Final Project Report:

CLOUD BURSTING TO AWS EC2 FROM VCL

Team 2
Rishabh Jain (Lead)
Abhash Jain (Vice-Lead)
Henil Shah
Srijani Hariraman
Sriram Guddati
Computer Science Department
NC State University

Abstract

We have implemented a solution to enhance the functionality of current VCL architecture to handle resource scarcity when there are load spikes on the system. We have developed and integrated a new provisioning engine that will enable the system to detect lack of resources and burst to the AWS EC2 public cloud to provision resources on-demand. This feature is transparent to the user and allows seamless instance reservations on AWS EC2 with minimum cost. We compare, investigate and discuss different options, design approaches and implementation strategies in this report to come up with a solution that can cater to a seamless and secure cloud burst mechanism for a large scale system.

Contents

List of Figures

List of Tables

Introduction

1.1 Problem Statement

We are provided with the task to improve the existing VCL architecture to cater to its increasing customer demand for cloud computing resources. Our cloud service faces resource scarcity on certain times during load spikes. The important aspect of this problem is that the increased customer demand is intermittent. Therefore, a compelling solution to this issue is to enhance the existing architectural design by adding the ability of bursting to a public cloud when there are not enough resources on VCL.
The project focuses on making this additional functionality transparent to the users to make sure that they can use this facility seamlessly. At the same time, the cost of bursting is kept at minimum by making sure that at any request, we are not redirecting it to reserve resources on AWS unnecessarily.

1.2 Motivation

The motivation of this project is to give users seamless experience with minimum added cost in increasing the capacity of entertaining more requests.
We had two options when deciding on a way to increase the capacity of the existing infrastructure:

1. Adding capital equipment to the existing VCL Cloud
2. Bursting to public cloud on-demand.

The increased customer demand is intermittent. Therefore, buying new physical infrastructure to VCL will not be cost effective. Also, the operational cost of this infrastructure will be high. Bursting to public cloud on-demand is a fair choice since that is a more economic option in this case.
When it comes to public cloud-vendors, there are plethora of options to choose from. Some of them are Microsoft Azure, Amazon AWS and Google Public Cloud. One of the chief reasons we picked AWS was because it is a pioneer in the market. It also provides a free tier student account which would make developing and testing our project very convenient for us. In addition to that, it easy to use, supports REST APIs and has a good community support.

Thus, our project aims to implement a seamless cloud bursting functionality in VCL to make it able to extend its requests to AWS in the case of resource scarcity.

1.3 Environment

For the demo and testing of this cloud burst environment, we are setting up the VCL test bed over CentOS 7 and also using the same image for VCL and AWS reservations.
Also, to burst to the public cloud AWS EC2, we are using the AWS educate account provided to us by the instructor.

1.4 Issues

A project isn't done right, if the team doing it doesn't face any issues. This section lists a few of the issues that we faced.

1. Unfamiliarity with PHP, Perl and Javascript:
   As a team we were very excited with the opportunity to work on a hybrid cloud environment. Understanding the VCL architecture was fun. But when we started the code-review, none of us had any experience with perl or php which basically sums up whole of VCL. We worked around it and developed our code in python, but faced multiple issues with calling our python code from php scripts. To debug, we performed inspection through console in browser, added print statements and fixed file permissions issues with the help of google.

2. Test-bed Setup Issue:
   We were able to setup the VCL test with three computers available in a single Compute Node. While creating a reservation, we are getting stuck at Reservation creation and Deletion on VCL also gets stuck with a need to remove the entry manually from the DB.

   Currently, this issue doesn't affect the cloud burst logic. We suspect the issue is with the CentOS image that we are using. We are in the progress of analyzing the logs and deciding the next steps accordingly.

3. AWS Key expiry:
   The AWS key that is used for calling the AWS EC2 APIs from VCL node expires every 3 hours and then have to be manually updated. This is a limitation with the AWS educate account and shouldn't be a problem in a solution using dedicated AWS account. For us,

the AWS instance delete cron job fails after every 3 hour until the key is updated due to this limitation.

Requirements

To begin working on a project, it is essential that the requirements of the project are first analyzed and understood by the members of the team which helps the team to go about the project by making sure that the requirements are met.

While considering these requirements, we can divide them into two types:

- Functional Requirements
- Non-functional Requirements

2.1 Functional Requirements

A functional requirement is that which specifies something the system should be doing. The expected behaviour or the functions of the system is being listed out under the functional requirements.

FR 1 Provisioning of AWS Instances

- FR 1.1 Creation of AWS Instances through EC2 APIs from VCL
- FR 1.2 Busting to AWS to provision the resources has to be managed through a single web portal. This bursting has to be seamless and transparent to the user.
- FR 1.3 Conditional burst logic to be implemented which checks for resource availability on VCL before provisioning the instances on AWS

FR 2 Deletion of Instances

- FR 2.1 Delete the AWS instances on user request
- FR 2.2 Cron job to check if there is a reservation expiry and delete the instance
- FR 2.3 Clean up the VM storage and remove all data that has been entered by the user

FR 3 Store AWS instances data

- FR 3.1 Populate the instance details of AWS EC2 reservation in a table in the VCL database
- FR 3.2 Delete the instance details from the table when the instance is deleted

2.2 Non-functional Requirements

The non-functional requirements can be defined as the quality attributes of a system. Under the non-functional requirements, the system operation is checked for its performance, scalability, availability, regulatory and security in our project.

NFR 1 Performance : Delay in bursting to the cloud and making an EC2 reservation in the public cloud should be kept at a minimum. In our system, the bursting to the cloud and reservation of an EC2 instance should be maintained at a minimum value.

NFR 2 Scalability : Multiple users should be able to burst to AWS. Mapping between the instance id and username has to be done which makes each user's reservation independent of each other

NFR 3 Availability: Bursting to AWS from VCL should not affect the normal functioning of the system. The VCL resources should be made available to the user even after bursting to the AWS has been done.

NFR 4 Regulatory: Cost of bursting to the AWS should be kept at a minimum. Bursting has to be done only if the resource is not available on VCL. Regulate the upper-limit of the reservation time of the instances on AWS to avoid incurring additional cost.

NFR 5 Security: User has to be provided with a user name and password to log into the EC2 instance. Block root/admin access for the AWS instance to the user

2.3 List of Tasks

Now that we have established the what (requirements) of the project, below is the order of the tasks that have to be performed to build a system that adheres with the above listed requirements.

- Install host OS (CentOS 7) on the available machine
- Connect the machine to the internet by performing the necessary networking configuration
- Setup the VCL private cloud on the machine
- Add the code form Git-hub to enable the cloud bursting features
- Configure AWS EC2 APIs for AWS VMs to be spawned from EC2
- Verify and Validate the System

2.4 System Environment

The hardware specifics of the test-bed in Net labs are listed below on which we have implemented the Sandbox version of VCL with cloud bursting logic.

System Environment			
Host OS	CPU	Memory	Storage
CentOS 7	Intel(R) Xeon(R) CPU X3430 @2.40GHz (4 core)	8 GB	500 GB

Table 2.1: System Environment

System Design

To start with designing a cloud burst solution, we first have to explore the existing infrastructure i.e. VCL, analyze what features it provides, figure out what is missing and required and then come up with a solution that adds the missing requirement to the existing solution without affecting the functionality of the VCL and add the required functionality.

3.1 VCL ARCHITECTURE

The VCL architecture is modular which comprises of the following components.

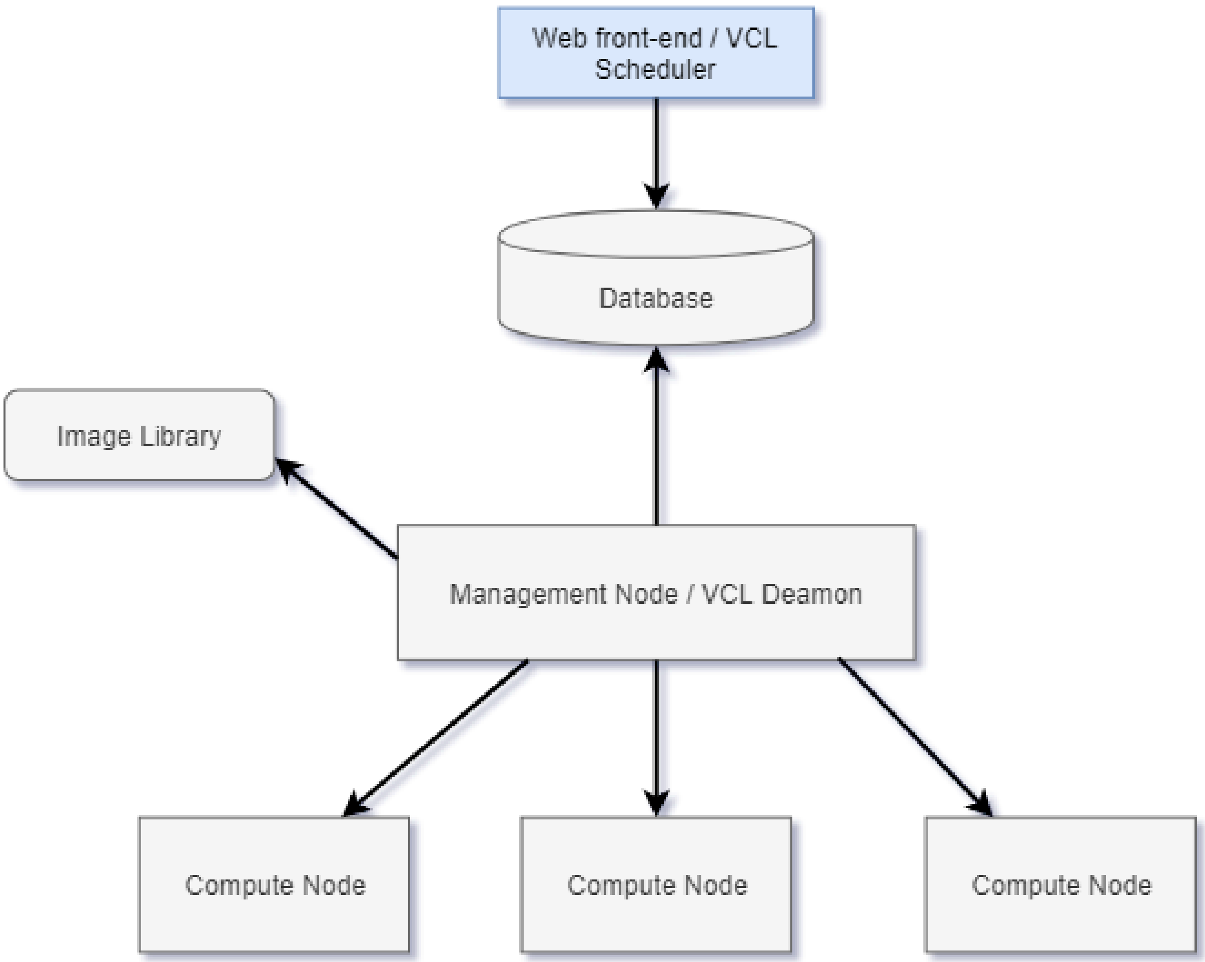


Figure 3.1: VCL Architecture

1. Web-portal: This is the web-interface through which the user logs in to create/manage/delete reservations. It acts as the access point for the end user. It also performs Authentication and Authorization for the users. It sends post requests to the database to keep a record of all requests.

   The VCL API uses the XML RPC protocol which is a remote procedure call protocol which uses XML to encode its calls and HTTP as a transport mechanism. The front end is also used by the admin to perform administrative tasks such as creating and managing Computers, Management node, images and user groups.

2. Database : VCL has the option to integrate with a MySQL or a MariaDB database. The VCL database provides a platform to have a consistent view of the reservation requests, resource inventory, resource mappings, user privileges, Image profiles and meta-data between the Web portal and VCL Management Nodes. Both the front end and vcld daemon have read and write privileges to the VCL database.

3. Management Node: The processing engine : The management nodes each control a subset of the VCL resources. These can be blades or virtual machines. Currently, a set of individual blades or virtual machines can only be managed by a single management node. There can be more than one management node.

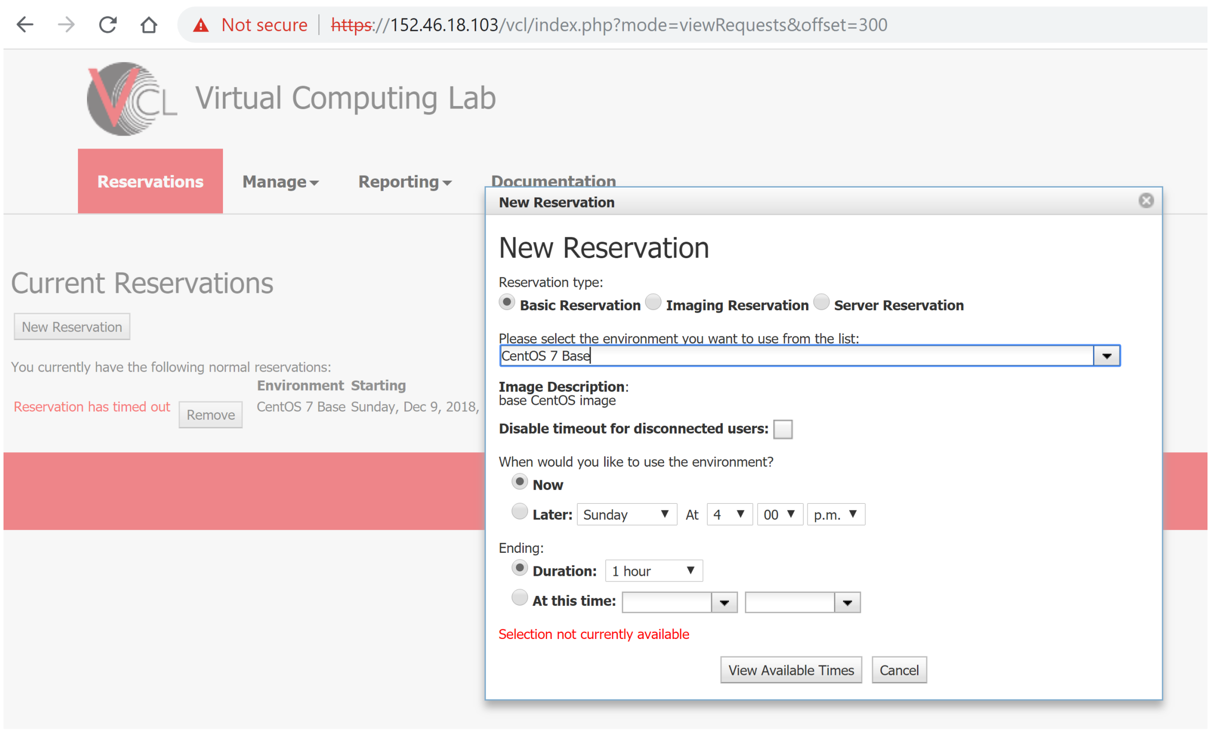
   vcld: VCL middle-ware that processes reservations/jobs assigned by the VCL web portal. Based on the type of environment requested whether it is a bare metal image, or a virtual machine image, vcld ensures the image is loaded and makes it available for the requesting user.

4. Image Library : It holds the available Image files, meta-data, Linux Install tree and Image profiles to be loaded into Computers/VMs on the compute nodes.

5. Compute Nodes : These are the compute machines which holds the resources required to allocate IaaS to the users. The compute nodes can be used individually as bare-metal machines or can host multiple VMs through the Libvirt Virtualization API/ VMware. These are managed by the management nodes.

3.2 DESIGN WORK FLOW

VCL works well with providing IaaS until it has enough compute resources available. Once all the resources are allocated, it displays the following message:


Not secure | ~~https~~://152.46.18.103/vcl/index.php?mode=viewRequests&offset=300

Virtual Computing Lab

Reservations Manage Reporting Documentation

Current Reservations

New Reservation

You currently have the following normal reservations:

		Environment	Starting
Reservation has timed out	Remove	CentOS 7 Base	Sunday, Dec 9, 2018,

New Reservation

New Reservation

Reservation type:
[✓ Basic Reservation] Imaging Reservation Server Reservation

Please select the environment you want to use from the list: CentOS 7 Base

Image Description: base CentOS image

Disable timeout for disconnected users: ☐

When would you like to use the environment?
[✓ Now]
Later: Sunday At 4 00 p.m.

Ending:
[✓ Duration: 1 hour]
At this time:

Selection not currently available

View Available Times Cancel

Figure 3.2: VCL Not Available

Now that we know what features VCL provides by default, we have created a work flow for the expected flow in case of create/delete reservation requirement:

1. User logs in to the VCL account.
2. VCL front-end takes the user input in the form of create/delete reservation.
3. For create reservation, check if enough resources available on VCL, if yes, process the request through vcld.
4. If no, then spawn an instance on AWS EC2 for the user, create user/password pair and add it to the vcl database.
5. For delete request, check if the instance ID belongs to VCL, if yes, let vcld delete that reservation and clear the resources occupied by it.
6. If not, delete the AWS instance and clear the database entry and clean up the resources on AWS EC2.

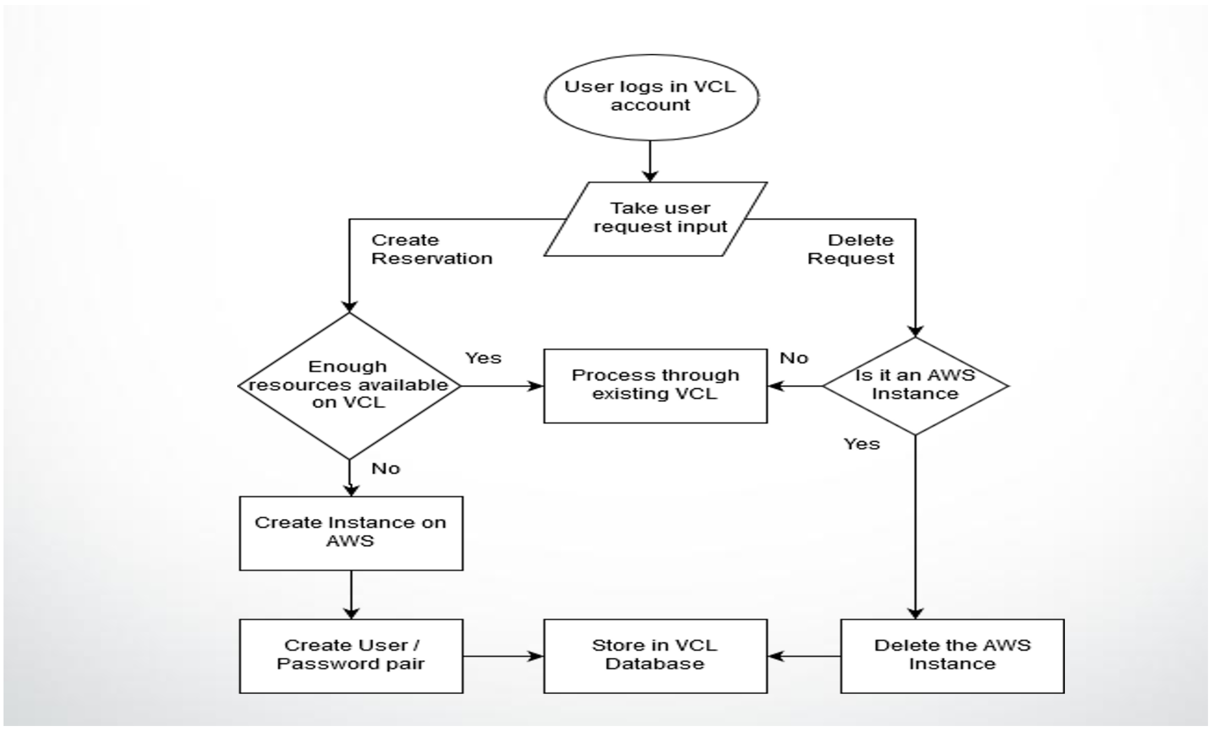


Figure 3.3: Design Work flow

3.3 Bursting to AWS

To create a reservation, Cloud Burst logic passes the user and instance details that has to be created on AWS to the AWS handler. AWS Handler uses the create_instance module with the Image ID, Min Count, Max Count, Instance type and Key Name passed as parameters. Generates a user name and password for the user to log into the EC2 instance after password authentication has been enabled on the instance. The instance reservation details from AWS is returned to the AWS handler which is populated to the VCL database.

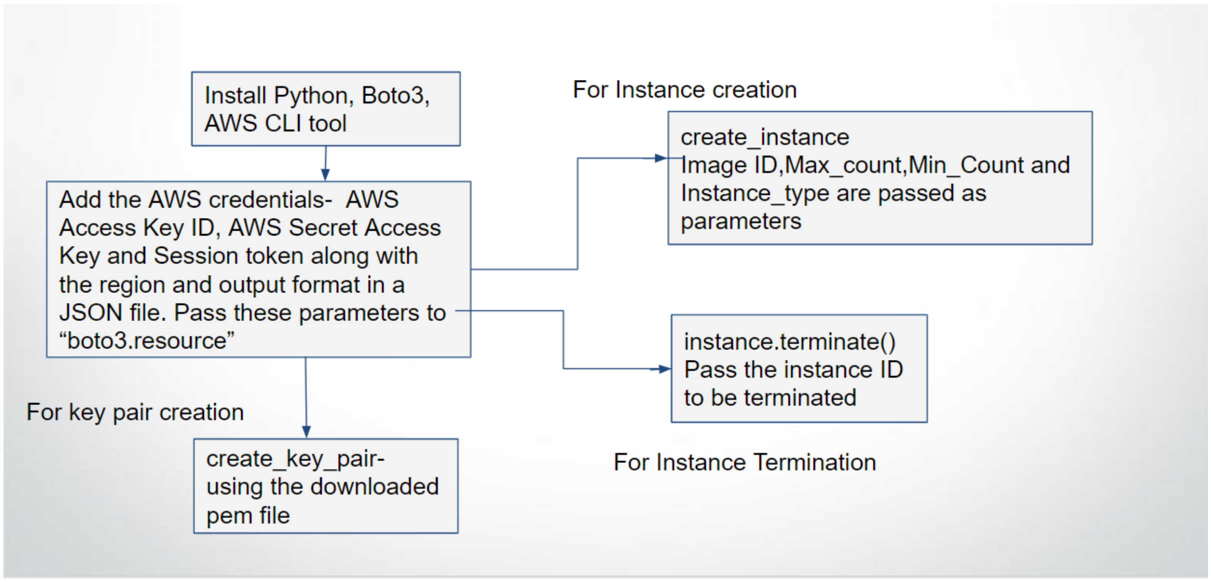

Install Python, Boto3, AWS CLI tool
Add the AWS credentials- AWS Access Key ID, AWS Secret Access Key and Session token along with the region and output format in a JSON file. Pass these parameters to "boto3.resource"
For Instance creation
create_instance Image ID,Max_count,Min_Count and Instance_type are passed as parameters
instance.terminate() Pass the instance ID to be terminated
For Instance Termination
For key pair creation
create_key_pair- using the downloaded pem file

Figure 3.4: AWS Reservation Flow

3.4 Cloud Burst Architecture

With the given requirement the additional logic to handle the cloud burst scenario is written in the cloud burst logic module and AWS handler.

This additional logic can be integrated with the VCL architecture at two locations:

- VCL front-end(Web-portal)
- VCL back-end(vcld)

There are two reasons for choosing to integrate the additional logic between the front-end and database:

1. There is a non-functional requirement that states that the burst to the public cloud should not interfere with the normal functioning and reservation of VCL
2. While going through the logic of VCL code, we found there is an existing module in front-end which checks for available compute resources and seemed like a perfect integration point for our code.

With these points in mind, we came up with the below architecture:

The additional components are explained below:

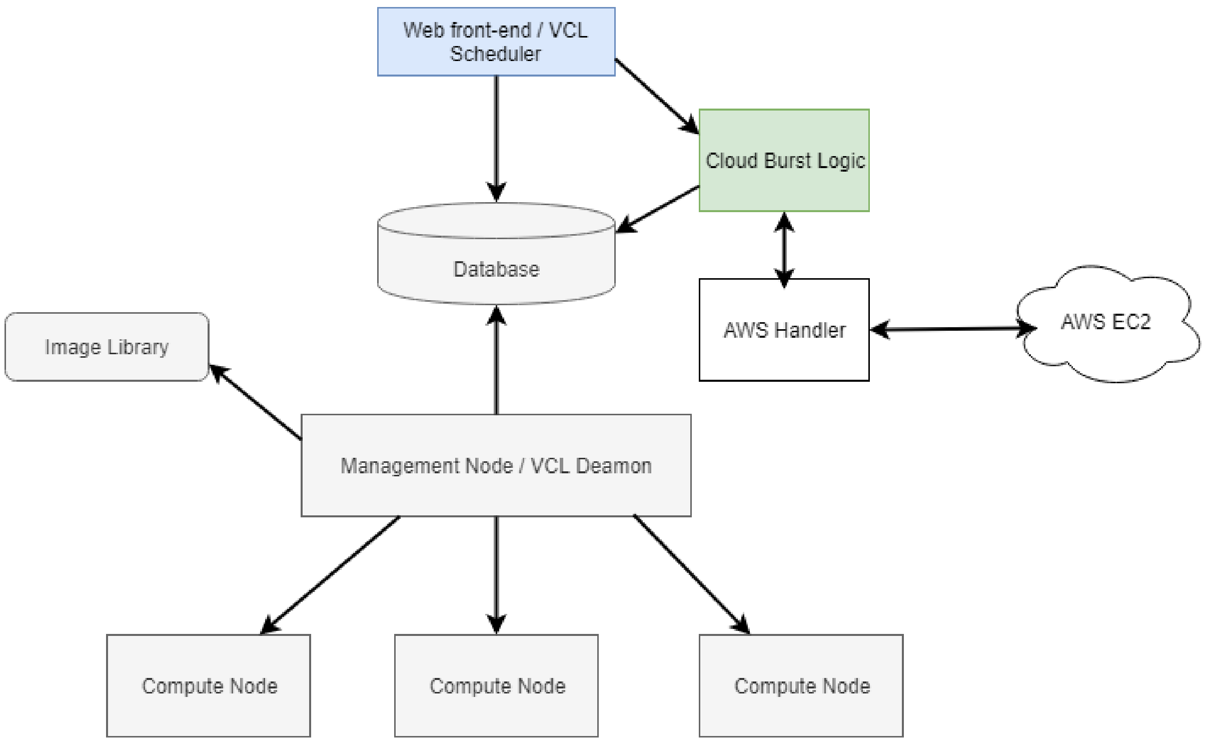


Figure 3.5: Cloud Burst Architecture

- Cloud Burst logic : This module gets called when there is a reservation request and there are no resources available in VCL to attend to this request. It stores the parameters of the requested reservation, calls the AWS handler. After receiving the instance details of created instance, combines the locally stored data of the instance and stores it in the database.
- AWS handler : This module handles the create/list/delete instance requests for the AWS EC2 instances and is called by the cloud burst module. It performs a set of operations such as creating user password pair for the instance.

VCL Test-Bed Preparation

4.1 Pre-requisites

1. Install CentOS 7 on the host machine.
2. Connect the Gigabit interface 1 to the rack switch.
3. Fetch a public IP address by running the "dhclient" command.
4. Login to the session through ssh.
5. Install the prerequisite libraries listed below.

4.2 Setting up KVM

1. Run the following yum commands to install KVM and its associated packages.

   ```
   # yum install qemu-kvm qemu-img virt-manager bridge-utils
   # yum install libvirt libvirt-python libvirt-client virt-install virt-viewer
   ```

2. Start and enable the libvirtd service

   ```
   # systemctl start libvirtd
   # systemctl enable libvirtd
   ```

3. Run the beneath command to check whether KVM module is loaded or not

   ```
   # lsmod | grep kvm
   ```

4. In Case you have Minimal CentOS 7 and RHEL 7 installation , then virt-manger will not start for that you need to install x-window package.

   ```
   # yum install xorg-x11-xauth xorg-x11-fonts-* xorg-x11-utils -y
   ```

5. Reboot the Server and then try to start virt manager.

4.3 Get the VCL code for installation

1. Make sure that you are logged in to the machine and have internet connectivity
2. Pull from the VCL Apache mirror site

   ```
   # wget http://apache.cs.utah.edu/vcl/2.5/apache-VCL-2.5.tar.bz2
   ```

3. untar the file

   ```
   # tar -jxvf apache-VCL-2.5.tar.bz2
   ```

4.4 Database Installation

1. Install MariaDB Server

   ```
   # yum install mariadb-server -y
   ```

2. Configure the database daemon to start automatically

   ```
   # /sbin/chkconfig --level 345 mariadb on
   ```

3. Start the database daemon

   ```
   # /sbin/service mariadb start
   ```

4. Create the VCL Database
   (a) Run the MySQL command-line client

   ```
   # mysql
   ```

   (b) Create a database

   ```
   > CREATE DATABASE vcl;
   ```

   (c) Create a user with SELECT, INSERT, UPDATE, DELETE, and CREATE TEMPORARY TABLES privileges on the database you just created (NOTE Use your own password):

   > GRANT SELECT,INSERT,UPDATE,DELETE,CREATE TEMPORARY TABLES ON vcl.* TO 'vcl_admin'@'localhost' IDENTIFIED BY 'Admin123~';

(d) Exit the MySQL command-line client

```
> exit
```

5. Import the vcl.sql file into the database. The vcl.sql file is included in the mysql directory within the Apache VCL source code

```
# mysql vcl < apache-VCL-2.5/mysql/vcl.sql
```

4.5 Install and Configure the Web Components

1. Install the Required Linux Packages PHP Modules

   (a) If your web server is running a Red Hat-based OS, the required components can be installed with:

   ```
   # yum install httpd mod_ssl php php-mysql php-xml php-xmlrpc php-ldap -y
   ```

   (b) Configure the web server daemon (httpd) to start automatically:

   ```
   # /sbin/chkconfig --level 345 httpd on
   ```

   (c) Start the web server daemon

   ```
   # /sbin/service httpd start
   ```

   (d) If SELinux is enabled, run the following command to allow the web server to connect to the database:

   ```
   # /usr/sbin/setsebool -P httpd_can_network_connect=1
   ```

   (e) If the iptables firewall is being used, port 80 and 443 should be opened up in the iptables config file:

   ```
   # vi /etc/sysconfig/iptables
   ```

   Add these rules:

   ```
   -A INPUT -m state --state NEW -p tcp --dport 80 -j ACCEPT
   -A INPUT -m state --state NEW -p tcp --dport 443 -j ACCEPT
   ```

   Restart iptables

   ```
   # service iptables restart
   ```

2. Install the VCL Frontend Web Code

   (a) Copy the web directory to a location under the web root of your web server and navigate to the destination .ht-inc sub-directory:

   ```
   #cp −ar apache−VCL−2.5/web/ /var/www/html/vcl−2.5
   #ln −s /var/www/html/vcl−2.5 /var/www/html/vcl
   #cd /var/www/html/vcl/.ht−inc
   ```

   (b) If SELinux is enabled, run the following command to set the context of the web code to httpd_sys_content_t

   ```
   #chcon −R −t httpd_sys_content_t /var/www/html/vcl−2.5
   ```

   (c) Copy secrets-default.php to secrets.php:

   ```
   #cp secrets−default.php secrets.php
   ```

   (d) Edit the secrets.php file:

   ```
   #vi secrets.php
   ```

   Set the following variables to match your database configuration: <TBE> $vclhost$vcldb $vclusername$vclpassword Create random passwords for the following variables: $cryptkey(generatewith"opensslrand32|base64")$pemkey Save the secrets.php file

   (e) Run the genkeys.sh

   ```
   #./genkeys.sh
   ```

   (f) Copy conf-default.php to conf.php:

   ```
   #cp conf−default.php conf.php
   ```

   (g) Modify conf.php to match your site:

   (h) Set the owner of the .ht-inc/maintenance and .ht-inc/cryptkey directories to the web server user (normally 'apache'):

   ```
   chown apache maintenance
   chown apache cryptkey
   ```

   (i) If SELinux is enabled, run the following command to allow the web server to write to maintenance and cryptkey

```
chcon -t httpd_sys_rw_content_t maintenance
chcon -t httpd_sys_rw_content_t cryptkey
```

(j) Open the testsetup.php page in a web browser:
https://localhost/vcl/testsetup.php

3. Log In to the VCL Website

   (a) Open the index.php page in your browser (https://localhost/vcl/index.php)
      i. Select Local Account
      ii. Username: admin
      iii. Password: adminVc1passw0rd

   (b) Set the admin user password (DO NOT skip this step):
      i. Click User Preferences
      ii. Enter the current password: adminVc1passw0rd
      iii. Enter a new password
      iv. Click Submit Changes

4. Add a Management Node to the Database

   (a) Click the Management Nodes link
      - Select Edit Management Node Profiles
      - Click Submit
      - Click Add New Management Node
      - Fill in these required fields:
        – Hostname - The name of the management node server. This value doesn't necessarily need to be a name registered in DNS nor does it need to be the value displayed by the Linux hostname command. For example, if you are installing all of the VCL components on the same machine you can set this value to localhost.
        – IP address - the public IP address of the management node
        – SysAdmin Email Address - error emails will be sent to this address
        – Install Path - this is the parent directory under which image files will be stored - only required if doing bare metal installs or using VMWare with local disks

    - End Node SSH Identity Key Files - enter /etc/vcl/vcl.key unless you know you are using a different SSH identity key file

- Click Add Management Node
- A dialog will pop up informing you to add the management node to a group, read it and click Close
- select the allManagementNodes group on the right
- click <-Add
- click Close

4.6 Install & Configure the Management Node Components

1. Install the VCL Management Node Code - Perl Daemon

```
cp -ar apache-VCL-2.5/managementnode /usr/local/vcl-2.5
ln -s /usr/local/vcl-2.5 /usr/local/vcl
```

2. Install the Required Linux Packages Perl Modules

```
perl /usr/local/vcl/bin/install_perl_libs.pl
```

3. Configure vcld.conf

   (a) Create the /etc/vcl directory:

   ```
   mkdir /etc/vcl
   ```

   (b) Copy the stock vcld.conf file to /etc/vcl:

   ```
   cp /usr/local/vcl/etc/vcl/vcld.conf /etc/vcl
   ```

   (c) Edit /etc/vcl/vcld.conf:

   ```
   vi /etc/vcl/vcld.conf
   ```

   The following lines must be configured in order to start the VCL daemon (vcld) and allow it to check in to the database:

   - FQDN - the fully qualified name of the management node, this should match the name that was configured for the management node in the database

- server - the IP address or FQDN of the database server
- LockerWrtUser - database user account with write privileges
- wrtPass - database user password
- xmlrpc_pass - password for xmlrpc api from vcld to the web interface(can be long). This will be used later to sync the database vclsystem user account
- xmlrpc_url - URL for xmlrpc api https://my.server.org/vcl/index.php?mode=xmlrpccall

(d) Save the vcld.conf file

4. Configure the SSH Client The SSH client on the management node should be configured to prevent SSH processes spawned by the root user to the computers it controls from hanging because of missing or different entries in the known_hosts file.

   (a) Edit the ssh_config file:

   ```
   vi /etc/ssh/ssh_config
   ```

   (b) Set the following parameters:

   ```
   UserKnownHostsFile /dev/null
   StrictHostKeyChecking no
   ```

5. Install and Start the VCL Daemon (vcld) Service

   (a) Copy the vcld service script to /etc/init.d and name it vcld:

   ```
   # cp /usr/local/vcl/bin/S99vcld.linux /etc/init.d/vcld
   ```

   (b) Add the vcld service using chkconfig:

   ```
   # /sbin/chkconfig --add vcld
   ```

   (c) Configure the vcld service to automatically run at runtime levels 3-5:

   ```
   # /sbin/chkconfig --level 345 vcld on
   ```

   (d) Start the vcld service:

   ```
   # /sbin/service vcld start
   ```

   (e) Check the vcld service by monitoring the vcld.log file:

   ```
   # tail -f /var/log/vcld.log
   ```

6. Set the vclsystem account password for xmlrpc api Using the vcld -setup tool, set the vclsystem account. This is needed to properly use the block allocation features.

   ```
   /usr/local/vcl/bin/vcld --setup
   ```

   Select the options listed below to set the password. When prompted paste or type the password from xmlrpc_pass variable in the vcld.conf file and hit enter.

   ```
   Select 5. Set Local VCL User Account Password
   Select 2. vclsystem
   Enter the password you set for xmlrpc_pass in /etc/vcl/vcld.conf
   ```

   After setting the password for the vclsystem user, test that RPC-XML Access works correctly by selecting

   ```
   2: Test RPC-XML Access
   ```

Function Modules and Implementation

Now that we have a new Architecture, the flow of API calls between the modules changes. This section gives an overview of the function modules and their interaction. Image 5.1 shows the overall functional block diagram of the implementation:

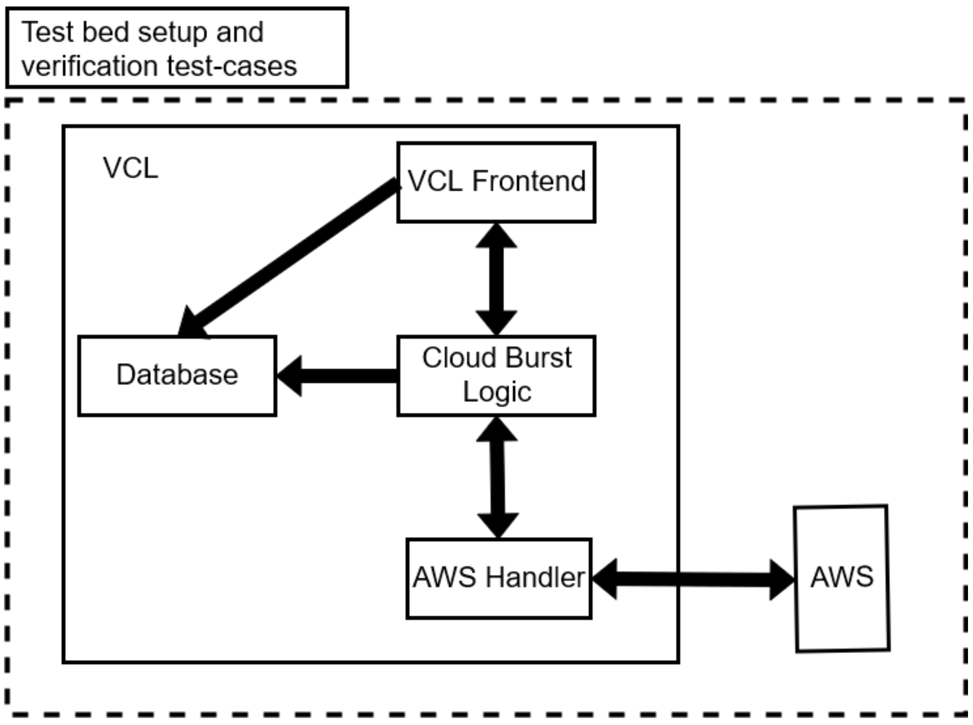


Figure 5.1: Functional Block Diagram

In this section, we discuss in detail how each module is implemented and how it communicates with the rest.

5.1 Cloud Burst Logic Module

Cloud Burst Logic is the central module that passes information between front-end, database and AWS handler. Figure 5.2 shows its role in the overall design. Its responsibilities include:

1. Identifying that VCL does not have sufficient resources for given user request and set the condition to spawn AWS instance true.
2. Call create instance script of AWS handler script with the request information received from the front-end.

3. Call delete instance script of AWS handler script when front-end signals that user wants to explicitly delete an AWS instance.

4. Reflect these changes in database table.

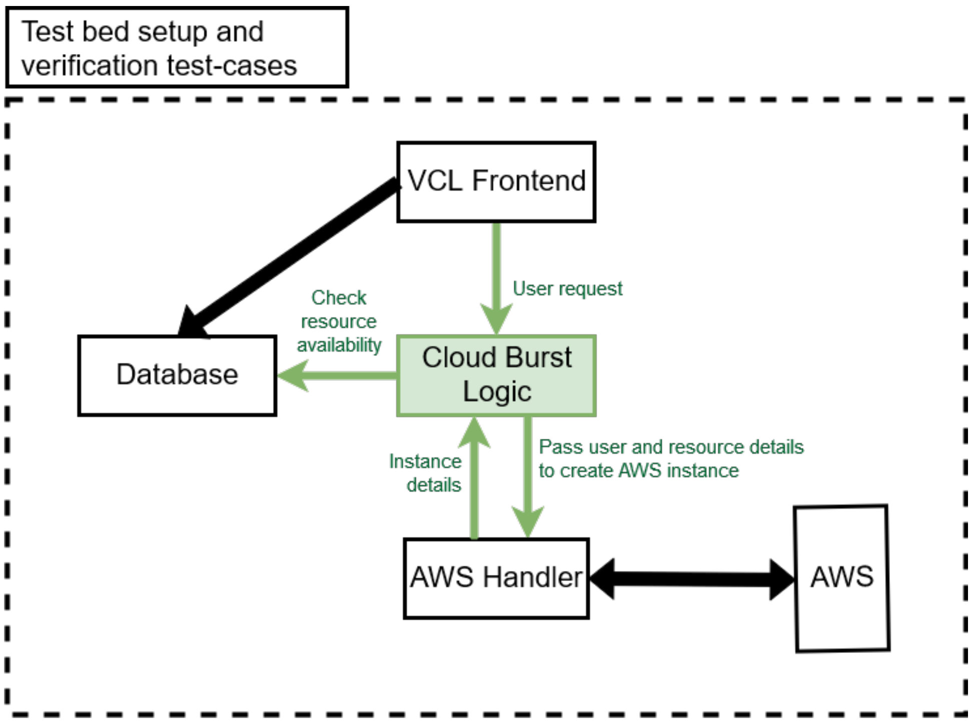

Test bed setup and verification test-cases
VCL Frontend
User request
Check resource availability
Database
Cloud Burst Logic
Instance details
Pass user and resource details to create AWS instance
AWS Handler
AWS

Figure 5.2: Cloud Burst Logic

5.1.1 Identifying Resource Scarcity on VCL

This part of the Cloud Burst Logic module sits in the existing PHP code of the frontend.

1. It uses an existing front-end function to check whether VCL has enough resources for the user request for the time that user requested.

2. In the existing VCL code base, this condition is checked before user clicks on Create Reservation button. Every time user changes input options for the New Reservation dialogue box, the isAvailable() function is called with currently filled input parameters.

3. If VCL does not have enough resources, it will set the AWS condition indicating that on user's click on the Create Reservation button, front-end should redirect this request to the CGI script that takes care of instance creation on AWS.

4. Cloud-burst logic resets AWS condition to false everytime before checking for resources. This takes care of the case where initially, a request was not satisfied by VCL and redirected to AWS. But later on some VCL resources were freed and could suffice for the next upcoming user request. Resetting the AWS condition every time ensures that this request is reserved on VCL and not sent to AWS.

5.1.2 Handling Create and Delete Requests for AWS

This requires Cloud Burst Logic Module to talk to both AWS handler and Database. Its implementation involves CGI and Python scripts.

1. Whenever the CGI script for create instance is called, the script extracts the user-name and duration of the request and calls the create script of AWS handler.
2. It then waits for the AWS creation to finish. As soon as the AWS handler returns the created instance's detail, the Cloud Burst Module creates a connection with the database.
3. To create this connection, we use mysql-connector python package.
4. Once we connect to the database, we populate the aws_reservations table with information of the newly created instance (i.e. instance-id user name, password, public DNS etc).
5. Whenever the delete cgi script receives a user request for instance deletion from front-end, it takes the instance-id and calls AWS handler to delete that instance.
6. When AWS handler returns success for that instance, it deletes the respective row from the aws_reservations table in the database.

5.2 DATABASE MODULE

VCL uses MariaDB for its database component and all VCL reservation details are stored in it. Complying with the requirement of seamless cloud burst and managing the AWS instances through VCL, it fits us to store the persistent data for AWS instances in VCL Database.

The database schema for aws reservation is shown below. And all the required details are inserted to the table for later use.

Figure 5.4 shows overall responsibilities of database module.

aws_reservation	
instance_id	VARCHAR(50) - PRIMARY KEY
username	VARCHAR(50)
start_time	datetime
end_time	datetime
daterequested	datetime
instance_username	VARCHAR(50)
password	VARCHAR(50)
dns_ip	VARCHAR(100)

Figure 5.3: AWS Schema

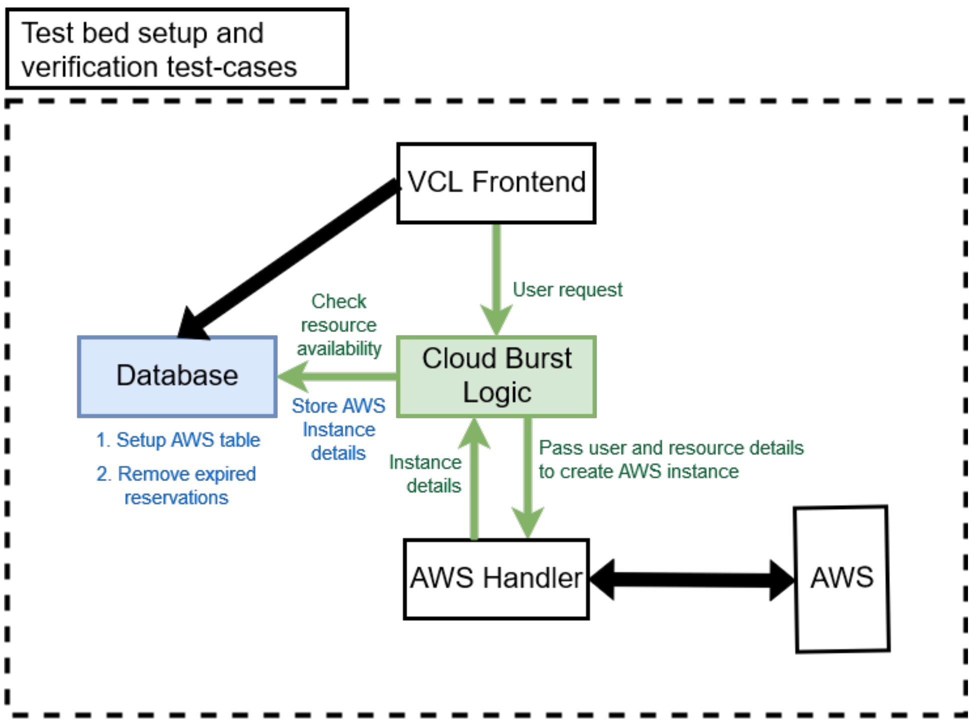
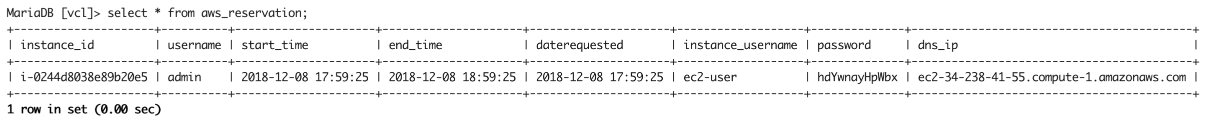

Test bed setup and verification test-cases
VCL Frontend
User request
Check resource availability
Database
Cloud Burst Logic
Store AWS Instance details
1. Setup AWS table
2. Remove expired reservations
Instance details
Pass user and resource details to create AWS instance
AWS Handler
AWS

Figure 5.4: Database Module

5.2.1 Adding an AWS instance Entry

The cloud burst logic gathers the data required for an AWS instance from the user request and through the AWS EC2 APIs and updates the aws_reservation table in the VCL Database.

The below figure shows the VCL Database entry for AWS EC2 Instance created after bursting to public cloud.



```
MariaDB [vcl]> select * from aws_reservation;
+---------------------+----------+---------------------+---------------------+---------------------+-------------------+--------------+------------------------------------------+
| instance_id         | username | start_time          | end_time            | daterequested       | instance_username | password     | dns_ip                                   |
+---------------------+----------+---------------------+---------------------+---------------------+-------------------+--------------+------------------------------------------+
| i-0244d8038e89b20e5 | admin    | 2018-12-08 17:59:25 | 2018-12-08 18:59:25 | 2018-12-08 17:59:25 | ec2-user          | hdYwnayHpWbx | ec2-34-238-41-55.compute-1.amazonaws.com |
+---------------------+----------+---------------------+---------------------+---------------------+-------------------+--------------+------------------------------------------+
1 row in set (0.00 sec)
```

Figure 5.5: AWS reservation table

5.2.2 AWS Entry Deletion cron

There are two ways a user's reservation can be ended. A user can prompt to delete an reservation and the reservation end time might be reached.

While the 1st option is handled by the front-end logic, we have to handle the reservation expiry, which is being done through a cron job that runs every 15 mins, similar to how VCL does it for VCL reservations.

This cron job checks the aws_reservation table for any entry whose end time is past the current time. If any such instance is found, it calls the aws_delete.py script which terminates the instance on EC2 and deletes the entry from the database.

Below snippet shows the crontab:

```
[root@dhcp-152-14-112-53 ] crontab -l
15 * * * * /usr/bin/python /var/www/cgi-bin/aws_delete_cron.py
```

5.3 AWS HANDLER MODULE

AWS Handler Module is the link connecting Cloud Burst Logic to the EC2 web service of AWS. As shown in figure 5.6, this module is responsible for three main tasks:

1. Configuring an AWS account for VCL root, enabling it to create remote VMs on EC2 as required.

2. Spawn new VMs every time a Create request is received from the Cloud Burst Logic.

3. Delete existing VMs every time a Delete request is received from the Cloud Burst Logic.

These tasks were implemented using Python scripts using boto3. Boto3 is a Python package that enable us to communicate through AWS Python APIs. Following sections go into further details of the implementation of these scripts.

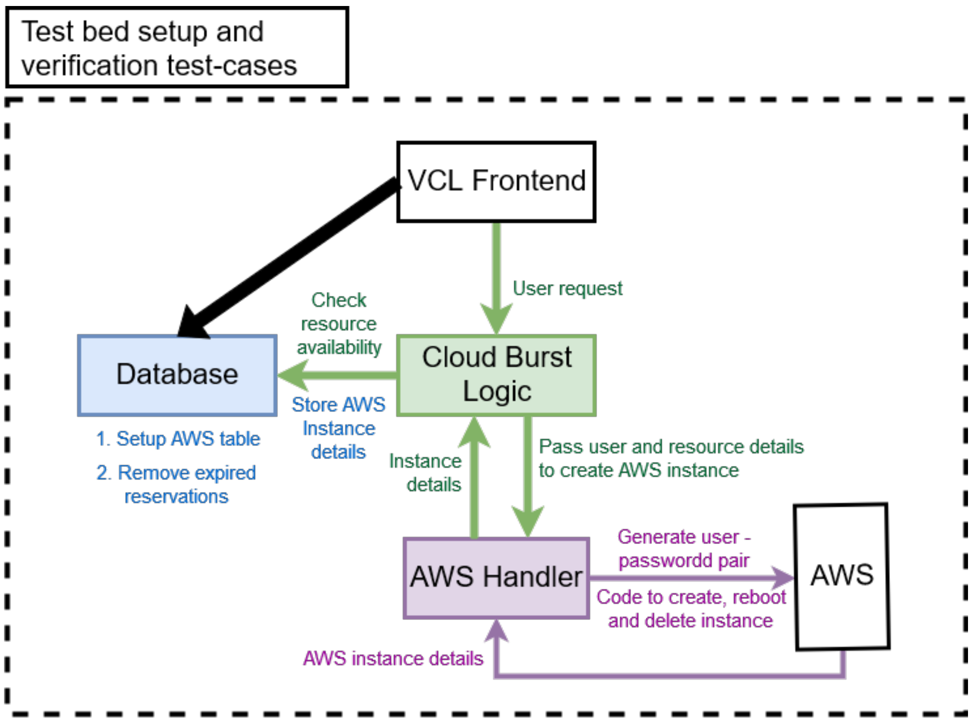

Test bed setup and verification test-cases
VCL Frontend
User request
Check resource availability
Database
Cloud Burst Logic
Store AWS Instance details
1. Setup AWS table
2. Remove expired reservations
Instance details
Pass user and resource details to create AWS instance
Generate user - passwordd pair
AWS Handler
AWS
Code to create, reboot and delete instance
AWS instance details

Figure 5.6: AWS Handler Module

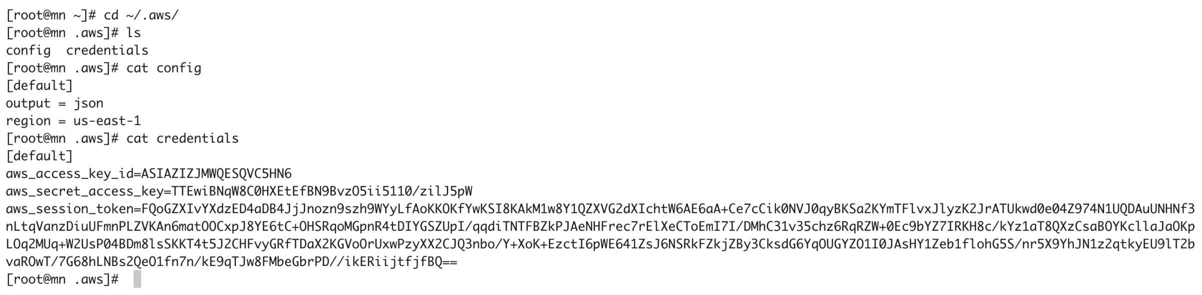
5.3.1 Spawning an AWS VM

This process can be delineated into following steps:

1. Initiating a session through boto3 API calls which requires us to provide certain configuration settings that are associated with our AWS account.

   (a) One of the way to do that is to use ~/.aws folder to store the credential and configuration details. This information is stored in two files.
   ~/.aws/credentials: Stores credential information like AWS access key-id, AWS secret access key, and AWS session token.
   ~/.aws/config: Stores the format of your AWS output and which default region should be selected for your ec2 instances.

   (b) The second way to provide configuration settings is to provide the credential details in form of JSON. We can read these values on python run time from a JSON file and pass it to the call "boto3.resource" while creating an EC2 object. Reading values at run time makes this approach very effective. Since these values are changed very frequently, it will be easy for us to update it in the JSON formatted



```
[root@mn ~]# cd ~/.aws/
[root@mn .aws]# ls
config  credentials
[root@mn .aws]# cat config
[default]
output = json
region = us-east-1
[root@mn .aws]# cat credentials
[default]
aws_access_key_id=ASIAZIZJMWQESQVC5HN6
aws_secret_access_key=TTEwiBNqW8C0HXEtEfBN9BvzO5ii5110/zilJ5pW
aws_session_token=FQoGZXIvYXdzED4aDB4JjJnozn9szh9WYyLfAoKKOKfYwKSI8KAkM1w8Y1QZXVG2dXIchtW6AE6aA+Ce7cCik0NVJ0qyBKSa2KYmTFlvxJlyzK2JrATUkwd0e04Z974N1UQDAuUNHNf3
nLtqVanzDiuUFmnPLZVKAn6matOOCxpJ8YE6tC+OHSRqoMGpnR4tDIYGSZUpI/qqdiTNTFBZkPJAeNHFrec7rElXeCToEmI7I/DMhC31v35chz6RqRZW+0Ec9bYZ7IRKH8c/kYz1aT8QXzCsaBOYKcllaJaOKp
LOq2MUq+W2UsP04BDm8lsSKKT4t5J2CHFvyGRfTDaX2KGVoOrUxwPzyXX2CJQ3nbo/Y+XoK+EzctI6pWE641ZsJ6NSRkFZkjZBy3CksdG6YqOUGYZO1I0JAsHY1Zeb1flohG5S/nr5X9YhJN1z2qtkyEU9lT2b
vaROwT/7G68hLNBs2QeO1fn7n/kE9qTJw8FMbeGbrPD//ikERiijtfjfBQ==
[root@mn .aws]#
```

Figure 5.7: ~/.aws Folder and its Content

file.

These AWS credentials can be obtained from the AWS account dashboard in account details section.

These values change periodically every 3 hours. Currently we are updating them manually in our files.

The format of the json object is as follows:

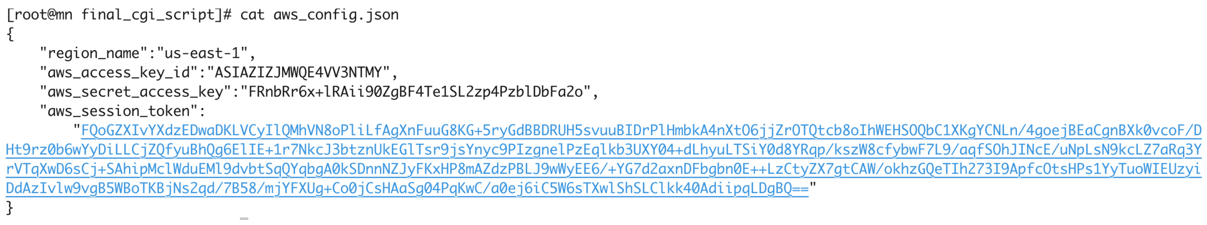
aws_access_key_id : The AWS access key.

aws_secret_access_key : The AWS secret key.

aws_session_token : The AWS session token. A session token is only required if you are using temporary security credentials.

region : The default AWS region to send requests to from this profile.

output : The default output format for this profile.

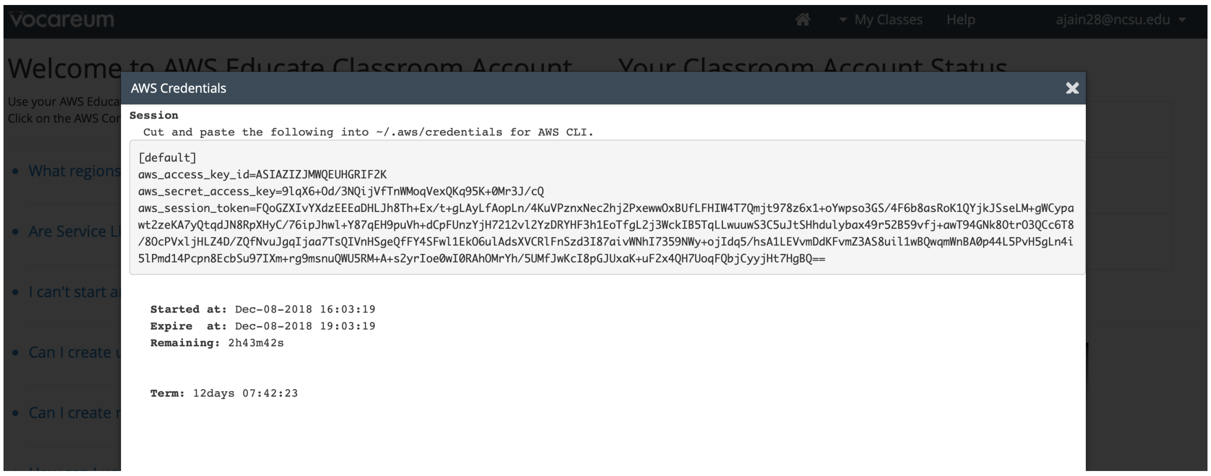


```
[root@mn final_cgi_script]# cat aws_config.json
{
    "region_name":"us-east-1",
    "aws_access_key_id":"ASIAZIZJMWQE4VV3NTMY",
    "aws_secret_access_key":"FRnbRr6x+lRAii90ZgBF4Te1SL2zp4PzblDbFa2o",
    "aws_session_token":
        "FQoGZXIvYXdzEDwaDKLVCyIlQMhVN8oPliLfAgXnFuuG8KG+5ryGdBBDRUH5svuuBIDrPlHmbkA4nXtO6jjZrOTQtcb8oIhWEHSOQbC1XKgYCNLn/4goejBEaCgnBXk0vcoF/D
Ht9rz0b6wYyDiLLCjZQfyuBhQg6ElIE+1r7NkcJ3btznUkEGlTsr9jsYnyc9PIzgnelPzEqlkb3UXY04+dLhyuLTSiY0d8YRqp/kszW8cfybwF7L9/aqfSOhJINcE/uNpLsN9kcLZ7aRq3Y
rVTqXwD6sCj+SAhipMclWduEMl9dvbtSqQYqbgA0kSDnnNZJyFKxHP8mAZdzPBLJ9wWyEE6/+YG7d2axnDFbgbn0E++LzCtyZX7gtCAW/okhzGQeTIh273I9ApfcOtsHPs1YyTuoWIEUzyi
DdAzIvlw9vgB5WBoTKBjNs2qd/7B58/mjYFXUg+Co0jCsHAaSg04PqKwC/a0ej6iC5W6sTXwlShSLClkk40AdiipqLDgBQ=="
}
```

Figure 5.8: Sample JSON File to Store AWS Credentials

Our project implementation employs second method of providing configurations. The main reason we went for this method is because accessing files in the root folder requires root access. Since we've automated this process, the scripts are called through the PHP code which does not have root access.

2. Once we have setup our AWS credentials and account details, we can go ahead and create an instance on EC2. This can be done by using ec2.create_instances() python API



```
Session
  Cut and paste the following into ~/.aws/credentials for AWS CLI.

[default]
aws_access_key_id=ASIAZIZJMWQEUHGRIF2K
aws_secret_access_key=9lqX6+0d/3NQijVfTnWMoqVexQKq95K+0Mr3J/cQ
aws_session_token=FQoGZXIvYXdzEEEaDHLJh8Th+Ex/t+gLAyLfAopLn/4KuVPznxNec2hj2PxewwOxBUfLFHIW4T7Qmjt978z6x1+oYwpso3GS/4F6b8asRoK1QYjkJSseLM+gWCypa
wt2zeKA7yQtqdJN8RpXHyC/76ipJhwl+Y87qEH9puVh+dCpFUnzYjH7212vl2YzDRYHF3h1EoTfgL2j3WckIB5TqLLwuuwS3C5uJtSHhdulybax49r52B59vfj+awT94GNk8Otr03QCc6T8
/8OcPVxljHLZ4D/ZQfNvuJgqIjaa7TsQIVnHSgeQfFY4SFwl1EkO6ulAdsXVCRlFnSzd3I87aivWNhI7359NWy+ojIdq5/hsA1LEVvmDdKFvmZ3AS8uil1wBQwqmWnBA0p44L5PvH5gLn4i
5lPmd14Pcpn8EcbSu97IXm+rg9msnuQWU5RM+A+s2yrIoe0wI0RAhOMrYh/5UMfJwKcI8pGJUxaK+uF2x4QH7UoqFQbjCyyjHt7HgBQ==

  Started at: Dec-08-2018 16:03:19
  Expire  at: Dec-08-2018 19:03:19
  Remaining: 2h43m42s

  Term: 12days 07:42:23
```

Figure 5.9: Credential Details on AWS Dashboard

Call. This API takes different arguments like image ID, minimum number of instances, maximum number of instances, instance type and login key-pair for the VM.

3. Generating login key-pair is an important step to keep in mind when we create the EC2 instance. Amazon EC2 uses public key cryptography to encrypt and decrypt login information. Public key cryptography uses a public key to encrypt a piece of data like password. The recipient uses the private key to decrypt the data. The public and private keys are known as a key pair. To login to your instance, you must create a key pair, specify the name of the key pair when you launch the instance, and provide the private key when you connect to the instance. This key pair is generated using python API calls. We can use ec2.create_key_pair() API call which return us the Key pair which can be stored in a file.

4. Once the instance is created we can login to the device using public key. But we can't pass this PEM key to the user since it is common between all EC2 instances created through VCL. We can handle this problem in two ways.

   (a) Create individual PEM key for each user.

   (b) Generate user password pair for each user.

   The project statement states creating user-password for each user. To create this we have generated a 12 character long random password which consist of upper and lower case character. Once the password is generated this password can be set for the VM default user. Password based login is enabled by editing system file /etc/ssh/sshd_config.

The process of updating these details is automated and is followed by restarting the ssh service.

A better and a scalable approach would be to map the pem key to each user and keep a copy of each user's pem key in each Management node. For the scope of the project we are going ahead with the 1st approach.

5. Once the VM is ready, we store the credentials, public DNS and other relevant details in the VCL database through the Cloud Burst Logic Module so that it is persistent and accessible to the user through front-end.

5.3.2 Deleting an AWS VM

Whenever we receive a delete request from the Cloud Burst Logic, we delete the instance on AWS through a Python script. The background crontab job responsible for expired VMs also uses this script to delete instances on AWS. This process can be delineated in following steps:

1. Receive the instance ID of the instance to be deleted.
2. Terminate the reservation from AWS using instance.terminate() API call.
3. Pass the return code to the Cloud Burst Logic.

This process can be polished a little by first checking if the instance for the given instance-id exists on the AWS account before making the terminate API call.

5.4 VCL FRONT-END MODULE

VCL Front-end Module is responsible to incorporate the new changes at back-end and make it accessible to the user. It involves two main responsibilities:

1. Redirect reservation requests related to AWS instances to Cloud Burst Logic instead of the old VCL back-end modules.
2. List AWS instances on the VCL user dashboard.

Figure 5.10 shows its role in the overall design and its interaction with other modules.

It involved modifying the existing VCL frontend PHP and Javascript code and writing additional PHP files.

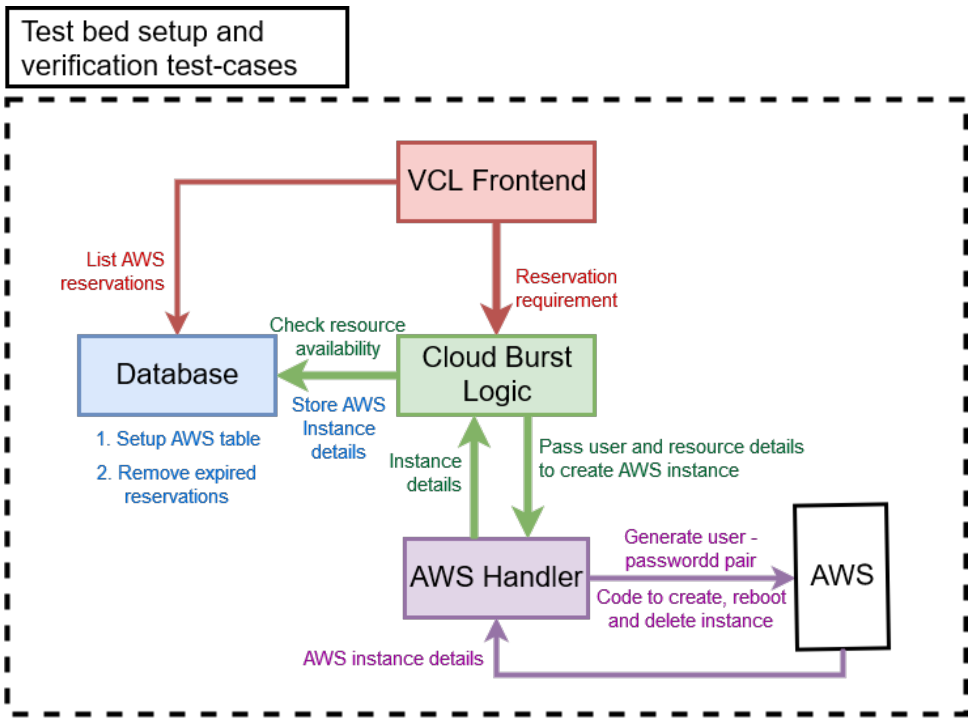

Test bed setup and verification test-cases
VCL Frontend
List AWS reservations
Reservation requirement
Check resource availability
Database
Cloud Burst Logic
Store AWS Instance details
1. Setup AWS table
2. Remove expired reservations
Instance details
Pass user and resource details to create AWS instance
Generate user - passwordd pair
AWS Handler
AWS
Code to create, reboot and delete instance
AWS instance details

Figure 5.10: VCL Frontend Module

5.4.1 Redirecting AWS Create Reservation Requests

1. Every time user submits a request to create a new VM, the Cloud Burst Logic uses the existing condition check on available VCL resources to determine if VCL has sufficient resources to cater to the current request. It sets the condition for creating AWS instance true whenever VCL is out of resources.

2. Whenever this condition is set, the front-end makes an AJAX POST call to Cloud Burst Logic's CGI script which is then responsible for letting AWS-Handler know about the create request on AWS.

3. It passes the user-name, start time and duration of the reservation to this script.

4. The current back-end only handles requests with start time as current time. As soon as they start handling requests for future times, front-end can pass them start-time and end-time to handle those requests.

5.4.2 Redirecting AWS Delete Reservation Requests

1. Whenever user clicks Delete Reservation for an existing AWS reservation from the list page on dashboard, the front-end makes an Ajax call to Cloud Burst Logic's CGI script which lets AWS handler know about the delete request and updates database on success.

5.4.3 Listing AWS Reservations on the Dashboard

The AWS instances of a user are listed in the dashboard in a manner similar to the VCL reservations list but on a separate page. This is implemented using PHP and therefore can be incorporated in the existing PHP code to view VCL reservations. The sole reason for listing them on a separate page is unfamiliarity of the team with PHP which lead to troubles in debugging issues while working with it.

1. The user is provided a link to open their AWS reservation page from VCL reservations page.
2. Whenever user opens the page, the underlying code dynamically queries the AWS table in the database with the user's user-id and populates the table with instance's public dns, start time, end time, and password to login to that isntance.

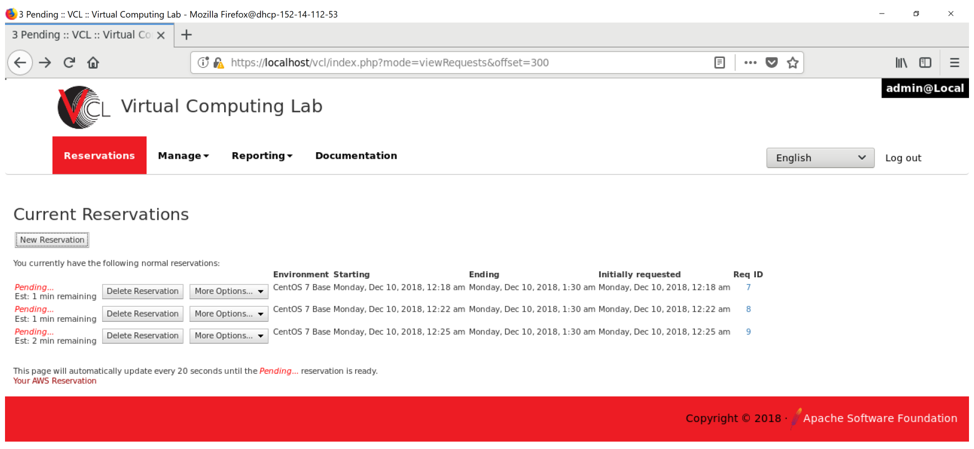
Verification and Validation

We have created our test cases keeping in mind the functional and non-functional requirements and have mapped each test case to the requirement that it fulfilled.

Test no.	Description	Expected Result	Actual Result	Requirement
T 1.1	Create 3 VCL Instance reservations, then create another reservation request	New VM instance should get created on AWS EC2	New VM instance is created on AWS EC2	FR1
T 1.2	Check if the reservation details are visible from VCL web-portal	AWS instance login details to be visible on VCL Web Portal	AWS instance login details are visible on VCL Web Portal	FR1
T 1.3	Free one VCL VM and then try to create a reservation	The next instance should be spawned on VCL	The next instance is spawned on VCL	NFR4

Table 6.1: Test case 1

The 1st image shows that we have 3 VCL reservations created/pending which is the max computers available on VCL lab test bed setup.



3 Pending :: VCL :: Virtual Computing Lab - Mozilla Firefox@dhcp-152-14-112-53
3 Pending :: VCL :: Virtual Co
https://localhost/vcl/index.php?mode=viewRequests&offset=300
admin@Local
Virtual Computing Lab
Reservations
Manage
Reporting
Documentation
English
Log out
New Reservation
Delete Reservation
More Options...
Copyright © 2018 · Apache Software Foundation

Current Reservations

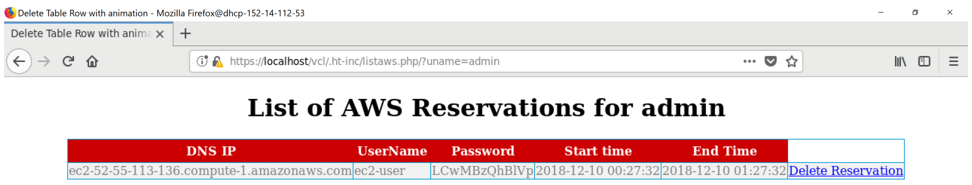
You currently have the following normal reservations:

			Environment	Starting	Ending	Initially requested	Req ID
Pending... Est: 1 min remaining	Delete Reservation	More Options...	CentOS 7 Base	Monday, Dec 10, 2018, 12:18 am	Monday, Dec 10, 2018, 1:30 am	Monday, Dec 10, 2018, 12:18 am	7
Pending... Est: 1 min remaining	Delete Reservation	More Options...	CentOS 7 Base	Monday, Dec 10, 2018, 12:22 am	Monday, Dec 10, 2018, 1:30 am	Monday, Dec 10, 2018, 12:22 am	8
Pending... Est: 2 min remaining	Delete Reservation	More Options...	CentOS 7 Base	Monday, Dec 10, 2018, 12:25 am	Monday, Dec 10, 2018, 1:30 am	Monday, Dec 10, 2018, 12:25 am	9

This page will automatically update every 20 seconds until the *Pending...* reservation is ready.
Your AWS Reservation

Figure 6.1: List VCL Reservations

The 2nd image shows that the next reservation is bursted to public cloud i.e. AWS EC2 and can be seen on the next Web page of VCL.


Delete Table Row with animation - Mozilla Firefox@dhcp-152-14-112-53
Delete Table Row with anim
https://localhost/vcl/.ht-inc/listaws.php/?uname=admin

List of AWS Reservations for admin

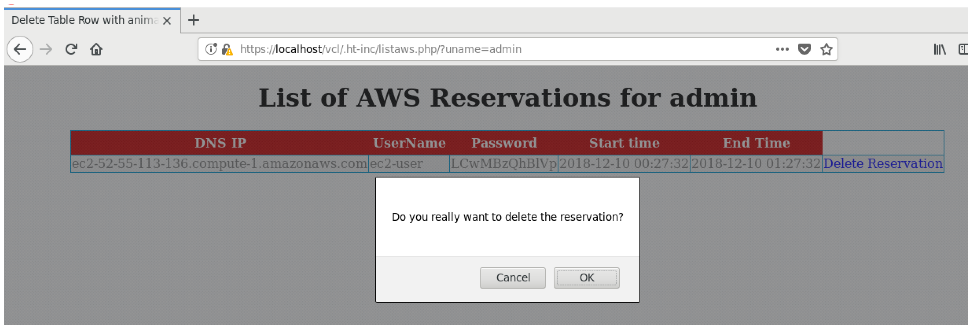
DNS IP	UserName	Password	Start time	End Time	
ec2-52-55-113-136.compute-1.amazonaws.com	ec2-user	LCwMBzQhBlVp	2018-12-10 00:27:32	2018-12-10 01:27:32	Delete Reservation

Figure 6.2: List AWS Reservations

Test no.	Description	Expected Result	Actual Result	Requirement
T 2.1	On the VCL AWS reservations page, click on Delete Reservation Tab	The AWS instance should get deleted and removed from Database	The AWS instance is deleted and entry is cleared from database	FR2
T 2.2	Wait for an AWS reservation to expire and check if the running cron job deletes it	The AWS instance should get deleted and removed from Database	The AWS instance is deleted. This cannot be validated through screen shot. To be shown in Demo	FR2

Table 6.2: Test case 2

The below image shows the prompt to confirm , when an AWS instance deletion is initiated by the user. The next image shows that after the instance is deleted, the user admin does not have any AWS reservations.

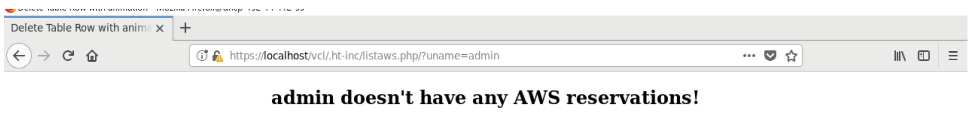

Delete Table Row with anim
https://localhost/vcl/.ht-inc/listaws.php/?uname=admin
Cancel
OK

List of AWS Reservations for admin

DNS IP	UserName	Password	Start time	End Time	
ec2-52-55-113-136.compute-1.amazonaws.com	ec2-user	LCwMBzQhBlVp	2018-12-10 00:27:32	2018-12-10 01:27:32	Delete Reservation

Do you really want to delete the reservation?

Figure 6.3: Delete Prompt



Delete Table Row with anim

https://localhost/vcl/.ht-inc/listaws.php/?uname=admin

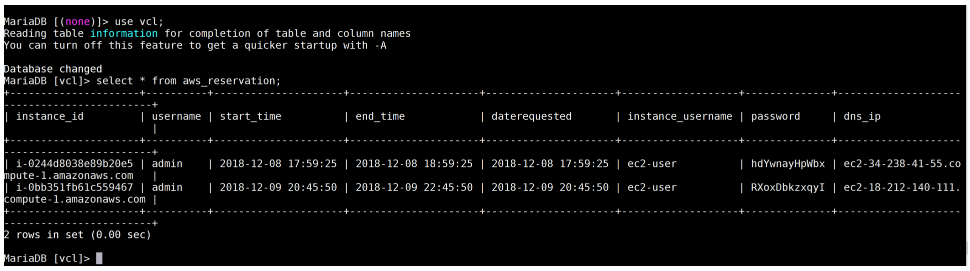
admin doesn't have any AWS reservations!

Figure 6.4: AWS list upon deletion

Test no.	Description	Expected Result	Actual Result	Requirement
T 3.1	Check VCL Database for aws_reservation table when an EC2 instance is created	VCL database has an entry created for the instance	VCL database has an entry created for the instance	FR3
T 3.2	Check VCL Database for aws_reservation table when an EC2 instance is deleted	VCL database entry should be deleted for the instance	VCL database entry is deleted for the instance	FR3

Table 6.3: Test case 3

The 1st image shows the aws_reservation table in VCL database with the AWS instance details. The 2nd image shows the empty aws_reservation table post AWS instance deletions.

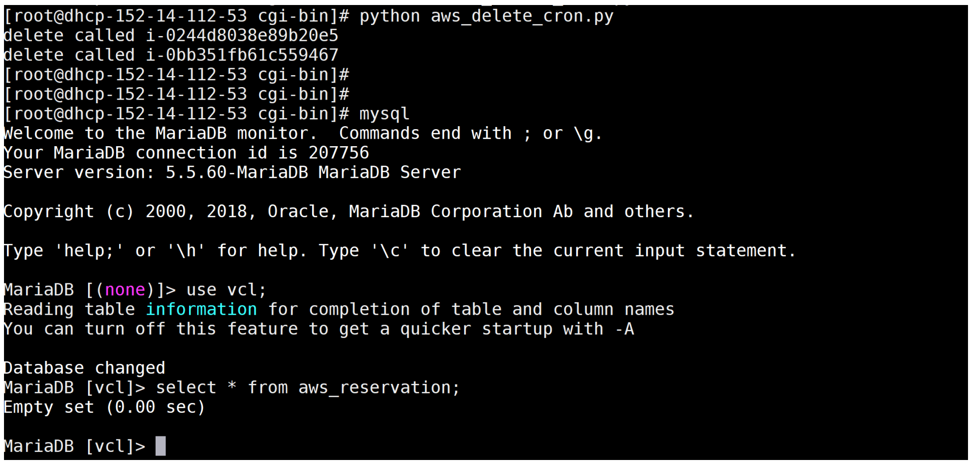


```
MariaDB [(none)]> use vcl;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [vcl]> select * from aws_reservation;
+---------------------+----------+---------------------+---------------------+---------------------+-------------------+--------------+-------------------
------------------------+
| instance_id         | username | start_time          | end_time            | daterequested       | instance_username | password     | dns_ip
                        |
+---------------------+----------+---------------------+---------------------+---------------------+-------------------+--------------+-------------------
------------------------+
| i-0244d8038e89b20e5 | admin    | 2018-12-08 17:59:25 | 2018-12-08 18:59:25 | 2018-12-08 17:59:25 | ec2-user          | hdYwnayHpWbx | ec2-34-238-41-55.co
mpute-1.amazonaws.com   |
| i-0bb351fb61c559467 | admin    | 2018-12-09 20:45:50 | 2018-12-09 22:45:50 | 2018-12-09 20:45:50 | ec2-user          | RXoxDbkzxqyI | ec2-18-212-140-111.
compute-1.amazonaws.com |
+---------------------+----------+---------------------+---------------------+---------------------+-------------------+--------------+-------------------
------------------------+
2 rows in set (0.00 sec)

MariaDB [vcl]>
```

Figure 6.5: AWS Reservation table with reservations



```
[root@dhcp-152-14-112-53 cgi-bin]# python aws_delete_cron.py
delete called i-0244d8038e89b20e5
delete called i-0bb351fb61c559467
[root@dhcp-152-14-112-53 cgi-bin]#
[root@dhcp-152-14-112-53 cgi-bin]#
[root@dhcp-152-14-112-53 cgi-bin]# mysql
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 207756
Server version: 5.5.60-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> use vcl;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [vcl]> select * from aws_reservation;
Empty set (0.00 sec)

MariaDB [vcl]> 
```

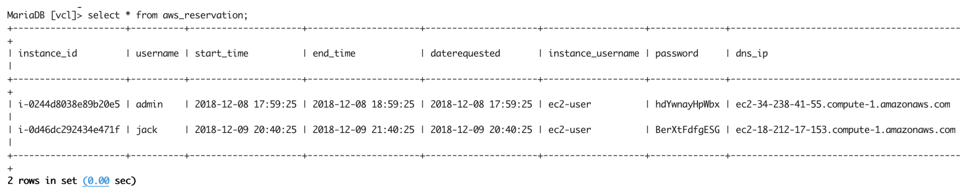
Figure 6.6: Empty AWS Reservation table

Test no.	Description	Expected Result	Actual Result	Requirement
T 4	Performance - Check the time taken to spawn an EC2 instance	The average time for reservation on VCL is 2 mins	The average AWS EC2 instance creation time is 45 secs	NFR 1

Table 6.4: Test case 4

Test no.	Description	Expected Result	Actual Result	Requirement
T 5	Scalability - Multiple users can simultaneously login to VCL portal and request AWS instances	Each user should be able to request AWS EC2 instances	Each user is able to request AWS EC2 instances	NFR 2

Table 6.5: Test case 5

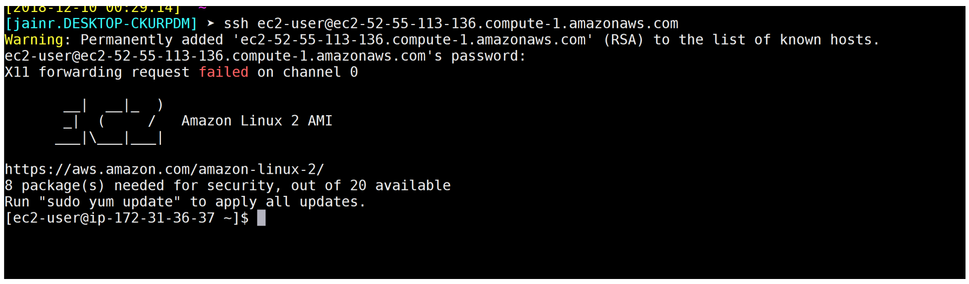


```
MariaDB [vcl]> select * from aws_reservation;
+---------------------+----------+---------------------+---------------------+---------------------+-------------------+--------------+-------------------------------------------
+
| instance_id         | username | start_time          | end_time            | daterequested       | instance_username | password     | dns_ip
|
+---------------------+----------+---------------------+---------------------+---------------------+-------------------+--------------+-------------------------------------------
+
| i-0244d8038e89b20e5 | admin    | 2018-12-08 17:59:25 | 2018-12-08 18:59:25 | 2018-12-08 17:59:25 | ec2-user          | hdYwnayHpWbx | ec2-34-238-41-55.compute-1.amazonaws.com
|
| i-0d46dc292434e471f | jack     | 2018-12-09 20:40:25 | 2018-12-09 21:40:25 | 2018-12-09 20:40:25 | ec2-user          | BerXtFdfgESG | ec2-18-212-17-153.compute-1.amazonaws.com
|
+---------------------+----------+---------------------+---------------------+---------------------+-------------------+--------------+-------------------------------------------
+
2 rows in set (0.00 sec)
```

Figure 6.7: Multiple Users

Test no.	Description	Expected Result	Actual Result	Requirement
T 6	Try to login to the created instance with the shared username/password	User should be able to login to the AWS EC2 instances with the shared username/password pair	User is able to login to the AWS EC2 instances with the shared username/password pair	NFR5
T6	Login to the created instance and try to execute a root privilege command	The user should not be able to execute the command	The user is not able to execute the command	NFR5

Table 6.6: Test case 6



```
[2018-12-10 00:29:14] ~
[jainr.DESKTOP-CKURPDM] ➤ ssh ec2-user@ec2-52-55-113-136.compute-1.amazonaws.com
Warning: Permanently added 'ec2-52-55-113-136.compute-1.amazonaws.com' (RSA) to the list of known hosts.
ec2-user@ec2-52-55-113-136.compute-1.amazonaws.com's password:
X11 forwarding request failed on channel 0

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
8 package(s) needed for security, out of 20 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-36-37 ~]$ 
```

Figure 6.8: Successful Login attempt

Schedule and Personnel

Schedule					
No	Tasks	Owner	Contributors	Start Date	End Date
1.	Discussion of problem Statement	Team		11/07/2018	11/08/2018
2.	Understanding the VCL Architecture	Team		11/08/2018	11/09/2018
3.	Understanding the AWS Architecture	Team		11/09/2018	11/10/2018
4.	Requirement Gathering	Team		11/10/2018	11/11/2018
5.	Cloud Burst Workflow and Design	Team		11/11/2018	11/13/2018
6.	Perl and PHP primer	Team		11/13/2018	11/15/2018
7.	VCL front-end Code review	Abhash	Rishabh	11/15/2018	11/23/2018
8.	VCL back-end Code review	Henil		11/15/2018	11/23/2018
9.	AWS API testing	Srijani	Sriram	11/15/2018	11/23/2018
10.	Cloud burst logic Development	Abhash	Rishabh	11/25/2018	12/01/2018
11.	VCL front-end Development	Henil	Abhash	11/27/2018	12/03/2018
12.	AWS scripts Development	Srijani	Abhash	11/28/2018	12/02/2018
13.	Net-labs test bed preparation	Rishabh	Sriram	12/04/2018	12/07/2018
14.	VCL and Cloud Burst logic Integration	Abhash,Henil	Rishabh	12/07/2018	12/09/2018
15.	Verification and Validation	Team		12/09/2018	In Progress
16.	Project Documentation	Team		12/08/2018	12/09/2018

Table 7.1: Schedule

Results

After verification and validation, we can conclude that:

- As long as the private cloud i.e. VCL has enough compute resources to handle user requests, the cloud burst logic doesn't interfere with the functioning of VCL.
- When there is resource scarcity on VCL, the user's request for a new reservation is fulfilled by bursting to the public cloud i.e. AWS.
- The AWS instance details are stored consistently in the VCl database.
- The proposed design and implementation complies with all the functional requirements and most of the non-functional requirements

References

ReQtest, 5 Apr. 2012, *Functional vs Non Functional Requirements* Available at: <https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements> [Accessed 4 Dec 2018]

Linux Academy, 26 Jan. 2017, *Automating AWS With Python and Boto3* Available at: <https://linuxacademy.com/howtoguides/posts/show/topic/14209-automating-aws-with-\\python-and-boto3> []Accessed 21 Nov 2018]

boto v2.49.0, *An Introduction to boto's EC2 interface* Available at: <http://boto.cloudhackers.com/en/latest/ec2_tut.html> [Accessed 21 Nov. 2018]

19 Jul. 2018, *Enable Password Login for Connecting to EC2 ... - AWS - Amazon.com* Available at: <https://aws.amazon.com/premiumsupport/knowledge-center/ec2-password-login/> [Accessed 23 Nov. 2018]

Installing the AWS Command Line Interface - AWS Documentation. Available at: <https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html> [Accessed 21 Nov. 2018]

Architecture - Apache VCL - The Apache Software Foundation! Available at: <https://vcl.apache.org/info/architecture.html> [Accessed 8 Nov. 2018]

23 Apr. 2015, *Apache VCL - Apache Software Foundation* <https://cwiki.apache.org/confluence/display/VCL/Apache+VCL> [Accessed 8 Nov. 2018]

Appendices

10.1 BOTO3 APPENDIX

Boto is the AWS SDK for Python which helps us in using its object-oriented APIs to create, list and terminate EC2 instances on AWS.

- Installing latest boto3 release: pip install boto3
- The session details which includes the region name, aws_access_key_id, aws_secret_access_key and aws_session_token are passed as parameters to the session call of boto3 and the resource in our case is going to be ec2.
- create_instance( ): For creating an instance, we pass the parameters ImageId, MinCount, MaxCount, InstanceType and KeyName to the create_instance( ) function of the ec2 resource
- stop( ) and terminate( ): For deleting the instances, we pass the instance ID to the calls from the ec2.instances to these functions

10.2 MYSQL-CONNECTOR APPENDIX

mysql-connector is a python package to connect with MariaDB MySQL database.

- Installing mysql-connector: pip install mysql-connector
- Connect to mysql DB, arguments are host name, user name ,password and DB name: mysql.connector.connect()
- Execute mysql query: mycursor.execute(sqlquery)
- Commit the changes to db: mydb.commit()

10.3 CGI SCRIPTS

CGI scripts are dynamic pages that run on server. We have used them to execute the tasks of creating and deleting AWS instances. In apache server, the the default configured directory

for CGI scripts is /var/www/html/cgi-bin.
We placed our cgi scripts in this directory. Our front-end Javascript program functions would perform POST operation on these pages to execute said operations.

10.4 SOURCE CODE GITHUB REFERENCE

The code for this project can be accessed from the below link using a NCSU GitHub Account.
`https://github.ncsu.edu/engr-csc-547/2018FallCloudBursting`