INTR 8015 Project Implementation Report
Building a Hosting Service for Websites

David Monaghan
B.Sc.(Hons.) in IT Management
Department of Computing

Student ID: R00103763
Supervisor: Meabh O'Connor
Second Reader: Noreen Gubbins
Date: 5th May 2016

1 – Document Overview

1.1 – Revision History

Table 1: Revision History

Version	Author	Supervisor	Second Reader	Date
V1.0	David Monaghan	Meabh O'Conner	Noreen Gubbins	05/05/2016
V2.0 (Final)	David Monaghan	Meabh O'Connor	Noreen Gubbins	12/05/2016

1.2 – Declaration of Original Work

I hereby certify that this material which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent, that such work has been cited and acknowledged within the text of my work. I understand that my project documentation may be stored in the library at CIT, published on Blackboard and may be referenced by others in the future.

Date: _____ **Signature** _____

Table of Contents

2 – Executive Summary

The primary goal of this project is to create a service that will allow a customer to create, quickly and easily, a website without having to worry about any of the underlying technical issues involved in such an undertaking such as installing or configuring the servers required. The service will be designed with scalability and security in mind. The rationale for this has been laid out in the research document. This document concerns itself with the implementation of the conclusions reached during the research phase.

The implementation report begins with an introduction explaining the purpose, goals and desired outcomes of the project. The author also describes himself and acknowledges those who have been helpful throughout the implementation.

The next chapter outlines the final design of the project implementation. Text in this section is kept to a minimum and instead the design is illustrated through a series of activity and network diagrams. There are 31 in total.

The project implementation chapter concerns itself with the major design decisions made during the project, the rationale for their inclusion, the reason for their inclusion in the project as opposed to another technology. The chapter also outlines samples of how these technologies where used during implementation.

The penultimate chapter deals with an overall evaluation of the projects requirements and whether these requirements have been achieved. Out of a total of 48 requirements only 2 where not satisfied. This chapter also shows the testing done on the implementation and shows the robustness of the system under the most obvious of stresses, such as a back-end server going down.

The final chapter concludes with some thoughts on lessons learned during the project and of some immediate improvements that can be brought to the current implementation during the next round of development.

The appendices include a bibliography of sources used during the implementation phase. The final appendix provides the code used during the implementation. The code is for reference purposes and can also be found at https://github.com/davidmonaghan/4thyearproject.

3 – Introduction

3.1 – About The Project

Setting up, configuring and maintaining a server(s) is a tedious and painstaking task. A single mistake and one will inevitably have to start from scratch. The project's goal is to take away some of that tedium by automating many of these initial tasks and allowing the client to get to the important part of building a website, namely designing the site and creating its content.
The author set himself the task of creating a service that can:

1. Create an Ubuntu 14.04 virtual machine
2. Install and configure an Apache web server
3. Install and configure a Wordpress CMS
4. Install and configure a MySQL database for Wordpress site
5. A web interface to initiate and control the virtual machine
6. Finally, to complete these tasks as quickly as possible

This service should, once this project is completed, be easy to continue expanding if there is another development cycle, where ultimately the goal in mind is to create the service in full. The author aims to use best industry practices to ensure that another individual could take the project forward from where the project finishes.

3.2 – About The Author

The author is a mature student having returned to College after many years in the workforce and looking for a new challenge. While the author has enjoyed his time in CIT, he is very much looking forward to finishing studying and moving onto the next challenge.

3.3 – Acknowledgements

The author of this report would like to acknowledge the support provided by CIT in producing this report and through the implementation phase, particularly the project supervisor Meabh O'Connor and the second reader Noreen Gubbins. The author would also like thank the developers of LXC and the community that is growing around them for their help with some of the technical issues during the project. Finally the author would like to thank the judging panel for warding the author the prize of best final year project in the IT Management category, an honour for which the author is quite proud.

4 – Project Design

4.1 – Introduction

The following section outlines the final design for the implementation stage of the project. Included are a series of activity diagrams describing the functionality of the system. The project design overall does not stray too far from that outlined during the research phase. The exceptions to this are:

1. A service devoted to managing the MySQL database
2. The use of JSON data-objects for encapsulation of messages between components of the service
3. The use of Web-Sockets for communication between the customers web-browser and the Proxy-Control-Server
4. The use of Network-Sockets for communication between back end components of the service
5. Instead of using PHP for the Web-Front-End, the project used Javascript
6. There has also been a change to naming of services
    I. Master Container Management Daemon is now called Proxy-Control-Server
    II. Slave Container Management Daemon is now called LXC-Control-Server
7. Further to the change in naming convention
    I. The server managing the MySQL database is called the Database-Control-Server

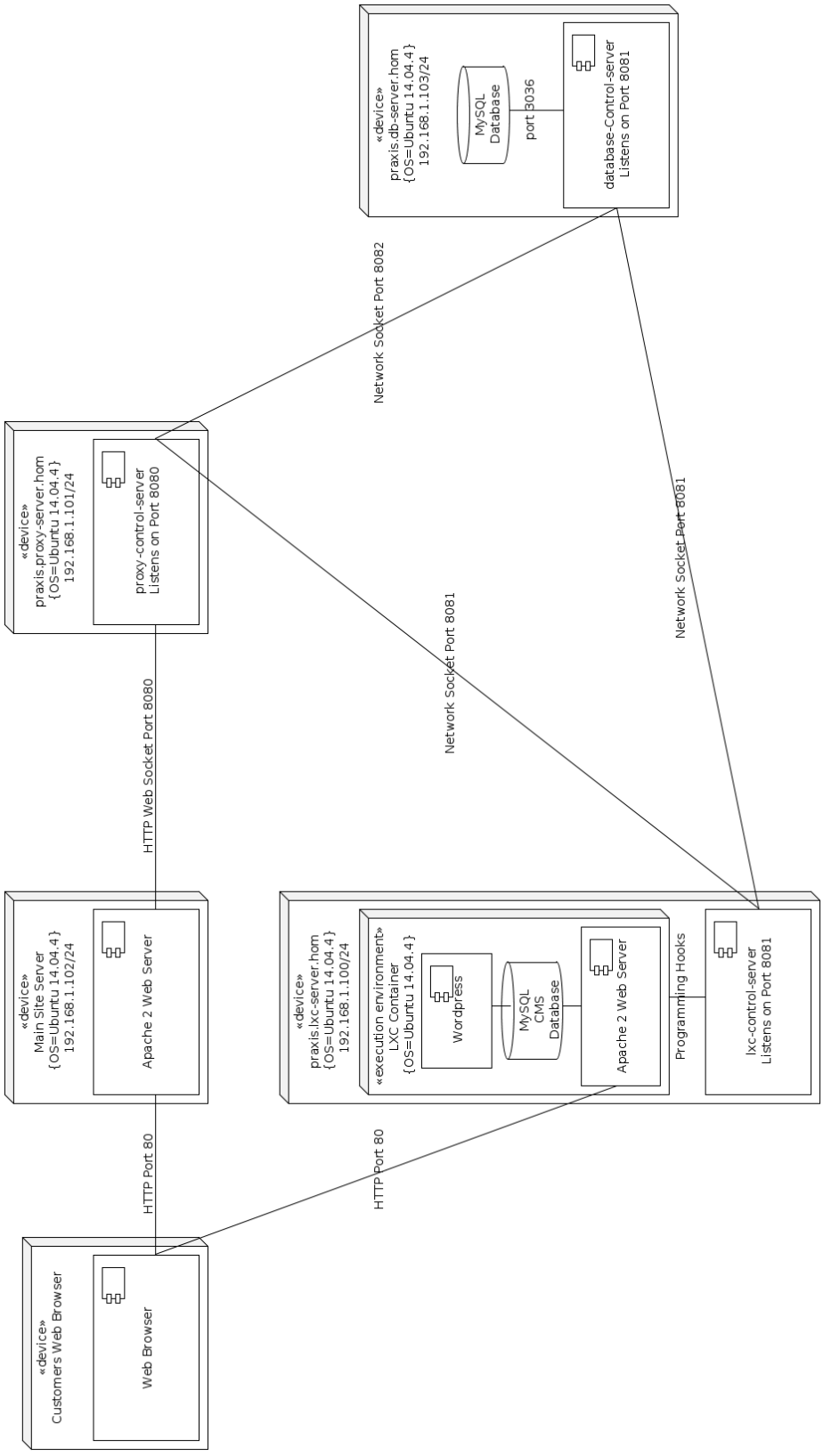
Full details for rationale of these decisions can be found in the project implementation chapter

The activity diagrams, use the following to denote particular events:

- Each individual service is denoted by a box
- A concurrent routine is represented by a green oval
- An ordinary routine is denoted by a red oval
- A cyan hexagon denotes a decision being made
- A yellow rectangle denotes a signal being sent or received by a server
- A solid black circle denotes the beginning of a service
- A dotted circle denotes the end of a service

4.2 – Network Architecture

Figure 1: Implementation Network Diagram



4.3 – Web-Front-End

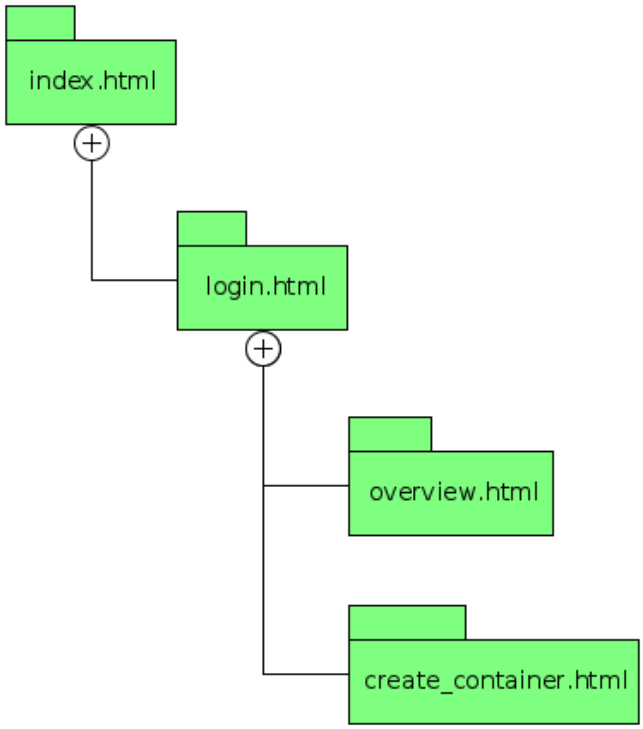
4.3.1 – Introduction

A web front-end is required in order to create and manage containers. As per the project requirements, the Web-Front-End is not of primary concern but instead is intended to primarily show the functionality by the back-end of the service such as creating and managing the customers containers and websites.

In order to interact with the back-end services the front-end requires the ability to set-up and maintain a Web-Socket with the Proxy-Control-Server. The Proxy-Control-Server will then manage the rest of the connection. Once our service has finished completing the customers request it will return either a message advising completion of the request or an error message advising failure to do so. A JSON data-object will be used to provide both the customers request and relay a messages back from the service back-end. As such the Web-Front-End will require the ability to both encode and decode JSON data objects to the/ from the Web-Socket connection with Proxy-Control-Server.

The web-site will have a basic structure, outlined in Figure 2. The index page uses place-holder text and a simple design but has no functionality. Clicking the login link will bring one to the login page where there will be a choice of three users, no password is required. Clicking on one the users will bring a person to the overview page, which provides details of current containers for that user plus all management function currently available. The overview page provides a link to the create_container page where one can create a new container instance. The logout link will bring one back to the index page.

Figure 2: Web-Site File Tree



4.3.2 – Overview.html

The overview page provides the user with the ability to see which containers exist for a user or if logged in as an administrator, all containers for all users. The overview page also contains commands for the containers

To provide this functionality and to send customer commands to the back-end, the overview page must create a Web-Socket connection with the Proxy-Control-Server, illustrated in figure 5. Communication between the front-end and the back-end of the service is provided using a JSON data object. The JSON data, called NewContainerJSON, will maintain the same structure in all parts of the service. The JSON data objects structure is outlined in figures 3.

Once a Web-Socket has been first established with Proxy-Control-Server on page loading it will send a ***'getListOfContainers'*** command, which returns a list of containers from the Database-Control-Server. With the list of containers returned the customer will then have a choice of commands to perform on a container, outlined above. This command is sent as a JSON data-object to the Proxy-Control-Server on the Web-Socket previously established. The back-end service will respond with the status of the request or with an error message.

Further to the list of containers, the overview page can send a number of commands as a JSON to the Proxy-Control-Server. From here the current status of the container will be confirmed as, e.g. '***Stopped***' with the Database-Control-Server before the command is sent to the LXC-Control-Server. Once the LXC-Control-Server has e.g. '***Stopped***' the container, it's status will be updated with Database-Control-Server before a message is returned to the Web-Front-End that contains it's current status and the containers IP address. The full functionality for this command is illustrated in figures 6 – 12.

the overview page allows the user to manage their containers with following commands:

- **Start**
  - Starts a container. This command will only function if a container is already stopped.
- **Stop**
  - Stops a container. This command will only function on already running containers.
- **Restart**
  - Restarts a container. This command will only function on already running containers.
- **Delete**
  - Deletes a container. This command will only function if a container is already stopped.
- **Back-Up**
  - Creates a full copy of the original container. This command will only function if a container is already stopped.
- **Snapshot**
  - Creates a snapshot of the containers current state. This command will only function if a container is already stopped.

Figure 3: JSON data-object

```
CustomerType string             `json:"CustomerType,omitempty"`
CustomerUname string            `json:"CustomerUname,omitempty"`
Action string                   `json:"Action,omitempty"`
ContainerName string            `json:"ContainerName,omitempty"`
ContainerStatus string          `json:"ContainerStatus,omitempty"`
ContainerIPaddress string       `json:"ContainerIPaddress,omitempty"`
WordpressStatus string          `json:"WordpressStatus,omitempty"`
```

- `CustomerType string` `` `json:"CustomerType,omitempty"` ``
  - CustomerType: Type of user: *'user'* or *'admin'*.
  - Data-type is ***'string.'***
- `CustomerUname string` `` `json:"CustomerUname,omitempty"` ``
  - *CustomerUname:* Customers user-name.
  - Data-type is ***'string.'***
- `Action string` `` `json:"Action,omitempty"` ``
  - *Action:* Command to be performed on the container: *'getListOfContainers'*, *'conatinerStart'*, *'containerStop'*, *'containerRestart'*, *'containerDelete'*, *'containerBackup'*, *'containerSnapshot'*.
  - Data-type is ***'string.'***
- `ContainerName string` `` `json:"ContainerName,omitempty"` ``
  - *ContainerName:* Name of container upon which the command is to be performed or for whom the back-end is sending a message
  - Data-type is ***'string.'***
- `ContainerStatus string` `` `json:"ContainerStatus,omitempty"` ``
  - *ContainerStatus:* This value blank when sent by overview.html but the back-end service will return the status of the container in response or an error message.
  - Data-type is ***'string.'***

- `ContainerIPaddress string` `` `json:"ContainerIPaddress,omitempty"` ``
  - *ContainerIPaddress:* This value blank when sent by overview.html but the back-end service will return the IP address if the containers status has been changed to *'Running'*.
  - Data-type is ***'string.'***

4.3.3 – Create_Container.html

The Create_Container page provides the user with the ability to create a new container. This function was conceived as providing the customer with a number of choices regarding choice of base server, web-server, DBMS (Database Management System) and CMS (Content Management System).

Similar to the overview page, the Create_Container page must create a Web-Socket connection with the Proxy-Control-Server, illustrated in figure 13. Communication between the front-end and the back-end of the service is provided using a JSON data object called NewContainerJSON, will maintain the same structure in all parts of the service. The JSON data objects structure is outlined in figure 4.

Once a Web-Socket has been first established with Proxy-Control-Server on page loading and the customer has submitted the form with their configuration details, a command call '*createNew*' will be sent as JSON to Proxy-Control-Server on the Web Socket previously established. The Proxy-Control-Server will send the JSON to the Database control server which will add a new entry for the new container once it confirms that a container of that name does not exist for the customer sending the request. The full sequence is outlined in figure 13.

Figure 4: JSON data-object

```
CustomerType string             `json:"CustomerType,omitempty"`
CustomerUname string            `json:"CustomerUname,omitempty"`
Action string                   `json:"Action,omitempty"`
ContainerName string            `json:"ContainerName,omitempty"`
BaseServer string               `json:"BaseServer,omitempty"`
CMS string                      `json:"CMS,omitempty"`
WebsiteName string              `json:"WebsiteName,omitempty"`
DBrootPWD string                `json:"DBrootPWD,omitempty"`
DBadminUname string             `json:"DBadminUname,omitempty"`
DBadminPWD string               `json:"DBadminPWD,omitempty"`
ContainerStatus string          `json:"ContainerStatus,omitempty"`
ContainerIPaddress string       `json:"ContainerIPaddress,omitempty"`
WordpressStatus string          `json:"WordpressStatus,omitempty"`
```

- `CustomerType string` `json:"CustomerType,omitempty"`
  - CustomerType: Type of user: *'user'* or *'admin'*.
  - Data-type is ***'string.'***
- `CustomerUname string` `json:"CustomerUname,omitempty"`
  - *CustomerUname:* Customers user-name.
  - Data-type is ***'string.'***
- `Action string` `json:"Action,omitempty"`
  - *Action:* Command to be performed on the container: *'getListOfContainers'*, *'conatinerStart'*, *'containerStop'*, *'containerRestart'*, *'containerDelete'*, *'containerBackup'*, *'containerSnapshot'*.
  - Data-type is ***'string.'***
- `ContainerName string` `json:"ContainerName,omitempty"`
  - *ContainerName:* Name of container upon which the command is to be performed or for whom the back-end is sending a message
  - Data-type is ***'string.'***
- `BaseServer string` `json:"ContainerName,omitempty"`
  - *BaseServer:* This is the server that makes up the containers operating system, by the default during the implementation only Ubuntu 14.04 will be available.
  - Data-type is ***'string.'***
- `CMS string` `json:"CMS,omitempty"`
  - *CMS:* Content Management System for the management of the system.
  - Data-type is ***'string.'***
- `WebsiteName string` `json:"WebsiteName, omitempty"`
  - *CMS:* Used for configuration of the CMS database and CMS configuration.
  - Data-type is ***'string.'***
- `DBrootPwd string` `json:"DbrootPwd, omitempty"`
  - *DBrootPwd:* Used to change the default Root password of the CMS MySQL database.
  - Data-type is ***'string.'***

- `DBadminUname string` `` `json:"DBadminUname,omitempty"` ``
  - *DBadminUname:* This used to create a user that Wordpress installation can use to modify the web site.
  - Data-type is ***'string.'***
- `DBadminPWD string` `` `json:"DBadminPWD,omitempty"` ``
  - *DBadminPWD:* Password for the Wordpress administrator user.
  - Data-type is ***'string.'***
- `ContainerIPaddress string` `` `json:"ContainerIPaddress,omitempty"` ``
  - *ContainerIPaddress:* This value blank when sent by overview.html but the back-end service will return the IP address if the containers status has been changed to *'Running'*.
  - Data-type is ***'string.'***

Note: The Datbase-Control-Server and LXC-Control-Server sections in the following activity diagrams have been simplified for the sake of readability. A more detailed view of both servers follows in the next sub-chapter.

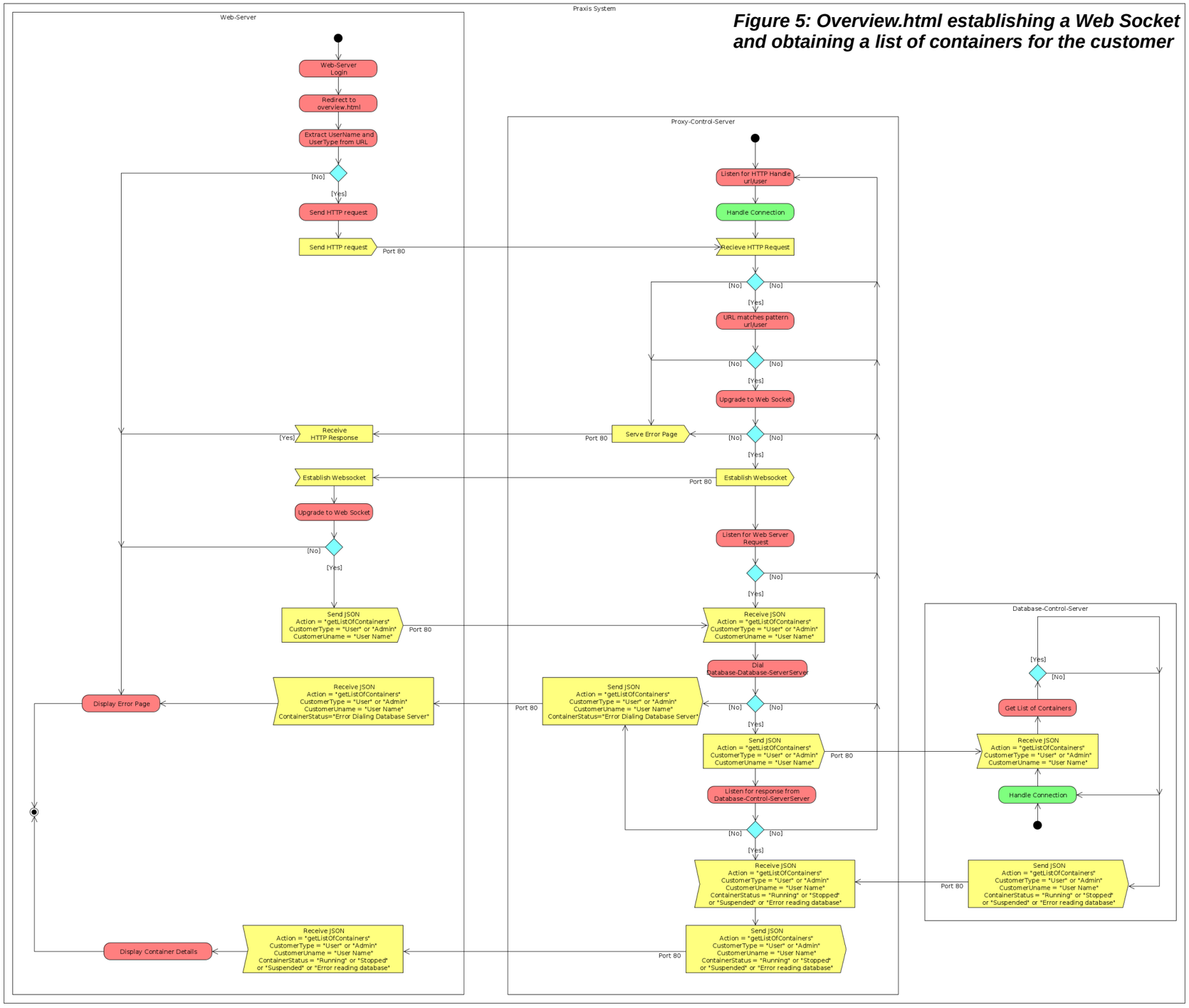


Figure 5: Overview.html establishing a Web Socket and obtaining a list of containers for the customer

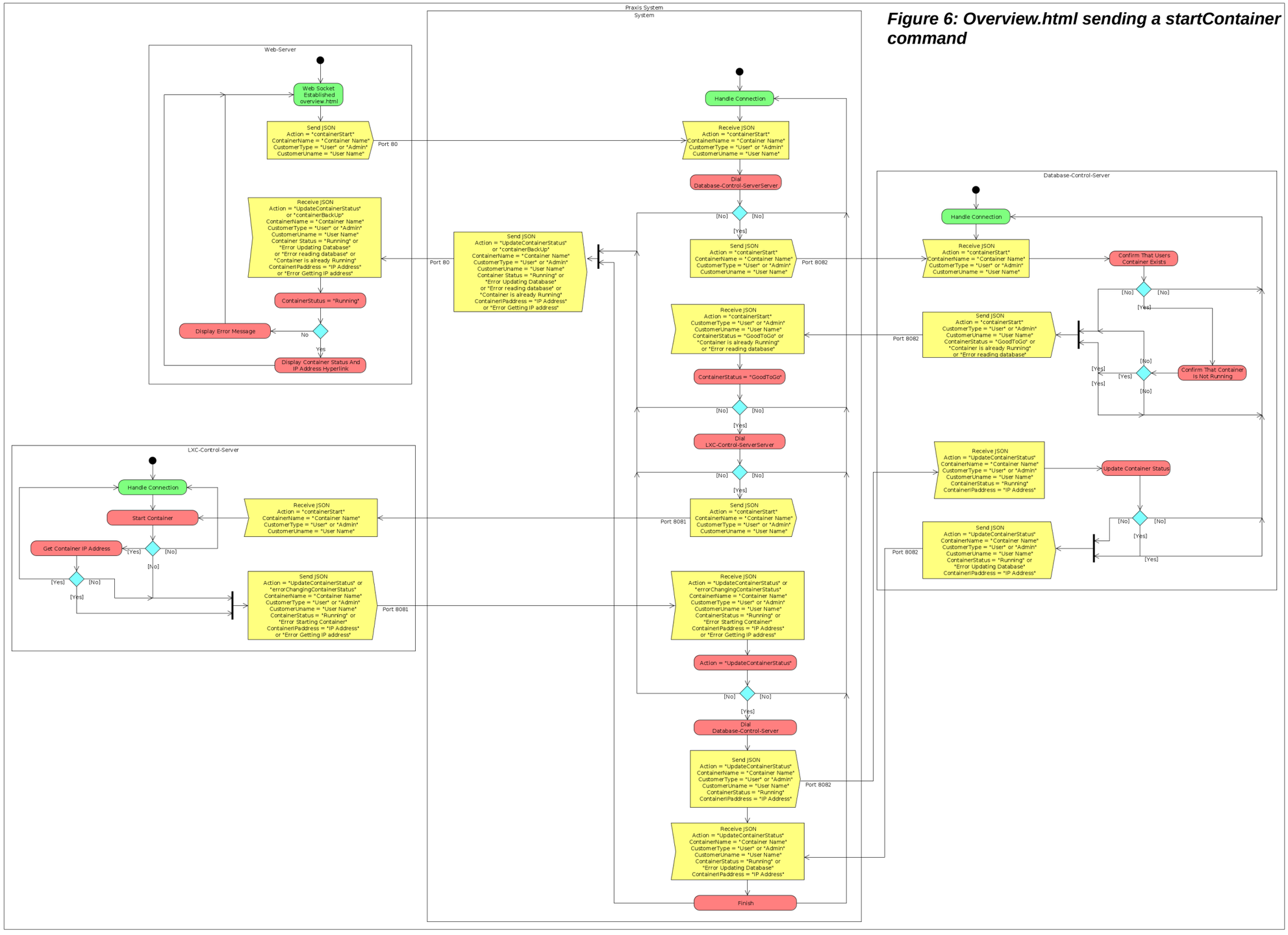


Figure 6: Overview.html sending a startContainer command

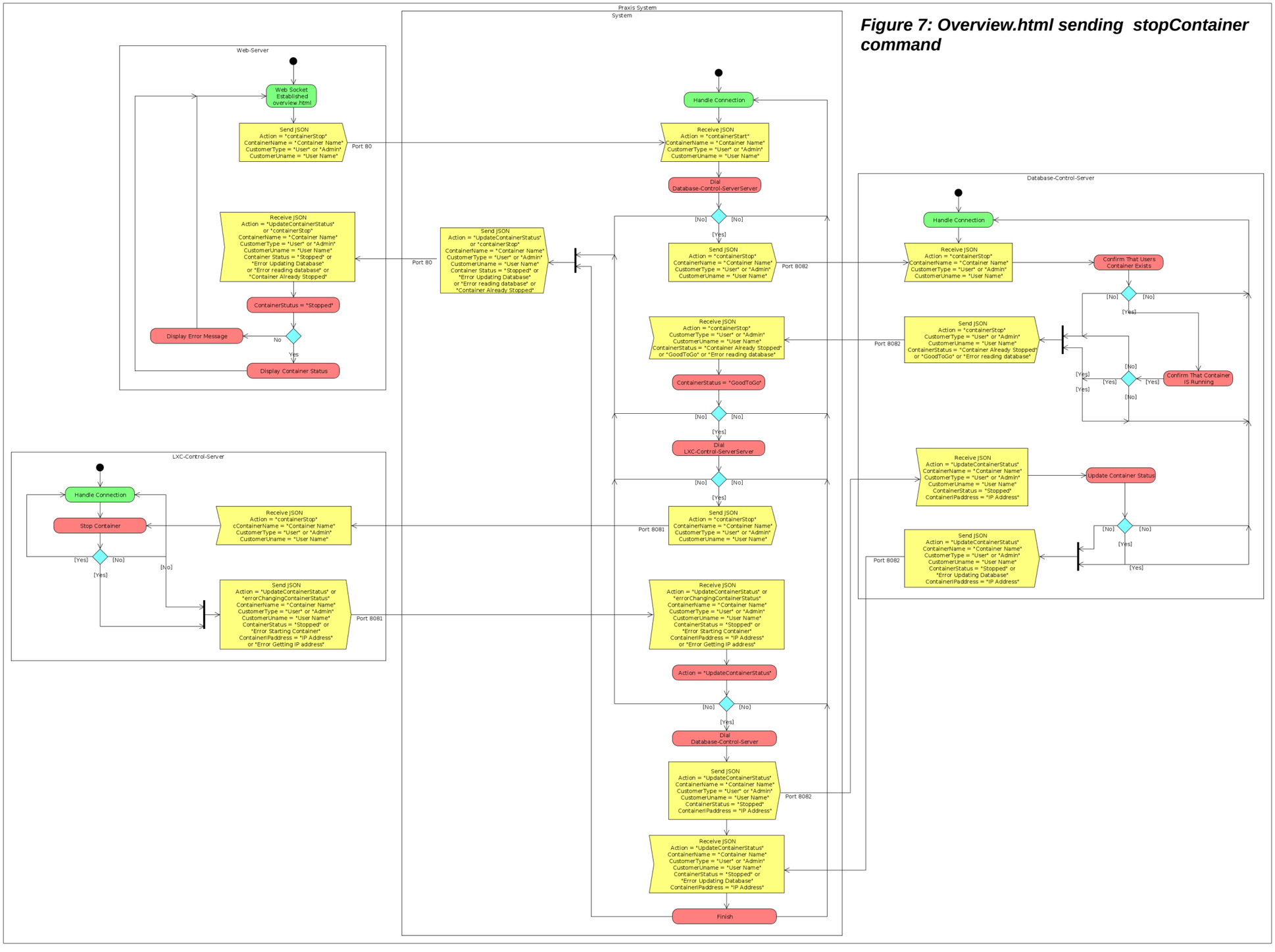


Figure 7: Overview.html sending stopContainer command

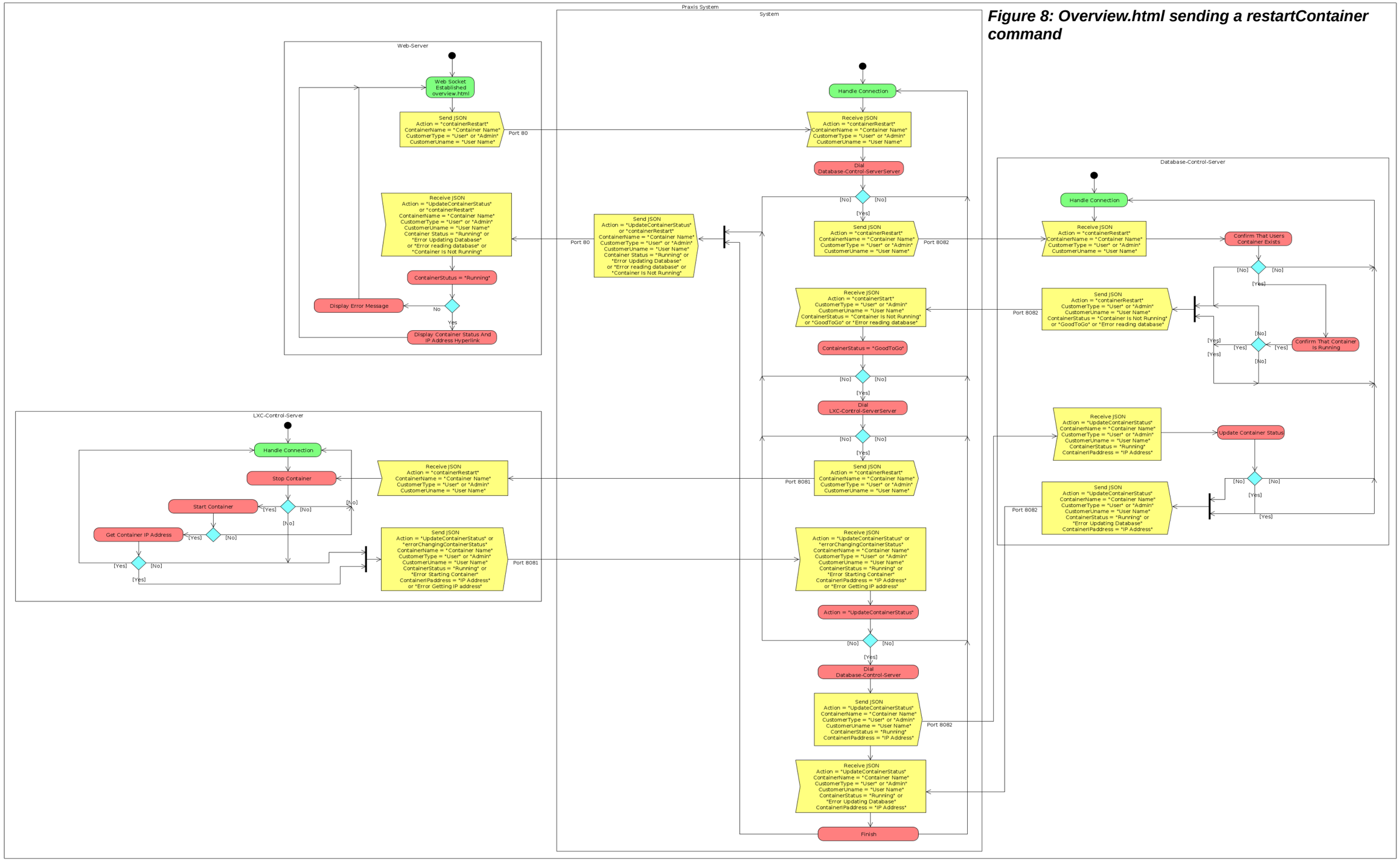


Figure 8: Overview.html sending a restartContainer command

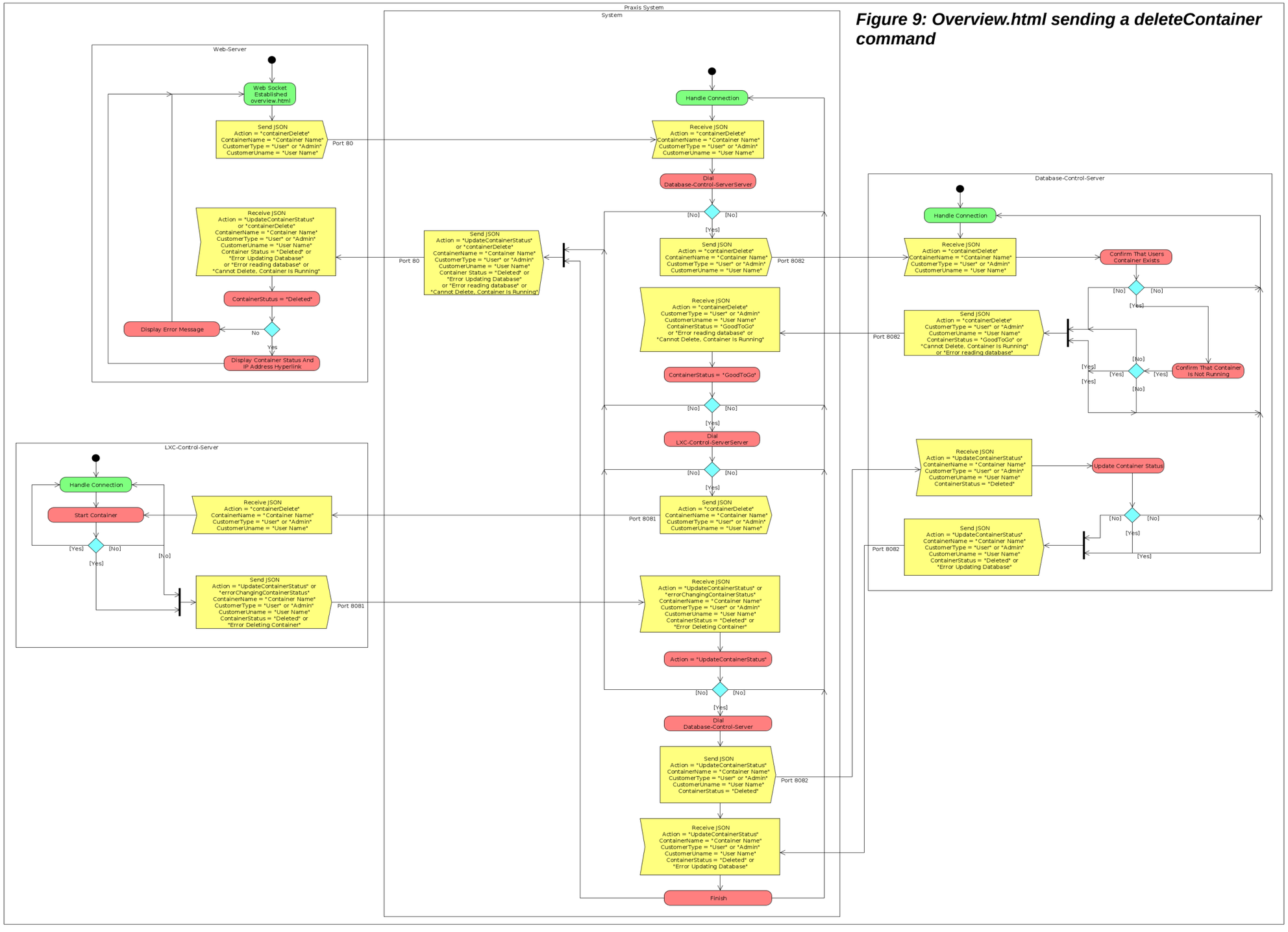


Figure 9: Overview.html sending a deleteContainer command

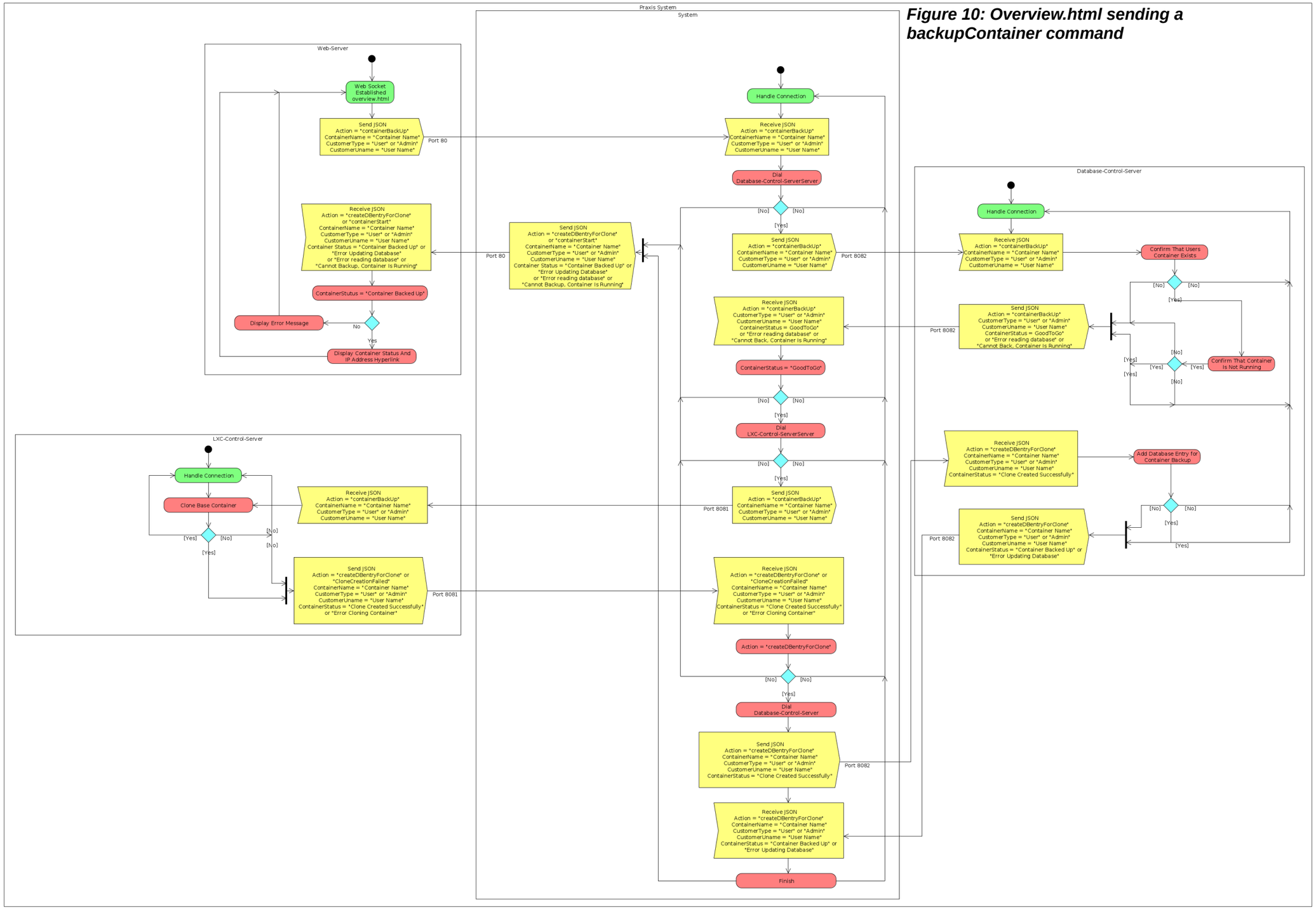


Figure 10: Overview.html sending a backupContainer command

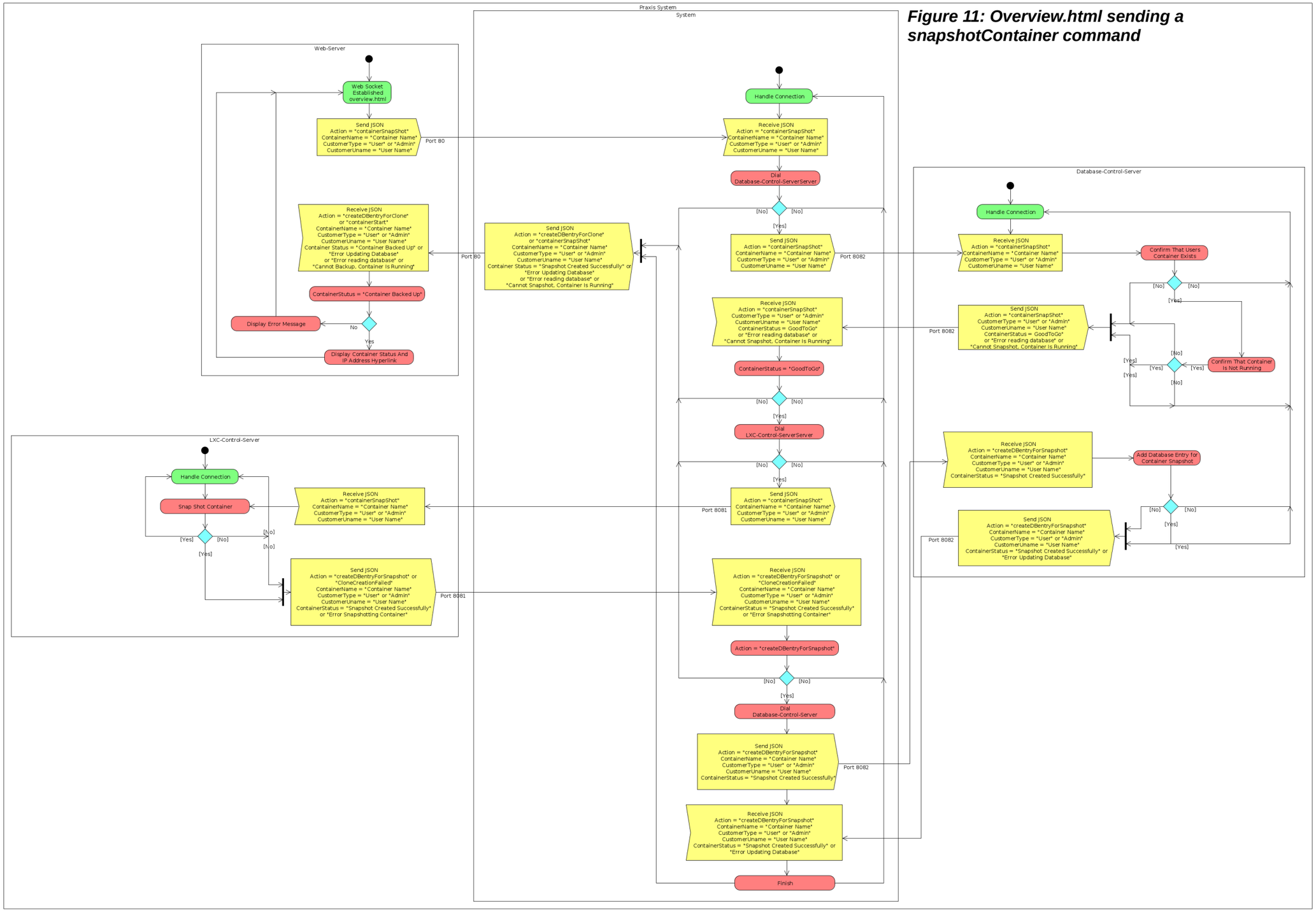


Figure 11: Overview.html sending a snapshotContainer command

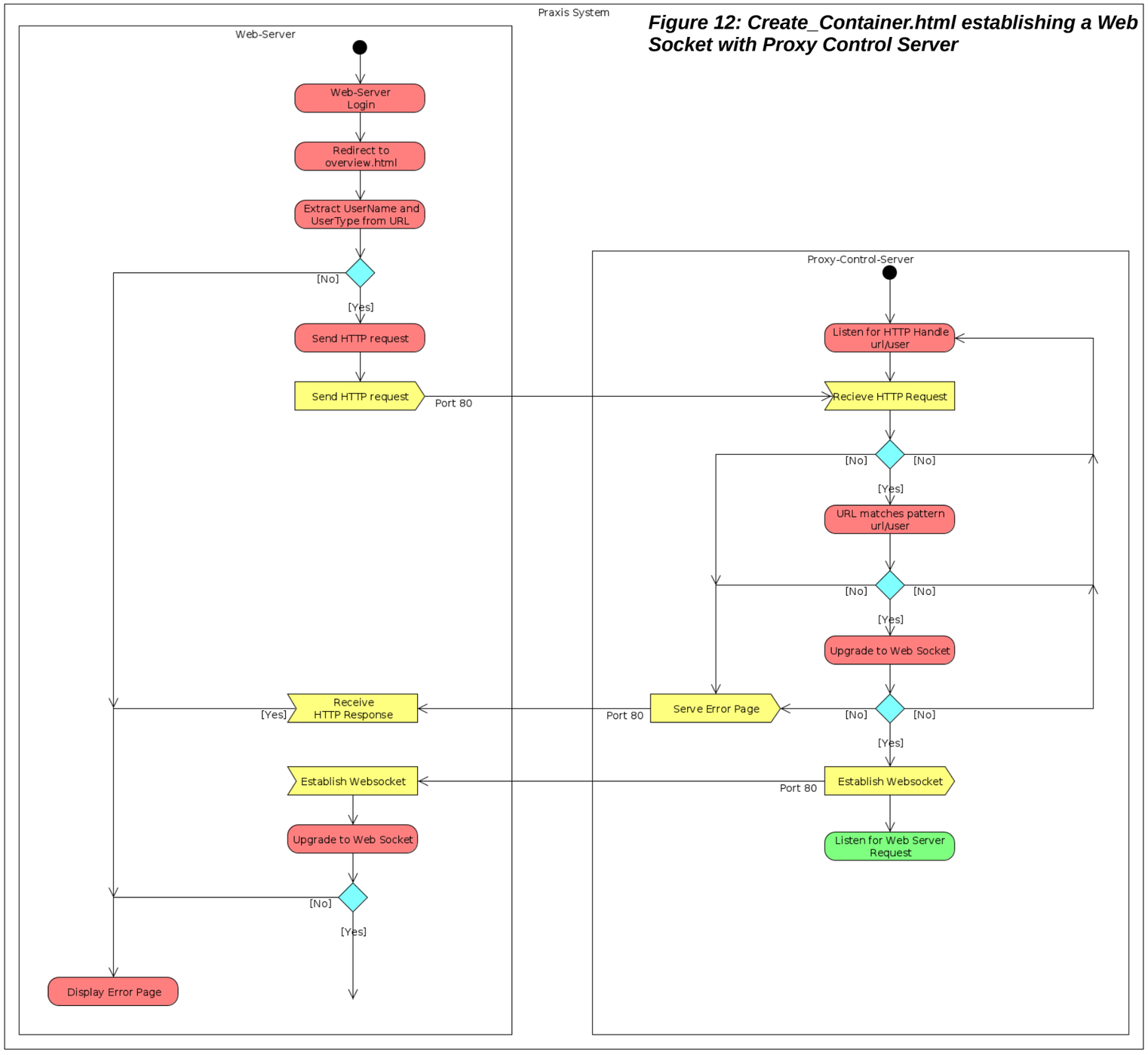


Figure 12: Create_Container.html establishing a Web Socket with Proxy Control Server

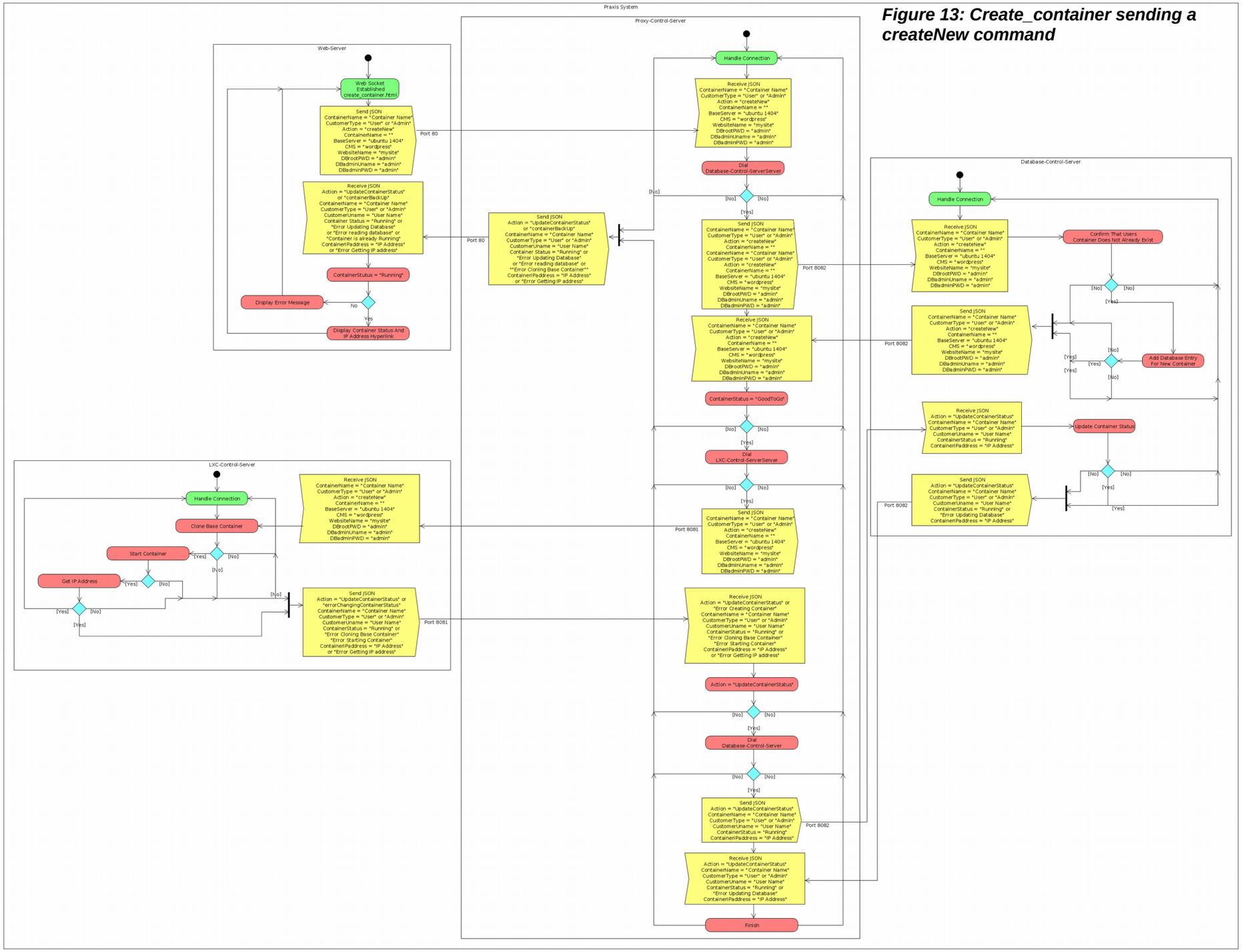


Figure 13: Create_container sending a createNew command

4.4 – Proxy-Control-Server

The Proxy-Control-Server primary responsibility is to co-ordinate activity between the Web-Front-End, the Database-Control-Server and the LXC-Control-Server.

Upon start-up the Proxy-Control-Server will begin listening on Port 80 for any incoming HTTP request from the Web-Front-End. After it receives a request it will upgrade the HTTP request to a Web Socket as illustrated in figure 14. The Proxy-Control-Server is designed to act on any requests it receives from the Web-Front-End concurrently. After step 1 and 2 have been completed, steps 3 – 7 are run separately for each connection that the Proxy-Control-Server receives as illustrated in figure 15.

The actions it takes depends upon the command it is acting upon, illustrated by figures 6 – 13.

In essence the Proxy-Control-Server performs the following tasks:

1. Listen for a HTTP request from the Web-Front-End using a URL in the format of URL/user
2. Assuming no error with the HTTP request, upgrade connection to Web Socket
3. Confirm command is valid with the Database-Control-Server
4. Assuming the command is valid, forward command to the LXC-Control-Server
5. Assuming command has been completed, update the Database-Control-Server with the change in containers status
6. Assuming the Database-Control-Server updates the status correctly, forward the updated status to the Web-Front-End
7. If any of the steps between 1 – 7 produces an error, the Proxy-Control-Server will terminate the request and send an error message to the Web-Front-End

Figure 14: Proxy-Control-Server Start-up Sequence

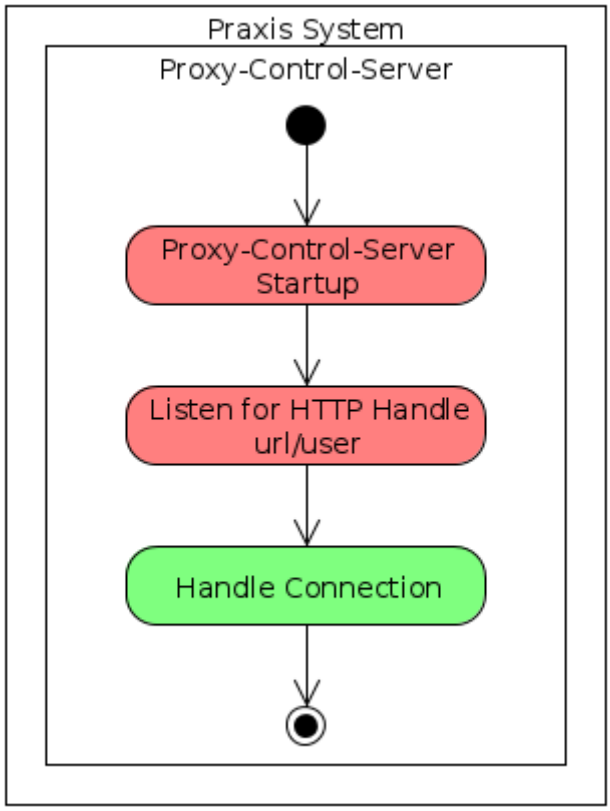
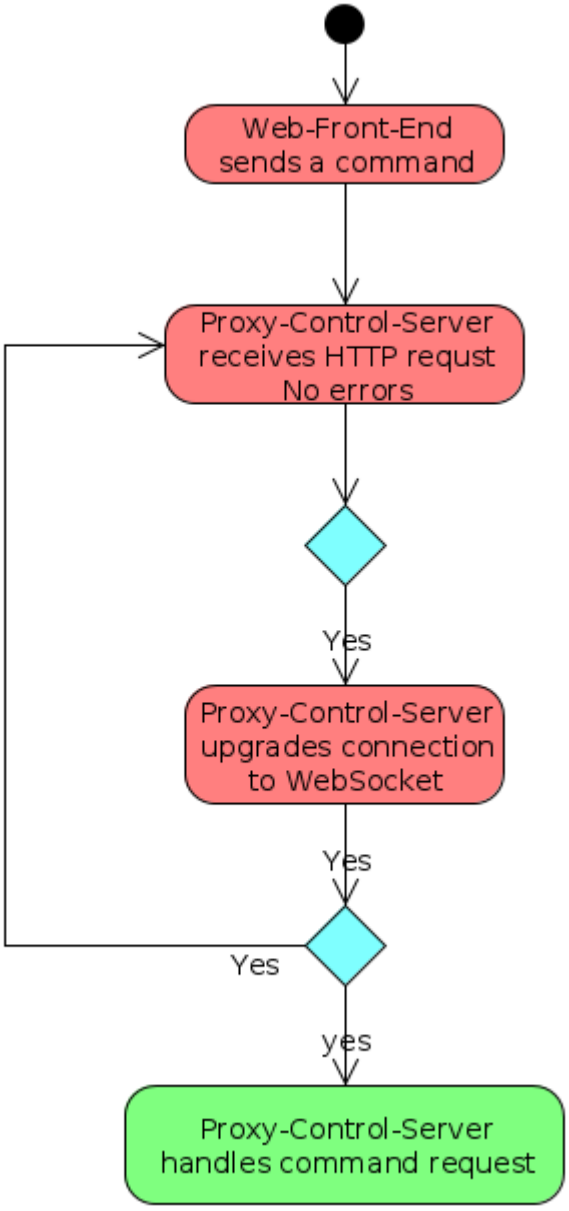


Figure 15: Proxy-Control-Server Concurrency



4.5 – Database-Control-Server

The Database-Control-Servers primary responsibility is to receive requests and validate those requests. Another function of the Database-Control-Server is to synchronise the status of existing containers between it and the LXC-Control-Server upon boot up and ensure that customer is receiving the correct statuses of their containers.

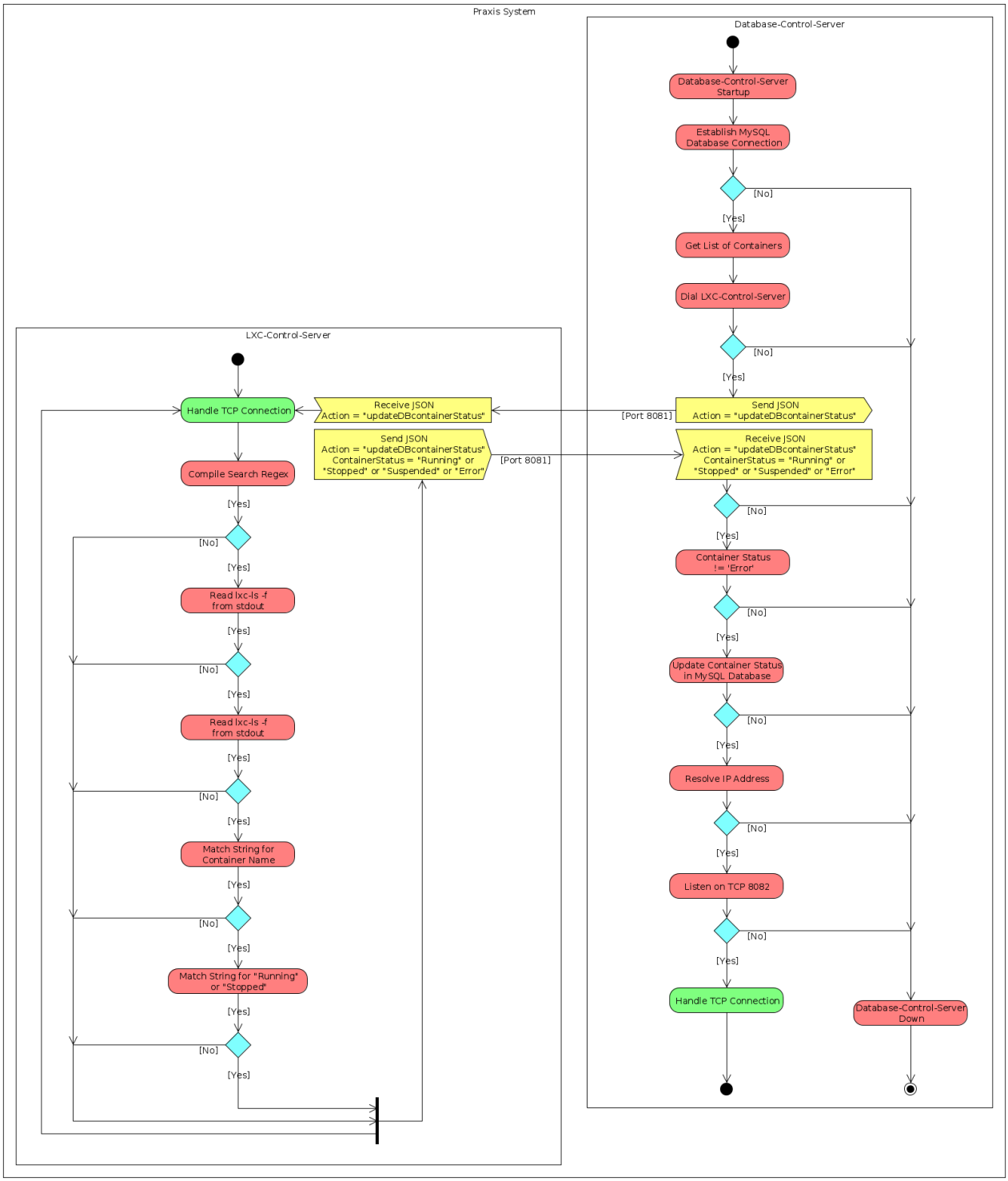
Upon boot, the server will open TCP Network Socket with the LXC-Control-Server. It will send a list of containers that it has registered and request the LXC-Control-Server to return that containers current status, either 'Running' or 'Stopped'. If the LXC-Control-Server has no record of that container or is for another unable to ascertain it's status, it will return a status of 'Suspended'.

Once this operation has been completed, the Database -Control-Server will listen on TCP port form incoming connection from the Proxy-Control-Server. The Proxy-Control-Server will contact the Database-Control-Server for the following reasons:

1. To establish the current status of a container and that it's status does not conflict with the command it received for that container, such as stopping a container that is already stopped.
2. To update the status of a container that has had it's status changed by the LXC-Control-Server, such as starting a stopped container.
3. To add a new entry to the database for a new container, a container back-up or a container snapshot.

Note: The following activity diagrams represent a more detailed image of the Database-Control-Server operation for the command it receives than activity diagrams previously shown for overview.html and create_new.html sections..

Figure 16: Database-Control-Server Start-up Sequence



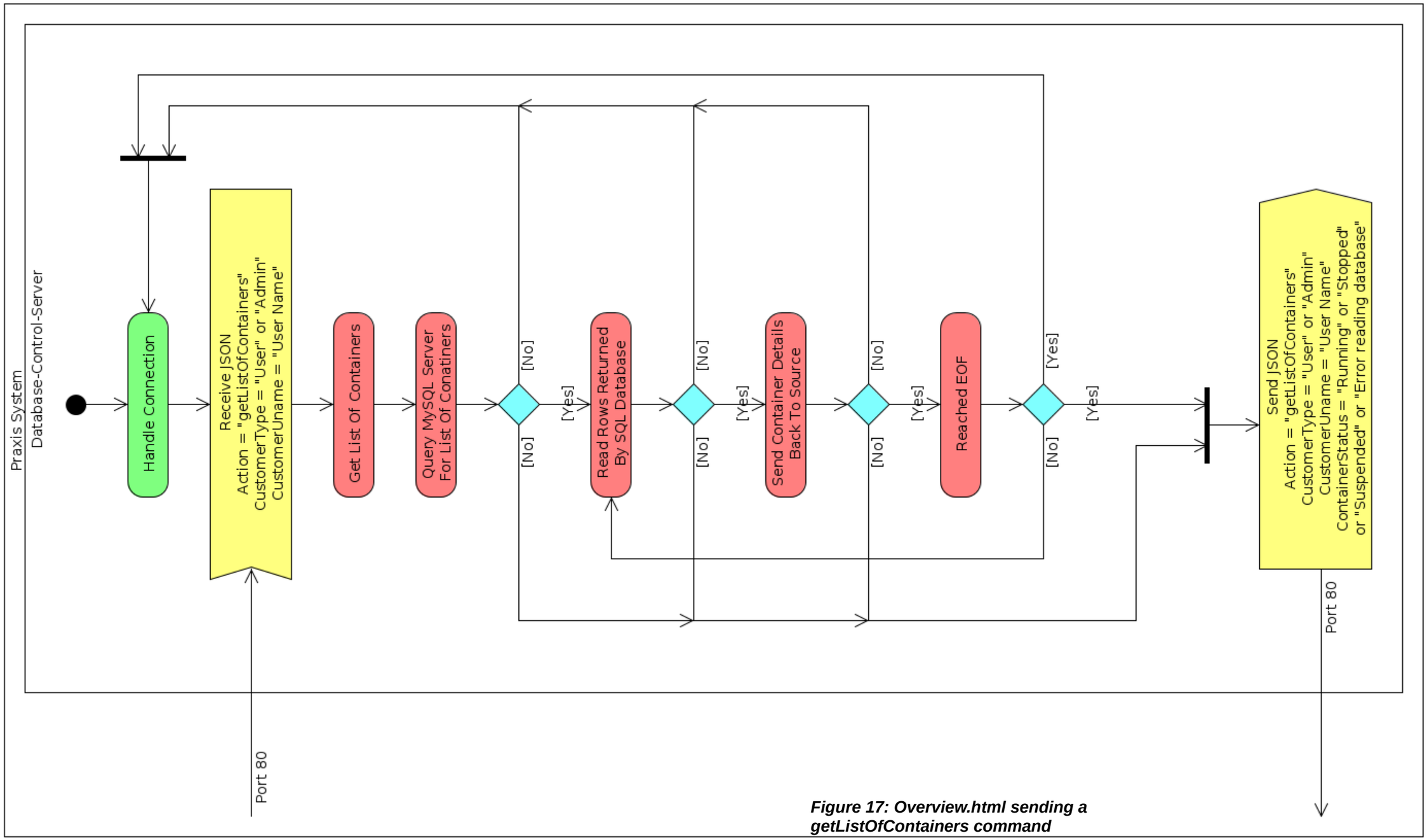


Figure 17: Overview.html sending a getListOfContainers command

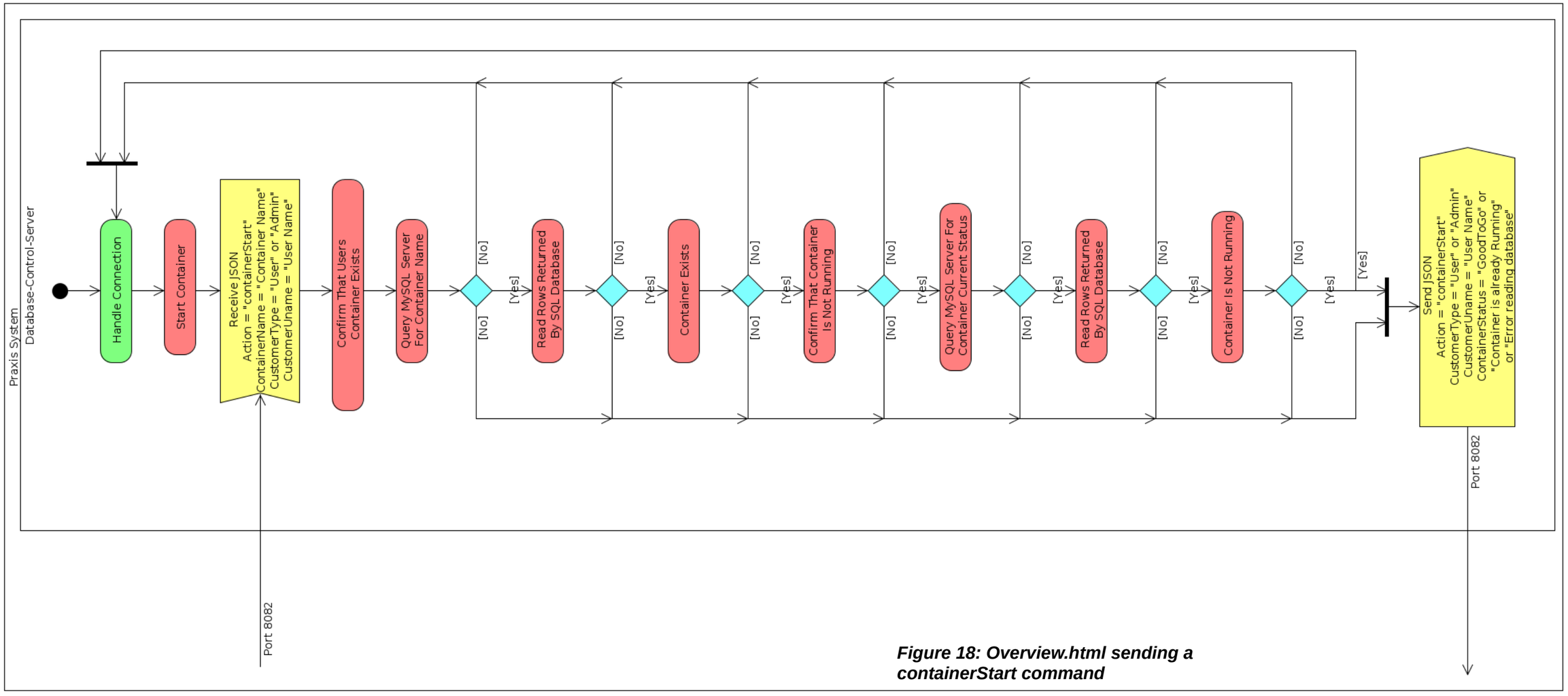


Figure 18: Overview.html sending a containerStart command

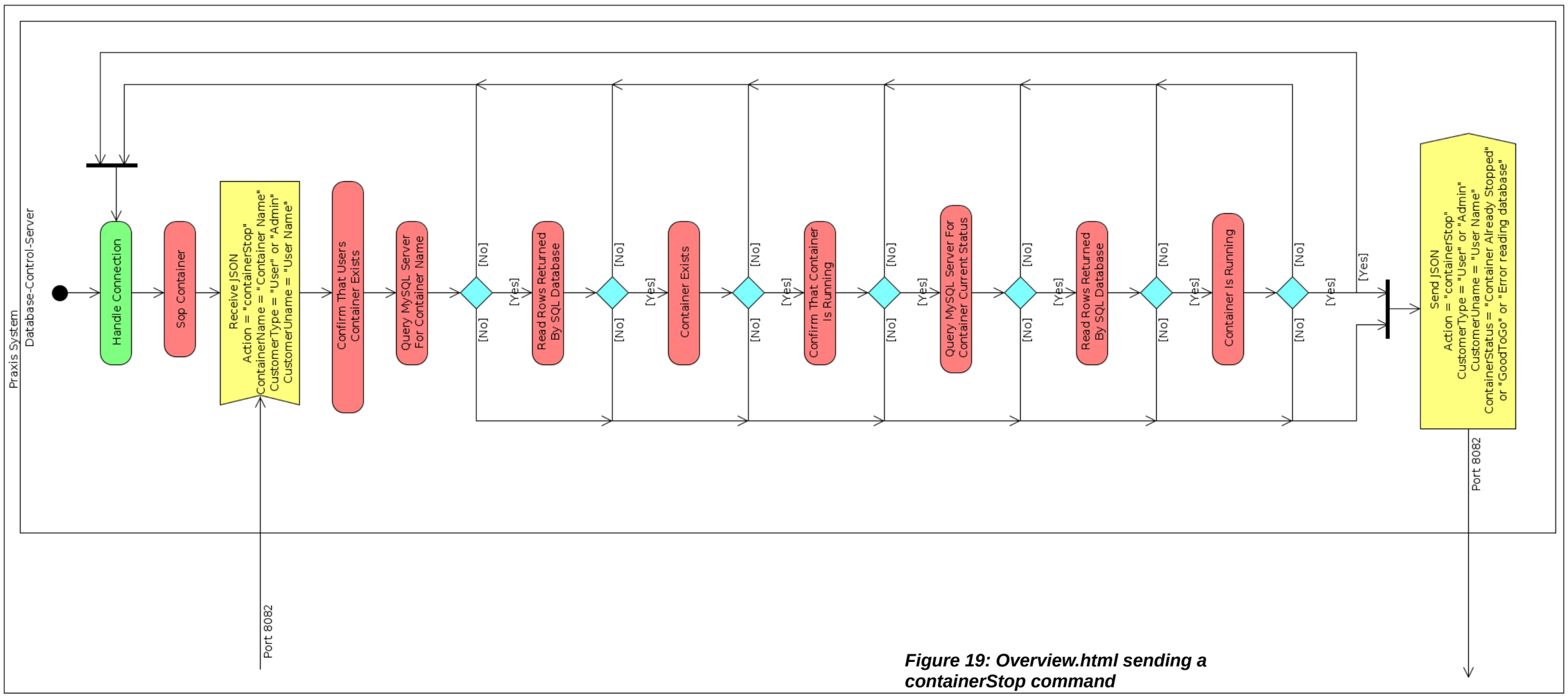


Figure 19: Overview.html sending a containerStop command

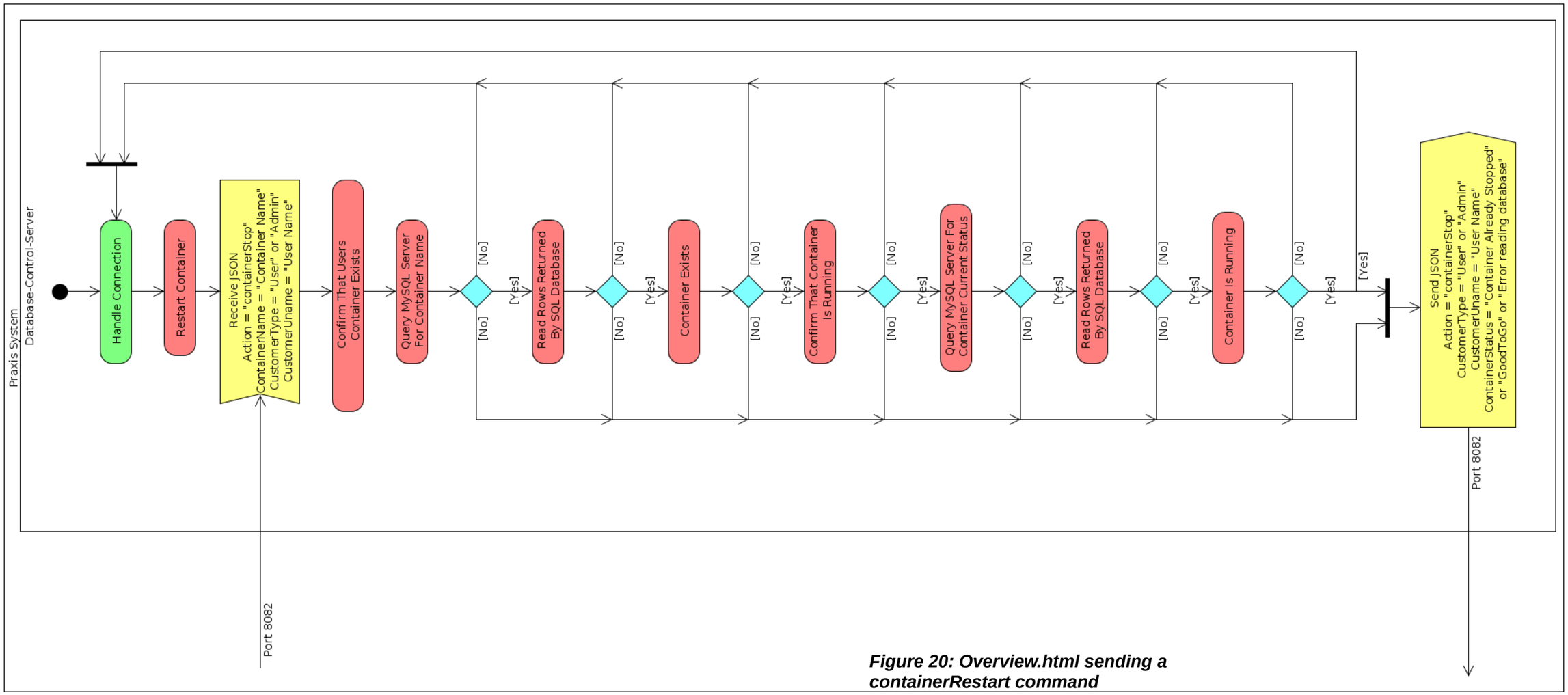


Figure 20: Overview.html sending a containerRestart command

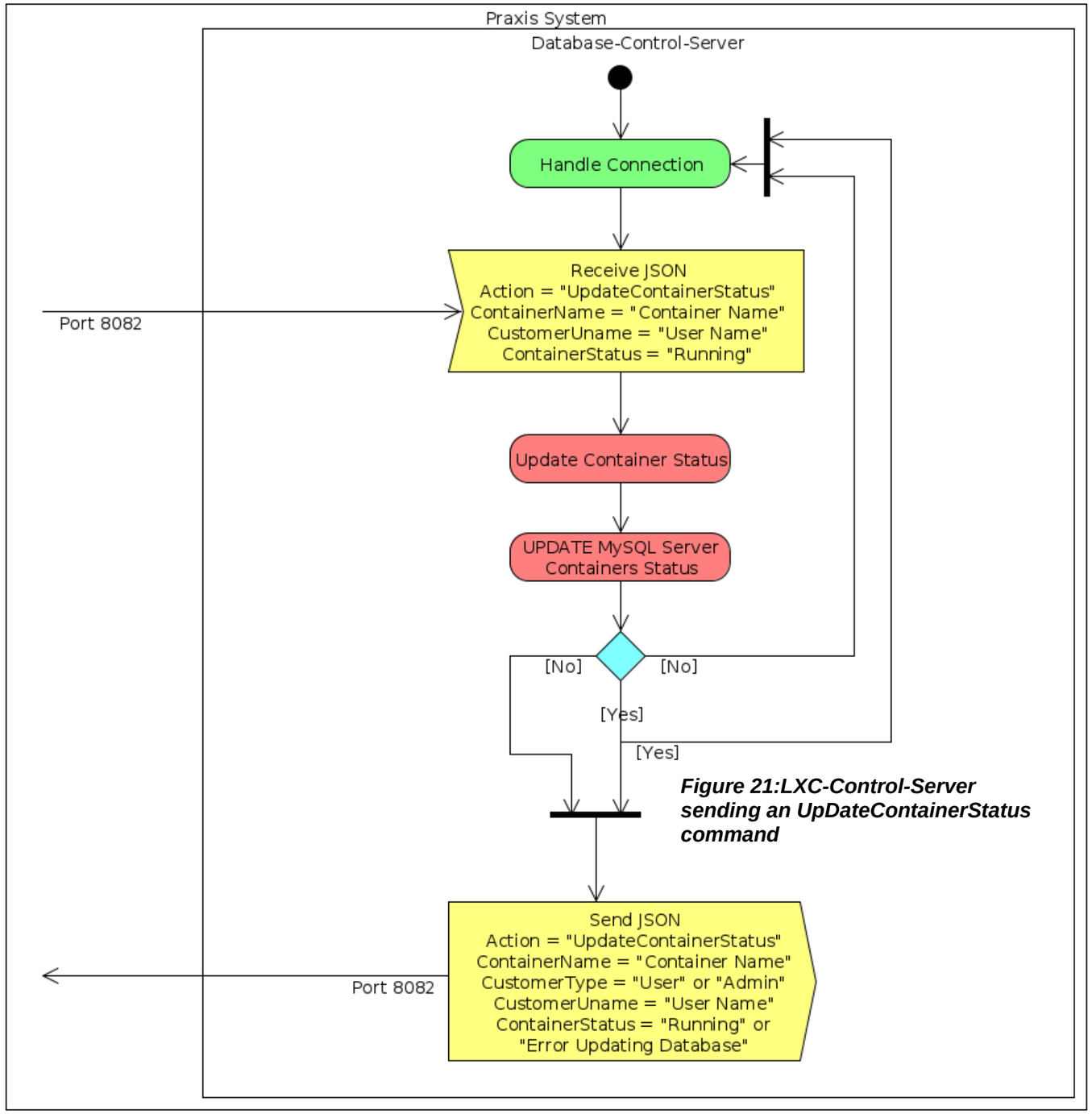


Figure 21:LXC-Control-Server sending an UpDateContainerStatus command

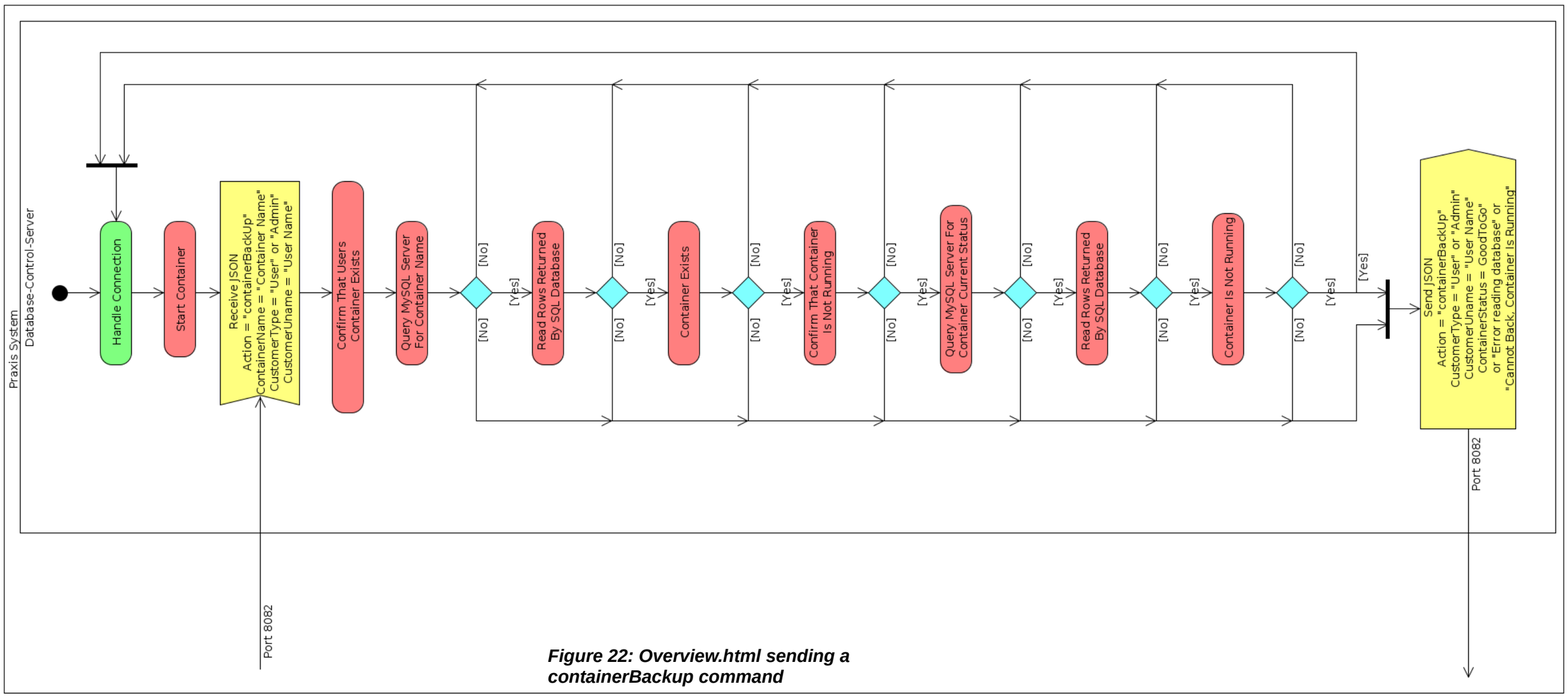


Figure 22: Overview.html sending a containerBackup command

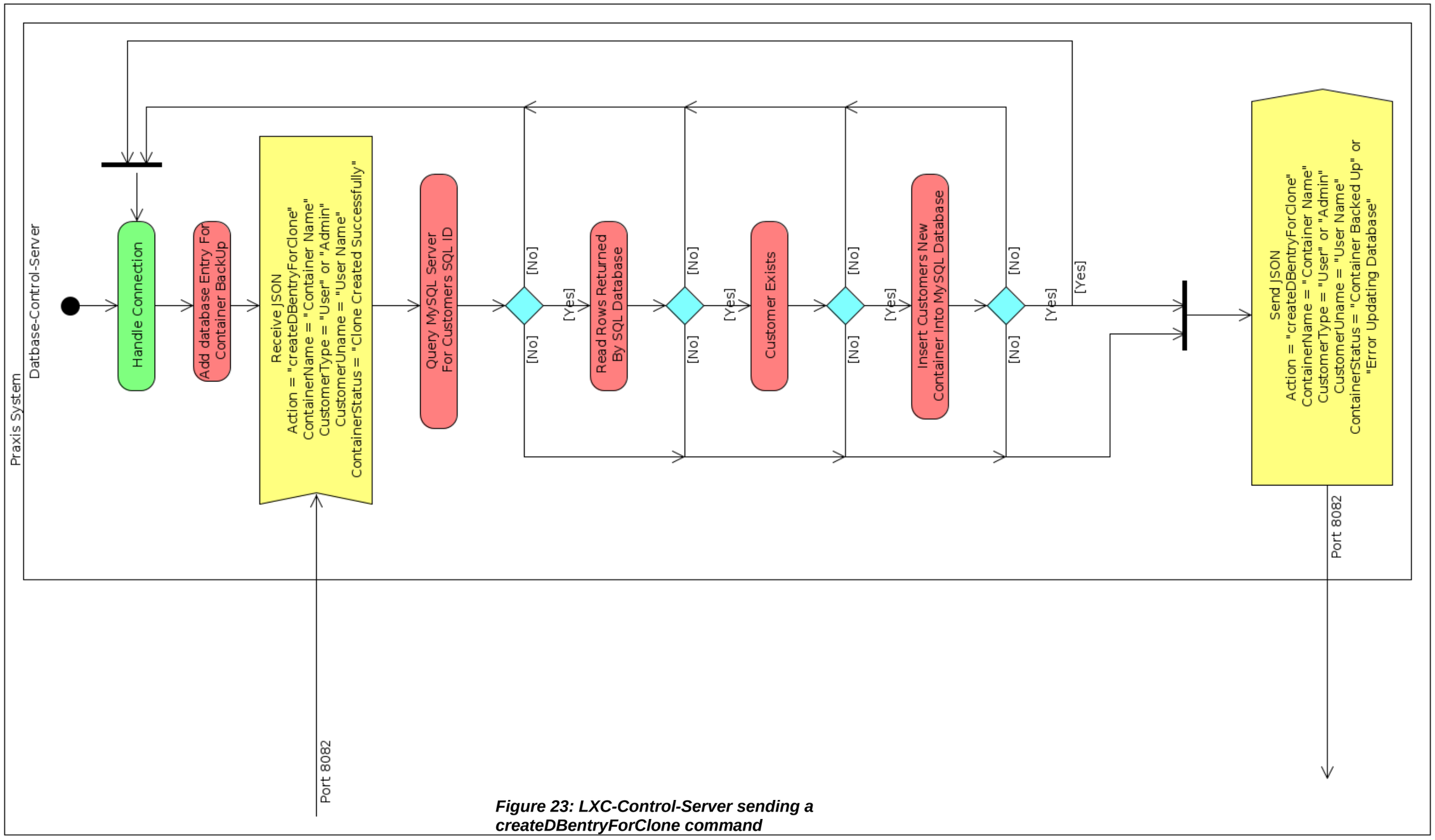


Figure 23: LXC-Control-Server sending a createDBentryForClone command

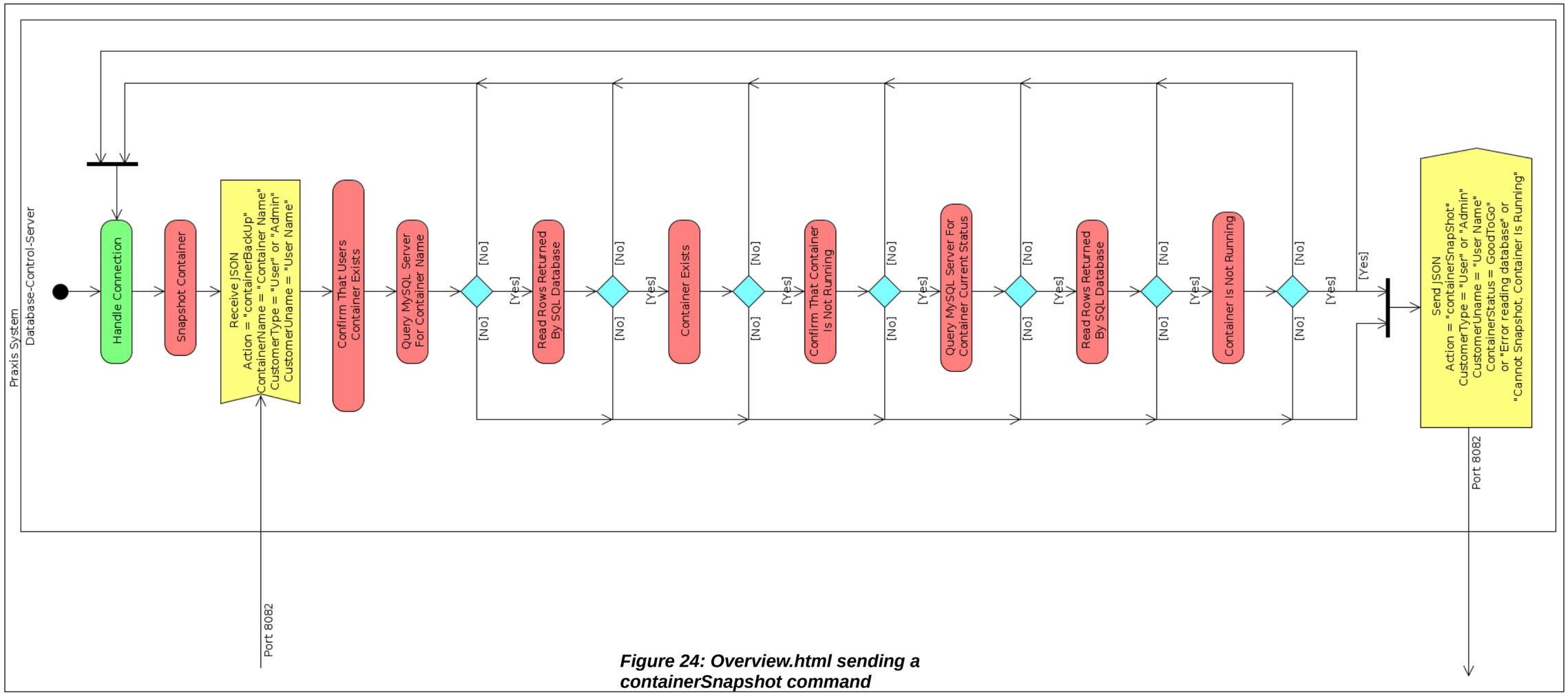


Figure 24: Overview.html sending a containerSnapshot command

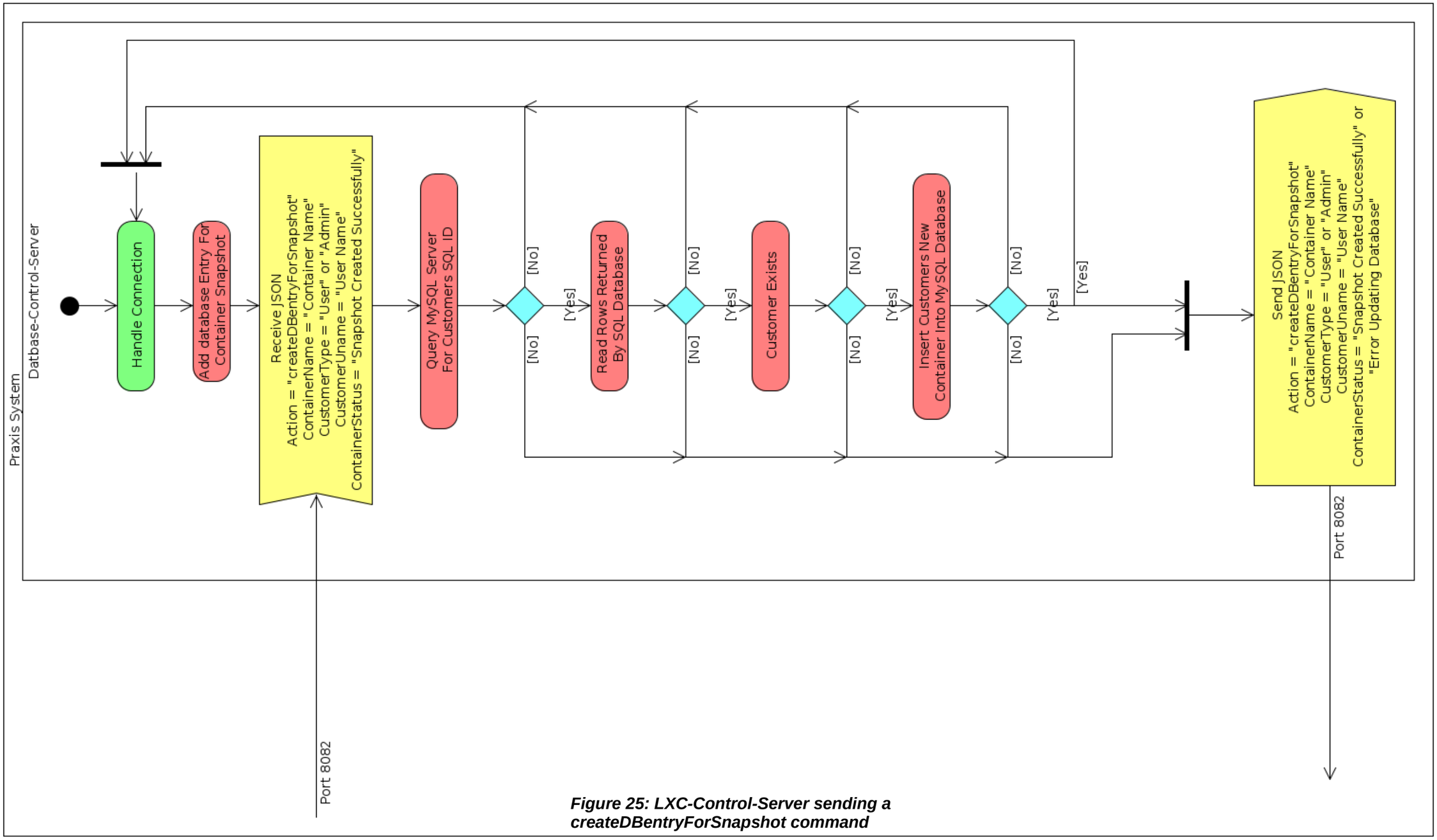


Figure 25: LXC-Control-Server sending a createDBentryForSnapshot command

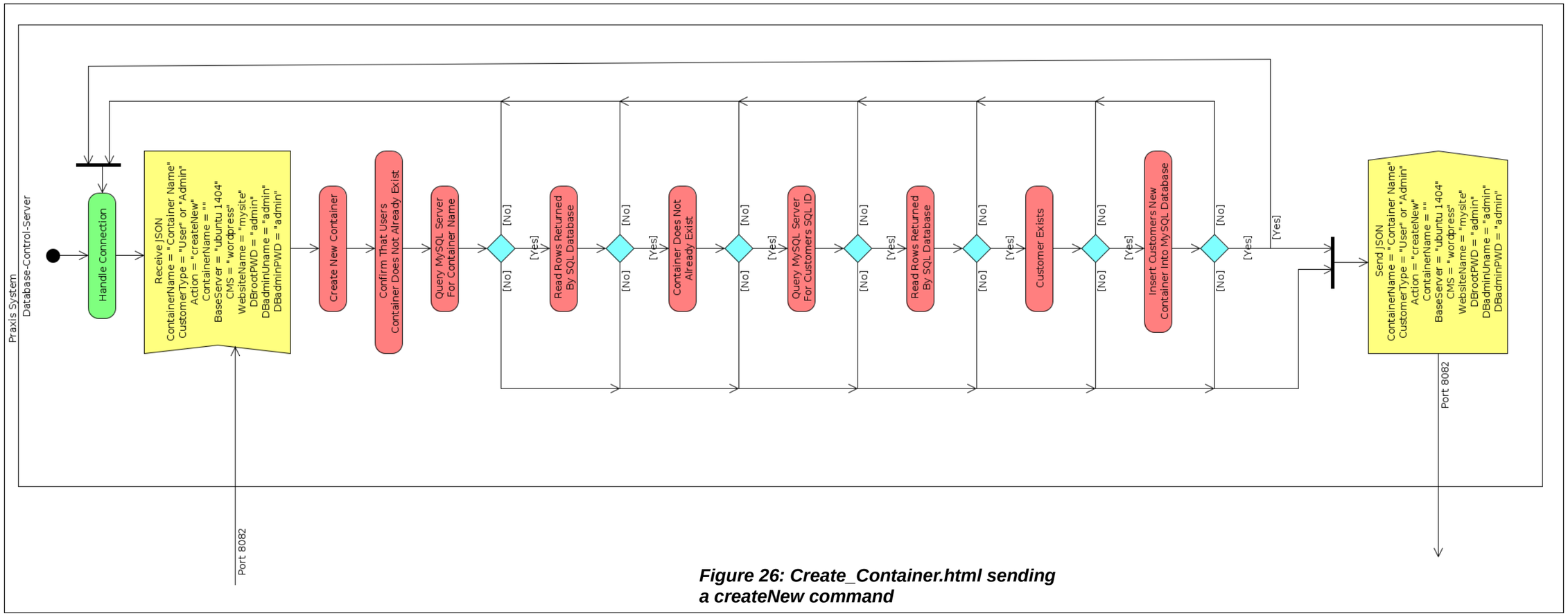


Figure 26: Create_Container.html sending a createNew command

4.6 – LXC-Control-Server

The primary purpose of the LXC-Control-Server is to create, destroy, back-up, snapshot and change the running status of containers. Requests for these actions have already been validated by the Proxy-Control-Server and Database-Control-Server. In the event that is unable to take the requested action on a particular container will result in an error message being returned to the Web-Front-End.

Upon the boot up of the Database-Control-Server it request the current status of any registered containers in the Database. The LXC-Control-Server perform Regex match against the shell command **'lxc-ls -f'** and matches container name and current status, which it then returns to the Database-Control-Server. If unable to find a match it will return a status of 'suspended' for that container.

Note: The following activity diagrams represent a more detailed image of the LXC-Control-Server operation for the command it receives than activity diagrams previously shown for overview.html and create_new.html sections..

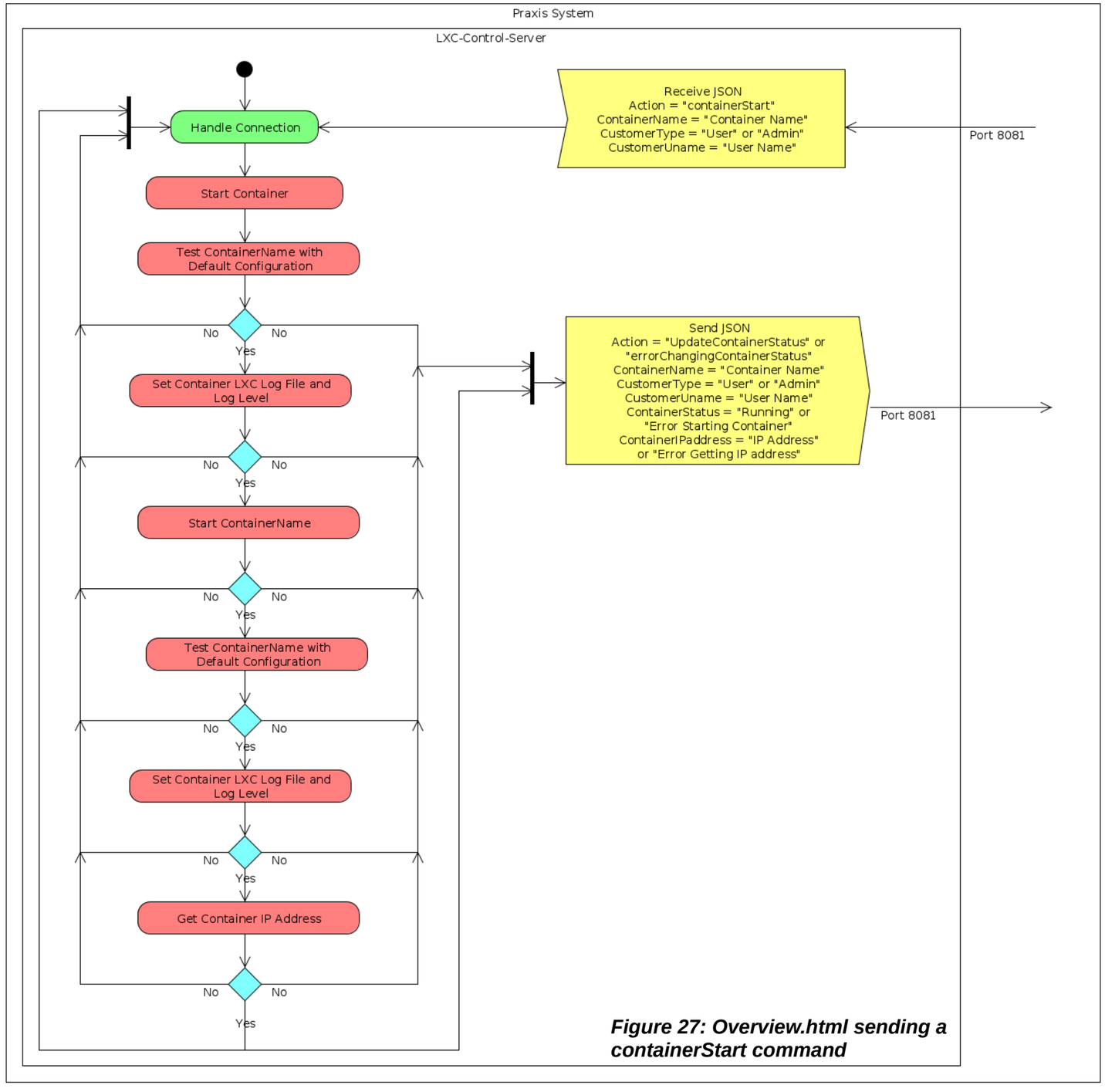


Figure 27: Overview.html sending a containerStart command

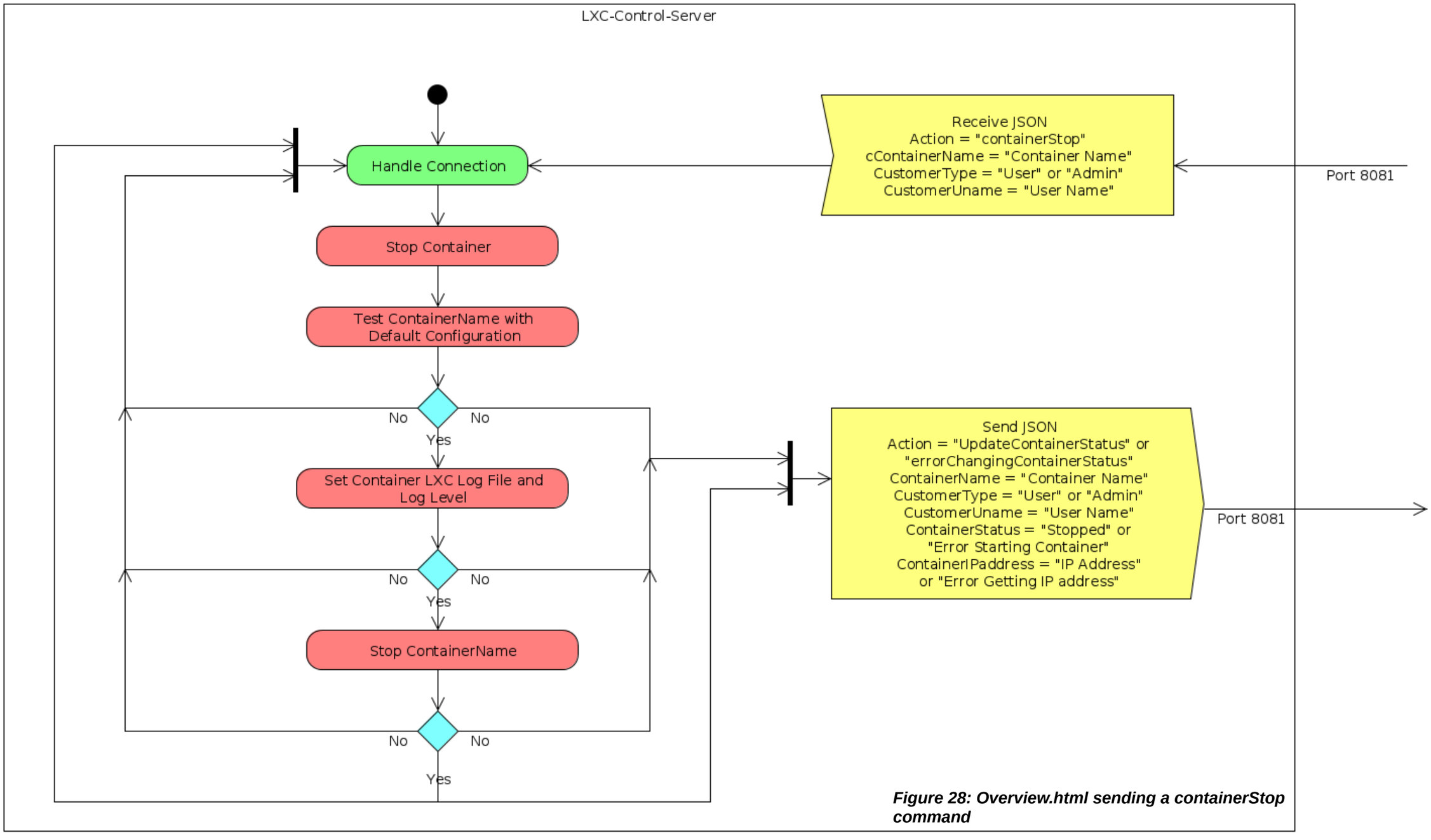


Figure 28: Overview.html sending a containerStop command

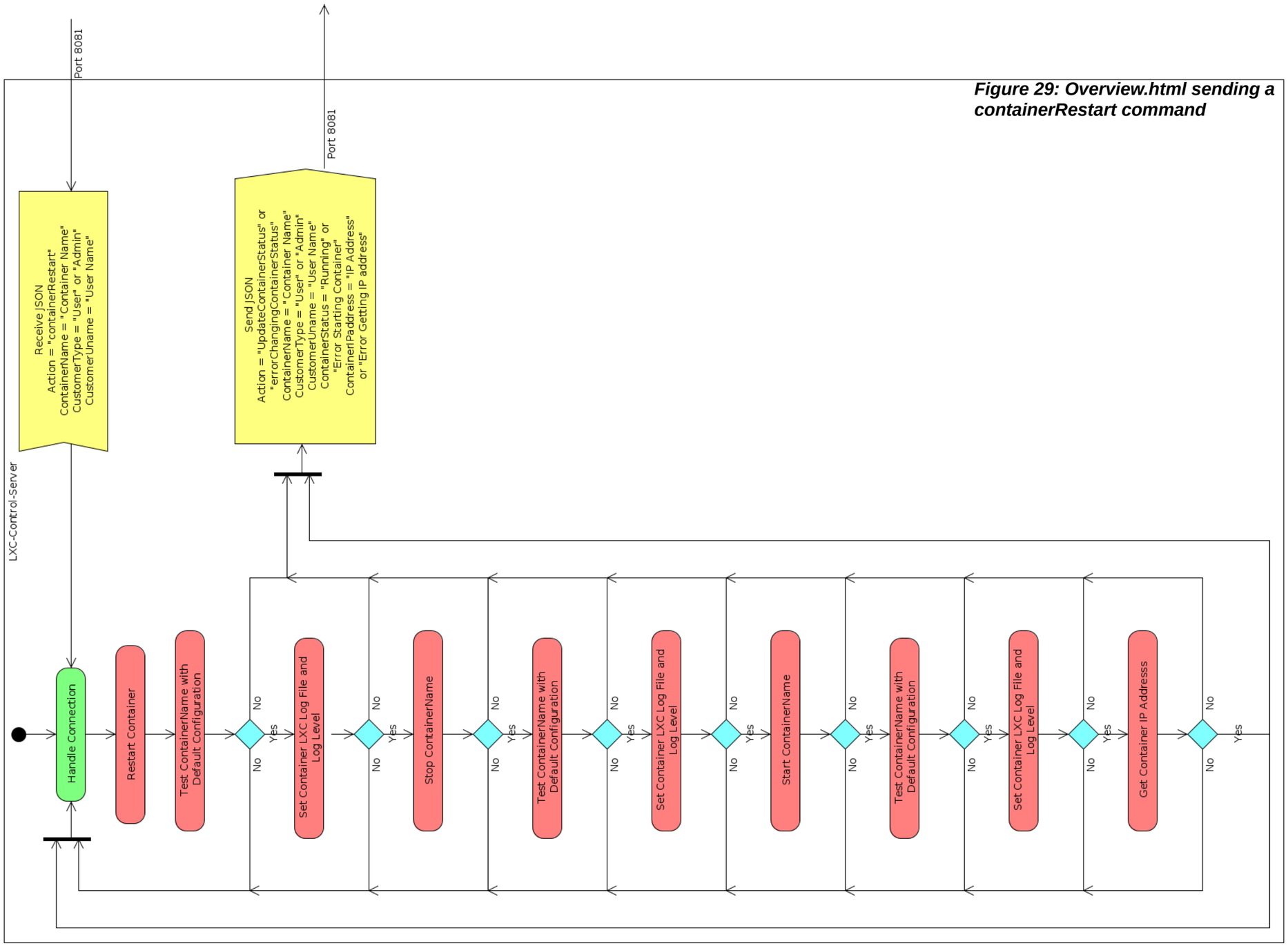


Figure 29: Overview.html sending a containerRestart command

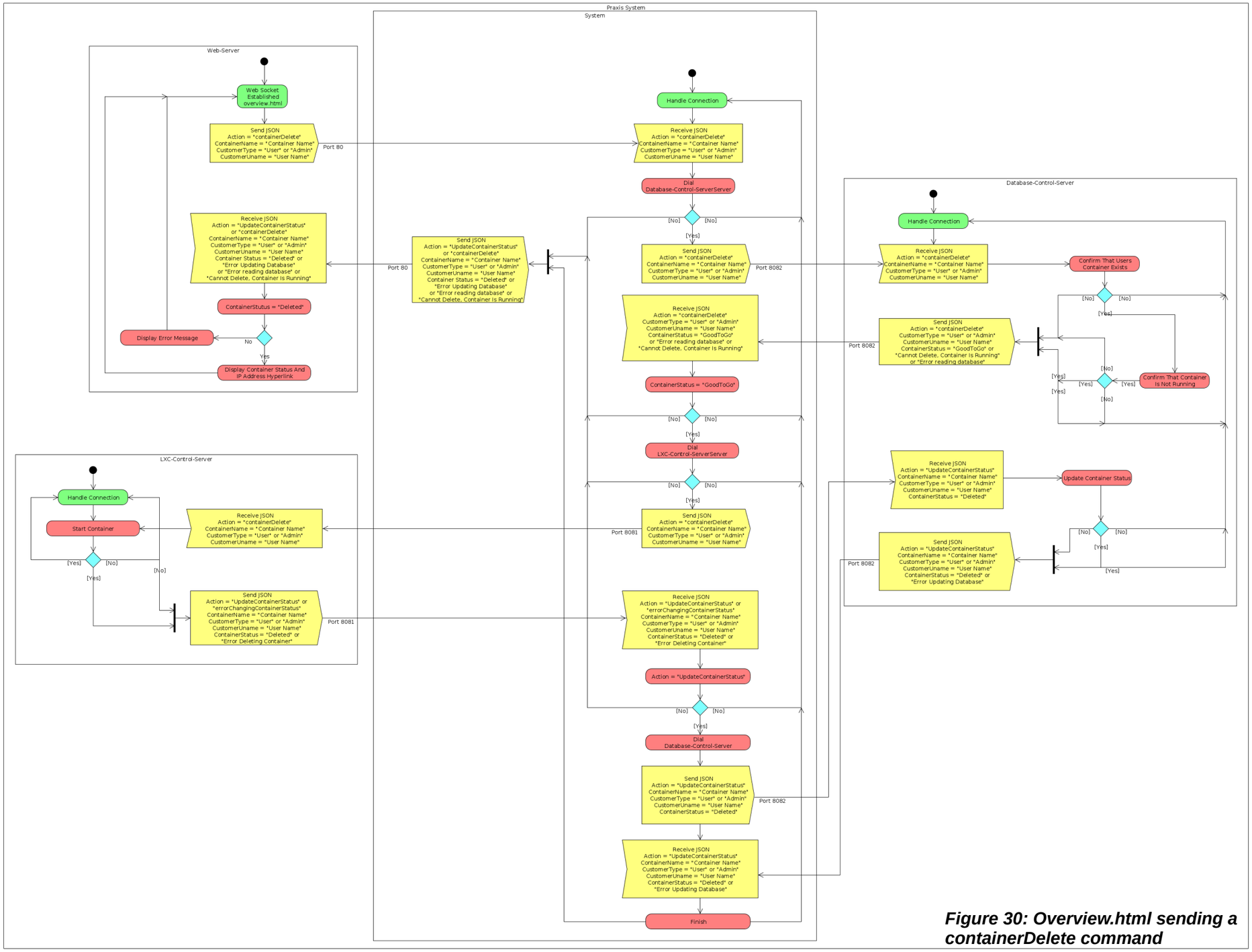


Figure 30: Overview.html sending a containerDelete command

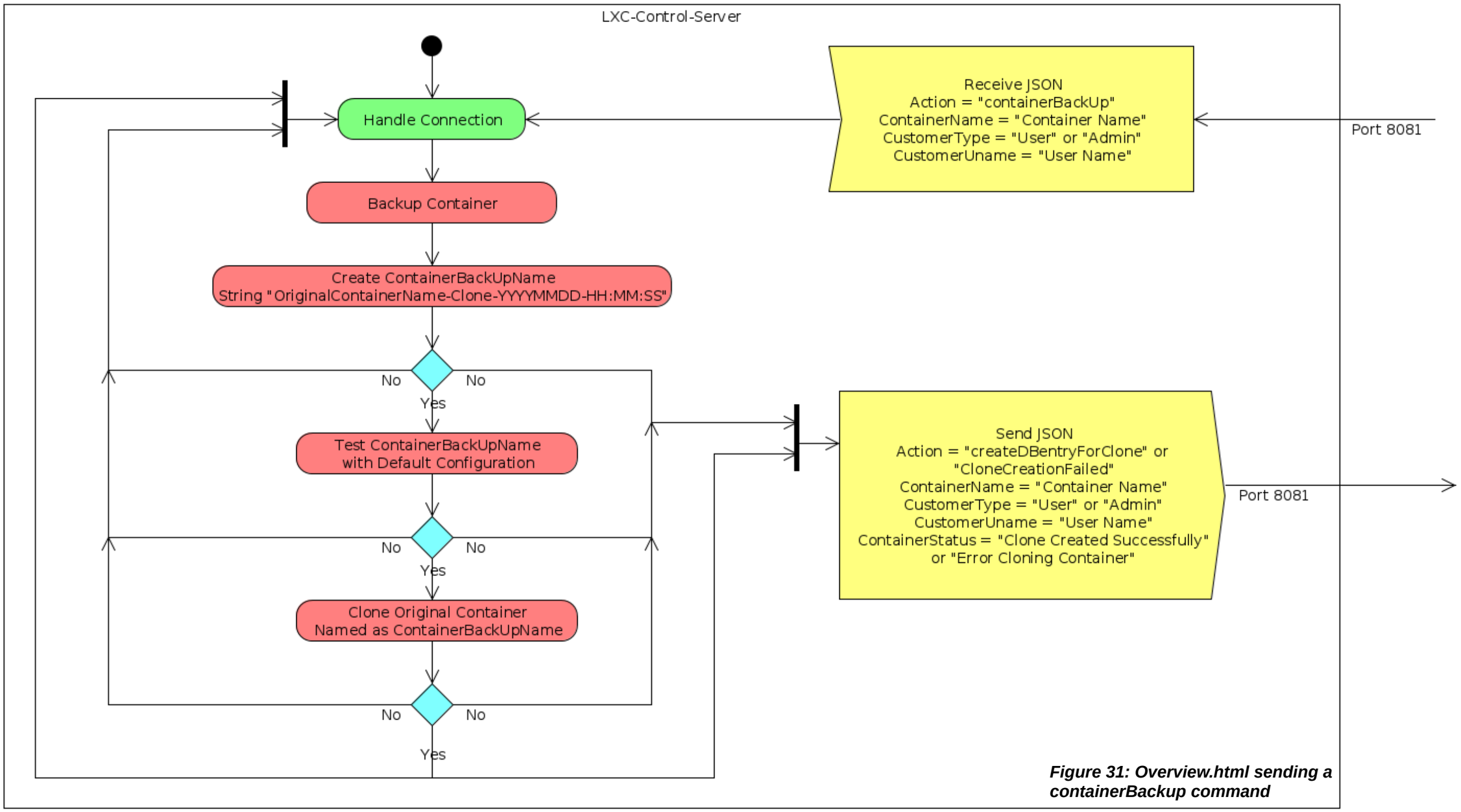


Figure 31: Overview.html sending a containerBackup command

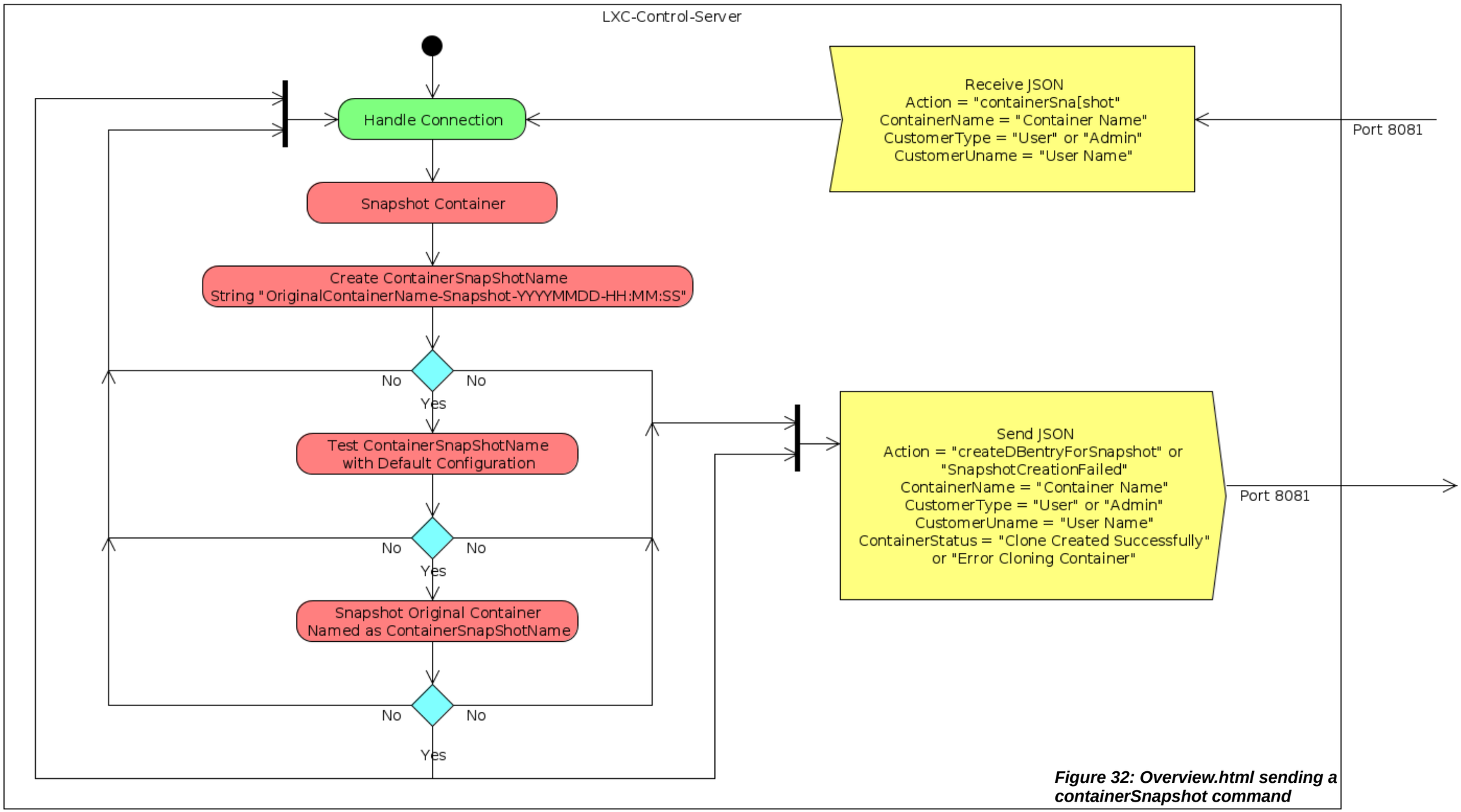


Figure 32: Overview.html sending a containerSnapshot command

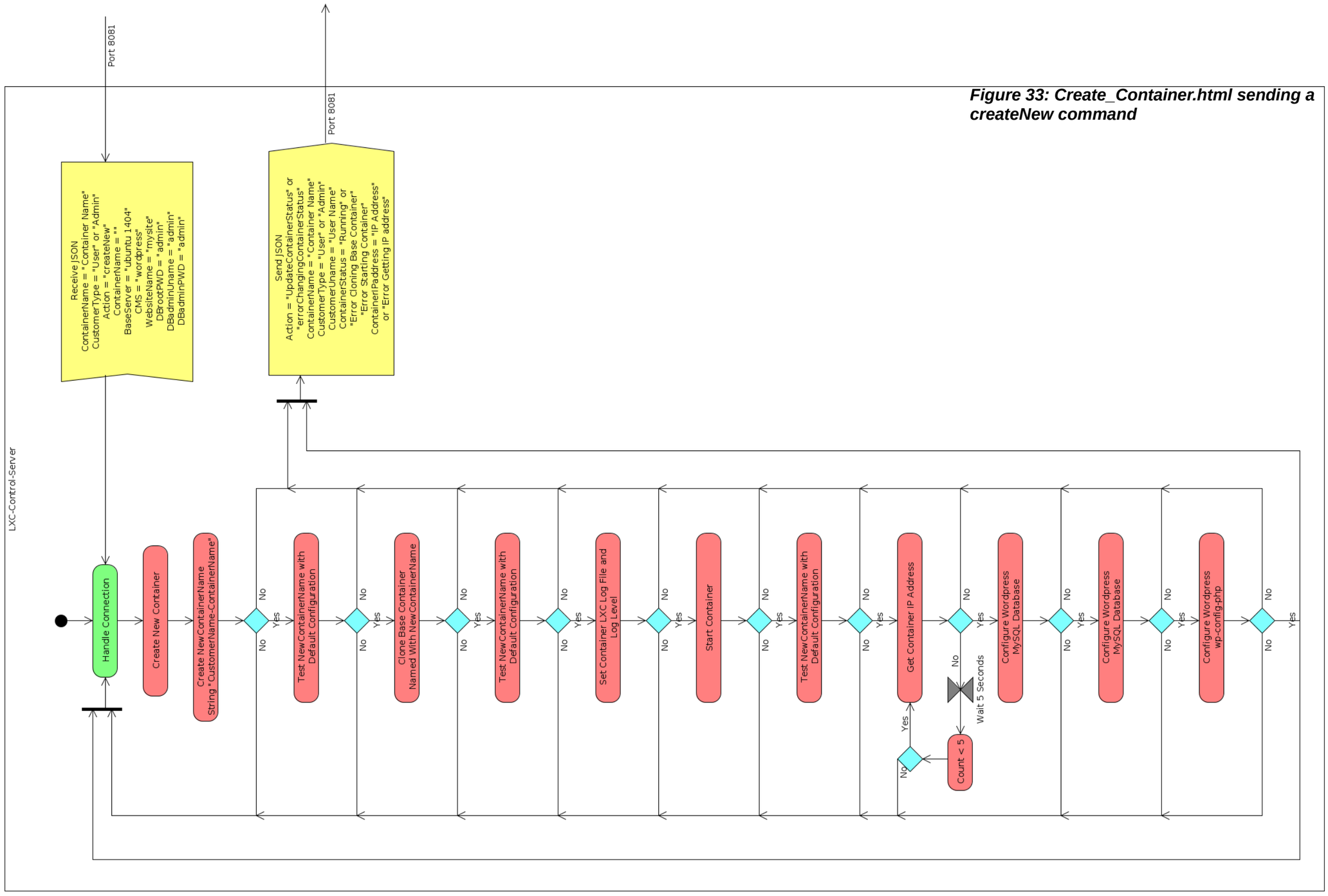


Figure 33: Create_Container.html sending a createNew command

5 – Project Implementation

5.1 – Introduction

The implementation of the project was a massive undertaking for the short time allowed. As such it will be impossible to go into detail on every element of the projects implementation and rationale behind every decision. For this reason the following chapter is limited to those elements which are considered essential to the projects implementation. Each sub-heading will provide a description of the technology, the reason for it's inclusion and where appropriate the reason for not using a different technology.

The following is line count of the various code elements of the project.

Table 2: Breakdown of Code Usage Across Services

#	Service Name	Technology	Total Lines of Code
	Web-Front-End	HTML	425
	Web-Front-End	CSS	279
	Web-Front-End	Javascript	354
	Proxy-Control-Server	Go	301
	Database-Control-Server	Go	854
	Database-Control-Server	MySQL	250
	LXC-Control-Server	Go	531
	LXC-Control-Server	Bash	50

Table 3: Total of Code Usage by Technology

#	Technology	Total Lines of Code
	HTML	425
	CSS	279
	Javascript	354
	Go	1686
	MySQL	250
	Bash	50
Total		3044

5.2 – Web-Front-End

The Web-Front-End was created using a blend of HTML, CSS and Javascript. The requirements for the project stipulated only a basic site that was capable of showing the functionality of the back end. To that effect the index page contains only Lorem Ipsum place holder text, see figure 34. Some thought was given to a colour scheme for use in later development cycles. As stated earlier this was not a requirement and any Web Design decisions are to be completely re-evaluated upon completion of the back-end of the service.

On the index page there is a login link that brings a person to page that provides three buttons. Each button is labelled with a users name, see figure 35:

	Name	User Name	User Type
1.	David Monaghan	david.monaghan	user
2.	Brain Gleeson	brian.gleeson	user
3.	Administrator	admin.123	admin

Upon clicking of a button, a user is brought to the overview page for that user, no password is currently required. The overview page will establish a Web Socket with Proxy-Control-Server and once established a list of containers will be displayed for User Type: user and a list of all containers displayed for User Type: Admin, see figures 36. The overview page provides a form with a series of Radio Button options that the user can use to perform an action upon a single container. The author would have liked to have provided this functionality for all containers but due to time constraints and the lack of a requirement, this functionality is to be left for a later development cycle. The overview page uses simple Javascript error checking to prevent a user sending a command that would cause an error on the back-end such as starting a container that is already running, see figure 37. Once the back-end of the service has completed performing the command it will return a status message. The status message will be displayed once it has been received, see figure 38.

The overview page also provides a link to the create_container page. The create container page provides a form that allows a user to create a new container, see figure 39. The form will only submit if all fields have a value provided, see figure 40. The form has the following values:

1. Text Field: Container Name
2. Drop Down Menu: Base Server [Defaults to Ubuntu 14.04, currently only option available]
3. Drop Down Menu: Content Management System [Defaults to Wordpress, currently only option available]
4. Text Field: New Root Database Password
5. Text Field: Database Admin User-name
6. Text Field: Database Admin Password

Once the form has been submitted and sent to the Proxy-Control-Server, a status will be returned and the status of the new container will be displayed along with a link to new Wordpress site, see figure 42.

Figure 34: Index.html

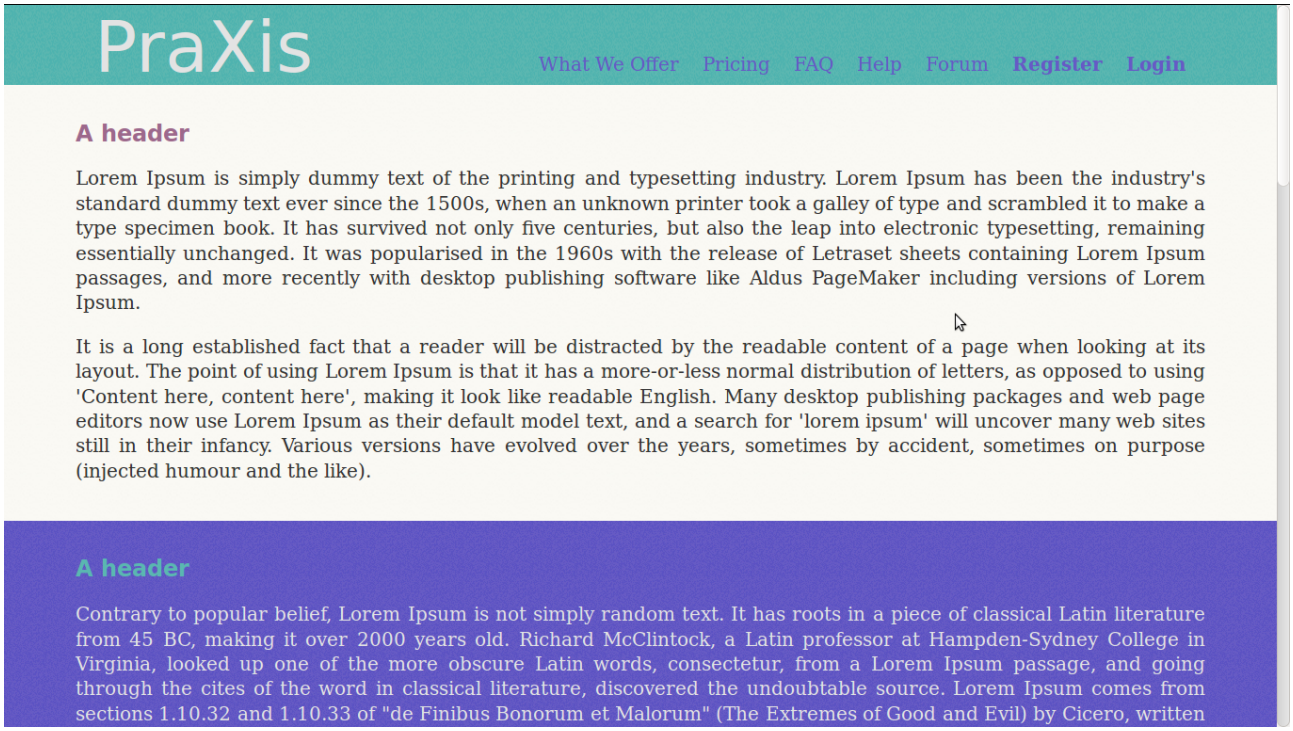


Figure 35: Login.html

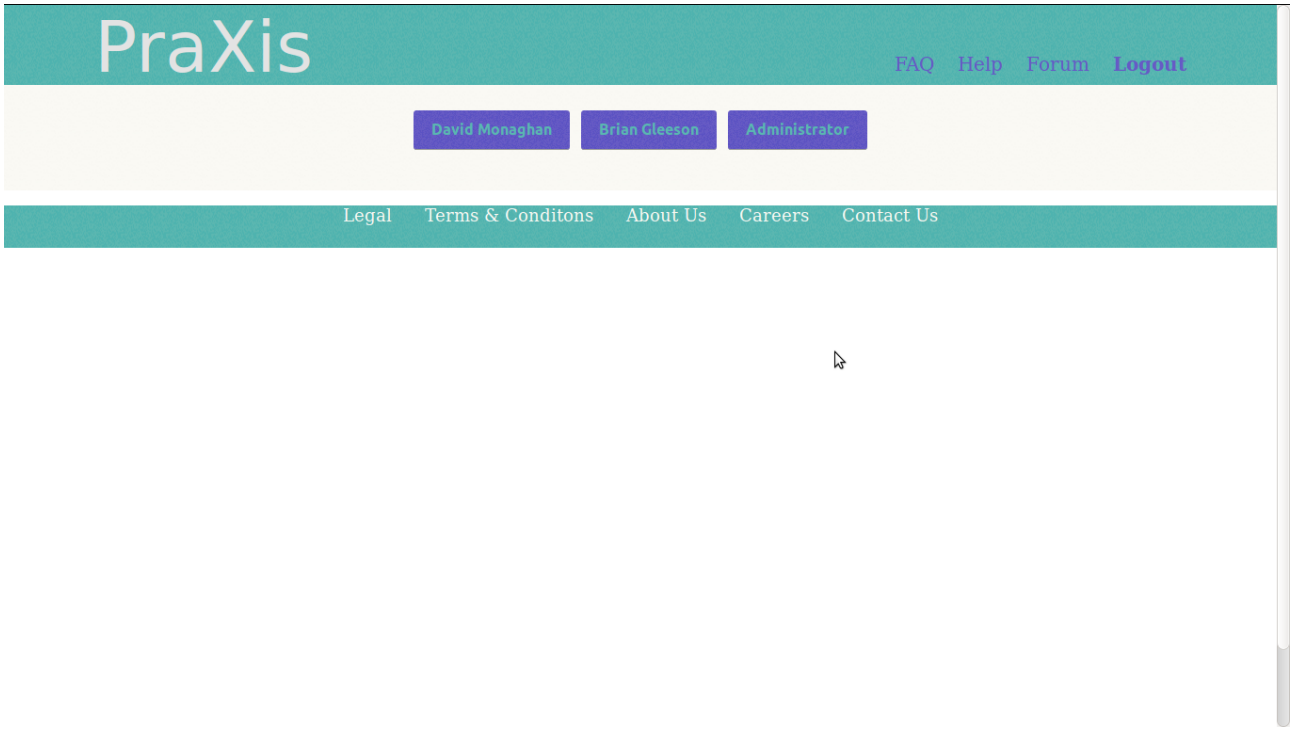


Figure 36: Overview.html for User david.monaghan

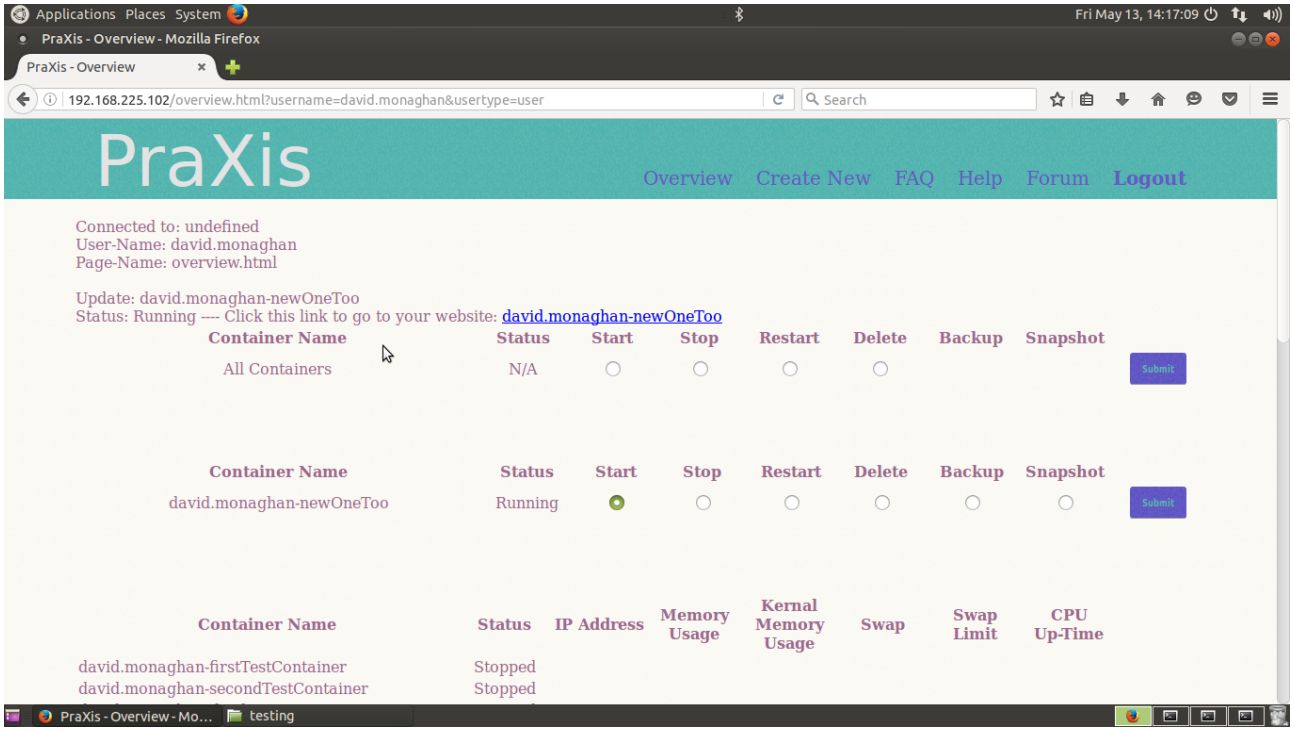


Figure 37: Overview.html showing error checking

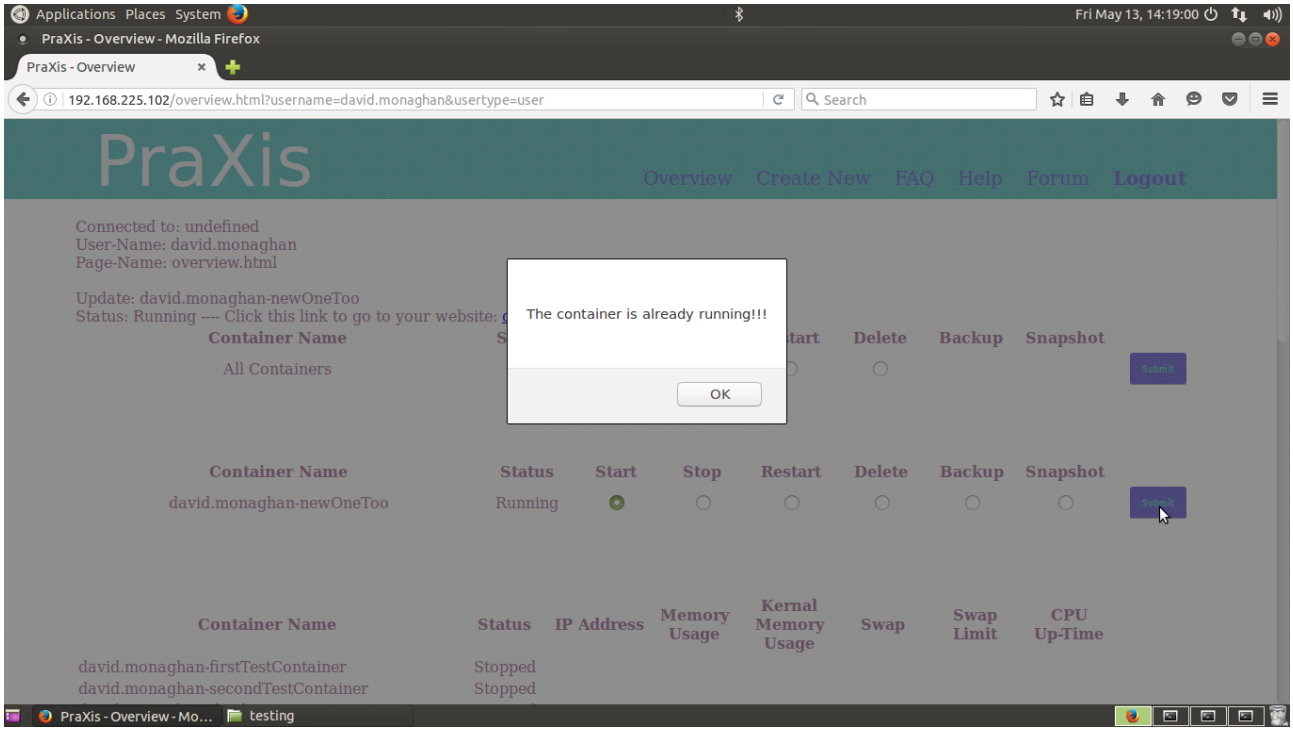


Figure 38: Overview.html showing the status of an action performed on a containers

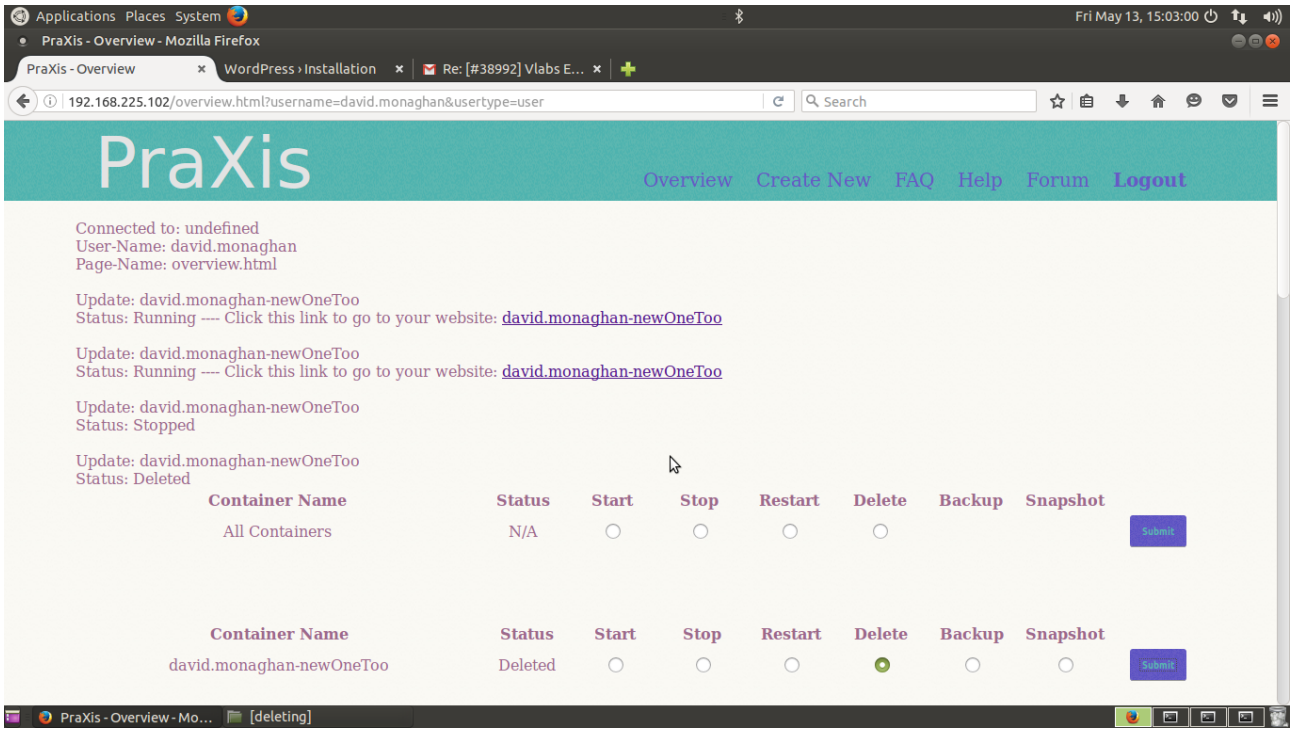


Figure 39: Create_Container.html for User david.monaghan

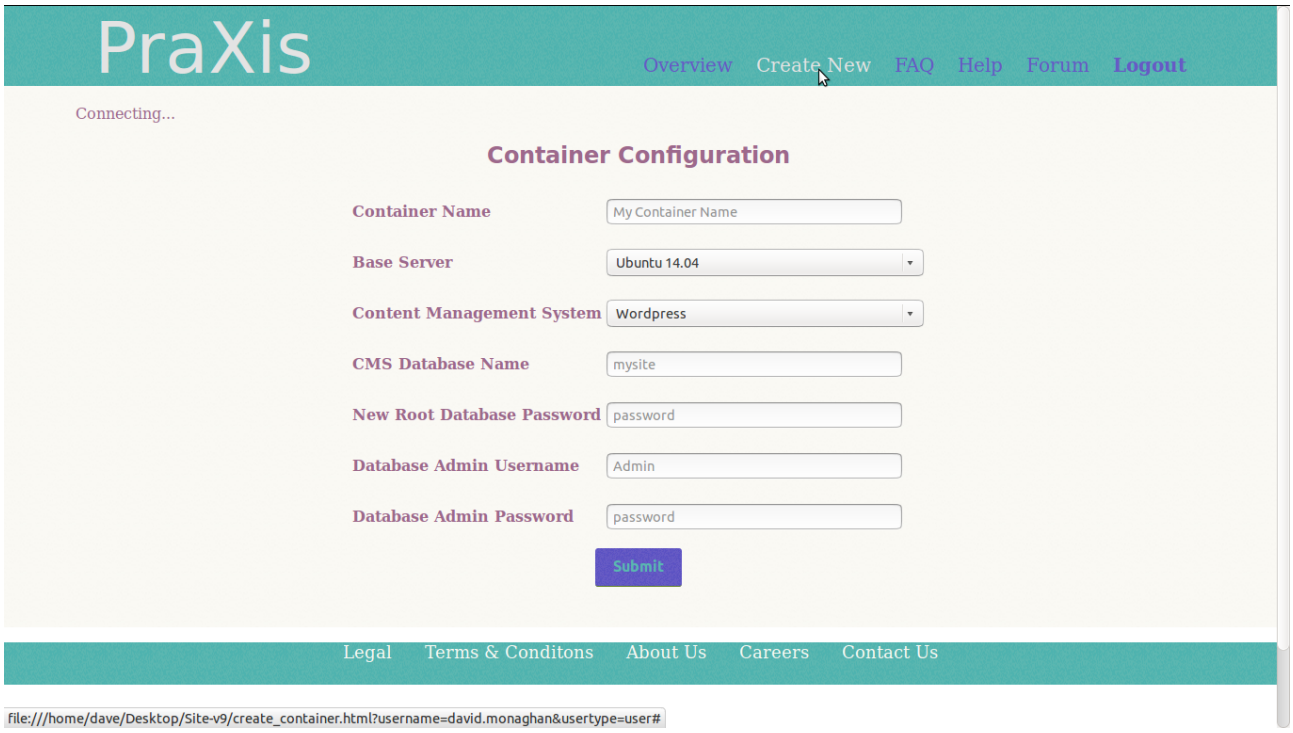


Figure 40: Create_Container.html form error checking

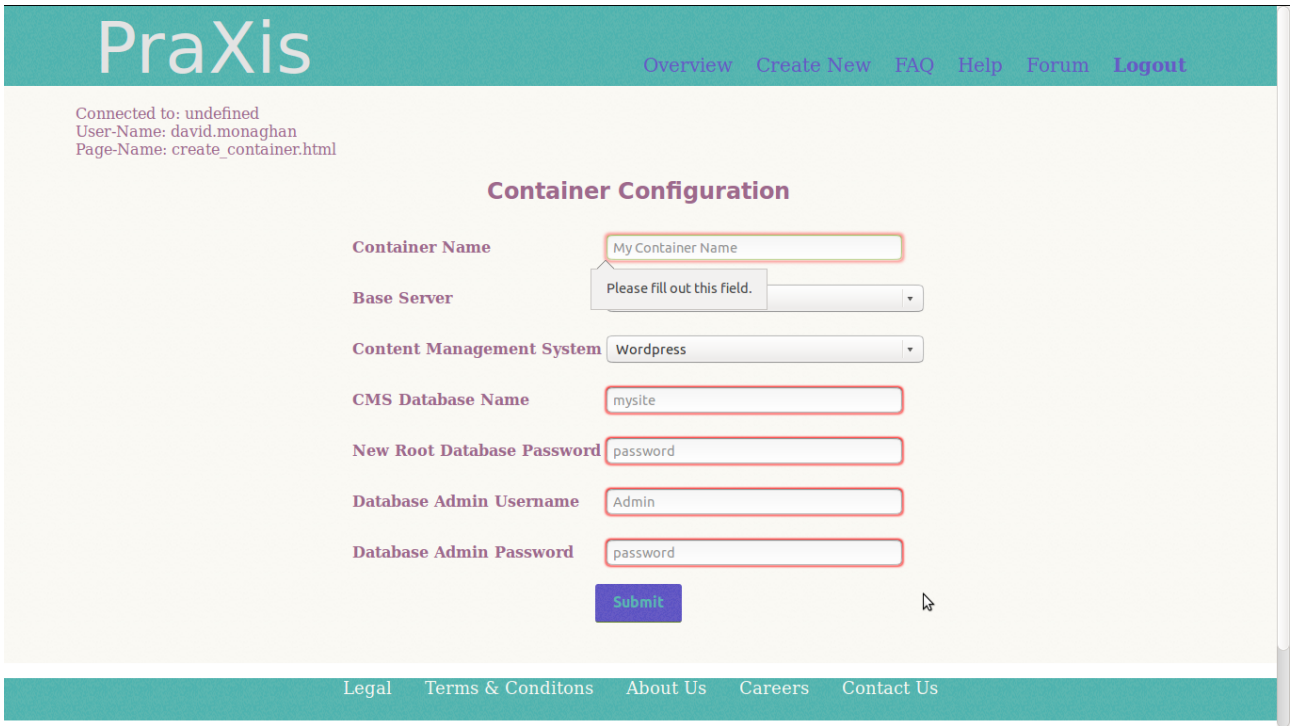


Figure 41: Create_Container.html showing status of a newly created container and it's IP address

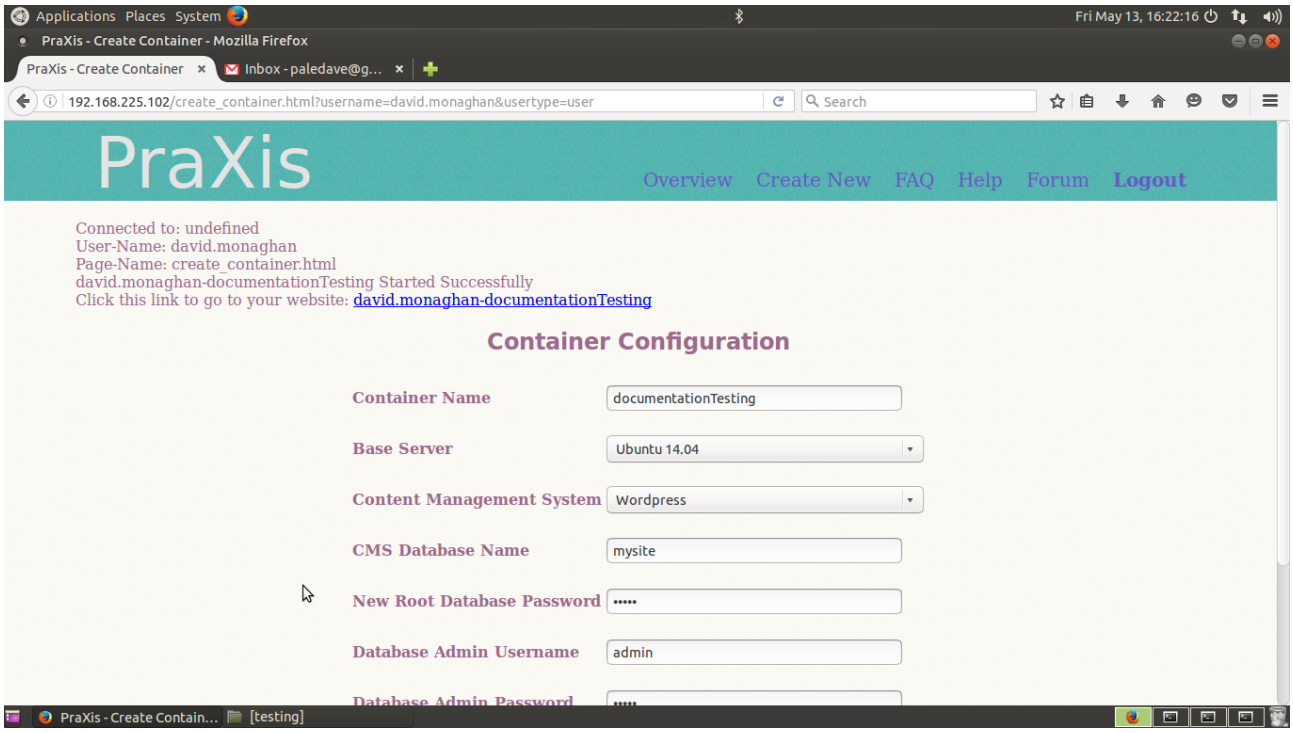
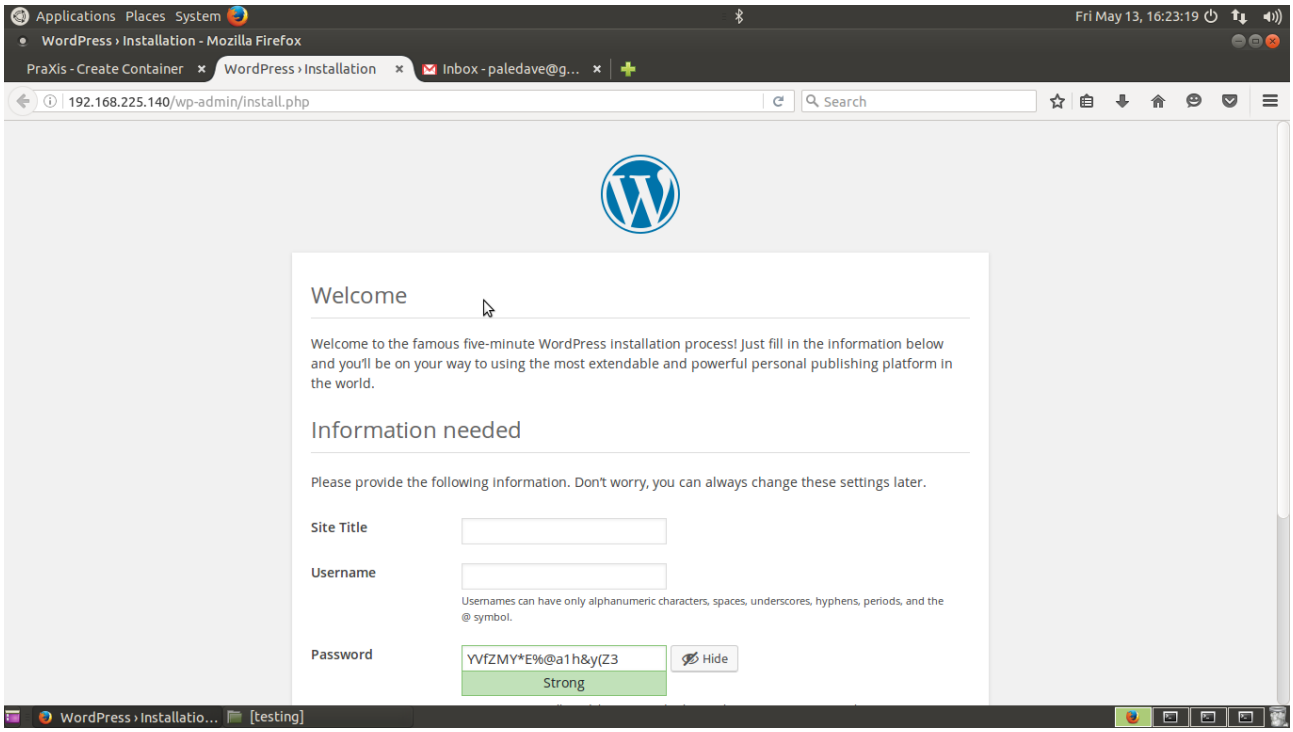


Figure 42: The wordpress installation



5.3 – Establishing a Web-Socket Connection

A web-Socket provides full duplex communication between two hosts using a TCP connection. This connection is maintained until it is broken by either side. Web Sockets is part of the HTML5 standard and as such is supported by all modern browsers. While not implemented during the project, it would be ideal for a web page looking to establish a web-socket to perform a check on the users browser to confirm that it supports the protocol.

The alternative to using a Web Socket is to use RESTful architecture. A RESTful architecture is a stateless one and requires many connections to provide the semblence of a stateful connection. In essence a RESTful API is designed to bypass the limitations of the HTML standard. In previous years a project of this nature would have required this architecture. With the advent of Web-Sockets it is largely becoming unneccessary for new web applications. For that reason and due to the inherant complexity of a RESTful API, the author chose to use Web-Sockets in the implementation phase.

When establishing a Web-Socket, a browser sends a HTTP request and performs a TCP 3 Way Handshake with the server. After the 3 Way Handshake has been completed the Server will upgrade the connection to a Web-Socket and there will be full duplex communication between host and client.

On the client-side, using Javascript, a Web-Socket can be established using the following code:

Figure 43: Javascript for establishing and using a Web-Socket

```
// Create a new WebSocket.
var socket = new WebSocket('ws://192.168.1.101:8080/user');

// Handle any errors that occur.
socket.onerror = function(error) {
        console.log('WebSocket Error: ' + error);
        // Do Something
};

// Show a disconnected message when the WebSocket is closed.
socket.onclose = function(event) {
        // Do Something
};

socket.onopen = function(event) {
        // Do Something
}

// Handle messages sent by the server.
socket.onmessage = function(event) {
        // Do Something
}

// Send the message through the WebSocket.
socket.send(json);
```

On the server side using Go, a Web-Socket can be established using the following code:

Figure 44: Go for establishing and using a Web-Socket

```
// Variable containing IP address and Port number to set Web-Socket Upon
var addr = flag.String("addr", "192.168.1.101:8080" , "http service address") //
"127.0.0.1:8080"

// These are the default values required for a Web-Socket
var upgrader = websocket.Upgrader{
    ReadBufferSize:  1024,
    WriteBufferSize: 1024,
    CheckOrigin: func(r *http.Request) bool {
        return true
    },
}

func main() {
    // HTTP Server listens on Port 80 for URL matching url/user
        // Calls function 'user' for each connection
        http.HandleFunc("/user", user)

        // Returns an error page if URL does not match pattern
        log.Fatal(http.ListenAndServe(*addr, nil))
}

func user(w http.ResponseWriter, r *http.Request) {
        // Sends HTPP request to upgrader and checks values match default
        // Returns an error if not
        c, err := upgrader.Upgrade(w, r, nil)

        if err != nil {
                log.Printf("Websocket upgrade error:", err)
                return
        }
        if err == nil {
                log.Printf("Websocket upgrade completed")
        }

        go handleWebServerConnection(c)
}
```

5.4 – Using JSON In Service Communication

JSON is human readable format for the transmission of data between two. JSON is well established protocol with with wide support in the IT industry, in the project implementations case it is supported by both Javascript and Go.

The project implementation used a single JSON data-object for the purposes of communicating between components of the service as outlined in figure **<insert figure#>**. The data-object element called Action provides the instruction to be performed by a particular component on a container named in ContainerName. Each Action will return a status/ error message in the ContainerStatus section of JSON data-object.

Encoding a JSON using Javascript
In Javascript a JSON structure first needs to be declared:

Figure 45: Javascript JSON declaration

```
var messageJSON = {
        "CustomerType":  userType,
        "CustomerUname": userName,
        "Action":        "createNew",
        "ContainerName":       cName.value,
        "BaseServer":          sName.value,
        "CMS":                 cmsName.value,
        "WebsiteName":         wsName.value,
        "DBrootPWD":           DBrootPWD.value,
        "DBadminUname":  DBadminUname.value,
        "DBadminPWD":          DBadminPWD.value
}
```

And then converted to a JSON data structure

Figure 46: Javascript converting to JSON structure

```
var json = JSON.stringify(messageJSON);
```

Before being encoded and transmitted to the Proxy-Control-Server

Figure 47: Transmitting JSON on a Web-Socket

```
socket.send(json);
```

Decoding a JSON using Javascript
To decode a JSON transmitted on a Web-Socket we first need to set Javacript to listen on the Web-Socket. To use elements of the JSON we just refer to to the ID name of that element.

```
socket.onmessage = function(event) {
      var message = event.data;

      var serverResponse = JSON.parse(message)

      if ( serverResponse.ContainerStatus == "Container Already Exists" ) {
            containerStatus.innerHTML = "Error: Container Already Exists";
      }

}
```

Encoding a JSON using Javascript

Using a JSON in Go is relatively straight forward. Before using JSON the JSON package needs to be imported from the Go repositories, declare the following code before the main function:

Figure 48: Importing the JSON library in Go

```
import {
    "encoding/json"
}
```

Next a data structure that represents the JSON needs to be declared:

Figure 49: Declaring a JSON object in Go

```
type NewContainerJSON struct {
    CustomerType string             `json:"CustomerType,omitempty"`
    CustomerUname string            `json:"CustomerUname,omitempty"`
    Action string                   `json:"Action,omitempty"`
    ContainerName string            `json:"ContainerName,omitempty"`
    BaseServer string               `json:"BaseServer,omitempty"`
    CMS string                      `json:"CMS,omitempty"`
    WebsiteName string              `json:"WebsiteName,omitempty"`
    DBrootPWD string                `json:"DBrootPWD,omitempty"`
    DBadminUname string             `json:"DBadminUname,omitempty"`
    DBadminPWD string               `json:"DBadminPWD,omitempty"`
    ContainerStatus string          `json:"ContainerStatus,omitempty"`
    ContainerIPaddress string       `json:"ContainerIPaddress,omitempty"`
    WordpressStatus string          `json:"WordpressStatus,omitempty"`
}
```

The NewContainerJSON structure will be used to both encode and encode JSON objects. We next declare a variable using the JSON structure outline above:

Figure 50: Declaring a variable using JSON data structure on Go

```
var b NewContainerJSON
```

Once the JSON variable has been declared, we need to listen to Web-Socket, represented by 'ws' shorthand.

Figure 51: Decoding a JSON on a Web-Socket using Go

```
err := ws.ReadJSON(&b)
```

Figure 52: Decoding a JSON on Network Socket using Go

```
decoder := json.NewDecoder(conn)

var b NewContainerJSON
```

To use the JSON data-object elements is outlined in figure 41.

```
if b.Action == "getListOfContainers" {
        // Do something
}
```

Figure 53: JSON data-object

```
CustomerType string             `json:"CustomerType,omitempty"`
CustomerUname string            `json:"CustomerUname,omitempty"`
Action string                   `json:"Action,omitempty"`
ContainerName string            `json:"ContainerName,omitempty"`
BaseServer string               `json:"BaseServer,omitempty"`
CMS string                      `json:"CMS,omitempty"`
WebsiteName string              `json:"WebsiteName,omitempty"`
DBrootPWD string                `json:"DBrootPWD,omitempty"`
DBadminUname string             `json:"DBadminUname,omitempty"`
DBadminPWD string               `json:"DBadminPWD,omitempty"`
ContainerStatus string          `json:"ContainerStatus,omitempty"`
ContainerIPaddress string       `json:"ContainerIPaddress,omitempty"`
WordpressStatus string          `json:"WordpressStatus,omitempty"`
```

- `CustomerType string` `json:"CustomerType,omitempty"`
  - CustomerType: Type of user: *'user'* or *'admin'***.**
  - Data-type is ***'string.'***
- `CustomerUname string` `json:"CustomerUname,omitempty"`
  - *CustomerUname:* Customers user-name**.**
  - Data-type is ***'string.'***
- `Action string` `json:"Action,omitempty"`
  - *Action:* Command to be performed on the container: *'getListOfContainers'*, *'conatinerStart'*, *'containerStop'*, *'containerRestart'*, *'containerDelete'*, *'containerBackup'*, *'containerSnapshot'***.**
  - Data-type is ***'string.'***
- `ContainerName string` `json:"ContainerName,omitempty"`
  - *ContainerName:* Name of container upon which the command is to be performed or for whom the back-end is sending a message
  - Data-type is ***'string.'***
- `BaseServer string` `json:"ContainerName,omitempty"`
  - *BaseServer:* This is the server that makes up the containers operating system, by the default during the implementation only Ubuntu 14.04 will be available.
  - Data-type is ***'string.'***
- `CMS string` `json:"CMS,omitempty"`
  - *CMS:* Content Management System for the management of the system.
  - Data-type is ***'string.'***
- `WebsiteName string` `json:"WebsiteName, omitempty"`
  - *CMS:* Used for configuration of the CMS database and CMS configuration.
  - Data-type is ***'string.'***
- `DBrootPwd string` `json:"DbrootPwd, omitempty"`
  - *DBrootPwd:* Used to change the default Root password of the CMS MySQL database.
  - Data-type is ***'string.'***

- `DBadminUname string` `` `json:"DBadminUname,omitempty"` ``
  - *DBadminUname:* This used to create a user that Wordpress installation can use to modify the web site.
  - Data-type is ***'string.'***
- `DBadminPWD string` `` `json:"DBadminPWD,omitempty"` ``
  - *DBadminPWD:* Password for the Wordpress administrator user.
  - Data-type is ***'string.'***
- `ContainerIPaddress string` `` `json:"ContainerIPaddress,omitempty"` ``
  - *ContainerIPaddress:* This value blank when sent by overview.html but the back-end service will return the IP address if the containers status has been changed to *'Running'*.
  - Data-type is ***'string.'***

5.5 – Establishing a Network Socket In Go

Communication between the back-end components of the service is facilitated by the use of Network Sockets, see figure. Each service component listens on it's own TCP port:

	Server Name	TCP Port
1.	Proxy-Control-Server	Port 8080
2.	Database-Control-Server	Port 8082
3.	LXC-Control-Server	Port8081

At present the project implementation establishes a Network Connection each time it needs to communicate with another component. This system works for the current implementation because the client-side is able to gracefully drop the connection but does run the risk that a system interruption causes the connection to be dropped. In such a scenario the server side of the connection may not release the port on it's side for several minutes, to resolve this requires a full reboot of the effected server.

Ideally one would want to open a connection on boot up of the server and have that connection persist throughout the lifetime of the service. On this persistent connection each time a request or action needs to be taken it would use this socket connection for communication. This was the authors preferred design but was unable to properly implement it. The issue arose due to the complexity of using Go concurrent routines and ensuring communication is synchronised between all users requests. An example of an issue arising might a user who sends two or more almost simultaneous requests for the same container. In the current design this is not an issue as these requests would be handled individually in sequence but in a fully concurrent system, the second request may be actioned before the first. This design is one that would be implemented in a later development cycle.

Establishing a Server Listening on a Port in Go

To create a network connection in Go, the server side needs to have imported the '*net*' library.

Figure 54: Import the 'net' library in Go

```
import (
        "net"
)
```

Figure 55: Establishing a Server listening on a TCP port in Go

```
func main() {
        // This is the IP address and Port number we are listening on
        service := "192.168.1.100:8081"

        // Resolve IP address before listening on port
        tcpAddr := net.ResolveTCPAddr("tcp", service)

        // Listen on Port
        ln net.ListenTCP("tcp", tcpAddr)

        // Handle each incoming connection
        for{
                conn = ln.Accept()
                go handleConnection(conn)
        }
}
```

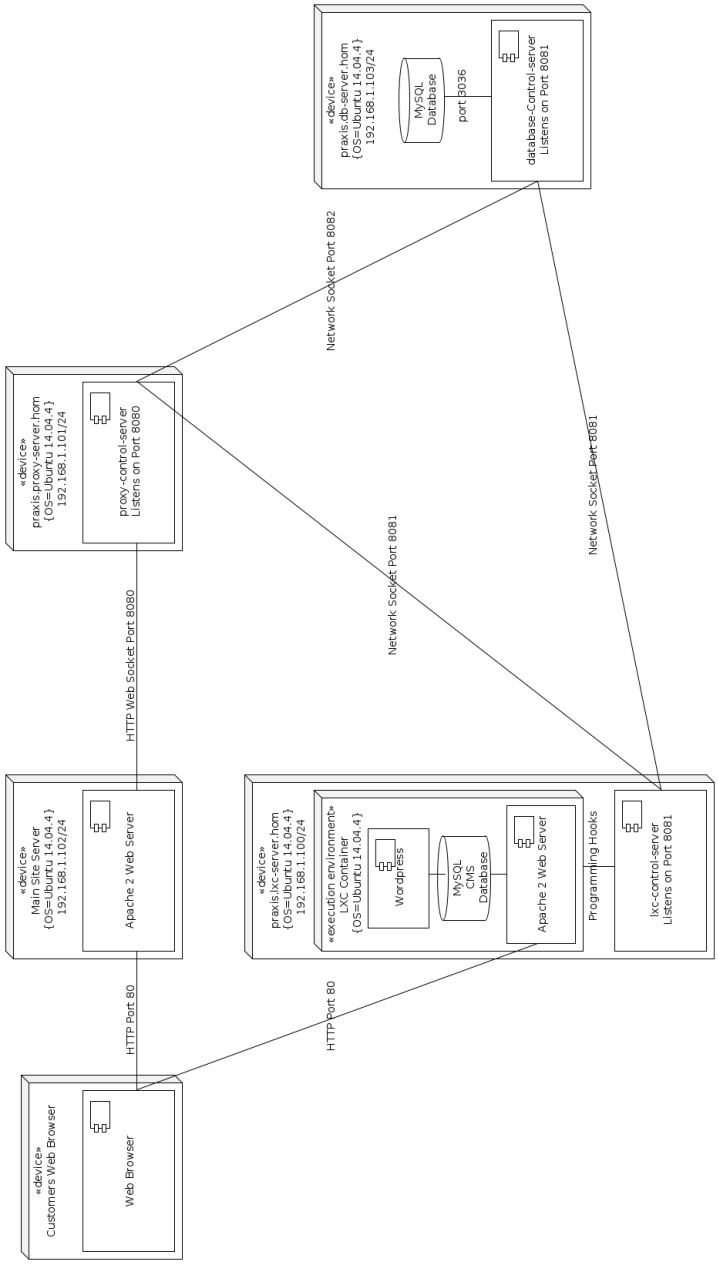
Establishing a Client Dialling a Port in Go

Similarly to the server the client-side needs to import the Go 'net' library.

Figure 56: Client dialling a server in Go

```
// This is IP address and Port number we are dialling
service := "192.168.1.100:8081" //"127.0.0.1:8081"

// Dial the server at IP address and Port number
conn := net.Dial("tcp", service)

// Prevent Network socket from closing automatically
defer conn.Close()
```

Figure 57: Network Diagram for the Project Implementation



5.6 – Interacting with the Service Database

The initial research and design phases of the project did not foresee the need of a separate service to manage interactions with the database. This lack of foresight was to prove costly and led to an overrun of one week for the project as this aspect of the project was to be nearly as large as the Proxy-Control-Server and the LXC-Control-Server combined.

The Database-Control-Server primary responsibility is to receive requests and validate those requests such as when a customer requests to start a Container, it will ensure that the container is not already running or when a customer requests to create a container that one of that name does not already exist.

Another function of the database is to synchronise the status of existing containers between it and the LXC-Control-Server upon boot up and ensure that customer is receiving the correct statuses of their containers. Ideally this is a function that would happen upon boot up of the LXC-Control-Server but there was no time within the project time constraints to develop this functionality. This functionality would be especially useful if there was an interruption to the service due to a fault or dropped Network Socket.

Establishing a connection with MySQL Database Server
In the project implementation two Go libraries where used to manage interaction with the database. The ‘database/sql’ library is the standard Go library for interacting with a database and creating SQL queries. The ‘github.com/go-sql-driver/mysql’ library allows one to use standard SQL queries which it will format for use with ‘database/sql’, this is useful for later development if we wish to change to another database as the queries in this round of development would not need to be changed.

Figure 58: Importing the Go Database Libraries

```
import (
        "database/sql"
        _ "github.com/go-sql-driver/mysql"
)
```

Next it is neccessary to establish a connection with the MySQL server. Before doing this one will need to ascertain which port the MySQL server is listening upon. In our case the port is 3306

Figure 59: Connecting to a MySQL database in Go

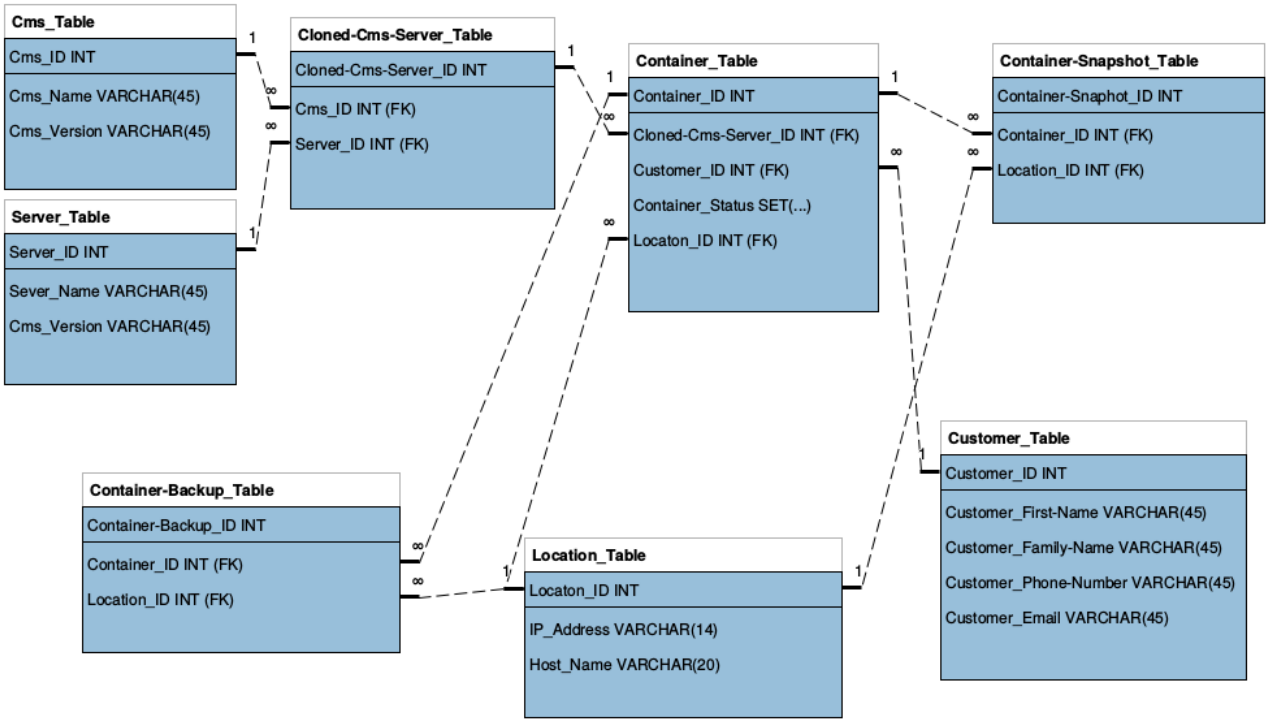
```
func main() {
        // These are IP address, port and login details required for login
        db := sql.Open("mysql","dave:admin@tcp(127.0.0.1:3306)/praxis")

        // Prevent the database from closing automatically
        defer db.Close()

        // Ping the database to confirm it is accepting connections
        db.Ping()
}
```

Figure 60: Typical example of SQL query in Go

Notice the (?) in the SQL query, we can use variable here, in this case cName, declared at end of the query.

```
var qryResult string

rows := db.Query("SELECT Container_Snapshot_Table.Container_Snapshot_Name FROM Container_Snapshot_Table INNER JOIN Container_Table ON Container_Snapshot_Table.Container_ID=Container_Table.Container_ID WHERE Container_Table.Container_Name=(?);", cName)

defer rows.Close()

defer rows.Close()

for rows.Next() {
        rows.Scan(&qryResult)

        log.Println("This is the query result", qryResult)
}
```

Figure 61: Database Schema Design



5.7 – Concurrency in Go

Concurrency is similar to parallel computing but is distinguished by the fact that it is not necessarily run in parallel each time or even most times. This is desirable feature for the service that is being developed for this project as we may have many users making simultaneous but different requests. In order to achieve a concurrent service, concurrency needs to to introduced as early as possible in the run-time of each service component.

For the implementation, concurrency was introduced after each connection has been successfully established by each server. Limitations in the ability of the service to provide concurrent functionality depends entirely upon the servers speed at accepting a connection and passing to the concurrent routine. This largely an artifice of the hardware and OS specifications. If the service where to go into production this element of the service would need to stress tested so as to ensure effective load balancing could be maintained.

Establishing a concurrent sub-routine is amazingly simple using Go. Using the keyword ‘go’ before a subroutine call is enough to create a concurrent sub routine, called a Go Routine. Care must be taken to ensure that concurrent sub routine calls do not ultimately crash the service. As such it is advised not use Go Routines for all sub routine calls and to only to use them where concurrency would benefit the program.

Figure 62: Establishing a Concurrent Sub Routine using Go

```
func main() {
        // This is the IP address and Port number we are listening on
        service := "192.168.1.100:8081"

        // Resolve IP address before listening on port
        tcpAddr := net.ResolveTCPAddr("tcp", service)

        // Listen on Port
        ln net.ListenTCP("tcp", tcpAddr)

        // Handle each incoming connection
        for{
                conn = ln.Accept()
               //  For  each  connection  accepted  start  a  concurrent
             //  subroutine
                go handleConnection(conn)
        }
}
```

5.8 – Error Checking in Go

In a project such as this one, error checking is of primary importance due to the independent nature of the service components involved. Previous code snippets have foregone error checking in aid of clarity and simplicity. The implementation of the project uses error checking at every stage with subsequent action to be taken in case an error occurring. Every attempt has been made to ensure that an error does not lead to a crash of any service, although the author concedes that fully error checking and testing the system in such a short period of time is nearly impossible. With that stated, all errors that cropped during the testing phase of the project have logic built into the system to deal with said error. An area that requires attention is the recovery of the system in the event of a component(s) crashing unexpectedly regardless of the reason for said crash. The example outlined in figure **<Insert figure#>** is the establishment of Network Socket by a server.

Figure 63: Error Checking Example during Establishment of a Network Socket

```
func main() {
        log.Println("LXC-Control-Sever is starting up")
        service := "192.168.1.100:8081"

        tcpAddr, err := net.ResolveTCPAddr("tcp", service)
        // If we fail to resolve IP/Port address
        if err != nil {
                log.Println("Failed to Resolve TCP Address: %s", err)
                log.Println("LXC-Server is down")
        } else {
                log.Println("Resolved TCP Address: ", service)

                // If we fail to listen on the designated TCP port
                ln, err := net.ListenTCP("tcp", tcpAddr)
                if err != nil {
                        log.Println("Failed to listen: %s", err)
                        log.Println("LXC-Server is down")
                } else {
                        log.Println("LXC-Server is up")
                        log.Println("Listening on: ", tcpAddr)

                        for{
// If there is no error accepting incoming connection, send to concurrent Go
// Routine
                                if conn, err := ln.Accept(); err == nil {
                                        go handleConnection(conn)
                                }
                        }
                }
        }
}
```

The code above in figure figure 62 is idiomatic of all error checking throughout the implementation of the project. If one was to compare this code snipped against the one provided in **<Insert figure#>** one would notice that the error checking routines more than double the code required.

6 – Project Evaluation

6.1 – Requirements Achieved/ Not Achieved

Table 4: System/ Application Requirements

#	The System Shall...	Type	Fit Criterion	P	Achieved Y/N
1	Not depend on non-free and/ or proprietary products	Constraint	No features or components to depend on any proprietary products or to have any licensing costs	1	Yes
2	Support Ubuntu 14.04 +	Constraint	Any products used in the implementation stage must support Ubuntu14.04 or newer	1	Yes
3	Services is built securely	Constraint	Each application component configured to best security practices	1	Yes

Table 5: Customer Requirements

#	The System Shall...	Type	Fit Criterion	P	Achieved Y/N
1	Web browser interface to manage customers web-site	Functional		1	Yes
2	Web-site is HTML 5 compliant	Constraint	This gives best out of box cross-platform support	1	Yes
3	Web-site responsive to form factor	Constraint	This gives best out of box cross-platform support	1	No
4	Ability to create web-server via web browser interface	Functional		1	Yes
6	Ability to create CMS via web browser interface	Functional		1	Yes
7	Ability to configure CMS via web browser interface	Functional		1	Yes
8	Ability to stop customer web-site via web browser interface	Functional		2	Yes

Table 6: Customer Requirements (Continued)

#	The System Shall...	Type	Fit Criterion	P	Achieved Y/N
9	Ability to start customer web-site via web browser interface	Functional		2	Yes
10	Ability to restart customer web-site via web browser interface	Functional		2	Yes
11	Ability to create a back-up of customer web-site via web browser interface	Functional		3	Yes
12	Ability to restore customer web-site from back-up via web browser interface	Functional		3	No
13	Ability to create a snap-shot of customer web-site via web browser interface	Functional		3	Yes
14	Ability to restore customer web-site from snap-shot via web browser interface	Functional		3	No

Table 7: Administrator Requirements

#	The System Shall...	Type	Fit Criterion	P	Achieved Y/N
1	Web browser interface to manage customers web-site	Functional		1	Yes
2	Ability to create web-server via web browser interface	Functional		1	Yes
4	Ability to create CMS via web browser interface	Functional		1	Yes
5	Ability to configure CMS via web browser interface	Functional		1	Yes
6	Ability to stop customer web-site via web browser interface	Functional		2	Yes
7	Ability to start customer web-site via web browser interface	Functional		2	Yes
8	Ability to restart customer web-site via web browser interface	Functional		2	Yes
9	Ability to create a back-up of customer web-site via web browser interface	Functional		3	Yes
10	Ability to restore customer web-site from back-up via web browser interface	Functional		3	No
11	Ability to create a snap-shot of customer web-site via web browser interface	Functional		3	Yes
12	Ability to restore customer web-site from snap-shot via web browser interface	Functional		3	No
13	Ability to delete customer web-site from snap-shot via web browser interface	Functional		2	Yes
14	Command line interface to manage customer web-sites	Functional		1	Yes
15	Ability to create web-server via command line interface	Functional		1	Yes

Table 7: Administrator Requirements (Continued)

#	The System Shall...	Type	Fit Criterion	P	Achieved Y/N
16	Ability to configure web-server via command line interface	Functional		1	Yes
17	Ability to create CMS via command line interface	Functional		1	Yes
18	Ability to configure CMS via command line interface	Functional		1	Yes
19	Ability to stop customer web-site via command line interface	Functional		1	Yes
20	Ability to start customer web-site via command line interface	Functional		1	Yes
21	Ability to restart customer web-site via command line interface	Functional		1	Yes
22	Ability to create a back-up of customer web-site via command line	Functional		1	Yes
23	Ability to restore customer web-site from back-up via command line	Functional		1	No
24	Ability to create a snap-shot of customer web-site via command line	Functional		1	Yes
25	Ability to restore customer web-site from snap-shot via command line	Functional		1	No
26	Ability to delete customer web-site from snap-shot via command line interface	Functional		1	Yes
28	Ability to add new web-server options via command line interface	Functional		1	Yes
30	Ability to add new CMS options via command line interface	Functional		1	Yes

6.2 – Testing

6.2.1 – Servers Down

The following tests show the functionality of the back-end servers when one or more of those servers is down. The intention is to avoid the servers that remain running from crashing as well.

When the Database-Control-Server is down, no updates can take place as such the Proxy-Control-Server will return an error message to the Web-Front-End advising as much. This prevents the system from creating or changing status of a container without updating the Database.

When the Proxy-Control-Server is down the Web-Front-End will not be able to establish a Web-Socket and in effect will not be able to communicate with either the Database-Control-Server or the LXC-Control-Server. This will prevent any adverse changes being made to the system.

When the LXC-Control-Server is down, the Proxy-Control-Server will return an error message to the Web-Front-End and prevent any actions being taken with the Database-Control-Server.

6.2.1.1 – Database-Control-Server is Down

Figure 64: Error checking in effect on Web-Site when Database-Control-Server is down

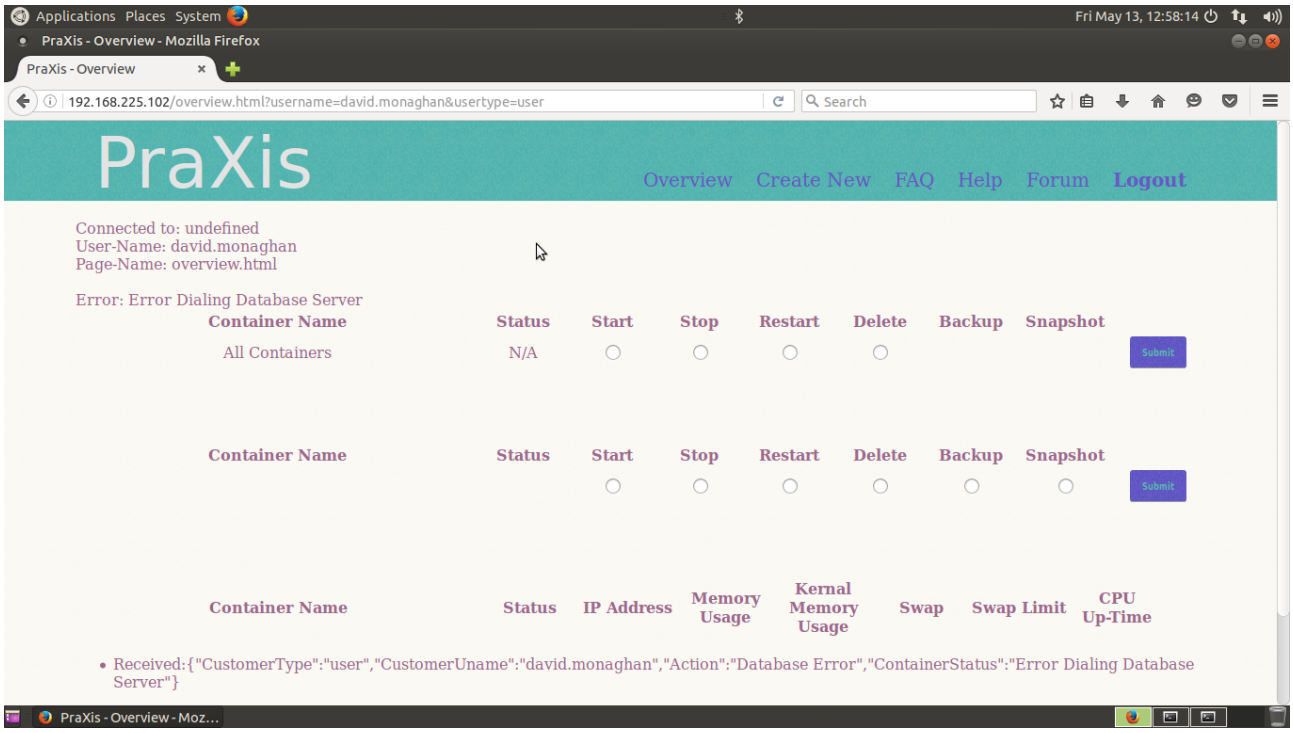


Figure 65: Error checking in effect on Proxy-Control-Server when Database-Control-Server is down

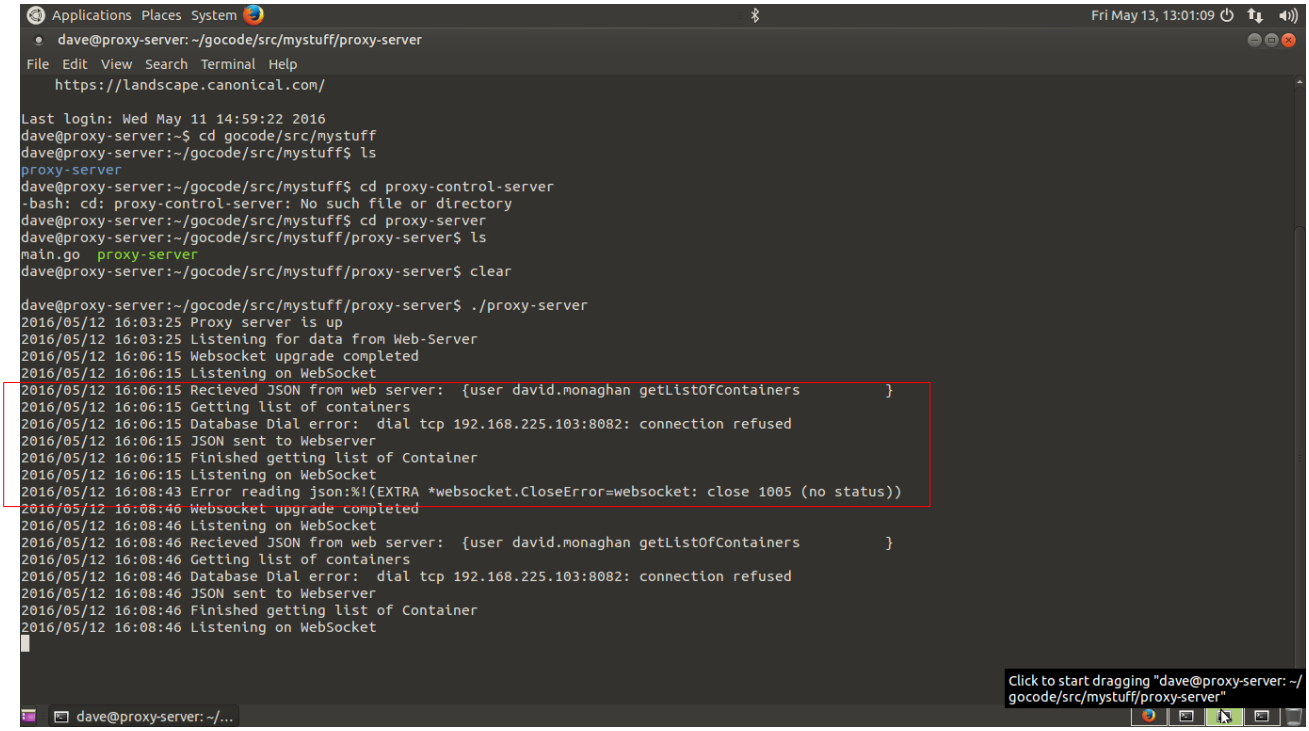
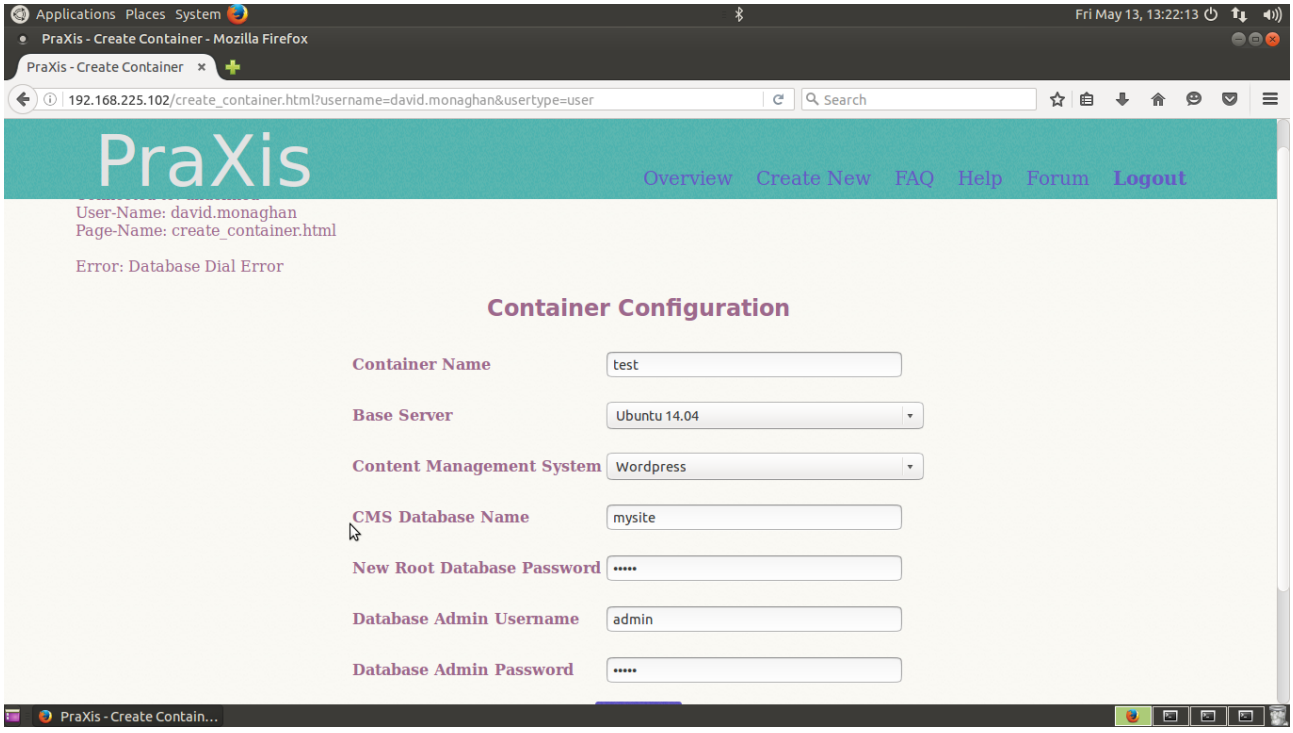


Figure 66: Error checking in effect on Web-Site when Database-Control-Server is down and trying to create a container



6.2.1.2 – Proxy-Control-Server is Down

Figure 67: Error checking in effect on Web-Site when Proxy-Control-Server is down

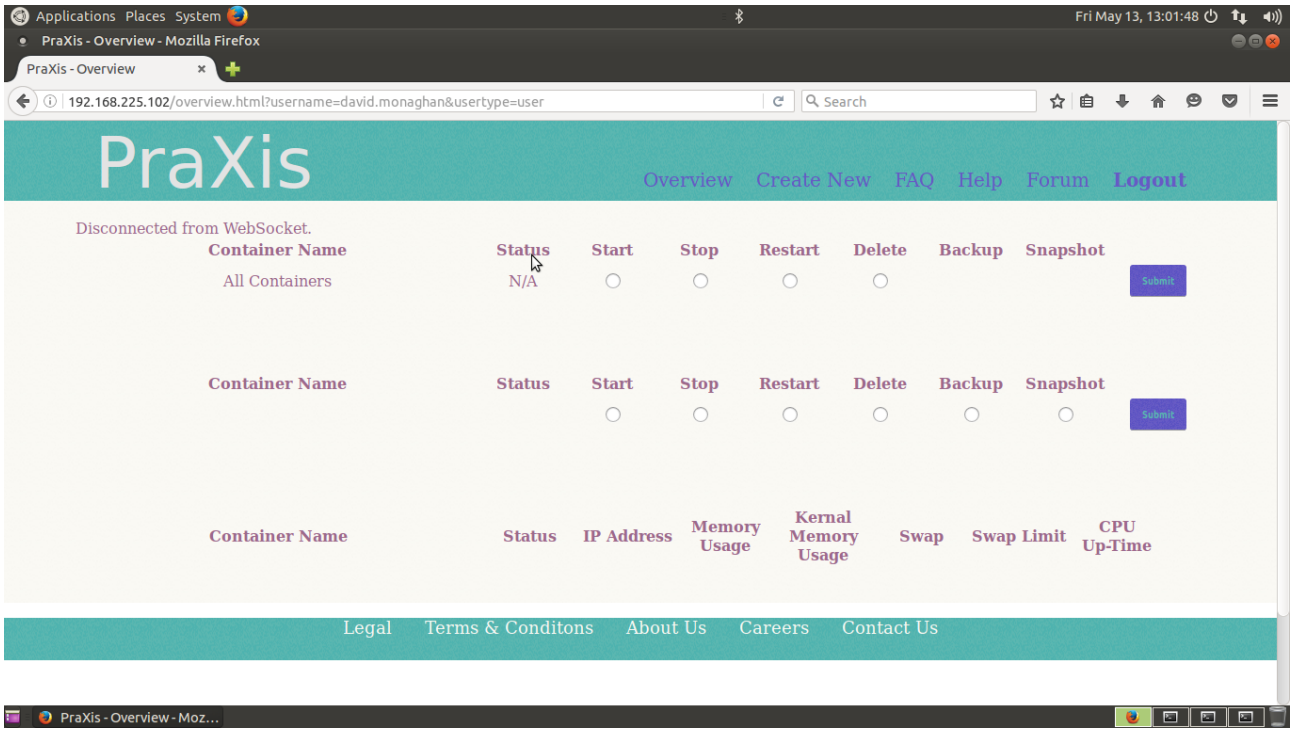
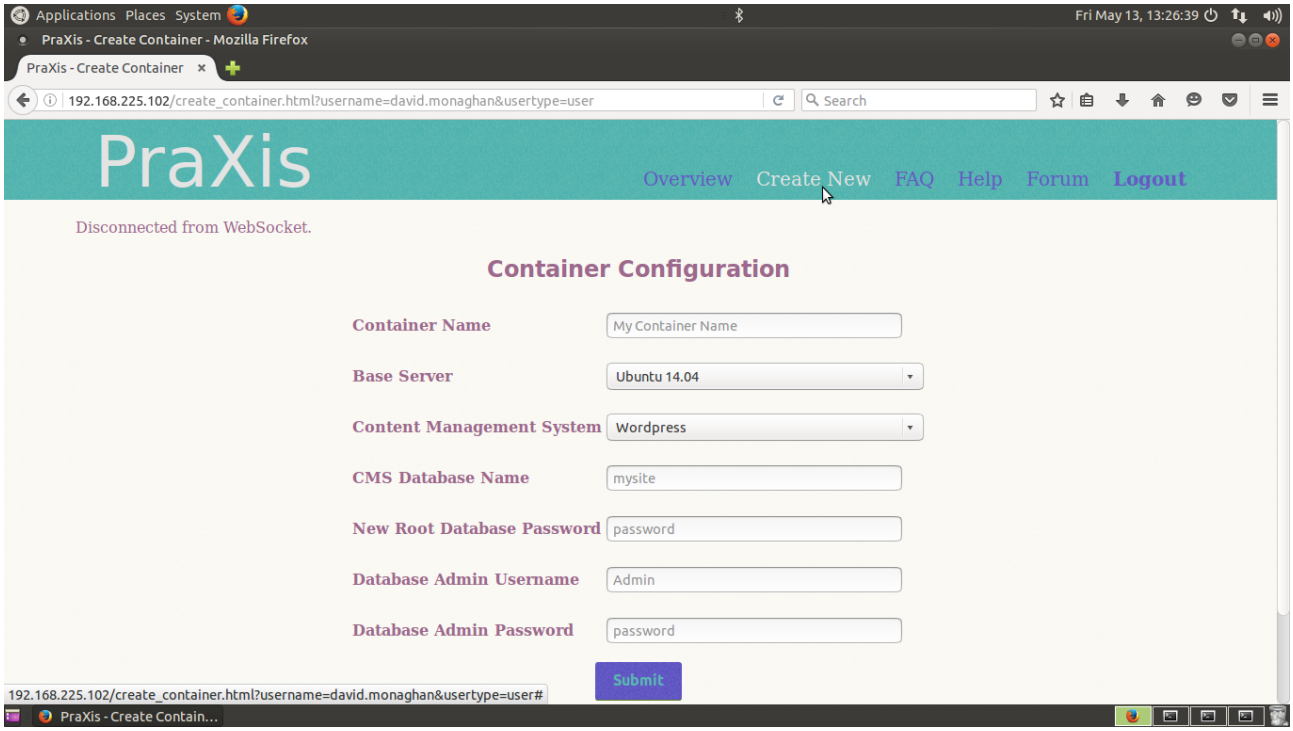


Figure 68: Error checking in effect on Web-Site when Proxy-Control-Server is down



6.2.1.3 – LXC-Control-Server is Down

Figure 69: Error checking in effect on Database-Control-Server when LXC-Control-Server is down

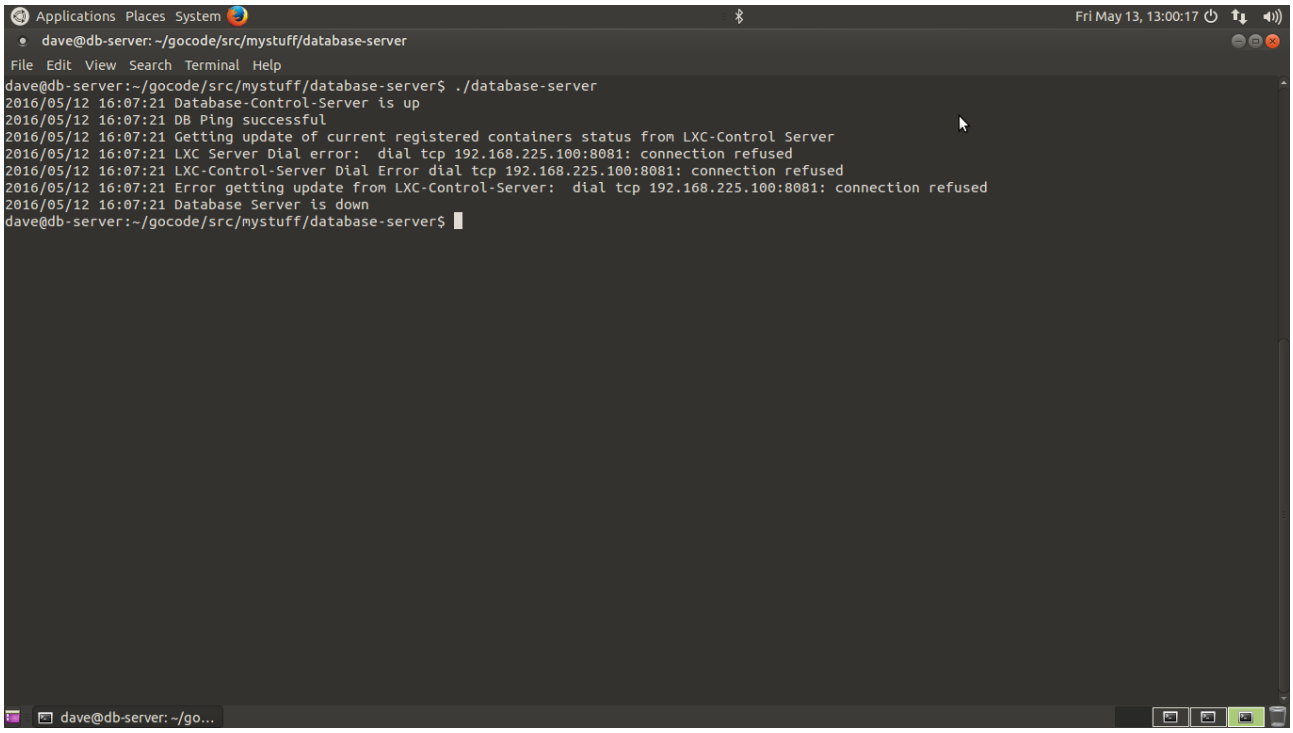


Figure 70: Error checking in effect on Proxy-Control-Server when LXC-Control-Server is down after overview.html and create_container.html send requests

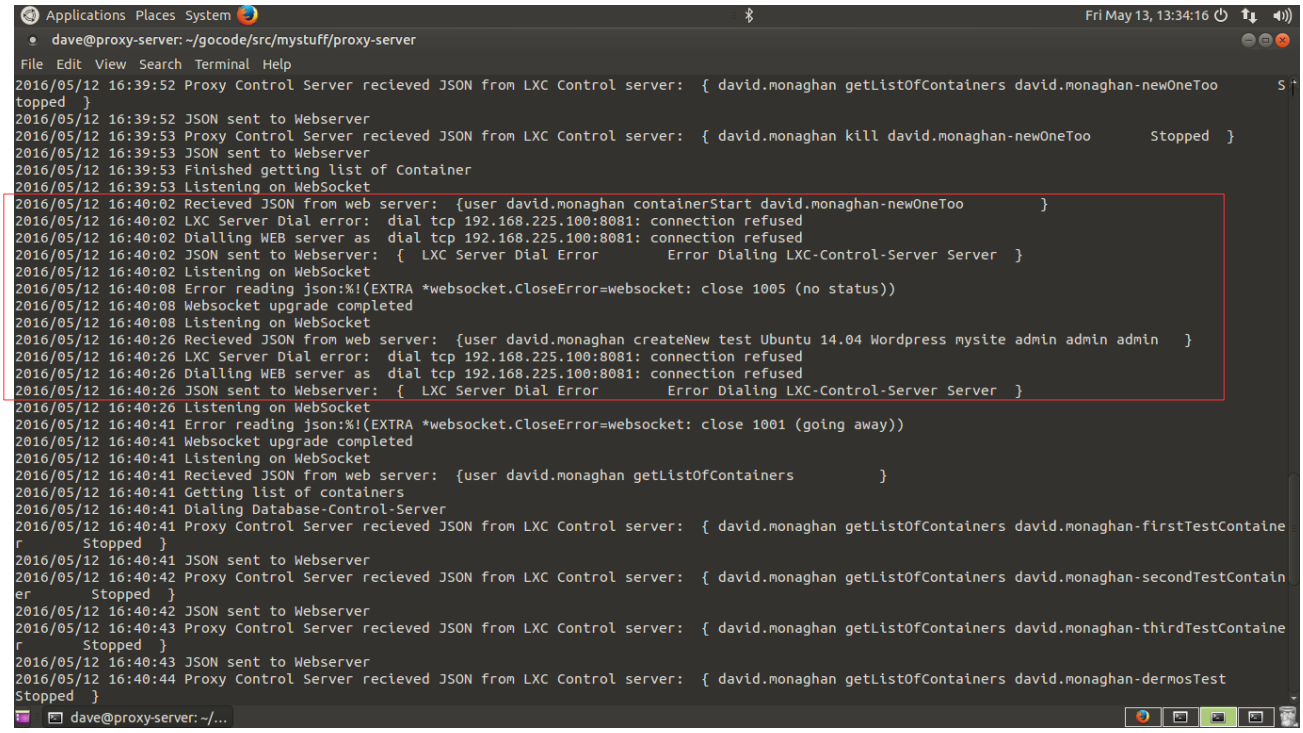


Figure 71: Error checking in effect on Web-Site when LXC-Control-Server is down

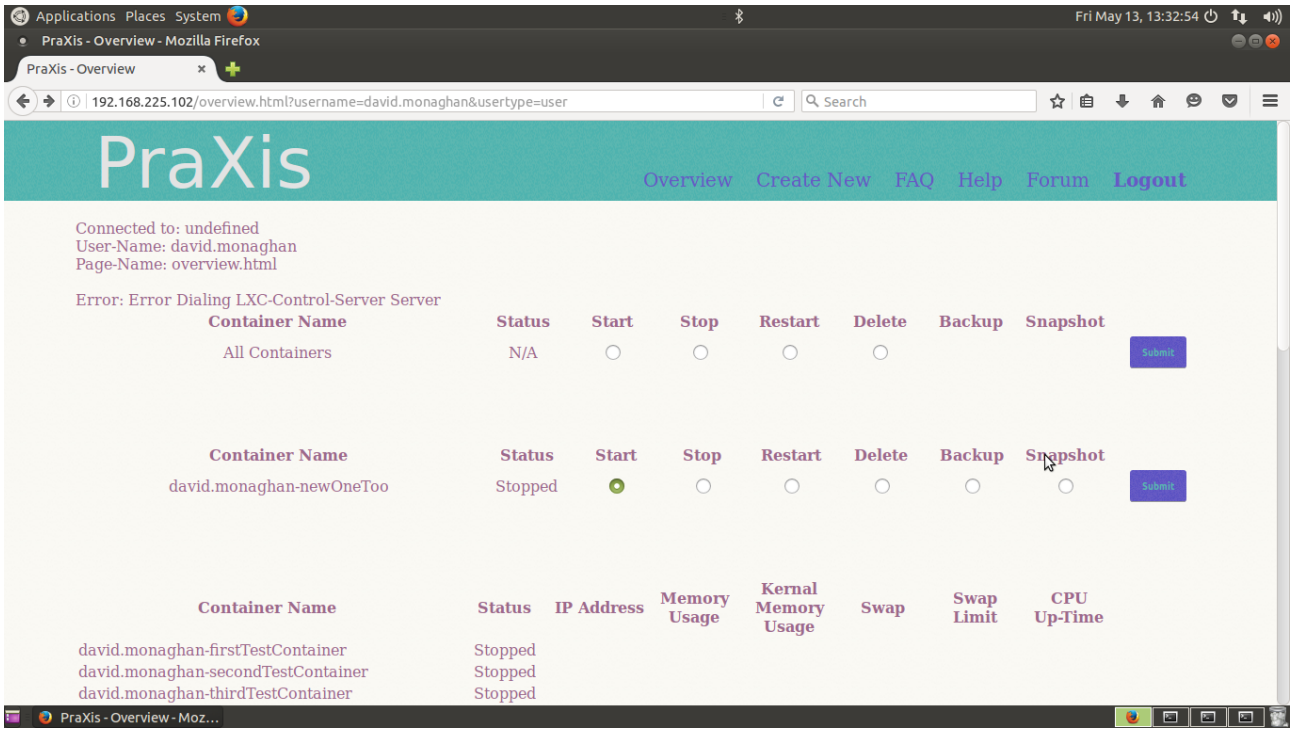
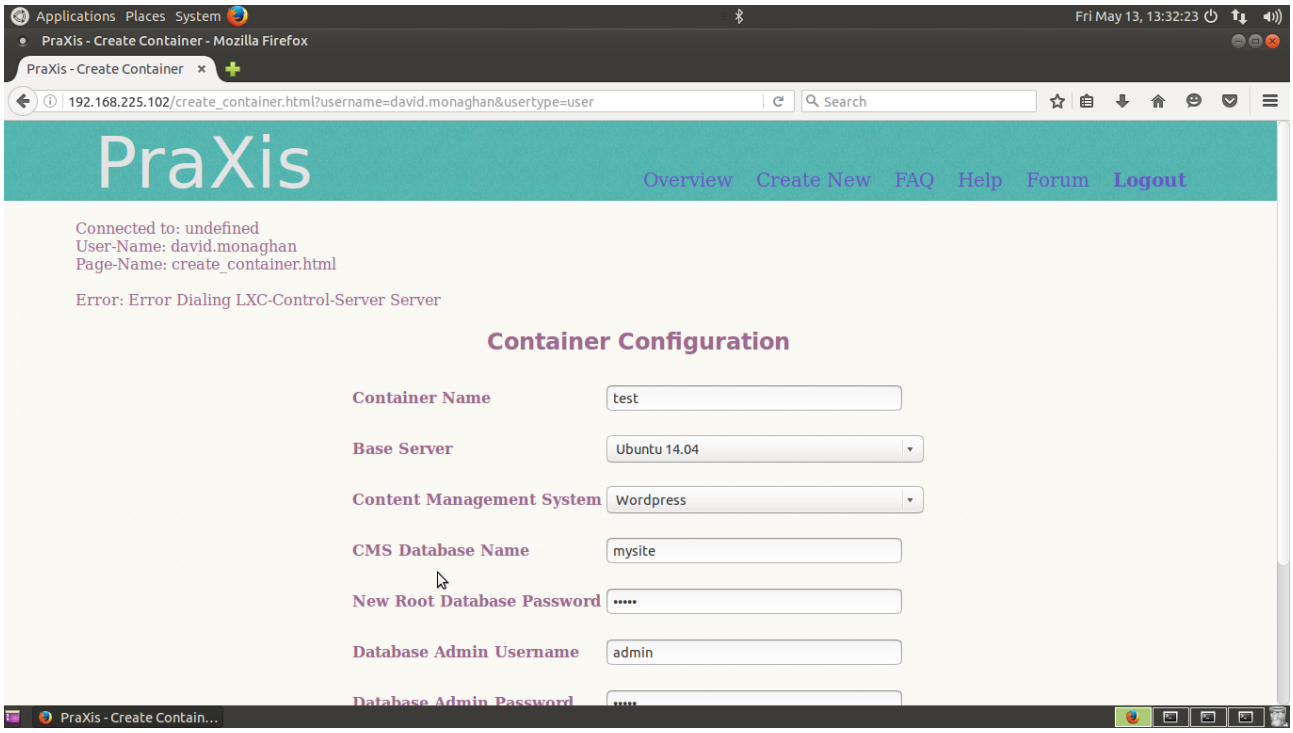


Figure 72: Error checking in effect on Web-Site when LXC-Control-Server is down



6.2.2 – Servers Are Up

Each server will perform a series of checks on their respective listening ports. In the case where a successful connection is made, a message will be logged. Where a server is unable to listen on it's port, the program will shut-down and hence the server as well. The author was unable to determine a way of demonstrating this functionality but believes that the demonstration of what happens when a server is unable to communicate with another server adequately shows that system would be able to handle this exception.

When the Database-Control-Server boots up it will establish a connection with the LXC-Control-Server and they will synchronise the status of containers that are registered in the Database. Where the status of a container cannot be ascertained, the LXC-Control-Server will return a status of 'Suspended'

6.2.2.1 – Proxy-Control-Server is Up

Figure 73: Proxy-Control-Server is up via CLI

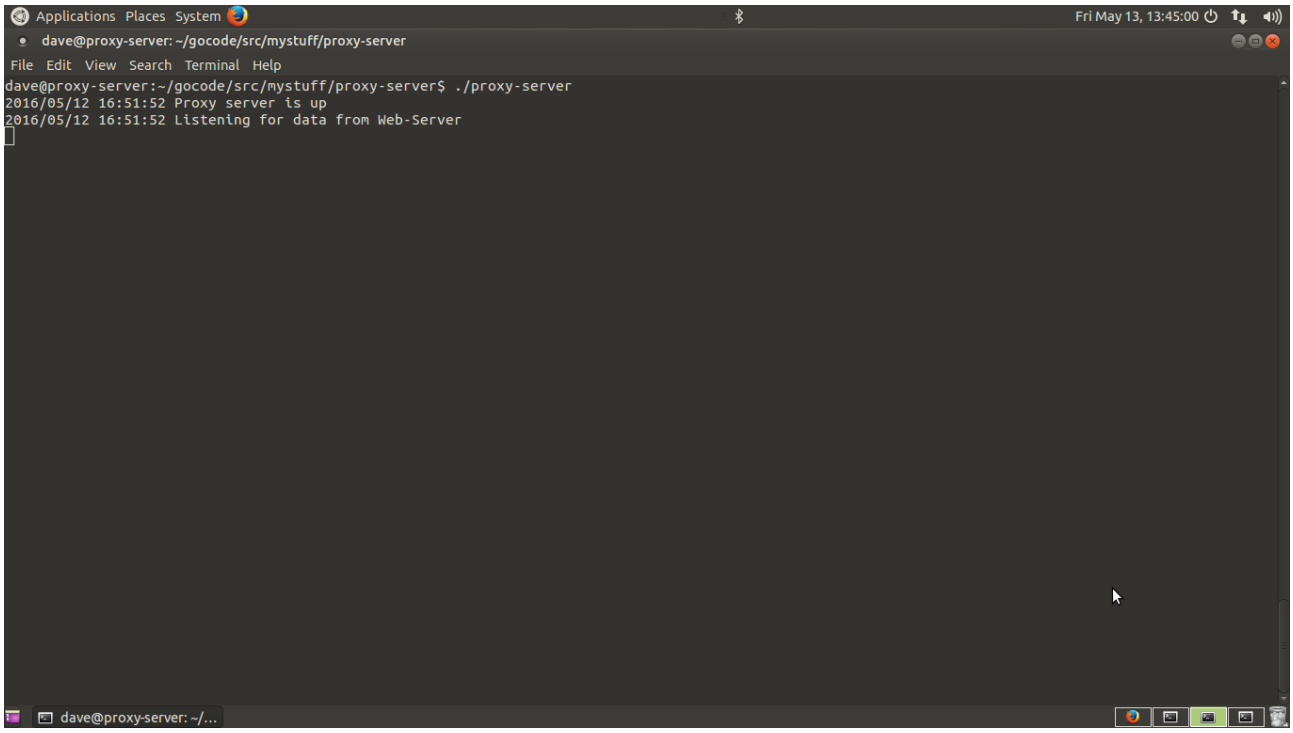


Figure 74: Proxy-Control-Server is up via Overview.html

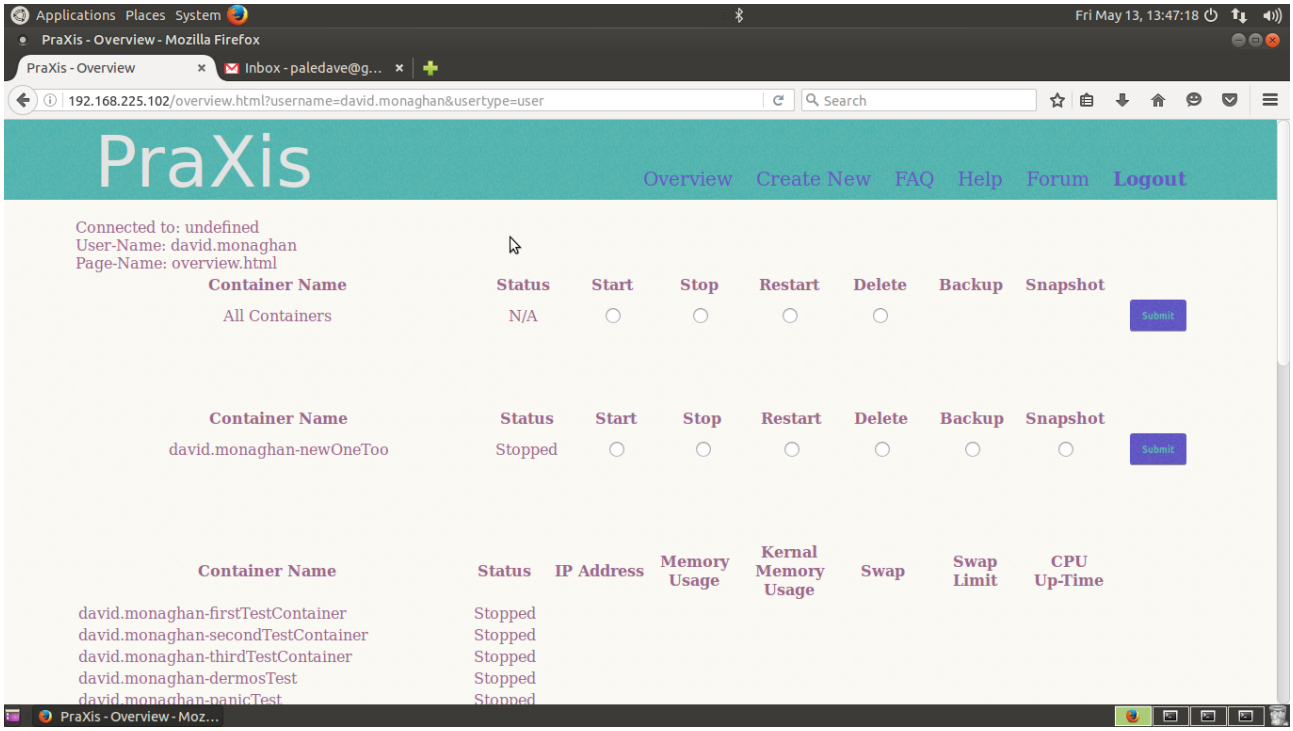
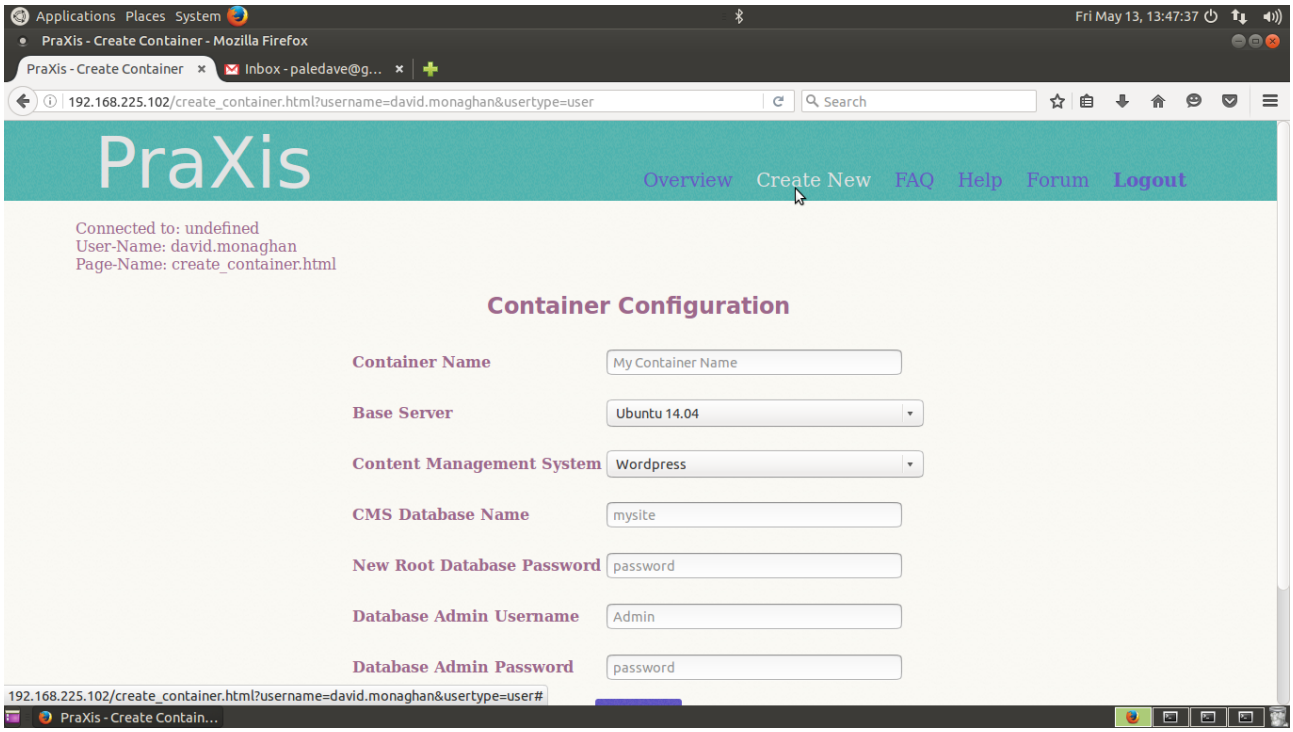


Figure 75: Proxy-Control-Server is up via Create_new.html



6.2.2.2 – LXC-Control-Server is Up

Figure 76: LXC-Control-Server is up prior to syncing with Database-Control-Server

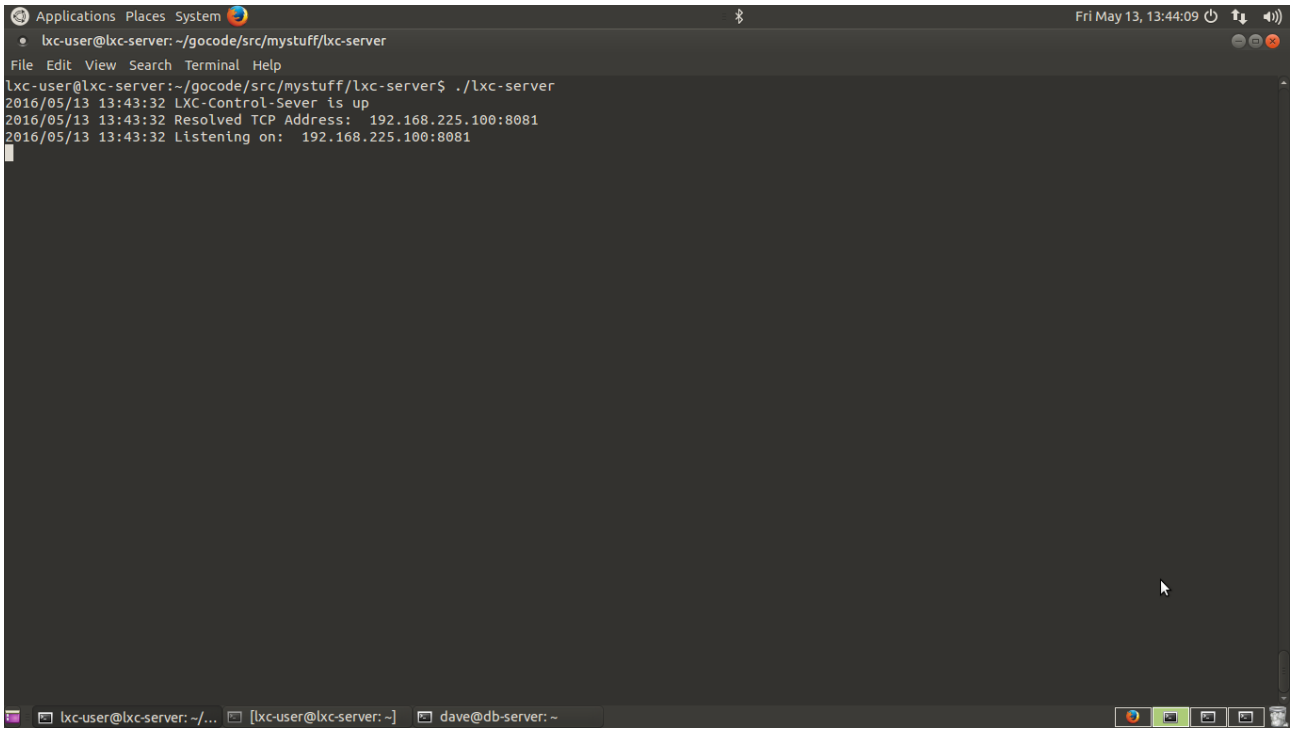
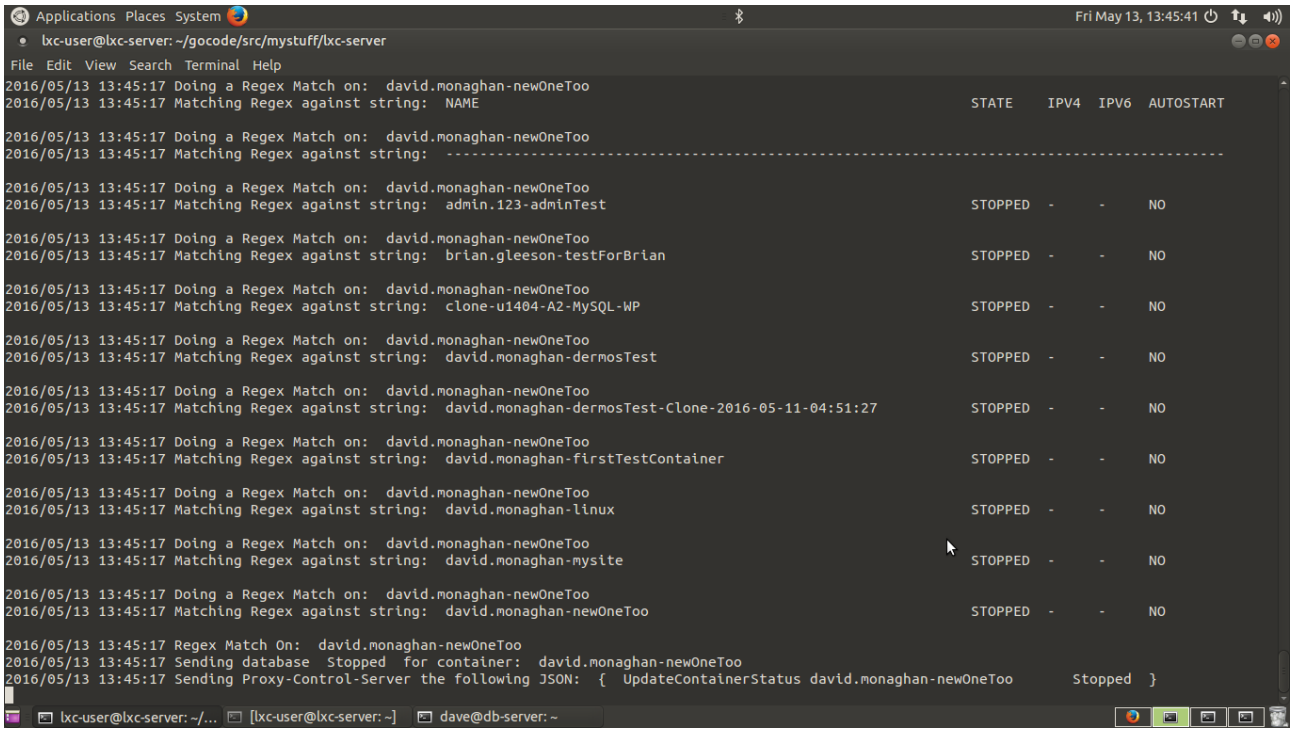
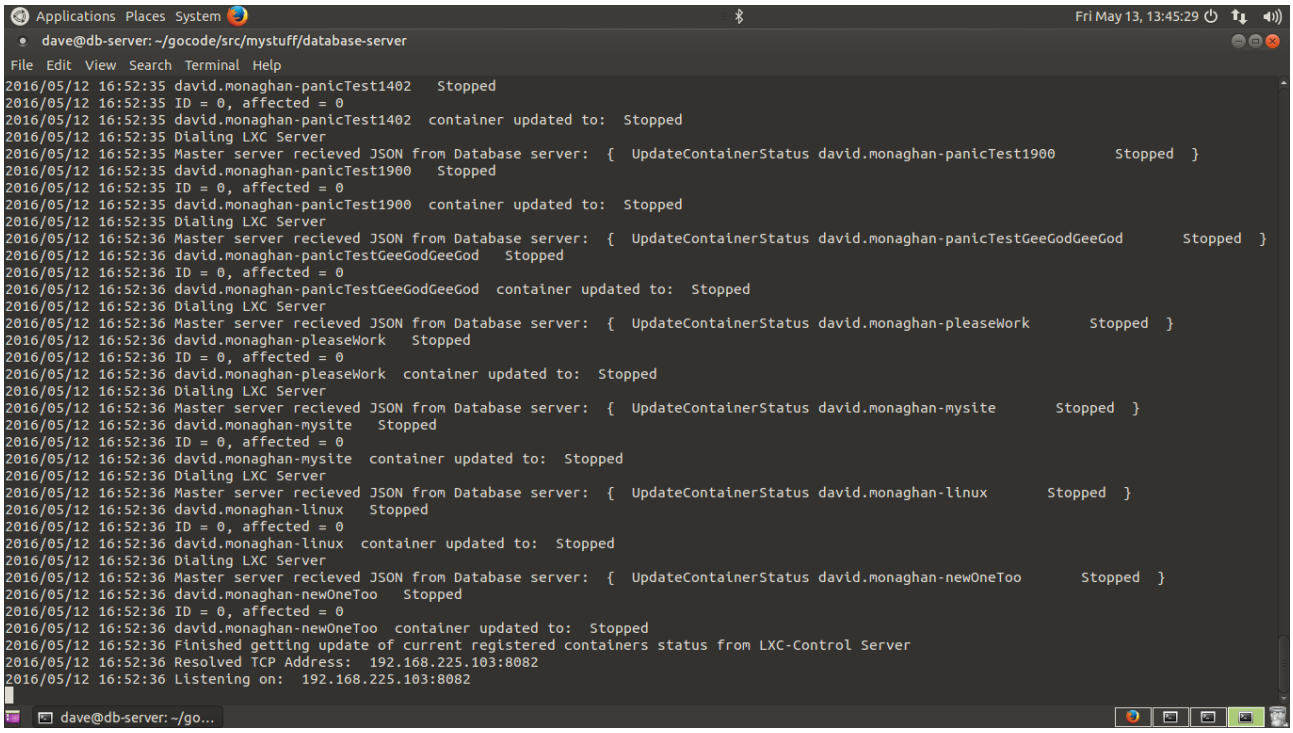


Figure 77: LXC-Control-Server is up after syncing with Database-Control-Server



6.2.2.3 – Database-Control-Server is up

Figure 78: Database-Control-Server is up after syncing with LXC-Control-Server



6.2.3 – Testing overview.html

6.2.3.1 – Stopping a Container

Figure 79: Overview.html showing a stopped container

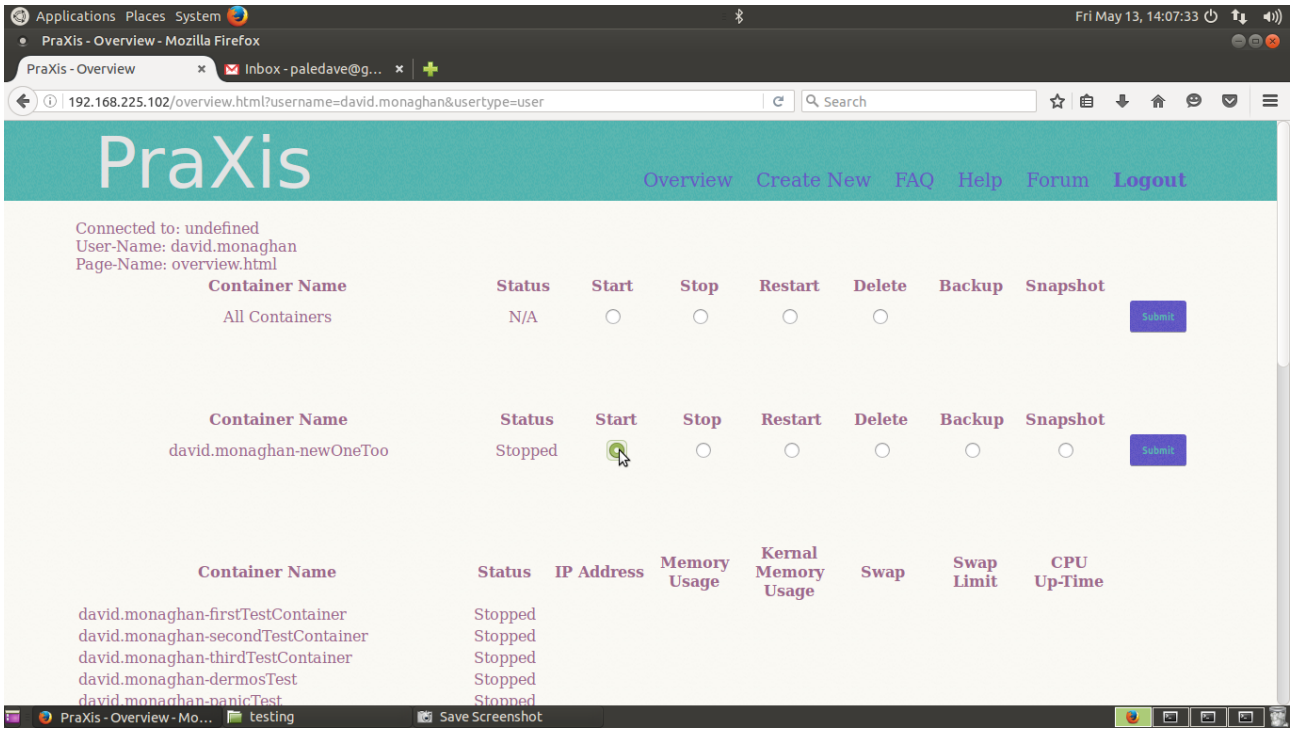


Figure 80: LXC-Control-Server showing a stopped container

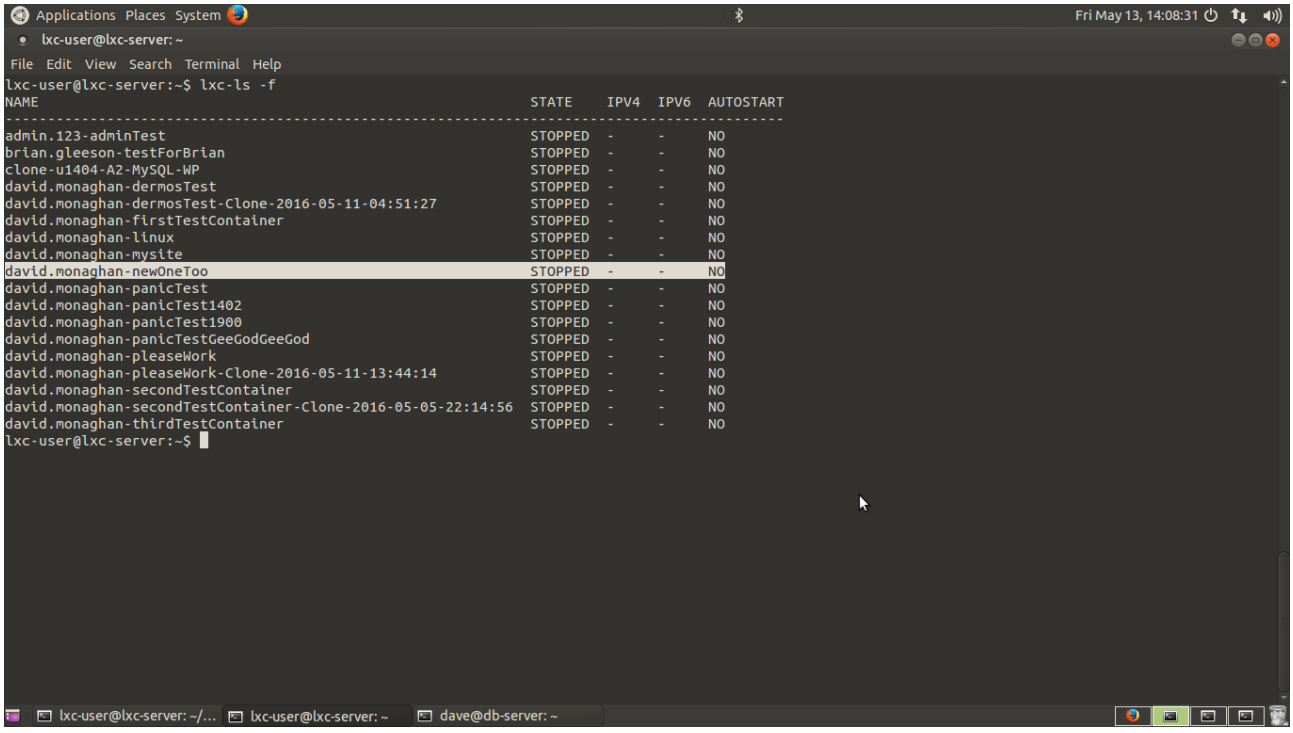


Figure 90: Database-Control-Server showing a stopped container

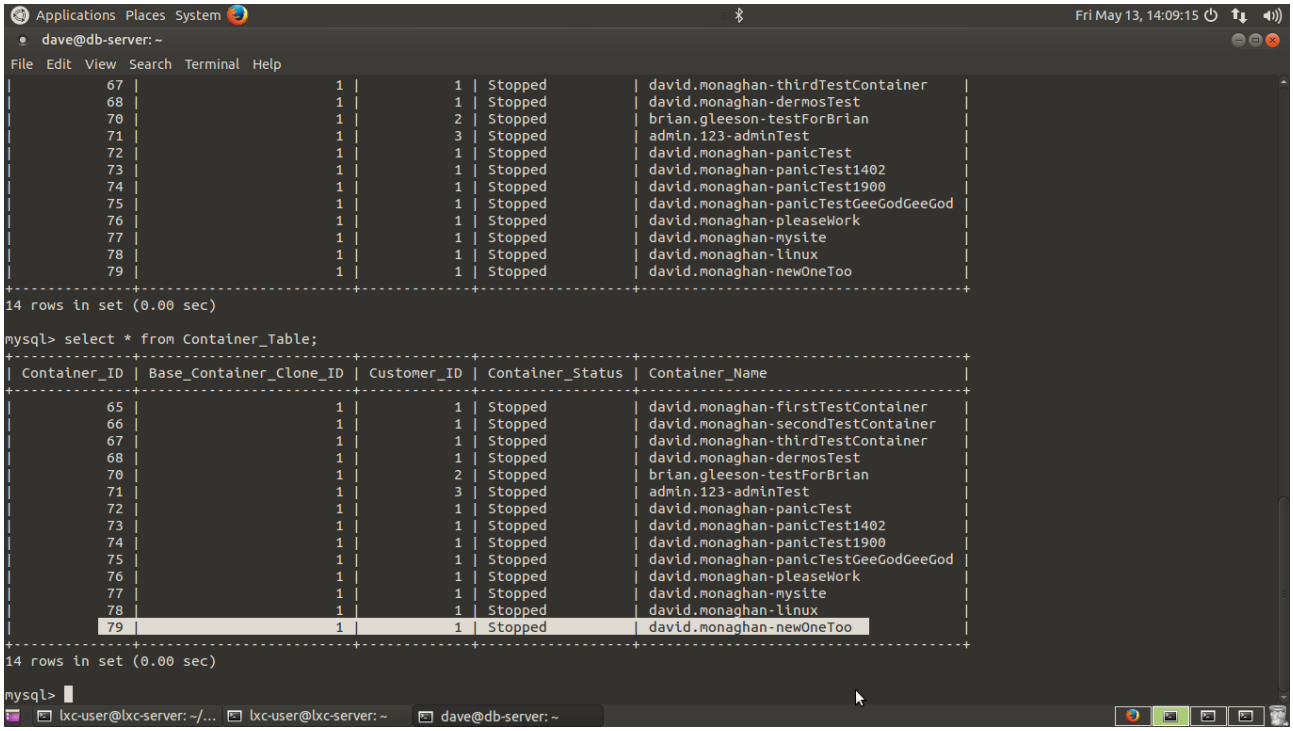
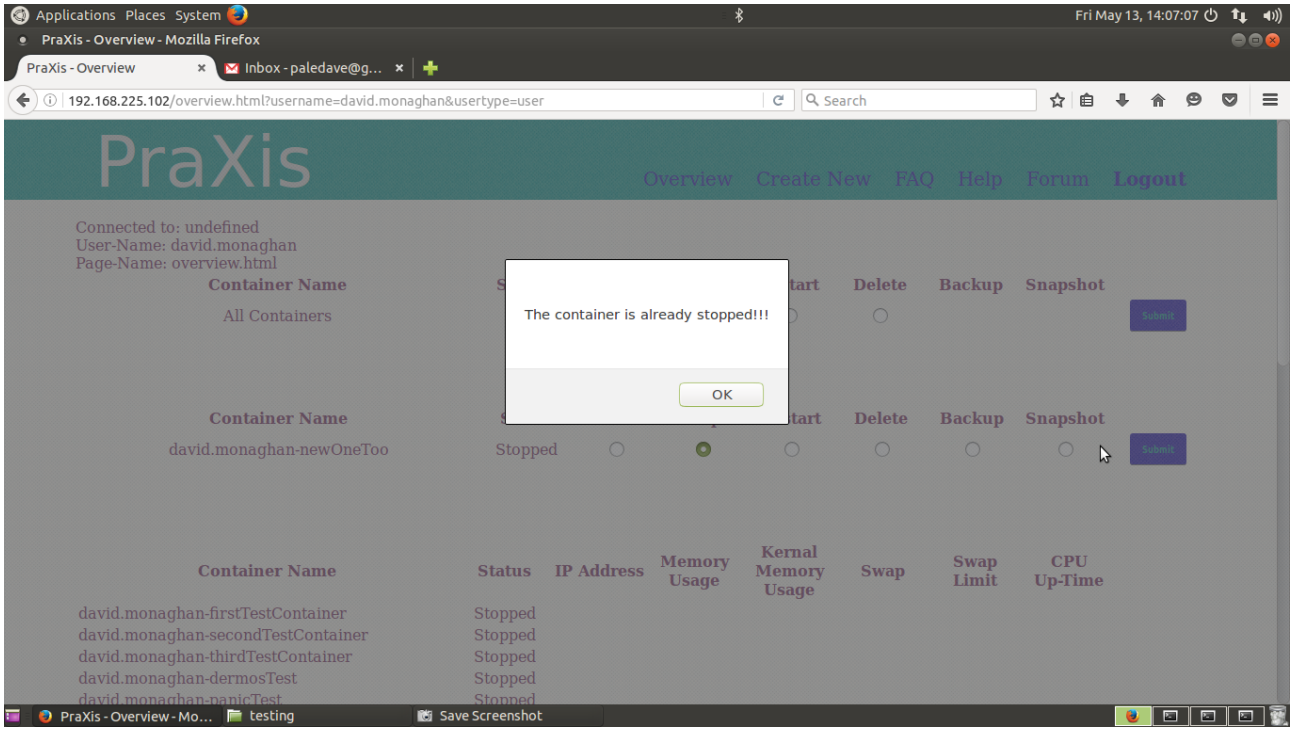


Figure 91: Overview.html showing alert message when trying to stop a container already stopped



6.2.3.2 – Starting A Container

Figure 92: Overview.html showing a stopped container

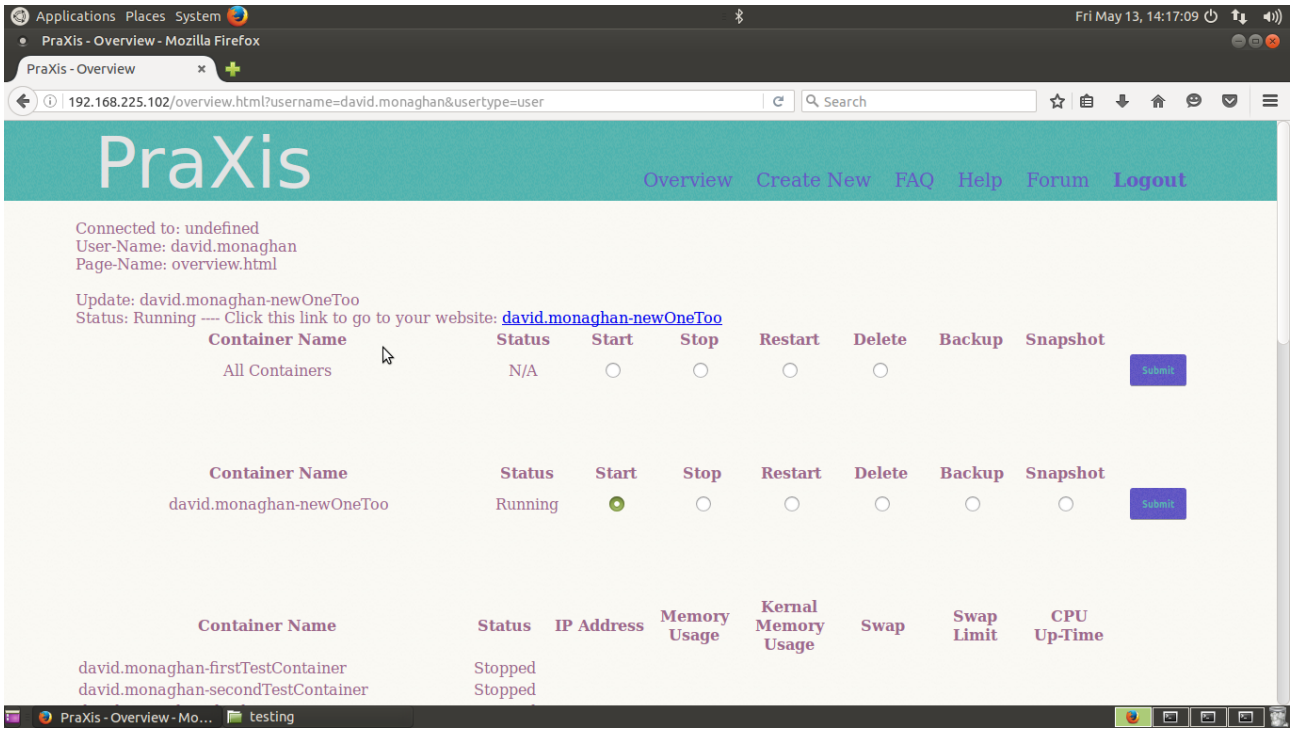


Figure 93: Wordpress installation on a running container

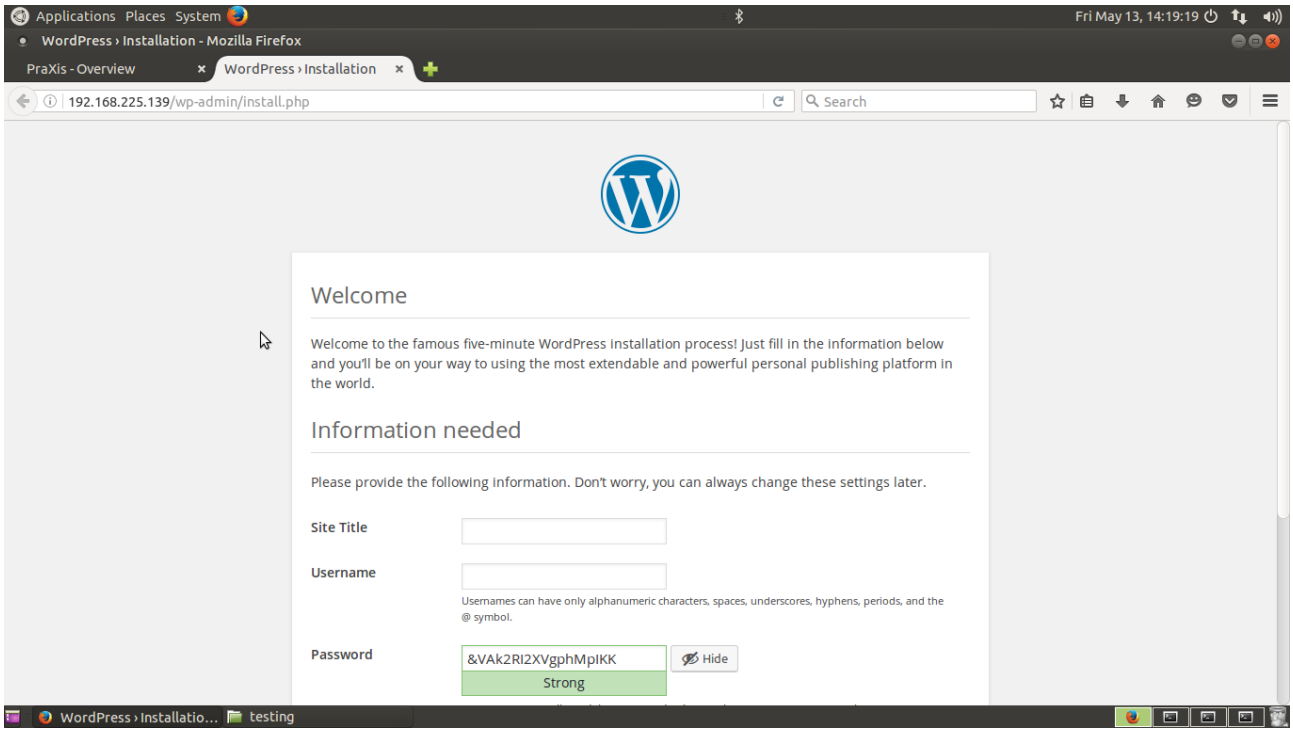


Figure 94: LXC-Control-Server showing a stopped container

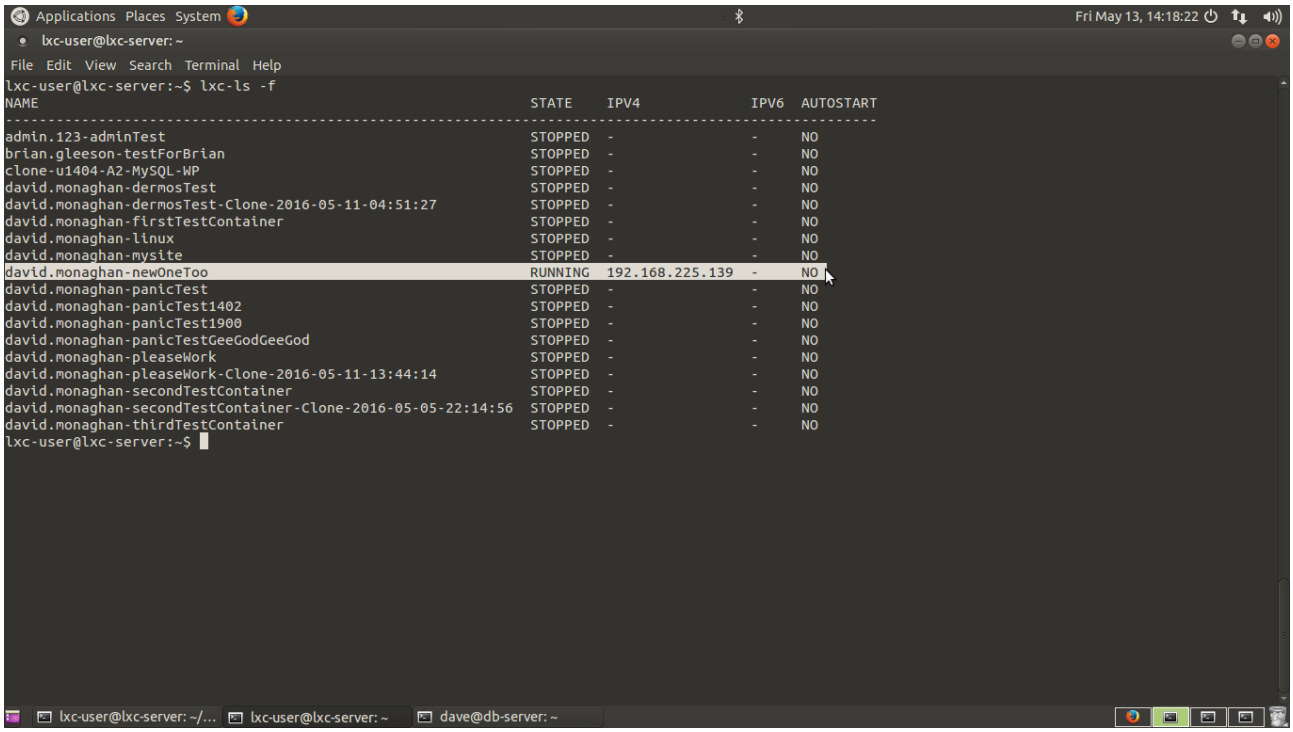


Figure 95: Database-Control-Server showing a stopped container

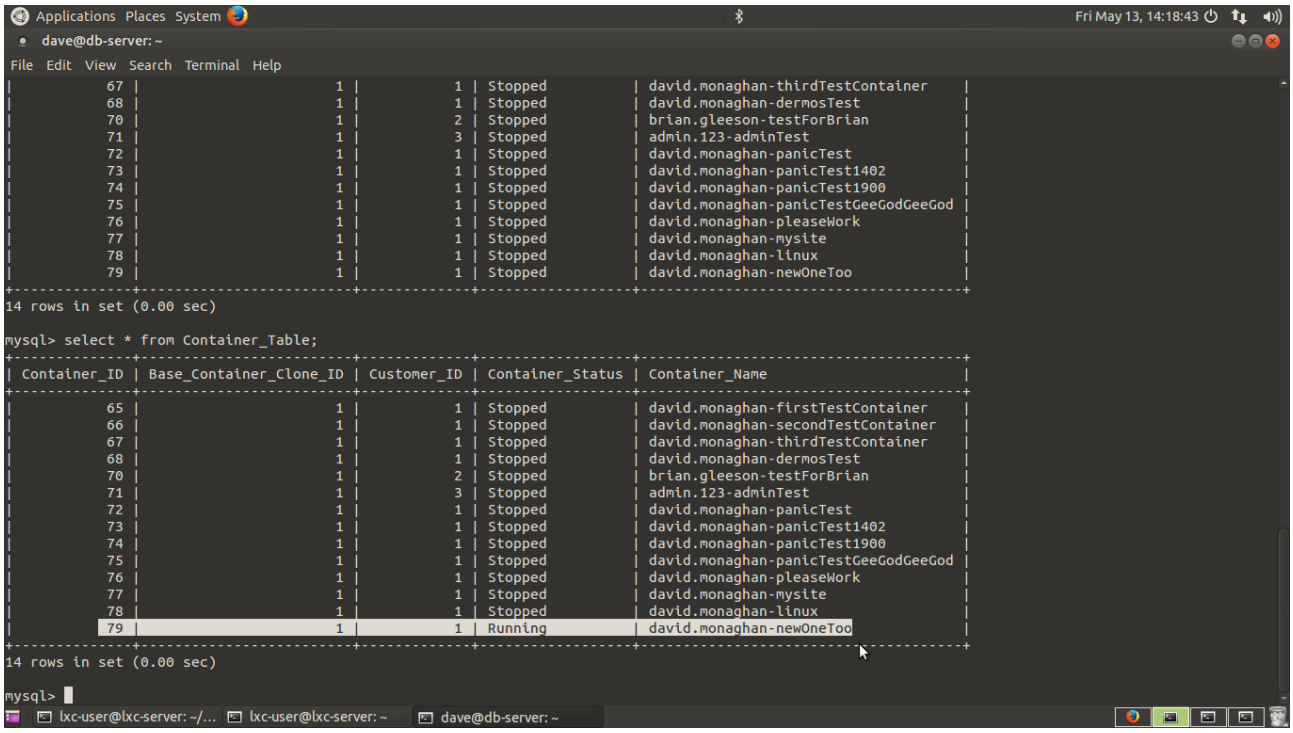
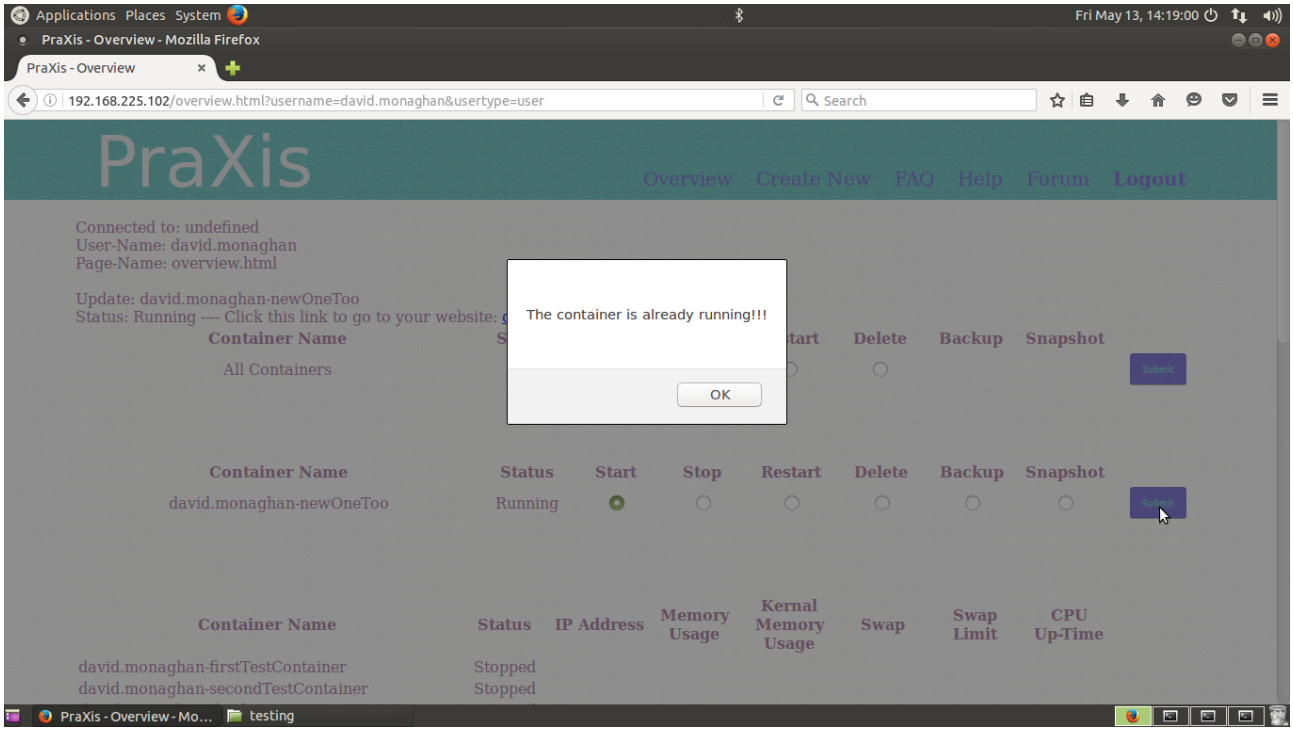


Figure 96: Overview.html showing alert message when trying to start a container already running



6.2.3.3 – Stopping a Running Container

Figure 97: Overview.html stopping a running container

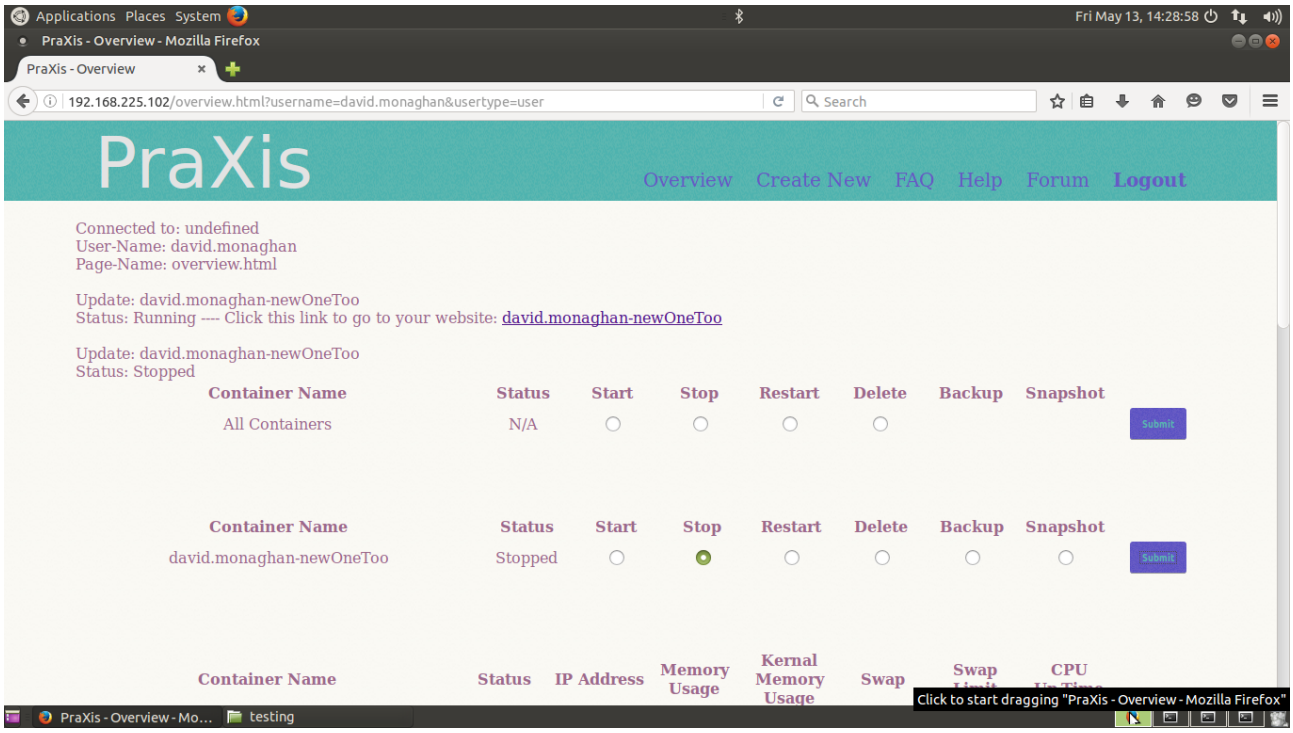


Figure 98: LXC-Control-Server showing the now stopped container

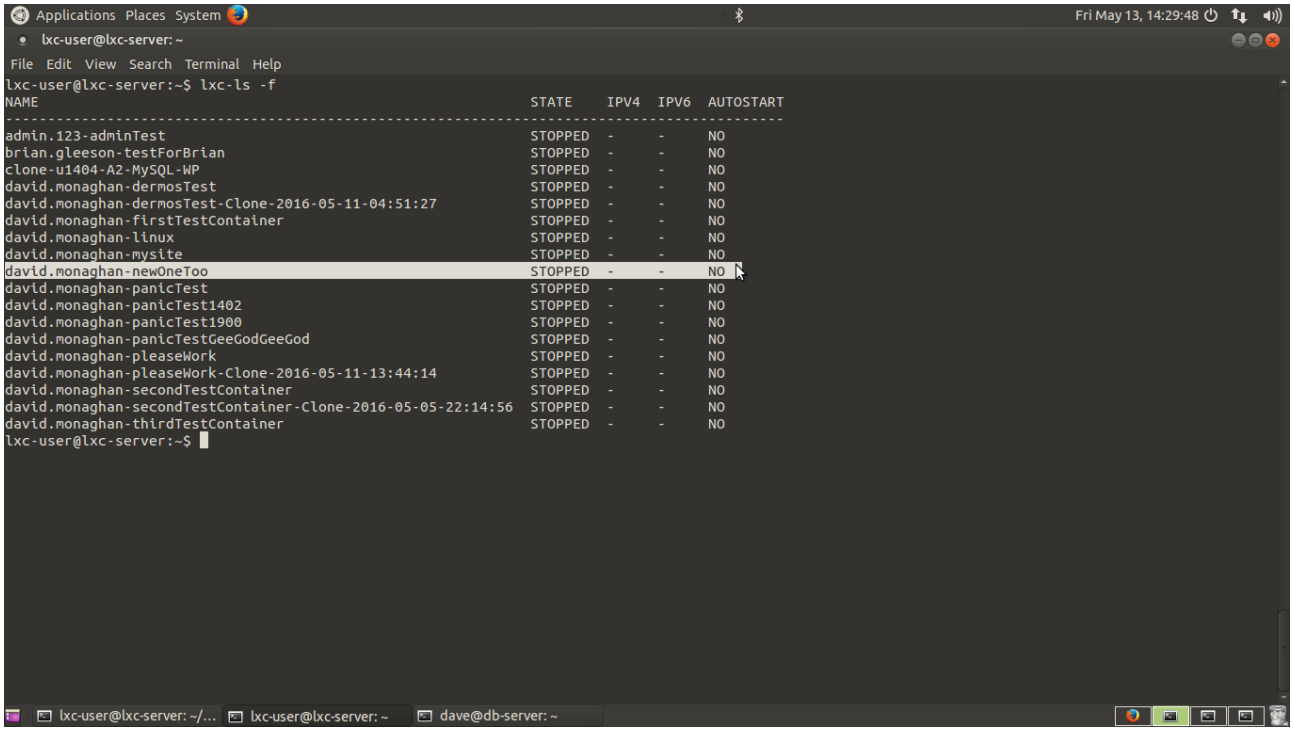


Figure 99: Database-Control-Server showing a stopped container

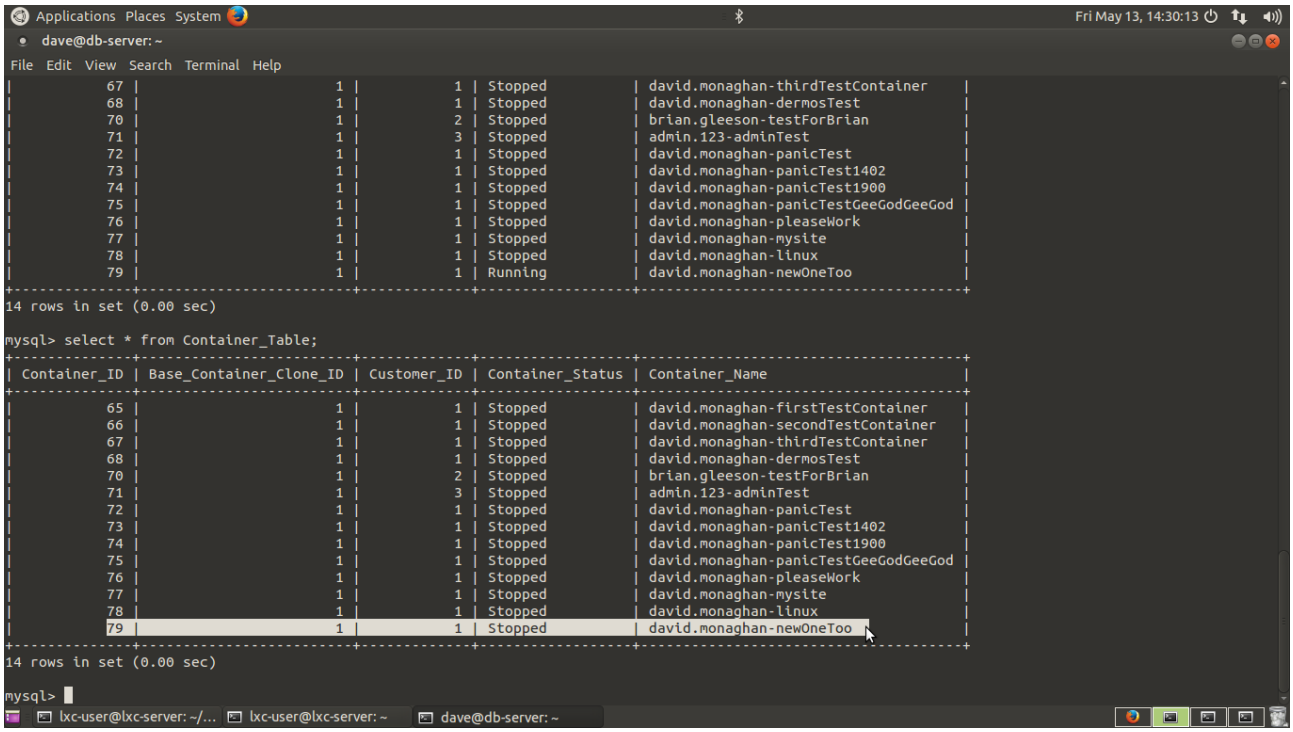
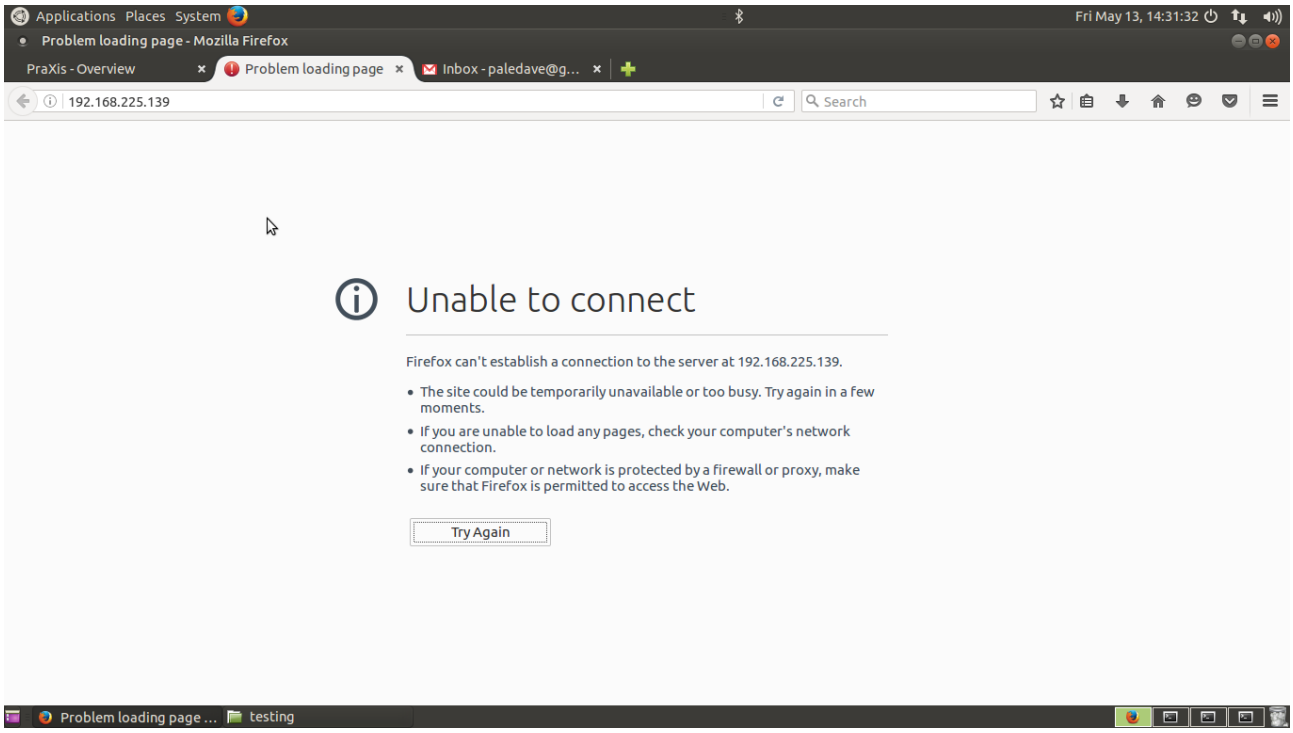


Figure 100: Wordpress no longer running on a stopped container



6.2.3.4 – Restarting a Container

Figure 101: LXC-Control-Server Output for previous tests

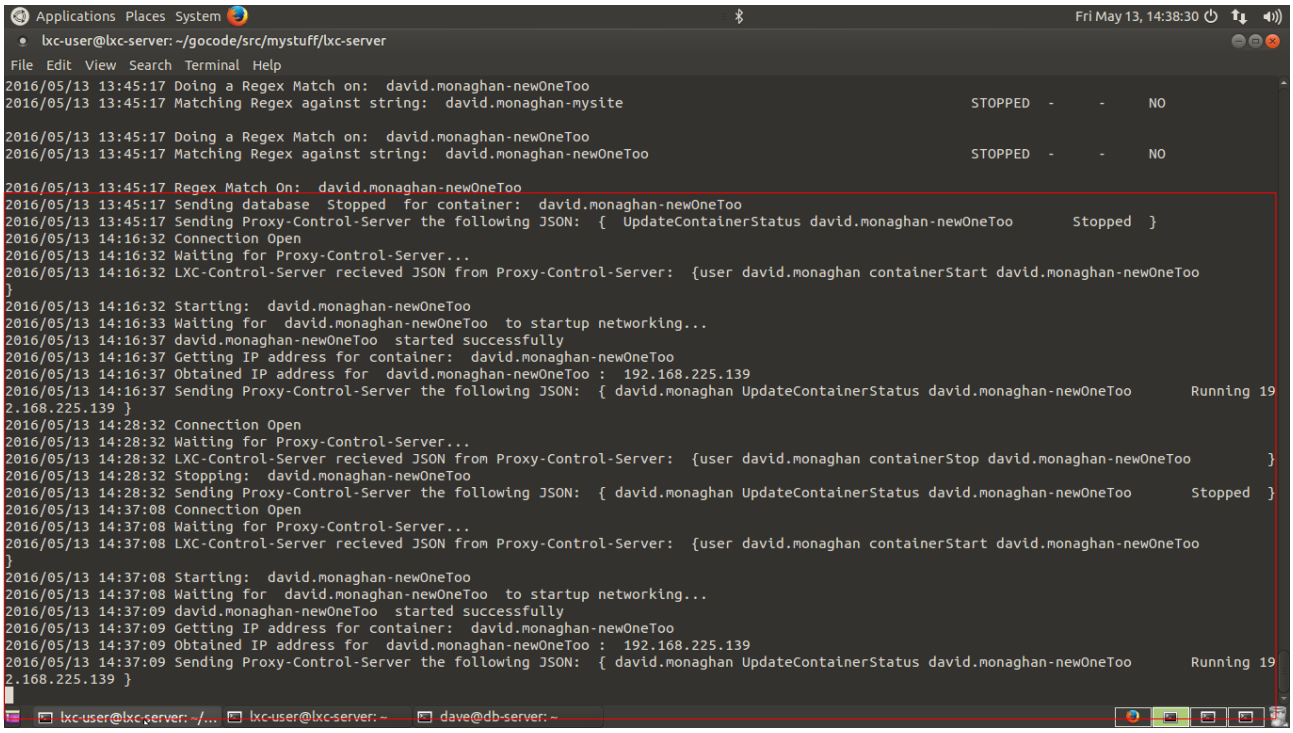


Figure 102: LXC-Control-Server status prior to container restart

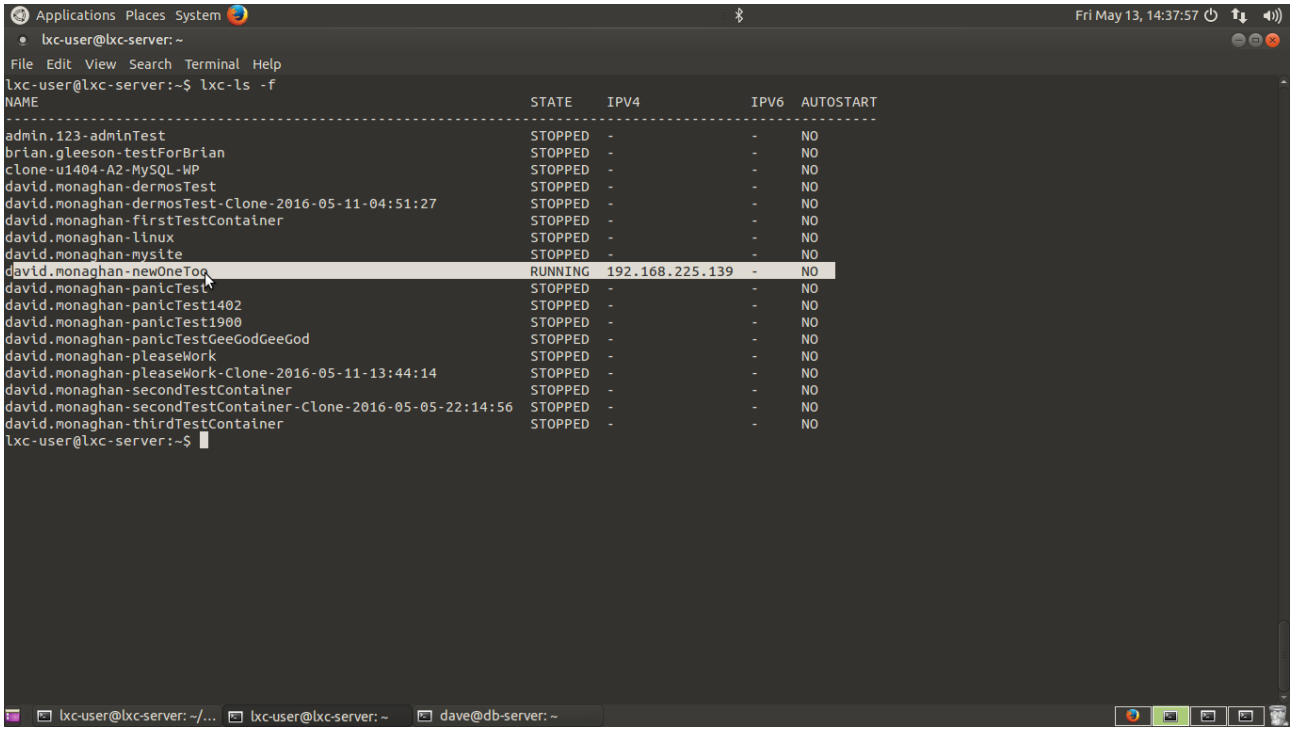


Figure 103: Database-Control-Server status prior to container restart

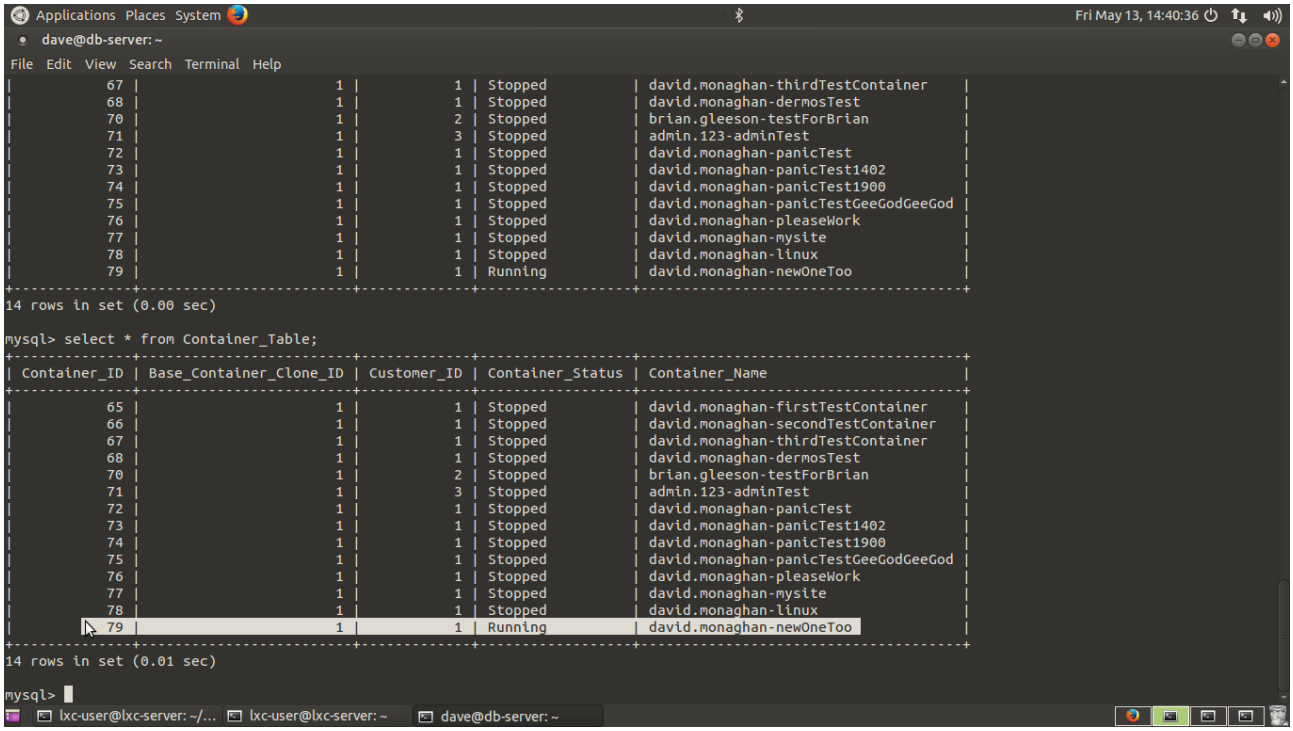


Figure 104: Overview.html restarting a running container

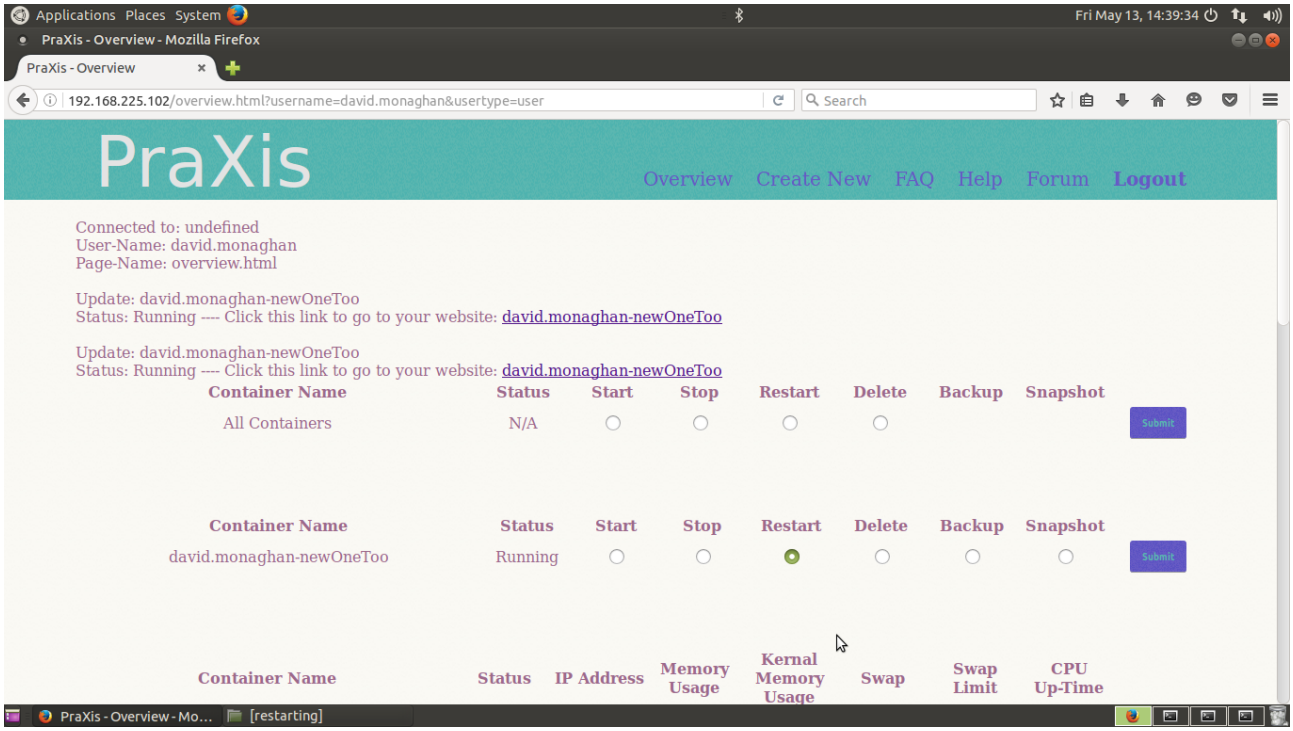


Figure 105: Wordpress running on a restarted container

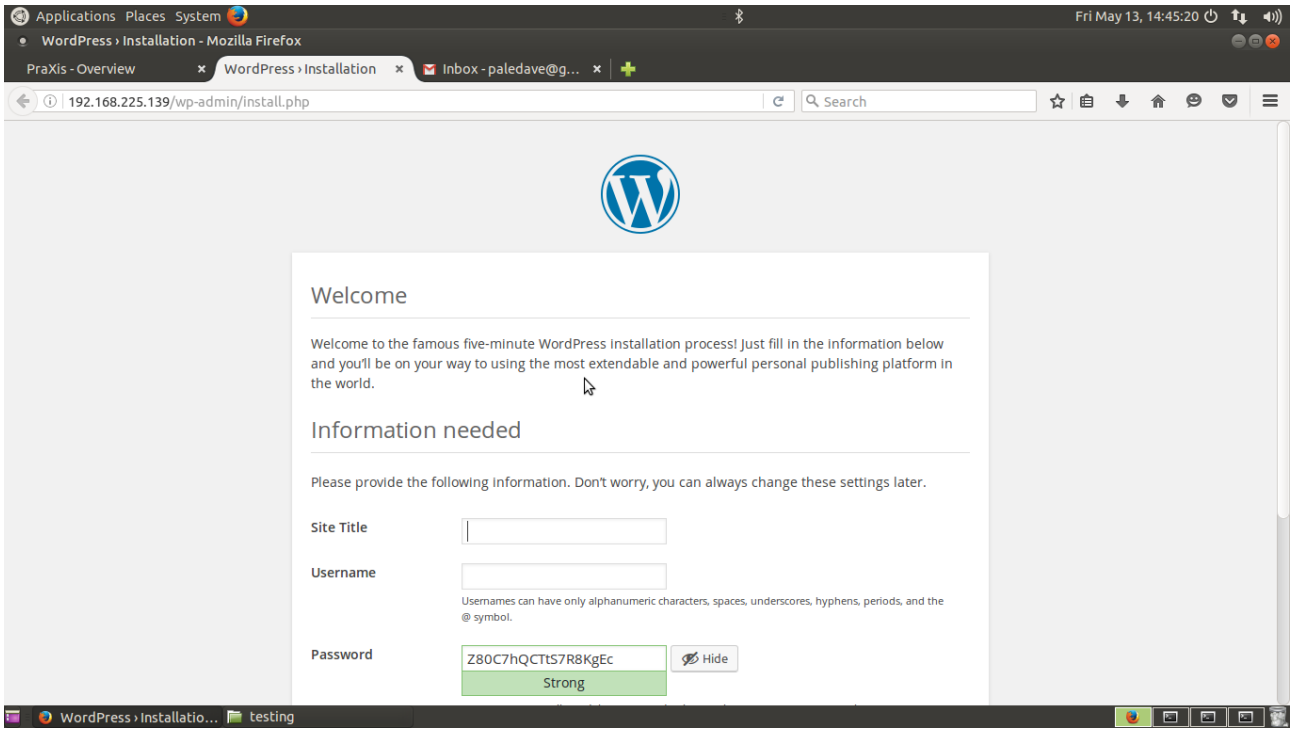


Figure 106: LXC-Control-Server status after container restart

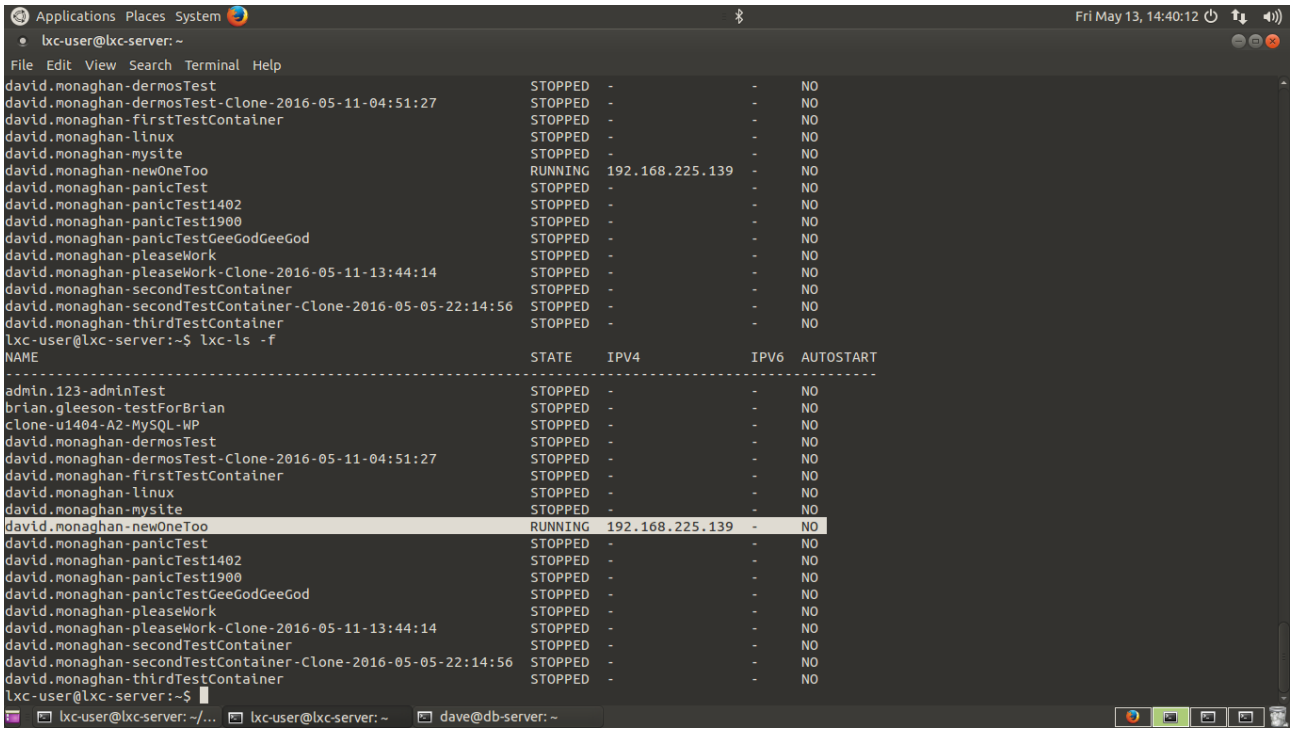


Figure 107: Database-Control-Server status after container restart

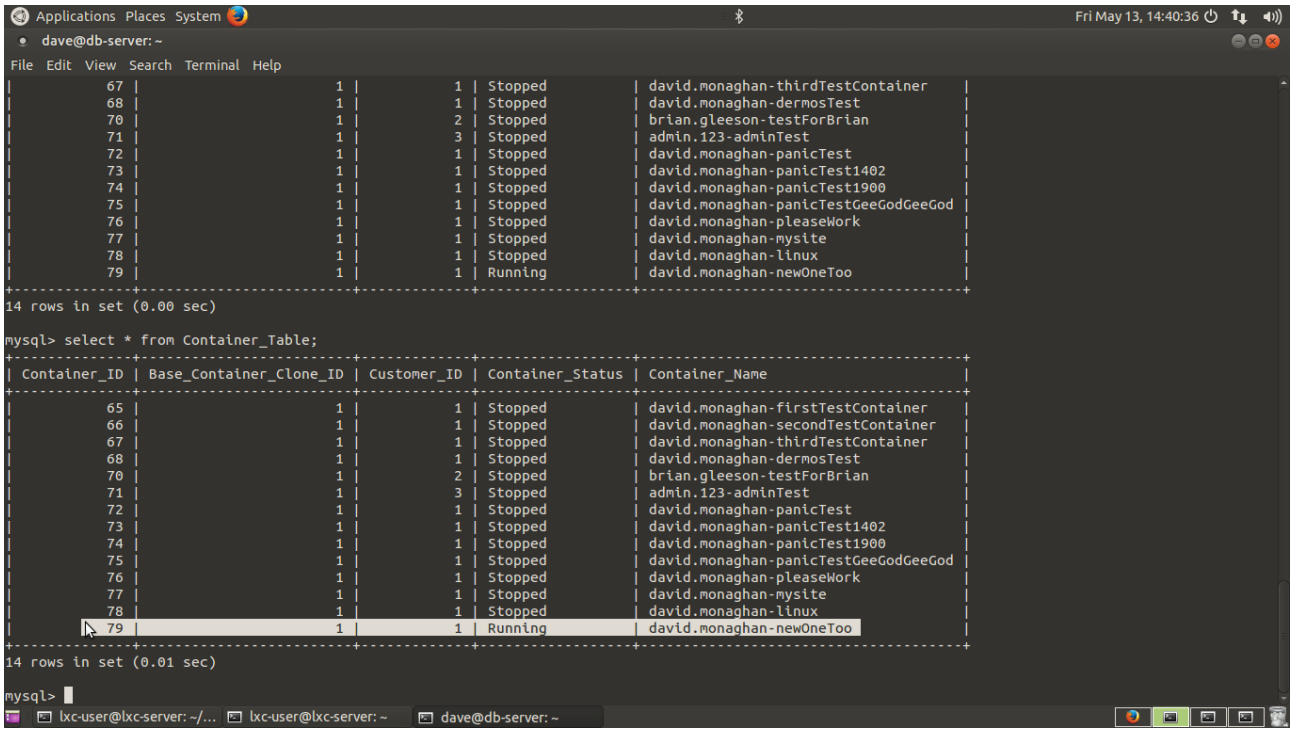
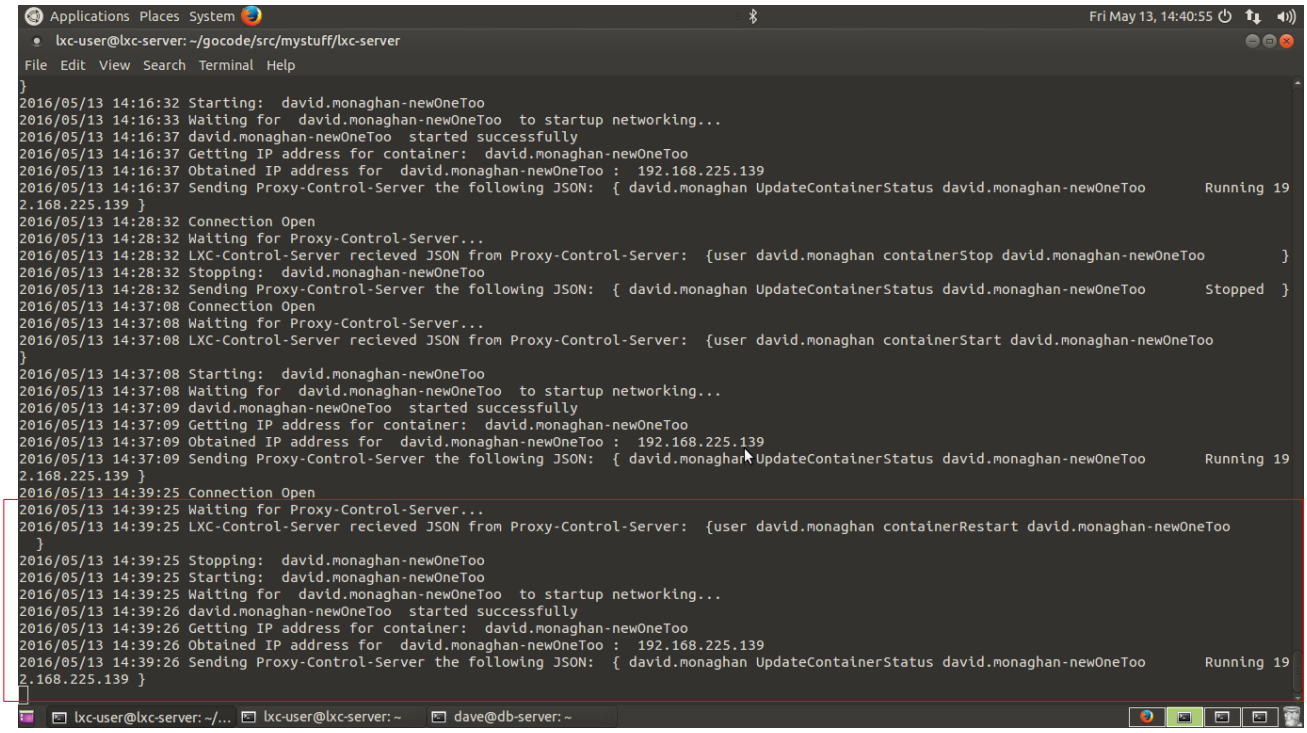


Figure 108: LXC-Control-Server output after restarting the container



6.2.3.5 – Deleting a Container

Figure 109: Overview.html showing an alert when deleting a running container

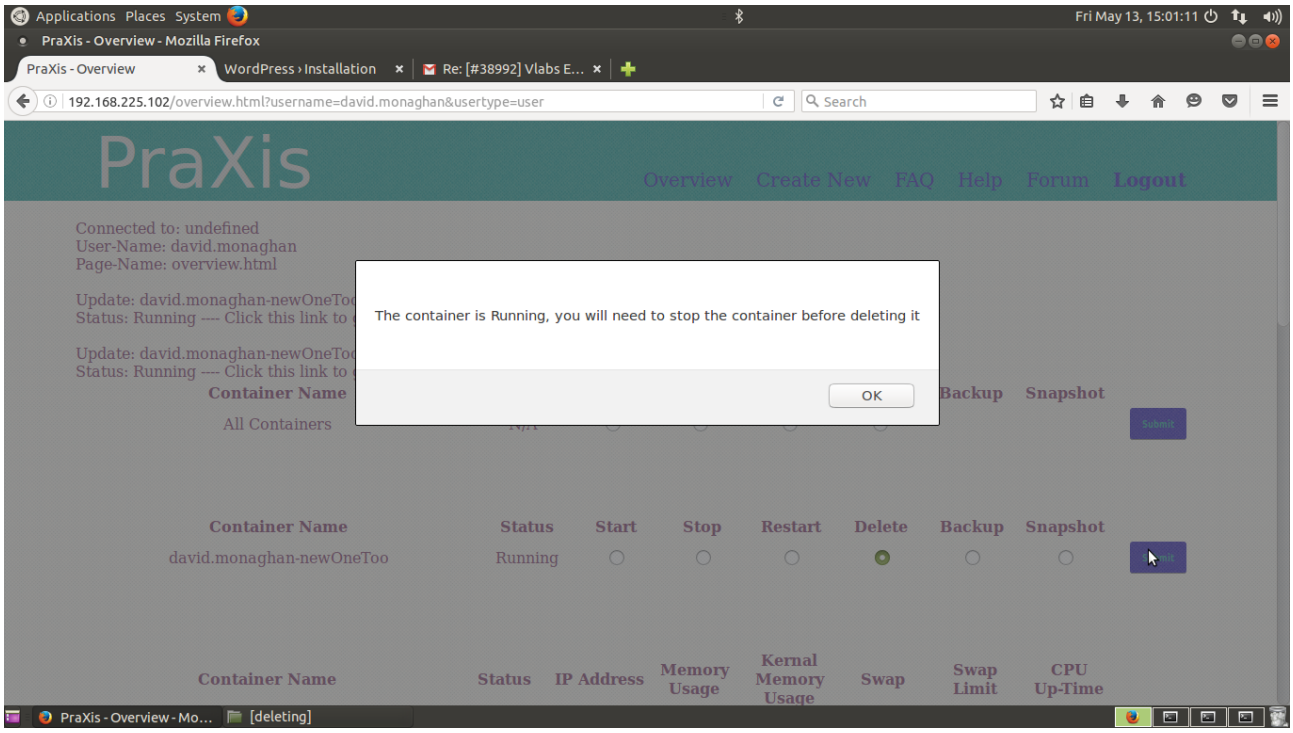


Figure: 110: LXC-Control-Server status prior to deleting container

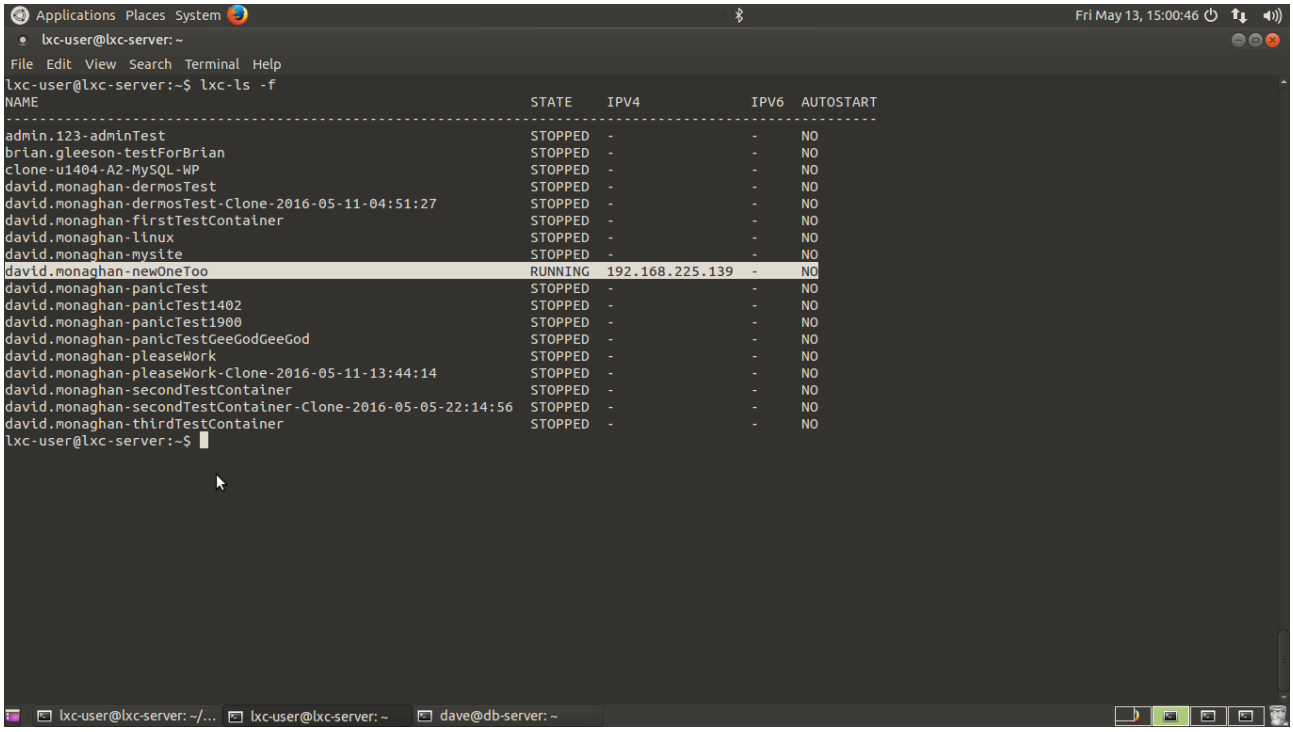


Figure: 111: Database-Control-Server status prior to deleting container

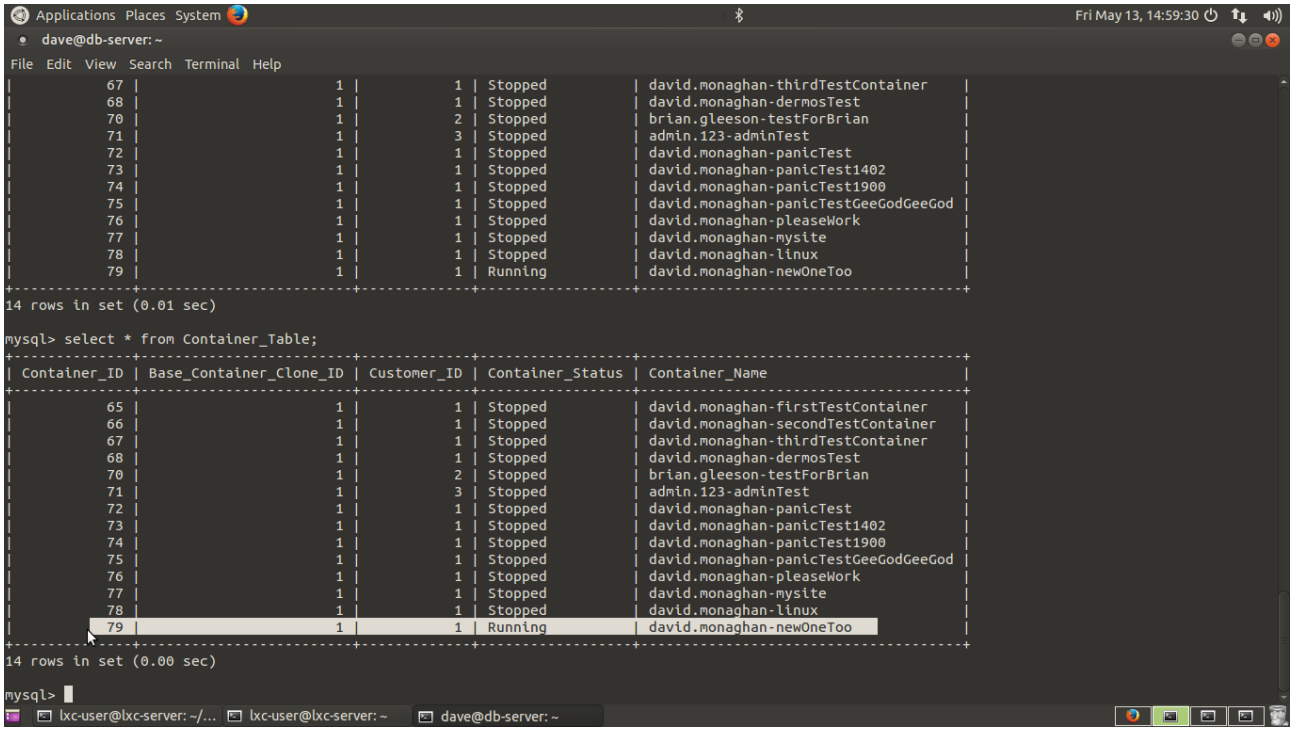


Figure 112: Overview.html showing a stopped container

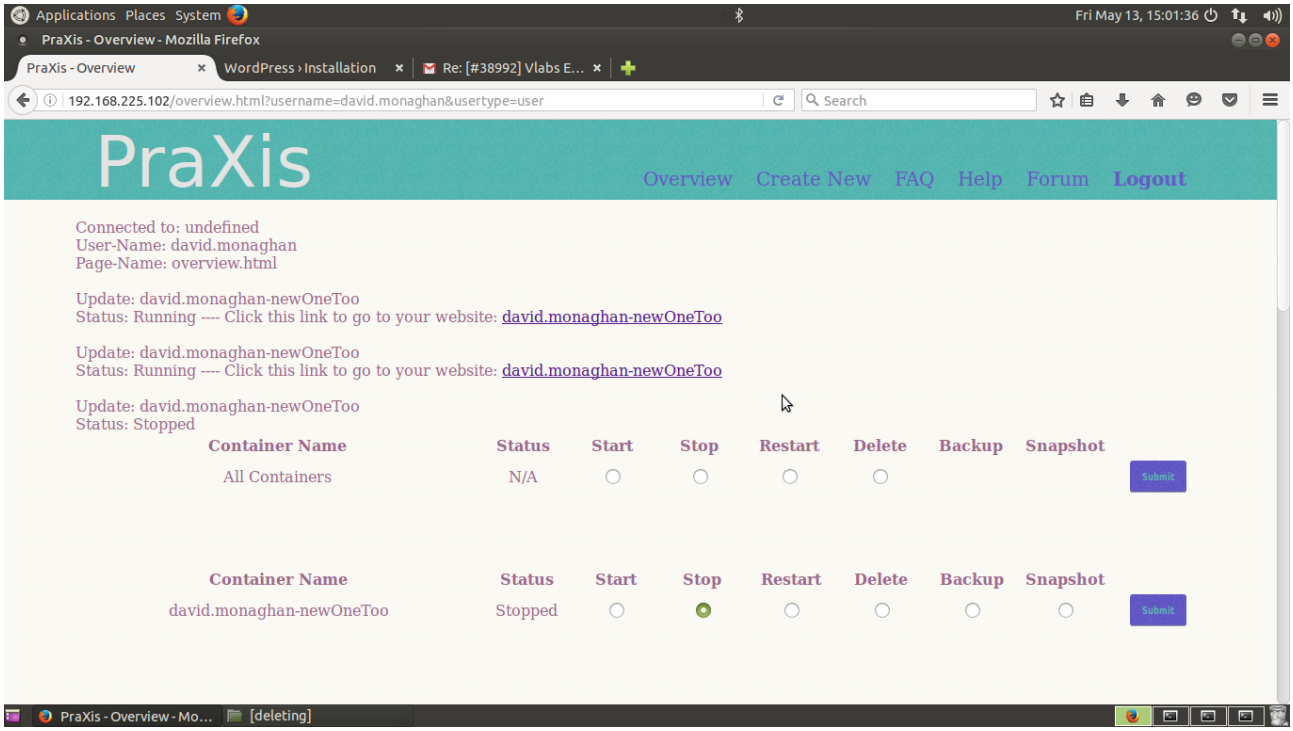


Figure 113: LXC-Control-Server showing stopped container

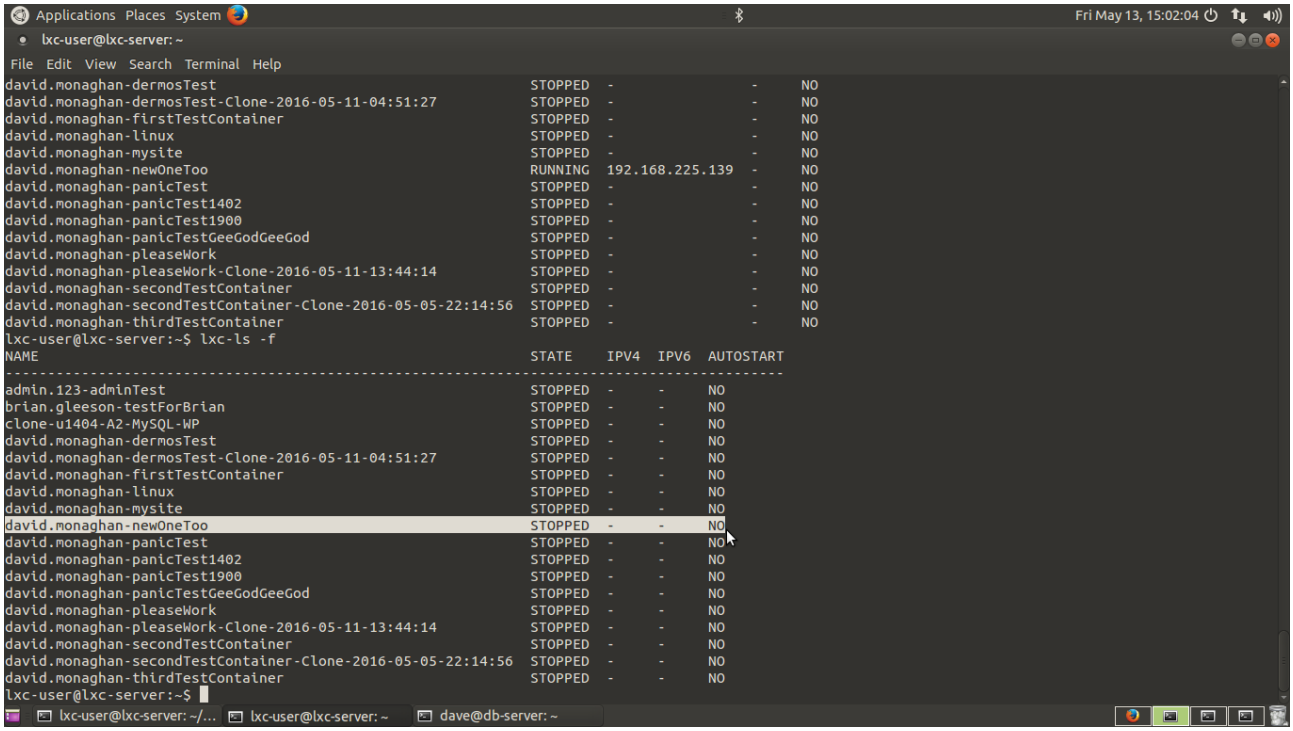


Figure 114: Database-Control-Server showing stopped container

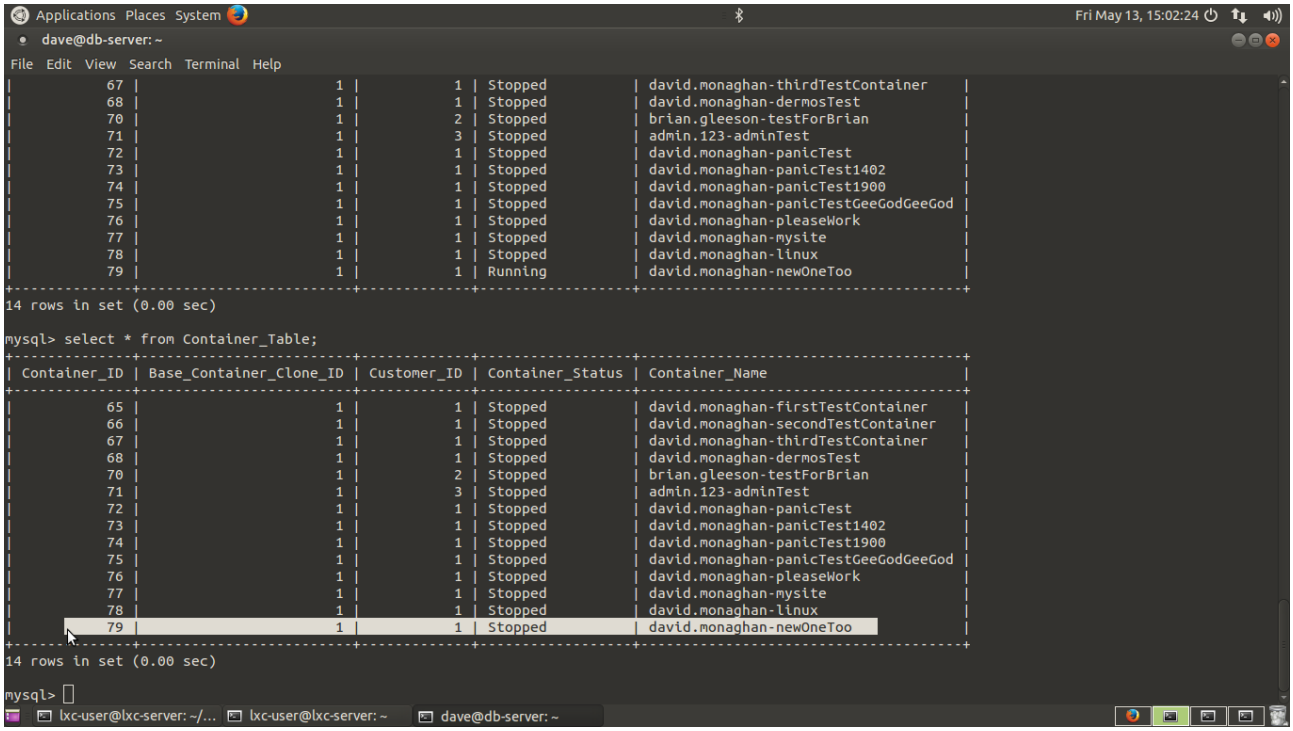


Figure 115: Overview.html showing a deleted container

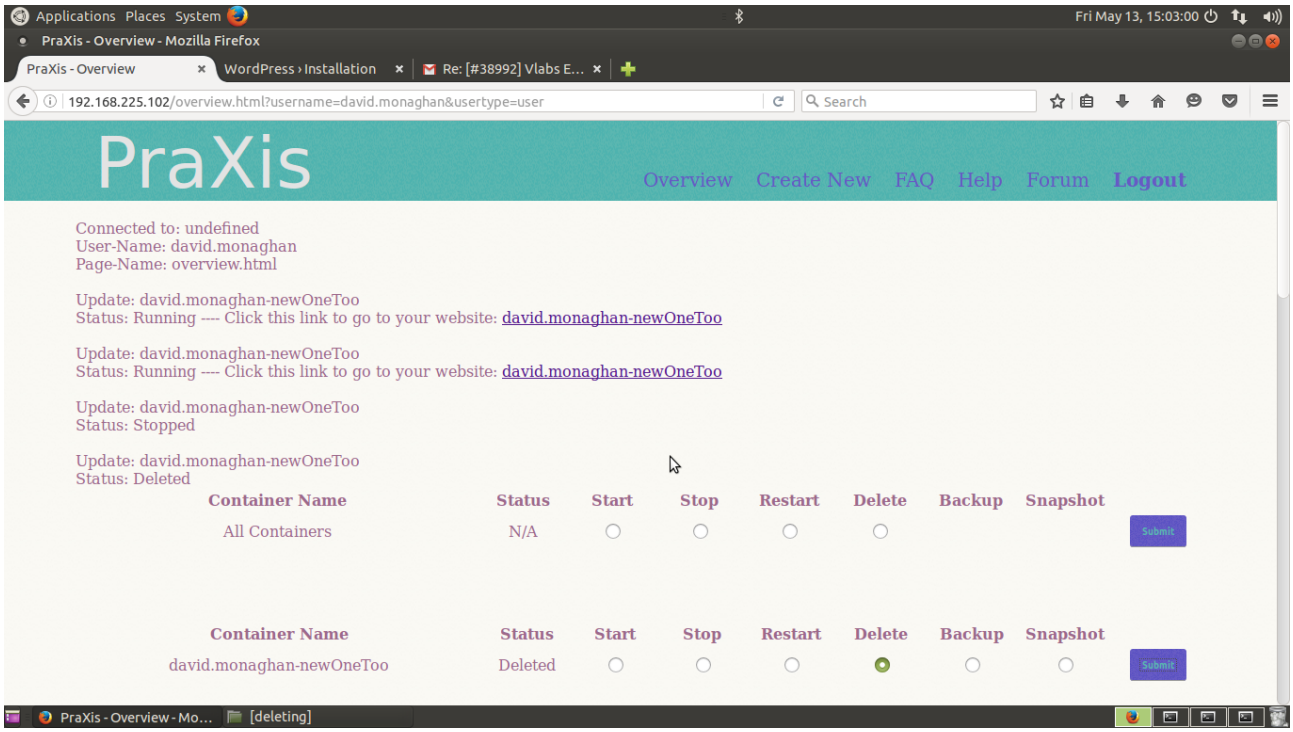


Figure 116: LXC-Control-Server showing a stopped container

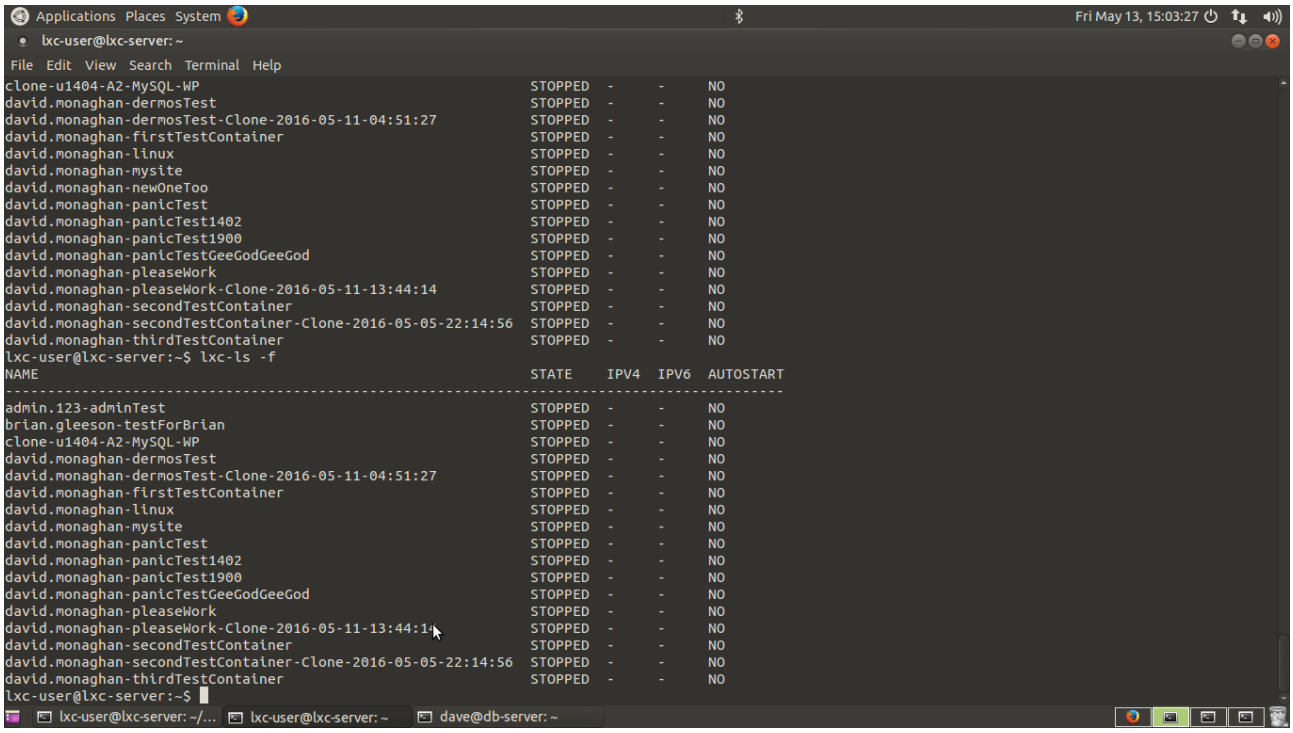


Figure 117: Database-Control-Server showing a deleted container

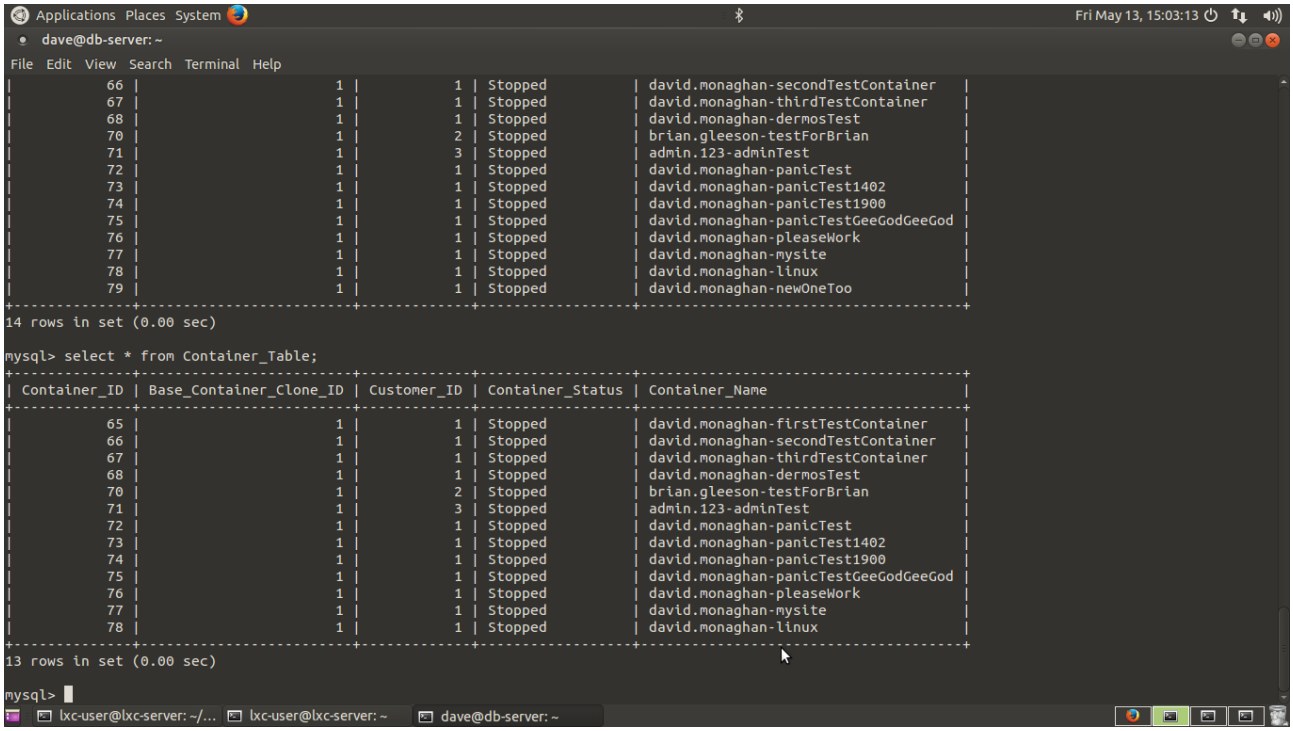
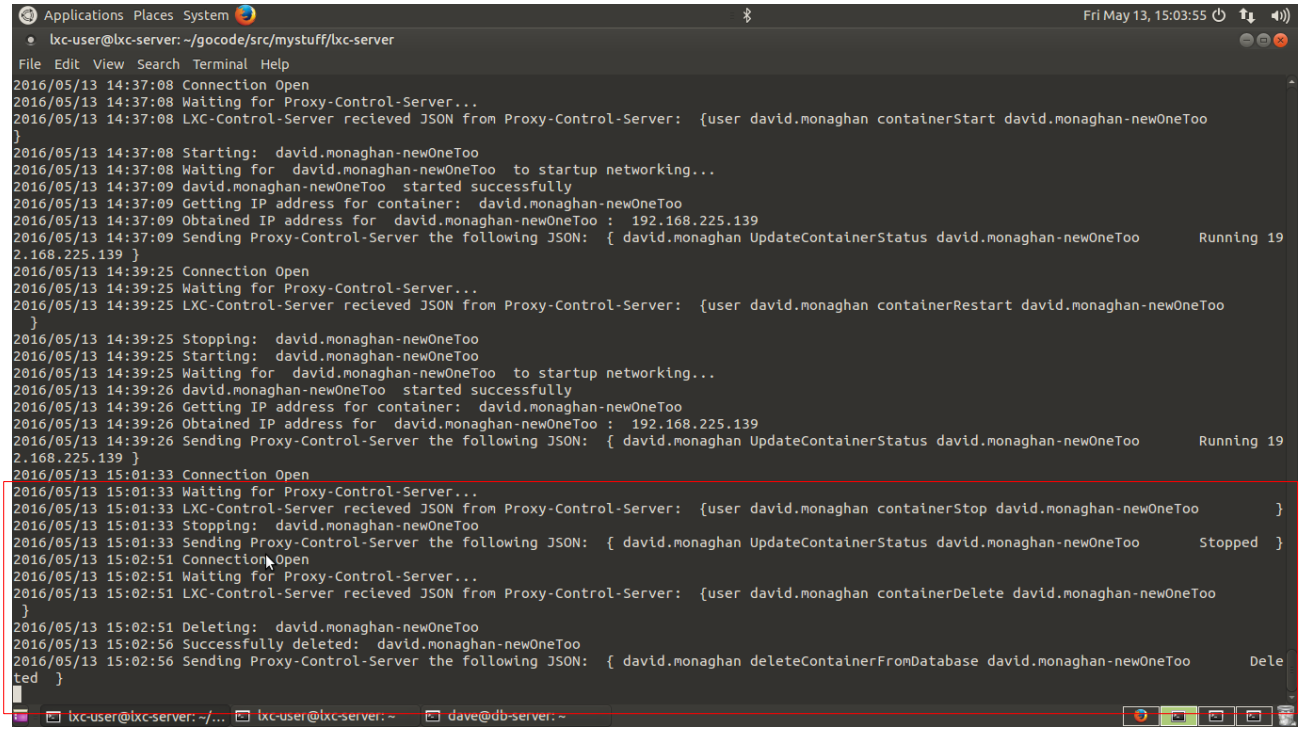


Figure 118: LXC-Control-Server output after container deleted



6.2.3.6 – Backing up a Container

Figure 119: Overview.html showing error message when backing up a running container

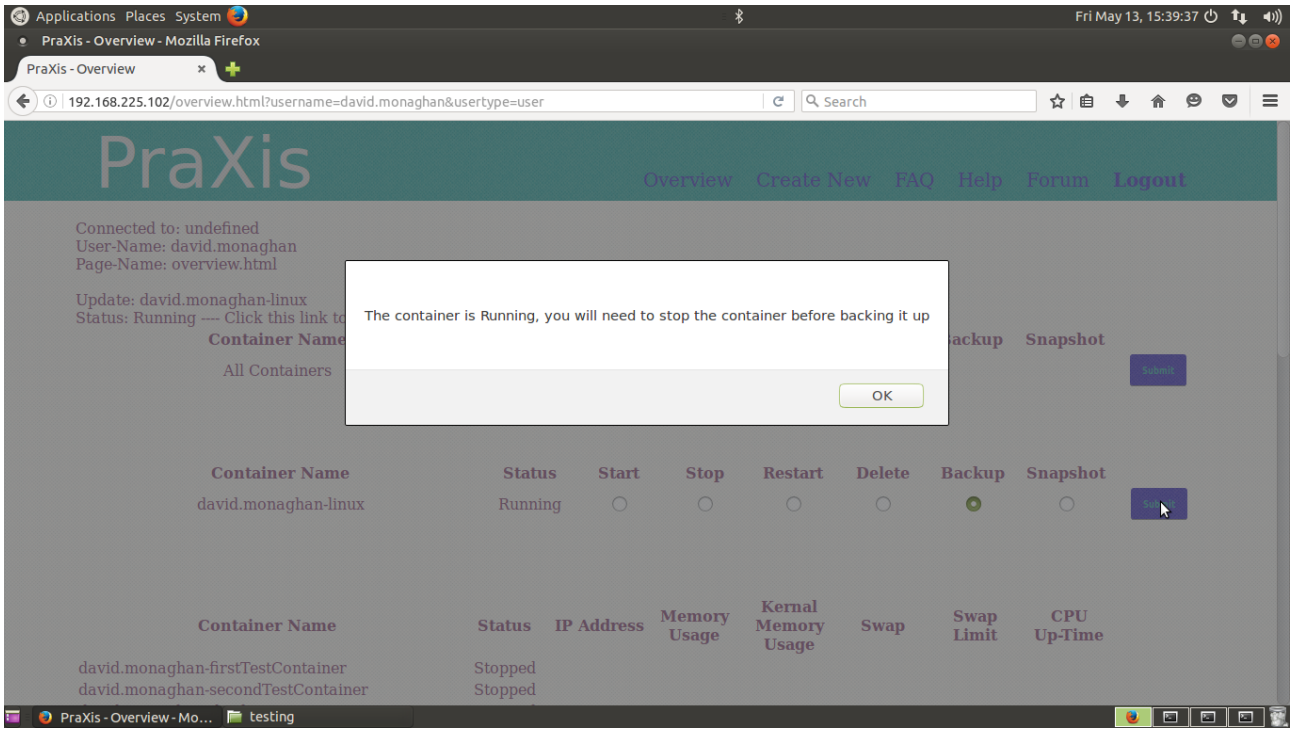


Figure 120: LXC-Control-Server showing a running container

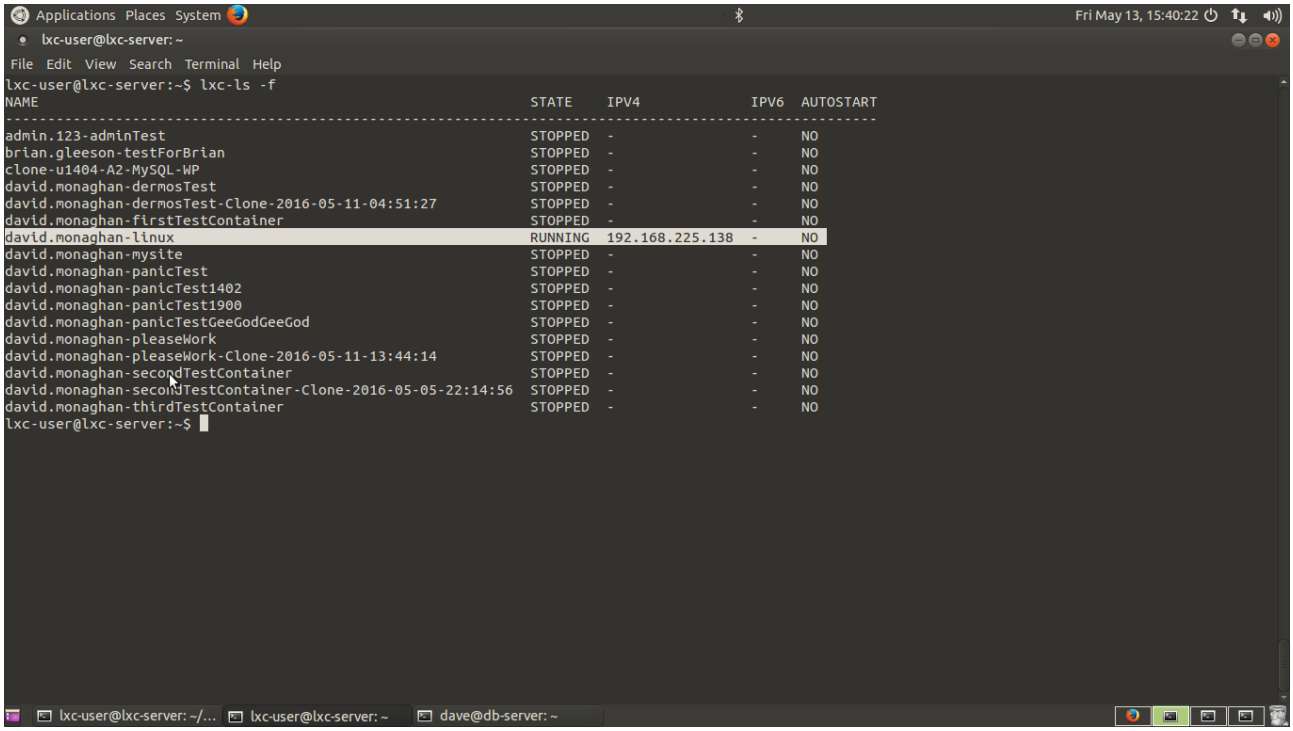


Figure 121: Database-Control-Server showing a running container

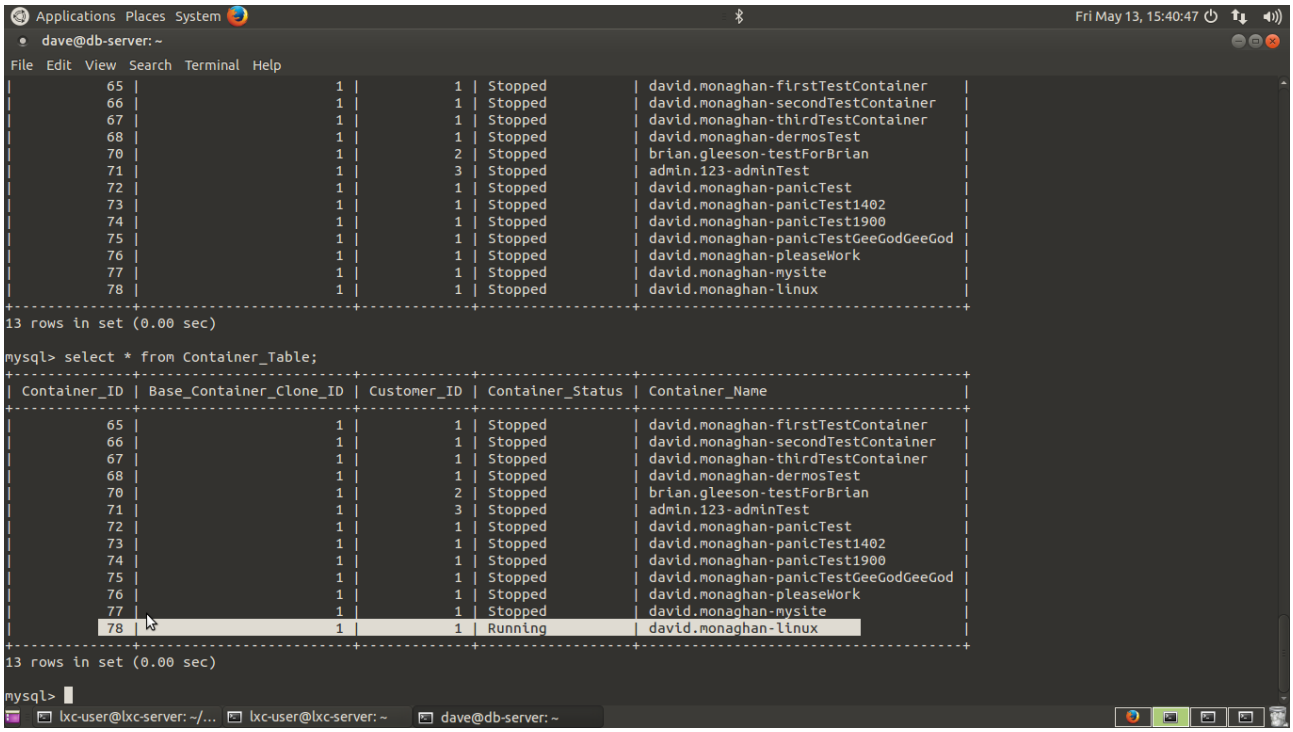


Figure 122: Overview.html showing a stopped container

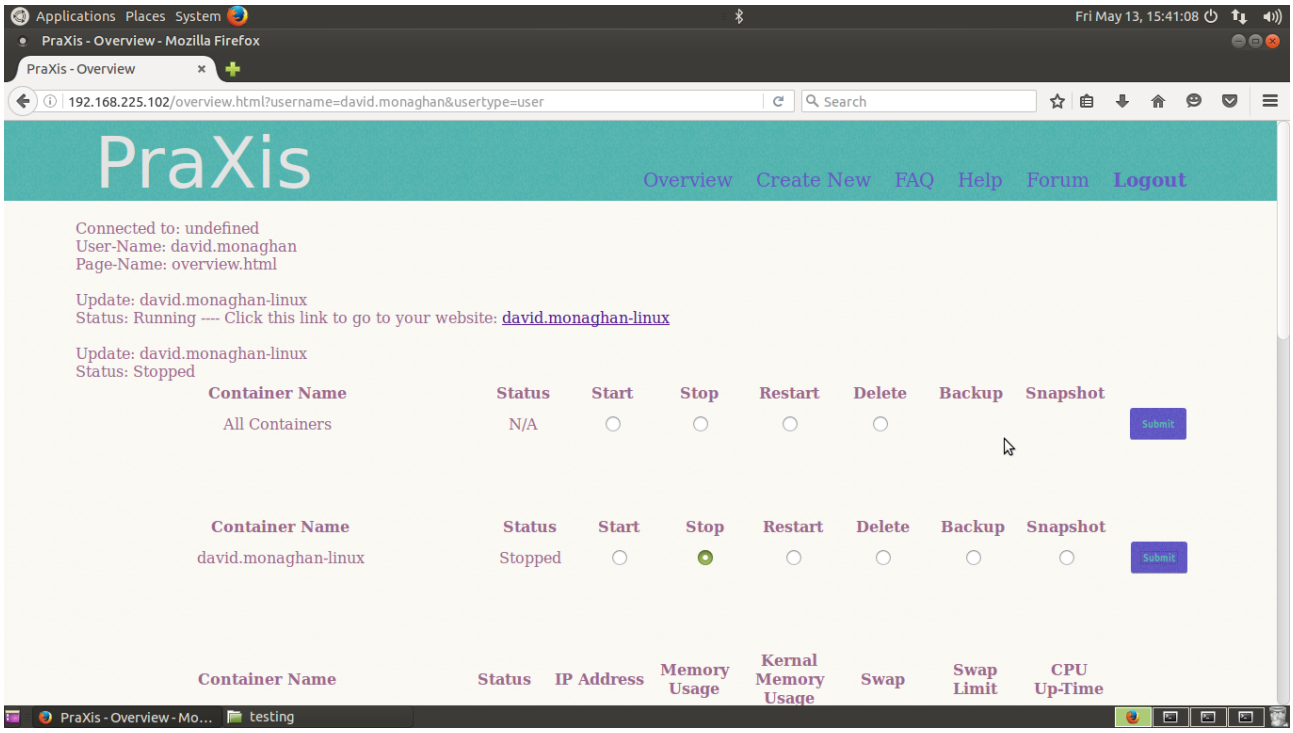


Figure 123: LXC-Control-Server showing a stopped container

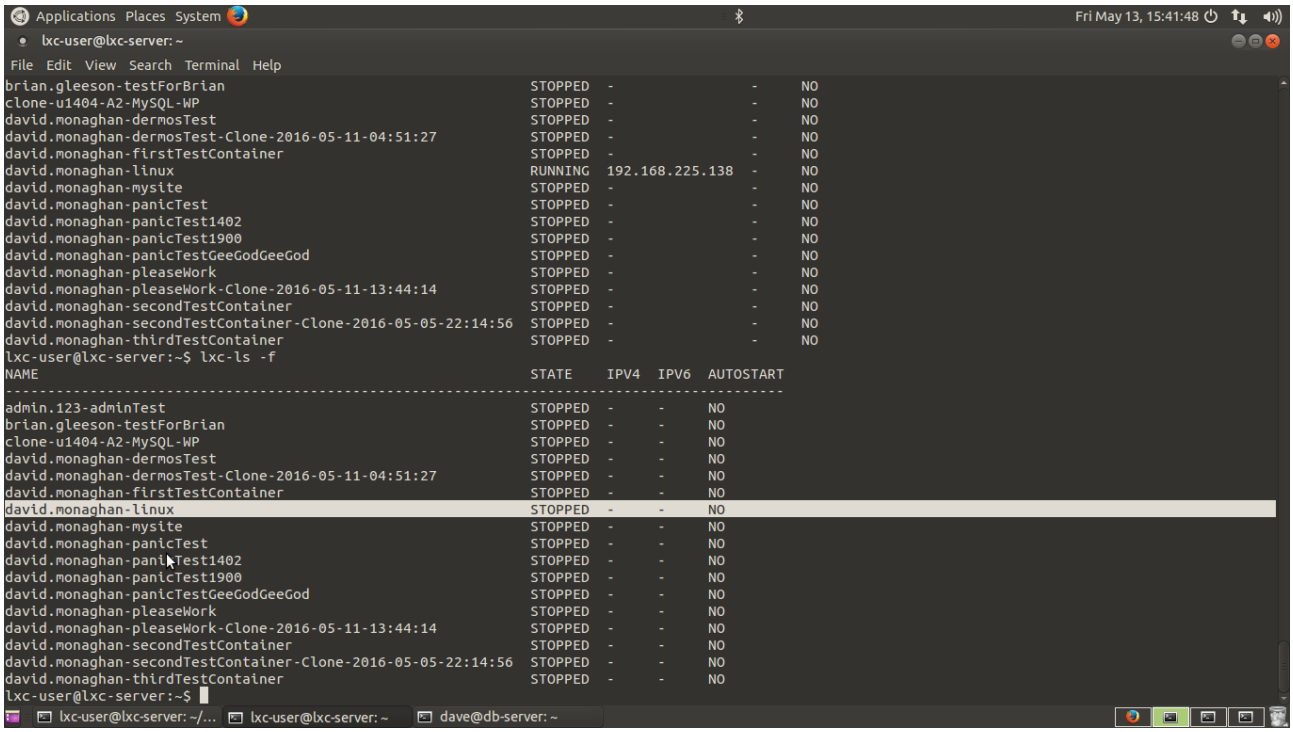


Figure 124: Database-Control-Server showing a stopped container

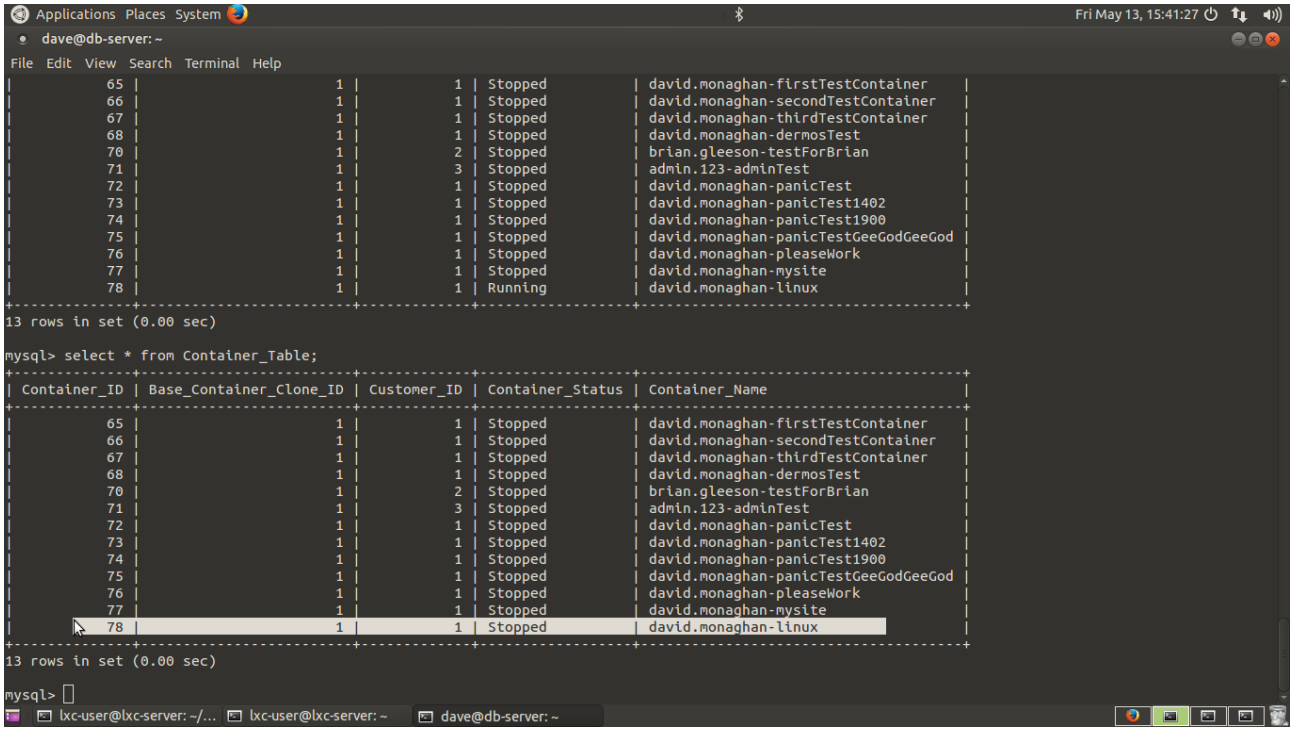


Figure 125: Overview.html showing a backed up container

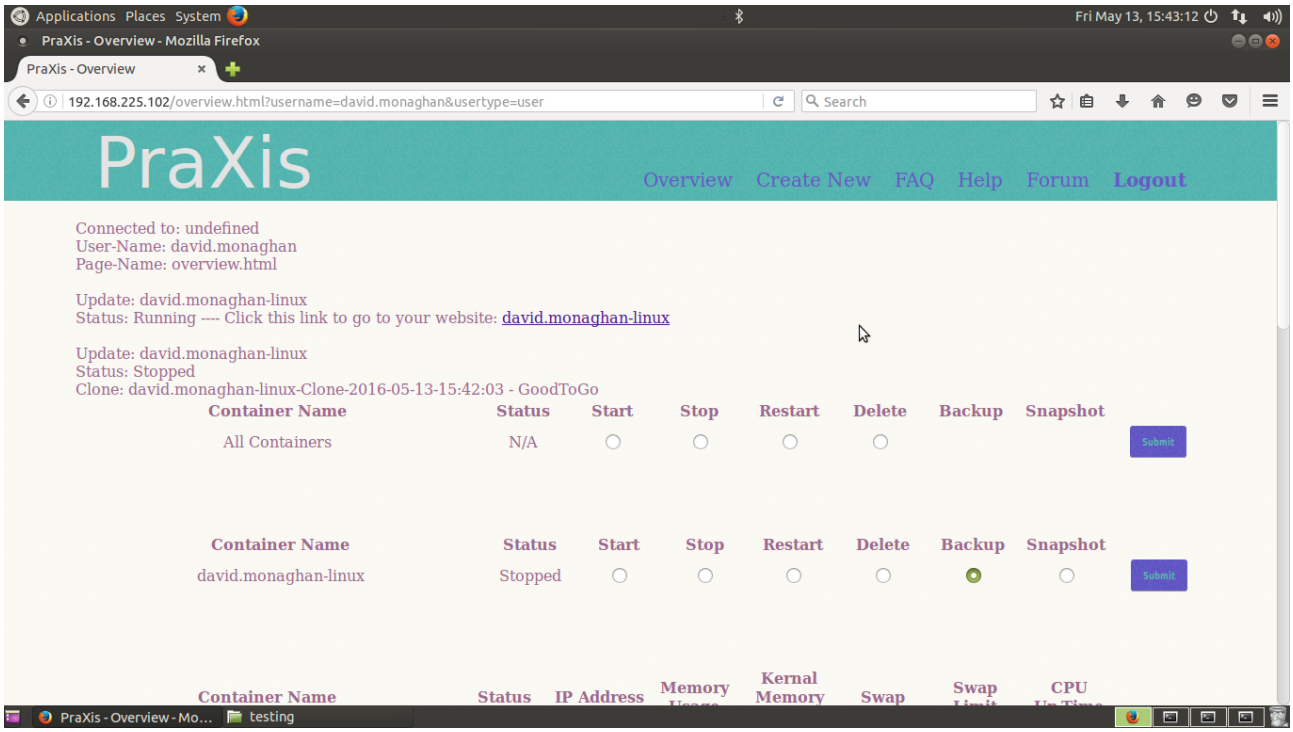


Figure 126: LXC-Control-Server showing a backed up container

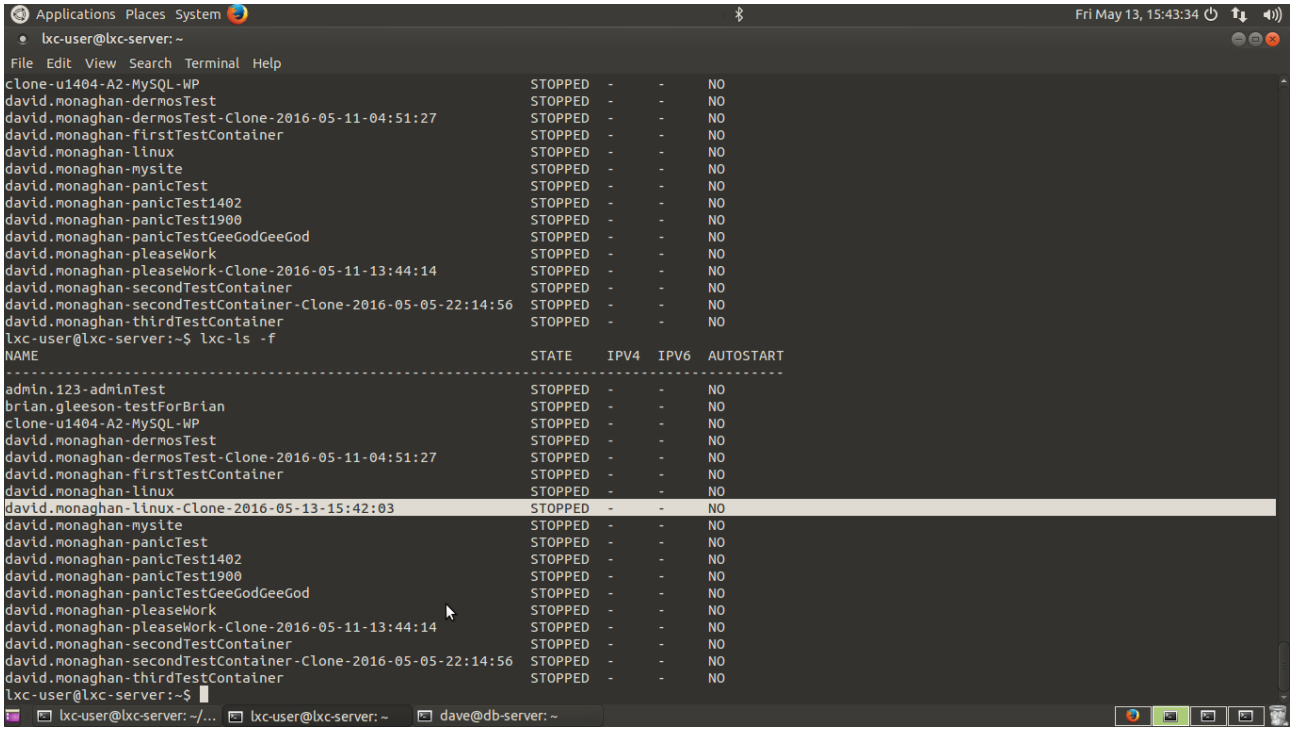


Figure 127: Database-Control-Server showing a stopped container

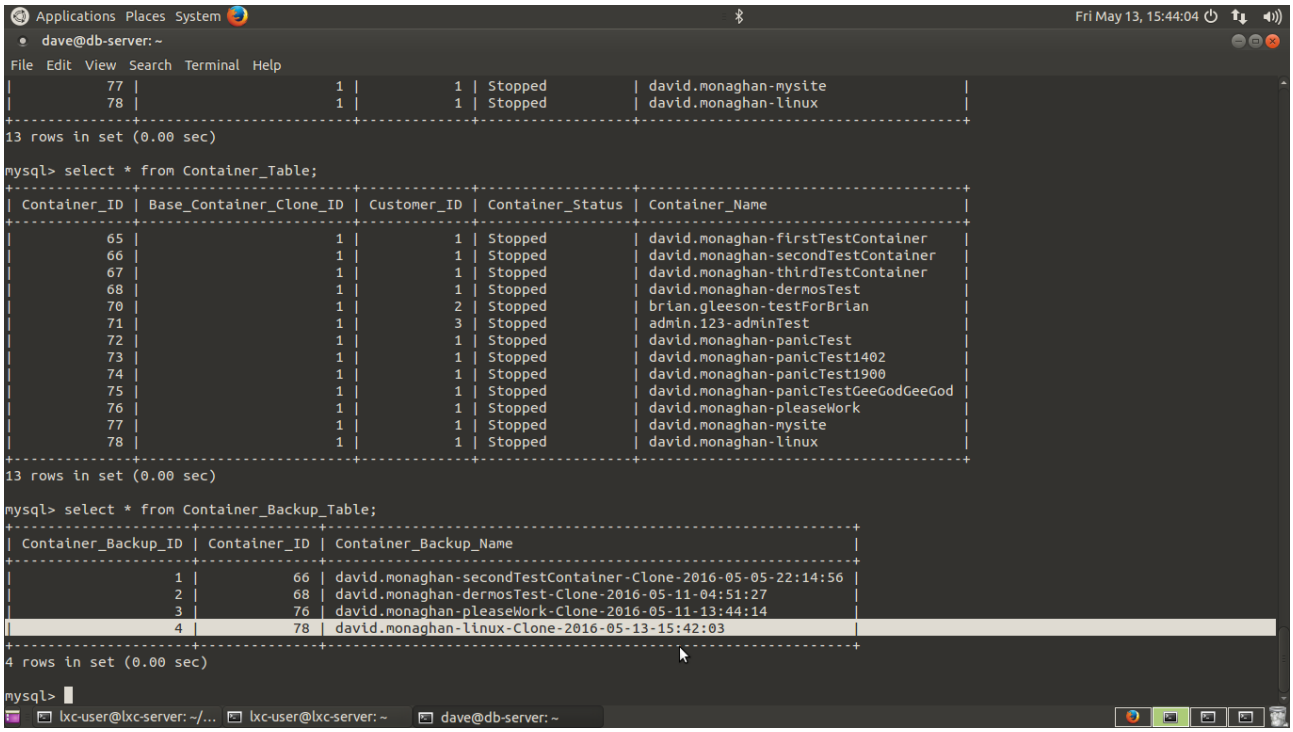
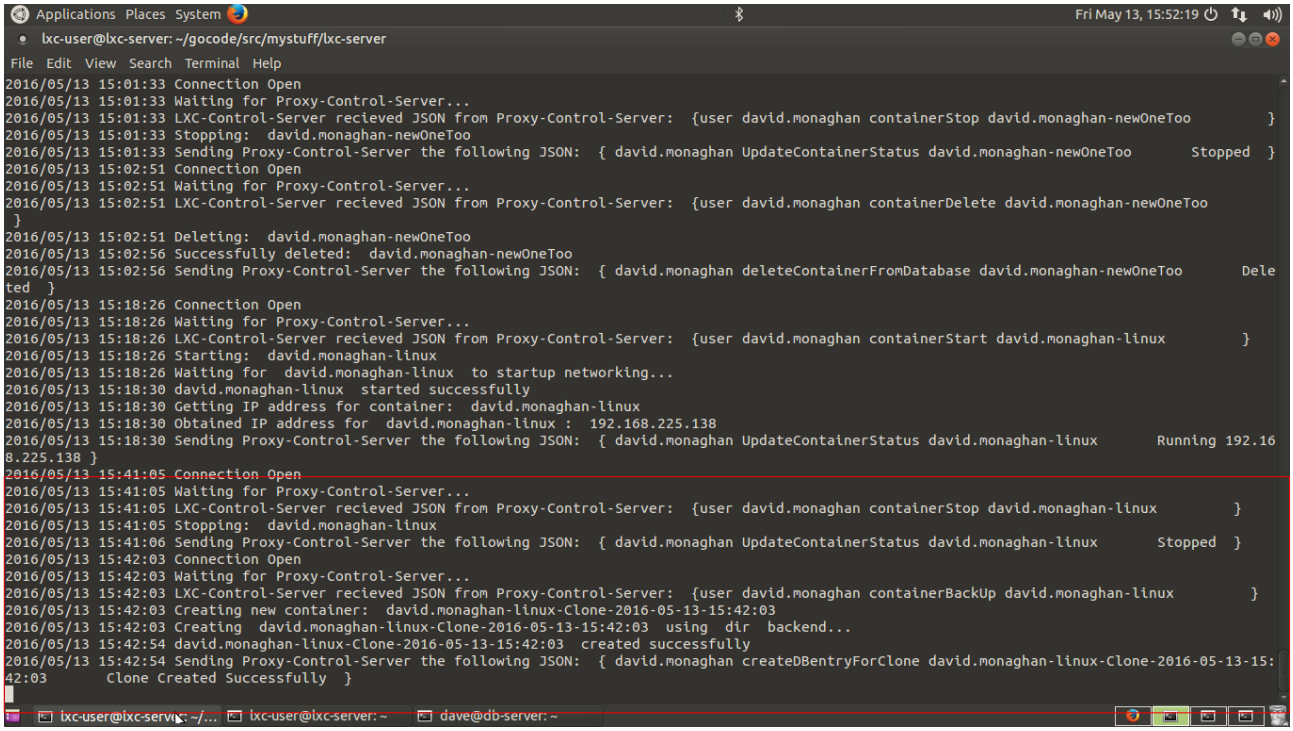


Figure 128: LXC-Control-Server output after container has been backed up



6.2.3.7 – Snapshotting a Container

Figure 129: Overview.html showing error message when snapshotting a running container

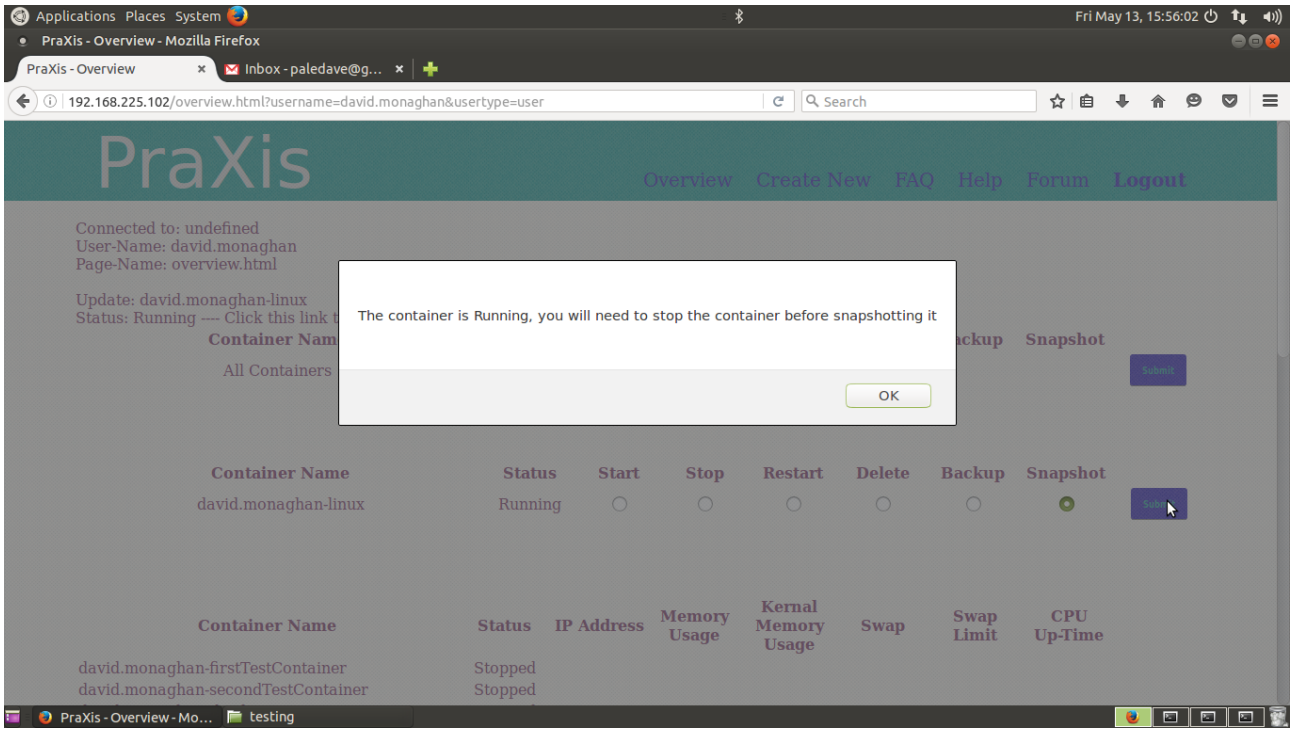


Figure 130: LXC-Control-Server showing a running container

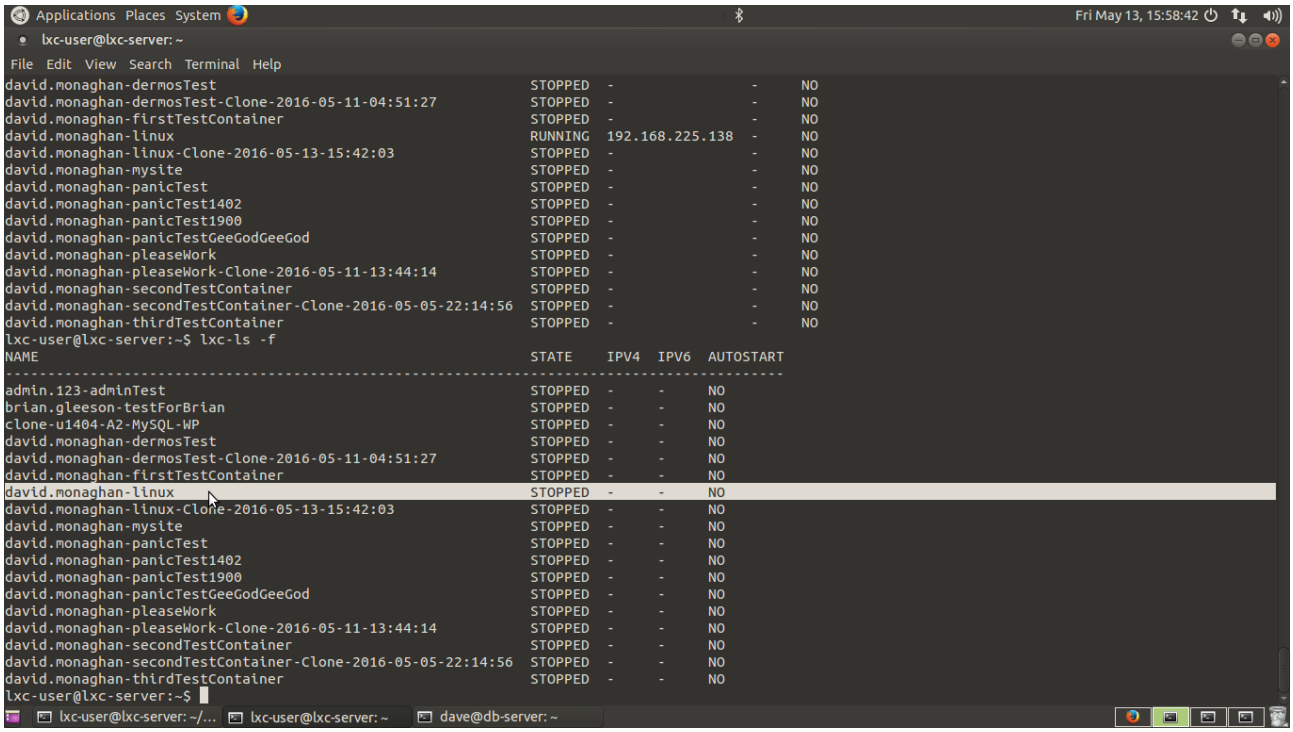


Figure 131: Database-Control-Server showing a running container

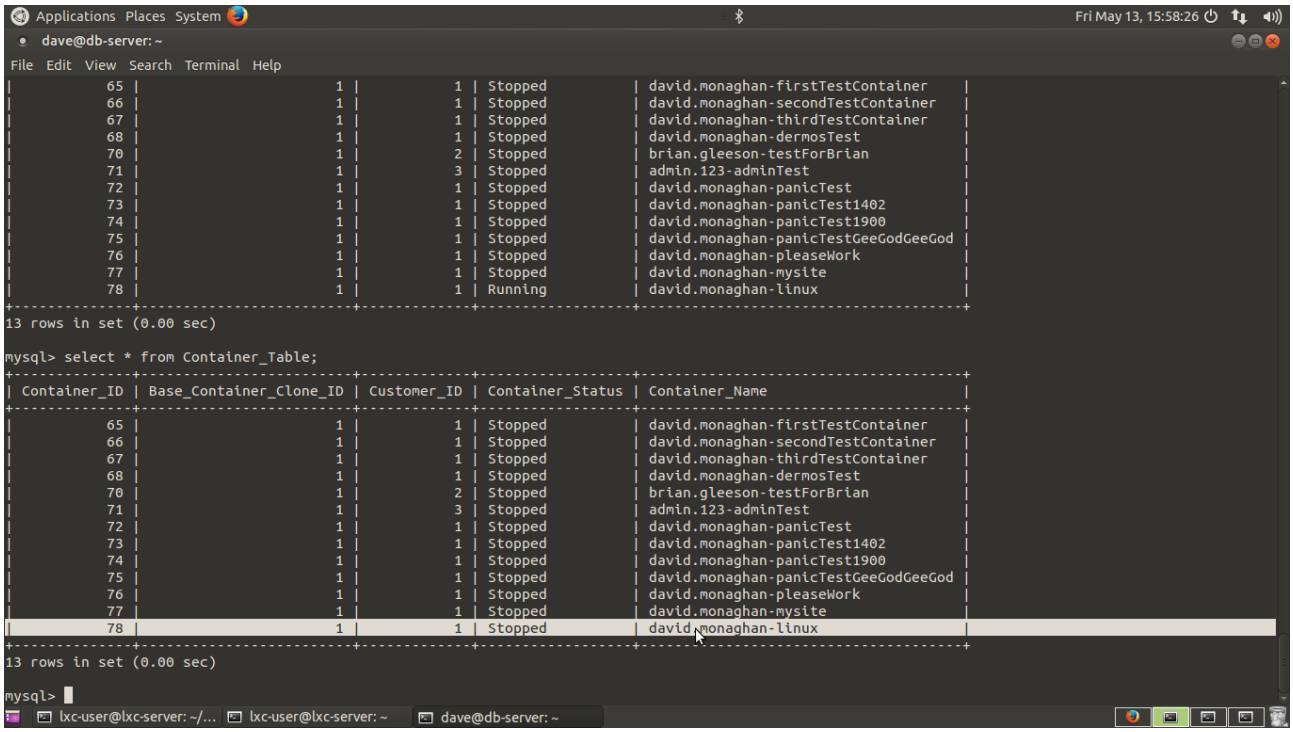


Figure 132: Overview.html showing a stopped container

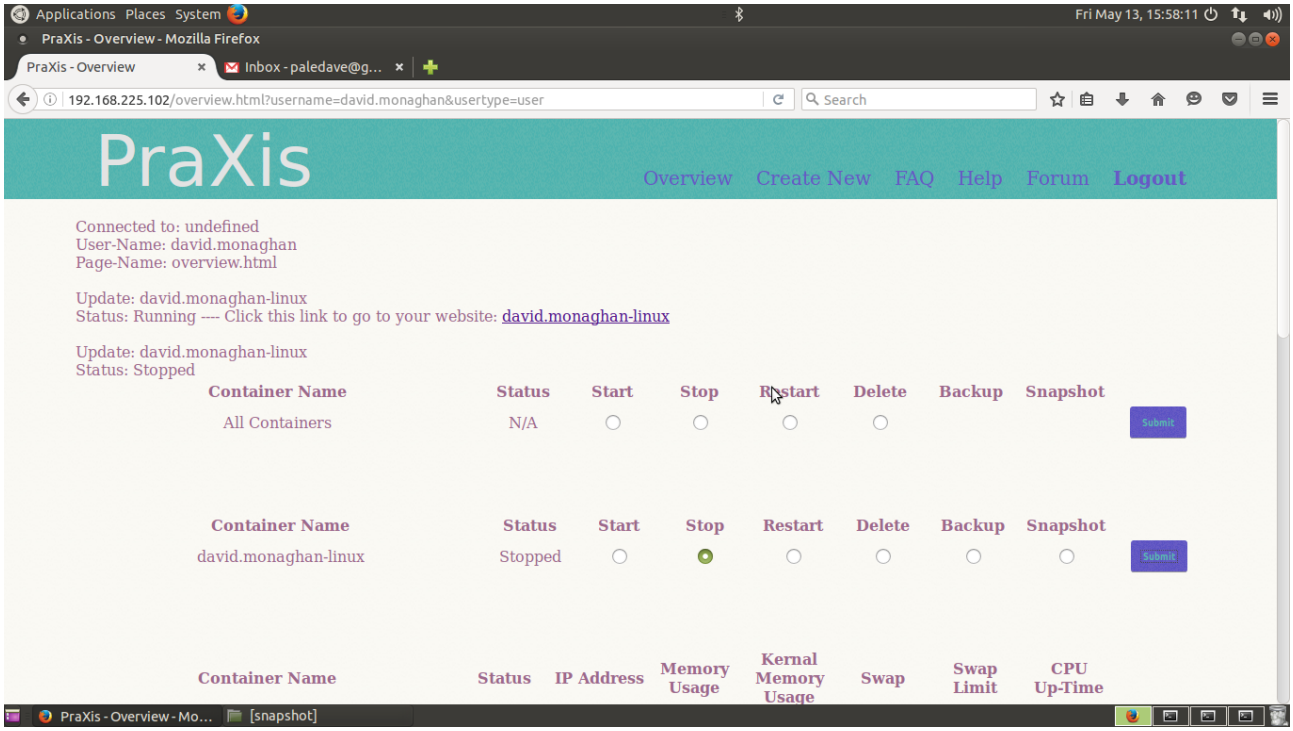


Figure 134: LXC-Control-Server showing a stopped container

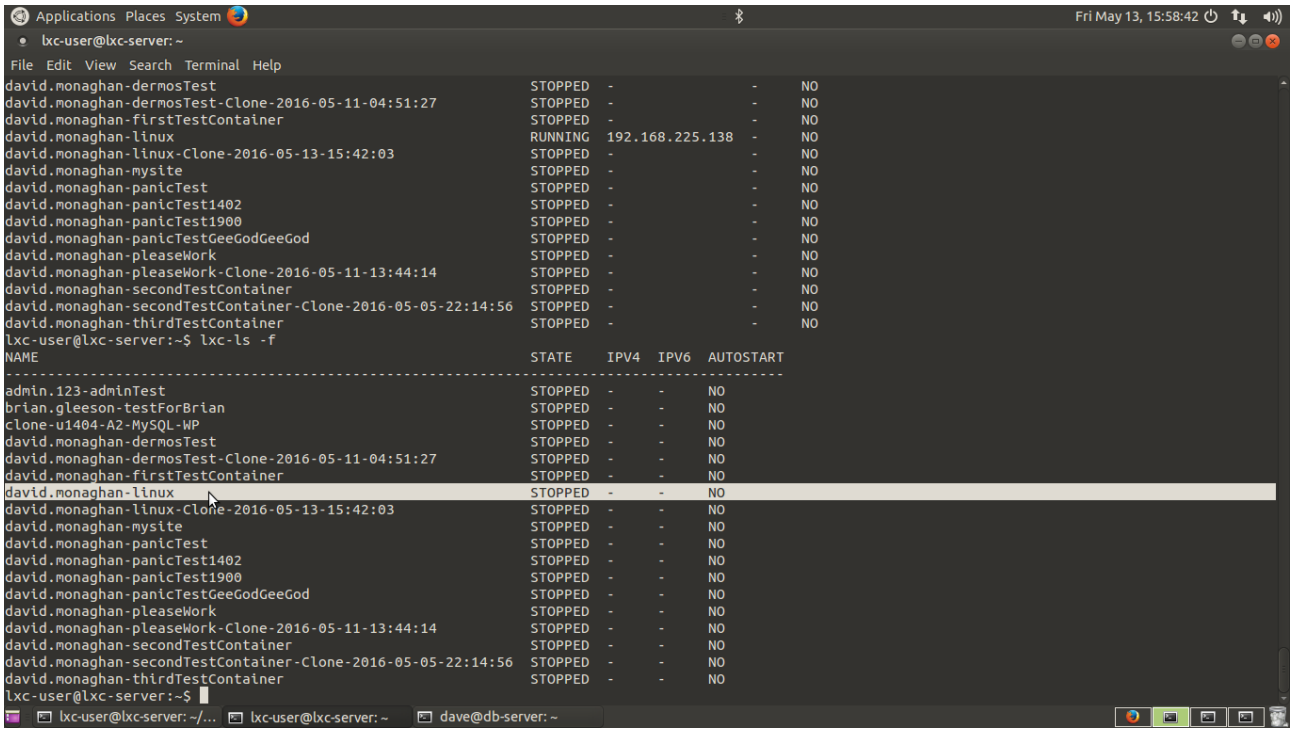


Figure 135: Database-Control-Server showing a stopped container

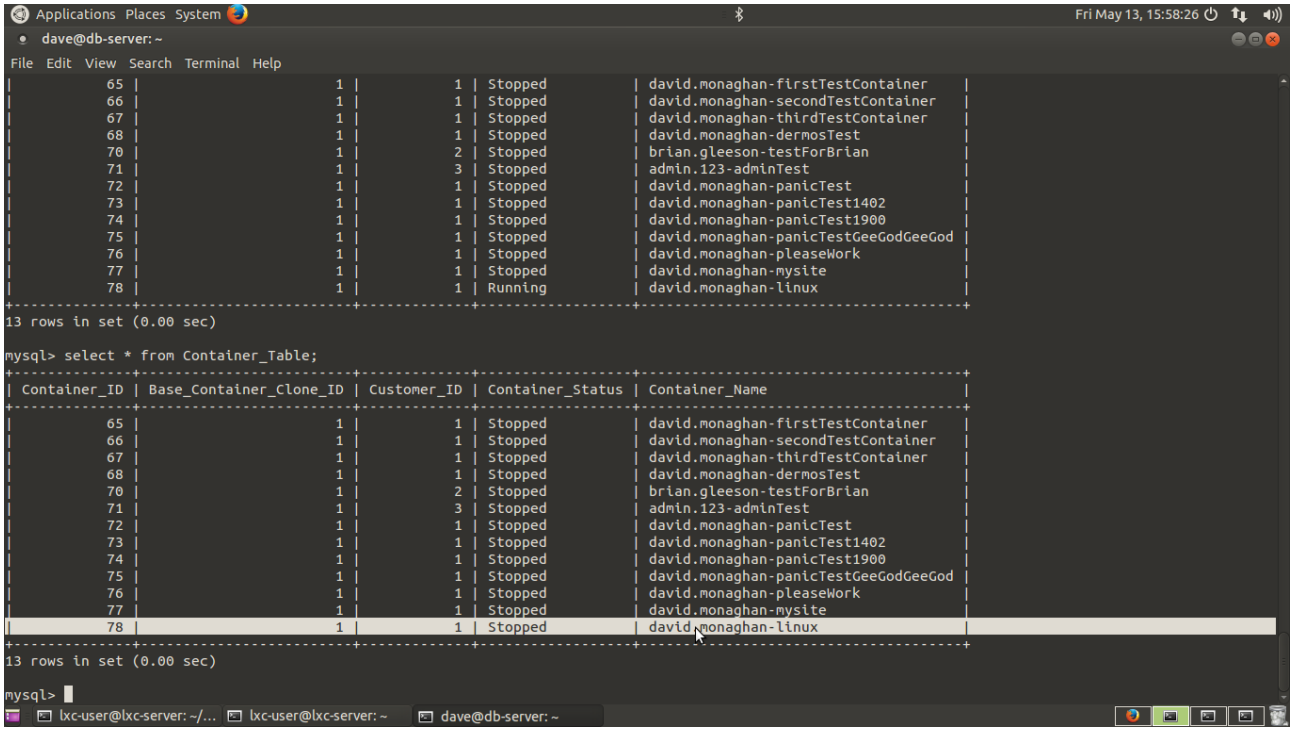


Figure 136: Overview.html showing a snapshotted container

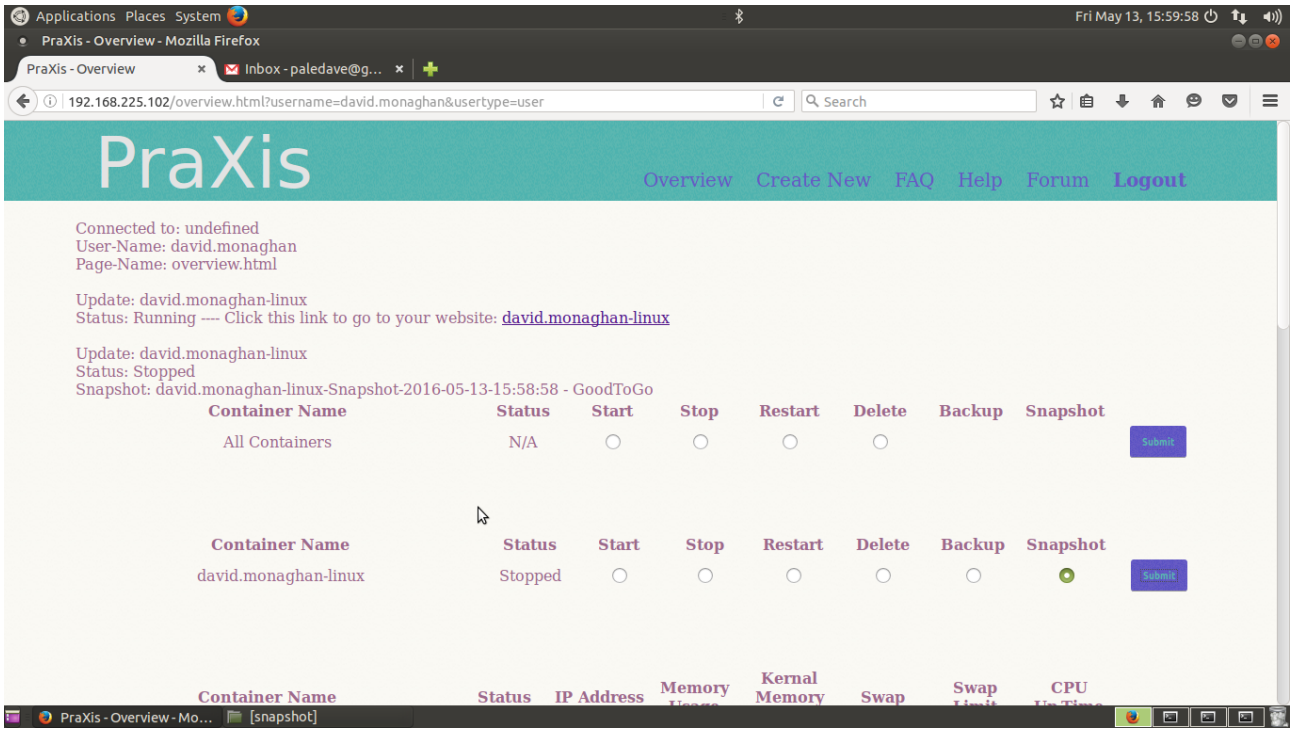


Figure 138: LXC-Control-Server showing a snapshotted container

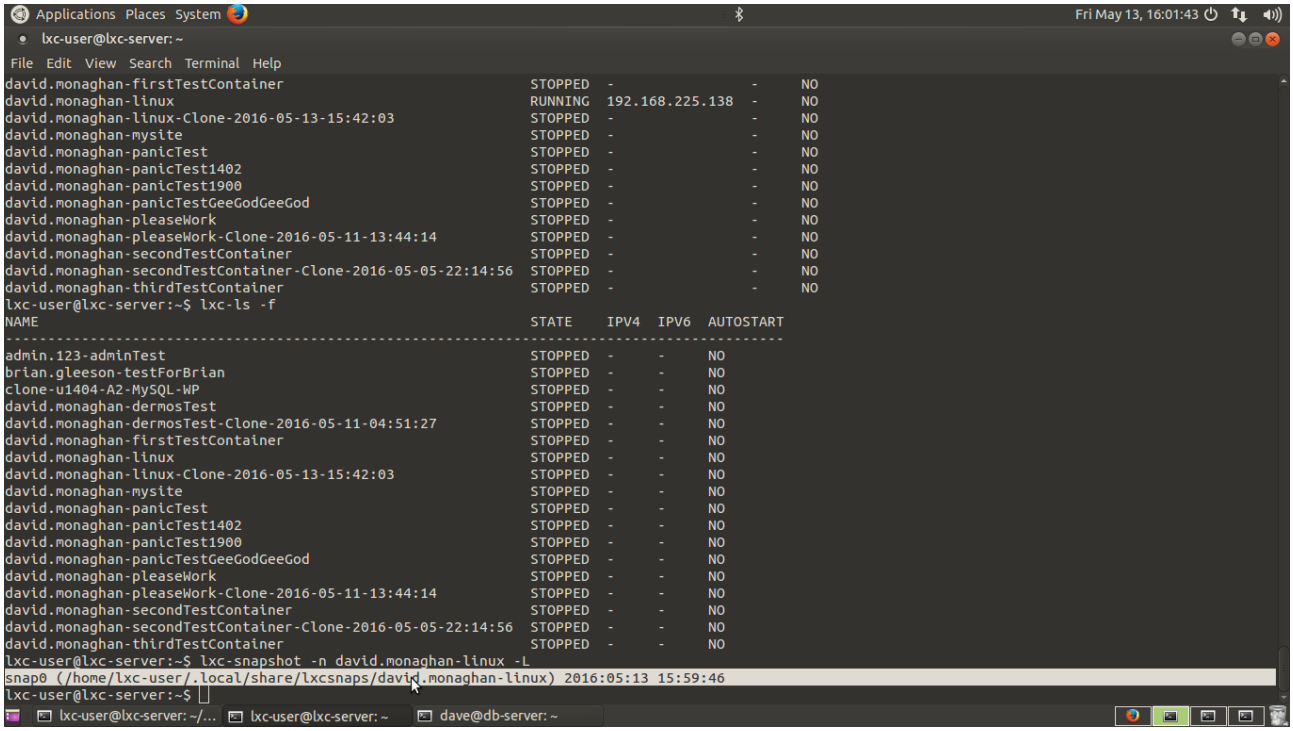


Figure 139: Database-Control-Server showing a snapshotted container

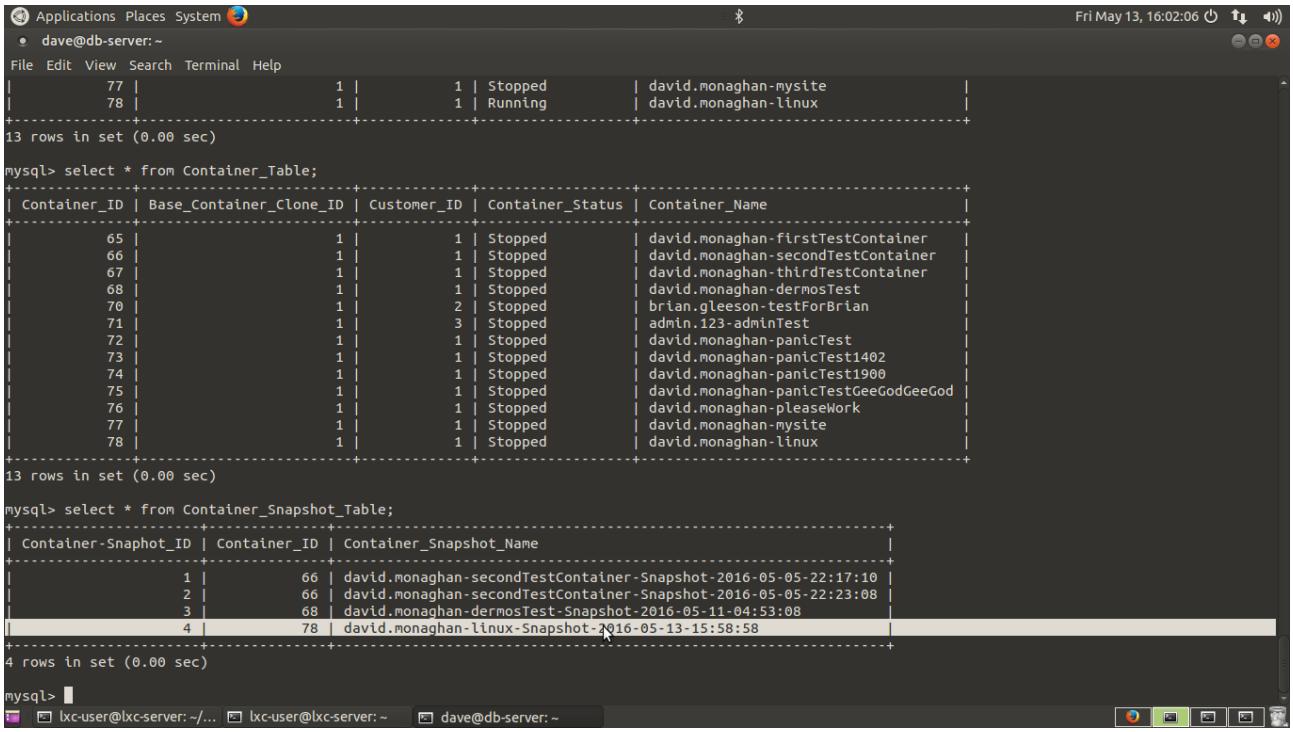
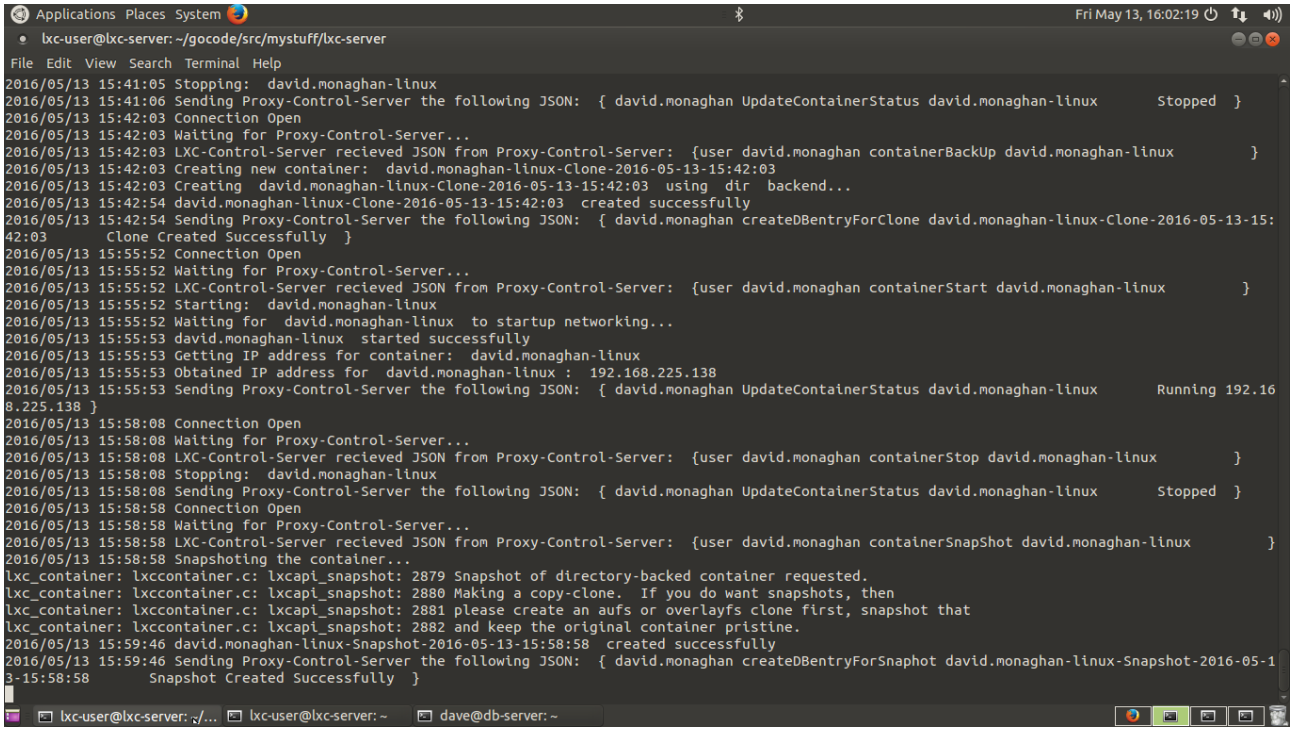


Figure 140: LXC-Control-Server output after container has been snapshotted



6.2.4 – Testing Create_container.html

Figure 141: Create_container.html showing error message as no fields have been filled outlined

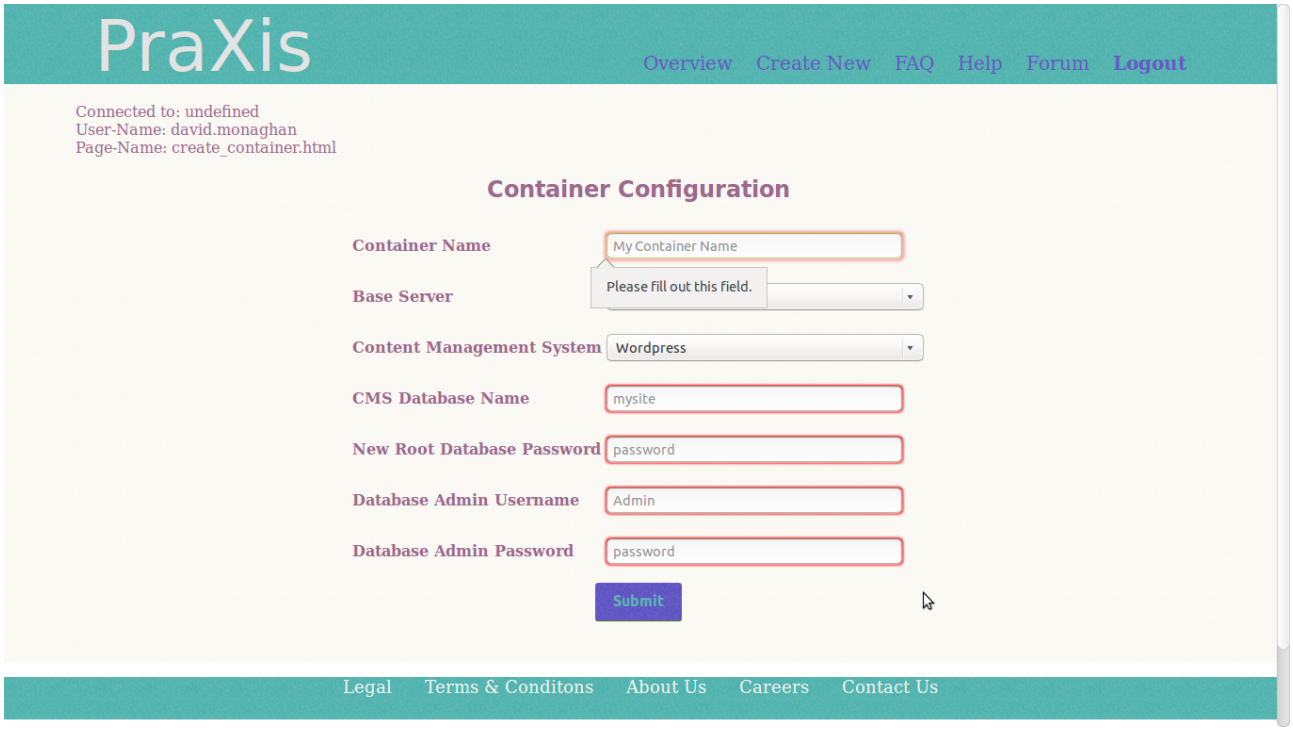


Figure 142: LXC-Control-Server showing a container named "david.monaghan-documentationTesting" does not already exist

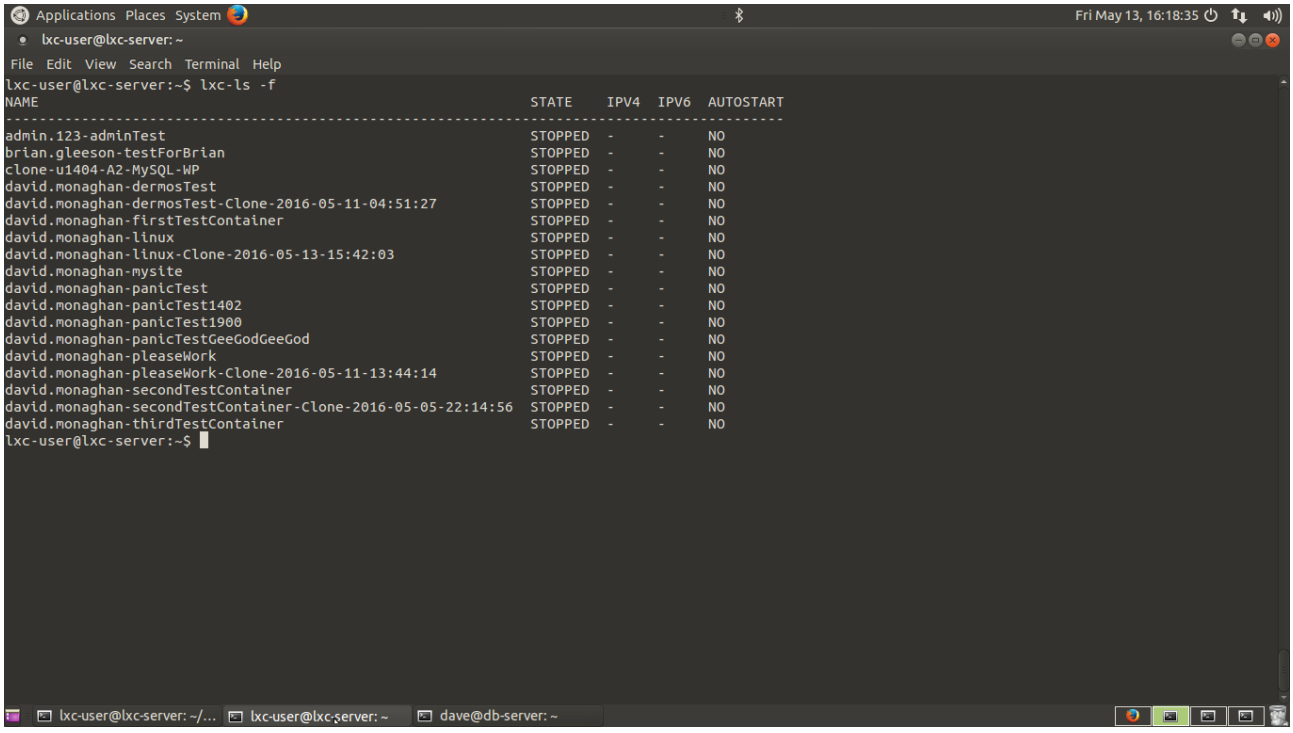


Figure 143: Database-Control-Server showing a container named "david.monaghan-documentationTesting" does not already exist

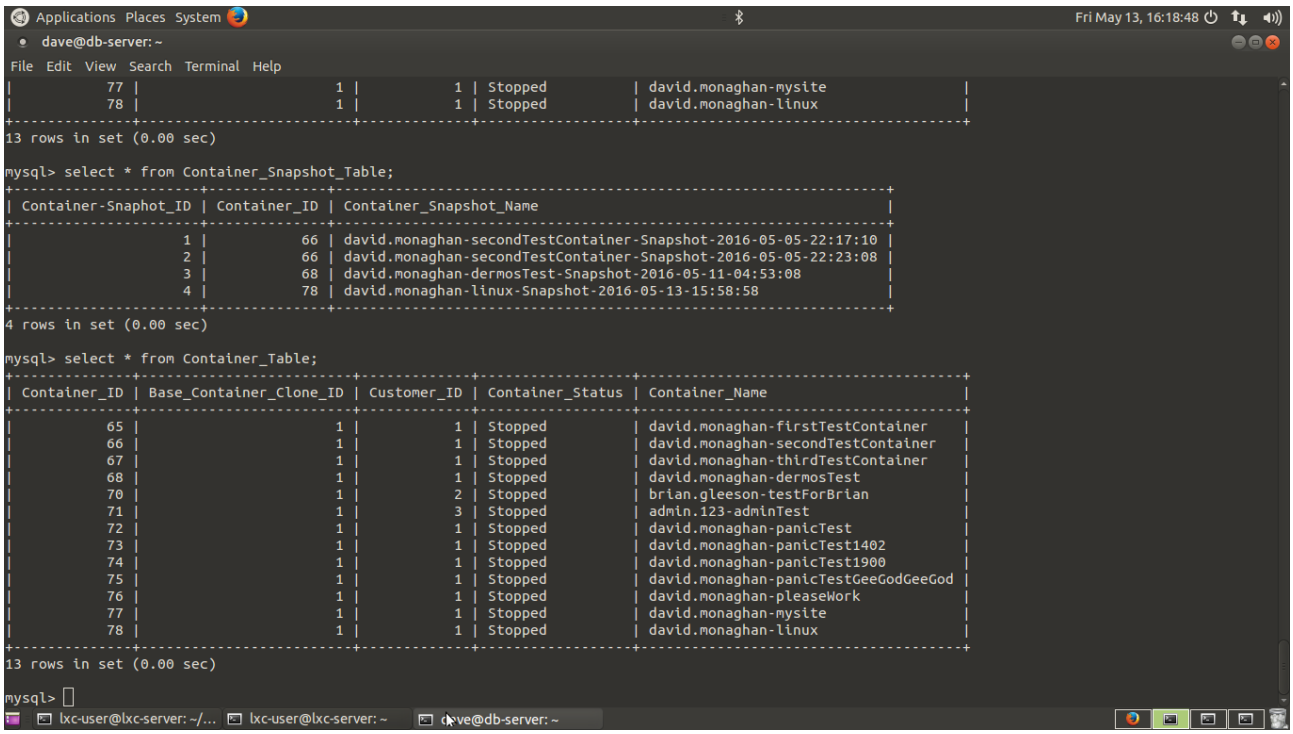


Figure 144: Create_container.html showing a successfully created containers

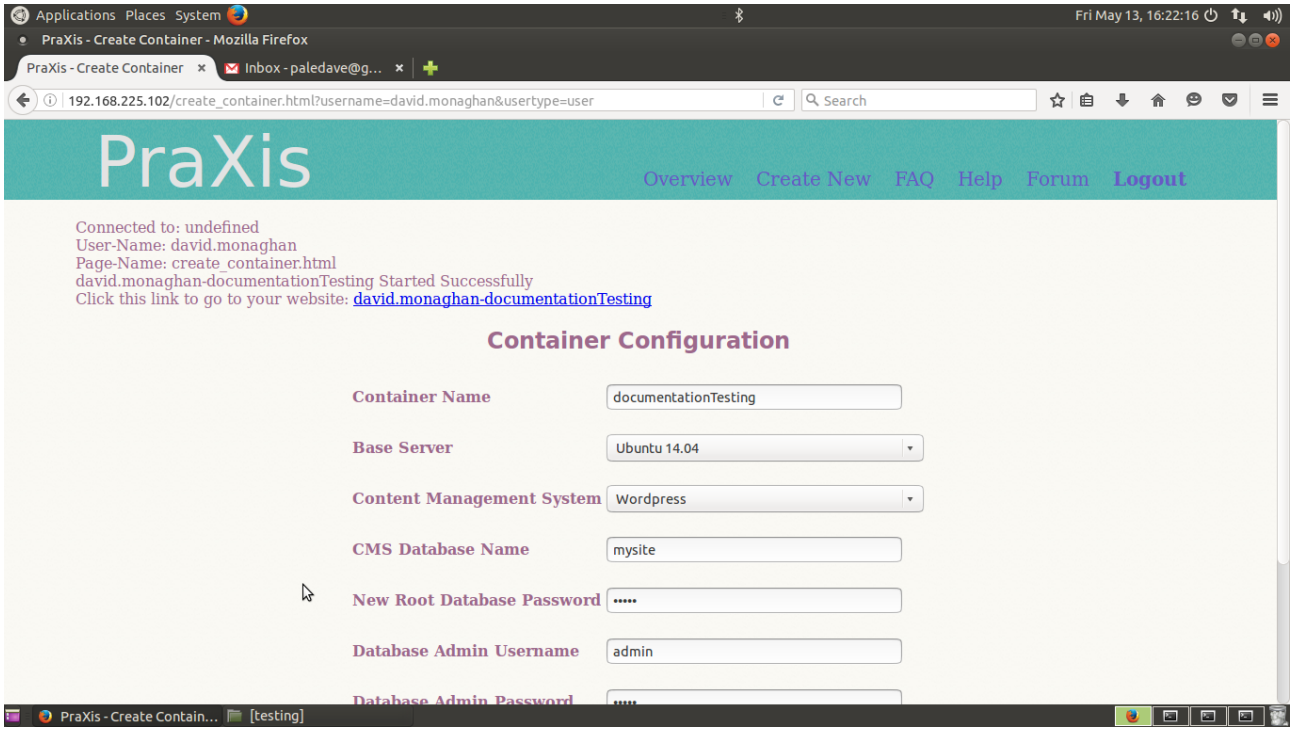


Figure 145: Wordpress running on a newly created container

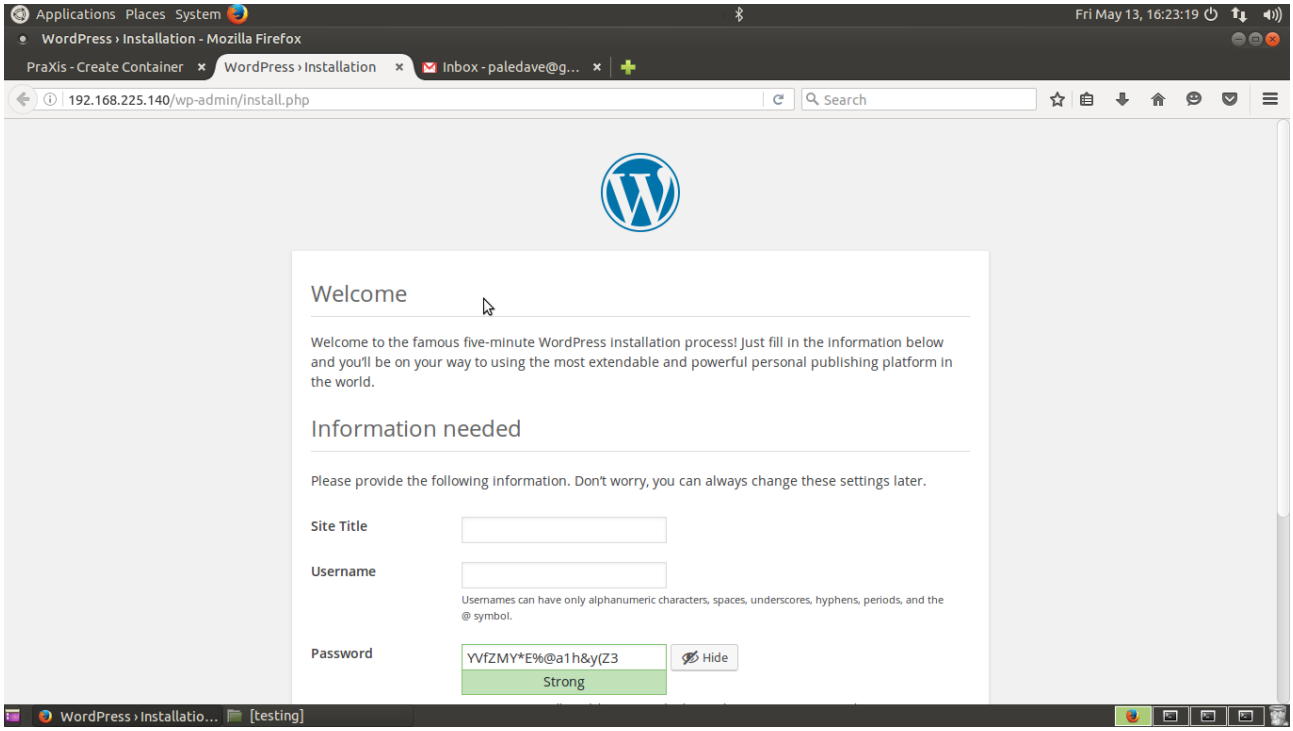


Figure 146: LXC-Control-Server showing the created container

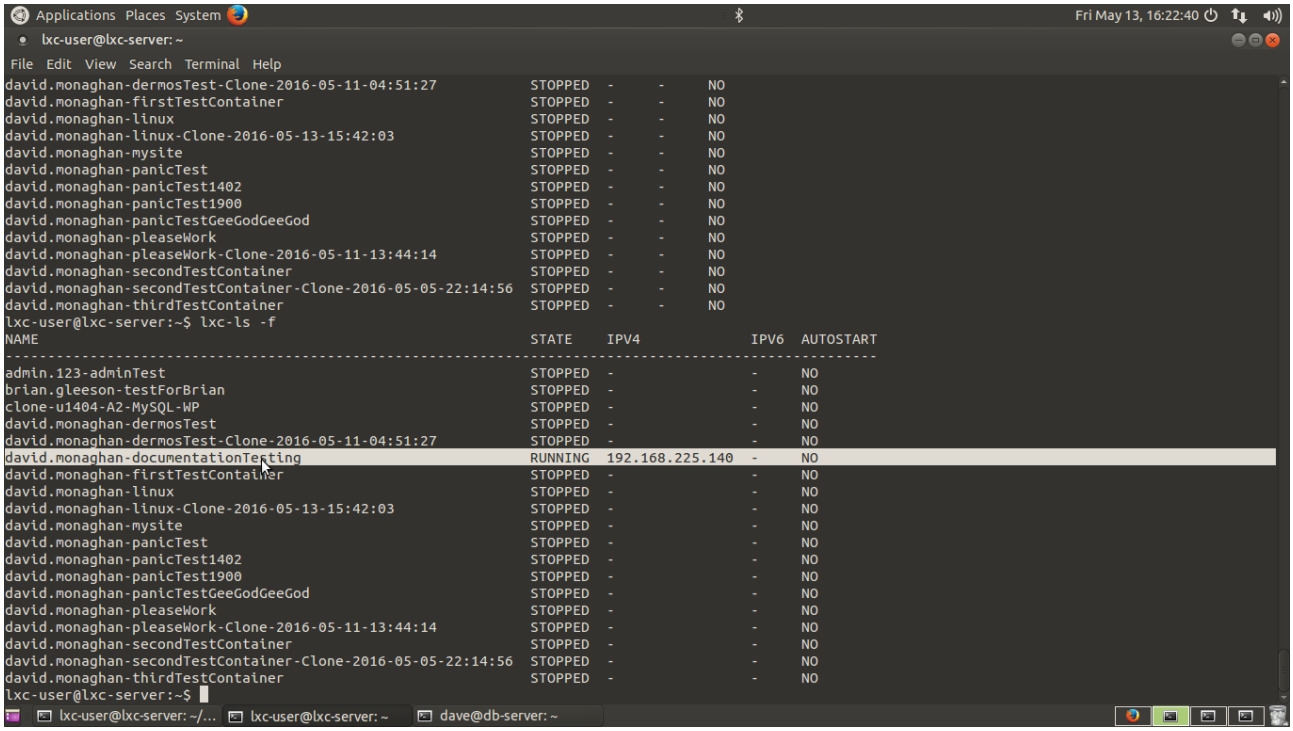


Figure 147: Database-Control-Server output after error message

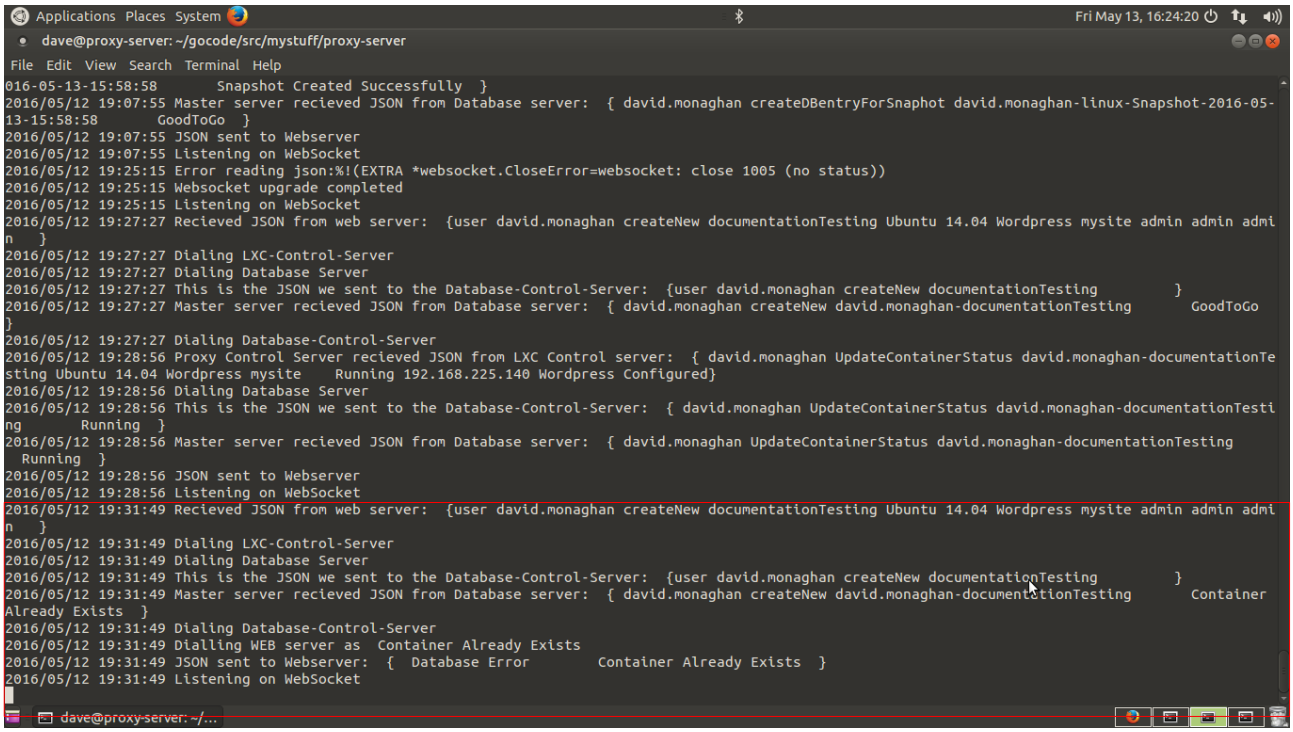


Figure 148: LXC-Control-Server confirming container of same name not added to LXC-Control-Server

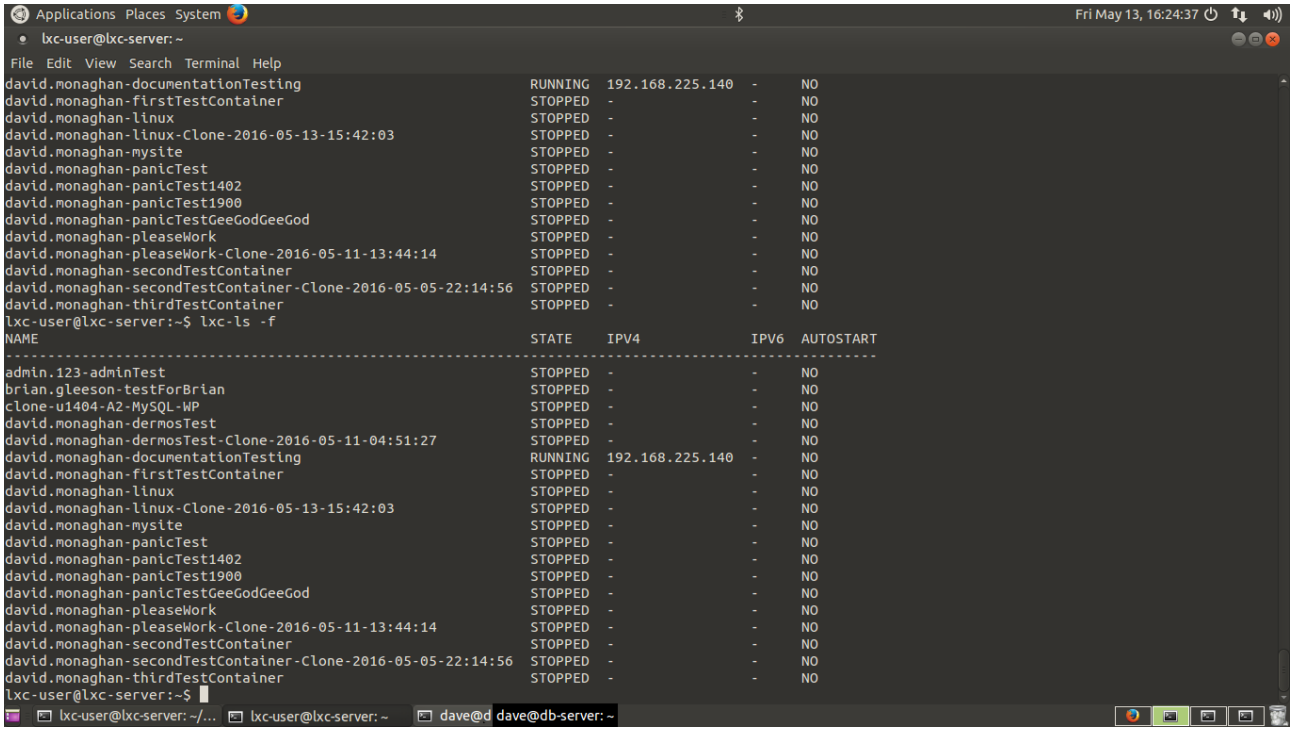


Figure 149: Database-Control-Server confirming container of same name not added to database

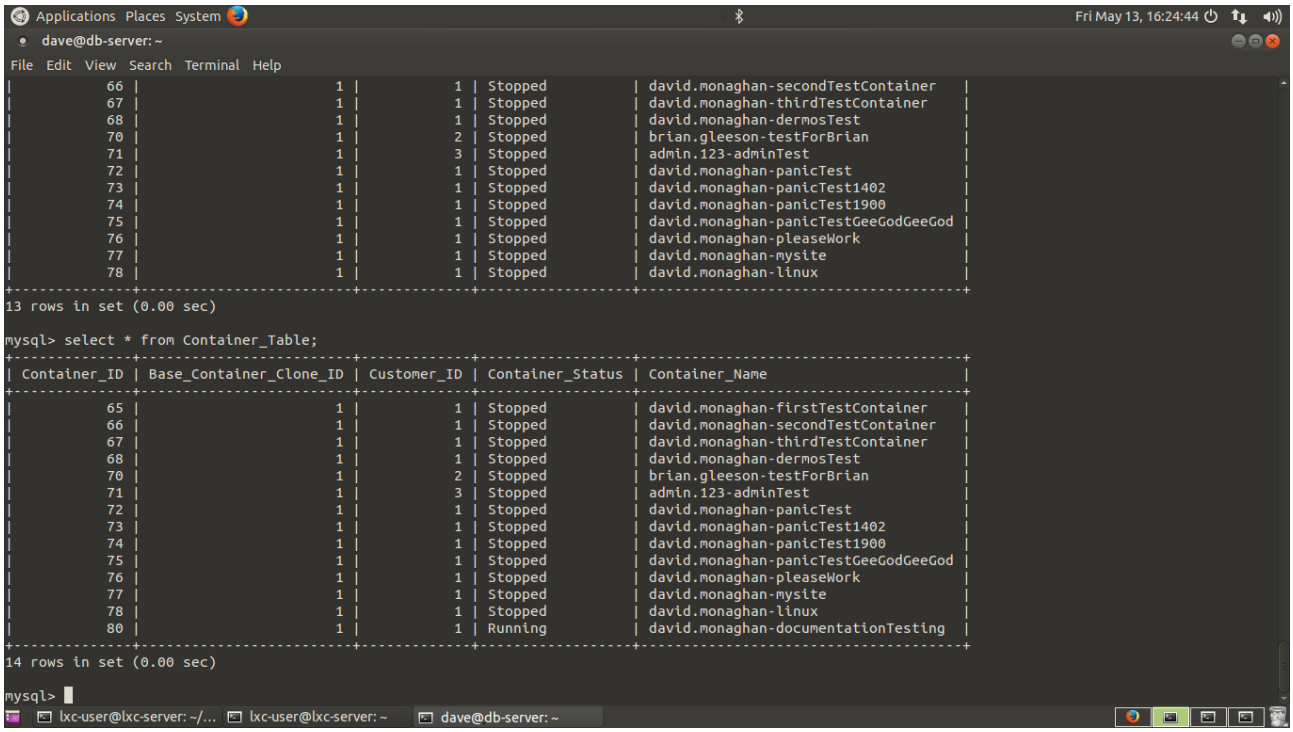
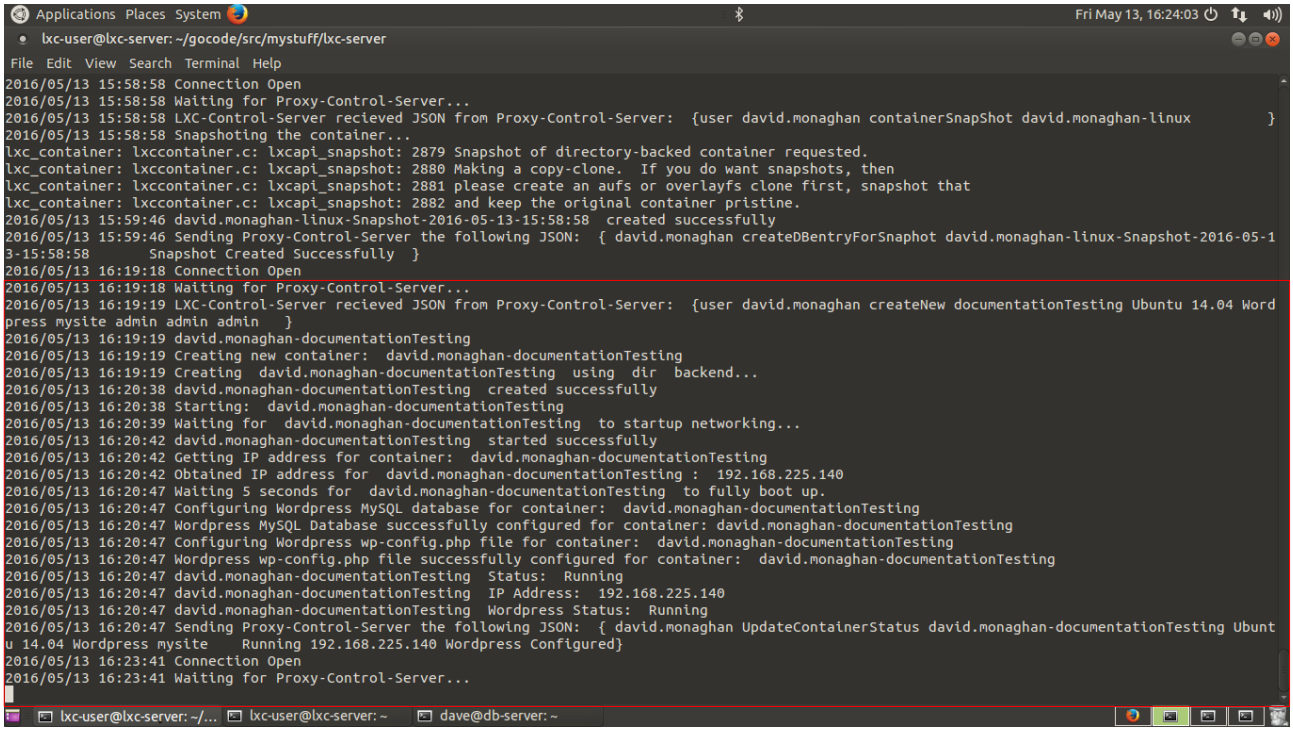


Figure 150: LXC-Control-Server output after container created and error message generated



7 – Conclusion

7.1 – Lessons Learned

1. Never underestimate the time required to complete a software project. The author miscalculated by 2.5 weeks and as such left himself under enormous pressure to deliver final implementation.
2. The initial project plan asserted that the Web-Front-End should be completed before beginning work on the back-end of the service. In hindsight this is a terrible oversight but one made naively. Add this error to the underestimation of the project completion time only increased the pressure upon the author.
3. Full stack development and its inherent difficulties. The author now understands why large software projects are broken into multiple pieces, as the difficulty of managing multiple moving pieces cannot be over-estimated.
4. An appreciation for the work that has gone into designing and engineering the Go programming language. This technology has made the implementation of project easier by many factors. The author would extend his appreciation to the authors of Go.
5. A full understanding of working Linux OS. Until the project conclusion the author had some issues understanding permissions, user right, user land of the Linux environment. With the projects conclusion the author can safely say that he now understands how they work in practice.
6. The final lesson is the hardest one to accept as it is one the author should already learned. Planning and documentation. Planning in the form of the use case diagrams should have been laid out at the beginning of the project, this would have aided in the design headaches that frequently cropped during development. Finally, documentation should have been completed incrementally over the course of the project.

7.2 – Further Improvements

Overall the project has to be considered to be a success. Of the 47 requirements laid out in the research phase, only two where not achieved, namely being to restore a backup or a snapshot of customers container. The testing shows that under most normal conditions that the service can avoid catastrophic failure. This is not to claim that there are no issues with the service such as it not being able to automatically recover if a server where to go down. Areas for improvement include but are not limited to:

1. Using TLS on the Web-Sockets and Network Sockets
2. Using HTTPS for the Web-Servers
3. Creating an individual LXC-User for each customer to aid in further isolating containers from each other
4. Redesigning the Proxy-Control-Server to establish a single Network Socket with each of the other back-end servers. This would prevent the server side of the connection from locking the Port due to ungraceful disconnect.
5. Redesigning the Proxy-Control-Server to use channels to communicate between Go routines would allow for more efficient use of concurrency
6. Re-factoring of the code base will be required as there is currently a lot of repetition.
7. Re-factoring would also aid the goal of a Service Oriented Architecture

The improvements listed above are just for this stage of the project and is likely to change to further as development continues

8 – Appendix

8.1 – Bibliography

1. Graber S. LXC 1.0: Your first Ubuntu container [1/10] | Stéphane Graber's website [Internet]. [cited 2016 May 13]. Available from: https://www.stgraber.org/2013/12/20/lxc-1-0-your-first-ubuntu-container/

2. The lxc-devel Archives [Internet]. [cited 2016 May 13]. Available from: https://lists.linuxcontainers.org/pipermail/lxc-devel/

3. The lxc-users Archives [Internet]. [cited 2016 May 13]. Available from: https://lists.linuxcontainers.org/pipermail/lxc-users/

4. How to create unprivileged LXC container on Ubuntu Linux 14.04 LTS [Internet]. [cited 2016 May 4]. Available from: http://www.cyberciti.biz/faq/how-to-create-unprivileged-linux-containers-on-ubuntu-linux/

5. How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu 14.04 [Internet]. DigitalOcean. [cited 2016 Apr 1]. Available from: https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-14-04

6. Working with unprivileged and privileged containers in Ubuntu 14.04 and LXC – geeklee's Notes [Internet]. [cited 2016 Apr 1]. Available from: http://www.geeklee.co.uk/unprivileged-privileged-containers-ubuntu-14-04-lxc/

7. ethereum/go-ethereum [Internet]. GitHub. [cited 2016 Mar 2]. Available from: https://github.com/ethereum/go-ethereum

8. Go database/sql tutorial [Internet]. [cited 2016 Apr 28]. Available from: http://go-database-sql.org/

9. go-sql-driver/mysql [Internet]. GitHub. [cited 2016 Apr 28]. Available from: https://github.com/go-sql-driver/mysql

10. Price L. Setting Up a Go Environment in Ubuntu - Larry Price [Internet]. [cited 2016 Mar 2]. Available from: https://larry-price.com/blog/2013/12/15/setting-up-a-go-environment-in-ubuntu-12-dot-04/

11. Building Services in Go - YouTube [Internet]. [cited 2016 Mar 3]. Available from: https://www.youtube.com/watch?v=MeOK1UzGHYw

12. Ubuntu. Certificates [Internet]. help.ubuntu.com. 2016 [cited 2016 Mar 2]. Available from: https://help.ubuntu.com/lts/serverguide/certificates-and-security.html

13. Creating Web Applications with Go | Pluralsight [Internet]. [cited 2016 Mar 28]. Available from: https://www.pluralsight.com/courses/creating-web-applications-go

14. NG A. Golang : Secure(TLS) connection between server and client - SocketLoop [Internet]. socketloop.com. 2015 [cited 2016 Mar 2]. Available from: https://www.socketloop.com/tutorials/golang-secure-tls-connection-between-server-and-client

15. NG A. Golang : Simple client server example - SocketLoop [Internet]. socketloop.com. 2015 [cited 2016 Mar 2]. Available from: https://www.socketloop.com/tutorials/golang-simple-client-server-example

16. How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu 14.04 [Internet]. DigitalOcean. [cited 2016 May 4]. Available from: https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-14-04

17. How To Install Wordpress on Ubuntu 14.04 [Internet]. DigitalOcean. [cited 2016 May 4]. Available from: https://www.digitalocean.com/community/tutorials/how-to-install-wordpress-on-ubuntu-14-04

18. Ubuntu. HTTPD - Apache2 Web Server [Internet]. help.ubuntu.com. 2015 [cited 2016 Mar 2]. Available from: https://help.ubuntu.com/lts/serverguide/httpd.html

19. OpenSSH Server [Internet]. [cited 2016 Mar 2]. Available from: https://help.ubuntu.com/14.04/serverguide/openssh-server.html

20. cube2222. Practical Golang: Using websockets [Internet]. Jacob Martin. 2016 [cited 2016 Mar 7]. Available from: https://jacobmartins.com/2016/03/07/practical-golang-using-websockets/

21. arungupta. REST vs WebSocket Comparison and Benchmarks [Internet]. Miles to go 3.0 ... 2014 [cited 2016 Mar 4]. Available from: http://blog.arungupta.me/rest-vs-websocket-comparison-benchmarks/

22. Apache. SSL/TLS Strong Encryption: How-To - Apache HTTP Server Version 2.4 [Internet]. apache.org. 2015 [cited 2016 Mar 2]. Available from: https://httpd.apache.org/docs/2.4/ssl/ssl_howto.html

23. WebSockets vs REST: Understanding the Difference | PubNub [Internet]. [cited 2016 Mar 4]. Available from: https://www.pubnub.com/blog/2015-01-05-websockets-vs-rest-api-understanding-the-difference/(1,1–23)

8.2 – Project Code

8.2.1 – Web-Server – Javascript

```
window.onload = function() {
	// Get Username and User Type from current URL
	var userName = getParameterByName('username');
	var userType = getParameterByName('usertype');

	function getParameterByName(name, url) {
		if (!url) url = window.location.href;
		name = name.replace(/[\[\]]/g, "\\$&");
		var regex = new RegExp("[?&]" + name + "(=([^&#]*)|&|#|$)"),
			results = regex.exec(url);
		if (!results) return null;
		if (!results[2]) return '';
		return decodeURIComponent(results[2].replace(/\+/g, " "));
	}

	// Create a new WebSocket.
	var socket = new WebSocket('ws://192.168.1.101:8080/user');
	var socketStatus = document.getElementById('status');
	var currentPage = location.pathname.substring(location.pathname.lastIndexOf("/") + 1);

	// Handle any errors that occur.
	socket.onerror = function(error) {
		console.log('WebSocket Error: ' + error);
	};

	// Show a disconnected message when the WebSocket is closed.
	socket.onclose = function(event) {
		socketStatus.innerHTML = 'Disconnected from WebSocket.';
		socketStatus.className = 'closed';
	};

	// Close Web-Socket on Logout
	document.getElementById('logOutHref').onclick = function() {
		socket.close();
	};

	// Show a connected message when the WebSocket is opened.
	socket.onopen = function(event) {
		socketStatus.innerHTML = 'Connected to: ' + event.currentTarget.URL +
                                                                '</br>User-Name: ' +
userName +
                                                                '</br>Page-Name: ' +
currentPage + '</br>';

		if (currentPage == "overview.html") {
			var messageJSON = {
				"CustomerType": userType,
				"CustomerUname": userName,
				"Action": "getListOfContainers"
			}

			var json = JSON.stringify(messageJSON);

			// Send the message through the WebSocket.
```

```
                        socket.send(json);
                }
        };

        // Handle messages sent by the server.
        socket.onmessage = function(event) {
                var message = event.data;

                var serverResponse = JSON.parse(message)

                if (currentPage == "create_container.html") {

                        var containerStatus =  document.getElementById('containerStatus');
                        var containerIPaddress =  document.getElementById('ContainerIPaddress');

                        if ( serverResponse.ContainerStatus == "Container Already Exists" ) {
                                containerStatus.innerHTML = "Error: Container Already Exists";
                        }

                        if ( serverResponse.ContainerIPaddress != null )        {
                                  socketStatus.innerHTML += serverResponse.ContainerName + ' Started
Successfully </br>' +
                                                                                                'Click this
link to go to your website: <a href="http://' +
                                                                                            serverRespon
se.ContainerIPaddress +
                                                                                                    '"
target="_blank">' + serverResponse.ContainerName + '</a></br>';
                        }

                        if ( serverResponse.ContainerStatus == "Error Dialing Database Server" ) {
                                                socketStatus.innerHTML  +=  '</br>Error:  '  +
serverResponse.ContainerStatus;
                        }

                        if ( serverResponse.Action == "Database Error" ) {
                                                socketStatus.innerHTML  +=  '</br>Error:  '  +
serverResponse.ContainerStatus;
                        }

                        if ( serverResponse.Action == "LXC Server Dial Error" ) {
                                                socketStatus.innerHTML  +=  '</br>Error:  '  +
serverResponse.ContainerStatus;
                        }
                }

                var userContainerName = document.getElementById('userContainerName');
                var userContainerStatus = document.getElementById('userContainerStatus');
                var containerList = document.getElementById('containerForm-3');

                if (currentPage == "overview.html") {

                        var messagesList = document.getElementById('messages');
                        messagesList.innerHTML += '<li class="received"><span>Received:</span>' +
                                                                        message + '</li>';

                        if ( serverResponse.Action == "getListOfContainers" ) {
                                    if ( serverResponse.ContainerStatus != "Error Dialing Database
Server" ) {
```

```
                                    userContainerName.innerHTML = serverResponse.ContainerName;

                                                                    userContainerStatus.innerHTML  =
serverResponse.ContainerStatus;

                                    containerList.innerHTML  += '<table><tr>' + '<td align="left"
width="425px">' +

serverResponse.ContainerName + '</td> ' +

'<td align="center" width="100px">' + serverResponse.ContainerStatus + '</td>' +

'<td align="center" width="100px"></td>' +

'<td align="center" width="100px"></td>' +

'<td align="center" width="100px"></td>' +

'<td align="center" width="100px"></td>' +

'<td align="center" width="100px"></td>' +

'<td align="center" width="100px"></td>' +

'<td align="center" width="100px"></td></tr></table>';
                                } else {
                                                        socketStatus.innerHTML  += '</br>Error' +
serverResponse.ContainerName + ': ' +
                                                                                                serv
erResponse.ContainerStatus;
                                }
                        }

                        if ( serverResponse.Action == "Database Error" ) {
                                                socketStatus.innerHTML  +=  '</br>Error:  '  +
serverResponse.ContainerStatus;
                        }

                        if ( serverResponse.Action == "LXC Server Dial Error" ) {
                                                socketStatus.innerHTML  +=  '</br>Error:  '  +
serverResponse.ContainerStatus;
                        }

                        if ( serverResponse.Action == "createDBentryForClone" ) {
                                socketStatus.innerHTML += 'Clone: ' + serverResponse.ContainerName +
' - ' + serverResponse.ContainerStatus;
                        }

                        if ( serverResponse.Action == "createDBentryForSnaphot" ) {
                                                  socketStatus.innerHTML  +=  'Snapshot:  '  +
serverResponse.ContainerName + ' - ' + serverResponse.ContainerStatus;
                        }

                        if ( serverResponse.Action == "UpdateContainerStatus" ) {
                                if ( serverResponse.ContainerIPaddress != null ) {
                                                socketStatus.innerHTML  += "</br>Update: " +
serverResponse.ContainerName +
                                                                                                "</b
```

```
r>Status:  " + serverResponse.ContainerStatus +
                                                                                '
---- Click this link to go to your website: <a href="http://'
                                                                                +
serverResponse.ContainerIPaddress +
                                                                                '"
target="_blank">' + serverResponse.ContainerName + '</a></br>';


                                                                userContainerStatus.innerHTML  =
serverResponse.ContainerStatus;
                                }

                                if ( serverResponse.ContainerIPaddress == null ) {
                                                socketStatus.innerHTML  += "</br>Update:  " +
serverResponse.ContainerName +

"</br>Status:  " + serverResponse.ContainerStatus + '</br>';

                                                                userContainerStatus.innerHTML  =
serverResponse.ContainerStatus;
                                }
                        }

                        if ( serverResponse.Action == "deleteContainerFromDatabase" ) {
                                                socketStatus.innerHTML  += "</br>Update:  " +
serverResponse.ContainerName +
                                                                                "</br>Status
:  " + serverResponse.ContainerStatus + '</br>';

                                userContainerStatus.innerHTML = serverResponse.ContainerStatus;
                        }
                }
        };


          // Submit forms and send to Proxy-ControlServer, controls are based on current html page
title
        if (currentPage == "overview.html") {
                // CReate href to include current Username and Usertype
                document.getElementById('createContainerHref').onclick = function() {
                        socket.close();
                                                document.getElementById('createContainerHref').href  =
'create_container.html?username=' + userName + '&usertype=' + userType;
                };

                var overviewFormA = document.getElementById('overview-form-1');
                var userContainerStatus = document.getElementById('userContainerStatus');

                overviewFormA.onsubmit = function(e) {
                        e.preventDefault();

                        var ovFormRdOptsA = document.getElementsByName('cStartStop-1');
                                                                                var    ovFormConNameA    =
document.getElementById('containerForm').rows[2].cells.namedItem("allContainerName").innerHTML;
                        var reqAction;


                        for (var i = 0, length = ovFormRdOptsA.length; i < length; i++) {
```

```
                        if (ovFormRdOptsA[i].checked) {
                                // do whatever you want with the checked radio
                                alert(ovFormRdOptsA[i].value);
                                break;
                        }
                }
                var messageJSON = {
                        "CustomerType": userType,
                        "CustomerUname":userName,
                        "Action": reqAction,
                        "ContainerName": ovFormConNameA
                }

                var json = JSON.stringify(messageJSON);

                // Testing Purposes Only
                var messagesList = document.getElementById('messages');

                messagesList.innerHTML += '<li><span>Sent:</span>' + json + '</li>';

                // Send the message through the WebSocket.
                socket.send(json);

                return false;
        };

        var overviewFormB = document.getElementById('overview-form-2');

        overviewFormB.onsubmit = function(e) {
                e.preventDefault();

                var ovFormRdOptsB = document.getElementsByName('cStartStop-2');
                        var ovFormConNameB = document.getElementById('containerForm-
2').rows[2].cells.namedItem("userContainerName").innerHTML;
                                var currentUserContainerStatus =
document.getElementById('userContainerStatus').innerHTML;
                var reqAction;

                for (var i = 0, length = ovFormRdOptsB.length; i < length; i++) {
                        if (ovFormRdOptsB[i].checked) {
                                // do whatever you want with the checked radio
                                switch ( ovFormRdOptsB[i].value ) {
                                        case "containerStart":
                                                if ( currentUserContainerStatus=="Running" )
{
                                                        alert( "The container is already
running!!!");
                                                return false;
                                                } else {
                                                        alert(ovFormRdOptsB[i].value);
                                                }
                                                break;
                                        case "containerStop":
                                                if ( currentUserContainerStatus=="Stopped" )
{
                                                        alert( "The container is already
stopped!!!");
                                                return false;
                                                } else {
```

```
                                                        alert(ovFormRdOptsB[i].value);
                                                }
                                                break;
                                        case "containerRestart":
                                                if ( currentUserContainerStatus=="Stopped" )
{
                                                        alert( "The container is already
stopped!!!");
                                                        return false;
                                                } else {
                                                        alert(ovFormRdOptsB[i].value);
                                                }
                                                break;
                                        case "containerDelete":
                                                if ( currentUserContainerStatus=="Running" )
{
                                                        alert( "The container is Running, you
will need to stop the container before deleting it");
                                                        return false;
                                                } else {
                                                        alert(ovFormRdOptsB[i].value);
                                                }
                                                break;
                                        case "containerBackUp":
                                                if ( currentUserContainerStatus=="Running" )
{
                                                        alert( "The container is Running, you
will need to stop the container before backing it up");
                                                        return false;
                                                } else {
                                                        alert(ovFormRdOptsB[i].value);
                                                }
                                                break;
                                        case "containerSnapShot":
                                                if ( currentUserContainerStatus=="Running" )
{
                                                        alert( "The container is Running, you
will need to stop the container before snapshotting it");
                                                        return false;
                                                } else {
                                                        alert(ovFormRdOptsB[i].value);
                                                }
                                                break;
                                        default:
                                                alert(ovFormRdOptsB[i].value);
                                }
                                alert(ovFormRdOptsB[i].value);
                                break;
                        }
                }

                var messageJSON = {
                        "CustomerType": userType,
                        "CustomerUname":userName,
                        "Action": ovFormRdOptsB[i].value,
                        "ContainerName": ovFormConNameB
                }

                var json = JSON.stringify(messageJSON);
```

```
                        // Testing Purposes Only
                        var messagesList = document.getElementById('messages');

                        messagesList.innerHTML += '<li><span>Sent:</span>' + json + '</li>';

                        // Send the message through the WebSocket.
                        socket.send(json);
                        return true;
                };
        }

        if (currentPage == "create_container.html") {
                document.getElementById('overViewHref').onclick = function() {
                        socket.close();
                          document.getElementById('overViewHref').href = 'overview.html?username=' +
userName + '&usertype=' + userType;
                };

                // Deal with form values to create a new container
                // Send a message when the form is submitted.
                var newContainerForm = document.getElementById('newContainerForm');
                var cName = document.getElementById('containerName');
                var sName = document.getElementById('BaseServer');
                var cmsName = document.getElementById('CMS');
                var wsName = document.getElementById('websiteName');
                var DBrootPWD = document.getElementById('dbRootPWD');
                var DBadminUname = document.getElementById('dbAdminUname');
                var DBadminPWD = document.getElementById('dbAdminPwd');

                newContainerForm.onsubmit = function(e) {
                        //e.preventDefault();
                        var messageJSON = {
                                "CustomerType": userType,
                                "CustomerUname":userName,
                                "Action": "createNew",
                                "ContainerName": cName.value,
                                "BaseServer": sName.value,
                                "CMS": cmsName.value,
                                "WebsiteName": wsName.value,
                                "DBrootPWD": DBrootPWD.value,
                                "DBadminUname": DBadminUname.value,
                                "DBadminPWD": DBadminPWD.value
                        }
                        var json = JSON.stringify(messageJSON);

                        // Testing Purposes Only
                        //var messagesList = document.getElementById('messages');

                        //messagesList.innerHTML += '<li><span>Sent:</span>' + json + '</li>';

                        // Send the message through the WebSocket.
                        socket.send(json);

                        return false;
                };
        }
};
```

8.2.2 – Proxy-Control-Server

```
package main
    import (
        "flag"
        "log"
        "net"
        "net/http"
        "encoding/json"
        "github.com/gorilla/websocket"
    )
    type NewContainerJSON struct {
        CustomerType string             `json:"CustomerType,omitempty"`
        CustomerUname string            `json:"CustomerUname,omitempty"`
        Action string                   `json:"Action,omitempty"`
        ContainerName string            `json:"ContainerName,omitempty"`
        BaseServer string               `json:"BaseServer,omitempty"`
        CMS string                      `json:"CMS,omitempty"`
        WebsiteName string              `json:"WebsiteName,omitempty"`
        DBrootPWD string                `json:"DBrootPWD,omitempty"`
        DBadminUname string             `json:"DBadminUname,omitempty"`
        DBadminPWD string               `json:"DBadminPWD,omitempty"`
        ContainerStatus string          `json:"ContainerStatus,omitempty"`
        ContainerIPaddress string       `json:"ContainerIPaddress,omitempty"`
        WordpressStatus string          `json:"WordpressStatus,omitempty"`
    }

    var addr = flag.String("addr", "192.168.1.101:8080" , "http service address")

    var upgrader = websocket.Upgrader{
        ReadBufferSize:  1024,
        WriteBufferSize: 1024,
        CheckOrigin: func(r *http.Request) bool {
            return true
        },
    }

    func main() {
        // http.HandleFunc("/", sayhelloName) // setting router rule
        log.Print("Proxy server is up")
        flag.Parse()
        log.Println("Listening for data from Web-Server 1")
        http.HandleFunc("/user", user)
        log.Fatal(http.ListenAndServe(*addr, nil))
    }
```

```
   func user(w http.ResponseWriter, r *http.Request) {
        c, err := upgrader.Upgrade(w, r, nil)

        if err != nil {
            log.Printf("Websocket upgrade error:", err)
            return
        }
        if err == nil {
            log.Printf("Websocket upgrade completed")
        }

        go handleWebServerConnection(c)
    }

   func handleWebServerConnection(ws *websocket.Conn) {
        defer ws.Close()

        for {
            log.Println("Listening on WebSocket")

            var b NewContainerJSON

            err := ws.ReadJSON(&b)
            if err != nil {
                log.Printf("Error reading json:", err)
                break
            }

            log.Println("Recieved JSON from web server: ", b)

    /////------------- Get List of Containers -------------\\\\\
            if b.Action == "getListOfContainers" {
                log.Println("Getting list of containers")
                           if  err  :=  getListOfContainers(ws,  b.Action,  b.CustomerType,
b.CustomerUname); err == nil {
                    log.Println("Finished getting list of Container")
                }
                if err != nil {
                    log.Println("Error getting list of Container: ", err)
                }
            }
```

```
/////------------- Create New Container or Backup Container -------------\\\\\
              if b.Action == "createNew" || b.Action == "containerStart" || b.Action ==
"containerStop" || b.Action == "containerDelete" || b.Action == "containerRestart" ||
b.Action == "containerBackUp" || b.Action == "containerSnapShot" {

                dialService := "192.168.1.100:8081" //"127.0.0.1:8081"
                conn, err := net.Dial("tcp", dialService)
                if err != nil {
                    log.Println("LXC Server Dial error: ", err)
                    log.Println("Dialling WEB server as ", err)
                    // dialWebServerContainerAlreadyExists(ws, dbConStatus)
                     c := NewContainerJSON {
                         Action: "LXC Server Dial Error",
                        ContainerStatus: "Error Dialing LXC-Control-Server Server",
                    }

                    err = ws.WriteJSON(c)
                    if err != nil {
                        log.Printf("Error sending JSON to Webserver:", err)
                        break
                    } else {
                        log.Println("JSON sent to Webserver: ", c)
                    }

                } else {
                    log.Println("Dialing LXC-Control-Server")

                             dbConStatus, err := updateDatabaseContainerStatus(b.Action,
b.CustomerUname, b.ContainerName, b.ContainerStatus)
                    if err != nil {
                        log.Printf("Error Dialing Database Server: ", err)
                        b.ContainerStatus = "Error Dialing Database Server"
                    } else {
                        log.Println("Dialing Database-Control-Server")
                    }

                    // Database returns an error, break loop
                    if dbConStatus != "GoodToGo" {
                        log.Println("Dialling WEB server as ", dbConStatus)
                        // dialWebServerContainerAlreadyExists(ws, dbConStatus)
                         c := NewContainerJSON {
                             Action: "Database Error",
                            ContainerStatus: dbConStatus,
                        }

                        err = ws.WriteJSON(c)
                        if err != nil {
                            log.Printf("Error sending JSON to Webserver:", err)
                            break
                        } else {
                            log.Println("JSON sent to Webserver: ", c)
                        }
```

```
                    }
                      // Database returns NO error, send signal to LXC server to perform
action
                    if dbConStatus == "GoodToGo" {
                        encoder := json.NewEncoder(conn)
                        if err := encoder.Encode(b); err != nil {
                            log.Printf("JSON Encode error: ", err)
                        }
                        decoder := json.NewDecoder(conn)
                        var d NewContainerJSON

                        if err := decoder.Decode(&d); err != nil {
                            log.Printf("JSON Decode error: ", err)
                        }
                         log.Println("Proxy Control Server recieved JSON from LXC Control
server: ",d)

                        var dbStatus string

                        // Update database with status of customers request
                                 dbStatus, err = updateDatabaseContainerStatus(d.Action,
d.CustomerUname, d.ContainerName, d.ContainerStatus)
                        if err != nil {
                            log.Printf("Error Dialing Database Server: ", err)
                            break
                        }

                        // Send web-server status of customer request
                        d.ContainerStatus = dbStatus

                        err = ws.WriteJSON(d)
                        if err != nil {
                            log.Printf("Error JSON to Webserver:", err)
                            break
                        } else {
                            log.Printf("JSON sent to Webserver")
                        }
                    }

                }

            }
        }
    }
```

```
func getListOfContainers(ws *websocket.Conn, action string, cusType string, cusUname
string) (error) {
        var b NewContainerJSON
        b.Action = action
        b.CustomerUname = cusUname
        b.CustomerType = cusType
        dialService := "192.168.1.103:8082" //"127.0.0.1:8081"
        conn, err := net.Dial("tcp", dialService)
        if err != nil {
            log.Println("Database Dial error: ", err)

            b.Action = "Database Error"
            b.ContainerStatus = "Error Dialing Database Server"

            err = ws.WriteJSON(b)
            if err != nil {
                log.Println("Error JSON to Webserver:", err)
                return err
            } else {
                log.Println("JSON sent to Webserver")
            }

            return err
        }

        log.Println("Dialing Database-Control-Server")

        encoder := json.NewEncoder(conn)
        if err := encoder.Encode(b); err != nil {
            log.Printf("JSON Encode error: ", err)
        }


        killSig := false
        for killSig == false {
            decoder := json.NewDecoder(conn)
            var d NewContainerJSON
            if err := decoder.Decode(&d); err != nil {
                log.Println("JSON Decode error: ", err)
            }
            log.Println("Proxy Control Server recieved JSON from LXC Control server: ",d)

            if d.Action == "kill" {
                killSig = true
            }
            err = ws.WriteJSON(d)
            if err != nil {
                log.Println("Error JSON to Webserver:", err)
                return err
            } else {
                log.Println("JSON sent to Webserver")
            }
```

```
        }

        return nil
    }

func updateDatabaseContainerStatus(action string, custUname string, cName string,
conStatus string) (string, error) {
        if action == "CloneCreationFailed" || action == "snapshotCreationFailed" {
            return conStatus, nil
        }

        dialService := "192.168.1.103:8082" //"127.0.0.1:8081"
        conn, err := net.Dial("tcp", dialService)
        if err != nil {
            log.Printf("Database Server Dial error: ", err)
            return "Database Dial Error", err
        }
        defer conn.Close()

        log.Printf("Dialing Database Server")

        var b NewContainerJSON
        b.Action = action
        b.CustomerUname = custUname
        b.ContainerName = cName
        b.ContainerStatus = conStatus
        encoder := json.NewEncoder(conn)
        if err := encoder.Encode(b); err != nil {
            log.Printf("JSON Encode error: ", err)
        }
        decoder := json.NewDecoder(conn)
        var c NewContainerJSON
        if err := decoder.Decode(&c); err != nil {
            log.Printf("JSON Decode error: ", err)
            return "Database Dial Error", err
        }
        log.Println("Master server recieved JSON from Database server: ",c)

        return c.ContainerStatus, nil
    }
```

8.2.3 – Wordpress Configuration Scripts

```
#! /bin/bash

if [ $# -ne 3 ]
then
        echo "Too few arguments provided"
        exit 1
fi

uDBsitename=$1  # Wordpress installation database name
uDBuname=$2     # MySQL username for wordpress installation database
uDBpassword=$3  # MySQL password for wordpress installation database

# ********************************************************************************
# 3. Update Wordpress wp-config file.
# 4. Give ownership of /var/www/site folder Apache2.
# ********************************************************************************

sed -i -e 's/database_name_here/'${uDBsitename}'/g' /var/www/html/wp-config.php
sed -i -e 's/username_here/'${uDBuname}'/g' /var/www/html/wp-config.php
sed -i -e 's/password_here/'${uDBpassword}'/g' /var/www/html/wp-config.php

exit 0

#! /bin/bash

if [ $# -ne 4 ]
then
        echo "Too few arguments provided"
        exit 1
fi

newDBrootPW=$1  # New root password for MySQL
uDBsitename=$2  # Wordpress installation database name
uDBuname=$3     # MySQL username for wordpress installation database
uDBpassword=$4  # MySQL password for wordpress installation database

# ********************************************************************************
# 1. Change root password an MySQL database.
# 2. Create user/password/database for website CMS.
# ********************************************************************************

oldDBrootPW="root"
db="SET PASSWORD FOR root@localhost = PASSWORD('$newDBrootPW');create database $uDBsitename;GRANT ALL PRIVILEGES ON $uDBsitename.* TO $uDBuname@localhost IDENTIFIED BY '$uDBpassword';FLUSH PRIVILEGES;"
mysql -u root -p$oldDBrootPW -e "$db"

exit 0
```

8.2.4 – LXC-Control-Server

```
// +build linux,cgo

package main

import (
    "fmt"
    "net"
    "encoding/json"
    "log"
    "time"
    "flag"
    "regexp"
    "io"
    "os"
    "os/exec"
    "bufio"
    "gopkg.in/lxc/go-lxc.v2"
    "github.com/codeskyblue/go-sh"
)

type NewContainerJSON struct {
    CustomerType string             `json:"CustomerType,omitempty"`
    CustomerUname string            `json:"CustomerUname,omitempty"`
    Action string                   `json:"Action,omitempty"`
    ContainerName string            `json:"ContainerName,omitempty"`
    BaseServer string               `json:"BaseServer,omitempty"`
    CMS string                      `json:"CMS,omitempty"`
    WebsiteName string              `json:"WebsiteName,omitempty"`
    DBrootPWD string                `json:"DBrootPWD,omitempty"`
    DBadminUname string             `json:"DBadminUname,omitempty"`
    DBadminPWD string               `json:"DBadminPWD,omitempty"`
    ContainerStatus string          `json:"ContainerStatus,omitempty"`
    ContainerIPaddress string       `json:"ContainerIPaddress,omitempty"`
    WordpressStatus string          `json:"WordpressStatus,omitempty"`
}

var lxcpath string

func init() {
   flag.StringVar(&lxcpath, "lxcpath", lxc.DefaultConfigPath(), "Use specified container
path")
    flag.Parse()
}
```

```
func main() {
        log.Println("LXC-Control-Sever is up")
        service := "192.168.1.100:8081"

        tcpAddr, err := net.ResolveTCPAddr("tcp", service)
        if err != nil {
                log.Println("Failed to Resolve TCP Address: %s", err)
        } else {
                log.Println("Resolved TCP Address: ", service)

                ln, err := net.ListenTCP("tcp", tcpAddr)
                if err != nil {
                        log.Println("Failed to listen: %s", err)
                } else {
                        log.Println("Listening on: ", tcpAddr)

                        for{
                                if conn, err := ln.Accept(); err == nil {
                                        go handleConnection(conn)
                                }
                        }
                }
        }
}
```

```
func handleConnection(conn net.Conn){
        defer conn.Close()

        log.Println("Connection Open")
        log.Println("Waiting for Proxy-Control-Server...")

        decoder := json.NewDecoder(conn)

        var b NewContainerJSON
        if err := decoder.Decode(&b); err != nil {
                log.Println("JSON Decode error: ", err)
        }

    var (
         newContainerName, containerAction, cStatus, cIPaddress, wordpressStatus string
        )

    log.Println("LXC-Control-Server recieved JSON from Proxy-Control-Server: ", b)

        if b.Action == "createNew" {
                newContainerName = fmt.Sprint(b.CustomerUname + "-" + b.ContainerName)
                  originalContainer := "standardClone"  // Temporary measure until logic
develpoed to determine which original Template Clone is to be used
                                                cStatus,   cIPaddress,   wordpressStatus   =
createNewContainer(originalContainer,   newContainerName,   b.DBrootPWD,   b.WebsiteName,
b.DBadminUname, b.DBadminPWD)
                containerAction = "UpdateContainerStatus"
        }

        if b.Action == "updateDBcontainerStatus" {
                newContainerName = b.ContainerName
                cStatus = updateDBcontainerStatus(b.ContainerName)
                containerAction = "UpdateContainerStatus"
        }

        if b.Action == "containerBackUp" {
                t := time.Now()
                var err error
        newContainerName = fmt.Sprint(b.ContainerName + "-Clone-" + t.Format("2006-01-02-
15:04:05"))
                cStatus, err = cloneNewContainer(b.ContainerName, newContainerName)
                if err != nil {
                        containerAction = "CloneCreationFailed"
                } else {
                        containerAction = "createDBentryForClone"
                }
        }

        if b.Action == "containerSnapShot" {
                t := time.Now()
                var err error
        newContainerName = fmt.Sprint(b.ContainerName + "-Snapshot-" + t.Format("2006-01-
```

```
02-15:04:05"))
                cStatus, err = snapshotNewContainer(b.ContainerName, newContainerName)
                if err != nil {
                        containerAction = "snapshotCreationFailed"
                } else {
                        containerAction = "createDBentryForSnaphot"
                }
        }

        if b.Action == "containerStart" {
                cStatus, cIPaddress = startContainer(b.ContainerName)
                containerAction = "UpdateContainerStatus"
                newContainerName = b.ContainerName
        }

        if b.Action == "containerStop" {
                cStatus = stopContainer(b.ContainerName)
                containerAction = "UpdateContainerStatus"
                newContainerName = b.ContainerName
        }

        if b.Action == "containerDelete" {
                cStatus = deleteContainer(b.ContainerName)
                containerAction = "deleteContainerFromDatabase"
                newContainerName = b.ContainerName
        }

        if b.Action == "containerRestart" {
                cStatus = stopContainer(b.ContainerName)

                if cStatus == "Stopped" {
                        cStatus, cIPaddress = startContainer(b.ContainerName)
                        containerAction = "UpdateContainerStatus"
                        newContainerName = b.ContainerName
                } else {
                        containerAction = "UpdateContainerStatus"
                        newContainerName = b.ContainerName
                }

        }

   d := NewContainerJSON {
                CustomerUname: b.CustomerUname,
                Action: containerAction,
        ContainerName: newContainerName,
        BaseServer: b.BaseServer,
        CMS: b.CMS,
        WebsiteName: b.WebsiteName,
        ContainerStatus: cStatus,
        ContainerIPaddress: cIPaddress,
        WordpressStatus: wordpressStatus,
    }
```

```
    log.Println("Sending Proxy-Control-Server the following JSON: ", d)

    // Send JSON back to source
    encoder := json.NewEncoder(conn)
        if err := encoder.Encode(d); err != nil {
                fmt.Println("JSON Encode error:: ", err)
        }
}

func updateDBcontainerStatus(containerName string) (string) {
        x, err := regexp.Compile(containerName)
        if err != nil {
                log.Println("Error compiling '", containerName, "' regex: ", err)
                cStatus := "Error"
                return cStatus
        }

        y, err := regexp.Compile("RUNNING")
        if err != nil {
                log.Println("Error compiling 'RUNNING' regex: ", err)
                cStatus := "Error"
                return cStatus
        }

        z, err := regexp.Compile("STOPPED")
        if err != nil {
                log.Println("Error compiling 'STOPPED' regex: ", err)
                cStatus := "Error"
                return cStatus
        }

        cmd := exec.Command("lxc-ls", "-f")

        // capture the output and error pipes
        stdout, err := cmd.StdoutPipe()
        if err != nil {
                log.Println("Error opening stdout pipe: ", err)
                cStatus := "Error"
                return cStatus
                os.Exit(1)
        }

        stderr, err := cmd.StderrPipe()
        if err != nil {
                log.Println("Error opening stderr pipe: ", err)
                cStatus := "Error"
                return cStatus
                os.Exit(1)
        }

        err = cmd.Start()
```

```
        if err != nil {
                log.Println("Error running shell commands: ", err)
                cStatus := "Error"
                return cStatus
                os.Exit(1)
        }

        defer cmd.Wait()

        go io.Copy(os.Stderr, stderr)

        buff := bufio.NewScanner(stdout)
        var allText []string

        for buff.Scan() {
                 allText = append(allText, buff.Text()+"\n")
        }

        for i := range allText {
                log.Println("Doing a Regex Match on: ", containerName)
                log.Println("Matching Regex against string: ", allText[i])

                if x.MatchString(allText[i]) == true {
                        log.Println("Regex Match On: ", containerName)

                        if y.MatchString(allText[i]) == true {
                                cStatus := "Running"
                                        log.Println("Sending database ", cStatus, " for
container", containerName)
                                return cStatus
                        }

                        if z.MatchString(allText[i]) == true {
                                cStatus := "Stopped"
                                        log.Println("Sending database ", cStatus, " for
container: ", containerName)
                                return cStatus
                        }
                }
        }

        log.Println("No Regex Match On: ", containerName)
        cStatus := "Suspended"
        log.Println("Sending database ", cStatus, " for container: ", containerName)
        return cStatus
}
```

```
func createNewContainer(oldContainerName string, newContainerName string, cDBrootPWD string, cWebsiteName string, cDBadminUname string, cDBadminPWD string) (string, string, string) {

	log.Println(newContainerName)

	var (
		cStatus, cIPaddress, wordpressStatus string
		err error
	)

	 if cStatus, err = cloneNewContainer(oldContainerName, newContainerName); err == nil {
		cStatus, cIPaddress = startContainer(newContainerName)
			wordpressStatus = configureWordpressContainer(newContainerName, cDBrootPWD, cWebsiteName, cDBadminUname, cDBadminPWD)
	} else {
			log.Printf("Not getting IP address or configuring container due to container creation failure")
		cIPaddress = ""
		wordpressStatus = "Wordpress not started"
		cStatus = "Error Creating Container"
	}

	log.Println(newContainerName, " Status: ", cStatus)
	log.Println(newContainerName, " IP Address: ", cIPaddress)
	log.Println(newContainerName, " Wordpress Status: ", cStatus)

	return cStatus, cIPaddress, wordpressStatus

}
```

```
func snapshotNewContainer(originalContainerName string, newContainerName string) (string,
error) {
        var cStatus string

        c, err := lxc.NewContainer(originalContainerName, lxcpath)
        if err != nil {
                cStatus = "Error Creating Snapshot"
                return cStatus, err
        }

        log.Printf("Snapshoting the container...\n")
        if _, err := c.CreateSnapshot(); err != nil {
                cStatus = "Error Creating Snapshot"
                return cStatus, err
        }

        cStatus = "Snapshot Created Successfully"
        log.Println(newContainerName, " created successfully")
        return cStatus, nil

}
```

```
func cloneNewContainer(originalContainer string, containerName string) (string, error) {
        var (
                backend lxc.BackendStore
                cStatus string
        )

        if originalContainer == "standardClone" {
                originalContainer = "clone-u1404-A2-MySQL-WP"
        }

        lxcpath = lxc.DefaultConfigPath()
        backend = 2

        log.Println("Creating new container: ", containerName)

        c, err := lxc.NewContainer(originalContainer, lxcpath)
        if err != nil {
                log.Println("Error setting up details for ", containerName, ": ", err)
                cStatus = "Error Creating Container"
                return cStatus, err
        }

        if backend == 0 {
                 log.Println("Error setting up Backend Store for ", containerName, ": ",
lxc.ErrUnknownBackendStore)
                cStatus = "Error Creating Container"
                return cStatus, err
        }

        log.Println("Creating ", containerName, " using ", backend, " backend...")
        err = c.Clone(containerName, lxc.CloneOptions{
                Backend: backend,
        })
        if err != nil {
                log.Println("Error creating ", containerName, ": ", err)
                cStatus = "Error Creating Container"
                return cStatus, err
        }

        cStatus = "Clone Created Successfully"
        log.Println(containerName, " created successfully")
        return cStatus, nil
}
```

```
func startContainer(containerName string) (string, string) {
        var (
                cStatus, cIPaddress string
        )

        lxcpath = lxc.DefaultConfigPath()

        c, err := lxc.NewContainer(containerName, lxcpath)
        if err != nil {
                log.Println("Error Starting Container: ", err)
                cStatus = "Error Starting Container"
                cIPaddress = "Error Starting Container"
                return cStatus, cIPaddress
        }

        c.SetLogFile("/tmp/" + containerName + ".log")
        c.SetLogLevel(lxc.TRACE)

        log.Println("Starting: ", containerName)
        if err := c.Start(); err != nil {
                log.Println("Error Starting", containerName, " : ", err)
                cStatus = "Error Starting Container"
                cIPaddress = "Error Starting Container"
                return cStatus, cIPaddress
        }

        log.Println("Waiting for ", containerName, " to startup networking...")
        if _, err := c.WaitIPAddresses(5 * time.Second); err == nil {
                cStatus = "Running"
        }
        if err != nil {
                log.Println("Error starting ", containerName, ": ", err)
                cStatus = "Error Starting Container"
                cIPaddress = "Error Starting Container"
                return cStatus, cIPaddress
        }

        log.Println(containerName, " started successfully")
        cStatus, cIPaddress = getContainerIPaddress(containerName)

        return cStatus, cIPaddress
}
```

```
func getContainerIPaddress(containerName string) (string, string){
        var (
                lxcpath string
                cIPaddress, cStatus string
        )

        log.Println("Getting IP address for container: ", containerName)
        lxcpath = lxc.DefaultConfigPath()

        c, err := lxc.NewContainer(containerName, lxcpath)
        if err != nil {
                log.Println("Error Getting IP address: ", err)
                cStatus = "Error Starting Container"
                cIPaddress = "Error Getting IP address"
                return cStatus, cIPaddress
        }

        loopControl := 0
        for loopControl != -1 && loopControl <= 5 {
                if addresses, err := c.IPv4Addresses(); err != nil {
                        if loopControl == 4 {
                                        log.Println("Error Getting IP address for ",
containerName, ": ", err)
                                cStatus = "Error Starting Container"
                                cIPaddress = "Error Getting IP address"
                                return cStatus, cIPaddress
                        }
                        log.Println("Error Getting IP address for ", containerName, ": ",
err)
                        log.Println("Waiting 5 seconds before trying again.")
                        loopControl += 1
                        time.Sleep(5 * time.Second)
                } else {
                        for i := range addresses {
                                cIPaddress = addresses[i]
                                cStatus = "Running"
                                log.Println("Obtained IP address for ", containerName, ":
", cIPaddress)
                                loopControl = -1
                        }
                }
        }

        return cStatus, cIPaddress
}
```

```
func stopContainer(containerName string) (string) {
        var (
                cStatus string
        )

        c, err := lxc.NewContainer(containerName, lxcpath)
        if err != nil {
                log.Println("Error stopping ", containerName,": ", err)
                cStatus = "Error stopping container"
                return cStatus
        }

        c.SetLogFile("/tmp/" + containerName + ".log")
        c.SetLogLevel(lxc.TRACE)

        log.Println("Stopping: ", containerName)
        if err := c.Stop(); err != nil {
                log.Println("Error stopping ", containerName,": ", err)
                cStatus = "Error stopping container"
                return cStatus
        } else {
                cStatus = "Stopped"
                return cStatus
        }
}

func deleteContainer(containerName string) (string) {
        var (
                cStatus string
        )

        c, err := lxc.NewContainer(containerName, lxcpath)
        if err != nil {
                log.Println("Error deleting ", containerName, ": ", err)
                cStatus = "Error Deleting container"
                return cStatus
        }

        c.SetLogFile("/tmp/" + containerName + ".log")
        c.SetLogLevel(lxc.TRACE)

        log.Println("Deleting: ", containerName)
        if err := c.Destroy(); err != nil {
                log.Println("Error deleting ", containerName, ": ", err)
                cStatus = "Error Deleting containe"
                return cStatus
        } else {
                log.Println("Successfully deleted: ", containerName)
                cStatus = "Deleted"
                return cStatus
        }
}
```

```
func configureWordpressContainer(containerName string, cDBrootPWD string, cWebsiteName string, cDBadminUname string, cDBadminPWD string) (string) {
        time.Sleep(5 * time.Second) // Sleeping to give time for container to fully boot
        var wpStatus string

        log.Println("Waiting 5 seconds for ", containerName, " to fully boot up.")
                log.Println("Configuring Wordpress MySQL database for container: ", containerName)

        loopControl := 0
        for loopControl != -1 && loopControl <= 5 {
                        if err := sh.Command("lxc-attach", "-n", containerName, "--", "/home/dave/lxc-config-1.sh", cWebsiteName, cDBadminUname, cDBadminPWD).Run(); err != nil {
                        if loopControl == 4 {
                                        log.Println("Wordpress MySQL Database configuration failed: ", err)
                                wpStatus = "Error Configuring Worpress MySQL Database.\n"
                        }
                          log.Println("Waiting 5 more seconds for ", containerName, " to fully boot up.")
                        loopControl += 1
                        time.Sleep(5 * time.Second)
                } else {
                        log.Println("Wordpress MySQL Database successfully configured for container: " + containerName)
                        loopControl = -1
                }
        }

              log.Println("Configuring Wordpress wp-config.php file for container: ", containerName)

          if err := sh.Command("lxc-attach", "-n", containerName, "--", "/home/dave/lxc-config-2.sh", cDBrootPWD, cWebsiteName, cDBadminUname, cDBadminPWD).Run(); err != nil {
                log.Println("Error Configuring Worpress wp-config.php file: ", err)
                wpStatus2 := "Error Configuring Worpress wp-config.php file."
                wpStatus = fmt.Sprint(wpStatus + " " + wpStatus2)

        }

         log.Println("Wordpress wp-config.php file successfully configured for container: ", containerName)
        wpStatus = "Wordpress Configured"
        return wpStatus
}
```

8.2.5 – Database-Control-Server

```
package main

import(
        "net"
        "encoding/json"
        "log"
        "fmt"
        "strings"
        "time"
        "database/sql"
        _ "github.com/go-sql-driver/mysql"
)

type NewContainerJSON struct {
    CustomerType string             `json:"CustomerType,omitempty"`
    CustomerUname string            `json:"CustomerUname,omitempty"`
    Action string                   `json:"Action,omitempty"`
    ContainerName string            `json:"ContainerName,omitempty"`
    BaseServer string               `json:"BaseServer,omitempty"`
    CMS string                      `json:"CMS,omitempty"`
    WebsiteName string              `json:"WebsiteName,omitempty"`
    DBrootPWD string                `json:"DBrootPWD,omitempty"`
    DBadminUname string             `json:"DBadminUname,omitempty"`
    DBadminPWD string               `json:"DBadminPWD,omitempty"`
    ContainerStatus string          `json:"ContainerStatus,omitempty"`
    ContainerIPaddress string       `json:"ContainerIPaddress,omitempty"`
    WordpressStatus string          `json:"WordpressStatus,omitempty"`
}
```

```
func main() {
        func main() {
        log.Println("Database-Control-Server is up")

        service := "192.168.1.103:8082" //"127.0.0.1:8081"

        db, err := sql.Open("mysql","dave:admin@tcp(127.0.0.1:3306)/praxis")
        if err != nil {
                log.Println("Database Connection failed: ", err)
        }

        defer db.Close()

        err = db.Ping()
        if err != nil {
                log.Println("MySQL Database Ping failed: ", err)
                log.Println("Database Server is down")
        } else {
                log.Println("DB Ping successful")

                err = updateDBcontainerStatusOnStartUp(db)
                if err != nil {
                            log.Println("Error getting update from LXC-Control-Server: ",
err)
                        log.Println("Database Server is down")
                } else {
                              log.Println("Finished getting update of current registered
containers status from LXC-Control Server")

                        tcpAddr, err := net.ResolveTCPAddr("tcp", service)
                        if err != nil {
                                log.Println("Failed to Resolve TCP Address: %s", err)
                                log.Println("Database Server is down")
                        } else {
                                log.Println("Resolved TCP Address: ", service)
                                ln, err := net.ListenTCP("tcp", tcpAddr)

                                if err != nil {
                                        log.Println("Failed to listen: %s", err)
                                        log.Println("Database Server is down")
                                } else {
                                        log.Println("Listening on: ", tcpAddr)

                                        for{
                                                if conn, err := ln.Accept(); err == nil {
                                                        go handleConnection(conn, db)
                                                }
                                        }
                                }
                        }
                }
    } }  }
```

```
func handleConnection(conn net.Conn, db *sql.DB){

        defer conn.Close()

        decoder := json.NewDecoder(conn)

        var b NewContainerJSON
        if err := decoder.Decode(&b); err != nil {
                log.Println("JSON Decode error: ", err)
        }

          log.Println("Database Control server recieved JSON from Master Control server:
",b)

        var cStatus, cName string

        if b.Action == "createNew" {
                cName = fmt.Sprint(b.CustomerUname + "-" + b.ContainerName)
        } else {
                cName = b.ContainerName
        }

        if b.Action == "getListOfContainers" {

                        if err := getListOfContainers(conn, db, b.Action, b.CustomerUname,
b.CustomerType); err==nil {
                                log.Println ("All containers sent to Praxis-Proxy-Server")
                } else {
                                log.Println ("Error sending containers to Praxis-Proxy-Server: ",
err)
                }
```

```
//      Action      "createNew"      "UpdateContainerStatus"      "createDBentryForClone"
"deleteContainerFromDatabase" "getListOfContainers"
//   Action   "containerStart"   "containerStop"   "containerDelete"   "containerRestart"
"containerBackUp" "containerSnapShot" createDBentryForSnaphot
        } else {
                        cStatus = handleDatabaseQuery(db, b.Action, b.CustomerUname, cName,
b.ContainerStatus)

                c := NewContainerJSON {
                        Action: b.Action,
                        CustomerUname: b.CustomerUname,
                        ContainerName: cName,
                        ContainerStatus: cStatus,
                }

                log.Println("This is the new JSON we will send back: ", c)

                // Send JSON back to source
                encoder := json.NewEncoder(conn)
                if err := encoder.Encode(c); err != nil {
                        log.Println("JSON Encode error:: ", err)
                }
        }

}
```

```
func handleDatabaseQuery(db *sql.DB, action string, cusUname string, cName string,
conStatus string) (string) {
        var cStatus string

        if action == "createNew" {

                cStatus = runCheckIfContainerExist(db, cusUname, cName, action)
                if cStatus == "GoodToGo" {
                         cStatus = addDataBaseEntryForNewContainer(db, action, cusUname,
cName)
                }
                log.Println("Here 3", cStatus)
        }

             if action == "containerStart" || action == "containerStop" || action ==
"containerRestart" || action == "containerBackUp" || action == "containerSnapShot" {
                cStatus = runCheckIfContainerExist(db, cusUname, cName, action)
        }

        if action == "containerDelete" {
                cStatus = runCheckIfContainerExist(db, cusUname, cName, action)

                if cStatus == "GoodToGo" {
                         cStatus = checkIfExistingContainerHasAClone(db, cusUname, cName,
action)

                        if cStatus == "GoodToGo" {
                                        cStatus = checkIfExistingContainerHasASnaphot(db,
cusUname, cName, action)
                        }
                }

                return cStatus
        }

    if action == "createDBentryForClone" || action == "createDBentryForSnaphot" {
        cStatus = addDataBaseEntryForNewContainer(db, action, cusUname, cName)
        log.Println("Here 5")
    }

        if action == "UpdateContainerStatus" {
                cStatus = updateContainerStatus(db, cName, conStatus)
        }

        if action == "deleteContainerFromDatabase" {
                cStatus = deleteContainerFromContainer_Table(db, cusUname, cName)
        }

        log.Println("Here 4", cStatus)

        return cStatus
}
```

```
func checkIfExistingContainerHasASnaphot(db *sql.DB, cusUname string, cName string, action string) (string) {
        var qryResult string

          rows, err := db.Query("SELECT Container_Snapshot_Table.Container_Snapshot_Name FROM Container_Snapshot_Table INNER JOIN Container_Table ON Container_Snapshot_Table.Container_ID=Container_Table.Container_ID WHERE Container_Table.Container_Name=(?);", cName)

        if err != nil {
                log.Println("Error creating MySQL Query: ", err)
                cStatus := "Error Reading Database"
                        return cStatus
        }

        defer rows.Close()

        for rows.Next() {
                err := rows.Scan(&qryResult)
                if err != nil {
                        log.Println("Error 1 reading MySQL output: ", err)
                        cStatus := "Error Reading Database"
                        return cStatus
                }
                log.Println("This is the query result", qryResult)
        }

        err = rows.Err()
        if err != nil {
                log.Println("Error 2 reading MySQL output: ", err)
                cStatus := "Error Reading Database"
                return cStatus
        }

        // Container Clone and Snapshot will be evaulated here
        if qryResult == "" {
                log.Println("Here 1100: Exit GoodToGo")
                cStatus := "GoodToGo"
                return cStatus
        } else {
                log.Println("Here 1200: Backup of original container already exists")
                cStatus := "Cannot delete container as a backup already exists"
                return cStatus
        }
}
```

```
func checkIfExistingContainerHasAClone(db *sql.DB, cusUname string, cName string, action
string) (string) {
        var qryResult string

         rows, err := db.Query("SELECT Container_Backup_Table.Container_Backup_Name FROM
Container_Backup_Table          INNER          JOIN          Container_Table          ON
Container_Backup_Table.Container_ID=Container_Table.Container_ID                    WHERE
Container_Table.Container_Name=(?);", cName)

        if err != nil {
                log.Println("Error creating MySQL Query: ", err)
                cStatus := "Error Reading Database"
                        return cStatus
        }

        defer rows.Close()

        for rows.Next() {
                err := rows.Scan(&qryResult)
                if err != nil {
                        log.Println("Error 1 reading MySQL output: ", err)
                        cStatus := "Error Reading Database"
                        return cStatus
                }
                log.Println("This is the query result", qryResult)
        }

        err = rows.Err()
        if err != nil {
                log.Println("Error 2 reading MySQL output: ", err)
                cStatus := "Error Reading Database"
                return cStatus
        }

        // Container Clone and Snapshot will be evaulated here
        if qryResult == "" {
                log.Println("Here 900: Exit GoodToGo")
                cStatus := "GoodToGo"
                return cStatus
        } else {
                log.Println("Here 1000: Backup of original container already exists")
                cStatus := "Cannot delete container as a backup already exists"
                return cStatus
        }
}
```

```
func getListOfContainers(conn net.Conn, db *sql.DB, cAction string, cusUname string, cusType string) (error){
	defer conn.Close()

	var (
		containerStatus string
		containerName string
	)

		rows, err := db.Query("SELECT Container_Table.Container_Name, Container_Table.Container_Status FROM Container_Table INNER JOIN Customer_Table ON Container_Table.Customer_ID=Customer_Table.Customer_ID WHERE Customer_Table.Customer_UserName=(?)",cusUname)

	c := NewContainerJSON {
		Action: cAction,
		CustomerUname: cusUname,
	}

	if cusType == "admin" {
			rows, err = db.Query("select Container_Table.Container_Name, Container_Table.Container_Status from Container_Table;")
	}

	if err != nil {
		log.Println("Error creating MySQL Query: ", err)
		return err
	}

	defer rows.Close()

	for rows.Next() {
		err := rows.Scan(&containerName, &containerStatus)
		if err != nil {
			log.Println("Error 1 reading MySQL output: ", err)
			return err
		}
			log.Println("This is the query result: ", containerName, "-",containerStatus)

		c.ContainerName = containerName
		c.ContainerStatus = containerStatus
		log.Println("This is the new JSON we will send back: ", c)

		// Send JSON back to source
		encoder := json.NewEncoder(conn)
		if err := encoder.Encode(c); err != nil {
			log.Println("JSON Encode error: ", err)
		}

		log.Println("Waiting 1 second")
		time.Sleep(1 * time.Second)
```

```
        }

        err = rows.Err()
        if err != nil {
                log.Println("Error 2 reading MySQL output: ", err)
                return err
        }

        c.Action = "kill"
        log.Println("This is the new JSON we will send back: ", c)

        encoder := json.NewEncoder(conn)
        if err := encoder.Encode(c); err != nil {
                log.Println("JSON Encode error:: ", err)
        }

        return nil
}
```

```
func runCheckIfContainerExist(db *sql.DB, cusUname string, cName string, action string)
(string) {
        var qryResult string

        rows, err := db.Query("SELECT Container_Table.Container_Name FROM Container_Table
INNER JOIN Customer_Table ON Container_Table.Customer_ID=Customer_Table.Customer_ID WHERE
Container_Table.Container_Name=(?)  AND    Customer_Table.Customer_UserName=(?)",  cName,
cusUname)

        if err != nil {
                log.Println("Error creating MySQL Query: ", err)
                cStatus := "Error Reading Database"
                        return cStatus
        }

        defer rows.Close()

        for rows.Next() {
                err := rows.Scan(&qryResult)
                if err != nil {
                        log.Println("Error 1 reading MySQL output: ", err)
                        cStatus := "Error Reading Database"
                        return cStatus
                }
                log.Println("This is the query result", qryResult)
        }

        err = rows.Err()
        if err != nil {
                log.Println("Error 2 reading MySQL output: ", err)
                cStatus := "Error Reading Database"
                return cStatus
        }

         // If it's a change to an existing containers status, check that it is at that
status already
            if action == "containerStart" || action == "containerStop" || action ==
"containerDelete" || action == "containerRestart" || action == "containerBackUp" /*||
action == "containerSnapShot"*/ {
                if qryResult == "" {
                        log.Println("Here 500: Container Does Not Exist")
                        cStatus := "Container Does Not Exist"
                        return cStatus
                } else {
                        log.Println("Here 600: Exit GoodToGo")
                              cStatus := checkExistingContainersCurrentStatus(db, action,
cName)
                        return cStatus
                }
        }

         // We need to invert result for createNew Container as we only want to create
```

```
container if it does not already exist
        if action == "createNew" {
                if qryResult == "" {
                        log.Println("Here 100: Exit GoodToGo")
                        cStatus := "GoodToGo"
                        return cStatus
                } else {
                        log.Println("Here 200: Container Already Exists")
                        cStatus := "Container Already Exists"
                        return cStatus
                }
        }

        // Container Clone and Snapshot will be evaulated here
        if qryResult == "" {
                log.Println("Here 300: Container Does Not Exist")
                cStatus := "Container Does Not Exist"
                return cStatus
        } else {
                log.Println("Here 400: Exit GoodToGo")
                cStatus := "GoodToGo"
                return cStatus
        }
}
```

```
func checkExistingContainersCurrentStatus(db *sql.DB, action string, cName string)
(string) {
        var qryResult, containerStatus string

        switch action {
                case "containerStart": containerStatus = "Running"
                case "containerStop": containerStatus = "Stopped"
                case "containerRestart": containerStatus = "Stopped"
                case "containerDelete": containerStatus = "Running"
                case "containerBackUp": containerStatus = "Running"
                case "containerSnapShot": containerStatus = "Running"
        }
        rows, err := db.Query("SELECT Container_Table.Container_Name FROM Container_Table
WHERE Container_Name=(?) and Container_Status=(?)", cName, containerStatus)

        if err != nil {
                log.Println("Error creating MySQL Query: ", err)
                cStatus := "Error Updating Database"
                        return cStatus
        }

        defer rows.Close()

        for rows.Next() {
                err := rows.Scan(&qryResult)
                if err != nil {
                        log.Println("Error 1 reading MySQL output: ", err)
                        cStatus := "Error Updating Database"
                        return cStatus
                }
                log.Println("This is the query result", qryResult)
        }

        log.Println("This is the query result", qryResult)

        err = rows.Err()
        if err != nil {
                log.Println("Error 2 reading MySQL output: ", err)
                cStatus := "Error Updating Database"
                return cStatus
        }

        if qryResult == "" {
                log.Println("Here 800: Exit GoodToGo")
                cStatus := "GoodToGo"
                return cStatus
        } else {
                log.Println("Here 700: Container Does Not Exist")
                cStatus := "Containers current staus is not compatable with your request"
                return cStatus
        }
}
```

```
func addDataBaseEntryForNewContainer (db *sql.DB, action string, cusUname string, cName
string) (string) {
        var cStatus string
        var customerID string

              rows, err := db.Query ("select Customer_ID FROM Customer_Table WHERE
Customer_UserName=(?);",cusUname)

        if err != nil {
                log.Println("Error creating MySQL Query: ", err)
                cStatus := "Error Updating Database"
                        return cStatus
        }

        defer rows.Close()

        for rows.Next() {
                err := rows.Scan(&customerID)
                if err != nil {
                        log.Println("Error 1 reading MySQL output: ", err)
                        cStatus := "Error Updating Database"
                        return cStatus
                }
                log.Println(customerID)
        }

        err = rows.Err()
        if err != nil {
                log.Println("Error 2 reading MySQL output: ", err)
                cStatus := "Error Updating Database"
                return cStatus
        }
        if customerID == "" {
                log.Println("User does not exist")
                cStatus := "User does not exist"
                return cStatus
        }

        if action == "createNew" {
                        cStatus = dbQryInsertIntoContainer_Table(db, 1, customerID, cName,
cusUname)
        }
        if action == "createDBentryForClone" {
                cStatus = dbQryInsertIntoContainer_Backup_Table(db, customerID, cName)
        }

        if action == "createDBentryForSnaphot" {
                cStatus = dbQryInsertIntoContainer_Snapshot_Table(db, customerID, cName)
        }
        return cStatus
}
```

```
func deleteContainerFromContainer_Table(db *sql.DB, cusUname string, cName string)
(string) {
	var cStatus string

		stmt, err := db.Prepare("DELETE FROM Container_Table WHERE
Container_Table.Container_Name=(?)")

	if err != nil {
		log.Println("Error preparing database statement: ", err)
		cStatus := "Error Updating Database"
		return cStatus
	}

	res, err := stmt.Exec(cName)
	if err != nil {
		log.Println("Error 1 Adding New Container To Database: ", err)
		cStatus := "Error Updating Database"
		return cStatus
	} else {
		cStatus = "Deleted"
	}

	lastId, err := res.LastInsertId()
	if err != nil {
		log.Println("Error 2 Adding New Container To Database: ", err)
		cStatus := "Error Updating Database"
		return cStatus
	}

	rowCnt, err := res.RowsAffected()
	if err != nil {
		log.Println("Error 3 Adding New Container To Database: ", err)
		cStatus := "Error Updating Database"
		return cStatus
	}
	log.Printf("ID = %d, affected = %d\n", lastId, rowCnt)

	return cStatus
}
```

```
func dbQryInsertIntoContainer_Snapshot_Table (db *sql.DB, custNumber string, clonedContainerName string) (string) {
	strArray := strings.Split(clonedContainerName,"-")

	fmt.Println(strArray)

	originalContainerName := fmt.Sprint(strArray[0] + "-" + strArray[1])

	fmt.Println(originalContainerName)

	var cStatus string
	var container_Table_ID string

	     rows, err := db.Query ("select Container_ID FROM Container_Table WHERE Container_Name=(?);", originalContainerName)

	if err != nil {
		log.Println("Error creating MySQL Query: ", err)
		cStatus := "Error Updating Database"
			return cStatus
	}

	defer rows.Close()

	for rows.Next() {
		err := rows.Scan(&container_Table_ID)
		if err != nil {
			log.Println("Error 1 reading MySQL output: ", err)
			cStatus := "Error Updating Database"
			return cStatus
		}
		log.Println(container_Table_ID)
	}

	err = rows.Err()
	if err != nil {
		log.Println("Error 2 reading MySQL output: ", err)
		cStatus := "Error Updating Database"
		return cStatus
	}

	if container_Table_ID == "" {
		log.Println("Container does not exist")
		cStatus := "Container does not exist"
		return cStatus
	}

	  stmt, err := db.Prepare("INSERT INTO Container_Snapshot_Table ( Container_ID, Container_Snapshot_Name ) VALUES (?,?)")

	if err != nil {
		log.Println("Error preparing database statement: ", err)
```

```
                cStatus := "Error Updating Database"
                return cStatus
        }

        res, err := stmt.Exec(container_Table_ID ,clonedContainerName)
        if err != nil {
                log.Println("Error 1 Adding New Container To Database: ", err)
                cStatus := "Error Updating Database"
                return cStatus
        } else {
                cStatus = "GoodToGo"
        }

        lastId, err := res.LastInsertId()
        if err != nil {
                log.Println("Error 2 Adding New Container To Database: ", err)
                cStatus := "Error Updating Database"
                return cStatus
        }

        rowCnt, err := res.RowsAffected()
        if err != nil {
                log.Println("Error 3 Adding New Container To Database: ", err)
                cStatus := "Error Updating Database"
                return cStatus
        }
        log.Printf("ID = %d, affected = %d\n", lastId, rowCnt)

        return cStatus

}
```

```
func dbQryInsertIntoContainer_Backup_Table (db *sql.DB, custNumber string,
clonedContainerName string) (string) {
        strArray := strings.Split(clonedContainerName,"-")

        fmt.Println(strArray)

        originalContainerName := fmt.Sprint(strArray[0] + "-" + strArray[1])

        fmt.Println(originalContainerName)

        var cStatus string
        var container_Table_ID string

             rows, err := db.Query ("select Container_ID FROM Container_Table WHERE
Container_Name=(?);", originalContainerName)

        if err != nil {
                log.Println("Error creating MySQL Query: ", err)
                cStatus := "Error Updating Database"
                        return cStatus
        }

        defer rows.Close()

        for rows.Next() {
                err := rows.Scan(&container_Table_ID)
                if err != nil {
                        log.Println("Error 1 reading MySQL output: ", err)
                        cStatus := "Error Updating Database"
                        return cStatus
                }
                log.Println(container_Table_ID)
        }

        err = rows.Err()
        if err != nil {
                log.Println("Error 2 reading MySQL output: ", err)
                cStatus := "Error Updating Database"
                return cStatus
        }

        if container_Table_ID == "" {
                log.Println("Container does not exist")
                cStatus := "Container does not exist"
                return cStatus
        }

           stmt, err := db.Prepare("INSERT INTO Container_Backup_Table ( Container_ID,
Container_Backup_Name ) VALUES (?,?)")

        if err != nil {
                log.Println("Error preparing database statement: ", err)
```

```
                cStatus := "Error Updating Database"
                return cStatus
        }

        res, err := stmt.Exec(container_Table_ID ,clonedContainerName)
        if err != nil {
                log.Println("Error 1 Adding New Container To Database: ", err)
                cStatus := "Error Updating Database"
                return cStatus
        } else {
                cStatus = "GoodToGo"
        }

        lastId, err := res.LastInsertId()
        if err != nil {
                log.Println("Error 2 Adding New Container To Database: ", err)
                cStatus := "Error Updating Database"
                return cStatus
        }

        rowCnt, err := res.RowsAffected()
        if err != nil {
                log.Println("Error 3 Adding New Container To Database: ", err)
                cStatus := "Error Updating Database"
                return cStatus
        }
        log.Printf("ID = %d, affected = %d\n", lastId, rowCnt)

        return cStatus

}
```

```
func dbQryInsertIntoContainer_Table(db *sql.DB, baseContainerNum int, custNumber string,
cName string, cusUname string) (string) {
        var cStatus string

         stmt, err := db.Prepare("INSERT INTO Container_Table ( Base_Container_Clone_ID,
Customer_ID, Container_Name ) VALUES (?,?,?)")

        if err != nil {
                log.Println("Error preparing database statement: ", err)
                cStatus := "Error Updating Database"
                return cStatus
        }

        res, err := stmt.Exec(baseContainerNum ,custNumber ,cName)
        if err != nil {
                log.Println("Error 1 Adding New Container To Database: ", err)
                cStatus := "Error Updating Database"
                return cStatus
        } else {
                cStatus = "GoodToGo"
        }

        lastId, err := res.LastInsertId()
        if err != nil {
                log.Println("Error 2 Adding New Container To Database: ", err)
                cStatus := "Error Updating Database"
                return cStatus
        }

        rowCnt, err := res.RowsAffected()
        if err != nil {
                log.Println("Error 3 Adding New Container To Database: ", err)
                cStatus := "Error Updating Database"
                return cStatus
        }
        log.Printf("ID = %d, affected = %d\n", lastId, rowCnt)

        return cStatus
}
```

```
func updateContainerStatus(db *sql.DB, containerName string, conStatus string) (string) {
        var cStatus string

        log.Println(containerName, " ", conStatus)
          stmt, err := db.Prepare("UPDATE Container_Table SET Container_Status=(?) WHERE
Container_Name=(?)")

        if err != nil {
                log.Println("Error preparing database statement: ", err)
        }

        res, err := stmt.Exec(conStatus ,containerName)
        if err != nil {
                log.Println("Error Updating Database", err)
                cStatus = "Error Updating Database"
        } else {
                cStatus = conStatus
        }

        lastId, err := res.LastInsertId()
        if err != nil {
                log.Println(err)
        }
        rowCnt, err := res.RowsAffected()
        if err != nil {
                log.Fatal(err)
        }
        log.Printf("ID = %d, affected = %d\n", lastId, rowCnt)

        return cStatus
}
```

```
//////////  Runs On Startup Only //////////

func updateDBcontainerStatusOnStartUp(db *sql.DB) (error) {
        var (
                ContainerID int
                baseContainerID int
                customerID int
                containerStatus string
                containerName string

        )
          log.Println("Getting update of current registered containers status from LXC-Control Server")
         rows, err := db.Query("SELECT * FROM Container_Table WHERE Container_Status != 'Deleted'")
        if err != nil {
                log.Println("Error creating MySQL query", err)
        }

        defer rows.Close()

        for rows.Next() {
                        err := rows.Scan(&ContainerID, &baseContainerID, &customerID, &containerStatus, &containerName)
                if err != nil {
                        log.Println("Error reading MySQL query results", err)
                        return err
                }

                if containerStatus, err = dialLXCserver(containerName); err != nil {
                        log.Println("LXC-Control-Server Dial Error", err)
                        return err
                }
                if err == nil {
                                containerStatus = updateContainerStatus(db, containerName, containerStatus)
                                     log.Println(containerName," container updated to: ", containerStatus)
                }
        }

        err = rows.Err()
        if err != nil {
                log.Println("Error reading MySQL query", err)
                return err
        }

        return nil
}
```

```
func dialLXCserver(containerName string, ) (string, error) {
        service := "192.168.1.100:8081" //"127.0.0.1:8081"

        conn, err := net.Dial("tcp", service)
        if err != nil {
                log.Println("LXC Server Dial error: ", err)
                return "", err
        }
        defer conn.Close()

        log.Printf("Dialing LXC Server")

        var b NewContainerJSON
        b.Action = "updateDBcontainerStatus"
        b.ContainerName = containerName


        encoder := json.NewEncoder(conn)
        if err := encoder.Encode(b); err != nil {
                log.Println("JSON Encode error: ", err)
                return "", err
        }

        decoder := json.NewDecoder(conn)

        var c NewContainerJSON
        if err := decoder.Decode(&c); err != nil {
                log.Println("JSON Decode error: ", err)
                return "", err
        }

        log.Println("Master server recieved JSON from Database server: ",c)

        return c.ContainerStatus, nil
}
```

8.2.6 – MySQL Database Schema

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

-- -----------------------------------------------------
-- Schema praxis
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema praxis
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `praxis` DEFAULT CHARACTER SET utf8 ;
USE `praxis` ;

-- -----------------------------------------------------
-- Table `praxis`.`Cms_Table`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `praxis`.`Cms_Table` (
  `Cms_ID` INT NOT NULL AUTO_INCREMENT,
  `Cms_Name` VARCHAR(45) NOT NULL,
  `Cms_Version` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Cms_ID`),
  UNIQUE INDEX `Cms_ID_UNIQUE` (`Cms_ID` ASC))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `praxis`.`Web_Server_Table`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `praxis`.`Web_Server_Table` (
  `Web_Server_ID` INT NOT NULL AUTO_INCREMENT,
  `Sever_Name` VARCHAR(45) NOT NULL,
  `Server_Version` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Web_Server_ID`),
  UNIQUE INDEX `Server_ID_UNIQUE` (`Web_Server_ID` ASC))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `praxis`.`Base_Server_Table`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `praxis`.`Base_Server_Table` (
  `Base_Server_ID` INT NOT NULL AUTO_INCREMENT,
  `Base_Server_Name` VARCHAR(45) NOT NULL,
  `Server_Version` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Base_Server_ID`),
  UNIQUE INDEX `Base_Server_ID_UNIQUE` (`Base_Server_ID` ASC))
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `praxis`.`Base_Container_Clone_Table`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `praxis`.`Base_Container_Clone_Table` (
  `Base_Container_Clone_ID` INT NOT NULL AUTO_INCREMENT,
  `Base_Conatiner_Name` VARCHAR(45) NOT NULL,
  `Web_Server_ID` INT NOT NULL,
  `Cms_ID` INT NOT NULL,
  `Base_Server_ID` INT NOT NULL,
  PRIMARY KEY (`Base_Container_Clone_ID`),
  INDEX `fk_Base_Container_Clone_Table_1_idx` (`Base_Server_ID` ASC),
  INDEX `fk_Base_Container_Clone_Table_2_idx` (`Cms_ID` ASC),
  INDEX `fk_Base_Container_Clone_Table_3_idx` (`Web_Server_ID` ASC),
  UNIQUE INDEX `Base_Container_Clone_ID_UNIQUE` (`Base_Container_Clone_ID` ASC),
  CONSTRAINT `fk_Base_Container_Clone_Table_1`
    FOREIGN KEY (`Base_Server_ID`)
    REFERENCES `praxis`.`Base_Server_Table` (`Base_Server_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Base_Container_Clone_Table_2`
    FOREIGN KEY (`Cms_ID`)
    REFERENCES `praxis`.`Cms_Table` (`Cms_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Base_Container_Clone_Table_3`
    FOREIGN KEY (`Web_Server_ID`)
    REFERENCES `praxis`.`Web_Server_Table` (`Web_Server_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `praxis`.`Customer_Table`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `praxis`.`Customer_Table` (
  `Customer_ID` INT NOT NULL AUTO_INCREMENT,
  `Customer_Type` SET('Customer', 'Admin') NOT NULL DEFAULT 'Customer',
  `Customer_UserName` VARCHAR(45) NOT NULL,
  `Customer_First_Name` VARCHAR(45) NOT NULL,
  `Customer_Family_Name` VARCHAR(45) NOT NULL,
  `Customer_Phone_Number` VARCHAR(45) NOT NULL,
  `Customer_Email` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Customer_ID`),
  UNIQUE INDEX `Customer_ID_UNIQUE` (`Customer_ID` ASC))
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `praxis`.`Container_Table`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `praxis`.`Container_Table` (
  `Container_ID` INT NOT NULL AUTO_INCREMENT,
  `Base_Container_Clone_ID` INT NOT NULL,
  `Customer_ID` INT NOT NULL,
  `Container_Status` SET('Running', 'Stopped', 'Suspended') NOT NULL DEFAULT 'Stopped',
  `Container_Name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Container_ID`),
  INDEX `fk_Container_Table_1_idx` (`Base_Container_Clone_ID` ASC),
  INDEX `fk_Container_Table_2_idx` (`Customer_ID` ASC),
  UNIQUE INDEX `Container_ID_UNIQUE` (`Container_ID` ASC),
  UNIQUE INDEX `Container_Name_UNIQUE` (`Container_Name` ASC),
  CONSTRAINT `fk_Container_Table_1`
    FOREIGN KEY (`Base_Container_Clone_ID`)
    REFERENCES `praxis`.`Base_Container_Clone_Table` (`Base_Container_Clone_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Container_Table_2`
    FOREIGN KEY (`Customer_ID`)
    REFERENCES `praxis`.`Customer_Table` (`Customer_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `praxis`.`Container_Snapshot_Table`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `praxis`.`Container_Snapshot_Table` (
  `Container-Snaphot_ID` INT NOT NULL AUTO_INCREMENT,
  `Container_ID` INT NOT NULL,
  `Container_Snapshot_Name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Container-Snaphot_ID`),
  INDEX `fk_Container_Snapshot_Table_1_idx` (`Container_ID` ASC),
  UNIQUE INDEX `Container-Snaphot_ID_UNIQUE` (`Container-Snaphot_ID` ASC),
  CONSTRAINT `fk_Container_Snapshot_Table_1`
    FOREIGN KEY (`Container_ID`)
    REFERENCES `praxis`.`Container_Table` (`Container_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `praxis`.`Container_Backup_Table`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `praxis`.`Container_Backup_Table` (
  `Container_Backup_ID` INT NOT NULL AUTO_INCREMENT,
  `Container_ID` INT NOT NULL,
  `Container_Backup_Name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Container_Backup_ID`),
  INDEX `fk_Container_Backup_Table_1_idx` (`Container_ID` ASC),
  UNIQUE INDEX `Container_Backup_ID_UNIQUE` (`Container_Backup_ID` ASC),
  CONSTRAINT `fk_Container_Backup_Table_1`
    FOREIGN KEY (`Container_ID`)
    REFERENCES `praxis`.`Container_Table` (`Container_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

-- -----------------------------------------------------
-- Data for table `praxis`.`Cms_Table`
-- -----------------------------------------------------
START TRANSACTION;
USE `praxis`;
INSERT INTO `praxis`.`Cms_Table` (`Cms_ID`, `Cms_Name`, `Cms_Version`) VALUES (1,
'Wordpress', '4.5.1');

COMMIT;


-- -----------------------------------------------------
-- Data for table `praxis`.`Web_Server_Table`
-- -----------------------------------------------------
START TRANSACTION;
USE `praxis`;
INSERT INTO `praxis`.`Web_Server_Table` (`Web_Server_ID`, `Sever_Name`, `Server_Version`)
VALUES (1, 'Apache', '2');

COMMIT;
```

```
-- -----------------------------------------------------
-- Data for table `praxis`.`Base_Server_Table`
-- -----------------------------------------------------
START TRANSACTION;
USE `praxis`;
INSERT INTO `praxis`.`Base_Server_Table` (`Base_Server_ID`, `Base_Server_Name`,
`Server_Version`) VALUES (1, 'Ubuntu', '14.04.4');

COMMIT;


-- -----------------------------------------------------
-- Data for table `praxis`.`Base_Container_Clone_Table`
-- -----------------------------------------------------
START TRANSACTION;
USE `praxis`;
INSERT INTO `praxis`.`Base_Container_Clone_Table` (`Base_Container_Clone_ID`,
`Base_Conatiner_Name`, `Web_Server_ID`, `Cms_ID`, `Base_Server_ID`) VALUES (1, 'clone-
u1404-A2-MySQL-WP', 1, 1, 1);

COMMIT;


-- -----------------------------------------------------
-- Data for table `praxis`.`Customer_Table`
-- -----------------------------------------------------
START TRANSACTION;
USE `praxis`;
INSERT INTO `praxis`.`Customer_Table` (`Customer_ID`, `Customer_Type`,
`Customer_UserName`, `Customer_First_Name`, `Customer_Family_Name`,
`Customer_Phone_Number`, `Customer_Email`) VALUES (1, 'Customer', 'david.monaghan',
'David', 'Monaghan', '0866602797', 'david.monaghan@mycit.ie');

COMMIT;


-- -----------------------------------------------------
-- Data for table `praxis`.`Container_Table`
-- -----------------------------------------------------
START TRANSACTION;
USE `praxis`;
INSERT INTO `praxis`.`Container_Table` (`Container_ID`, `Base_Container_Clone_ID`,
`Customer_ID`, `Container_Status`, `Container_Name`) VALUES (1, 1, 1, 'Stopped',
'david.monaghan-testContainer');

COMMIT;
```