# Private-Engineered Cloud Platform

Mahney Mohsen
*Electrical Engineering Department, Kafrelsheikh University, Egypt, mahanaelbna@gmail.com*

*Supervisor:* Dr Ghada Hamissa, Lecturer in Computer Engineering & Systems Branch,

*Electrical Engineering Department, Kafr El-Shiekh University, Egypt. ghada.hemesa@eng.kfs.edu.eg*

*Abstract– Cloud computing is the evolution of a variety of technologies that have come together to change an organization's approach for building an IT infrastructure. The infrastructure of the Platform is built based on common computer network structure. In our research, we design a private-cloud based platform serves, called PLATRAIN. It serves mainly the learning organizations -such as Universities. It provides a digital infrastructure to enable the organization to detect, analyze and improve its Workflow. PLATRAIN based on the cloud architectural system which is private only for the concerned organization. It uses several technologies such as edge computing (IoT), cloud computing services and virtualized environment for users, applications, projects and other services (such as Remote Learning). Our PLATRAIN platform obtained the approval of intellectual property rights from Information Technology Industry Development Authority, Intellectual Property Rights Protection Office at 6/12/2021. It is now active as a global site for testing at the IP address: 20.39.198.54; for a year (from 1/7/2022).*

***Keywords—Cloud computing, Network Topology, Platform design, Cloud Models, PPDIOO, and Docker.***

## I. Introduction

Cloud computing is the evolution of a variety of technologies that have come together to change an organization's approach for building an IT infrastructure. Cloud computing term describes a variety of different types of computing. Concepts that involve many computers connected through a real-time communication network (typically the Internet) [1]. Cloud computing relies on sharing of various resources (networks, servers, storage, applications, and services) to achieve coherence and economies of scale, and gives the highest interest to how to maximize the effectiveness of utilization of the shared resources. The general structure of cloud model presents at Figure 1.

### *A. Private Cloud*

Private clouds are loosely defined as cloud environments solely dedicated to a single end user or group, where the environment usually runs behind that user at group's firewall. All clouds become private clouds when the underlying IT infrastructures dedicated to a single customer with completely isolated access. But Private cloud no longer have to be sourced from on premise IT infrastructure organizations are now building private clouds on rented, vendor-owned data centers located off-premises, which makes any location and ownership rules obsolete this also led to a number of private cloud subtypes.

### *B. Cloud Services*

Cloud computing consists of three distinct modules of computing services delivered remotely to clients via the internet. Clients typically pay a monthly or annual service fee to providers, to gain access to systems that deliver software as a service, platforms as a service and infrastructure as a service to subscribers. Clients who subscribe to cloud.

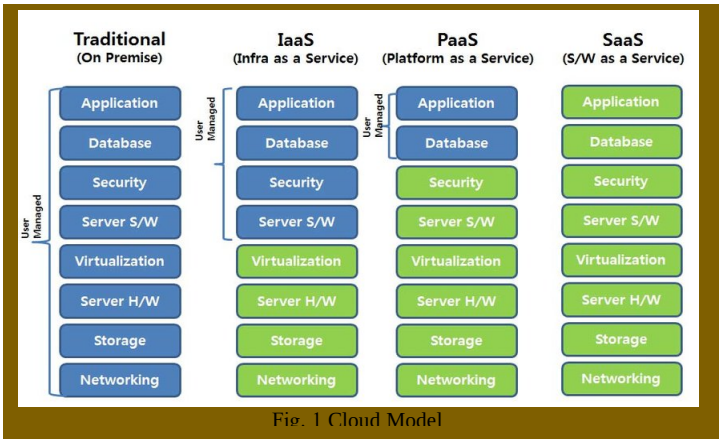


Fig. 1 Cloud Model

### *C. SaaS (Software as a Service)*

SaaS provides clients with the ability to use software applications on a remote basis via an internet web browser. Software as a service is also referred to as "software on demand". Clients can access SaaS applications from anywhere via the web because service providers host applications and associated data at their location. The main benefit of SaaS, is a lower cost of use, since subscriber fees require a much smaller investment than what is typically encountered under the traditional model of software delivery. Licensing fees, installation costs, maintenance fees and support fees that are routinely associated with the traditional model of software delivery can be virtually eliminated by subscribing to the SaaS model of software delivery. Examples of SaaS include: Google Applications and internet based email applications like Yahoo! Mail, Hot-mail and Gmail.

### *D. PaaS (Platform as a Service)*

PaaS provides clients with the ability to develop and publish customized applications in a hosted environment via the web. It represents a new model for software development that is rapidly increasing in its popularity. An example of PaaS is Salesforce.com. PaaS provides a framework for agile software development, testing, deployment and maintenance in an integrated environment. Like SaaS, the primary benefit of PaaS, is a lower cost of use, since subscriber fees require a much smaller investment than what is typically encountered when implementing traditional tools for software development, testing, and deployment. PaaS providers handle platform maintenance and system upgrades, resulting in a more efficient and cost-effective solution for enterprise-software development.

*E. "IaaS" (Infrastructure as a Service)*

"IaaS" allows clients to remotely use IT hardware and resources on a "pay-as-you-go" basis. It is also referred to as "HaaS" (hardware as a service). Major IaaS players include companies like IBM, Google and, "Amazon.com". "IaaS" employs virtualization, a method of creating and managing infrastructure resources in the "cloud". "IaaS" provides small start-up firms with a major advantage, since it allows them to gradually expand their IT infrastructure without the need for large capital investments in hardware and peripheral systems.

Our research is organized as follows. Section II presents the requirements and analysis of our private cloud. PLATRAIN platform architecture is detailed in section III. The Cloud Core Engine is explained in detailed in section IV. Section V describes the Virtualization System. Our PLATRAIN Platform Implementation is detailed in section VI. The Deployment and the conclusion are presented in sections IIV and IIIV, respectively.

## II. Private-Cloud Requirements & Analysis

The infrastructure of the Platform is built using computer network. It is an interconnection between computers. It acts as basis of communication in Information Technology (IT). It is system of connected computing devices and shares information and resources between them. The devices in network are connected by communication links (wired/wireless) and share data by Data-communication System.

*A. Network Design*

Network design is the practice of planning and designing a communications network. It starts with identifying business and technical requirements and continues until just before the network implementation stage. Also, it includes network analysis, IP addressing, hardware selection, and implementation planning. Before we dive into how to design a network, let's network lifecycle models. One of the most popular network lifecycle models is Cisco's PPDIOO (Prepare, Plan, Design, Implement, Operate and Optimize) model [2], shown in Figure 2.

- *Prepare.* This is where you define high-level requirements and strategy. For example, your deliverables from this phase may include requirements documentation.
- *Plan.* This stage deals with specific network requirements based on information gathered in the planning stages.
- *Design.* During the design stage, the information gathered from the previous two stages is used to create a detailed network design.
- *Implement.* This is where the work gets done to configure and deploy the network infrastructure. There is often testing to validate the design in this phase. Operate. This is the portion of the life-cycle where the network is in production use. During this stage, monitoring is an important part of validating that the network is working as designed and being able to quickly address issues when it isn't.

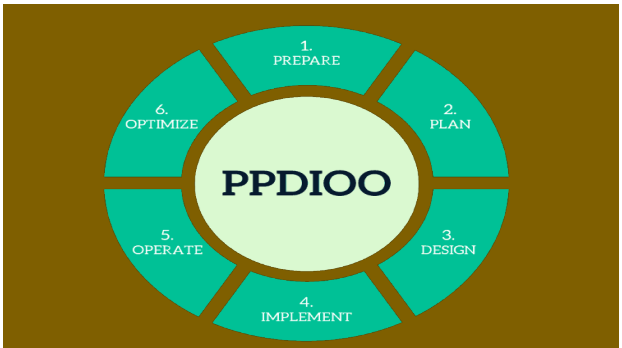


Fig. 2 PPDIOO Model
(Prepare, Plan, Design, Implement, Operate and Optimize)

- *Optimize.* At some point in most networks' life-cycle, tweaks and optimizations are needed [2]. This is the stage where those changes are identified. For major changes, the cycle begins again to plan and implement them. Now, after understanding the basics of a network lifecycle model let's start in network process designing.

*B. Network Topology*

The configuration of a network is key to determining its performance, at Figure 3. Network topology is the way a network is arranged, including the physical or logical description of how links and nodes are set up to related to each other. There are numerous ways a network can be arranged. Figure 3 refers to simple design which clarify how various nodes, devices and connections are physically or logically arranged to each other.

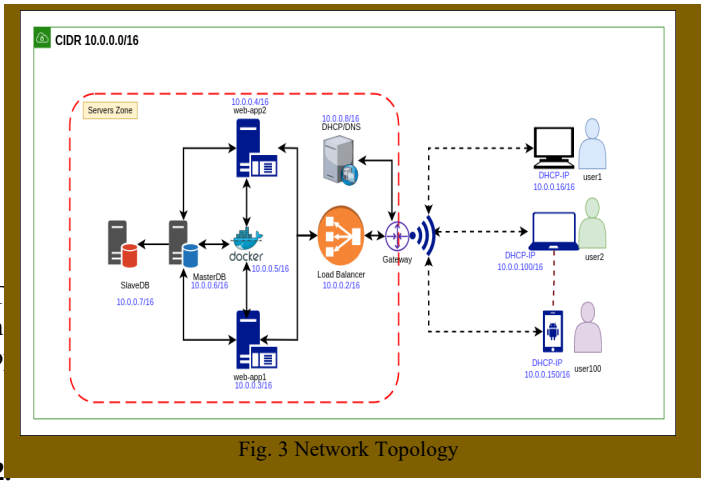


Fig. 3 Network Topology

PLATRAIN is a private cloud platform which provides a digitalized infrastructure for organizations especially educational ones. Our PLATRAIN Platform, at Figure 4, consists of two main parts: cloud system, analysis & telemetry management system. It uses several technologies such as edge computing (IoT)- which is responsible for analytics and telemetry-, cloud computing services -by implementing a micro cloud system to provide storage- and virtualized environment for users, applications, projects and other services (such as Remote Learning). By using such abilities we will extend the power of the platform unleashing the horizon of creativity.

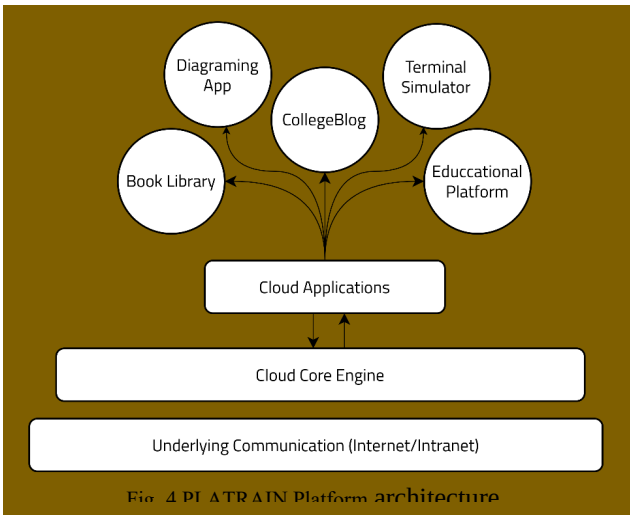


Fig. 4 PLATRAIN Platform architecture

## IV. Cloud Core Engine

In this section, we present how requests are handled and routed for the desired service and how to provide the user (i.e. app) with data and information to be processed and produce a full-filled application, see Figure 5.

### *A. Core Engine Components*

Five main parts in our cloud core engine [2]:

1. *Auth Server.* It handles the process of authentication and authorization process, protect API Routes, generating Tokens and verifying them.
2. *User Management.* It is responsible for registering Users and Query User's Information.
3. *App Engine.* It responsible for registering application to be available for users. It provides the user with core applications such E-learning Platform.
4. *Core Services:* contain the services to provide the core abilities to enable applications to perform actions. The main Core Service is "Container Management Service". This is one of most important services which provides the user with the ability of creating virtualized containers to be used as a virtual computer and create, delete, stop, start Containers. File Management Responsible for downloading, uploading files and files handling.
5. *Database part.* It is responsible for state persistence of the engine.

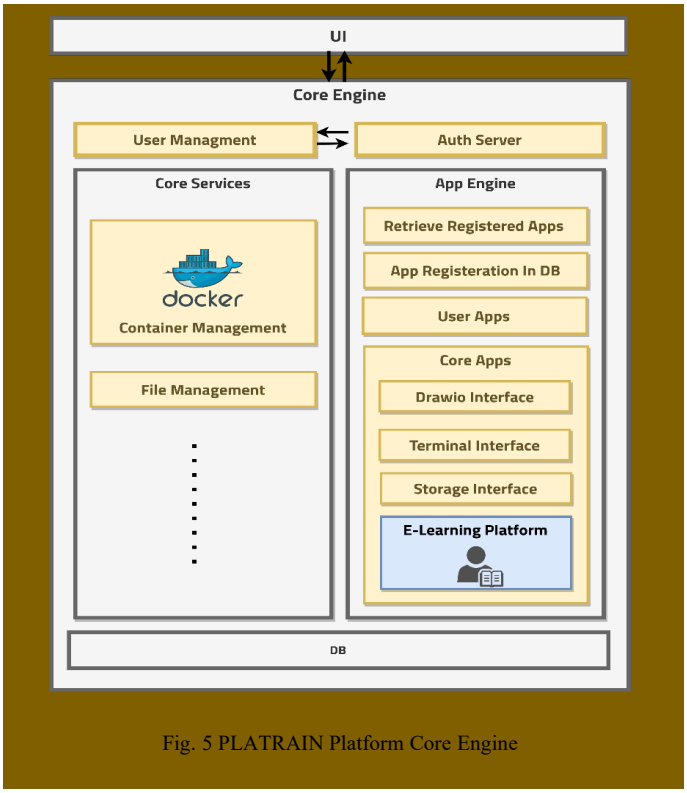


Fig. 5 PLATRAIN Platform Core Engine

## V. Virtualization System

In order to provide the capabilities of virtualization in our platforms we used *Docker* technology to make virtualization. We had to provide this easily to the user on the platform, through a simple interface with which he can easily deal with, and here we decided to use many technologies that interact with each other.

### *A. Virtualization*

Virtualization uses software to create an abstraction layer over computer hardware that allows the hardware elements of a single computer—processors, memory, storage and more—to be divided into multiple virtual computers, commonly called virtual machines (VMs). Each VM runs its own operating system (OS) and behaves like an independent computer, even though it is running on just a portion of the actual underlying computer hardware [2]. It enables more efficient utilization of physical computer hardware and allows a greater return on an organization's hardware investment. Today, virtualization is a standard practice in enterprise IT architecture. It is also the technology that drives cloud computing economics. Virtualization enables cloud providers to serve users with their existing physical computer hardware; it enables cloud users to purchase only the computing resources they need when they need it, and to scale those resources cost-effectively as their workloads grow.

### *B. Docker*

The Docker is an open source containerization platform. It enables developers to package applications into containers —standardized executable components combining application source code with the operating system (OS) libraries and de-

pendencies required to run that code in any environment [3]. Containers simplify delivery of distributed applications, and have become increasingly popular as organizations shift to cloud-native development and hybrid multi cloud environments.

Containers are made possible by process isolation and virtualization capabilities built into the Linux kernel. These capabilities enable multiple application components to share the resources of a single instance of the host operating system [3]. This behavior happens much the same way that a hypervisor enables multiple virtual machines (VMs) to share the CPU, memory and other resources of a single hardware server. As a result, container technology offers all the functionality and benefits of VMs, including application isolation, cost-effective scalability.

Why Docker? We choose Docker to achieve virtualization because of the following benefits:

• Reduced IT management resources.
• Smaller size means one physical machine can host many containers.
• Less code to transfer, migrate, and upload workloads.
• Lightweight and startup in milliseconds.
• Requires less memory space.
• Native performance and
• Easily portable and highly secure.

So, depending upon these configurations and advantages we could say that containers are overcoming virtual machines. The famous global researcher Gartner has predicted that by 2023, more than 50% of companies will adopt Docker containers [4 and 5].

*C. Technologies in PLATRAIN platform*

In our project, we used several technologies that work together to achieve the target of the project, we can split the technologies into two main parts: Backend Technologies the backend, we used the Python programming language, Django Framework and Docker SDK altogether to make the backend of our project. Docker SDK Docker provides an API for interacting with the Docker daemon (called the *Docker Engine API*), as well as SDK Python. The SDKs allow you to build and scale Docker apps and solutions quickly and easily. A Python library for the Docker Engine API [5].

*D. SSH Network Protocol*

The *SSH* protocol (refers to Secure Shell protocol) is a method for secure remote login from one computer to another, shown in Figure 6. It provides several alternative options for strong authentication, and it protects the communications security and integrity with strong encryption. The protocol works in the client-server model, which means that the connection is established by the SSH client connecting to the SSH server [2]. The *SSH* client drives the connection-setup process and uses public key cryptography to verify the identity of the SSH server. After the setup phase the SSH protocol uses strong symmetric encryption and hashing algorithms to ensure the privacy and integrity of the data that is exchanged between the client and server.

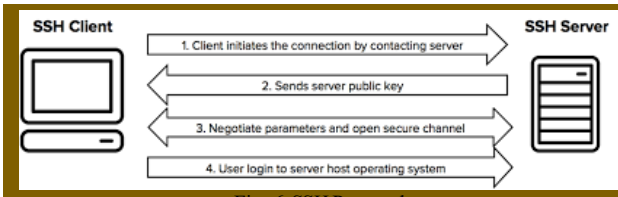

Fig. 6 *SSH* Protocol

*E. Back-End Structure:*

Include three parts: Docker server, platform hosting and client side, as shown in Figure 7.

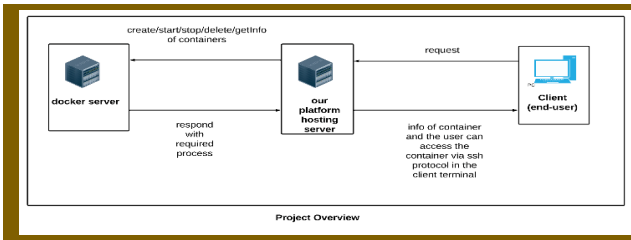

Fig. 7 Overview of our project Architectural

*F. VM (Virtual Machine) services*

Firstly, we install Docker and Docker container with it related databases on a separate machine to work as a server. This allows us to send it requests and receives responses from it. After that we use Docker SDK for Python programming language to write a middleware that will be executed by the requested restful API. Figure 8 presents briefly the steps of VM services.

We built middleware that includes Python function with the help of the Docker SDK. These middleware functions are responsible for dealing directly with Docker server to create, delete, start, stop specific container on the Docker server or get specific container details. Secondly, we built restful API on top of these middleware functions to allow us build the frontend for this service.

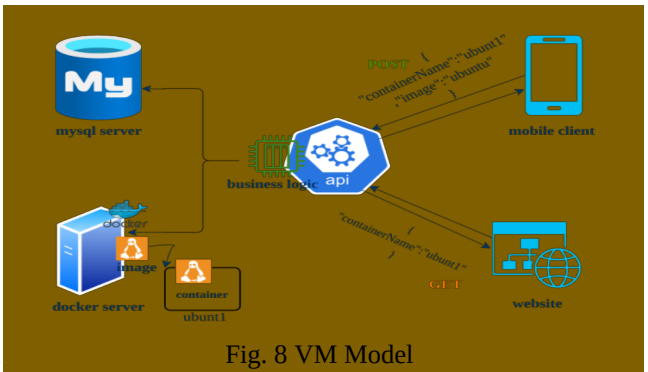

Fig. 8 VM Model

*G.* We used the REST API [6] built in the backend to build our simple front-end of the platform, as shown in Figure 9.

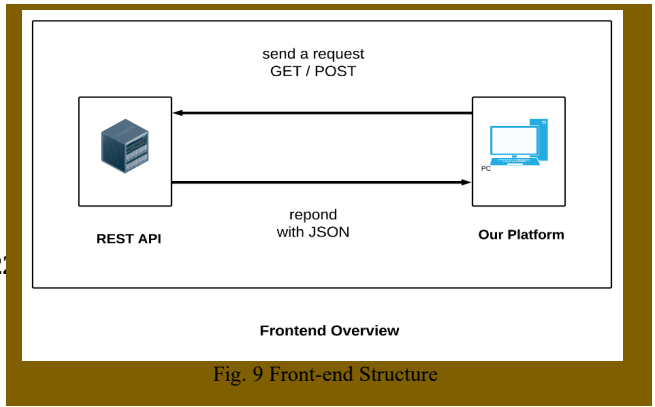

Fig. 9 Front-end Structure

In the designing our pages, we use the following software tools: CSS (styling language), Bootstrap (ver.5) which is CSS framework to direct the designs of the pages to be responsive, and JavaScript code. In this stage, we used the API provided by the backend to achieve the goal of the project. Here we requested all the API using JavaScript fetch API and handled those using JavaScript promises. In this stage, we also handled any errors provided by end-users by showing them helpful messages that help them to show out what the problem was. Based on the sequence of the pages and know how the flow of the user in our platform pages as shown in Figure 10.

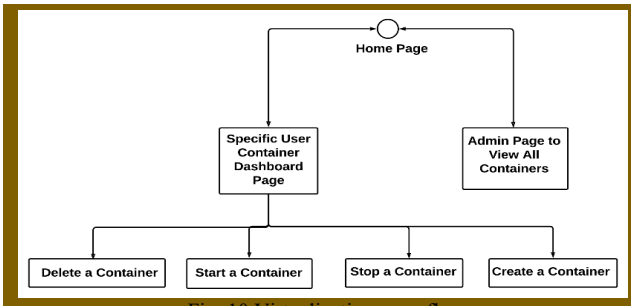

Fig. 10 Virtualization page flow

*H. Software as Service*

In this section, we present the cloud application serveries used in our PLATRAIN and their benefits that depend on SaaS (Software as a Service). The applications that are being prepared so far to be an investigation of an idea SaaS. One of these applications is (*draw.io)* [7] and terminal bases web view, as shown in Figure11.

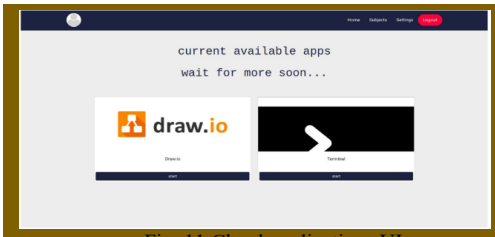

Fig. 11 Cloud applications UI

## VI. PLATRAIN PLATFORM IMPLEMENTATION

Our platform home page, at Figure 12, contains four main services are::

- Cloud services.
- E-learning platform.
- User files, notes, links, and manager.
- Cloud Application (SaaS)

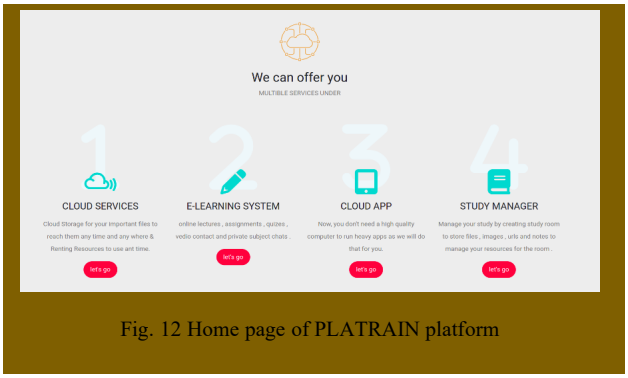

Fig. 12 Home page of PLATRAIN platform

*A. E-learning Platform FAQ*

E-learning strategy was a very good way out from the pickle of non-ability to communicate face to face. In our project we decide to build private E-learning Platform with specific requirements to be used in certain organization –according to its demand. For example, E-Learning educational platform at universities.

*B. E-learning System Design and implementation*

*B.1 Authentication System*

Website authentication is the security process that allows users to verify their identities in order to gain access to their personal accounts on a website. This process occurs behind the scenes any time an individual logs into an online account, including social media profiles, e-Commerce sites, rewards programs, online banking accounts, and more. When a user creates a new account on a website, they create a unique ID and key that will be used in the future to verify their identity and allow them back into the account. That ID and key are then stored in a highly secure web server to compare future credentials against.

Authentication steps are summarized as following:

1. Creating an account for all students and stuff.
2. Adding users to the system, as at Figure 13.

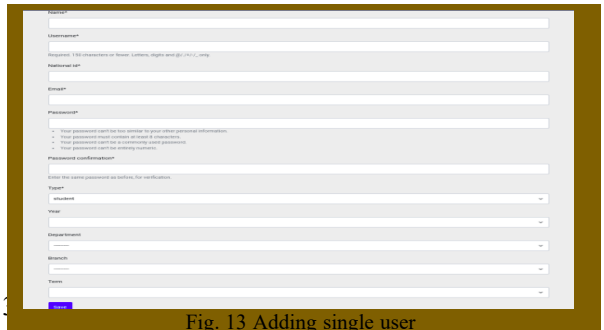
Fig. 13 Adding single user

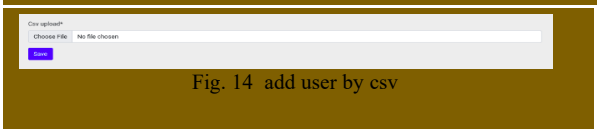
Fig. 14 add user by csv

*B.2 Login system*

All student, staff members have accounts that can be accessed by username and password and the system checks if this user has an account or not, shown in Figure 15.

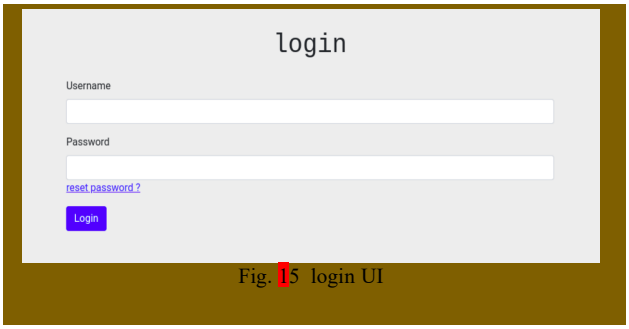


Fig. 15 login UI

*B.3 Reset Password*

In case that the user forgot his password, he can reset it by using his e-mail, shown in Figure 16.

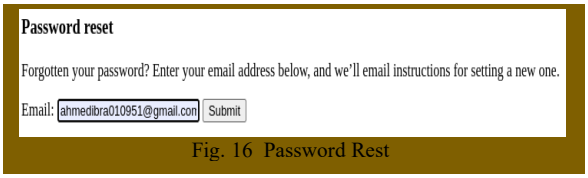


Fig. 16 Password Rest

*B.4 Logout*

There is a logout button that appears instead of the login button in the navigation bar so that we can logout the system when we finish using it, then user can login whenever he wants.

*B.5 Functionalities*

1. Subjects: Every subject has information about it, lectures, assignments, and quizzes. Figures 17 and 18 show the subjects is and how to create new subject, respectively.


Fig. 17 Subjects

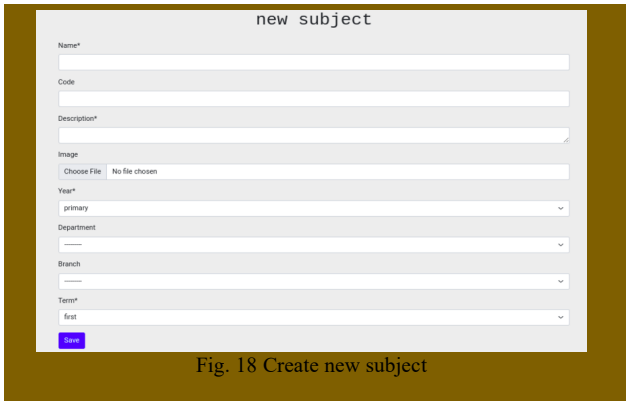


Fig. 18 Create new subject

2. Lectures: UI page to show available lectures, shown in Figure 19.

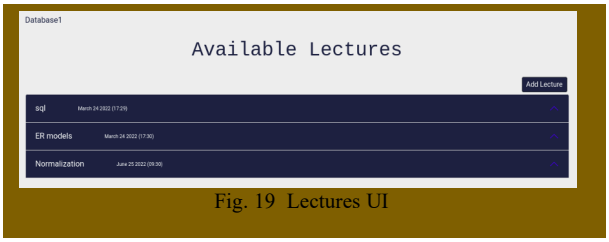


Fig. 19 Lectures UI

3. Assignments: Professor can add new assignment grade them. Also, Student can send an answer and edit or delete it before the assignment final date finishes, shown in Figure 20

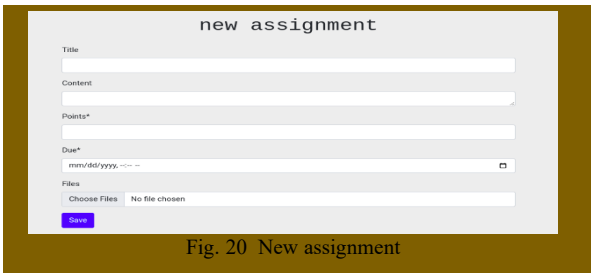


Fig. 20 New assignment

4. Quizzes: Every quiz consists of a number of questions (mcq, true false). Professor can add new quiz and grade them, shown in Figure 21.

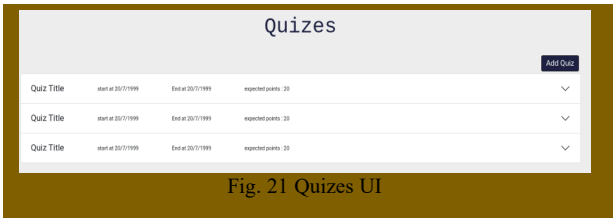


Fig. 21 Quizes UI

*B.6 Settings*

It allows the admin to add and modify and control the platform options, shown in Figure 22.

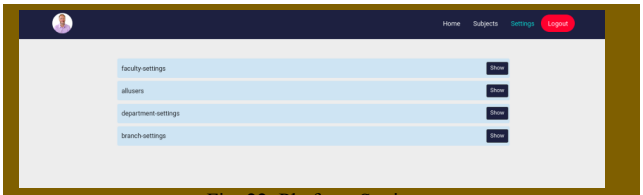

Fig. 22 Platform Settings

## VII. Deployment

Figure 23 presents the deployment model of our project. It shows the overall design of our project on servers including the devices, APIs, Docker or (containerization) and networking elements.

### *A. Distributed Systems*

Known as distributed computing, is a system with multiple components located on different machines that communicate and coordinate actions in order to appear as a single coherent system to the end-user.

A distributed system also known as computing environment in which various components are spread across multiple computers (or other computing devices) on a network.

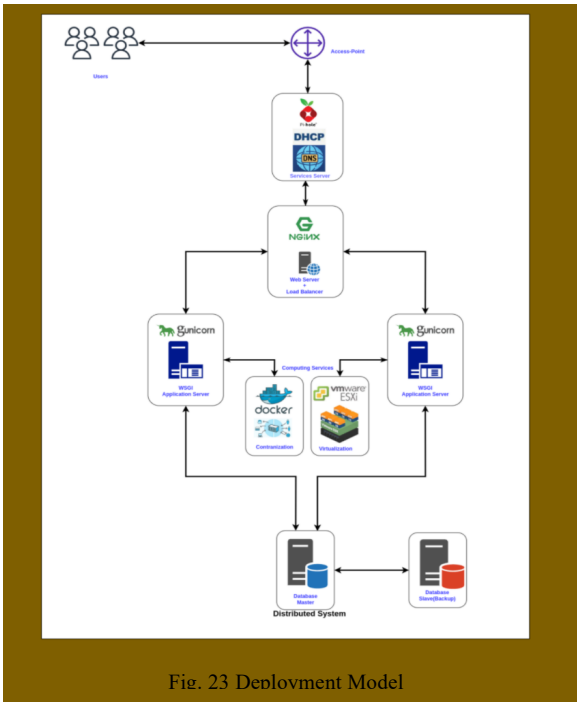

Fig. 23 Deployment Model

Distributed systems are an important development for IT and computer science as an increasing number of related jobs are so massive and complex that it would be impossible for a single computer to handle them alone. Distributed systems reduce the risks involved with having a single point of failure, bolstering reliability and fault tolerance. Modern distributed systems are generally designed to be scalable in near real-time; also, you can spin up additional computing resources on the fly, increasing performance and further reducing time to completion. The machines that are a part of a distributed system may be computers, physical servers, virtual machines, containers, or any other node that can connect to the network, have local memory, and communicate by passing messages. Historically, distributed computing was expensive, complex to configure and difficult to manage. But thanks to software as a service (SaaS) platforms that offer expanded functionality, distributed computing has become more streamlined and affordable for businesses large and small. As a result, all types of computing jobs use distributed computing. In fact, many types of software. As shown fig24

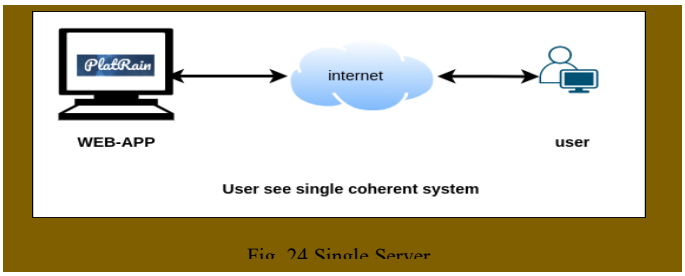

Fig. 24 Single Server

### *B. Messaging*

Messaging enables software applications to connect and scale by separating the sending and receiving of data. A messaging system manages the channels that define the paths of communication between the servers and the sending and receiving of messages. The main task of a messaging system is to reliably move messages from the sender's computer to the receiver's computer.

### *C. Servers Operating Systems*

Community Enterprise Operating System (CentOS) offers an open-source, enterprise-class free operating system that is practicably compatible with Red Hat Enterprise Linux (RHEL). "Gregory Kurtzer" is the founder of CentOS. CentOS developers use the RHEL source code to generate a product that is highly comparable to RHEL. CentOS provides a development platform in one of the best and most powerful available distributions. It is a community-driven free software project built to provide a robust platform for the open source communities to grow. It is highly adaptable, as well as safe and strong. In addition, it features several corporate-level security updates that declare it an excellent choice for any use, shown in Figure25

Fig. 25 CentOS

### *D. Load Balancer*

A load balancer is a device that acts as a reverse proxy and distributes network or application traffic across a number of servers, shown in Figure26. Load balancers are used to increase capacity (concurrent users) and reliability of applications. They improve the overall performance of applications by decreasing the burden on servers associated with managing and maintaining application and network sessions, as well as

by performing application-specific tasks. Load balancing is a core networking solution used to distribute traffic across multiple servers in a server farm.

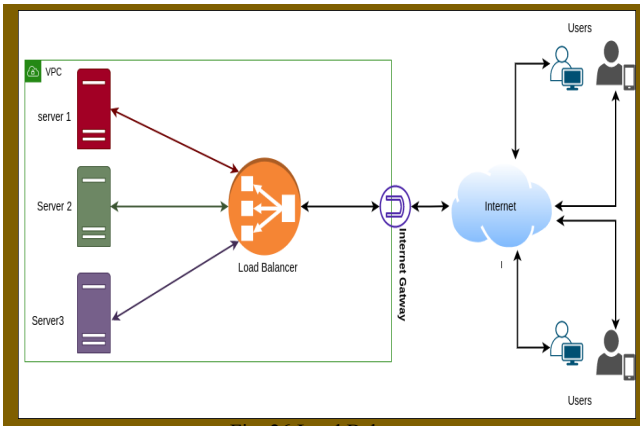


Fig. 26 Load Balancer

*D.1 Nginx as a load balancer*

NGINX is open source software for web serving, reverse proxying, caching, load balancing, media streaming, and more, shown in Figure 27. It started out as a web server designed for maximum performance and stability. In addition to its HTTP server capabilities, NGINX can also function as a proxy server for email (IMAP, POP3, and SMTP) and a reverse proxy and load balancer for HTTP, TCP, and UDP servers. Load balancing across multiple application instances is a commonly used technique for optimizing resource utilization, maximizing throughput, reducing latency, and ensuring fault-tolerant configurations. It is possible to use NGINX as a very efficient HTTP load balancer to distribute traffic to several application servers and to improve performance, scalability, and reliability of web applications with NGINX.

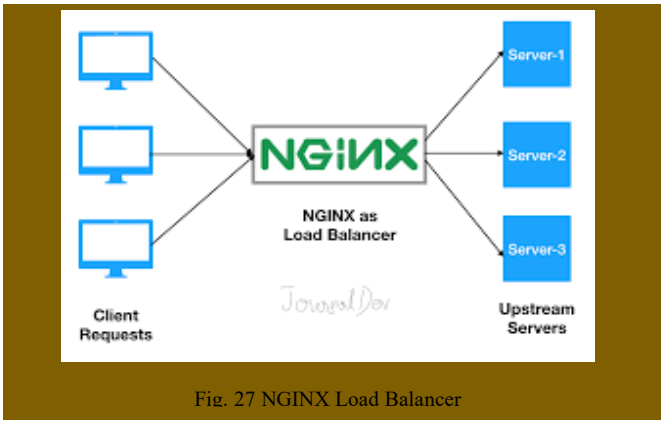


Fig. 27 NGINX Load Balancer

NGINX also supports health checks to mark a server as failed (for a configurable amount of time, default is 10 seconds) if its response fails with an error, thus avoids picking that server for subsequent incoming requests for some time.

*E. Gunicorn*

The Gunicorn server is broadly compatible with various web frameworks, simply implemented, light on server resources, and fairly speedy, shown in Figure 28. Green Unicorn, is a Web Server Gateway Interface server. Implementation that is commonly used to run Python web applications. Gunicorn is used to receive requests from the web server (e.g., Nginx) then passing it down to the python application to process the request. This could be your Django app or Flask or any other python web framework that uses WSGI.

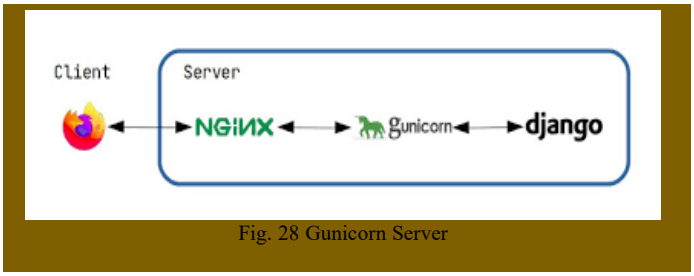


Fig. 28 Gunicorn Server

## VIII. Conclusion

In this project, we construct a private cloud based platform. We called it PLATRAIN. It provides clients and users with a lot of cloud services that are delivered remotely to the clients via the internal network of the organization. In this project we achieved over 90% efficiency performance.

## IX. References


[1] Cloud Computing, Concepts and Practices by Naresh Kumar Sehgal, Pramod Chandra P. Bhatt (2018)

[2] Cloud Computing , Principles, Systems and Applications by Nick Antonopoulos, Lee Gillam in Computer Communications and Networks (2017)

[3] Accelerating Development Velocity Using Docker, Docker Across Microservices by Kinnary Jangla (2018)

[4] https://www.Docker.com/

[5] https://Docker- py.readthedocs.io/

[6] Designing a RESTful API Interface by Brajesh De in API Management (2017)

[7] https://drawio-app.com/