Frankfurt University of Applied Sciences

–Faculty 2: Computer Science and Engineering–

Development of a Concept of a Modern Desktop-as-a-Service

High Integrity Systems

Master of Science (M.Sc.)

Presented By,
Vineeth Bhat
Matriculation Number: 1322415

Supervisor	:	**Prof. Dr. Christian Baun**
Second Supervisor	:	**Prof. Dr. Eicke Godehardt**

DECLARATION

I hereby confirm that I have written the present work independently and have not used any other sources than those given in the bibliography.

All passages that are taken verbatim or correspondingly from published or not yet published sources are marked as such.

The drawings or images in this work were created by myself or provided with a corresponding source reference.

This work has not been submitted to any other examination authority in the same or a similar form.

Frankfurt, 14. October 2021

Vineeth Bhat

ABSTRACT

Rapid growth in the field of Cloud Computing has led to the development of various cloud technologies like Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), Software-as-a-Service (SaaS), the need for moving everything to the cloud is ever-growing. Utilizing the advancement in Cloud Computing, such services are extended to provide remote desktops delivered as Virtual Desktop Infrastructure (VDI) and Desktop-as-a-Service (DaaS) which uses remote desktop connection (RDC) to provide the desktop to users via a web browser. Looking into such applications this thesis focuses on developing a concept of DaaS using modern available open-source technologies that uses RDC to provide a remote desktop over the web browser and also proposes a way of improving a similar open-source application called the Oneye, formerly known as eyeOS. Design, architecture, implementation, testing and comparison of such applications will give a good understanding of what DaaS is and how one can develop this to provide the market standard services desired from such applications.

ACKNOWLEDGMENTS

First, I would like to thank Prof. Dr. Christian Baun and Prof. Dr. Eicke Godehardt, my supervisors, for providing me the opportunity to work on this thesis, for the guidance they provided and for supporting me to succeed in writing this thesis. Also, I would like to thank everyone who has been helpful during the course of my thesis, for their help in solving problems and constant support during these challenging times. Lastly, thank you to all the tech community for the suggestions and solutions that aided me in finding the right path in my work.

CONTENTS

LIST OF FIGURES

LISTINGS

ACRONYMS

DaaS	Desktop-as-a-Service
PaaS	Platform-as-a-Service
IaaS	Infrastructure-as-a-Service
VDI	Virtual Desktop Infrastructure
RDC	Remote Desktop Client
RDP	Remote Desktop Protocol
VNC	Virtual Network Computing
SSH	Secure Shell
VM	Virtual Machine
SaaS	Software-as-a-Service
RFB	Remote Frame Buffer
API	Application Programming Interface
EJS	Embedded JavaScript Template
OS	Operating System
HTTP	Hypertext Transfer Protocol
HTML	HyperText Markup Language
RxJS	Reactive Extensions Library for JavaScript
XML	Extensible Markup Language
CSS	Cascading Style Sheet
TLS	Transport Layer Security
SSL	Secure Sockets Layer
CA	Certificate Authority
CSR	Certificate Signing Request
WS	Web Socket
DOM	Document Object Model
URL	Uniform Resource Locator

IV	Initialization Vector
AES	Advanced Encryption Standard
CBC	Cipher Block Chaining
UI	User Interface
IP	Internet Protocol
JSON	JavaScript Object Notation
dmg	Disk Image
REST	Representational State Transfer
CLI	Command Line Interface
LDAP	Lightweight Directory Access Protocol
DDoS	Distributed Denial-of-Service
SFTP	SSH File Transfer Protocol

Part I

THESIS

1 INTRODUCTION

With the advancement in technology especially in the area of cloud computing and cloud development, the majority of the software, data and various other resources are moving to the cloud. This is because of the on-demand availability of these computing resources making it convenient to manage and utilize these resources remotely. One such area of interest is Desktop-as-a-Service.

1.1 MOTIVATION

Desktop-as-a-Service (DaaS) as the name suggests, is a cloud computing service that provides the virtualized desktop as a service over the cloud. This is becoming more prominent with the modern trend of working remotely and to enable faster and more efficient maintenance of systems, providing high security, and managing the fluctuating demands. The principle behind DaaS is similar to that of a Virtual Desktop Infrastructure, which is providing virtual desktops and virtual desktop application environments to the users through remote desktop connections. The difference is that DaaS is a multi-tenant architecture similar to Software-as-a-Service (SaaS), hosted by a third-party service provider, where provider takes responsibility for the security, data storage, maintenance, backup and upgrades to the software as well as infrastructure and the clients can access it remotely using thin-clients, or tablets or mobile phones through the web browser. The user can easily log in to the application through the browser and have a desktop environment running on the browser. The VDI is deployed in-house on the private cloud, which requires huge upfront investment and constant maintenance support. DaaS on the other hand can be hosted on a public or private cloud infrastructure and is often delivered based on a subscription model that reflects Pay-As-You-Go.

There are multiple commercial solutions available that are designed and targeted for large-scale corporations. These solutions are expensive for a relatively small organization such as an educational institution that does not require a large investment in such solutions. Several open-source solutions perform similar tasks compared to commercial products, but these solutions are outdated and are not currently supported by the community and require a complex set up by the user to use the application. To overcome these issues, a new design for DaaS is proposed. Utilizing the modern open-source technologies and analyzing the requirements, the proposed design should be able to handle all the necessary functionalities that are required in the DaaS.

1.2 STATE OF THE ART

There are various solutions available, both open-source as well as commercial applications. To name a few commercial applications Amazon WorkSpaces [2], Azure Virtual Desktop [6] and Citrix VDI and DaaS solutions [9]. As explained in the motivation [1.1], these solutions are expensive and are suitable for large-scale corporation. Advancement in technology has paved the way for various researches and implementations to achieve DaaS in the open-source community, leading to the development of several solutions, that provide varying features and functionalities. These DaaS are also termed as cloudOS, webOS, web desktops and so on. This research [8] names a few such applications available and briefly explains about them.

Looking at one such application is the development of DaaS in Openstack[48]. Openstack has several components and one of those is the Horizon[34], as shown in Figure 1.1. This is a dashboard project that provides a web-based user interface to access and manage the services provided by Openstack. It supports SPICE for providing connection to virtual desktops. Before the integrations of SPICE into Openstack, it supported noVNC that uses Remote Frame Buffer (RFB) protocol to achieve the remote connection. Literature[7] looks into improving the capabilities of Horizon by using RDP and VNC. However, this web-based dashboard is used mainly to maintain the infrastructure rather than to provide virtualized desktops to users. In addition, the setup [36] of Openstack is complex and the user experience is not intuitive.

Similarly, OpenNebula provides a VDI solution out-of-the-box. However, to make it easier for the user, they have provided a solution by integrating the infrastructure front-end with Apache guacamole [53].

Another open-source application is Web Desktop Environment[54]. This is purely a web application that is built using react framework. It does not use any remote desktop connections to connect to a remote desktop rather renders a virtual desktop and utilizes the underlying desktop to operate. This is one of the main disadvantages of this application, without any remote desktop connections, users don't have access to Windows/Linux systems and use the native applications directly, they have to develop the application for Web desktop in Javascript as an interface to render the native application in Web Desktop. Figure 1.2 shows the Web Desktop Environment. The application might work well for a personal server, as it works as an interface for the underlying system. Wider utilization of the application is limited due to a custom interface that requires customized applications.

Considering the cloud technologies and cloud-native architecture, Kasm Technologies has developed a solution for DaaS[37] using cloud-native architecture. The DaaS solution is a containerized application with all the services and the desktop that is provided as a workspace for the user are con-

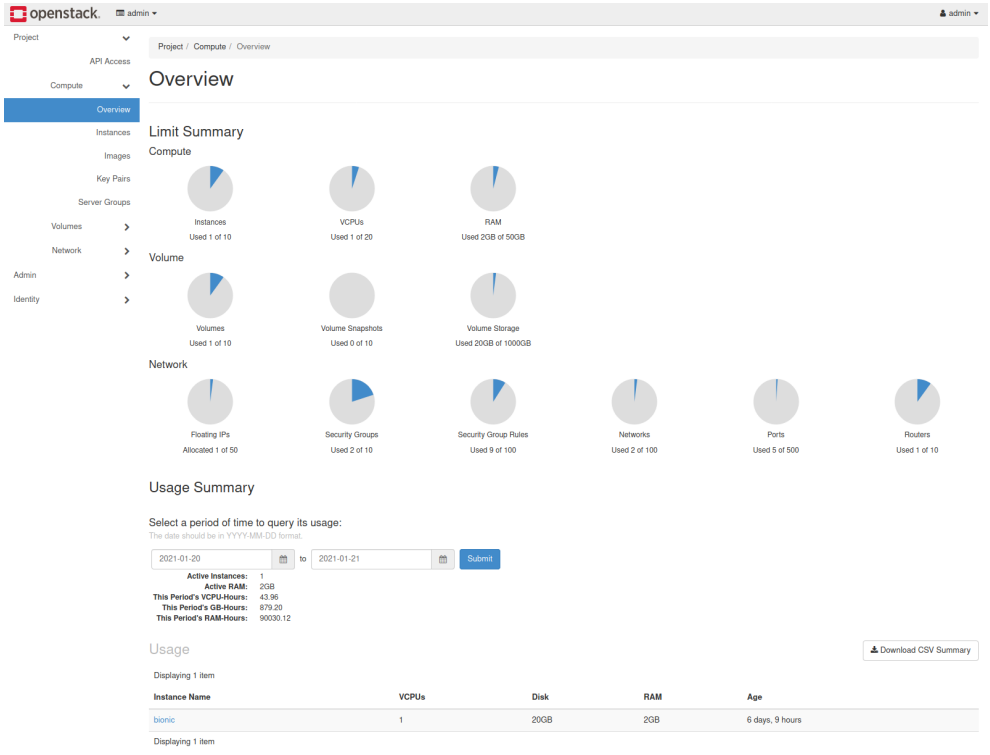


Figure 1.1: Openstack Horizon Dashboard

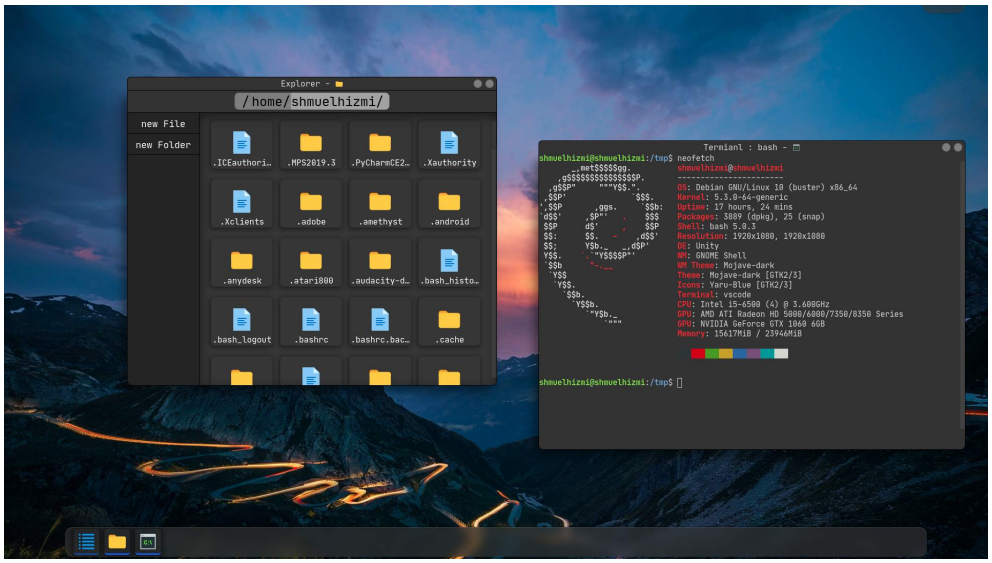


Figure 1.2: Web Desktop Environment[54]

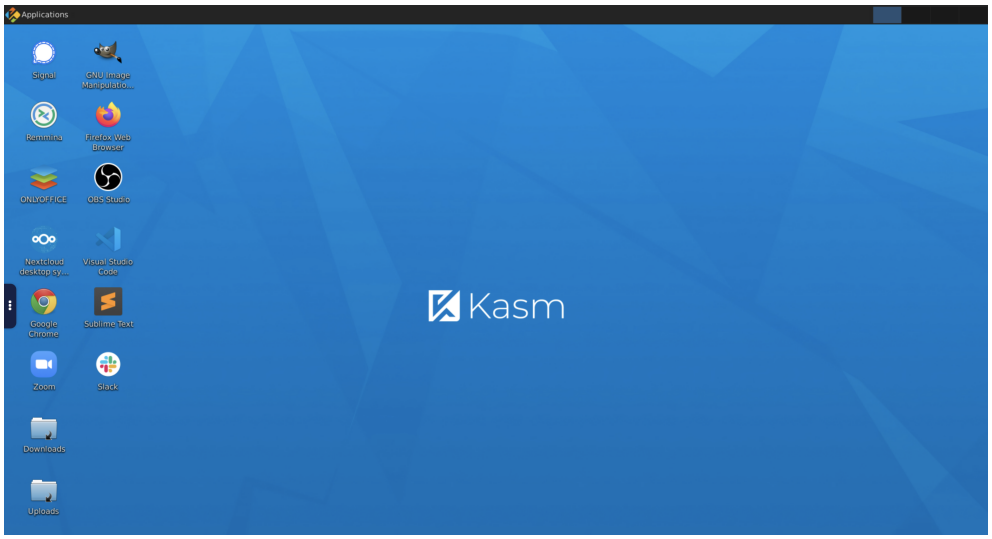


Figure 1.3: Kasm Remote Workspace[37]

tainerized in Docker. To connect to the containerized desktop Kasm uses a custom-built VNC called KasmVNC[39], which is an open-source project maintained by Kasm Technologies. Along with this open-source project the desktop images are also open-sourced [38]. The application is feature-rich with features like persistent sessions for users, various authentication mechanisms and many more. All the communication between the different services takes place internally within the docker. The product is open-sourced to the community with limited functionalities. One of the main drawbacks with this is it supports only Linux operating systems as virtual desktops. This is because of the containerization of virtual desktops. Figure 1.3 shows the remote workspace offered by Kasm.

An alternate open-source software is the Oneye[46]. This is similar to Web Desktop Environment mentioned earlier. This is one of the intriguing applications developed, formerly known as the eyeOS web operating system. The application behaves as a virtual desktop by utilizing the underlying server. It is majorly developed using PHP. It is a well-developed software, with several applications built into the software. There are few drawbacks with this software, it has outdated PHP code and the software runs with php5, which causes compatibility issues with the latest versions of PHP. Another major drawback is due to non-existing remote desktop connections to an operating system such as Windows/Linux, it does not allow the user to install or use Windows/Linux applications and restricts the user to the applications developed in Oneye. So, the application has to be developed in PHP to be used in Oneye. Figure 1.4 shows the desktop of the Oneye.

By analyzing all these applications, majorly Oneye and its drawbacks, the target was to design a solution for DaaS, which has the capability to provide the necessary feature like remote control of a virtual desktop, by utilizing

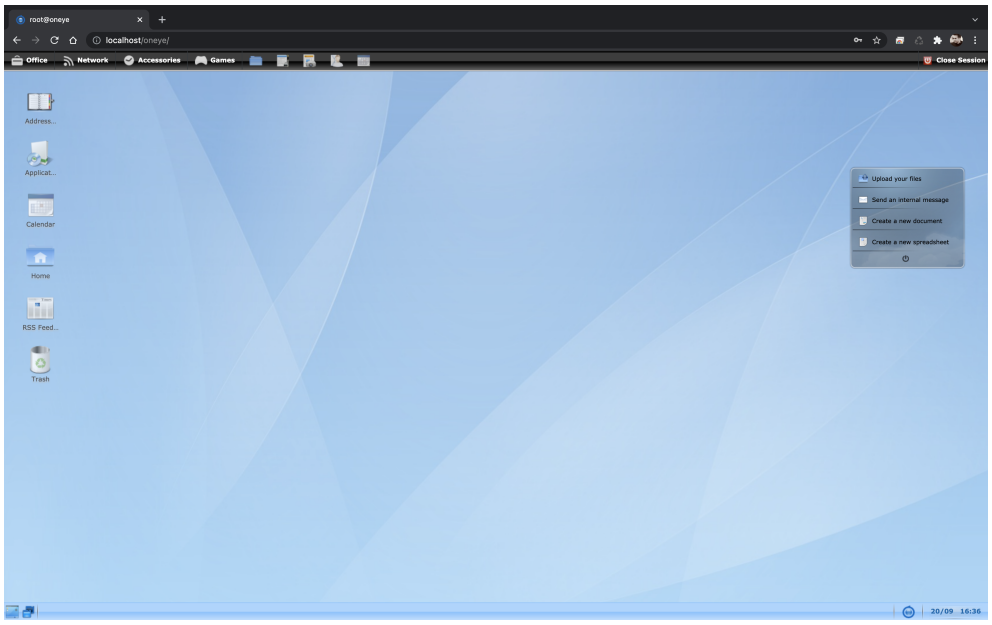


Figure 1.4: Oneye Desktop

the current technologies. This application can also be combined with Oneye to create a much better solution and to improve the Oneye as an application.

2 DESIGN

2.1 DESIGN AND ARCHITECTURE

For the design of this DaaS, inspiration, as well as components, are taken from Apache Guacamole[4]. The architecture of the concept of a DaaS is designed along with several components integrated together, of which Apache Guacamole forming the core of the application design. "Apache Guacamole is a clientless remote desktop gateway. It has built-in support for VNC, RDP, and SSH protocols"[4]. Apache Guacamole itself is an application that can be set up as it is with the Infrastructure-as-a-Service (IaaS) like Openstack or OpenNebula. However, for this concept the core engine of Apache Guacamole that is guacd is utilized as the rest of the components of Guacamole can be developed using alternate and advanced solutions that can be easily integrated into other web applications or can be used as a standalone application.

Figure 2.1, shows the architecture of the application and the connection between different components.

- **Guacd:** As shown in Figure 2.1, the application consists of guacd, which is the core component of the application that forms the server-side proxy of the application. The main functionality of guacd is to connect to the remote desktops using RDP/VNC/SSH. It keeps the port open for TCP connections incoming from the web applications and detects the type of connection needed for remote connection and connects the user to the appropriate desktop. This acts as a translation layer between the virtual desktops and the web applications.
- **Web Server:** To communicate with guacd, there should be an intermediary server that provides a connection between the application and guacd. Since the application is using guacd as the proxy for connecting to remote desktops. The server must implement the guacamole protocol. This is the protocol that Apache Guacamole uses for communication between guacd and the client. The reason behind this is guacd neither implements nor understands any of the protocol but implements guacamole protocol to understand the connection required from remote desktops.
- **Front-end:** The front-end component forms the outermost layer of the application, where the remote desktop is rendered to the web browser. This is also the component that implements keyboard, mouse and other events that are sent to the virtual desktop through web socket server and guacd.

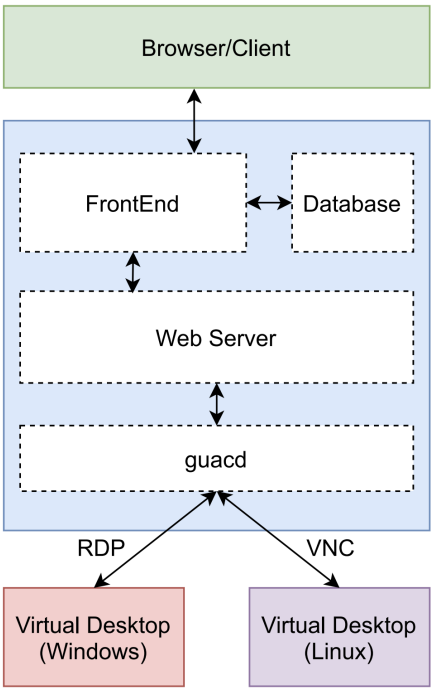


Figure 2.1: Architecture of concept of a DaaS

- **Database:** The inclusion of the database is for authentication purposes and also to save the user information such as the connection details for a particular user.

Figure 2.2 explains how this application can be combined with the Oneye which acts as a web server when deployed on apache, where the user will be presented with the Oneye desktop upon authentication with Oneye and an application with Oneye can redirect the user to the application server. A web server is a simple server that serves the static contents over HTTP. Whereas the application server serves the business logic to the client through a web server. Generally, application servers can contain web servers and deal with several protocols including HTTP.

2.2 TECHNOLOGIES

2.2.1 *Guacd*

Guacd is a server-side proxy that loads the needed support for remote desktop connections such as RDP/SSH/VNC. As explained in the architecture section 2.1, it acts as a translation layer between the virtual desktop and web application. This is because the web applications do not implement any RDP or VNC support. The connection from the web application is sent as guacamole protocol which has to be translated into appropriate protocol for remote connection and loads the so-called client plugin for the connection.[30]

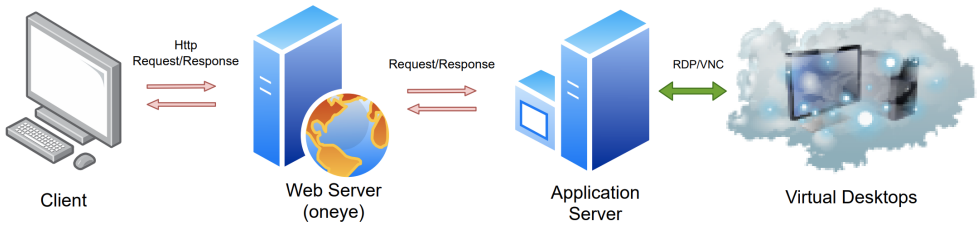


Figure 2.2: DaaS integrated with Oneye

These client plugins are extended from libguac[30]. This is the library that forms the base of all the implementations in guacamole. For the development of this application guacd docker image by glyptodon[61] is used. This is a stable and reliable docker image with constant updates available.

2.2.2 *Guacamole-lite*

According to Apache Guacamole the web server connecting to guacd, can be implemented by using a Java servlet container that extends by using the Java API which is called guacamole-common[63]. This provides doConnect(), doread() and dowrite() to handle the incoming requests. To make the integration with the front-end easier and the development quick an alternate solution is available in the form of guacamole-lite[64]. This is a Node.js alternative to the guacamole-common. Guacamole-lite is a simple node server that can be easily configured and it implements the guacamole protocol, which creates a connection with the guacd. Guacamole protocol is made of a set of instructions separated by a comma. More details regarding the guacamole protocol will be covered in the implementation section 3.2.2. Another advantage of guacamole-lite is that it provides the mechanism to decrypt the encrypted connection sent from the front-end using the crypto from npm (deprecated now[13], can be replaced with crypto.js[14]).

2.2.3 *Node.Js*

Node.js is an asynchronous, open-source, cross-platform server-side Javascript runtime environment, which is used by developers to develop-server side applications[43]. Node.js works on a single thread and avoids dead-locking as there are no locks. "Almost no function in Node.js directly performs I/O, so the process never blocks"[1]. Hence, scalable systems can be easily developed using Node.js. Guacamole-lite is developed using Node.js and also one of the front-end designs for this prototype is developed using Node.js along with EJS.

2.2.4 *Angular*

For the development of one of the versions of the front-end for this application, Angular is used. Angular is an open-source web application framework built on the typescript. Angular is led by a team dedicated to Angular Google and the open-source community. Angular is a component-based web framework that allows developers to build scalable web applications.[3]. Using angular single-page applications can be easily developed and the capabilities of Angular such as RxJS library compatibility, makes it a powerful front-end framework to build the application with. For the development of this application an open-source Angular project called the ngx-remote-desktop[42], which is developed for remote desktop applications using Apache Guacamole is used. This project includes a front-end demo application with some of the basic features like mouse control, clipboard and so on developed. This can be easily integrated into an application that is using guacamole as its back-end server.

2.2.5 *EJS*

EJS is called an Embedded Javascript Template. EJS is a simple templating language that enables the creation of HTML using Javascript[24]. EJS is not used extensively nowadays and has fallen out of favor compared to React or Angular. However, to test and develop a simple front-end for the DaaS application EJS is used here in the form of an open-source project from linuxserver[27]. This project uses the guacamole-lite as the server and implements the front-end using EJS.

2.2.6 *Docker*

Docker is a Platform-as-a-Service application that delivers software in packages known as containers using OS-level virtualization. Each of these containers has its environment and is isolated from each other. The initial release of Docker was in 2013 and is now developed by Docker, Inc. Docker enables easy setup of development environments and deployments are much faster and efficient[21]. It also supports a library called Docker Hub for container images where the developers or software vendors can share their container images[22].

2.2.7 *Apache*

Here Apache server is mainly used for the deployment of Oneye as a web server. Apache is also known as Apache HTTP Server, is an open-source free web server software. Apache 1.0 was released in 1995, quickly becoming the most popular server on the internet[5]. For the deployment of Oneye which is a PHP application, a docker image containing Apache server with PHP is utilized"[18].

2.2.8 *Oneye*

Oneye is an open-source web desktop that is mainly developed in PHP along with javascript and XML. As explained in the state of the art 1.2 it is a virtual desktop that takes advantage of the underlying server and renders the desktop on a web browser to the user. Formerly known as eyeOS which was initially released in 2005, is one of the unique web desktops available due to the features it provides like file management, built-in browser, document management tools to name a few. Due to its outdated source code and incompatibility with native Windows/Linux applications, it has not been in favor with users. For the deployment of Oneye in docker, a docker image of PHP version 5 in conjunction with Apache HTTPd is used.

3 IMPLEMENTATION

This chapter covers the implementation and integration of all the components described in the Design and Architecture section 2.1. The development and testing were carried out in a Linux environment and the application is deployed in a containerized environment using docker.

3.1 GUACD

The core component of the application is deployed first. Guacd can be deployed by downloading the guacamole server[32] and building the server locally. However, it is much easier and simple to deploy the server using docker as the guacd docker image is available in docker hub by glyptodon[61]. The listing 3.1 shows the instructions needed to deploy guacd in docker, the *container_name* specifies the name of the container which is given as guacd in this case, followed by the image that is the container *image* that needs to be deployed which is glyptodon/guacd, next is the environment variable which can either be passed as an argument named as *environment* in the docker-compose or as an environment file to the working directory. Here ACCEPT_EULA environment variable is set to Y, this accepts the glyptodon enterprise "End User License Agreement"[28]. Without setting this the guacd server will not start. This is followed by the port that needs to be exposed which is set using *expose* instruction and the port that guacd listens to is port 4822. On "docker compose up" command the container will be started with the name guacd and will start to listen on port 4822. This can be seen on docker desktop and the log will display that the guacd demon server is started along with the host and port as shown in Figure 3.1.

```
guacd:
    container_name: guacd
    image: glyptodon/guacd
    environment:
      ACCEPT_EULA: "Y"
    expose:
      - "4822"
    ports:
      - "4822:4822"
```

Listing 3.1: Docker compose code for guacd

To provide security, the connection between guacd and the web server can be encrypted by enabling the TLS/SSL in guacd using the guacd.conf file as explained in the guacamole manual[11]. Since the guacd in this application is deployed using docker the configurations are different. This is done

```
-------- Container started on 2021-09-22 14:48:40 UTC --------

/opt/glyptodon/share/docker/entrypoint.sh: line 22: [: : integer expression expected
/opt/glyptodon/share/docker/entrypoint.sh: line 29: [: : integer expression expected
guacd[12]: INFO: Guacamole proxy daemon (guacd) version 1.3.0-GLEN-2.4 started
guacd[12]: INFO: Listening on host 0.0.0.0, port 4822
```

Figure 3.1: guacd log in docker desktop

by loading the SSL certificate and key to the guacd container via volume mount. The guacd.conf is placed in /etc/guacamole/ within the container and the SSL certificate and the key path is mentioned in the guacd.conf[12].

Initially, docker image from guacamole was deployed[65], where the deployment was carried out similarly as explained earlier. Once the web application was set up, that is the web server that interacts with guacd was not able to connect to guacd on port 4822. The connection was refused by guacd, upon investigating it was mentioned in tech forums that this was due to the fact that guacd process was not started. However, this was not the case as the logs and the docker desktop showed the demon process was running and listening on port 4822. This was one of the issues that were encountered when running the guacd from guacamole. Unable to find a solution to this, guacd from glyptodon was used. "Glyptadon Enterprise is a company that offers commercial build of Apache guacamole"[29].

3.2 WEB SERVER

Web server interacts with gucad and provides guacd with a set of instructions to load the appropriate client plugin that has the protocols to connect to virtual desktops and also return the data from the remote desktop back to the client. All these must be carried out through guacamole protocol as the guacd can only take instruction based on guacamole protocol. This can be achieved using guacamole-lite[64].

3.2.1 *Guacamole-lite*

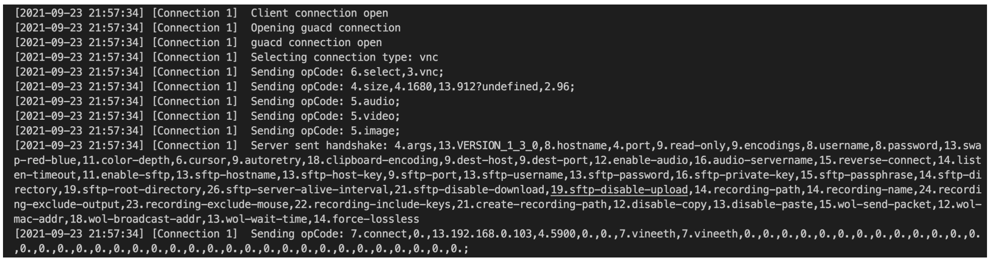
Guacamole-lite is an open-source project that is a replacement for guacamole-client implemented in node.js. Guacamole-lite can be easily installed as it is available as an npm package. Once installed, *GuacamoleLite* can be imported as shown in listing 3.2. Once imported guacamole-lite provides a constructor method to create a web server and connects to guacd. The parameters required to invoke this method are optional web server configurations that include a port number the server is listening to and SSL certificate and key, optional guacd host and port that it is running on and the clientOptions which is the encryption scheme and the secret key that is used to encrypt the connection request sent by the client, which is described in the listing 3.2.

```
var GuacamoleLite = require('guacamole-lite');

var https = require('https').createServer({
    key: fs.readFileSync("./localhost.key"),
    cert: fs.readFileSync("./localhost.crt"),
},app);

var clientOptions = {
    crypt: {
        cypher: 'AES-256-CBC',
        key: 'GUACAMOLELITESECRETKEY'
    },
    log: {
        verbose: false
    }
};

var guacServer = new GuacamoleLite({server: https,path: '/
    guaclite'},{host:'127.0.0.1',port:4822},clientOptions);
```

Listing 3.2: Guacamole-lite configuration

3.2.2 *Guacamole-protocol*

Guacamole-lite receives the encrypted connection request from the client through guacamole protocol as shown in Figure 3.10 and Figure 3.8, shows the encrypted connection, as further explanation on encryption is provided in 3.3.3. This encrypted connection is decrypted by using the encryption scheme and secret key provided in clientOptions. Once decrypted the connection is established with the guacd, which is indicated in the log as Opening guacd connection as shown in Figure 3.2. All these interactions take place using guacamole protocol. In guacamole protocol, the connection details are sent as a set of instructions along with arguments that are separated by comma as shown in Figure 3.2. These instructions are not just converted from the web server, since the client also uses guacamole library called guacamole-common-js 3.3.1, the request from the client itself is sent with guacamole protocol. Breaking down the instruction one can see that the initial element is the OPCODE followed by its arguments. Each element in that set is represented by a length of the value followed by a period and the value which is encoded in UTF-8, as LENGTH.VALUE. "For example, display size of 1024x768 is represented as 4.size,1.0,4.1024,3.768"[31]. There are several advantages of sending the connection in such format as this allows Javascript to easily parse the values or skip through the instructions by using the length and avoid iterating over all the instructions. The entire handshake process between the web server and guacd takes place using this guacamole protocol.

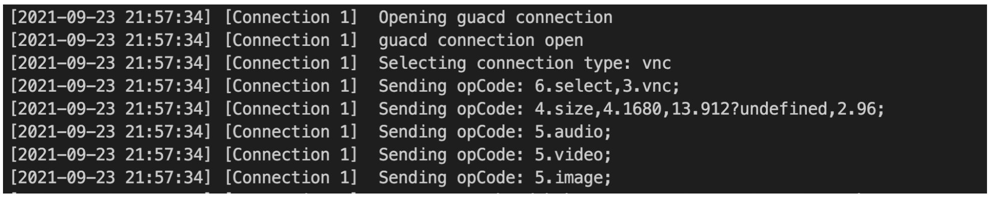


```
[2021-09-23 21:57:34] [Connection 1]  Client connection open
[2021-09-23 21:57:34] [Connection 1]  Opening guacd connection
[2021-09-23 21:57:34] [Connection 1]  guacd connection open
[2021-09-23 21:57:34] [Connection 1]  Selecting connection type: vnc
[2021-09-23 21:57:34] [Connection 1]  Sending opCode: 6.select,3.vnc;
[2021-09-23 21:57:34] [Connection 1]  Sending opCode: 4.size,4.1680,13.912?undefined,2.96;
[2021-09-23 21:57:34] [Connection 1]  Sending opCode: 5.audio;
[2021-09-23 21:57:34] [Connection 1]  Sending opCode: 5.video;
[2021-09-23 21:57:34] [Connection 1]  Sending opCode: 5.image;
[2021-09-23 21:57:34] [Connection 1]  Server sent handshake: 4.args,13.VERSION_1_3_0,8.hostname,4.port,9.read-only,9.encodings,8.username,8.password,13.swa
p-red-blue,11.color-depth,6.cursor,9.autoretry,18.clipboard-encoding,9.dest-host,9.dest-port,12.enable-audio,16.audio-servername,15.reverse-connect,14.list
en-timeout,11.enable-sftp,13.sftp-hostname,13.sftp-host-key,9.sftp-port,13.sftp-username,13.sftp-password,16.sftp-private-key,15.sftp-passphrase,14.sftp-di
rectory,19.sftp-root-directory,26.sftp-server-alive-interval,21.sftp-disable-download,19.sftp-disable-upload,14.recording-path,14.recording-name,24.recordi
ng-exclude-output,23.recording-exclude-mouse,22.recording-include-keys,21.create-recording-path,12.disable-copy,13.disable-paste,15.wol-send-packet,12.wol-
mac-addr,18.wol-broadcast-addr,13.wol-wait-time,14.force-lossless
[2021-09-23 21:57:34] [Connection 1]  Sending opCode: 7.connect,0.,13.192.168.0.103,4.5900,0.,0.,7.vineeth,7.vineeth,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.
,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.;
```

Figure 3.2: Guacamole protocol instructions

```
processConnectionOpen() {
    this.log(this.server.LOGLEVEL.VERBOSE, 'guacd connection
       open');

    this.log(this.server.LOGLEVEL.VERBOSE, 'Selecting
       connection type: ' +
       this.clientConnection.connectionType);

    this.sendOpCode(['select',
       this.clientConnection.connectionType]);
}
```

Listing 3.3: Handshake initiation between guacamole-lite and guacd



```
[2021-09-23 21:57:34] [Connection 1]  Opening guacd connection
[2021-09-23 21:57:34] [Connection 1]  guacd connection open
[2021-09-23 21:57:34] [Connection 1]  Selecting connection type: vnc
[2021-09-23 21:57:34] [Connection 1]  Sending opCode: 6.select,3.vnc;
[2021-09-23 21:57:34] [Connection 1]  Sending opCode: 4.size,4.1680,13.912?undefined,2.96;
[2021-09-23 21:57:34] [Connection 1]  Sending opCode: 5.audio;
[2021-09-23 21:57:34] [Connection 1]  Sending opCode: 5.video;
[2021-09-23 21:57:34] [Connection 1]  Sending opCode: 5.image;
```

Figure 3.3: Initial handshake process

3.2.3 *Guacamole protocol Handshake*

Once the guacd connection is open, the handshake phase begins with the *select* statement as shown in the listing 3.3. This instruction is sent by the client that contains the type of connection that needs to be established with the remote desktop and the appropriate protocol is loaded by guacd. This is followed by sending the display size, supported audio, video and image as shown in Figure 3.3.

```
[2021-09-23 21:57:34] [Connection 1]  Server sent handshake: 4.args,13.VERSION_1_3_0,8.hostname,4.port,9.read-only,9.encodings,8.username,8.password,13.swa
p-red-blue,11.color-depth,6.cursor,9.autoretry,18.clipboard-encoding,9.dest-host,9.dest-port,12.enable-audio,16.audio-servername,15.reverse-connect,14.list
en-timeout,11.enable-sftp,13.sftp-hostname,13.sftp-host-key,9.sftp-port,13.sftp-username,13.sftp-password,16.sftp-private-key,15.sftp-passphrase,14.sftp-di
rectory,19.sftp-root-directory,26.sftp-server-alive-interval,21.sftp-disable-download,19.sftp-disable-upload,14.recording-path,14.recording-name,24.recordi
ng-exclude-output,23.recording-exclude-mouse,22.recording-include-keys,21.create-recording-path,12.disable-copy,13.disable-paste,15.wol-send-packet,12.wol-
mac-addr,18.wol-broadcast-addr,13.wol-wait-time,14.force-lossless
```

Figure 3.4: Server reply for handshake

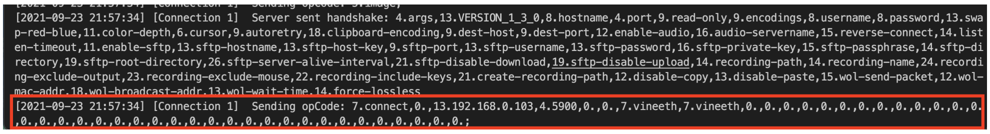


Figure 3.5: Connect instruction for handshake termination

Figure 3.6: Guacd logs, logging connection establishment

Upon receiving these instructions, the server will respond with a protocol version for compatibility between server and client along with other parameters that are required for connection as seen in Figure 3.4. Finally, the *connect* instruction is sent which is the last phase of the handshake, to terminate the handshake process. This instruction is followed by the connection parameters, that is the IP address of the remote desktop, port, username, password of remote desktop, if the parameters are empty it is sent with zero followed by a period and empty value. Zero indicates the length of the value which is sent is zero, which can be observed in Figure 3.5.

On completion of the handshake process, the server will try to establish the connection with the received instructions, upon successful connection it will generate a unique ID for this client connection and reply with a *ready* instruction. This is the end of the handshake process and the connection is considered active until it is terminated, also it marks the beginning of the interactive phase where the interaction between server and client takes place, which can be seen on the client console log Figure 3.7. After the ready instruction, upload and download of data is being carried out. The advantage of this ID is that any other connection that wants to join this session can do so by passing the *select* instruction along with the ID generated to connect to the active connection. The connection process using the connection ID and unique user ID can be seen in guacd logs as shown in Figure 3.6 and also on Figure 3.7 in the console, once the connection is ready and established and there is a constant exchange of data between the server and the client. The entire handshake process after the initial step is configured as explained in listing 3.4. The connection can be terminated by passing the *close* instruction.

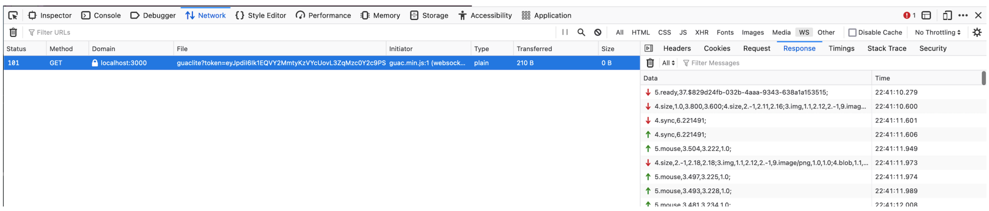


Figure 3.7: Client console log upon connection establishment

```
sendHandshakeReply() {
    this.sendOpCode([
        'size',
        this.clientConnection.connectionSettings.connection.width
            ,
        this.clientConnection.connectionSettings.connection.height
            ,
        this.clientConnection.connectionSettings.connection.dpi

    ]);
    this.sendOpCode(['audio'].concat(
        this.clientConnection.query.GUAC_AUDIO || []));
    this.sendOpCode(['video'].concat(
        this.clientConnection.query.GUAC_VIDEO || []));
    this.sendOpCode(['image']);

    let serverHandshake = this.getFirstOpCodeFromBuffer();

    this.log(this.server.LOGLEVEL.VERBOSE, 'Server sent
        handshake: ' + serverHandshake);

    serverHandshake = serverHandshake.split(',');
    let connectionOptions = [];

    serverHandshake.forEach((attribute) => {
        connectionOptions.push(this.getConnectionOption(
            attribute));
    });

    this.sendOpCode(connectionOptions);

    this.handshakeReplySent = true;

    if (this.state != this.STATE_OPEN) {
        this.state = this.STATE_OPEN;
        this.server.emit('open', this.clientConnection);
    }
}
```

Listing 3.4: Handshake process between guacamole-lite and guacd

3.2.4 *Setup TLS/SSL for web server*

For additional Security the connection between the client and web server was secured by providing a secure HTTP connection over TLS/SSL, that is HTTPS connection. This was done by generating a certificate and a key using

the OpenSSL command as explained here [35]. OpenSSL is an open-source library for creating secure communications using cryptography [47]. However, this is only recommended for testing the connection locally as the certificates were not verified by a trusted Certificate Authority(CA). It will be self-signed as no CA issues certificate for localhost. The following instructions to generate the certificate were referred from [35]. Listing 3.5 shows the instructions used to create the certificates. First, generate a root CA certificate, this is created without using the passphrase by passing -nodes argument in the first instruction along with other parameters that are required to create a certificate. SHA256 algorithm is used to hash the certificate which can be seen in listing 3.5.

```
$openssl req -x509 -nodes -new -sha256 -days 1024 -newkey rsa:2048
    -keyout RootCA.key -out RootCA.pem -subj "/C=US/CN=Root-CA"
$openssl x509 -outform pem -in RootCA.pem -out RootCA.crt
```

Listing 3.5: Generating root CA certificate

Using this root CA certificate, self-signed certificate can be generated. This is achieved by creating a .ext file that contains the information that should be passed for the certificate creation. This is shown in listing 3.6. Once this is done a certificate (.crt), a key (.key) and a Certificate Signing Request (CSR) is created as shown in listing 3.7.

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment,
   dataEncipherment
subjectAltName = @alt_names
[alt_names]
DNS.1 = localhost
```

Listing 3.6: localhost.ext file contents

```
$openssl req -new -nodes -newkey rsa:2048 -keyout localhost.key -
   out localhost.csr -subj "/C=DE/ST=Hessen/L=Frankfurt/O=DaaS-
   Certificates/CN=localhost.local"

$openssl x509 -req -sha256 -days 1024 -in localhost.csr -CA RootCA
   .pem -CAkey RootCA.key -CAcreateserial -extfile localhost.ext
   -out localhost.crt
```

Listing 3.7: Generate self signed certificate

These certificates are loaded to the node server that uses HTTPS to create the server along with express.js. This server configuration is then passed to the guacamole-lite constructor as shown in the listing 3.2.

3.2.5 *Alternate Web Server using Java Servlet*

As explained in the design section 3.3.3, the Apache recommended web server can be developed using Java Servlet and also provides documentation on developing the web server [56]. For the development of this web server, an open-source project called the "guacamole-test-server"[33] is used, which is developed for a similar purpose and which can be found on GitHub. This server uses a web socket tunnel for creating a connection with guacd, as Guacamole provides support for both web socket as well as HTTP connections.

The important part of this Java servlet web server is the implementation of a function that creates a connection. as other functions for read, write are implemented by guacamole. This can be done as shown in listing 3.8 which is used from guacamole-test-server[33]. This function returns a persistent connection with the guacd. It accepts the guacd hostname and the port along with remote desktop connection parameters as config and info. This server was only used to test the connection and in detail, development was not carried out using this web server as the guacamole-lite was looked into in detail.

```
public GuacamoleTunnel createTunnel(final TunnelRequest request)
    throws GuacamoleException {

        //get client information (hostname, port, username...)
        final GuacamoleClientInformation info =
            getClientInformation(request);
        final GuacamoleConfiguration config = buildConfig(request)
            ;

        // Connecting to guacd, proxying a connection to the
            remote server
        final GuacamoleSocket socket = new
            ConfiguredGuacamoleSocket(
                new InetGuacamoleSocket(this.configuration.
                    getGuacdHost(), this.configuration.
                    getGuacdPort()),
                config, info
        );

        // Create tunnel from now-configured socket
        return new SimpleGuacamoleTunnel(socket);
    }
```

Listing 3.8: Initiating Java web socket connection

3.3 FRONT-END

This is one of the most important components of the application, as this component acts an interface between user and application by interacting with the user and displaying the remote desktop. Multiple technologies can be utilized for developing a better interface for the application. Since the application uses the guacamole server, the front-end component must make use of a Javascript library provided by the Apache Guacamole called gucamole-common-js[62]. For the development of front-end two open-source projects were used and compared, with one using EJS called the "gclient"[27] and the other is Angular front-end which called the "ngx-remote-desktop"[42]. The following guacamole-common-js implementation is referred from "gclient"[27] project.

3.3.1 *guacamole-common-js*

The guacamole-common-js provides mouse, keyboard, touch and various other abstractions for implementation so that it can be easily integrated into the front-end component that is being used. Along with the abstraction it provides the implementation of guacamole-client, which is responsible for handling tunneling mechanism, connecting the front-end with the web server and converting the client data into guacamole protocol, that is it converts the input to a set of instructions which is the format of guacamole protocol as explained in the implementation of web server 3.2.2. The development of client application is explained in guacamole manual [56], where the event handling is also explained with code samples.

By importing the *Guacamole* from guacamole-common-js, which contains the client implementation, it can be used to create a tunnel as shown in listing 3.9. As the remote desktop will not be rendered directly by the inclusion of guacamole-client in the DOM, it has to be included manually in the element. Once this is done a web socket tunnel is created by passing the necessary parameters that include the display resolution, connection parameters, host and post. This will create a web socket tunnel which then is passed on to guacamole-client for creating a connection. The *connect()* function of guacamole-client invokes the *connect()* of web socket tunnel created to start the connection. It should also be noted that guacamole does not only support the web socket tunneling but also HTTP tunnel and chained tunnel. But the web socket tunnel is recommended by the Apache guacamole, if this connection is not available the connection falls over to the HTTP tunnel. Similarly, guacamole-client supports all three tunnels.

```
var display = document.getElementById("display");
var guac = new Guacamole.Client(
  new Guacamole.WebSocketTunnel( 'wss://' + host + ':' + port + '/
      guaclite?token=' + connectionstring + '&width=' + $(document
      ).width() + '&height=' + $(document).height()));
```

```
display.appendChild(guac.getDisplay().getElement());

guac.connect();
```

Listing 3.9: Integrating guacamole-client in front-end

3.3.2 *Event Handlers*

3.3.2.1 *Keyboard Events*

As explained earlier guacamole-common-js provides keyboard and mouse abstraction along with other functions, which enables easy integration of such input events. For keyboard events, guacamole offers *Guacamole.Keyboard*, which only recognizes keyup and keydown events. This is because the key-codes are abstracted away from the user as the guacamole protocol only deals with X11 keysyms[69]. In X11 keysyms, each key is unambiguously represented by unique codes, which makes it easier to map a key and find the state of the key if the key is pressed or not. Listing 3.10 shows the integration of keyboard events which is fairly simple.

```
var keyboard = new Guacamole.Keyboard(document);

keyboard.onkeydown = function (keysym) {
  guac.sendKeyEvent(1, keysym);
};

keyboard.onkeyup = function (keysym) {
  guac.sendKeyEvent(0, keysym);
};
```

Listing 3.10: Integrating keyboard events

3.3.2.2 *Mouse Events*

Mouse event abstractions are provided by the *Guacamole.Mouse* object. This object is invoked on onmousemove, onmouseup and onmousedown events. The function *sendMouseState()*, takes the state of the mouse from the display element which can be seen on the code 3.11. This mouse state contains the coordinates of the mouse pointer X and Y in pixels and the mouse button states that include left, right and middle along with the state if they are pressed down or up. All these parameters are sent as a mouse state to recognize the mouse events to the remote desktop. Another important note in mouse events is that, the *Guacamole.Mouse* object does not recognize touchpad mouse events, as this is a true mouse event only abstraction. For touchpad guacamole-common-js provides another abstraction.

```
var mouse = new Guacamole.Mouse(guac.getDisplay().getElement());

mouse.onmousedown =
mouse.onmouseup   =
mouse.onmousemove = function(mouseState) {
  guac.sendMouseState(mouseState);
};
```

Listing 3.11: Integrating mouse events

3.3.2.3 *Touchpad Events*

Similar to mouse events, touchpad events are provided by *Guacamole.Touchpad*. As with the mouse event touchpad event also takes in the state of the mouse as shown in listing 3.12. Touchpad emulates the functionalities of mouse events to map the touchpad events.

```
var touchpad = new Guacamole.Touchpad(guac.getDisplay().getElement
    ());

touchpad.onmousedown =
touchpad.onmousemove =
touchpad.onmouseup   = function(state) {
    guac.sendMouseState(mouseState);
};
```

Listing 3.12: Integrating touchpad events

3.3.2.4 *Touchscreen Events*

With the increase in usage of displays with touchscreen capabilities it is always convenient to have an implementation for touchscreens. For this guacamole-common-js provides *Guacamole.Touchscreen* object which is very similar to touchpad object and can be used interchangeably. The implementation is as shown in listing 3.13

```
var touch = new Guacamole.Mouse.Touchscreen(guac.getDisplay().
    getElement());

touch.onmousedown =
touch.onmouseup   =
touch.onmousemove = function(mouseState) {
  guac.sendMouseState(mouseState);
};
```

Listing 3.13: Integrating touchscreen events

3.3.2.5 *Clipboard*

Since the application is accessible via a web browser the clipboard functionality is quite different. To have the clipboard feature the user interface (UI) must have a text area where the user can save the clipboard content and also receive the clipboard content. To receive the data from the server, guacamole-common-js provides *onclipboard* event handler. This takes in two parameters the stream of data and the mime-type of the data. Listing 3.15 shows the implementation of clipboard. The streamed data is set to the text area and is triggered on change of remote client clipboard.

```
guac.onclipboard = function clientClipboardReceived(stream,
    mimetype) {
  var reader;

  if (/^text\//.exec(mimetype)) {
    reader = new Guacamole.StringReader(stream);

    reader.ontext = function textReceived(text) {
      $('#clipboard').val(text);
    };
  }
};
```

Listing 3.14: Integrating clipboard events

To set the remote desktop clipboard with the input text, *setClipboard()* method from guacamole-common-js is used. This takes in the string from the clipboard text area and ends it to the server as shown in listing **??**.

```
$('#clipboard').on('input', function() {
  guac.setClipboard($(this).val());
});
```

Listing 3.15: Set remote desktop clipboard

3.3.3 *Encrypting client connection parameters*

The connection parameters that need to be passed for creating a web socket tunnel can be encrypted to provide a secure way of sending the information. This is done by using the crypt module available in the node modules as shown in listing 3.16. Initially, a 16 bytes random character is created as the initialization vector (IV), since the Advanced Encryption Standard(AES) Cipher Block Chaining (CBC) method is used for encryption. The initial block needs an IV and the subsequent blocks use the ciphertext from the previous block as IV. Once the IV is generated *createCipheriv()* method is used that

takes the type of encryption, cipher key and IV as input and returns the ciphertext. This is converted to the appropriate format and combined with the IV and sent to the guacamole-lite server. The IV is included along with the ciphertext is because to decrypt the initial block of ciphertext IV is required similar to encryption. However this is not the ideal way to transmit an encrypted text, but in a TLS/SSL connection, this adds an additional layer of security as the user details on the connection are not transmitted as plain text. Since the crypto module is deprecated, crypto.js can be used as a replacement as explained in , which provides Secure Hash Algorithms (SHA) 256-bit encryption algorithm as well. Figure 3.8 shows the encrypted connection string that is being sent while trying to connect to a remote desktop.

```
var clientOptions = {
  crypt: {
    cypher: 'AES-256-CBC',
    key: 'GUACAMOLELITESECRETKEY'
  },
  log: {
    verbose: false
  }
};

var encrypt = (value) => {
  var iv = crypto.randomBytes(16);

  var cipher = crypto.createCipheriv(clientOptions.crypt.cypher,
      clientOptions.crypt.key, iv);

  let crypted = cipher.update(JSON.stringify(value), 'utf8', 'base
      64');
  crypted += cipher.final('base64');

  var data = {
    iv: iv.toString('base64'),
    value: crypted
  };
  return new Buffer(JSON.stringify(data)).toString('base64');
};
guac.connect();
```

Listing 3.16: Encrypting the connection URL

3.3.4 *Security*

The connection between the web server and front-end is secured by providing an HTTP connection over TLS/SSL as explained in 3.2.4. Due to this

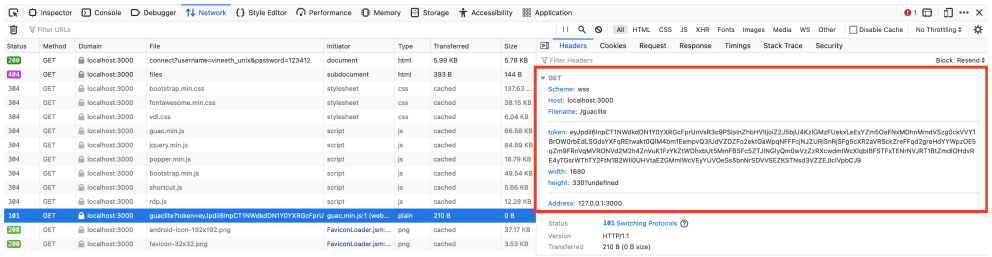


Figure 3.8: Encrypted connection parameters

the web socket server connection will be encrypted by SSL and the URL to the web socket server have to be wss:// as this is more secure than ws:// where the data will not be encrypted and is vulnerable to man in the middle attack. As seen in Figure 3.9, the handshake takes place to establish a secure connection over TLS/SSL. Along with this, the protocol that is used which is guacamole is displayed in Figure 3.10. This shows that the guacamole protocol is started from the client itself, this is in particular to guacamole-common-js which converts each interaction to guacamole protocol, which can be depicted from the Figure 3.7 as well.

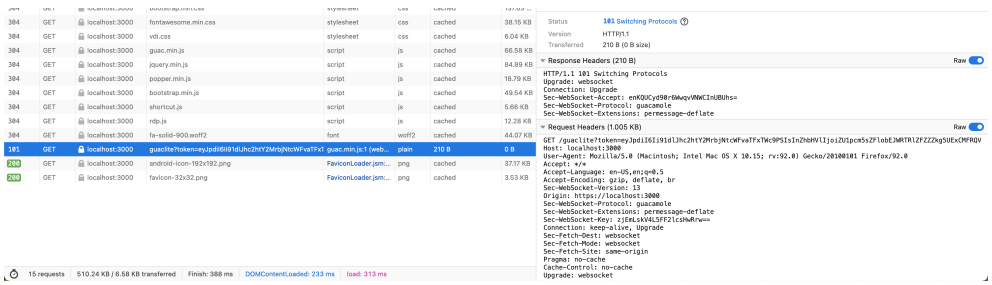


Figure 3.9: Secure web socket handshake process

3.3.5 *EJS*

The development of front-end for this prototype was done using EJS, as explained in design section 2.2.5. This was used for the easy and quick development and testing of the server even though the technology is outdated. An open-source project from the [27] is used as the project uses guacamole-lite and guacamole-common-js in the front-end and this meets the requirements of that are needed for this application. The front-end can be started by using the node server as the component uses the express.js and node to provide the API endpoints to render the embedded HTML as explained in the listing 3.17. The project from [27] only provides a single page application where the remote desktop is rendered, however, to include a login page and registration page from another open-source project [25]. This page is modified to include the University logo and the registration page is modified to include the user details along with the remote desktop IP address and the type of connection (VNC/RDP).

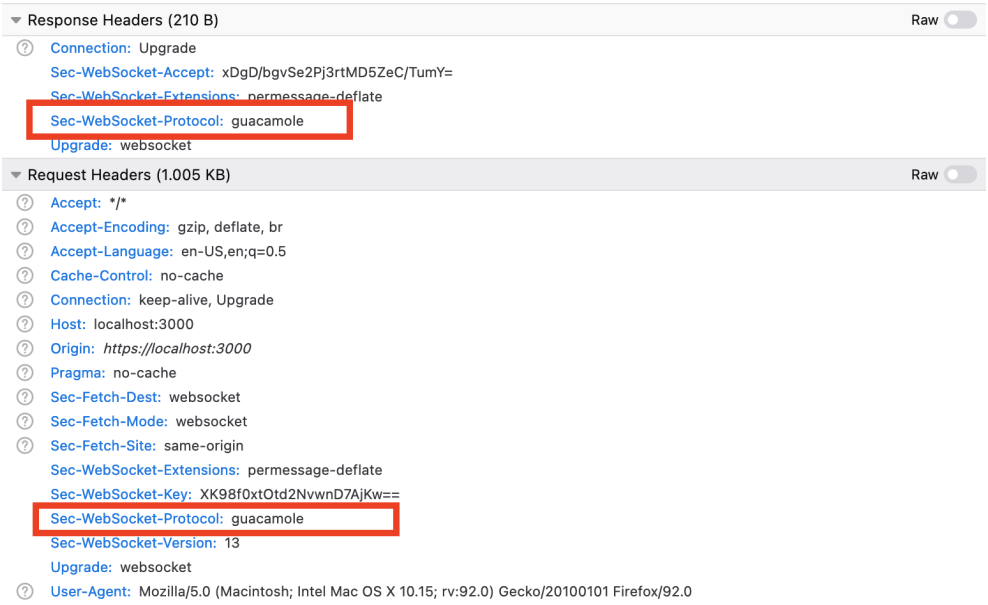


Figure 3.10: Guacamole Protocol is used from client

```
var baserouter = express.Router();

baserouter.get("/", function (req, res) {
  res.render(__dirname + '/login.ejs', {username : '', password: '
      '});
});
baserouter.get("/signup", function (req, res) {
  res.render(__dirname + '/signup.ejs');
});
baserouter.get("/connect", function (req, res) {
  var url_parts = url.parse(req.url, true);
  var query = url_parts.query;
  var connectionstring = encrypt(fetchCredentials(query.username))
      ;
  res.render(__dirname + '/rdp.ejs', {token:connectionstring,
      baseurl:"/"});
});
```

Listing 3.17: Using express.js to render the html

The Figure 3.11 and 3.12 shows the login page and registration page respectively. Figure 3.13 shows the remote desktop along with the clipboard. The clipboard works by pasting the contents that the user wants to paste in remote desktop in the provided text area as this will set the clipboard of the remote desktop as explained in 3.3.2.5 and simply pasting the remote desktop will copy the contents from the text area. Similarly to copy the contents from the remote desktop, the user has to just copy the text in the remote

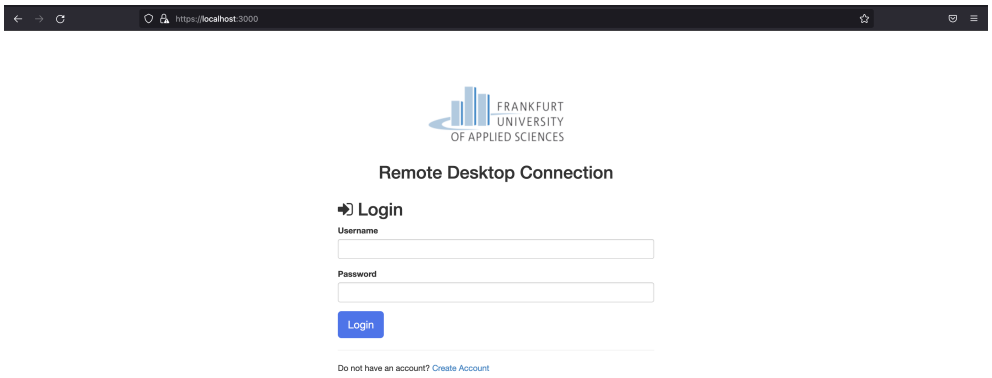


Figure 3.11: Login Page using EJS

desktop which will update the clipboard text area.

For user credentials, validation database authentication is not developed as explained in the architecture 2.1, but the functionality is replicated using a file as shown in listing 3.18 where the credentials are stored as a JSON object and on the registration page, upon sign-up the file is updated with new credentials.

```
{"vineeth_unix":{
    "connection":{
      "type":"vnc",
      "settings":{
        "hostname":"192.168.0.103",
        "port":"5900",
        "username":"vineeth",
        "password":"vineeth",
        "security": "any",
        "ignore-cert": true,
        "enable-stfp":"false" }}}
```

Listing 3.18: User credentials stored in a file

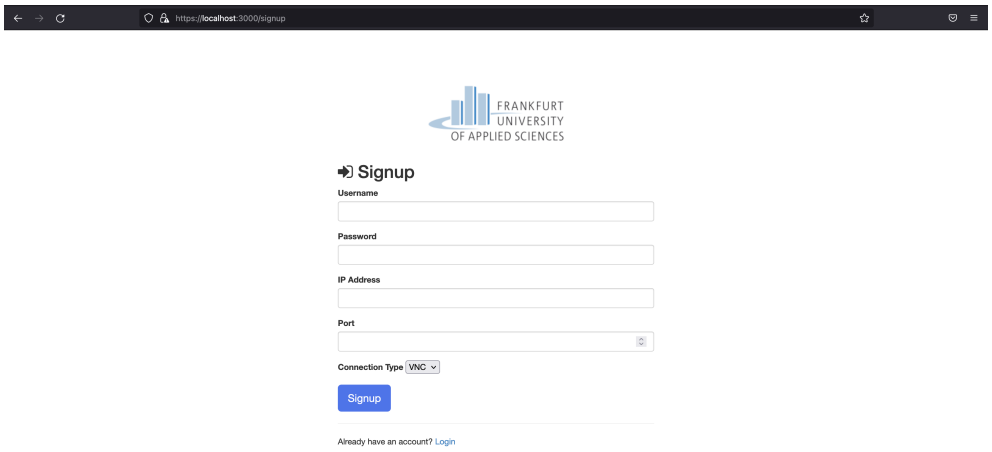


Figure 3.12: Registration Page using EJS

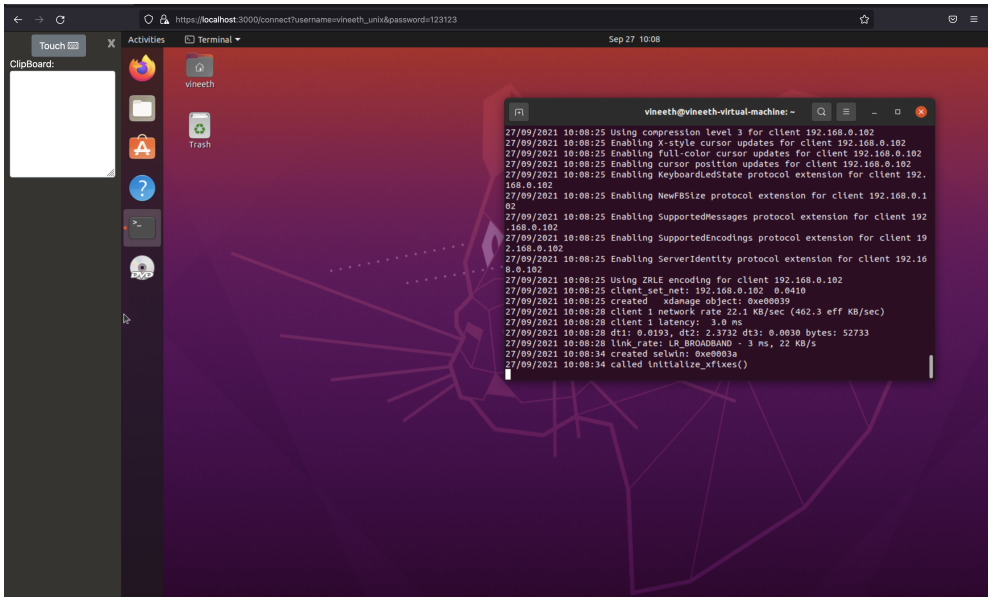


Figure 3.13: Remote desktop using EJS

3.3.6 *Angular*

As explained in the previous section the remote desktop works well with EJS. Another front-end component was integrated that was implemented in Angular. This front-end is also an open-source project that was developed specifically for a guacamole client application called ngx-remote-desktop[42]. This application provides a much richer user interface and is developed as single page application. Also, the project can be installed as a package using npm package installer[41], further documents on installation and configuration can be found here[40]. A demo application with several features are already implemented within the project that is made use of for the development of DaaS.

The application does not make use of guacamole-lite but this can be included with some customization or another web server can be made use of. So the Java web server explained in section 3.2.5 deployed as the web server that accepts the guacd host and port as parameters while deploying. However, the implementation of event handlers is similar as the components make use of guacamole-common-js. All the necessary functions that are required for the connection of remote desktop, as well as event handlers, are defined in the service module called remote-desktop-manager.service.ts. This module is imported and is made use of in the application page as shown in the listing 3.19. Upon initialization of the page function *ngOnInit()* is invoked in the app.components.ts file which includes the tunnel connection, handler for *connect()* method and initialization of clipboard which can be seen in listing 3.19. The client parameters are set in the *handleConnect()* method as shown in listing 3.20. Once connected the desktop will be displayed as shown in Figure3.14 and Figure3.18 shows the clipboard that is used.

```
ngOnInit() {
        const tunnel = new WebSocketTunnel('ws://localhost:8080/ws
            ?hostname=192.168.0.103&port=5900&type=rdp&width=1024&
            height=768&audio=audio/L16');

        this.manager = new RemoteDesktopManager(tunnel);

        this.handleConnect();

        this.manager.onRemoteClipboardData.subscribe(text => {
            const snackbar = this.snackBar.open('Received from
                remote clipboard', 'OPEN CLIPBOARD', {
                duration: 1500,
            });
            snackbar.onAction().subscribe(() =>
                this.handleClipboard());
        });
```

```
        this.manager.onReconnect.subscribe(reconnect =>
            this.handleConnect());
    }
```

Listing 3.19: Connection initialization in Angular

```
handleConnect() {
        const parameters = {
      hostname: '192.168.0.103',
      username:'vineeth',
      password:'vineeth',
            port: 5900,
      type: 'vnc',
            image: 'image/png',
            audio: 'audio/L16',
            dpi: 96,
            width: window.screen.width,
            height: window.screen.height,
      security: 'any',
      'ignore-cert': true
        };

        this.manager.connect(parameters);
    }
```

Listing 3.20: Client parameters and connect() in angular

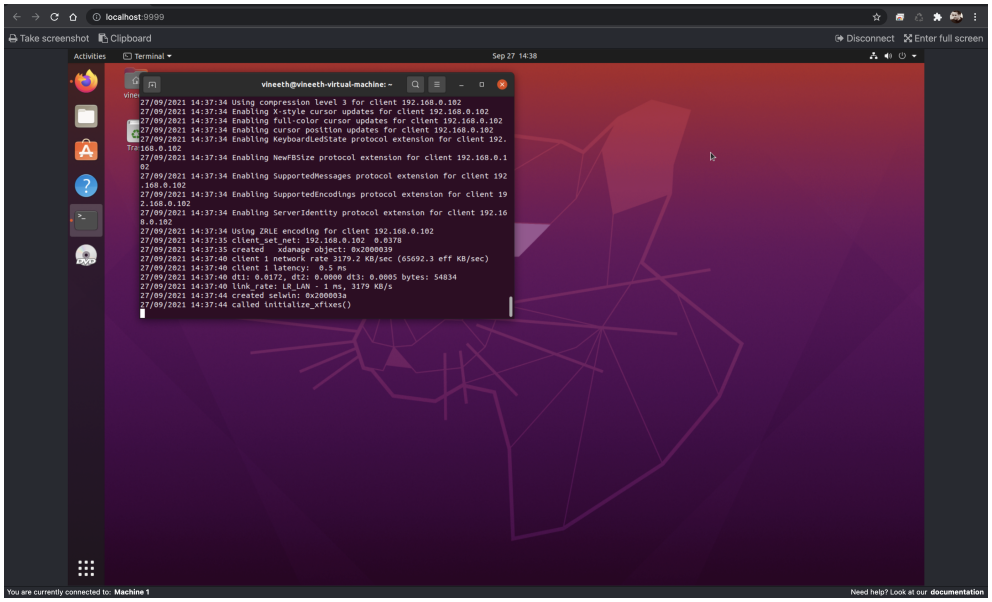


Figure 3.14: Remote desktop user interface of ngx-remote-desktop

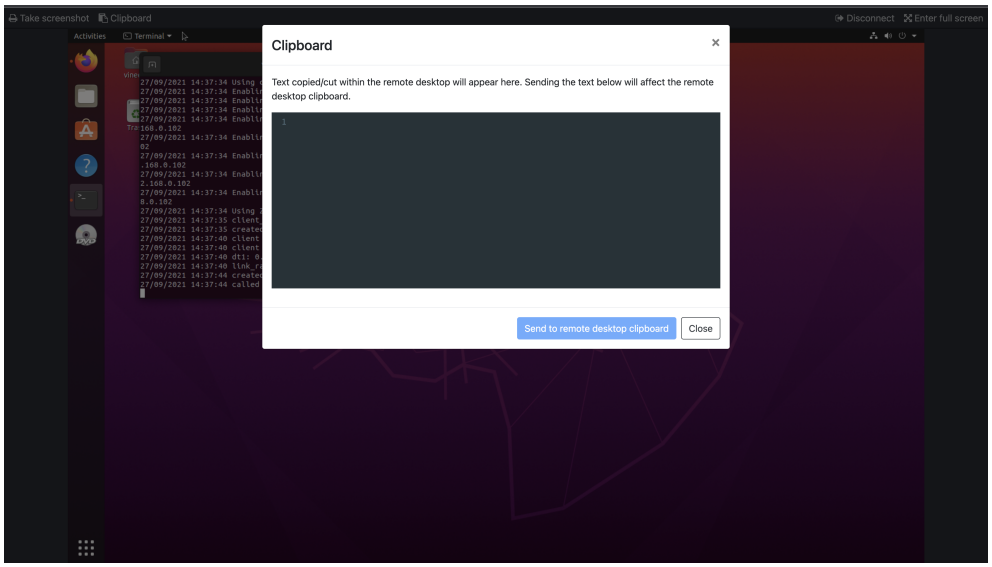


Figure 3.15: Clipboard of ngx-remote-desktop

3.4 ONEYE

Oneye provides applications as shown in Figure 3.16, that are developed in PHP specifically for Oneye following the developmental guidelines provided by eyeOS in their developer manual[60]. However it is outdated with the complex code base and the application do not support modern functionalities which have made it fall out of favor. For example, the browser in the Oneye does not connect to the websites available, this is due to the websites not being able to recognize the browser which can be seen in the Figure 3.17. As stated earlier in architecture 2.1, it can be combined with the DaaS developed as a web server by developing an application in the Oneye for connecting to DaaS. This will enable the Oneye to connect to an actual desktop and make it more convenient and powerful. First to deploy the Oneye, as it is compatible with php5 and not with the higher versions, as the functions that are used in developing applications are not available with the latest versions. There are a couple of ways of deploying it, one is by using the Apache server by installing XAMPP[58] and the other is by deploying it in a docker container that uses php5 and Apache image as a base.

3.4.1 *Deployment using XAMPP*

Initially, Oneye was deployed by using XAMPP which is used as an Apache web server to host the Oneye. The following instructions are tested and executed on a macOS and have not been tested in any other operating system. XAMPP is installed using the appropriate installer which is available from the download page of XAMPP. Since Oneye is compatible with php5 XAMPP 5.6.40 is used [57]. This will be an executable .dmg file, by double-clicking, the file will open the installation process. By following the on-screen instruc-

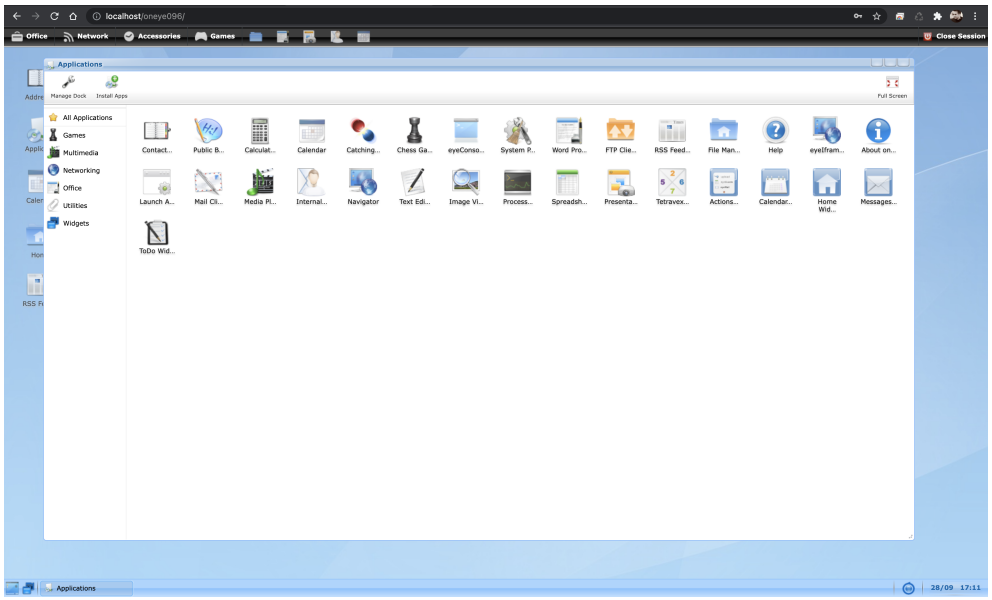


Figure 3.16: Applications provided by Oneye

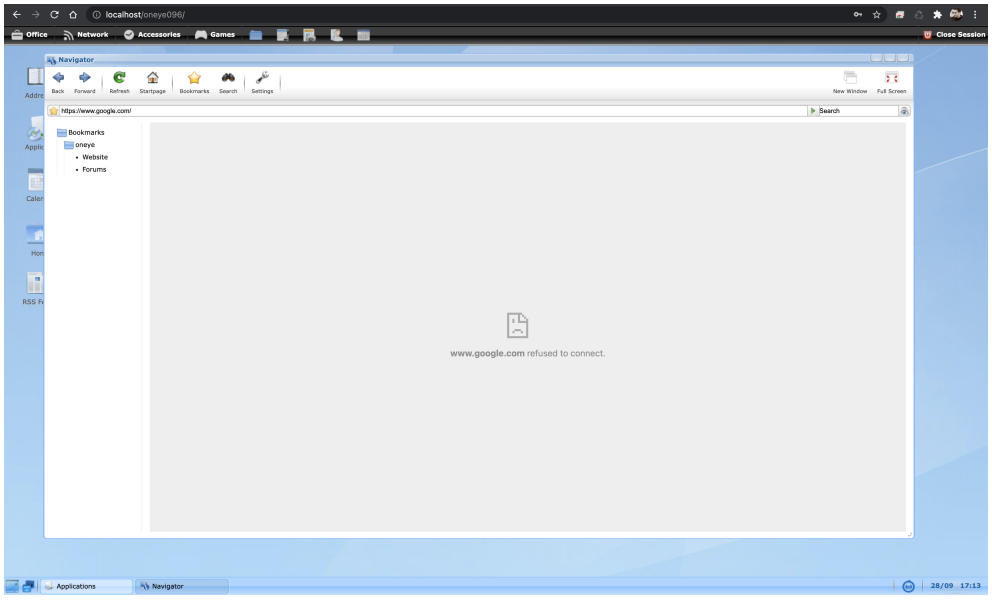


Figure 3.17: Oneye Web browser

tions XAMPP will be successfully installed and upon starting the application the Figure 3.18 will be displayed. The server can be managed by switching to the "Manage Servers" tab and by selecting the required server, configuration can be opened by clicking on the "Configure" button, the server can be started or stopped through the UI provided by XAMPP as shown in Figure3.19.

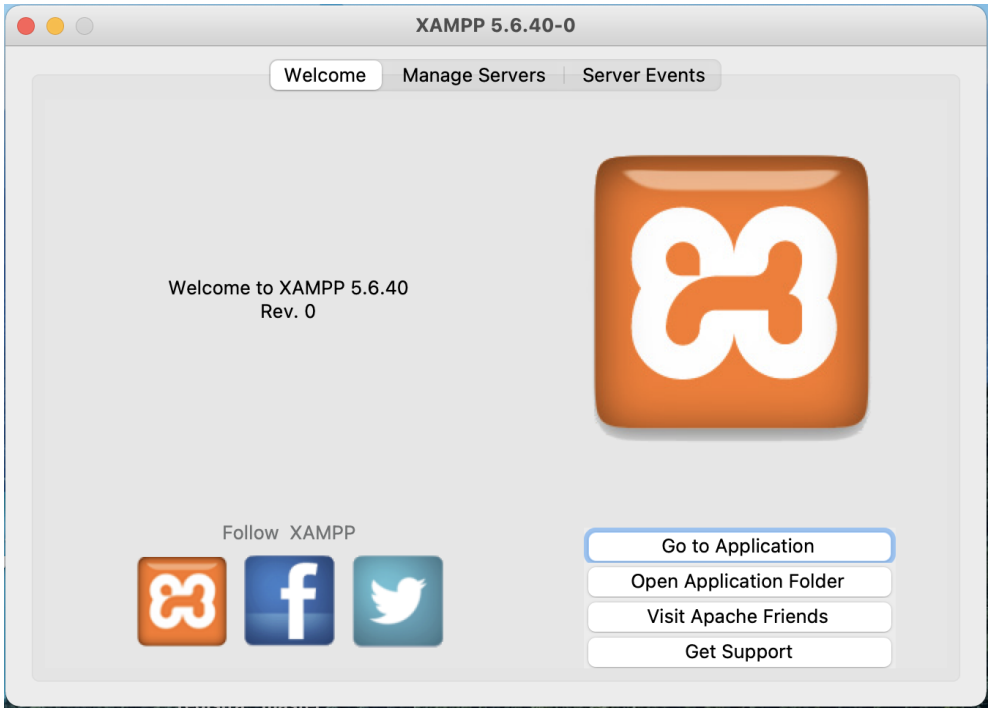


Figure 3.18: XAMPP application

Now the Apache server is setup, download the Oneye project zip file from the GitHub [44], which is a release version, source code from the git can also be downloaded[45] this contains all the applications in extracted form as opposed to the installer in which the application is compressed "package.eyepackage" file. Extract the Oneye project folder to XAMPP/htdocs in the Application folder. Modify the ownership htdocs as well as Oneye to include the current user and group as owners. Once this is done start the Apache server and enter the URL localhost/<Oneye folder name inside XAMPP>. This will load the Oneye login page.

3.4.2 *Deployment using Docker*

Deployment of Oneye using Docker is much easier. Here the instead of setting up an older version of the XAMPP Apache web server, the Apache web server is containerized to run Oneye. But the PHP version used must still remain 5. For this reason, a docker image of PHP version 5 in conjunction with Apache HTTPd is used.[18]. The listing 3.21 shows the docker file instructions used to deploy the Apache server and load Oneye to the container.

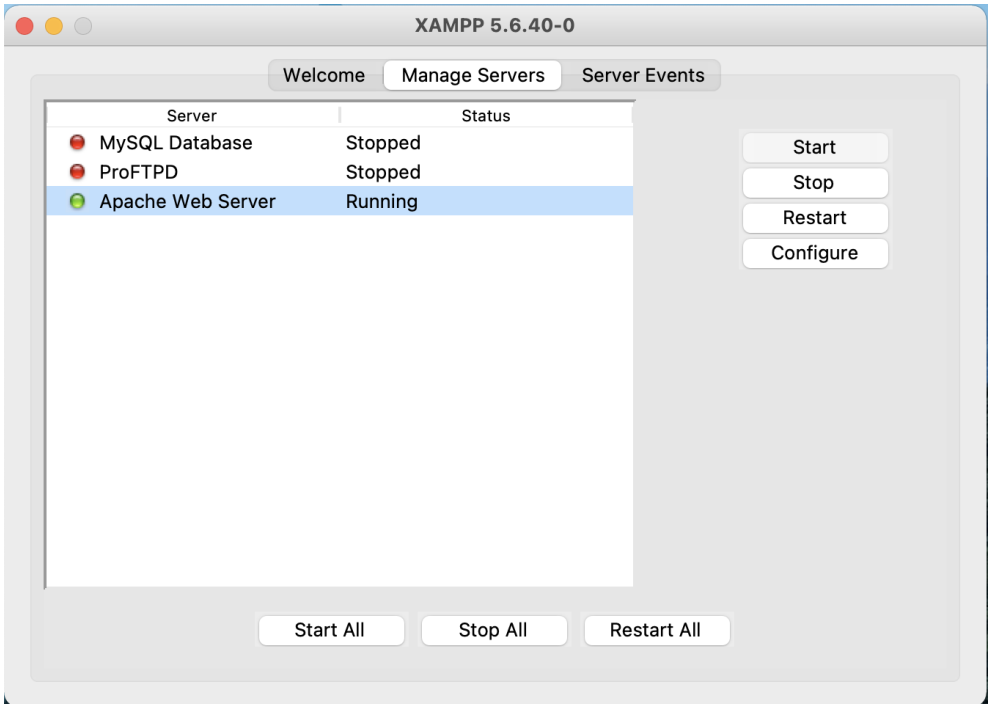


Figure 3.19: XAMPP Apache server

```
FROM php:5-apache
WORKDIR /var/www/html/
COPY . /var/www/html/
RUN chmod -R 777 /var/www/html
EXPOSE 80
```

Listing 3.21: Docker file for Oneye deployment

As seen in listing 3.21, php:5-apache will use the PHP version 5 docker image from the docker hub along with Apache. Once the working directory is set, the contents from the local current directory are copied to the specified location. This contains the Oneye source code downloaded from [44]. Upon building and running the docker container Oneye will be deployed as shown in Figure 3.20.

3.4.3 *Oneye Application*

To use Oneye as a web server, an application in Oneye has to be developed using the developer manual[60] and following existing applications for reference. For the development of this application, eyeIframize application is used, which has a similar behavior as that of eyeDaas. All the applications that are used in the Oneye are placed within a single folder called apps that is inside the system within the main Oneye folder as shown in listing 3.22. Each application contains three main files that are app.eyecode, events.eyecode and info.xml. To develop a fully functioning application these three files contain certain set of functions which are as follows.

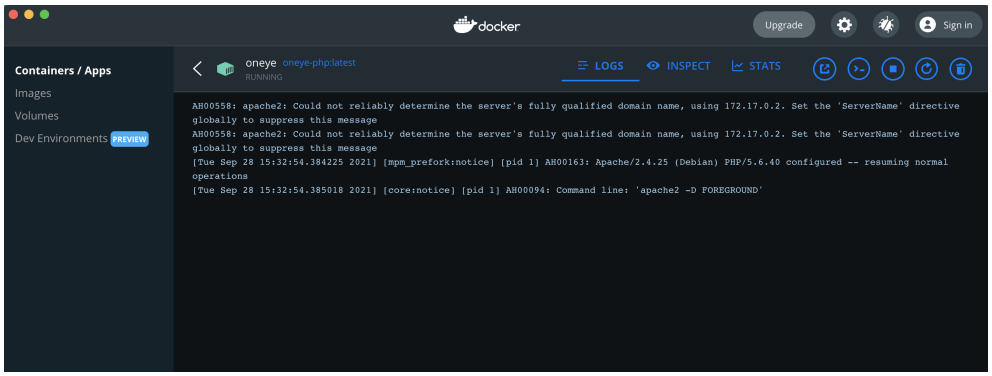


Figure 3.20: Oneye Docker container

```
oneye
 |__System
     |__apps
         |__eyeIframize
         |__eyecalendar
         |__eyeDaaS
         |...
```

Listing 3.22: File structure of oneye applciations

app.eyecode

app.eyecode is the most important file of all that is used to initialize and end an application. It contains two main functions namely ApplicationName_run and ApplicationName_end, with the _run function being the required function to initialize the application. Along with this the user interface is defined in app.eyecode. ApplicationName_run is invoked by the PROC when the application is launched. Similarly, ApplicationName_end is invoked when an application terminates. PROC is taking care of the process management and also takes care of launching, terminating, listing applications to name a few tasks. The code for the user interface is placed within the ApplicationName_run as this will load the interface while initializing the application. Taking a look at the development of eyeDaaS, the app.eyecode for this contains fairly minimum code as the application itself is simple. When the user opens the application, the application should connect to the DaaS application.

```
 function eyeDaas_run($params = '')
{
        if ($params[0]){
                include_once(EYE_ROOT . '/' . APP_DIR . '/eyeDaas/
                    events' . EYE_CODE_EXTENSION);
                eyeDaas_on_Do($params);
        }
```

```
        else{
                eyex('messageBox',
                    array(
                'buttons' => array(array('Do','Connect'),array('
                    Close','Cancel')),
                'content' => 'Do you want to connect to remote
                    desktop ?',
                'img' => 'index.php?theme=' . $_SESSION['usertheme
                    '] . '&extern=icons/48x48/daas.png',
                'title' => 'eyeDaas',
                'type' => 3,
                'win_name' => 'eyeDaas_Window',
                'win_style' => TITLE + CLOSE + MIN + LISTED
                    )
                );
        }

}

function eyeDaas_end($params = '') {
        eyeWidgets('unserialize',$params);
}
```

Listing 3.23: app.eyecode for eyeDaaS applciations

Listing 3.23 shows the app.eyecode for eyeDaaS. The function *eyeDaas_run()* initialized the eyeDaaS rendering the interface defined within the function. Care must be taken while definging the _run function for an application as the name preceding the _run must also be the name of the application folder. As shown in listing 3.22 and 3.23. The user interface is defined within the else block. Initially, when the application is launched, a window is opened asking the user to connect to DaaS or cancel as shown in Figure 3.21. To design this an array that contains the parameters for the component that needs to be displayed is defined. Here a messageBox has to be displayed hence the messageBox name is provided as a component name followed by the array that defines the contents of that component. This entire component is passed to a service called *eyex()*. In Oneye the front-end is rendered using XML which contains instructions to modify the UI. Handling of this XML is the responsibility of *eyex()*, which is sent from the server. The applications request *eyex()* for operations like rendering a window or message box and these instructions or requests are converted into XML and sent to UI. Once this is done the application should look like the Figure 3.21. Upon clicking of Connect, the button is linked to an event while defining the button as shown in listing 3.23. This event is defined in the events.eyecode.

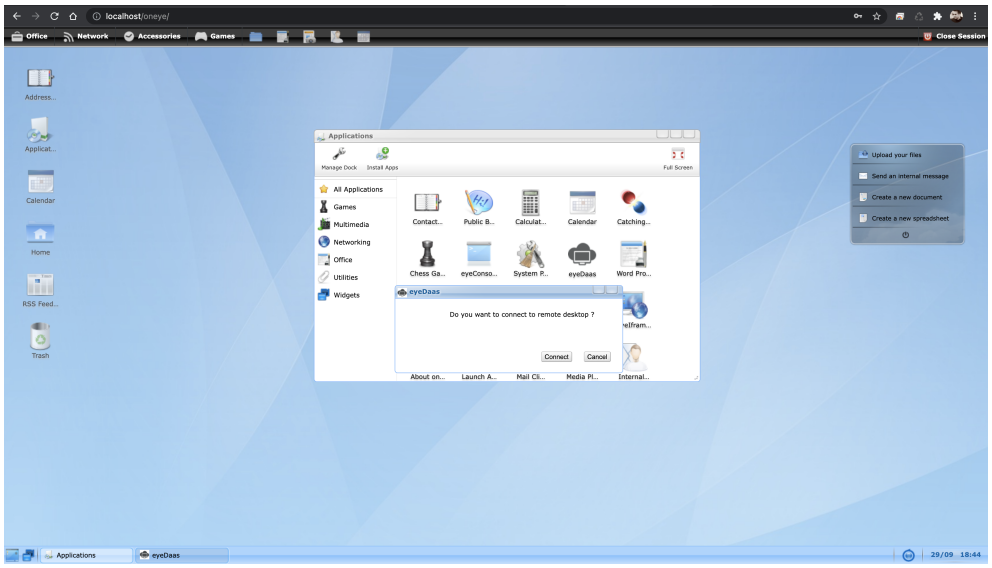


Figure 3.21: eyeDaaS application initialization

events.eyecode

events.eyecode processes the events received from the client. When the user clicks a button on the UI, an event binding to that button is triggered, which is sent as an event message towards the server where, the MMAP is responsible for organizing and delivering the messages to appropriate applications. The event messages will have a checknum of the application in them and the name of the event to recognize within the application, MMAP uses this checknum and event name to find the right event.eyecode file in the respective application directory that is mentioned in the message and triggers the function ApplicationName_on_EventName, with the body of the message as the argument to the function. So for the eyeDaaS application the function in app.eyecode *eyeDaas_run()* binds the connect button with Do event which is the event name as shown in listing 3.23 and that event is defined in events.eyecode as *eyeDaas_on_Do()* as shown in listing 3.24. When the user clicks on the connect button in the message box of the eyeDaas application, MMAP will receive the message that the user wants to connect to DaaS and the Do event is triggered, MMAP then finds the Do event within the events.eyecode of eyeDaas application and invokes the method. The *eyeDaas_on_Do()* checks the URL to determine if the URL is to a file by checking the protocol used. If it is a file then a Hidden widget is defined and the file is loaded onto this widget to be displayed. If not then the URL is processed for displaying by creating an iframe and a new window.

```
function eyeDaas_on_Do($params = '') {
  ....
      if ((strtolower(substr($url, 0, 6)) !== 'ftp://') && (
          strtolower(substr($url, 0, 7))  !== 'http://') && (
          strtolower(substr($url, 0, 8))  !== 'https://')) { //
          utf8
```

```
                $file = basename($url);
                $path = eyeFiles('cleanPath', array(substr(trim($
                    url, '/\\'), 0, -strlen($file)))); // utf8
                if (vfs('fileExists',array($path[0] . '/' . $file)
                    ) || $path[1] == 'real' && vfs('real_
                    fileExists',array($path[0] . '/' . $file)))
                {
                        $myHidden = new Hidden(array(
                                'father' => 'eyeDaas_Window_
                                    Content',
                                'name' => 'eyeDaas_Hidden',
                                'text' => $path[1] . '://' . $path
                                    [2] . '/' . $file
                        ));
                        $myHidden->show();
                        $title = $file . ' - eyeDaas';
                        $url = 'index.php?checknum=' . $checknum .
                             '&msg=getFile';
                }
        }
        .....
}
```

Listing 3.24: events.eyecode connect button function

Within the *eyeDaas_on_Do()* function, a new window is created for displaying the DaaS application. To create this window eyeOS offers a toolkit that helps the developers in defining the UI that is needed. As shown in listing 3.25, a window widget is created by using the respective class with an array of parameters. These parameters include,

- name: Name of the class.
- title: Title given to widget.
- width: Width of the widget.
- height: Height of the widget.
- cent: Centers the widget by ignoring the X,Y coordinates of the widget.
- father: Specifies the father widget within which current widget to be displayed. For main widget the father is always eyeApps.

This newly created widget is stored in a variable and is displayed using the *show()* method. Once the window is displayed a javascript function is executed to maximize the window. For this a function named *runjs*, is used which takes in the Javascript function name that needs to be executed and its parameters and executes that function. At the end, an IFrame is created within the window widget as shown in listing 3.26.

```
....
        $myWindow = new Window(array(
            'name' => 'eyeDaas_Window',
                'cent' => 1,
                'title' => $window_title,
                'father' => 'eyeApps',
                'height' => $window_height,
                'savePosition' => RELATIVE,
                'sendResizeMsg' => 1,
                'sigResize' => 'Resize',
                'width' => $window_width
        ));

        $myWindow->show();
    .....
        if ($start_maximized == TRUE) {
                eyex('runjs', array('js' => "Windows.Maximize('".$
                    myPid."_".$myWindow->name."');"));
        }
....
```

Listing 3.25: events.eyecode: creating new window

```
        $myIframe = new Iframe(array(
                'father' => 'eyeDaas_Window_Content',
                'name' => 'eyeDaas_Iframe',
                'height' => $myWindow->height - 24,
                'url' => $url,
                'width' => $myWindow->width - 3,
                'x' => 0,
                'y' => 0
        ));
        $myIframe->show();
        $myIframe->focus();
```

Listing 3.26: events.eyecode: creating an Iframe

info.xml

info.xml is used as an information file for the application it defines the application category, version, author, license, type of application, and the icon that needs to be loaded for the application as shown in listing 3.27. The icon should also be included in "\oneye\system\extern\apps\eyeX\themes \default\icons\48x48" for the icon to be displayed, which is the case for eyeDaas as shown in Figure 3.21.

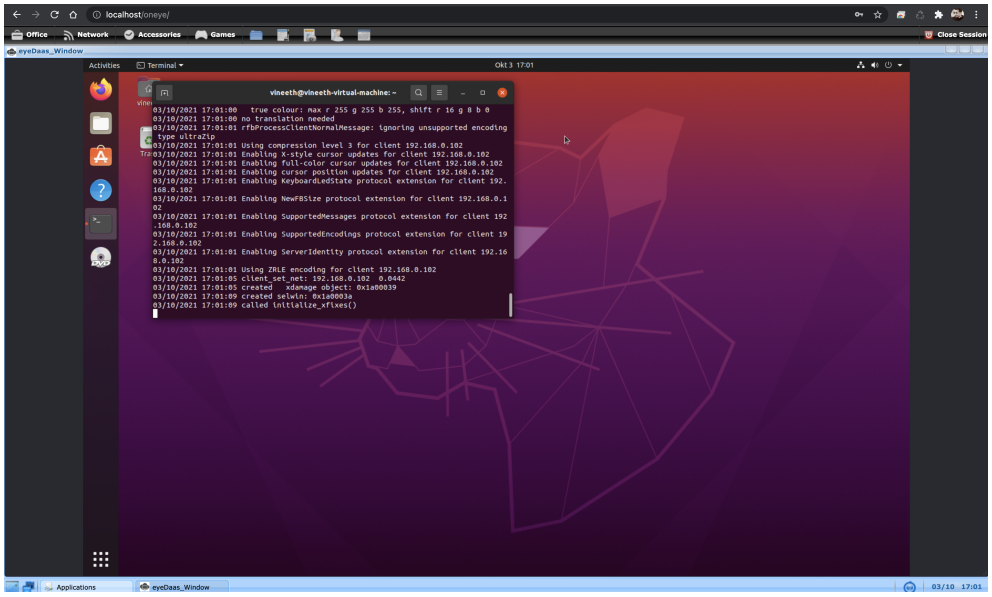


Figure 3.22: eyeDaaS application

```
<package>
        <name>eyeDaas</name>
        <category>Utilities</category>
        <version>1.9</version>
        <description>Remote Connection Utility</description>
        <author>oneye Team</author>
        <license>AGPL</license>
        <type>Application</type>
        <icon>index.php?theme=USERTHEME&amp;extern=icons/48x48/
            eyeDaas.png</icon>
        <openWith>1</openWith>
</package>
```

Listing 3.27: info.xml file of eyeDaaS

Figure 3.22 shows the eyeDaas application in Oneye which enables the user to connect to an actual operating system and make full use of the system and also improves the capability of Oneye.

3.4.4 *Problems in developing eyeDaas*

Development of application in Oneye was simple. But there was one issue for which a solution could not be found. This was connecting from eyeDaas application to an SSL/TLS connection, that is a connection to HTTPS was not achieved. Initially, the issue seemed to be connecting to any website but this was eliminated when the Iframe of eyeDaas connected to an unsecure connection (HTTP), which was the Front-end of DaaS application that used angular. On further review and using various solutions like using the IP address of the system to connect but this yielded no result.

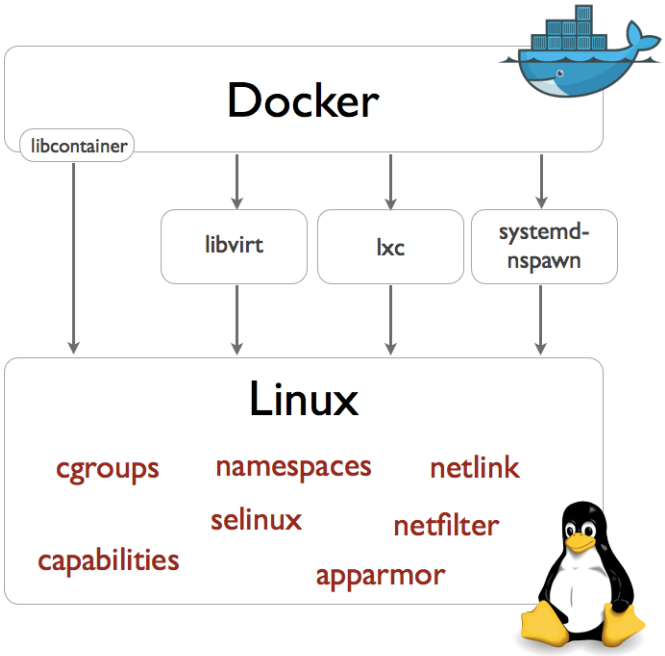


Figure 3.23: Docker utilizing Linux kernel virtualization features[15]

3.5 DOCKER

As explained in 2.2.6, docker uses containers, which is a "package of software for development, shipping and deployment"[52]. Docker enables the developers to develop the software without having to focus too much about the environment that it runs on and eliminates the problem of having compatibility issues with the underlying hardware and, the performance and behavior of the software remain the same regardless of the infrastructure. Docker facilitates smooth and quick development and operation of the applications, this is achieved by OS virtualization. Docker is written using Go programming language, when running it uses the system kernel for resource utilization and allocation. In Linux kernel it uses resource isolation features such as namespaces and cgroups when a container is created and docker can make use of different interfaces to utilize the virtualization of Linux kernel as shown in Figure 3.23.

Figure 3.25 shows the architecture of docker or more specifically a docker engine, that is a client-server architecture. Docker engine is a product offered by Docker Inc. that includes the Docker CLI and the docker daemon, dockerd. To interact with the docker containers and docker daemon, developers use docker client which can be the CLI or the docker desktop. This client interacts with the docker daemon called dockerd using REST API over a network interface or UNIX sockets. Docker daemon handles the majority of the work in docker such as building images, running images and man-

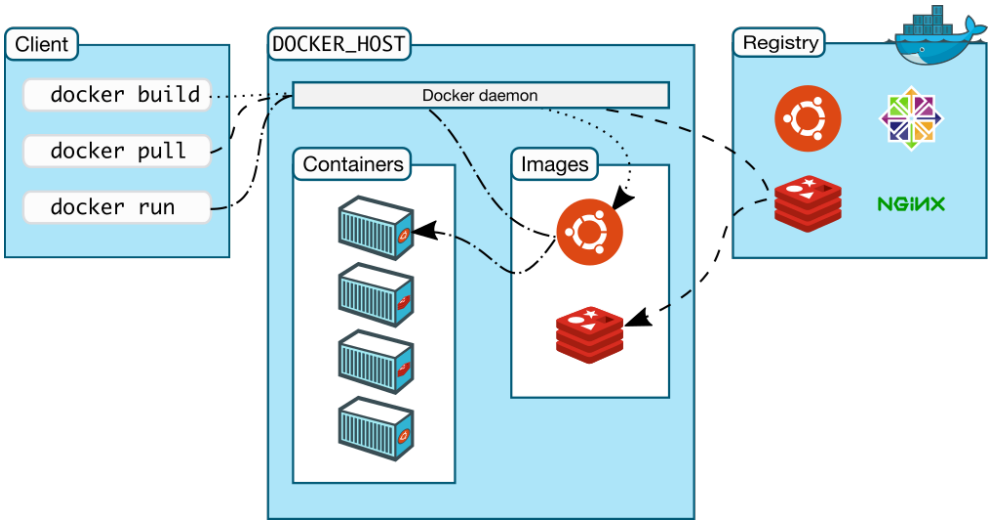


Figure 3.24: Docker engine architecture[16]

aging the containers. Then there is the docker registry where the docker container images can be stored called the docker hub. As seen in the Figure 3.25, when the commands like docker build, pull and run is executed in the client, docker daemon receives the commands and downloads the images from the docker registry if it is not locally available and then builds them to create the containers and runs the containers.

The containers created are isolated from one another for better utilization of resources, which is similar to that of a virtual machine. Docker has several advantages over VMs even though the behavior they provide at the end is quite similar. To name a few advantages of using containers,

- Lightweight: Containers are lightweight compared to running individual VMs.
- : Better Memory Utilization: The container images will utilize less memory and the containers also utilize less memory in comparison to VMs, as they have a full OS with libraries binaries, hence the docker will be lighter compared to VMs.
- Fast: Since containers are lightweight and utilize less memory they are considerably faster in performance, whereas the VMs are slow to boot, as they have an entire OS to load.
- : Due to less memory utilization several containers can be run on a single machine that can share the OS kernel space.

Figure 3.25, shows the difference between using containers and VMs to host few applications, as it is clear from the figure the requirements needed to deploy an application using containers and VMs.

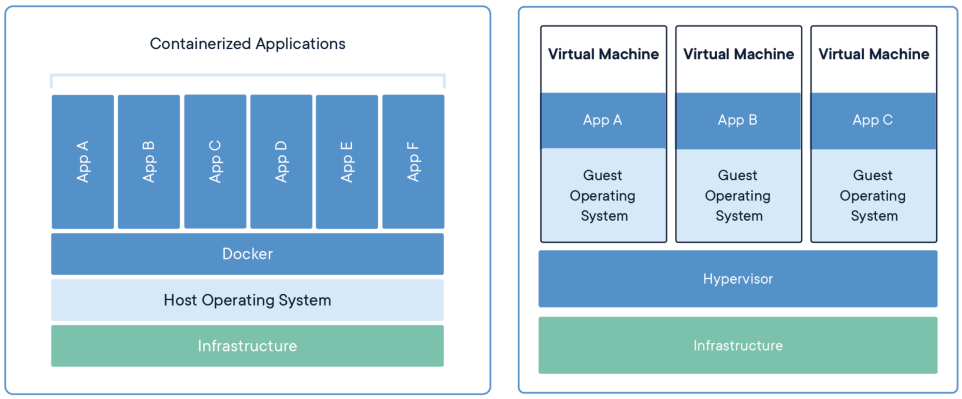


Figure 3.25: Comparison between Containers and VMs[52]

3.5.1 *Development using Docker*

For the development of docker containers and building images of containers for the application, docker allows the developers to define instructions in a file called Dockerfile [23]. Dockerfile contains instructions similar to that of instructions executed in CLI, docker daemon recognizes the instructions and executes them one after the other to build the required container image. Usually, the Docker file is placed within the application folder for which the image is to be created as the build command builds from the files that are in the context. The command used for building an image from Dockerfile is "docker build .". If a specific name has to be provided then the argument -t along with the name can be passed to the image(assuming the command is passed where the Dockerfile exists). Similarly, other options can be found in [17]. Once the docker image is built using "docker run <image-name>" with necessary arguments like exposing the ports with -p option [19], containers can be started.

3.5.1.1 *Front-end*

Listing 3.28 shows the Dockerfile for the front-end component developed using EJS as explained in . This is a simple Dockerfile, initially, the node image is loaded, as it is a node application. Once this is done the working directory is set and the package.json file is copied and the dependencies are installed using the npm install instruction. Then the contents of the application are moved to the container folder and port 3000 is exposed then the node command is executed. Similarly, the angular component is built and deployed using Dockerfile which is as shown on the listing 3.29.

```
FROM node:12.18.3-alpine
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
```

```
CMD ["node", "app.js"]
```

Listing 3.28: Dockerfile for EJS front-end

```
FROM node:12.18.3-alpine
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 9999
CMD [ "npm", "start" ]
```

Listing 3.29: Dockerfile for Angular front-end

3.5.1.2 *Web Server*

Guacamole-lite is included in the front-end node server so, the deployment is done along with the front-end component which is shown in the listing 3.28. However, the alternate java servlet web server explained in 3.2.5 is deployed using Dockerfile, which is as shown in the listing 3.30. The web server is built using maven and then the jar file is moved to the container. And the application is executed using the command at the end.

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
COPY target/gts.jar gts.jar
EXPOSE 8080
ENTRYPOINT exec java -jar gts.jar --guacd-host 172.18.0.3 --guacd-
    port 4822 --port 8080
```

Listing 3.30: Dockerfile for Java Webserver

3.5.1.3 *Oneye*

For the deployment of Oneye, docker is used as explained in 3.5. Deployment of Oneye using docker was much simpler and easier compared to setting up XAMPP Apache Server. Listing 3.21 refers to the Dockerfile that is used for Oneye. It uses php5 image in conjunction with Apache for the server.

3.5.1.4 *Docker compose*

To run multi-container docker applications, docker offers a tool called Compose [20]. A YAML file can be used to configure docker-compose where all the containers, network, database and volumes are defined and each container can be configured to build from their respective Dockerfile by providing the path to the Dockerfile in the context. Listing 3.31 shows the docker-compose file used for the deployment of the client with Angular, the Java

servlet web server and the guacd. Figure 3.26 shows the multi-container deployment used. To build and deploy this, a single command "docker compose up" is used and "docker compose down" brings down all the containers. Each container is defined under services with a specific name followed by the container configurations.

```
version: '3.8'
services:
    client:
      container_name: client
      build:
        context: ./ngx-remote-desktop-master
      expose:
        - "9999"
      ports:
        - "9999:9999"
    guacd:
      container_name: guacd
      image: glyptodon/guacd
      environment:
        ACCEPT_EULA: "Y"
      expose:
        - "4822"
      ports:
        - "4822:4822"
    server:
      build:
        context: ./guacamole-test-server-master
      container_name: server
      expose:
        - "8080"
      ports:
        - "8080:8080"
```

Listing 3.31: Docker Compose file used for the application

3.5.2 *Problems in deploying multi-container setup*

One of the main issues that was faced while setting up the docker containers was that the web server container was not able to connect to the guacd container. Even with the ports exposed, the web server was not recognizing the guacd host(0.0.0.0). However, when the web server was not deployed as a docker container guacd server was reachable at 0.0.0.0. Further analyzing the issue, it was noticed that all the docker containers have their internal network, as initially there was no network was set up, so to test this a bridged network was set up between all the containers. This attempt could not resolve the issue. Later moving along the same direction it was noted that the

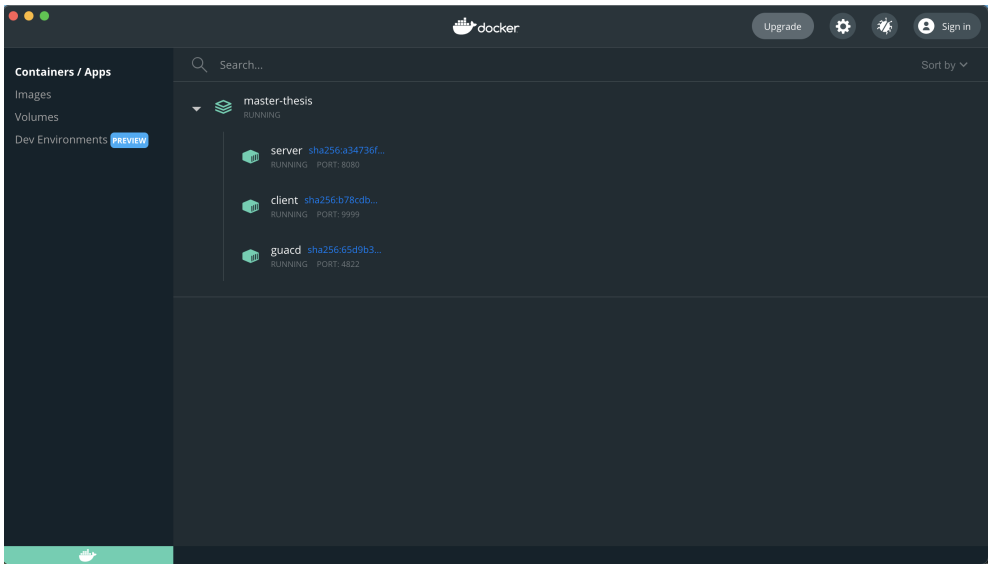


Figure 3.26: Multi-container docker setup using Docker compose

internal network will assign each docker container with an IP address and the IP address of each container were found out using the docker CLI and using the IP address assigned to guacd container, the web server was made to connect to guacd, in this case, there was a success in connection, but this IP address assigned to containers are dynamic as every time the containers are removed a new IP address might be assigned. Yet the issue of connecting to guacd container directly was not solved. Several attempts were made with various solutions such as by using the –link which was unsuccessful. Later in the end it was realized that setting up a custom network might work, however, this theory has not been tested to provide the results.

4

RESULT AND DISCUSSION

This section provides the result obtained from the development of a concept of a Desktop-as-a-Service and discusses the problems and advantages in the application.

4.1 RESULT

To test the application a virtual machine with Ubuntu and Windows OS is set up using VMware Fusion[68]. The reason for using two different operating systems is to test out the different connections that are VNC and RDP. However, a VNC server could still be installed in windows to test the application. To begin initially the X11VNC server is set up in Ubuntu. X11VNC is a VNC server that allows remote access to remote desktops. Installation is simple by using the command " sudo apt install x11vnc" once installed it can be started without any password by running the command "x11vnc" which will show the screen as shown in Figure 4.1. The same server can be started with a password by passing the argument "-usepw" along with "x11vnc".

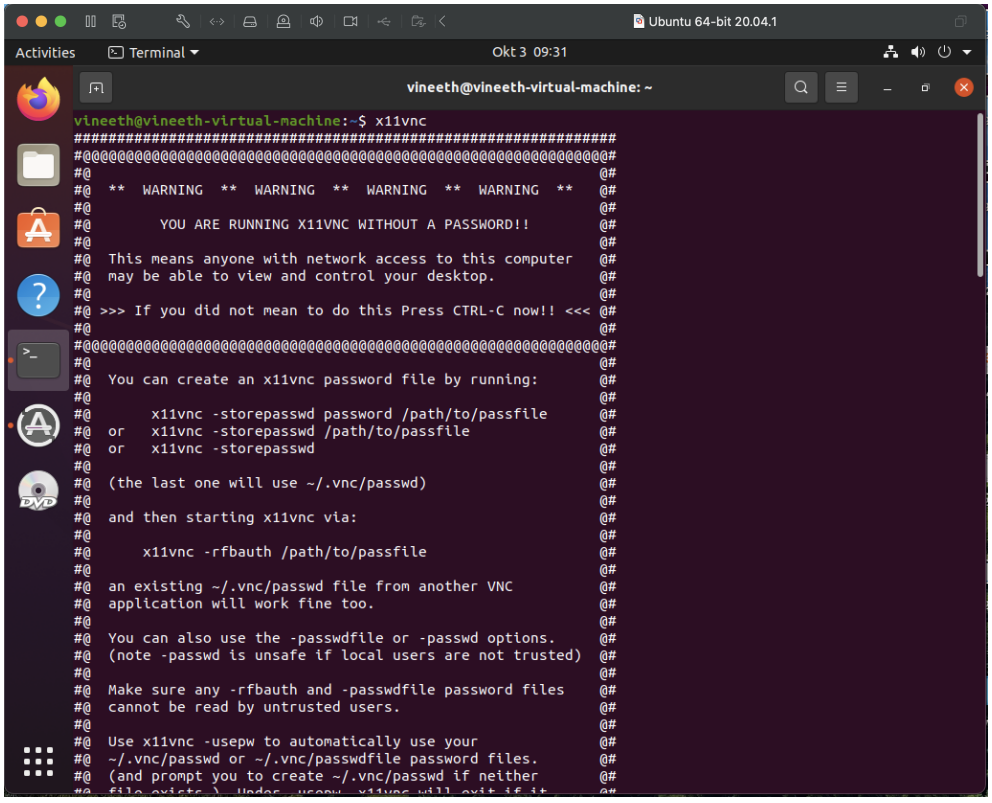


Figure 4.1: X11VNC server starting in Ubuntu

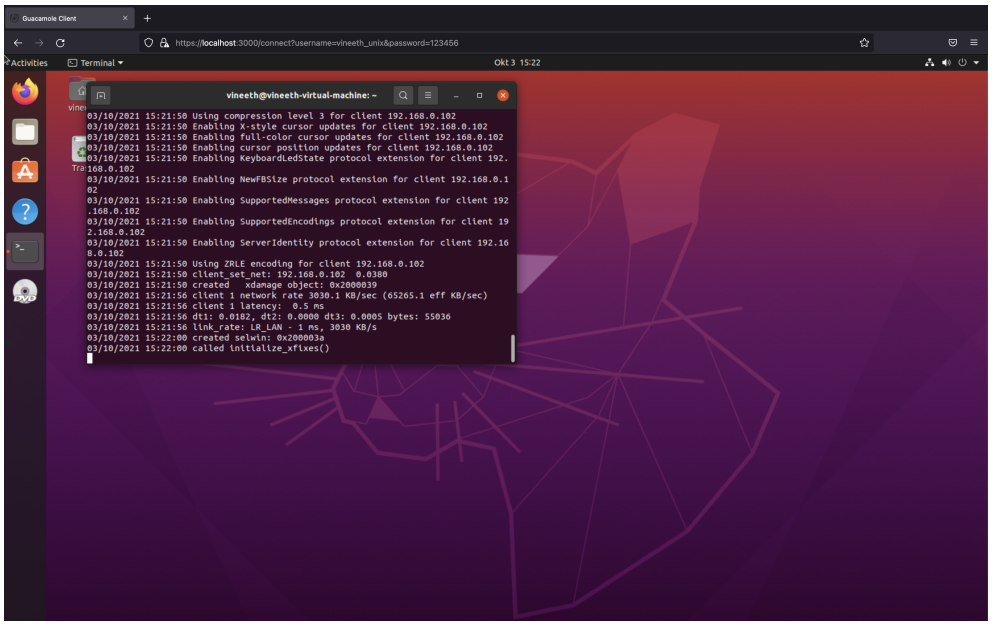


Figure 4.2: Ubuntu remote desktop

Once this is done, application is started and by logging into the application the remote desktop is displayed as shown in Figure 4.2. Here two accounts one with configurations for Ubuntu that uses VNC and another for Windows that uses RDP is used to connect to the remote systems. Figure 4.3 shows the Windows remote desktop that is displayed using the DaaS application and Figure 4.4 shows the Oneye application connected to DaaS.

Difference between VNC and RDP

Both VNC and RDP are used for remote access of a remote client, however, there are quite significant differences between the two technologies,

- VNC is available for most operating systems, whereas RDP is proprietary and supports Windows operating systems.
- VNC uses Remote Frame Buffer(RFB) protocol to provide remote access and connects itself to the VNC server. Whereas RDP logs into the remote user as if the user has logged into the remote client.
- RDP allows multiple users to connect to the same system isolating each user. In VNC the users connect to the same VNC server where they will see the same screen as the remote screen, mouse and keyboard are shared.
- RDP is semantic, that is it will transmit the information regarding the remote client to the user, this is used to compress the data, if there is a simple graphical output that needs to be sent to the client then instead of sending the image, the information such as location, size, color is sent this will significantly reduce the data size and increases

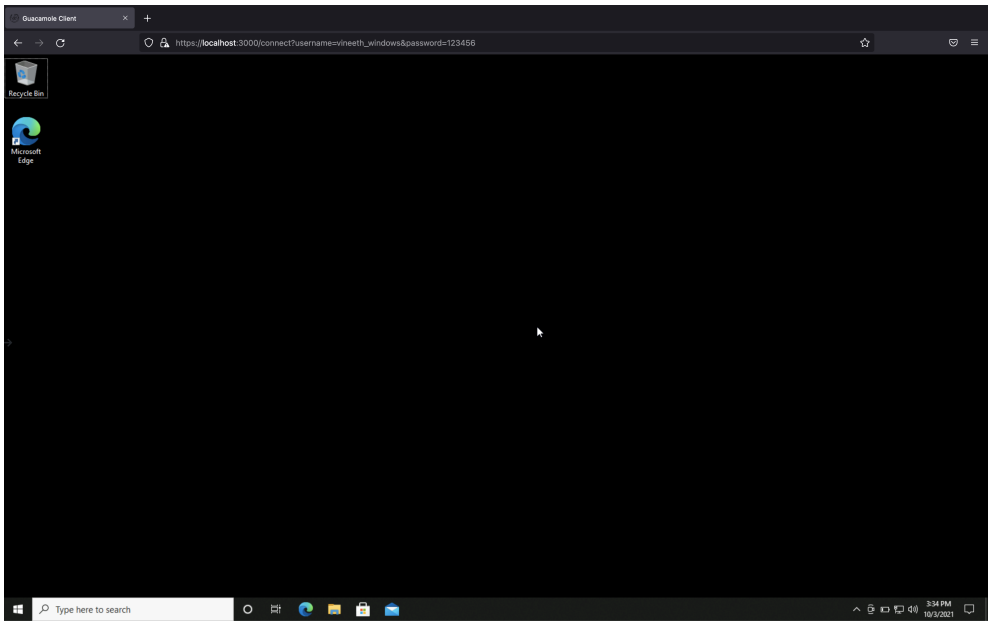


Figure 4.3: Windows remote desktop

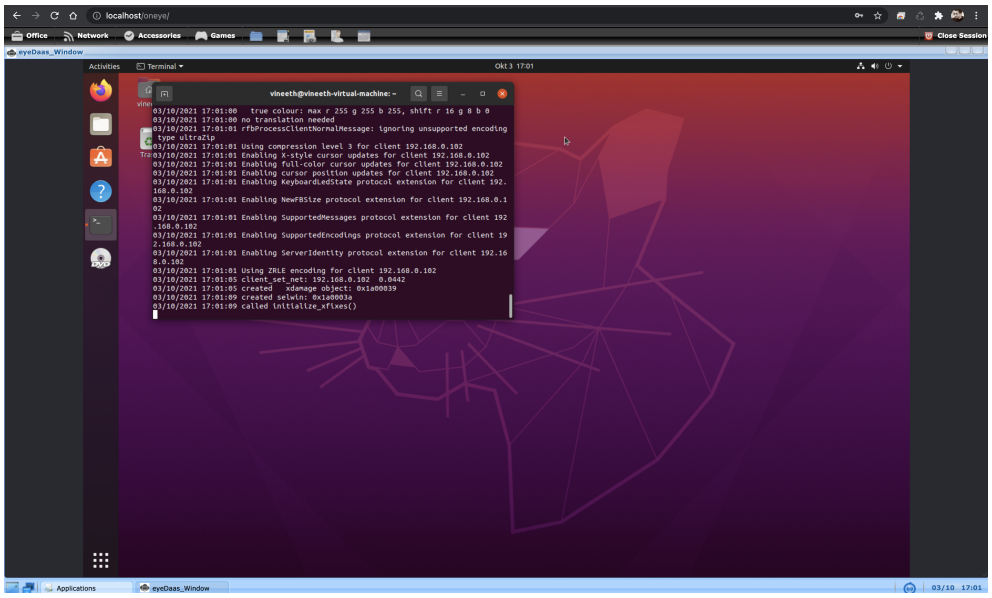


Figure 4.4: Remote Desktop in Oneye

the rate of transmission. Whereas, VNC cannot do this as it transmits the screen as images.

- VNC by defaults connects to port 5900, RDP sends TCP and UDP over port 3899.

These are few differences between RDP and VNC, RDP is better in terms of performance and higher graphical output than VNC, but VNC is much lighter compared to RDP.

4.2 DISCUSSION

The application was quite easy to set up and get the connection established with the remote desktop and its compatibility with other technologies and ability to provide support for different types of remote desktop protocols makes it powerful. Looking at the performance of the applications there were certain areas where the application lacked performance in general.

When testing the application it was found that there was a certain delay in getting back the response from the server. When checked further it was noted that the delay was initially from the client, in sending the request. The delay was around 5ms to 12ms. The request had to wait before being sent to the server. Due to this the mouse movement and the response was lagging and when playing a video this delay was increased. This was especially with the VNC connection with Ubuntu. The fact that a virtual machine was used to test the application also factored into this delay as the virtual machine itself was slow due to the limitation with the machine that was used for development and testing. When tested with RDP, the mouse movement was considerably better and the response was much quicker, however when opening a new window or playing a video through remote client, as it can be seen in the Figure 4.5 when too much load is put on the application the rendering becomes pixelated. Also, the mouse movement delays up a second in response time.

This delay was observed when using the front-end component that was developed using EJS. To compare and to figure out if this was specific to front-end the test was carried out with Angular, apart from rendering a much higher quality screen and smooth interface the response still was not better, but was better in terms of mouse movement and keyboard input and the initial connection was fast and overall experience was better. As the application was developed as a prototype some of the performance considerations that should be taken into account were ignored as a result the application was quite slow. Compared to the applications in the market like Kasm desktop as shown in Figure 4.6, which is quick in response, the application is quite a bit behind in performance.

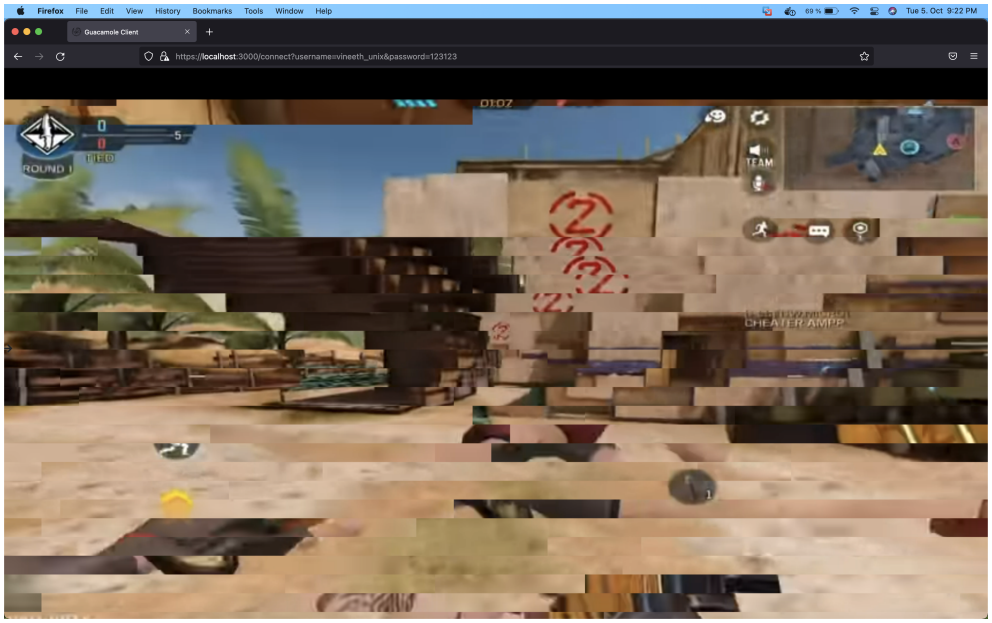


Figure 4.5: Pixelated rendering of remote desktop

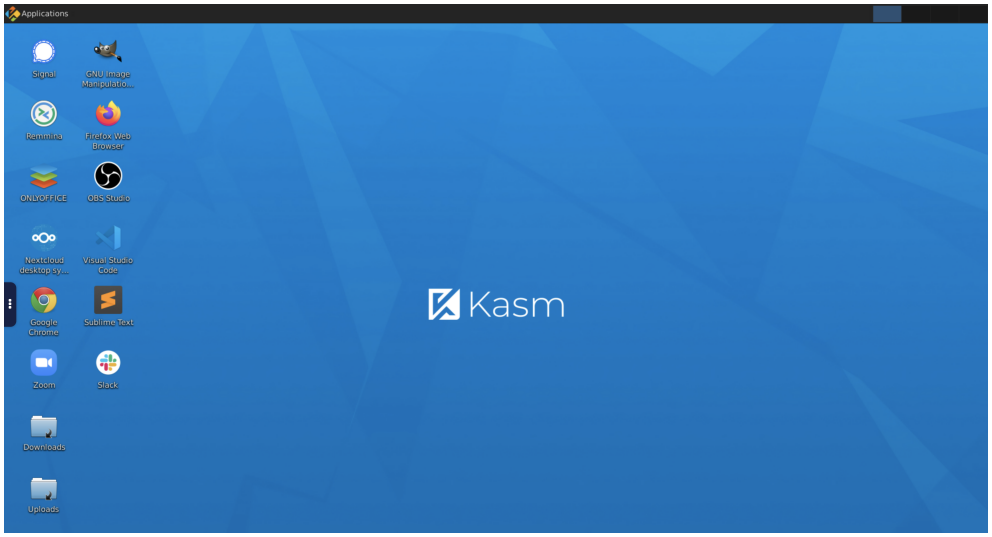


Figure 4.6: Kasm Remote Workspace[37]

Advantages

Considering developing the application with a much better UI that uses reactive js, and designing it to stream data continuously using React or the latest version of Angular would significantly improve the performance of the UI. Plus the server can be better developed to handle the lag in connection to avoid the standby time for the requests.

Some of the key take away from the application was,

- Guacd works seamlessly, as it provides options of different protocols and is convenient to switch from one protocol to another without having to implement it in the application.
- Guacamole protocol works well, as each action on the client-side is converted to instructions at the client end itself through guacamole-common-js as this reduces the load of converting each event to instructions at the server-side.
- guacamole-common-js provides an abstraction for event handlers so that it can be easily integrated with the front-end component.
- guacamole-lite is a good alternative as a web server that is easy to develop and integrate with the front-end and guacd.
- Apache guacamole in itself provides various security features, like LDAP, two-factor authentication and OAUTH. Utilization of these becomes easy by using the guacamole libraries.
- There is constant community support for Apache guacamole, hence its libraries are always updated.

4.3 PROPOSED SOLUTION FOR FILE TRANSFER

File transfer is an important feature that is required for any DaaS application. Apache Guacamole provides the support for file transfer through various technologies[51]. When using VNC connection which does not have any support for file transfer guacamole provides the file transfer feature using SSH file transfer protocol (SFTP). Similarly, when using RDP, it makes use of the native file transfer support provided by RDP called "drive redirection or RDPDR"[51], where the guacamole will create a virtual disk for the user to copy the file from or to the remote desktop. Using these protocols users can utilize the drag and drop feature provided by the guacamole. If any of the native protocols are not available for file transfer then it can make use of SFTP by enabling it in the configuration. Additionally, guacamole provides the option of file transfer through the command line using SSH and a utility developed to provide file transfer through the command line called "guacctl"[51]. This is a shell script[66], available along with guacamole, which enables the user to transfer the file through the command line.

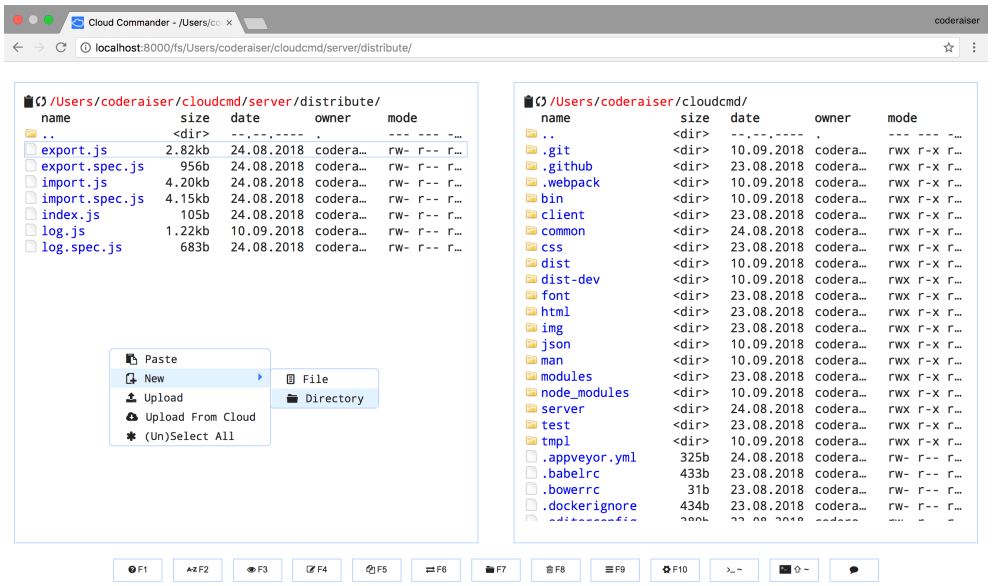


Figure 4.7: cloudcmd interface[10]

Cloud Commander

Another solution that was explored for file transfer was Cloud Commander[10] also called cloudcmd. It is an open-source node package that allows the user to transfer files through the browser. It is intended as a "file manager for the web"[10]. This can be set up with ease in the application server as the cloudcmd can operate as middleware by using the socket connection created as suggested in the cloudcmd npm registry [59]. Figure 4.7 shows the interface of cloud commander where the user can create a directory, copy or create a new file. This will serve as the ideal solution for file transfer with an interactive interface and easy set up and configuration.

4.4 ALTERNATE SOLUTION

Another alternate solution for the Desktop-as-a-Service was explored using web socket connections and transferring of remote desktop images, before exploring the current solution for Desktop-as-a-Service. The idea behind this solution is that the client connects to the remote desktop using web socket connection and takes a screenshot of remote desktop and sends it back to the client continuously at a short interval of time. This is ideally replicating the video streaming of remote desktop, a video is nothing but a continuous frame of images or similar to that of VNC protocol. The idea can be visualized using Figure 4.8. This is based on an open-source application found on GitHub[49].

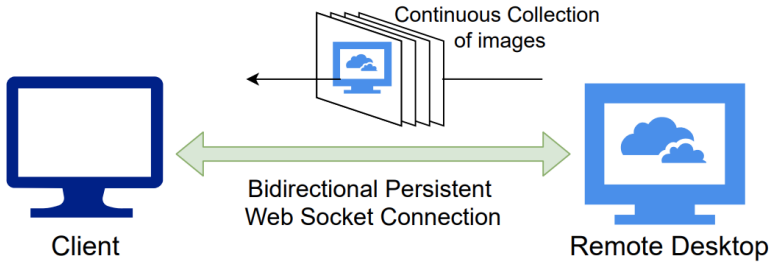


Figure 4.8: Alternate Solution for DaaS

Technologies that could be used in this proposed alternate solution and the reason behind them are as follows,

- Web Socket: The reason for web socket connection is because web socket connection is persistent and is bidirectional. Once the handshake is done the connection is kept open until one of the terminal ends disconnect it. This is ideal for continuous data streaming and receiving user inputs from the client. This is better when compared to HTTP, where the updates are served upon request only, also the request/response method of HTTP creates latency compared to the persistent web socket connection.
- Image Type: The screenshot image of the desktop can be sent over web socket using WebP or JPEG. However WebP would be preferred as it can provide images with lesser size compared to JPEG, but still, retain the quality which would suit the scenario to have a high-quality rendering of remote desktop.
- RobotJS[50]: RobotJS is a Node.js desktop automation library, mainly focused on integration with ElectronJS[26], but could be used in other Node applications. RobotJS provides functions to recognize mouse and keyboard events and send them to the remote desktop. For example, it provides a function for the keyboard as *keyTap()*, which recognizes the specific keypress by passing the keycode. Using this the mouse and the keyboard events can be sent to the remote desktop.
- Front-end: The front-end component can be developed using React or Angular.
- Server: Server can be developed with Node.js which has libraries for web socket connection creation and the RobotJS can be easily integrated.

These technologies give a good platform to get the prototype developed and depending on the requirement, changes might have to be adapted. It has to be noted that these technology suggestions have not been tried or tested with the architecture.

This solution was ignored as it simply outputs or video streams the remote desktop by taking screenshots of the remote desktop, unlike an RDP connection. Another main reason is there is not much support regarding RobotJS and due to limited usage it might fall out of favor in the near future and the application will need to implement a different solution for handling user events. Even with these minor drawbacks, this alternative could still prove to be an effective solution. By reducing the interval of taking screenshots to several milliseconds the delay can be reduced, but this would put a load on the connection the server should be able to handle large amounts of data in the form of an image.

5 FUTURE WORK AND CONCLUSION

5.1 FUTURE WORK

The DaaS application developed as a prototype has several areas where the application could improve and several areas where further development is required to make it a more efficient and robust application. These are initial steps that must be taken in the future in improving the application,

- To begin with a database has to be integrated into the application, as explained in 3.5.1.1, the temporary solution is just for testing the application and not for a production application. For this purpose, a robust and efficient database must be integrated to save the user details along with connection details.
- Once the database is integrated, the data stored must be encrypted by using a hashing algorithm like SHA256.
- Further, authentication services should be provided to improve the security of the application by not only utilizing the database for authentication, additional security like two-factor authentication must be provided.
- If Oneye is used as web server, then the application must be able to connect to SSL/TLS connections, that is the issue explained in 3.4.4 must be resolved.
- Issue with docker multi-container setup as explained in 3.5.2 must be resolved so that the containers can easily communicate with one another.
- The latency in rendering the remote desktop when there is a load or video is played must be resolved.
- End-to-end encryption can be provided by enabling the SSL/TLS in the guacd so that the connection to guacd is also encrypted.
- A DaaS application should allow the user to transfer files to the remote desktop with ease or download the file from remote desktop. This feature is missing in the developed application and can be implemented using cloudcmd as explained in 4.3.

Since the application was developed as a prototype and for academic purposes, it lacks the capability to be used in a large-scale environment where the application is used by multiple users and is connected to multiple remote desktops. So the architecture of the application must be refined to the requirements.

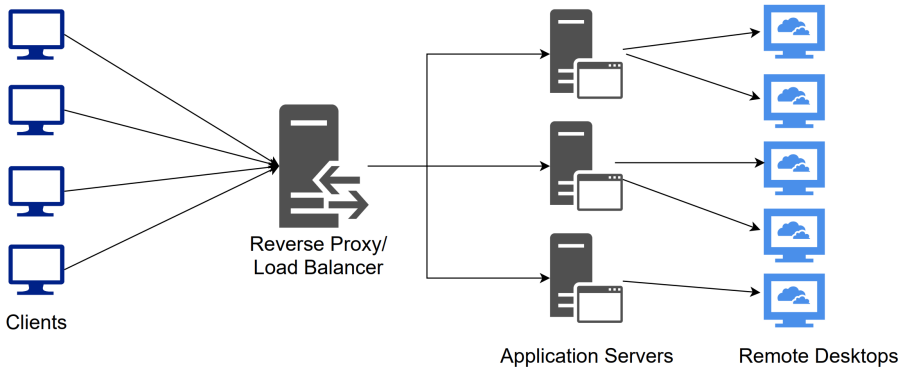


Figure 5.1: Architecture with Reverse Proxy / Load Balancer

5.1.1 *Architecture*

Figure 5.1 shows the architecture that uses a reverse proxy server or load balancer in between the client and the application server. This is because, as the requests increase the single application server cannot manage all the requests and the load balancer will distribute the load such that no application server is burdened with requests and avoids a single point of failure.

Reverse proxy and load balancers perform similar tasks but they are not the same. Load balancers are commonly deployed between the client and the application server to distribute the incoming requests. They also enhance the user experience by detecting the error responses from the server and redirecting the requests to healthy servers. Along with this, load balancers help keep a persistent session, by sending the requests that come from a particular client to a single server. Load balancers make only sense when there are multiple application servers deployed [55].

Reverse proxy on the other hand works with a single application server as well. It provides additional security as the information of application servers is not directly visible to the public and distributed denial-of-service (DDoS) attacks on the application server. It increases flexibility and scalability as the application server can be scaled up or down without impacting the incoming traffic. Bandwidth amount can be reduced with reverse proxy, by compressing server responses before sending them to the client. Several other operations like encryption and decryption of responses and requests can be moved to reverse proxy so that the application server can focus on business operations and reduce the load on them. The reverse proxy also allows caching, so that in case of a temporary error response from the server or if the same data is requested by the client then instead of requesting the application server, cached data can be served to reduce the response time [55].

5.2 CONCLUSION

Desktop-as-a-Service is a great application, with great potential for being an excellent remote workspace solution. The majority of the companies have started to offer DaaS solutions which show the importance and the need for such solutions in the modern world. This is because of the overhead of maintaining the infrastructure, handling confidential data, keeping up with the constant change in technologies it seems better to have a solution where a single entity takes care of all these problems and provides easy, user-friendly solutions to their customers through DaaS. There are numerous solutions in the market as named in the introduction 1.2. Exploring these solutions, it was noted that the applications tend to gain more popularity based on a few core set of factors such as the ability of the application to run Windows/Linux based apps, mainly softwares like word document, spreadsheet capabilities and email client or the application should be able to connect to an actual remote desktop. Then the user experience itself is it easier to use, set up and configure, the application should be fast. Then one of the main features is security. The commercial applications provide all these however they are expensive and focus on large-scale industry and the open-source applications lack one or the other feature that makes them fall behind in delivering the needed service.

The application developed using the Apache Guacamole and Oneye provides a reasonably good solution. As the Oneye is outdated and does lack the ability to connect to an actual desktop, the application developed for DaaS fills this void. Given the application is not up to standards as compared to Amazon workspaces or Citrix VDI, which have been through several iterations of development and have some of the advanced proprietary technologies. The concept of DaaS handles the remote connection well, in terms of connection, user events like mouse and keyboard and the performance output was acceptable. Also includes the capability to use any protocol for remote connection. Added to this the constant support from the community to guacamole will only contribute to its growth in the right direction. To conclude, the application developed using open-source technologies do lack the performance and feature to compete with the commercial products but with the drawbacks addressed in the right manner, and improve the applications in key areas the application will be an excellent alternative for open-source DaaS applications.

Part II

APPENDIX

A

SOURCE CODE

The source code for this application can be found in the following github repository[67]. This includes all the docker configurations and guacamole-lite and Java web servers, along with EJS and Angular front-end components.

BIBLIOGRAPHY

[1] *About Node.js*. URL: https://nodejs.org/en/about/. Accessed: 20.09.2021.

[2] *Amazon WorkSpaces*. URL: https://aws.amazon.com/workspaces/?workspaces-blogs.sort-by=item.additionalFields.createdDate&workspaces-blogs.sort-order=desc. (accessed: 16.09.2021).

[3] *Angular*. URL: https://angular.io/guide/what-is-angular. Accessed: 20.09.2021.

[4] *Apache Guacamole*. URL: https://guacamole.apache.org/. Accessed: 18.09.2021.

[5] *Apache*. URL: https://httpd.apache.org/ABOUT_APACHE.html. Accessed: 20.09.2021.

[6] *Azure Virtual Desktop*. URL: https://azure.microsoft.com/en-us/services/virtual-desktop/. Accessed: 16.09.2021.

[7] Antonio Celesti, Davide Mulfari, Maria Fazio, Massimo Villari, and Antonio Puliafito. "Improving desktop as a Service in OpenStack." In: *2016 IEEE Symposium on Computers and Communication (ISCC)*. 2016, pp. 281–288. DOI: 10.1109/ISCC.2016.7543755.

[8] Deka Ganesh Chandra and Dutta Borah Malaya. "A Study on Cloud OS." In: *2012 International Conference on Communication Systems and Network Technologies*. 2012, pp. 692–697. DOI: 10.1109/CSNT.2012.154.

[9] *Citrix VDI and DaaS*. URL: https://www.citrix.com/solutions/vdi-and-daas/s. Accessed: 16.09.2021.

[10] *Cloud Commander*. URL: https://cloudcmd.io/. Accessed: 12.10.2021.

[11] *Configuring Guacamole*. URL: https://guacamole.apache.org/doc/0.9.3/gug/configuring-guacamole.html. Accessed: 20.09.2021.

[12] *Configuring SSL in guacd within Docker*. URL: https://stackoverflow.com/questions/63778531/enable-ssl-between-guacd-and-guacamole-web-server-tomcat. Accessed: 20.09.2021.

[13] *Crypto*. URL: https://www.npmjs.com/package/crypto. Accessed: 20.09.2021.

[14] *Crypto.js*. URL: https://www.npmjs.com/package/crypto-js. Accessed: 20.09.2021.

[15] *Docker 0.9: introducing execution drivers and libcontainer*. URL: https://www.docker.com/blog/docker-0-9-introducing-execution-drivers-and-libcontainer/. Accessed: 20.09.2021.

[16] *Docker Architecture*. URL: https://docs.docker.com/get-started/overview/. Accessed: 20.09.2021.

[17] *Docker build*. URL: https://docs.docker.com/engine/reference/commandline/build/. Accessed: 20.09.2021.

[18] *Docker php*. URL: https://hub.docker.com/_/php. Accessed: 20.09.2021.

[19] *Docker run*. URL: https://docs.docker.com/engine/reference/commandline/run/. Accessed: 20.09.2021.

[20] *Docker run*. URL: https://docs.docker.com/compose/. Accessed: 20.09.2021.

[21] *Docker*. URL: https://www.docker.com/. Accessed: 20.09.2021.

[22] *Docker*. URL: https://hub.docker.com/. Accessed: 20.09.2021.

[23] *Dockerfile reference*. URL: https://docs.docker.com/engine/reference/builder/. Accessed: 20.09.2021.

[24] *EJS*. URL: httpshttps://ejs.co/. Accessed: 20.09.2021.

[25] *Easy Node Authentication*. URL: https://github.com/scotch-io/easy-node-authentication. Accessed: 24.09.2021.

[26] *ElectronJS*. URL: https://www.electronjs.org/. Accessed: 04.10.2021.

[27] *GClient From Linuxserver*. URL: https://github.com/linuxserver/gclient. Accessed: 20.09.2021.

[28] *Glyptodon Enterprise - End User License Agreement*. URL: https://glyp.to/eula/. Accessed: 20.09.2021.

[29] *Glyptodon Enterprise*. URL: https://glyp.to/. Accessed: 20.09.2021.

[30] *Guacamole Manual: Chapter 1. Implementation and architecture*. URL: https://guacamole.apache.org/doc/gug/guacamole-architecture.html. Accessed: 18.09.2021.

[31] *Guacamole Protocol*. URL: http://guacamole.apache.org/doc/gug/guacamole-protocol.html. Accessed: 20.09.2021.

[32] *Guacamole Server*. URL: wgethttps://downloads.apache.org/guacamole/1.3.0/source/guacamole-server-1.3.0.tar.gz-P~. Accessed: 20.09.2021.

[33] Jamie Hall. *guacamole-test-server*. URL: https://github.com/jamhall/guacamole-test-server. Accessed: 21.09.2021.

[34] *Horizon: The OpenStack Dashboard Project*. URL: https://docs.openstack.org/horizon/latest/. Accessed: 16.09.2021.

[35] *How to create an HTTPS certificate for localhost domains*. URL: https://gist.github.com/cecilemuller/9492b848eb8fe46d462abeb26656c4f8. Accessed: 30.09.2021.

[36] Vineeth Bhat Jathin Sreenivas Vidya Gopalakrishnarao. *Deployment of a Private/Hybrid Cloud IaaS OpenStack*. URL: https://www.christianbaun.de/CGC2021/Skript/Team_4_OpenStack_WS2021.pdf. Accessed: 16.09.2021.

[37] *Kasm Remote Workspaces*. URL: https://www.kasmweb.com/vdi.html#. Accessed: 17.09.2021.

[38] *Kasm Workspace Images*. URL: https://www.kasmweb.com/workspace_images.html. Accessed: 18.09.2021.

[39] *KasmVNC*. URL: https://www.kasmweb.com/kasmvnc.html. Accessed: 17.09.2021.

[40] Institut Laue-Langevin. *ngx-remote-desktop - installation doc*. URL: https://github.com/ILLGrenoble/ngx-remote-desktop/tree/master/additional-doc. Accessed: 24.09.2021.

[41] Institut Laue-Langevin. *ngx-remote-desktop - npm*. URL: https://www.npmjs.com/package/ngx-remote-desktop. Accessed: 24.09.2021.

[42] Institut Laue-Langevin. *ngx-remote-desktop*. URL: https://github.com/ILLGrenoble/ngx-remote-desktop. Accessed: 20.09.2021.

[43] *Node.js*. URL: https://nodejs.org/en/about/. Accessed: 20.09.2021.

[44] *Oneye 0.9.6 preview*. URL: https://github.com/oneye/oneye/releases/tag/v0.9.6-preview. Accessed: 25.09.2021.

[45] *Oneye Github*. URL: https://github.com/oneye/oneye. Accessed: 04.10.2021.

[46] *Oneye Project*. URL: https://oneye-project.org/. Accessed: 18.09.2021.

[47] *OpenSSL*. URL: https://www.openssl.org/. Accessed: 20.09.2021.

[48] *Openstack*. URL: https://www.openstack.org/. Accessed: 16.09.2021.

[49] *Remote desktop*. URL: https://github.com/kousik19/remote-desktop. Accessed: 06.10.2021.

[50] *RobotJS*. URL: http://robotjs.io/. Accessed: 04.10.2021.

[51] *Transferring files*. URL: https://guacamole.apache.org/doc/gug/using-guacamole.html#file-browser. Accessed: 12.10.2021.

[52] *Use containers to Build, Share and Run your applications*. URL: https://www.docker.com/resources/what-container. Accessed: 20.09.2021.

[53] *Using OpenNebula for Handling WFH Demands for VDI at the Edge*. URL: https://opennebula.io/opennebula-for-vdi-at-the-edge/. Accessed: 18.09.2021.

[54] *Web Desktop Environment*. URL: https://github.com/shmuelhizmi/web-desktop-environment. Accessed: 17.09.2021.

[55] *What is a Reverse Proxy vs. Load Balancer?* URL: https://www.nginx.com/resources/glossary/reverse-proxy-vs-load-balancer/. Accessed: 04.10.2021.

[56] *Writing your own Guacamole application*. URL: http://guacamole.apache.org/doc/gug/writing-you-own-guacamole-app.html. Accessed: 21.09.2021.

[57] *XAMPP installer*. URL: https://sourceforge.net/projects/xampp/files/XAMPP%20Mac%20OS%20X/5.6.40/. Accessed: 25.09.2021.

[58] *XAMPP*. URL: https://sourceforge.net/projects/xampp/. Accessed: 25.09.2021.

[59] *cloudcmd npm registry*. URL: https://www.npmjs.com/package/cloudcmd. Accessed: 12.10.2021.

[60] *eyeOS Developer Manual*. URL: https://oneye-project.org/wp-content/uploads/2011/07/developer-manual.pdf. Accessed: 24.09.2021.

[61] *glyptodon/guacd*. URL: https://hub.docker.com/r/glyptodon/guacd. Accessed: 20.09.2021.

[62] *guacamole-common-js*. URL: http://guacamole.apache.org/doc/gug/guacamole-common-js.html. Accessed: 22.09.2021.

[63] *guacamole-common*. URL: https://guacamole.apache.org/doc/gug/guacamole-common.html. Accessed: 18.09.2021.

[64] *guacamole-lite*. URL: https://www.npmjs.com/package/guacamole-lite. Accessed: 18.09.2021.

[65] *guacamole/guacd*. URL: https://hub.docker.com/r/guacamole/guacd. Accessed: 20.09.2021.

[66] *guacctl*. URL: https://raw.githubusercontent.com/apache/guacamole-server/master/bin/guacctl. Accessed: 12.10.2021.

[67] *master-thesis: source code*. URL: https://github.com/bhatvineeth/master-thesis. Accessed: 06.10.2021.

[68] *vmware Fusion*. URL: https://www.vmware.com/products/fusion.html. Accessed: 20.09.2021.

[69] *x11 keysyms*. URL: https://www.cl.cam.ac.uk/~mgk25/ucs/keysymdef.h. Accessed: 23.09.2021.