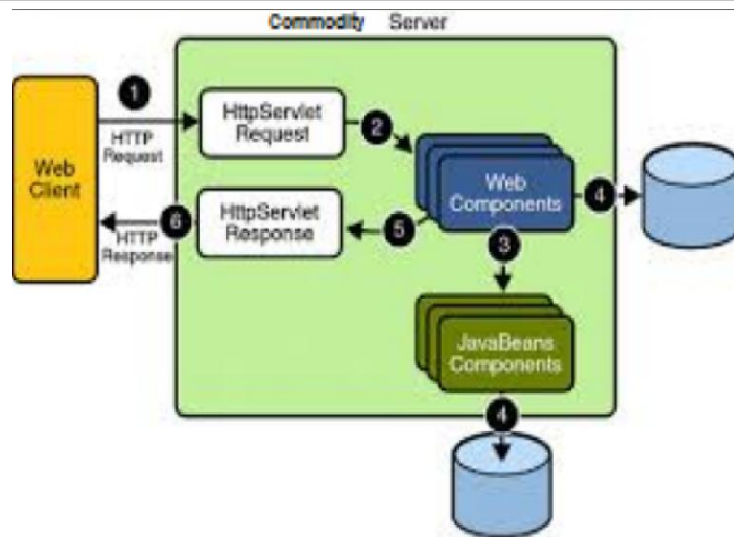
Our system will be based on Client-server architecture. We have employed a powerful commodity server. The server will be responsible for accepting client requests and giving them results accordingly.

There are three ways provided for the user to put his request up for the server:

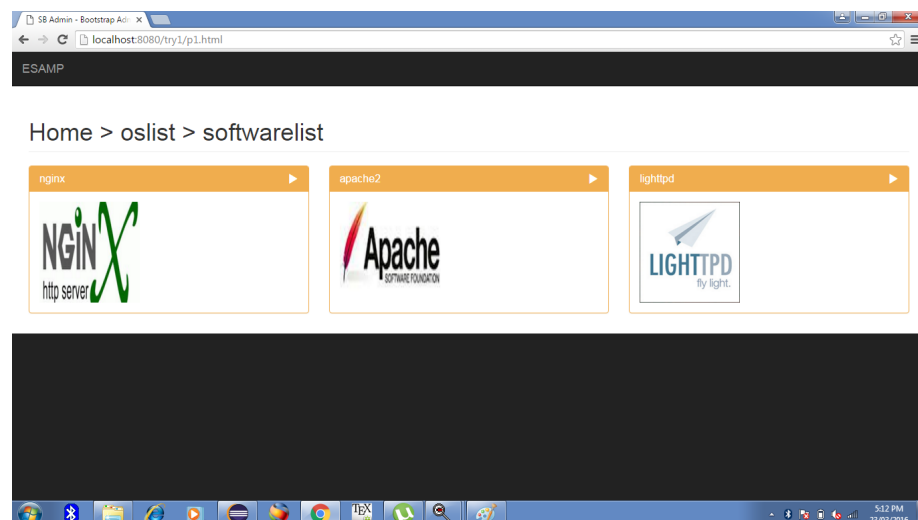
- a) Name of the software
- b) URL for the software

Imperial Journal of Interdisciplinary Research (IJIR)

Vol-2, Issue-6, 2016

ISSN: 2454-1362, <http://www.onlinejournal.in>

4.1. A Web Server Workflow



4.1.b UI sample

2. Software Installation

This feature will provide the option for the selection of the OS at the endpoint.

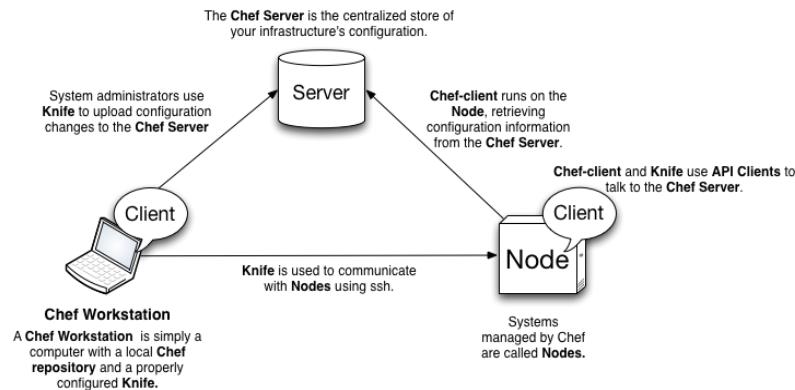
Depending on the OS selected the user then will provide the software name, IP address and other credentials.

2.1 Chef Client

The chef client will take the responsibility to install the software automatically by running the corresponding recipe.

2.2 Chef Workstation

The chef-workstation will be responsible to manage the nodes in the network.



4.2.a Chef Workflow

3. Application Environment provision

- User will provide the environment and the version along with the path of the file to be deployed and the user credentials.

3.1 Docker functionality

Users need to have Docker installed on their system with remote API feature enabled and started. As Docker will be installed on the client side itself, it will have all the containers as well as the Docker images. Server will be responsible to fire the API requests to the Docker daemon installed on the client machine, which in turn will be responsible for building the image and spawning the containers. The remote API provided by Docker is RESTful. Once the user provides all the details in the UI, server will be responsible to fire appropriate API requests to the Docker daemon present on the client. Docker daemon will then communicate with Docker engine to build the Docker image and spawn the container using it. The container spawned will have the necessary environment that the user desires.

5. Conclusion:

The End point Software and Application Management Portal has been successfully designed and tested. The user can get the software or application environment installed through the Chef and Docker Tools, thus making the system independent and automated. The required software or application can be easily deployed on the client machine and can be made ready to use without any human intervention.

References:

- [1] www.wikipedia.org
- [2] www.youtube.com
- [3] www.learn.chef.io
- [4] Steven c. Markey REST in the cloud www.ibm.com
- [5] docs.docker.com
- [6] www.digitalocean.com/community/tags/docker?type=tutorials
- [7] blog.flux7.com/blogs/docker/docker-tutorial-series-part-1-an-introduction