MiniProject

Aim: Building a Virtualization Orchestration Layer

Introduction - In today's world, Hypervisor is the new OS and Virtual Machines are the new processes. Many system programmers are familiar with the low level APIs that are exposed by the operating systems like Linux and Microsoft Windows. These APIs can be used to take control of the OS programmatically and help in developing management tools. Similar to the OS, Hypervisors expose APIs that can be invoked to manage the virtualized environments. Typical APIs include provisioning, de-provisioning, changing the state of VMs, configuring the running VMs and so on. While it may be easy to deal with one Hypervisor running on a physical server, it is complex to concurrently deal with a set of Hypervisors running across the datacenter. In a dynamic environment, this is a critical requirement to manage the resources and optimize the infrastructure. This is the problem that we try to solve in this project.

Problem Definition – Build a fabric that can coordinate the provisioning of compute resources by negotiating with a set of Hypervisors running across physical servers in the datacenter.

Expected Outcome –

1. Resource Discovery:

2. Resource Allocation

Decide what resources to allocate to fulfil the request. It should be loosely coupled in the sense that mechanism and implementation should be separate. What this means is it should be a possible to change the "Algorithm" for allocation with minimal code changes or still better with change in configuration only.

3. A REST API Server (that can be consumed by a variety of clients to deal with the virtual infrastructure).

Expected API:

VM APIs:

- VM_Creation:
    - Argument: name, instance_type.
    - Return: vmid(+ if successfully created, 0 if failed)
      {
      vmid:38201
      }
    - URL:

    -
    http://server/vm/create?name=test_vm&instance_type=type&image_id=id
- VM_Query
    - Argument: vmid
    - Return: instance_type, name, id, pmid (0 if invalid vmid or otherwise)
      {
      "vmid":38201,
      "name":"test_vm",
      "instance_type":3,
      "pmid": 2
      }
    -
    - URL: http://server/vm/query?vmid=vmid
- VM_Destroy
    - Argument: vmid
    - Return: 1 for success and 0 for failure.
      {
      “status”:1
      }
    - URL: http://server/vm/destroy?vmid=vmid
- VM_Type
    - Argument: NA
    - Return: tid, cpu, ram, disk
      ```
      {
         "types": [
           {
             "tid": 1,
             "cpu": 1,
             "ram": 512,
             "disk": 1
           },
           {
             "tid": 2,
             "cpu": 2,
             "ram": 1024,
             "disk": 2
           },
           {
             "tid": 3,
             "cpu": 4,
             "ram": 2048,
             "disk": 3
      ```

```
            }
        ]
    }
```
- URL: http://server/vm/types

Resource Service APIs:

- List_PMs
  - Argument: NA
  - Return: pmids
    ```
    {
    "pmids": [1,2,3]
    }
    ```
  - URL: http://server/pm/list
- List_VMs
  - Argument: pmid
  - Return: vmids (0 if invalid)
    ```
    {
    "vmids": [38201, 38203, 38205]
    }
    ```
  - URL: http://server/pm/listvms?pmid=id
- PM_Query
  - Argument: pmid
  - Return: pmid, capacity, free, no. of VMs running(0 if invalid pmid or otherwise)
    ```
    {
    "pmid": 1,
    "capacity":{
            "cpu": 4,
            "ram": 4096,
            "disk": 160
            },
    "free":{
            "cpu": 2,
            "ram": 2048,
            "disk": 157
            },
    "vms": 1
    }
    ```
  - URL: http://server/pm/query?pmid=id

Image Service APIs:

- List_Images
  - Argument: NA
  - Return: id, name

```
{
“images”:[
        {
        “id”: 100,
        “name”: “Ubuntu-12.04-amd64”
        },
        {
        “id”:101,
        “name”: “Fedora-17-x86_64”
        }
        ]
}
```

- URL: http://server/image/list

■ For all the other Restful calls return 0(that is the call is invalid)

Installation Script

Every student is required to submit a script for setting up their system. The testing will be done on Ubuntu 12.04/14.04 64 bit. The server will have libvirt, python and java pre-installed. This image will be shared so that you can test your script before submission. If you require any other packages, write the installation commands in the script itself.

Syntax:

```
./script pm_file image_file flavor_file
```

pm_file => Contains a list of IP addresses separated by new-line. These addresses the Physical machines to be used for hosting VMs. A unique ID is to be assigned by you.
image_file => Contains a list of Images(full path) to be used for spawning VMs. The name of the image is to be extracted from the path itself. A unique ID is to be assigned by you.
flavor_file => contains a dictionary of flavors which can be evaluated.
For example the dictionary could contain. For instance in python you can eval this as it is.

```
"types": [
        {
            "cpu": 1,
            "ram": 512,
            "disk": 1
        },
        {
            "cpu": 2,
            "ram": 1024,
            "disk": 2
        },
    ]
```

After running the script, the rest server should be up and running.

Bonus(optional marks)
These are just indicative. Though if you introduced any new feature write it in the doc.
4. One or More Clients:
At least a command line client to demonstrate the functionality that consumes the services. You can build more like android client or browser plugin(i.e. gui which is optional).

5. Persistent storage:
Use something like a sql or nosql database to keep the track all the data associated with your program. This will insure persistent storage for your data pertaining to vm details and so on and hence once you switch off the server and start it again you may be able to start it from that point.

Suggested Steps:

1. Select the right tools by studying various Hypervisor/Cloud controller like CloudStack, OpenStack and Eucalyptus.
2. Design the algorithm to coordinate the provisioning spanning multiple resources.
3. Design the REST interface that abstracts the fabric functionality.
4. Build clients that demonstrate the end-to-end use cases

Evaluation Method:

Almost fully automated. All the APIs will be called and the results will be checked against the actual. We will get the pass/fail report for API calls, so be sure to follow API correctly.

References:

1. EC2 APIs
2. OpenStack APIs and Architecture.
3. http://www.json.org/
4.http://libvirt.org/
5. Similar Open Source Project: http://archipelproject.org/
6. Android Client example: VM Manager
7. Json Validation: http://jsonlint.com/
8. You can use curl to test your server’s request-response.

Resources:

1. http://www.ibm.com/developerworks/linux/library/os-python-kvm-scripting1/index.html
2. http://libvirt.org/bindings.html
3. http://wiki.qemu.org/download/linux-0.2.img.bz2