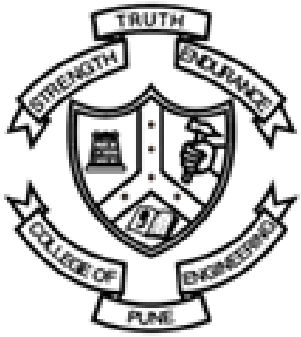
College of Engineering, Pune

An Autonomous Institute of the Government of Maharashtra



Department: Computer Engineering & Information Technology

COEP Aspyra

A container based cloud platform for end-to-end deployment of student projects

Students:

Shubham Patil
111203062

Mihir Pattani
111203070

Shubham Zapate
111203069

Inhouse Mentor:

Prof. Satish Kumbhar
Assistant Professor Computer Department
College of Engineering Pune

External Mentors:

Naveen Ramaswamy
Staff Engineer
VMware Ltd.

Prayas Gaurav
Team Manager
VMware Ltd.

Yogesh Gaikawad
Staff Engineer
VMware Ltd.

INDEX

Sr No.	Heading	Page No.
1.	Abstract	2
2.	Literature Survey	3-16
	1. Linux Containers	3
	2. Virtual Machines	3
	3. Docker Project	5-7
	a) Advantages	
	b) Architecture	
	4. Google Kubernetes	7-9
	5. Cloud Deployment	10
	6. HashiCorp	10
	7. Custom Dockers	10-11
	8. Problem Statement	11-12
	a) Statement	
	b) Stage wise objectives	
	9. Motivation behind the project	12
3.	System Design	13-16
	1. Front End	
	2. Back End	
	3. Deployment platform	
4.	Implementation	17-19
	1. System requirements specification	17
	a) Hardware requirements	
	b) Software requirements	
	2. Current status of implementation	18
	3. Work to be done	18
	4. Additional features	19
	5. Time table	19
5.	References	20

ABSTRACT

In our project we intend to create a tool for end-to-end deployment of user applications on a cloud platform using customized Docker containers.

Cloud computing has enabled companies to make efficient use of their IT resources. Deploying applications on a cloud platform has advantages such as scalability, high availability and better performance. Today a number of companies are moving from virtual machines to Linux containers for deploying applications on their cloud platforms. Virtual machines are resource intensive while containers provide a lightweight virtual environment.

Containers group and isolate a set of processes and resources such as memory, CPU, disk space, etc., from the host and any other containers. In order to deploy an application on a container, a developer needs to have all dependencies installed on that container instance. After installing these dependencies, this container is deployed on some suitable platform which is in most cases a cloud-based platform.

Our project will provide a web-based tool which will enable the user to package his application within a custom container image. The tool will further automate the process of deploying this container image on our cloud platform. This cloud platform will be pre-configured by us to manage Container clusters using Google's Kubernetes project.

LITERATURE SURVEY

Containers:

Container is an operating-system-level virtualization environment [5], which is used to run a number of isolated systems on a single control host. Linux containers are built on the concept of kernel namespaces. Namespaces are used to create an isolated container that has no visibility or access to objects outside the container. Processes running inside the container appear to be running on a normal Linux system although they are sharing the 0075nderlying kernel with processes located in other namespaces. There are two main types of containers:

1. System container - This container acts like a full-fledged OS and runs OS processes like init, syslogd etc.
2. Application container - This container runs a single application and uses limited resources.

Both these types of containers are useful in different situations. System containers can be used as substitutes for virtual machines while Application containers are generally used to run isolated programs like database servers or feedback softwares.

Containers isolate resources such as RAM, CPU, disk etc. from the host ensuring that the applications running on it are totally isolated from the host. Containers have a number of similarities to virtual machines which are the most used virtualization environments today.

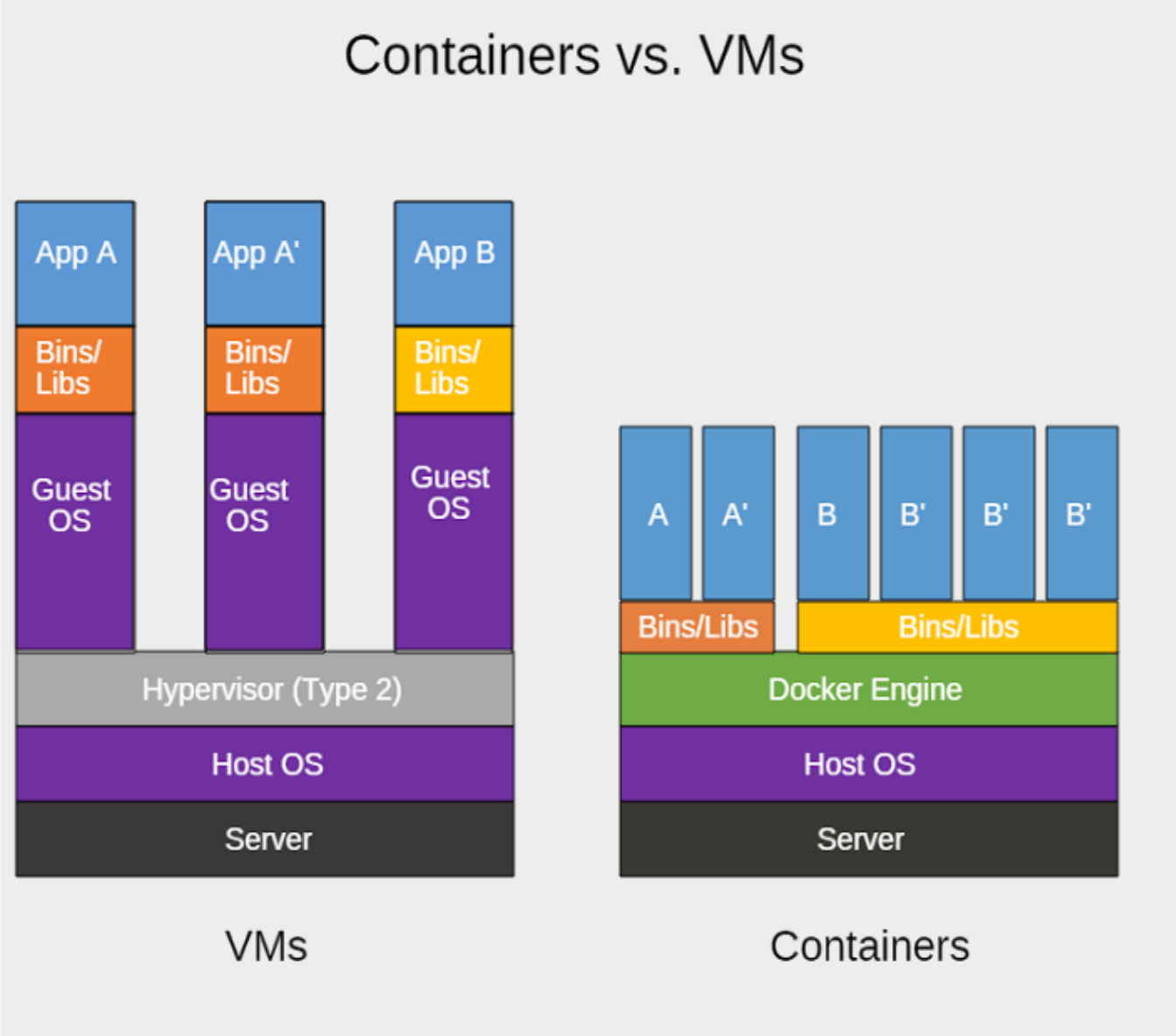
Virtual Machines:

A virtual machine (VM) is a virtualization environment that is installed on a software, known as a hypervisor, which imitates dedicated hardware [6]. No difference is visible to the end-user between an OS running as a virtual machine and one running on dedicated hardware. A virtual machine can support individual processes or a complete system depending on the abstraction level where virtualization occurs. Today virtualization is being applied at different levels in a computing system from operating systems to storage as well as networks. This has ensured improved performance and better usage of available resources.

VM’s provide higher levels of isolation as compared to containers. This is primarily because deploying virtual machines involves installing a guest operating system onto the hypervisor. But this also results in greater overhead in the form of the guest OS’s kernel. Containers, on the other hand, make use the host OS’s kernel for creating an isolated virtualized environment and hence are more ‘light-weight’ than VM’s.

This advantage of containers over VM’s plays a significant role in optimizing use of resources especially when the host OS is highly hardware intensive. As a result of this more number of containers can be deployed on the same host as opposed to virtual machines (varying from 2 to 6 times the number of VM’s).

The following figure [4] highlight the major difference between VM's and Containers:



Today intensive research is being put into the field of containers. Huge corporations like Google and Twitter are today investing heavily in developing container technology. The Docker project and Google's Kubernetes project are two open source projects which have caught huge traction recently. We shall be using these softwares for supporting different modules of our project.

The graph below compares the resources used by containers and virtual machines.

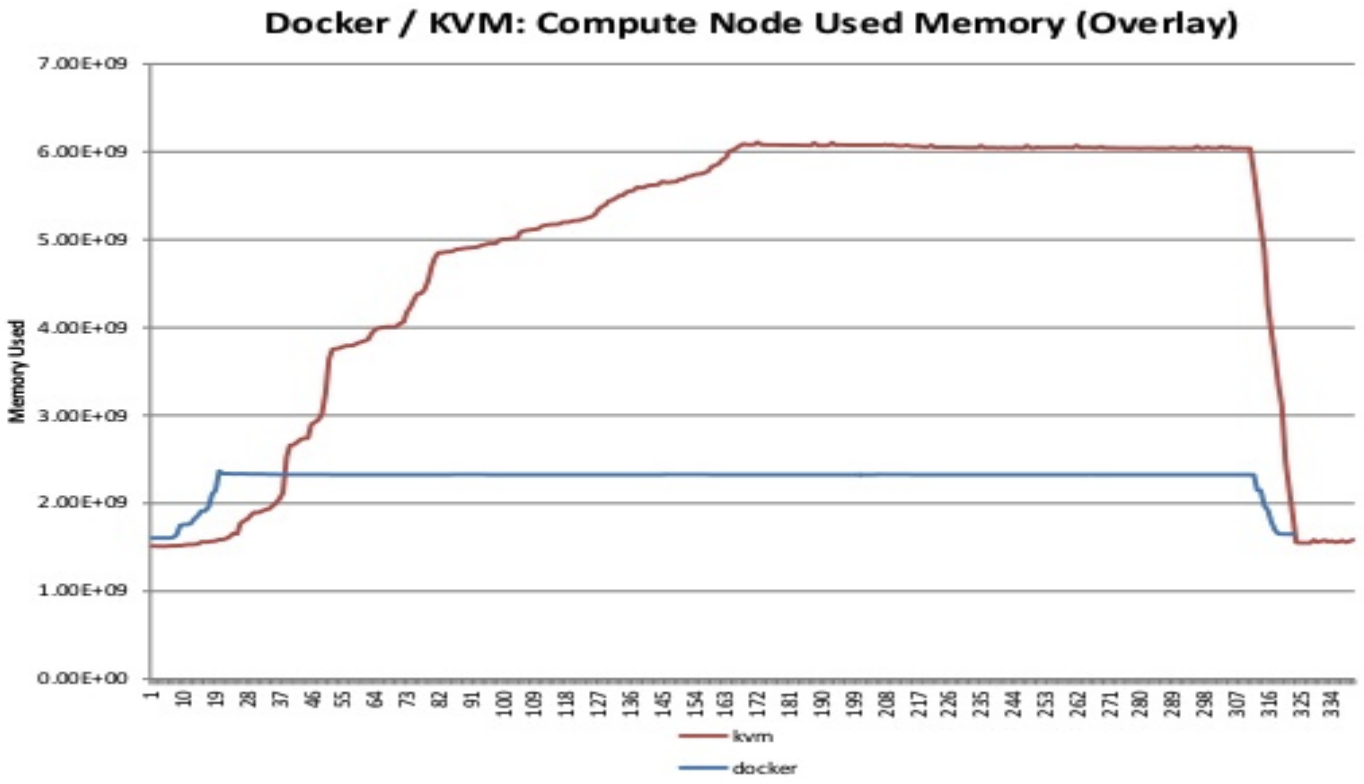


Figure: Docker vs KVM

(Source: https://opensource.com/sites/default/files/images/business-uploads/openstack-docker-1.png)

Dockers:

Docker is an open-source platform which streamlines and automates the [7] process of deploying application containers. Making use of the resource isolation feature offered by the linux kernel, Docker allows several independent containers to run on a Linux instance. Docker allows the developer to package the application source code with all of its required dependencies into a single deployable container making the process of shipping, running and building the application easier.

Advantages:

1. Encapsulating the application's source code with its [4] dependencies greatly reduces the effort required to deploy the application in any other environment or infrastructure.
2. Docker provide a light-weight virtual environment [9] to manage and run several container instances simultaneously which results in better utilization of the underlying hardware infrastructure.

3. Docker ensures high availability of services, ensuring low recovery point objective and low recovery time objective (RPO and RTO). Docker also ensures higher availability by containing the errors as they arise and containers being light-weight induces less overhead

4. The use of docker simplifies the process [8] application development by ensuring that applications run identically in development, test and production environments.

5. Dockers facilitates portability,creation and deployment of [4] container instances across multiple cloud services and operating systems including non-linux operating systems.

Architecture of Docker:

Docker operates on a client-server architecture. The docker client [10] communicates with docker daemon which is responsible for running,building and distributing docker containers. Both the docker client and daemon processes can run on the same machine or on two different systems. These two processes communicate via sockets or through a RESTFUL API.

The Docker [10] Engine consists of:

1. Docker Daemon:

   The docker daemon runs on a host machine.The user does not directly interact or communicate with the daemon, but it does so via the docker client.

2. Docker Client:

   The docker client is in the form of a docker binary which acts as a user-interface between the user and the docker engine. It accepts commands and requests from the user and communicates it back and forth with the docker daemon.

3. Docker Images :

   A docker image is similar to a template with read-only properties. An image could contain a CentOS operating with apache server installed on it.

4. Docker registries:

   Docker registries are responsible for storing and providing various container images. Docker registries can be public or private storages of these images. A user can upload to these registries or can download a suitable container image from the registry.

5. Docker Containers:

   Each Docker container is an isolated secure application platform similar to linux containers. Each container is created from a Docker image and holds everything that an applications needs to run.

Docker [10] engine architecture:

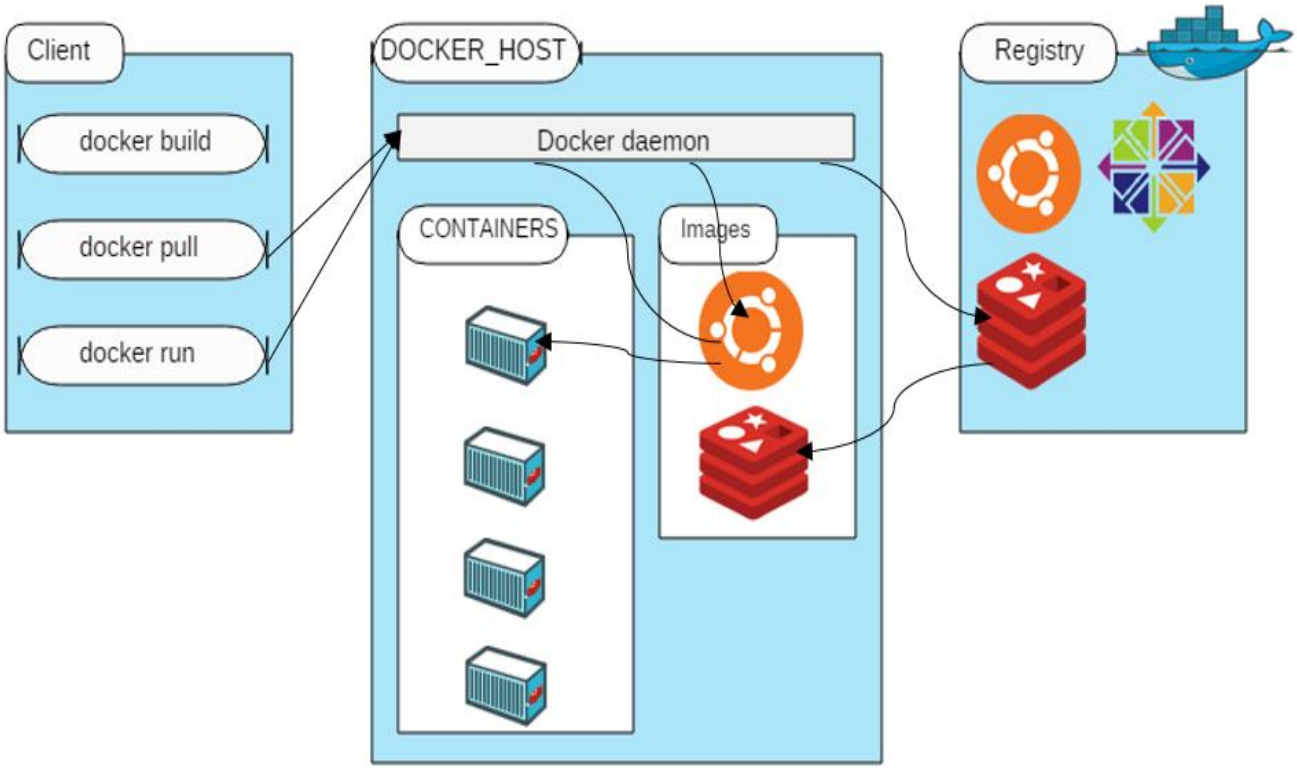


Figure: Docker Architecture

Kubernetes:

Kubernetes is an open-source system which [11] provides the functionality for managing container application in a clustered environment. Kubernetes provides an easy and efficient way to manage related, distributed components spread across varied infrastructure. Kubernetes relies heavily on primitives, which is a set of building blocks for the system and they look after the maintenance, deployment and scaling of the applications.
The components which build up Kubernetes are loosely coupled with low cohesion to handle a variety of workloads. The components of Kubernetes system are mainly subdivided into two categories:

1. Master Server components
2. Minion Server components

1. Master Server Components:

Master hub: Master hub is the main controlling unit in a Kubernetes cluster. The master hub runs a set of services which manage the cluster's workload and also the communications across the system. The master hub is the main management hub for the administrator. The master hub is made up of: 1) Etcd 2) API server 3) Controller manager server 4) Scheduler server.

I. Etcd: Kubernetes uses a globally available storage area to store all the configurations which is then used by each node in the cluster. The etcd component of the master hub looks after maintaining and storing the configuration data which is then used by each node in the cluster.

II. API server: API server is a process which the master server runs and is critical to the operation of the entire system. This is the main management point of any Kubernetes cluster and it allows the users to configure the workload in a Kubernetes cluster along with the organizational units. The API server is also responsible for maintaining the integrity between etcd store and the service details of the deployed containers. The API server makes use the RESTful architecture to communicate with different tools and libraries.

III. Controller Manager server: The controller manager server handles the replication processes which are created to perform the replication tasks in a cluster. The configurations and details of these replication tasks are written to the etcd store where the Controller Manager server continuously watches for changes. Whenever the controller manager recognizes a change in the etcd store, it reads this new information and runs a replication procedure to fulfill the requirements.

IV. Scheduler Server: The scheduler server is the process which assigns and distributes the workload across the nodes in a cluster. The scheduler process performs the following tasks:

a. Firstly, it reads the service's operating requirements.
b. It then analyzes infrastructure environment, calculating the total resources provided by each server along with the resources being used up by the already assigned workloads.

c. Finally, it then places the workload on a server such that the processing is feasible. The scheduler server also needs to make sure that the workloads scheduled on each node do not exceed the available resources present on that node.

2. Minion Server Components:

The nodes responsible for carrying out the actual work are known as minion servers. The responsibilities of the minion server are:

a) Communicate information to and fro from the master
b) Configuring and maintaining a stable state of the network system for the containers
c) Execute the workloads assigned to them.

Minion server consists of two main services:

a) Docker service
b) Kubelet service

I. Docker Service: The docker engine is the basic requirement for the minion servers to function. The docker engine is used to run multiple application containers simultaneously in an isolated and lightweight environment. Each unit of workload in its lowest level is run using a series of containers which are then deployed.

II. Kubelet Service: The kubelet service is responsible for providing a communication link between the minion server and the rest of the cluster group. The kubelet service is responsible for handling the bi-directional data transfer between the minion and master server. The kubelet also interacts with the etcd store to read and write configurations.

The architecture of Kubernetes:

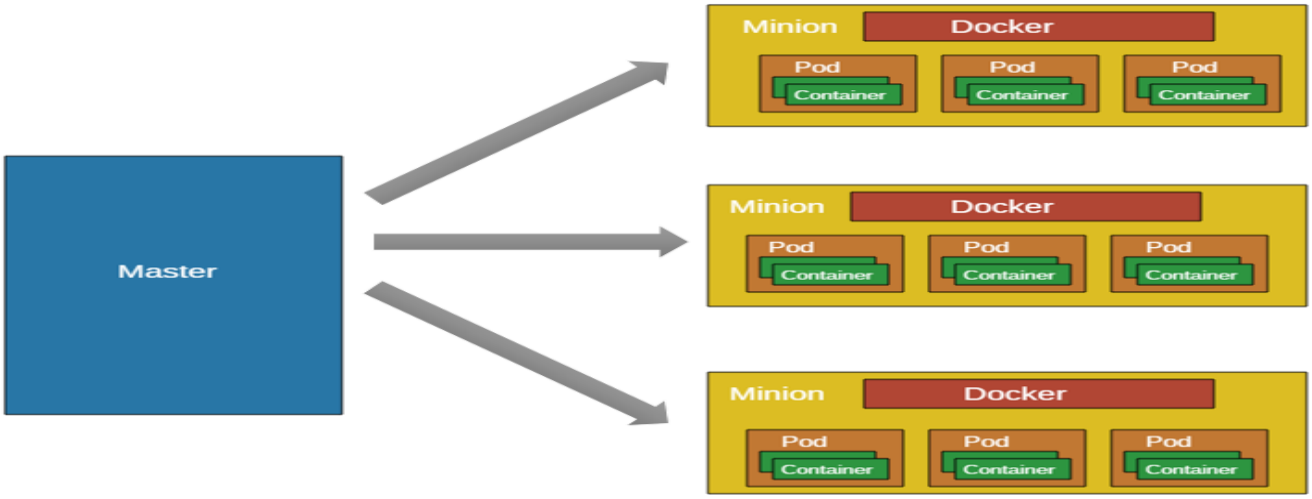


Figure: Kubernetes architecture

Cloud Deployment:

Our project involves deploying the Kubernetes project over a cloud infrastructure for better deployment of the application containers. The data center is in process of deploying a cloud on the college server and we intend to make use of that for our project. In case of any discrepancies in configuration we intend to use OpenStack for this purpose. OpenStack is a free and open-source software platform for cloud-computing, mostly deployed as an infrastructure-as-a-service (IaaS). This deployment is essential as it ensures optimum usage of hardware resources as the Kubernetes project does not provide features like thin provisioning etc.

Other softwares similar to our project:

HashiCorp is an organization which provides a number of open source products which help developers and users maximise their agility and productivity. HashiCorp's products particularly Atlas and Packer tools have similar functionalities as that of our project. Another project similar to our work is the DEIS project which like HashiCorp provides packages for efficient deployment of application on a cloud platform. Vagrant is a software which provides some interesting functions and has some resemblance to our software.

1. **Packer:** It is an open source tool for creating identical machine images for multiple platforms from a single source configuration. Packer is lightweight and runs on all major operating systems like Windows, different Linux distributions, Mac etc. It can create images for multiple platforms in parallel. Packer uses tools like Chef or Puppet to install software onto the image. A *machine image* is a static unit, containing a pre-configured OS and installed software which is used to quickly create new running machines. It creates and stores images to be run on Docker (container images), VMware, AWS, OpenStack etc.

2. **Atlas:** Atlas is a combination of 4 tools - Vagrant, Packer, Terraform and Consul. It manages creation of Vagrant boxes , creating different images for different platforms using Packer, automates management of hardware infrastructure and deployment of the images with Terraform and manages health of all deployments using Consul.Overall Atlas platform incorporates creation, deployment and management of user applications as well as development environments.

   Even our software aims at creation, deployment and management of different user applications. The difference from Atlas is as follows:

   a) Atlas encompasses it's different functionalities by using different packages. As a result the user has to use each of those services for proper deployment of their applications.

   b) As opposed to that our application does not require user to do many configurations and provides a one-shot application deployment solution. The user can use a simple 3 step GUI based process for efficient deployment of his application.

c) Our application relies heavily on Docker containers (only) for deployment of the applications as opposed to Atlas which has support for Docker containers, VM's and also for Vagrant.

d) We shall also be providing support for direct GitHub integration of source code as well as provide features to create, store and deploy Development Environment themes.

3. **Vagrant:** Vagrant creates easy to configure, reproducible, and portable work environments. These machines are provisioned on top of products like VMware, AWS, RackSpace etc. Using a single Vagrantfile, one needs to run the command vagrant up and everything is installed and configured for you to work. These development environments can easily be reproduced on different OS's like Linux, Mac OS X, or Windows and all deployments will have the same configurations and all necessary dependencies installed. Vagrant also provides us with a disposable environment with consistent workflow for developing and testing infrastructure management scripts. This is pretty useful in quickly testing things like Chef cookbooks and local virtualizations on VMware etc.

   Vagrant's functionality is similar to our extended idea of allowing users to create development environments using our container customization tool. The difference lies in the implementation. Our tool uses a GUI to take user's requirements and then creates a container image for the same as opposed to the concept of Vagrant file.

Problem Statement:

To create a tool for end-to-end deployment of user applications on a cloud platform using customized Docker containers.

Explanation:

1. Problem statement involves three parts:
    a. Customized Docker containers:
        i. The first part of the project that we shall create is the Docker container customizer.

        ii. What this code will do is, it will create a Docker container image with the required packages installed in it. The user's executable is also packaged into this image.

    b. Cloud Platform:
        i. We shall be making use of the cloud platform that is currently being deployed on the college server.

        ii. If there are any conflicts with that we shall be deploying our own platform using OpenStack.

        iii. This cloud platform will be used to deploy the generated Docker container image.

        iv. The deployment and lifecycle of the container image will be managed using Google's Kubernetes project.

    c. End-to-End deployment of applications:
        i. This means the tool shall manage the deployment, running as well as deletion / removal of the user's application.

2. So we shall be asking the user to upload his application's executable file to our **web-based tool** along with its dependencies and then package this executable along with its dependencies in a Docker container image which will be deployed on the college server's cloud platform.

3. One more part of the project involves the management tool which manages lifecycle of the container (pausing, deleting etc.)

Objectives of the project:

1. Create a simple GUI based application deployment system which is user friendly and does not require users to do a lot of configurations.
2. This application will be deployed on the COEP server and will be available for COEP faculty and students to use for development, testing and deployment of their applications.
3. The overall design architecture of the tool will be such that it shall ensure efficient use of the resources in the datacenter while aiding the faculty and students with their work.

Motivation Behind the Project:

Today virtualization technologies and cloud computing have taken the I.T. world by storm. A number of solutions today are being provided using cloud technologies. Cloud and virtualization are technologies that are not only being extensively used but also researched largely. The huge research potential in these fields was one of the major factors which influenced us to undertake a project based on virtualization and cloud computing.

We also wanted to develop an application that would be useful to large number of people especially those from our college. Our application shall provide an easy and robust platform for the students as well faculty of our college to shift their development work to the cloud platform. Using a cloud platform has a number of advantages like presence of large amount of resources, server consolidation, high availability etc. Also the programs shall not run on the local machines ensuring that the programs do not cause any damage to the local systems.

Apart from this we are using large open source projects like Docker and Kubernetes in this project which we believe shall motivate our juniors to look into such projects and also use them. The field of cloud computing is ever-changing and our project could be continued further by our juniors who could more features ensuring that this application of ours becomes a full-fledged long term project.

SYSTEM DESIGN

The system is broadly divided into 3 sub-systems:

1. Front end
2. Back end
3. Deployment platform

Front End:

The front-end part of the system basically deals with taking input from the user, processing this input and generating an output which can be processed by the back-end part.

Input from the user:

1. Executable of his / her application

2. Packages / services required for his application to run. (Eg. - Php, MySQL, Apache server etc.)

3. Amount of resources required for his application to run. (Eg. - CPU, RAM etc.)

Parts and working of Front End:

1. Package selection interface:

    a. When the user inputs the packages required for his application to run, this interface collects these requirements as one list which is then processed by the query formulation interface.

2. Query formulation interface:

    a. The query formulation interface will basically group all commands required to install necessary packages to the Docker container into a single object.
    b. It also stores the information about required resources (RAM etc.) in that query object.

3. Query and Executable forwarding:

    a. This part of front-end is responsible for sending the executable file of the user's application as well as the query object to the server (back end).

    b. This data is sent using REST API.

Back End:

Back-end of the tool basically involves the server side processing of the user's demands. This part of the tool is responsible for creating the Docker container image and then sending it to the deployment platform.

Input from the front end:

1. User's executable file
2. Query object containing commands for installing the dependencies as well as the details of required resources such as RAM, disk etc.

Parts and working of back-end:

1. Container creator:
    a. This part decodes the query object and obtains all commands to install the dependencies.

    b. It creates a script, using the received commands, which will be run by the Docker engine to create the customized Docker image.

    c. This script and the executable file is sent to the docker engine.

    d. After receiving the image from the Docker Engine, it generates a set of commands required to deploy this image of the deployment platform.

    e. After this the image and commands are sent to the deployment platform.

2. Docker Engine:
    a. It will run the scripts it receives from the Container Creator.

    b. It creates the container along with all dependencies installed in it packaged with the executable file.

    c. The Container Engine pulls the required dependencies from a container image registry.

    d. The image is then sent back to the Container creator.

Deployment Platform:

This platform deals with the actual deployment of the container image on the hardware.

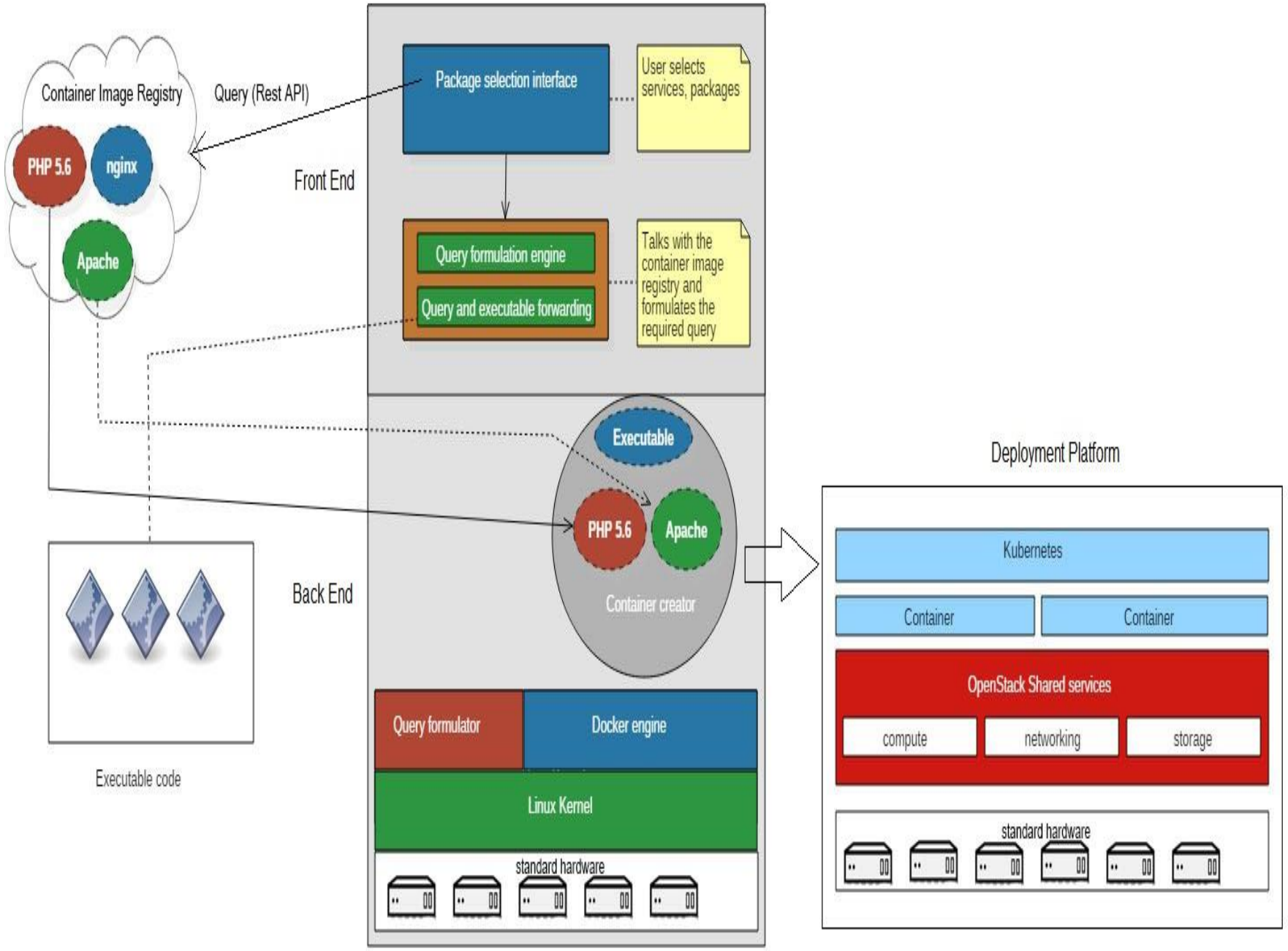
Input to this platform:

1. Container Image
2. Commands to deploy the image

Parts and working of Deployment Platform:

1. Kubernetes:
    a. Commands passed from Back End are used to manage the image appropriately.
    b. It interfaces with the cloud platform to properly deploy, delete, pause etc. the Docker container.

2. Cloud Platform: Actually deploys, removes, accesses the container on the physical hardware.


Container Image Registry
Query (Rest API)
PHP 5.6
nginx
Apache
Front End
Package selection interface
User selects services, packages
Query formulation engine
Query and executable forwarding
Talks with the container image registry and formulates the required query
Executable
PHP 5.6
Apache
Container creator
Back End
Executable code
Query formulator
Docker engine
Linux Kernel
standard hardware
Deployment Platform
Kubernetes
Container
Container
OpenStack Shared services
compute
networking
storage
standard hardware

IMPLEMENTATION

System Requirements Specification:

1. Hardware requirements:
    a. Server:
        i. RHEL v5 with 32 GB RAM, Dual processor (Intel Xeon e5 2650) with hot plug drive of 600GB SAS Technology (15K RPM) with RAID 1(X 2)
    b. Client:
        i. Minimum dual core processor. 4GB RAM.
        ii. Hard disk 500GB SATA with 7200rpm

2. Software requirements:
    a. Server softwares:
        i. Red Hat Linux / Ubuntu Server OS
        ii. OpenStack Cloud platform
        iii. Kubernetes
        iv. Docker
    b. Development requirements
        i. Eclipse IDE
    c. Languages used:
        i. Front end: HTML5,CSS3, JavaScript
        ii. Back end: PHP, Python, Shell(bash) script, MySQL

Current Status of Implementation:

As of now, we have thoroughly studied and understood the concepts and theory required for our project. Our focus has mainly been on the research and learning phase. We have successfully completed the following tasks:

1. Firstly, we studied and covered important topics related to our project like virtualization and cloud computing.

2. Based on our research we then zeroed in on the appropriate set of services, programming languages, softwares to be used for the implementation of our project. Google's Kubernetes project was chosen to manage container clusters on the cloud deployment platform and the docker engine will be used to manage the active container instances.

3. After deciding the appropriate tools and software we put efforts to get well versed with the development environment. We then took tutorials for the docker engine and Kubernetes to get hands on training and also to understand the working and functionalities of these services.

4. We then designed our system architecture and also decided the overall flow of our tool.

Work to be done:

The following are the tasks which are yet to be completed in a chronological order:

1. Container customizer back end: Developing the back end which is responsible for creating custom container images packaged with the executables provided by the user.

2. Container customizer user interface: Building the user interface for our tool which will allow the user to select the packages, services and also the hardware requirements for his project.

3. Cloud Configuration: Configuring the cloud deployment platform by installing openstack on it.

4. Kubernetes: Installing and configuring Kubernetes on openstack.

5. Interface between Kubernetes and customizer: Streamlining the interface between the kubernetes and the back end of our tool to facilitate deployment of containers on our cloud platform.

6. Combining all modules and testing: Integrating and connecting all the modules to achieve a complete end to end working of our tool.

7. Alpha Testing: The build of the tool after integration will be tested by the team-members, project guide, friends to uncover bugs.

8. Add-Ons and testing: Integrating and adding the additional features.

9. Beta testing: The final build after adding the additional tools will be tested by the college community to discover additional bugs.

10. Removal of bugs: The bugs discovered during the testing process will be fixed to make the application function smoothly.

Additional Features:

1. Github support: We further plan to provide github support to our tool which would fetch the user's code directly from their github repository.

2. Support for container cluster: A container cluster is a group of container instances each running a single service, running simultaneously to satisfy a common set of requirements. We plan to incorporate the support for container clusters on our cloud deployment platform to ensure availability, scalability and easier management.

3. Development environment support and themes: Adding this module will allow users to create their development environment and run them in the containers to develop their apps. These development environments could then also be stored as themes to be replicated for later use.

4. Support for defining security policy: We further plan to add support for allowing the user to enforce security policies to provide extra security.

Time Table:

Task	**Dec.**		**Jan.**			**Feb.**			**Mar.**			**Apr.**	
	20	30	10	20	31	10	20	29	10	20	31	10	20
Container customizer back end													
Container customizer user interface													
Cloud configuration (Open Stack)													
Kubernetes configuration													
Interface b/w Kubernetes and customizer													
Combining all modules and testing													
Alpha testing													
Add-Ons and testing													
Beta testing													
Removal of beta testing bugs													

REFERENCES

1. Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio ‘An Updated Performance Comparison of Virtual Machinesand Linux Containers’

2. Smith, J.E.; Nair, R., "The architecture of virtual machines," in *Computer* , vol.38, no.5, pp.32-38, May 2005

3. Chen, P.M.; Noble, B.D., "When virtual is better than real [operating system relocation to virtual machines]," in *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on* , vol., no., pp.133-138, 20-22 May 2001

4. Steven J. Vaughan-Nichols for Linux and Open Source Web blog post ‘What is Docker and why it is so darn popular?’

5. Bob Reselman Web blog post ‘A Developer’s Journey into Linux Conrainers’ 24 Sep 2015

6. Margaret Rouse; Brian Kirsch Web blog post ‘Virtual Machine Definition’ Oct 2014

7. O'Gara, Maureen (26 July 2013). *"Ben Golub, Who Sold Gluster to Red Hat, Now Running dotCloud"*. SYS-CON Media*. Retrieved 2013-08-09*

8. InstallAnywhere 2015 Help Library ‘Advantages of Using Docker Containers’ Aug 2015

9. Avi Cavale answer on Quora ‘What are the benefits of using software containers (such as Docker) vs the old approach of managing multiple VMs (using tools such as Puppet) and vice versa?’

10. ‘Docker Introduction Understanding the architecture’
    Retrieved from https://docs.docker.com/engine/introduction/understanding-docker/

11. Justin Ellingwood Web blog post ‘An Introduction to Kubernetes’ 1 Oct 2014