Virtual Machine Reservation System

Requirements:

Design a Virtual Machine Reservation System that will be used to manage virtual machines present in cloud. Users will be checking-out the virtual machines to use and then check-in them after they use them. The Administrator should be able to create and provision the virtual machines upto a set limit.

Use Cases:

- When a user / client calls the checkout method you provide, they should get the IP address of the VM that they can use. The Application should return the IP address of a VM that can be used.
- When they are done using it, the user will call the checkin method, with the IP address of the VM, to check it back in. The Application should accept that VM back.
- If all currently created VMs are in use, the administrator has the ability to create more VMs using the create_vm method. Which is limited by a finite number.
- If a client requests a VM and no VM is available to be checked out, then the application needs to give proper message so that the users can retry after some time.
- Multiple client processes/users can request VMs simultaneously and the application should be able to provide unique VMs to them.
- The same VM cannot be checked out by two clients at the same time.

Assumptions:

VMs can be simulated.
Need not care about the notion of the users for now.

Design Overview:

We have defined the Class VirtualMachineManager which contains the methods to manage the Virtual Machines.

Parameters:
max_vm_count: The count of the maximum number of the virtual machines that can be provisioned.

connection: The connection object to the database to be used. Where the information related to the Virtual Machines is stored.

Instance Variables:
_max_vm_count: The count of the maximum number of the virtual machines that can be provisioned.(passed as parameter)

_checkin_thread: Thread that takes care of checking in the vms.
_checkout_thread: Thread that takes care of checking in the vms.
_create_thread: Thread that takes care of checking in the vms.
_delete_thread: Thread that takes care of checking in the vms.
conn: Connection to the database that would be used (passed in as parameter)
cursor: Cursor of the database connection.
_vm_count: Count of the Virtual Machines that are created.

Class Methods:

generate_ip *() :*
Parameters: None
Return: ip (string)
Description: This method generates a random IPV4 and returns it as a string. Ips starting with 10, 172 and 192 are excluded as they are used for private networks.

create_vm*():*
Parameters: None
Return: json
Description: This method first checks the *_vm_count* to check if the maximum number of vms have been created. If yes, it returns a Json object with the error message "**Error: Maximum VM Count reached**".
If Maximum vm count is not reached, the function uses uuid library to generate a unique *vm_id* and *generate_ip*() method to create an ip_address. Before assigning the ip to the vm, it checks the used ips in the database table to make sure that same ip is not assigned to two vms. Then the database table is updated with the vm details. The vm created would be either in 'available' or in 'error' state. A Json response containing the vm details is returned. Thread locking is used to make sure no two processes create a vm at the same time which might end up in wrong count for the *_vm_count* and also might result is two vms having same ip address.

delete_vm*():*
Parameters:

- vm_id : unique id of the vm
- ip : ip address of the vm.

Return: json
Description: This method first checks if a vm exists with the given vm_id and the ip. Then deletes the vm only if it is in 'available' or 'error' state. An appropriate json response is returned. Thread locking is used to make sure no two processes access the same vm (database entry) at the same time.

get_vm*():*
Parameters:

- vm_id : unique id of the vm

Return: json

Description: This method first checks if a vm exists with the given vm_id and the ip. If not, a json response with the message '**Could not find a vm with given details'** is returned. If found, it returns an appropriate json response with the vm information (vm_id, ip, status)

gete_vm_status*():*
Parameters:

- vm_id : unique id of the vm

Return: json
Description: This method first checks if a vm exists with the given vm_id. If not, a json response with the message '**Could not find a vm with given details'** is returned. If found, it returns a json response with the vm status.

checkout_vm*():*
Parameters: None
Return: json
Description: This method checks if there are any vms that are in the 'available' state. If not, a json response with the message '**No VMs currently available, please try after some time'** is returned. If there is a vm in 'available' state, the status of the vm in the database table is updated to 'checked-out' and a json response with the vm info is returned. Thread locking is used to make sure no two processes access the same vm (database entry) at the same time.

checkin_vm*():*
Parameters:

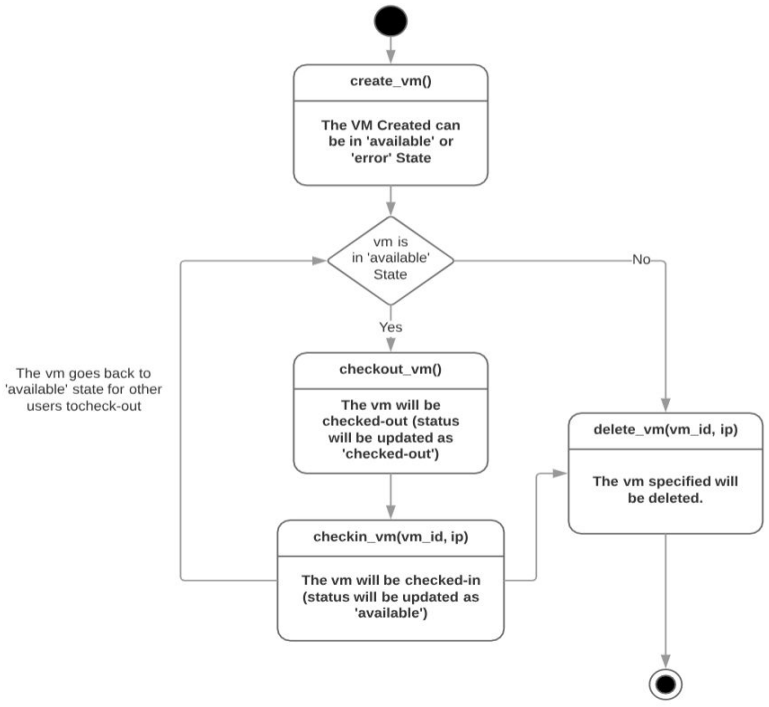
- vm_id : unique id of the vm
- ip : ip address of the vm.

Return: json
Description: This method first checks if a vm exists with the given vm_id and the ip. If not, a json response with the message '**Could not find a vm with given details'** is returned. If a vm is found, then the method checks if the vm_status is 'checked-out' and updates the vm_status as 'available'. Thread locking is used to make sure no two processes access the same vm (database entry) at the same time.

The json response returned by the Class methods is of the following format:
return_json = {**'status'**: This is the status of the request. It would be 'success' or 'error',
'data': This would be the data related to the vm,
'message': This would be the error message as per the issue occured. }

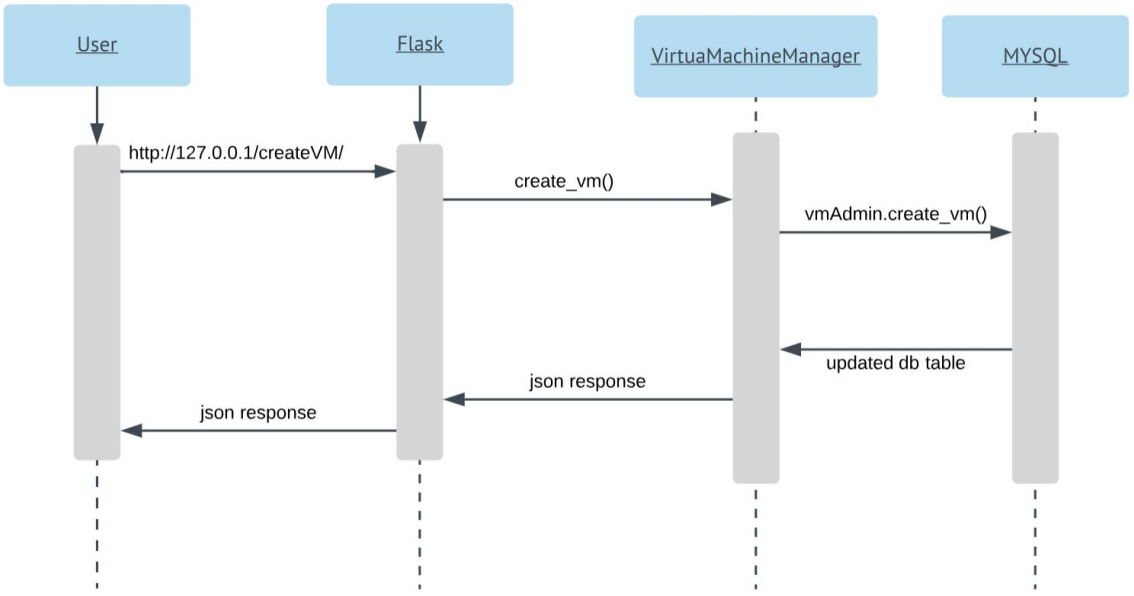
State Diagram:

The following would be a generic state diagram of a vm.



Sequence Diagram:

The following would be a generic sequence diagram of the application.



DataBase Design:

MYSQL has been used to store the information of the virtual machines. The database(*vm_db*) contains one table (*vm_reservations*) which has three columns.

1. *vm_id* : varchar(40) unique not null: This is used to store the unique id of the vm. This is also the primary key.
2. *ip_address*: varchar(30) unique not null: This is used to store the ip address of the vm.
3. *vm_status*: varchar(30) not null: This is used to store the status of the vm.

Installation:

1. Download / clone the code from the GitHub repo found at https://github.com/sushwanth/vm_manager
2. cd into the directory having the DockerFile.
3. If you do not have Docker installed in your linux box, you can install it by using the install.sh script found here: https://github.com/docker/docker-install
4. Use the command -> "$ docker-compose build " to build a docker image. This command would build a docker image by using the information given in the Dockerfile
5. Use the command -> "$ docker-compose up" to start app. This would start the mysql container first followed by the application container as specified in the docker-compose.yml file
6. Now, you can check if the server is running by visiting the url "http://127.0.0.1:5000/" (Alternatively, you can use the curl command as well)

Testing:

The unit tests have been written in the tests.py file. To run them use the following command:

```
$ python tests.py
```

The application and the unit tests have been developed using python3. Please make sure you use python3 to run the unit tests.

Alternatively, you can also test the RESTful API using parallel and curl commands

Example:

```
sushwanth$ seq 5 | parallel -j5 curl  "http://127.0.0.1:5000/createVM/"
sushwanth$ seq 5 | parallel -j5 curl  "http://127.0.0.1:5000/checkoutVM/"
```

Possible Improvements:

- Using cloud API (Example: AWS) to create/delete VMs
- Having the user as well so that the administrator can keep a track of the vms used per user.
- Having a set of static IPs which can be assigned to the VMs upon creation.
- Having a time out for every reservation after which the VM will be automatically checked-in.

- VMs that are not being checked-out for a period of time would be automatically deleted to free up the resources.
- The specifications of each VM can be configured individually.