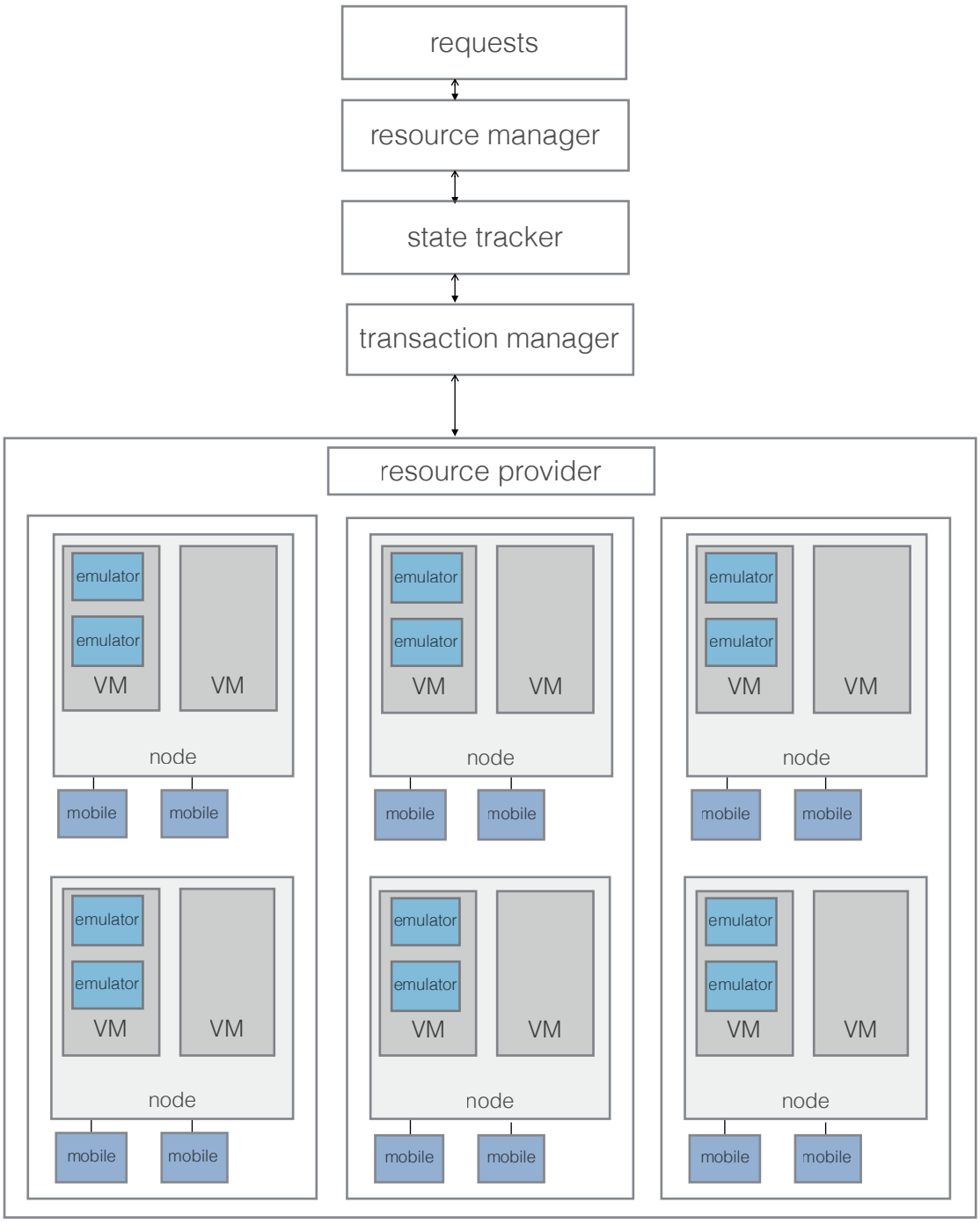
Resource Provision and Management

Resource management engine load requests from users. The resource manager of the engine keeps track of the status of all resources, and will make arrangement for the request. The arrangement is then executed by the transaction manger, to allocate/ deallocate resources. Then the records in state tracker will be updated. The architecture is shown in figure below.

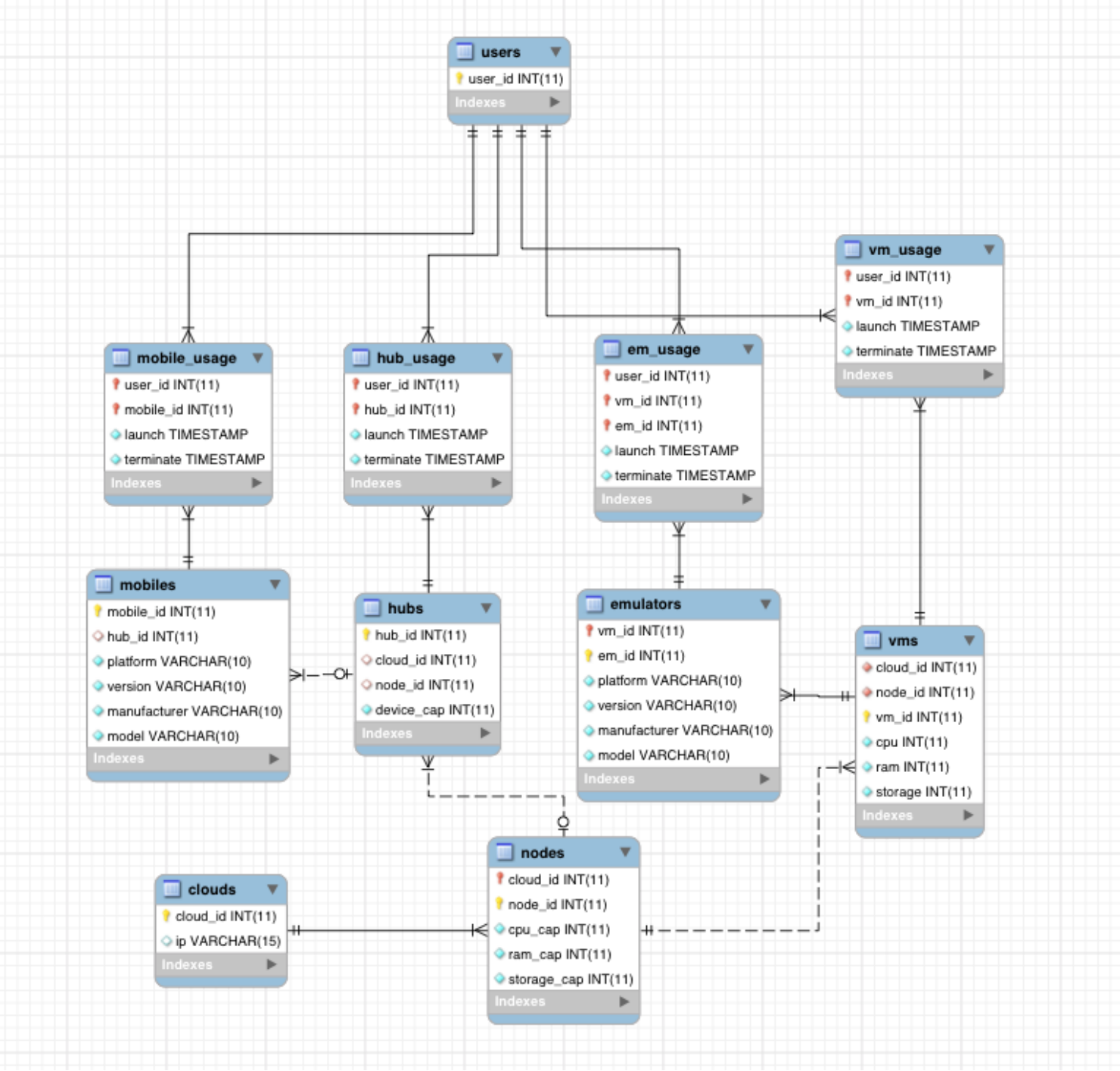


Four types of resources are considered in this system.

- VMs
  - VMs are hosted by computing nodes in clusters
  - Two types of VMs
    - system VMs - host emulators, managed by system
    - user VMs - serve as users' test servers, managed by users themselves
- Emulators
  - hosted in system VMs
  - the host system VM controls the accessibility of emulators to users
- mobiles
  - real mobile devices connected to nodes
  - once connected, not supposed to move frequently
  - managed by the controller program in host node
- hubs
  - real mobile hubs connecting nodes and mobile devices
  - each has a maximum capacity

The states are recorded by a database in the state tracker.
There are basically three layers of relations:

- Infrastructure Layer
  - Cloud - cluster of nodes, e.g Openstack
  - Node - a computing node, can host VMs
- Resource Layer
  - VM
  - Emulator
  - mobile device
  - hub - though devices and hubs are physical resources, kind of infrastructure, we consider them more as resources
- Usage Layer - the user behavior of occupying or releasing resources
  - VM usage
  - Emulator usage
  - mobile usage
  - hub usage
- User Layer
  - record active users, connected to the behavior of users


users
user_id INT(11)
Indexes
vm_usage
user_id INT(11)
vm_id INT(11)
launch TIMESTAMP
terminate TIMESTAMP
Indexes
mobile_usage
user_id INT(11)
mobile_id INT(11)
launch TIMESTAMP
terminate TIMESTAMP
Indexes
hub_usage
user_id INT(11)
hub_id INT(11)
launch TIMESTAMP
terminate TIMESTAMP
Indexes
em_usage
user_id INT(11)
vm_id INT(11)
em_id INT(11)
launch TIMESTAMP
terminate TIMESTAMP
Indexes
mobiles
mobile_id INT(11)
hub_id INT(11)
platform VARCHAR(10)
version VARCHAR(10)
manufacturer VARCHAR(10)
model VARCHAR(10)
Indexes
hubs
hub_id INT(11)
cloud_id INT(11)
node_id INT(11)
device_cap INT(11)
Indexes
emulators
vm_id INT(11)
em_id INT(11)
platform VARCHAR(10)
version VARCHAR(10)
manufacturer VARCHAR(10)
model VARCHAR(10)
Indexes
vms
cloud_id INT(11)
node_id INT(11)
vm_id INT(11)
cpu INT(11)
ram INT(11)
storage INT(11)
Indexes
nodes
cloud_id INT(11)
node_id INT(11)
cpu_cap INT(11)
ram_cap INT(11)
storage_cap INT(11)
Indexes
clouds
cloud_id INT(11)
ip VARCHAR(15)
Indexes

The typical lifecycle of a resource request in our system has these phases:

1. Generation
   vm request should include {os,cpu,ram,storage}
   emulator request should include {platform,version,manufacturer,model}
   mobile device request should include {platform,version,manufacturer,model}
   since hubs are used only to connect mobile devices and computing nodes, the request is automatically generated by default when user requires mobile devices.
2. Submission
   requests are kept in a queue stored in the resource manager
3. Processing
   The allocator will then search the resource states for available nodes.
   Different kinds of algorithms like least connection, simple match or ant colony
4. Allocation
   The chosen node will be allocated the desired resource for user.
   The usage will be recorded in the database, with the time of launching.
5. Deallocation
   The resource will be freed, while the usage in database will not be deleted, but set status to terminated, the time of termination will be recorded.

The python function to start a ubuntu test server virtual machine is like this:

```
def start_ubuntu():
    credentials = get_nova_credentials_v2()
    nova_client = nvclient.Client(**credentials)

    user_keystone = ksclient.Client(auth_url=<auth_url>,\
        username = "demo",password = "stack",tenant_name="demo")
    glance_endpoint = keystone.service_catalog.url_for(service_type="image")
    glance = glclient.Client(glance_endpoint,token = keystone.auth_token)
    imgs = glance.images.list()
    has_ubuntu=False
    for image in imgs:
        print(image.name)
        if name == "ubuntu"
            has_ubuntu=True
    if has_ubuntu ==False
        upload_ubuntu("~/Downloads/trusty-server-cloudimg-amd64-disk1.img")

    print(nova_client.servers.list())
    image = nova_client.images.find(name="ubuntu")
    flavor = nova_client.flavors.find(name="m1.tiny")
    net = nova_client.networks.find(label="private")
    nics = [{'net-id': net.id}]
    instance = nova_client.servers.create(name="vm2", \
        image=image, flavor=flavor, nics=nics)
    print("List of VMs")
    print(nova_client.servers.list())
```

The python function to start an emulator is like this:

```
    home = expanduser("~")
    sudoPassword = <pwd>
```

```
avd_name ='cloud_avd'

target_id='android-8'
abi_name='default/armeabi-v7a'

print home

android_sdk_dir = home+"/android"

os.chdir(android_sdk_dir)

android_sdk_source = android_sdk_dir+"/android-sdk-linux"

command = "echo no l " + android_sdk_source+"/tools/android create avd -n "+avd_-
name+" -t "+target_id
print command
p = os.system(command)

command = android_sdk_source+"/tools/emulator -avd "+avd_name

print command
p = os.system(command)
```

The transaction manager will call these functions to allocate resources, and corresponding de-allocation functions to deallocate them.

The mobile devices and emulators are never moved in our cloud system, they are assigned to different users through virtual hubs, which connect them to the user's test server.
Take Android for instance, the user will send adb commands through the hubs to the emulators.