

Contents

CH1: Introduction:

Traditional Computing Model	7
- One App/Server	7
- Traditional Switching	7
- Drawbacks	8
Cloud Computing Model	8
- Virtual Resources	8
- Advantages	8
- Drawbacks	8
Our Platform	9
- What Kloudak Offers	9
.....	

CH2: Kloudak High-Level Architecture:

Components & Roles	10
- Data Services	10
- Automation Services	10
- Infrastructure	10
Basic Concepts	12
- Areas	12
- Storage Pools	12
- JSON-WEB-TOKEN	12

CH3: Authentication & Authorization:

End-Users Security	13
- Login	13
- Acquiring Token	13
- Token Information	13
Administrators Security	14
- Superuser Account	14

CH4: Deployment Guide:

Infrastructure	16
- Storage (NFS Server)	17
- Hypervisors (KVM)	18
- Network (Open vSwitch)	18
- Configuring Pools	19
Middle-wares	20
- RabbitMQ	20
- Redis	20
- Zookeeper	21
Backend Services	22
- Database (PostgreSQL)	23
- Inventory	23
- Controller	25

- Notification	27
Automation Services	29
- Monitoring	30
- Compute	33
- Network	35
.....	

CH5: End-User Guide:

Using the Dashboard

- Workspaces
 - Users
 - Virtual Machines
 - Networks
-

CH6: Inventory:

Roles

- User Login/Sign-up
- Generating Tokens
- Data Storage

ERD

Class diagram

.....

CH7: Controller:

Roles

- Object Manipulation
- Queue Monitoring
- Retry & Rollback

Components

- ControllerAPI
- Network Notification Consumer
- VM Notification Consumer

ERD

Class Diagram

CH8: Notification:

Roles

- Workspace Notification Rooms
 - Handling End-User Connections
 - Handling Controller Connections
-

CH9: Monitoring:

Roles

- Monitoring Hosts & Pools
- Collecting Resource Usage Data

Components

- Celery Workers
 - Zookeeper Clients
 - RPC Servers
-

CH10: Compute:

Roles

- VM Creation & Deletion
- Generating Network Tasks

Components

- VM Worker
- Rollback Worker

Class Diagram

CH11: Network:

Roles

- Network Creation & Deletion
- Managing VM Interfaces

Components

- Network Worker
- Rollback Worker

Class Diagram

CH12: Availability, Reliability & Coordination:

Availability

- Multiple Instances
- Infrastructure Monitoring & Recovery

Reliability

- Handling Connection Failures
- Handling Service Failures
- Handling Host Failures
- Handling Execution Errors & Rollbacks

Coordination

- Compute-to-Network Coordination
 - Monitoring Coordination
-

CH13: System APIs:

Inventory API

Controller API

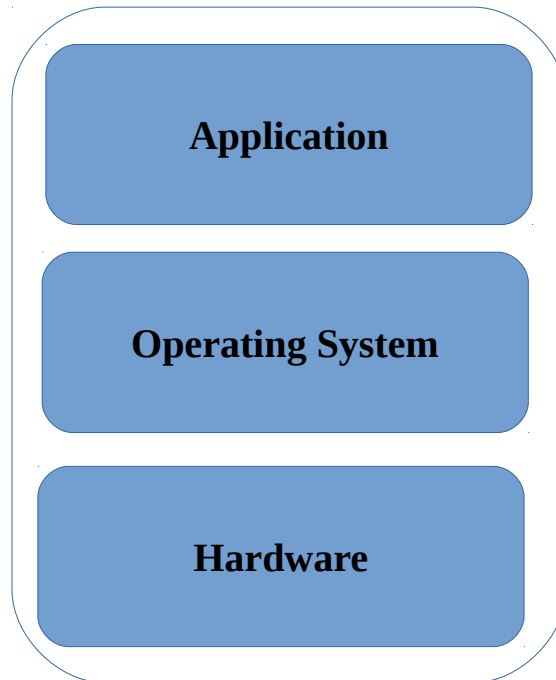
.....

Appendix A: System Code

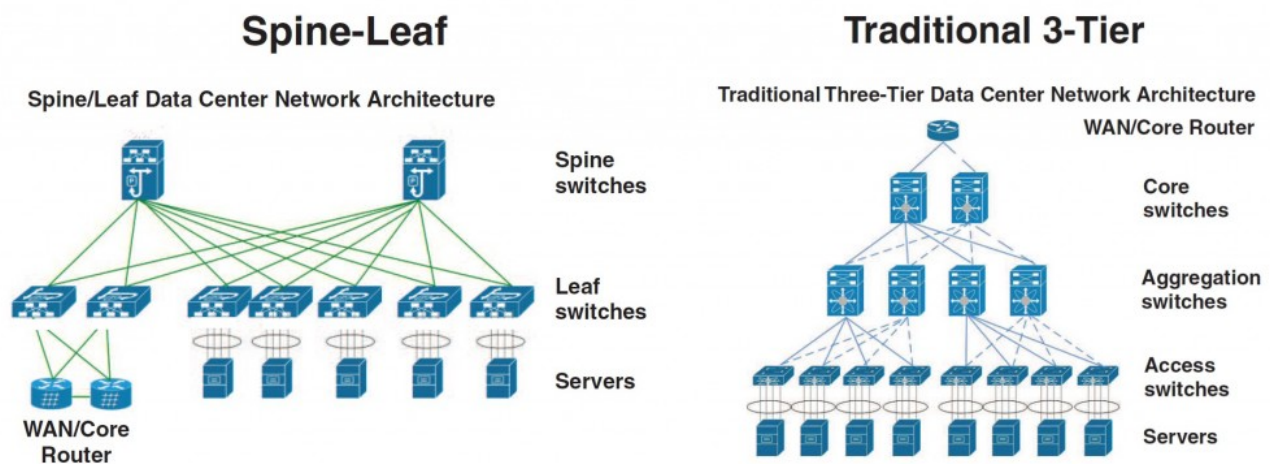
1- Introduction

Traditional Computing Model:

Traditional computing refers to using separate physical servers for each application or service. As for networking, this models uses the traditional 3-Tiers or Spine-Leaf network architecture.



Figure(1.A) App/Server



Figure(1.B) Traditional Network

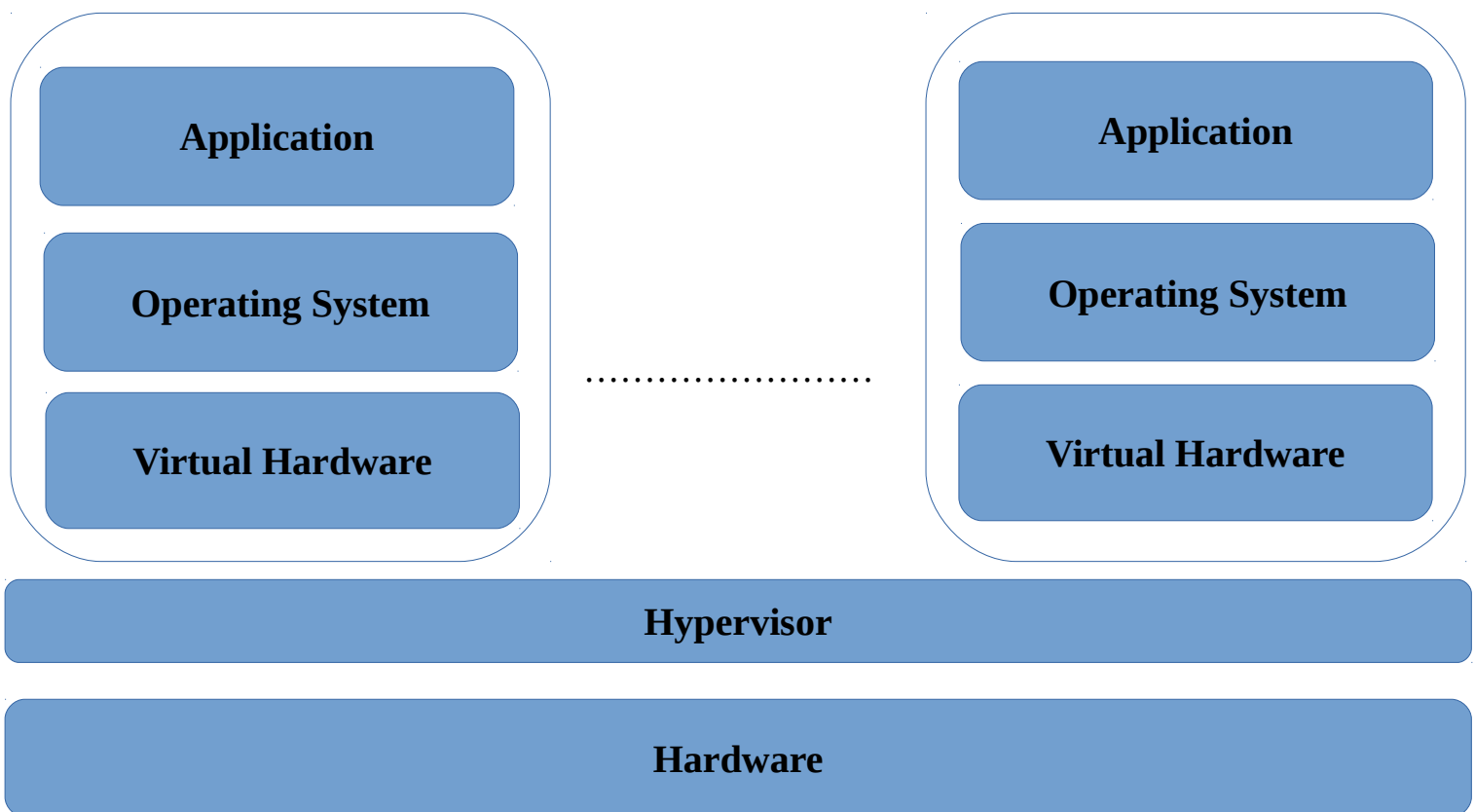
Drawbacks:

As one would expect, this model costs heavily and hard to manage and maintain. Not to mention the required time to add a new service or scale an existing one.

Cloud Computing Model:

The main goal of cloud computing is to add flexibility and self-service to infrastructure management which became possible thanks to virtualization technology. Nowadays, servers, networks and storage can be virtualized and run completely in software. Running in software means it is faster and easier to create and manipulate virtual resources using graphical dashboards, CLI tools or APIs which is provided by virtualization. Cloud adds a layer of automation and orchestration to virtualization and some form of user permission management.

Of course this model does have drawbacks especially in public cloud where security is basically zero (storing all your data at cloud providers). Also, there's a resource usage overhead by hypervisors and software used to manage the infrastructure.



Figure(1.C) Virtualization Model

Our Platform:

Kloudak is a simple cloud platform created to provide a scalable, easy-to-implement management and automation platform for Linux-based infrastructure. Kloudak offers the basic services of users and group management as well as managing the lifecycle of virtual compute and network resources.

2- Kloudak High-Level Architecture

Components & Roles:

Kloudak components can be organized in three main groups:

1- Data Services:

Or back-end services, they are responsible for handling end-user interactions, generating and storing their data and finally sending notifications to end-users.

They are: *Inventory, Controller, Notification and Dashboard.*

2- Automation Services:

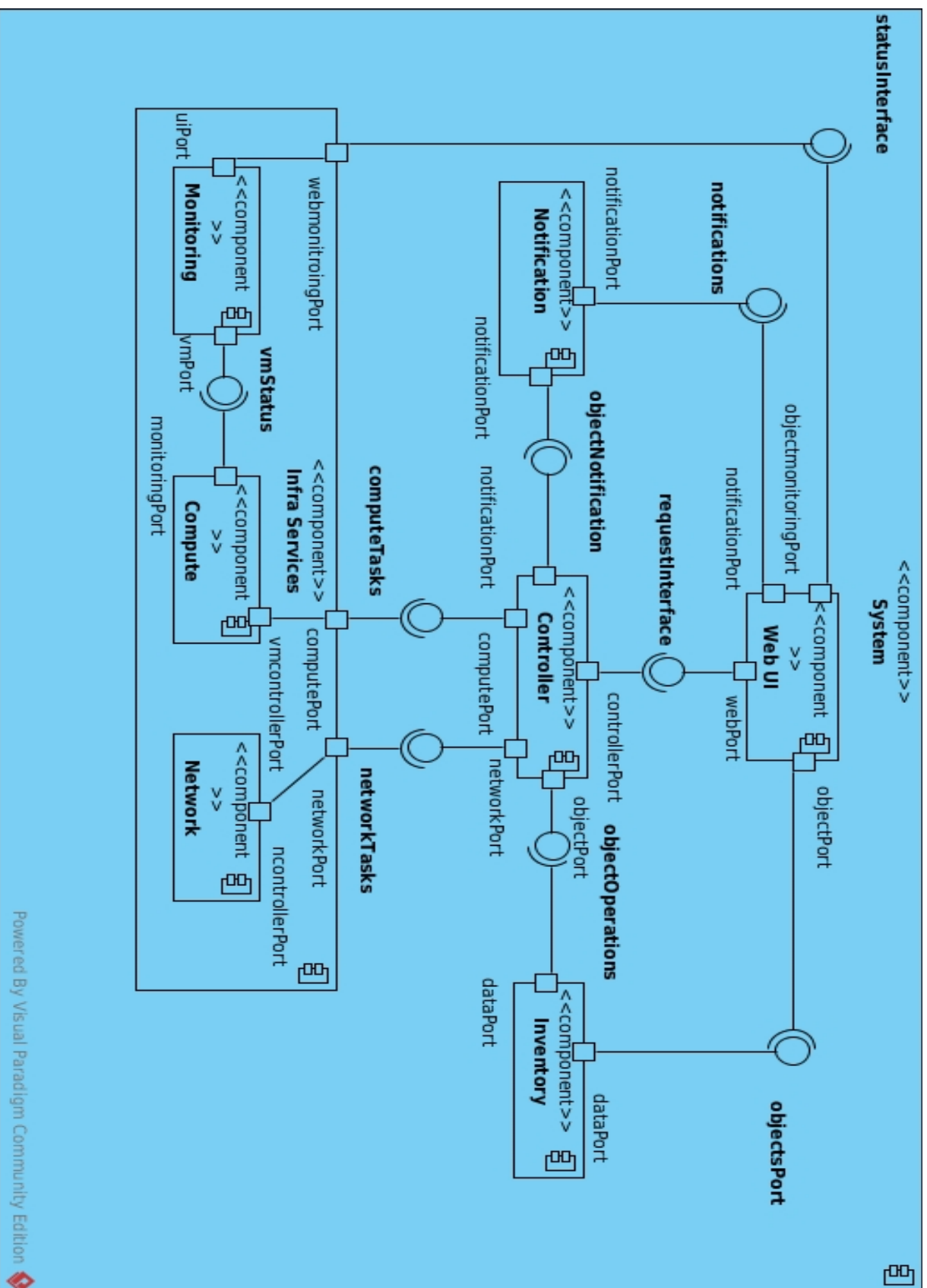
They manage the infrastructure, create and manage virtual resources. They are also used to monitor and collect infrastructure and virtual machines resource usage data.

They are: *Compute, Network, Monitoring*

3- Infrastructure:

This refers to the physical nodes used as hypervisors to host virtual machines and nodes used to provide shared storage.

(Note: there's also messaging and coordination middle-wares used in Kloudak and will be discussed in chapter 4: Deployment Guide)



Basic Concepts:

1- Areas:

An area represents a failover domain and a network subnet. Failover means that all hosts in the same area can be used to restart failed virtual machines due to their hosting hypervisor failure. For that purpose all hosts in an area are connected to all shared storage pools in that area.

2- Storage Pools:

They refer to storage space used to store virtual machine files. No matter what underlying technology you use for shared storage, the only restriction is that each storage pool is mounted in the same path on all hypervisors.

3- JSON-WEB-TOKEN:

or JWT is an authentication & authorization technology where each user have a unique token generated when logged in which contains user information. The user attaches his token in every request to be validated at the server-side.

We use JWT to store user permissions and to distinguish superusers who are allowed to manipulate inventory objects.

3- Authentication & Authorization

End-User Security:

1- Login:

Users can login by posting 'username' and 'password' to the inventory URL (/login/).

2- Acquiring Token:

Only inventory service contain user information and permissions and for that reason, it generates security token to be used as identification when interacting with other services.

After logging in to inventory, users should obtain their token to be able to interact with other services (controller and notification) from inventory URL (/get_token/) using GET method.

The token should be embedded in controller requests' header of 'token', and as a parameter when connecting to notification. For example (/Workspace-01/<token value>).

3- Token Information:

Information embedded inside user token basically looks like this:

```
{ 'username': <name>,  
'email': <email>,  
<workspace name>: {  
    'user_can_add': <True/False>,  
    'user_can_edit': <True/False>,  
}
```

```

    'user_can_delete': <True/False>,
    'vm_can_add': <True/False>,
    'vm_can_edit': <True/False>,
    'vm_can_delete': <True/False>,
    'network_can_add': <True/False>,
    'network_can_edit': <True/False>,
    'network_can_delete': <True/False>
  },

```

.....

```

}

```

For each workspace a user belongs to, a permissions dictionary is embedded in the token as a value for a key of the workspace name.

Administrators Security:

1- Superuser Account:

End-user accounts can only view objects in inventory (except for workspaces and users) but cannot add or delete objects directly from inventory as this must only be done by controller. Thus, we build superuser accounts used in internal interactions between services and require no login.

Superuser accounts use a token with a username of the superuser inside each service.

So basically it can look only like this:

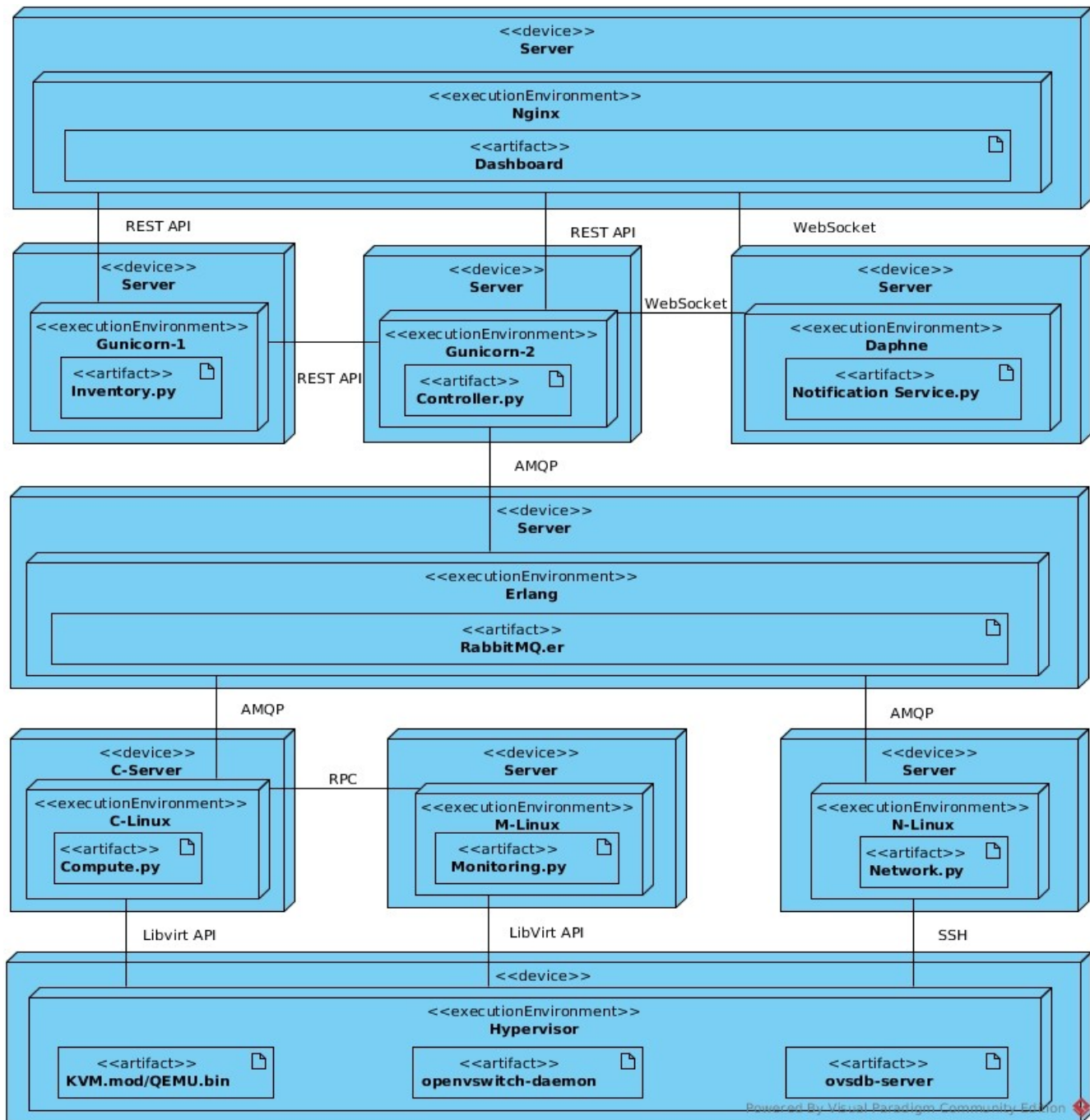
```

{'username': <superuser name>}

```

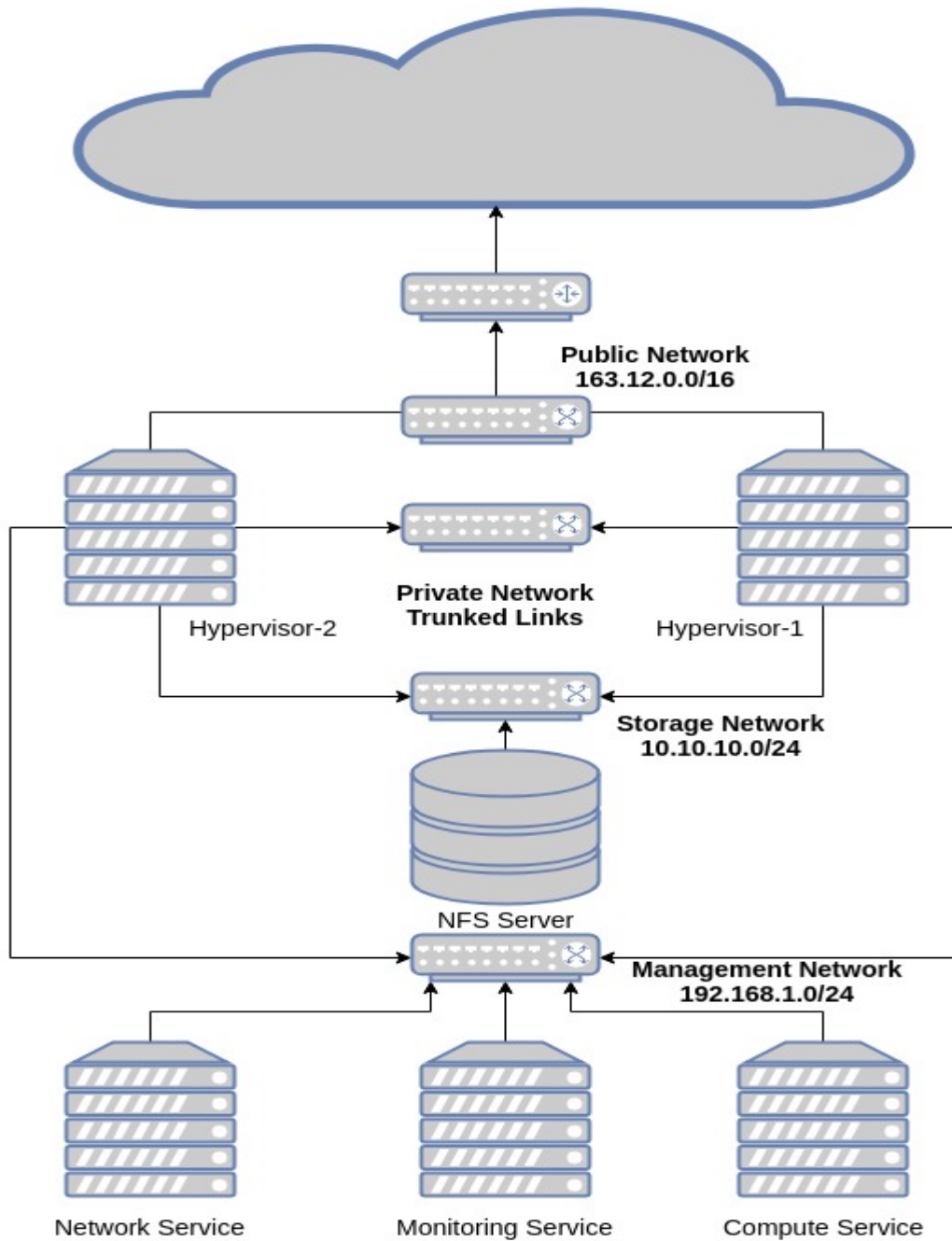
4- Deployment Guide

In this chapter we will walk through a simple deployment of Kloudak.



Infrastructure:

For simplicity the infrastructure will consist of one area with two hypervisors each with four NICs and a single storage pool in it of type NFS. It should look like this:



1- Storage (NFS-Server):

In this walk through we are using Fedora 27 to install the server.

First, install nfs-utils packages:

```
$ sudo dnf install -y nfs-utils
```

We also need to configure the firewall to allow nfs ports:

```
$ sudo firewall-cmd --add-service=nfs --permanent  
$ sudo firewall-cmd --reload
```

Now enable and restart the server:

```
$ sudo systemctl enable nfs  
$ sudo systemctl restart nfs
```

(Note: sometimes nfs server uses different ports which will need to be allowed manually depending on your case or you can just disable firewall :D).

Second, export directory to NFS:

```
$ cat /etc/exports  
  
/var/lib/nfsroot  
10.10.10.0/24(rw,sync,no_root_squash,no_subtree_check)  
$ sudo mkdir /var/lib/nfsroot  
$ sudo chmod 777 /var/lib/nfsroot  
$ sudo exportfs -ra
```

2- Hypervisors (NFS-Server):

We are using KVM as a hypervisor and QEMU for hardware emulation on top of Fedora 27.

First, install virtualization group:

```
$ sudo dnf groupinstall virtualization
$ sudo systemctl enable libvirtd
$ sudo systemctl start libvirtd
$ sudo systemctl enable sshd
$ sudo systemctl start sshd
```

3- Network (Open vSwitch):

We will use OVS bridges on hypervisors to connect virtual machines to public and private networks.

Each host should have four NICs:

- 1- Connected to storage network.
- 2- Connected to management network.
- 3- Connected to public network.
- 4- Connected to private network.

First, install Open vSwitch:

```
$ sudo dnf install openvswitch
$ sudo systemctl enable openvswitch
$ sudo systemctl enable openvswitch
```

Second, configure bridges:

```
$ sudo ovs-vsctl add-br public-br
$ sudo ovs-vsctl add-br private-br
$ sudo ovs-vsctl add-port public-br <interface-in-public-network>
```

```
$ sudo ovs-vsctl add-port private-br <interface-in-private-network>
$ sudo ip addr flush dev <interface-in-public-network>
$ sudo ip addr flush dev <interface-in-private-network>
$ sudo ip link set <interface-in-public-network> up
$ sudo ip link set <interface-in-private-network> up
$ sudo ovs-vsctl set port <interface-in-private-network>
vlan_mode=trunk
```

4- Configuring Pools:

First, create the directory where the pool will be mounted:

```
$ sudo mkdir /var/lib/pool-01
```

Second, create an XML file called 'pool-01.xml' for the pool:

```
<pool type='netfs'>
  <name>pool-01</name>
  <source>
    <host name='10.10.10.10'/>
    <dir path='/var/lib/nfsroot'/>
  </source>
  <target>
    <path>/var/lib/pool-01</path>
  </target>
</pool>
```

Finally, use virsh to define and start the pool on both hosts:

```
$ sudo virsh pool-define pool-01.xml
$ sudo virsh pool-start pool-01
```

(Execute the previous steps on both hypervisors)

Middle-wares:

We will walk through basic installation of the messaging and coordination middle-wares.

1- *RabbitMQ:*

RabbitMQ is an enterprise messaging middle-ware. We use it in Kloudak to queue and deliver messages between controller service and automation services. It is also used as RPC middle-ware for the server in monitoring service and client in compute service.

For the installation, we will use Fedora 27 as operating system.

First, install rabbitmq-server:

```
$ sudo dnf install rabbitmq-server
```

Second, enable and start the server:

```
$ sudo systemctl enable rabbitmq  
$ sudo systemctl start rabbitmq
```

2- *Redis:*

Redis is used as a back-end for the notification service to provide highly scalable real-time web application.

First, install redis:

```
$ sudo dnf install redis
```

Second, enable and start the server:

```
$ sudo systemctl enable redis
$ sudo systemctl start redis
```

3- Zookeeper:

Zookeeper provides many requirements for distributed systems such as: leader election and distributed coordination primitives. We use it in the monitoring service to elect a leader for the monitoring cluster which would be responsible for assigning tasks to the rest of the cluster using Celery.

First, install zookeeper:

```
$ sudo dnf install zookeeper
```

Second, create the server configuration file and name it zoo1.cfg:

```
tickTime=2000
```

```
initLimit=5
```

```
syncLimit=2
```

```
dataDir=/var/lib/zookeeper/zoo1
```

```
clientPort=2181
```

```
server.1=localhost:2666:3666
```

Finally, enable and start the server:

```
$ sudo systemctl enable zookeeper
$ sudo systemctl start zookeeper
$ zkServer start zoo1.cfg
```

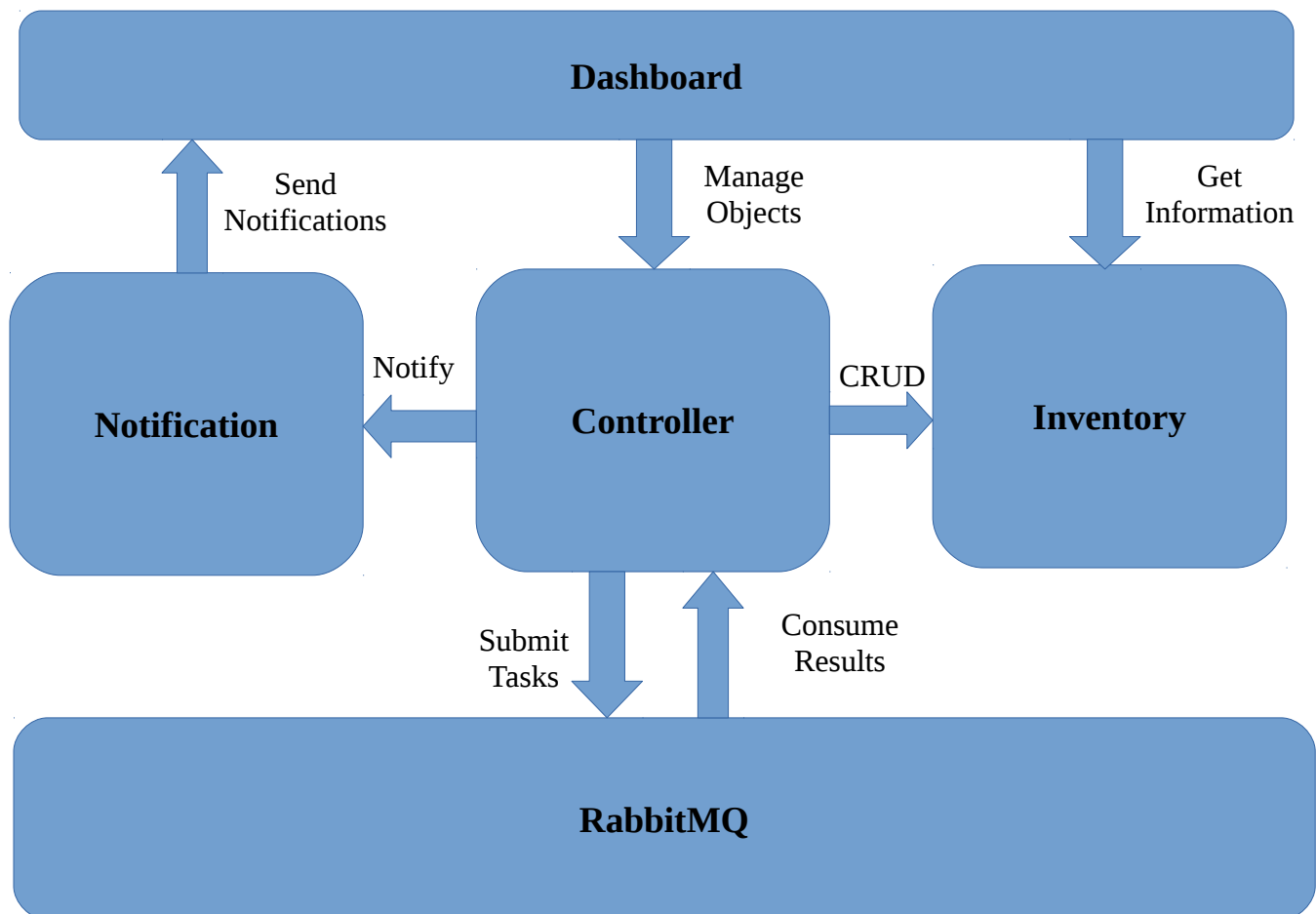
Backend Services:

Back-end services are written in Python 3.6 using Django 2.0. Each service is using PostgreSQL as a database.

To obtain Kloudak source code from github:

```
$ git clone git://github.com/m-motawea/Kloudak.git
```

The following diagram shows the relation between these services:



1- Database (PostgreSQL):

Here, we will install one database server, but you should install a separate database server for each service.

First, install these packages:

```
$ sudo dnf install postgresql postgresql-server postgresql-devel
```

Second, enable and start the server:

```
$ sudo systemctl enable postgres  
$ sudo systemctl start postgres
```

2- Inventory:

The next commands are executed inside Inventory directory in Kloudak source code except for database creation commands are executed on PostgreSQL server.

First, install the dependencies:

```
$ sudo pip3.6 install -r Inventory_Service/requirements.txt
```

Second, install Gunicorn server to run the application:

```
$ sudo pip3.6 install gunicorn
```

Third, edit the settings.py file inside Inventory_Service/Inventory_Service in database section use the following:

```
DATABASES = {  
  
    'default': {  
  
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
```

```
'NAME': 'inventory',  
  
'USER': 'inv_user',  
  
'PASSWORD': 'Maglab123!',  
  
'HOST': '172.17.0.1',  
  
'PORT': '5432',  
  
}  
  
}
```

Fourth, configure the inventory database inside PostgreSQL:

```
$ sudo su postgres  
$ psql  
$ CREATE DATABASE inventory;  
$ CREATE USER inv_user WITH PASSWORD 'Maglab123!';  
$ ALTER ROLE inv_user SET client_encoding TO 'utf8';  
$ ALTER ROLE inv_user SET default_transaction_isolation TO 'read committed';  
$ GRANT ALL ON DATABASE inventory TO inv_user;
```

Fifth, create the database schema using django migrations:

```
$ python3.6 Inventory_Service/manage.py makemigrations  
$ python3.6 Inventory_Service/manage.py migrate
```

Sixth, create a superuser:

```
$ python3.6 Inventory_Service/manage.py createsuperuser  
Username: maged  
Email: <whatever>  
Password: <whatever>
```


Finally, start the server:

```
$ Inventory_Service/start.sh
```

3- Controller:

The next commands are executed inside Contoller directory in Kloudak source code except for database creation commands are executed on PostgreSQL server.

First, install the dependencies:

```
$ sudo pip3.6 install -r Controller_Service/requirements.txt
```

Second, install Gunicorn server to run the application:

```
$ sudo pip3.6 install gunicorn
```

Third, edit conf.json inside Controller_Service/ControllerAPI with the following:

```
{  
    "inventory": "http://172.17.0.1:5000/",  
    "notification": "localhost",  
    "broker": "172.17.0.1",  
    "retries": 2,  
    "wait": 1  
}
```

Fourth, edit the database section inside settings.py file inside Controller_Service -/Controller_Service with the following:

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql_psycopg2',  
        'NAME': 'controller',  
        'USER': 'cont_user',  
        'PASSWORD': 'Maglab123!',  
        'HOST': '172.17.0.1',  
        'PORT': '5432',  
    }  
}
```

Fifth, configure the controller database inside PostgreSQL:

```
$ sudo su postgres  
$ psql  
$ CREATE DATABASE controller;  
$ CREATE USER cont_user WITH PASSWORD 'Maglab123!';  
$ ALTER ROLE cont_user SET client_encoding TO 'utf8';  
$ ALTER ROLE cont_user SET default_transaction_isolation TO 'read committed';  
$ GRANT ALL ON DATABASE controller TO cont_user;
```

Sixth, create the database schema using django migrations:

```
$ python3.6 Controller_Service/manage.py makemigrations  
$ python3.6 Controller_Service/manage.py migrate
```

Seventh, create a superuser:

```
$ python3.6 Controller_Service/manage.py createsuperuser
```

Username: maged

Email: <whatever>

Password: <whatever>

Eighth, start notification consumers:

```
$ python3.6 Controller_Service/QueueMonitoring vm_notification_consumer.py
```

```
$ python3.6 Controller_Service/QueueMonitoring network_notification_consumer.py
```

Finally, start the server:

```
$ Controller_Service/start.sh
```

4- Notification:

The next commands are executed inside Notification directory in Kloudak source code.

First, install the dependencies:

```
$ sudo pip3.6 install -r mysite/requirements.txt
```

Second, install Daphne server to run the application:

```
$ sudo pip3.6 install daphne
```

Third, create the database schema using django migrations:

```
$ python3.6 mysite/manage.py makemigrations
```

```
$ python3.6 mysite/manage.py migrate
```

Fourth, create a superuser:

```
$ python3.6 mysite/manage.py createsuperuser
```

Username: maged

Email: <whatever>

Password: <whatever>

Fifth, edit channels layer section in settings.py in mysite/mysite with the following:

```
CHANNEL_LAYERS = {  
    'default': {  
        'BACKEND': 'channels_redis.core.RedisChannelLayer',  
        'CONFIG': {  
            "hosts": [('172.17.0.1', 6379)],  
        },  
    },  
}
```

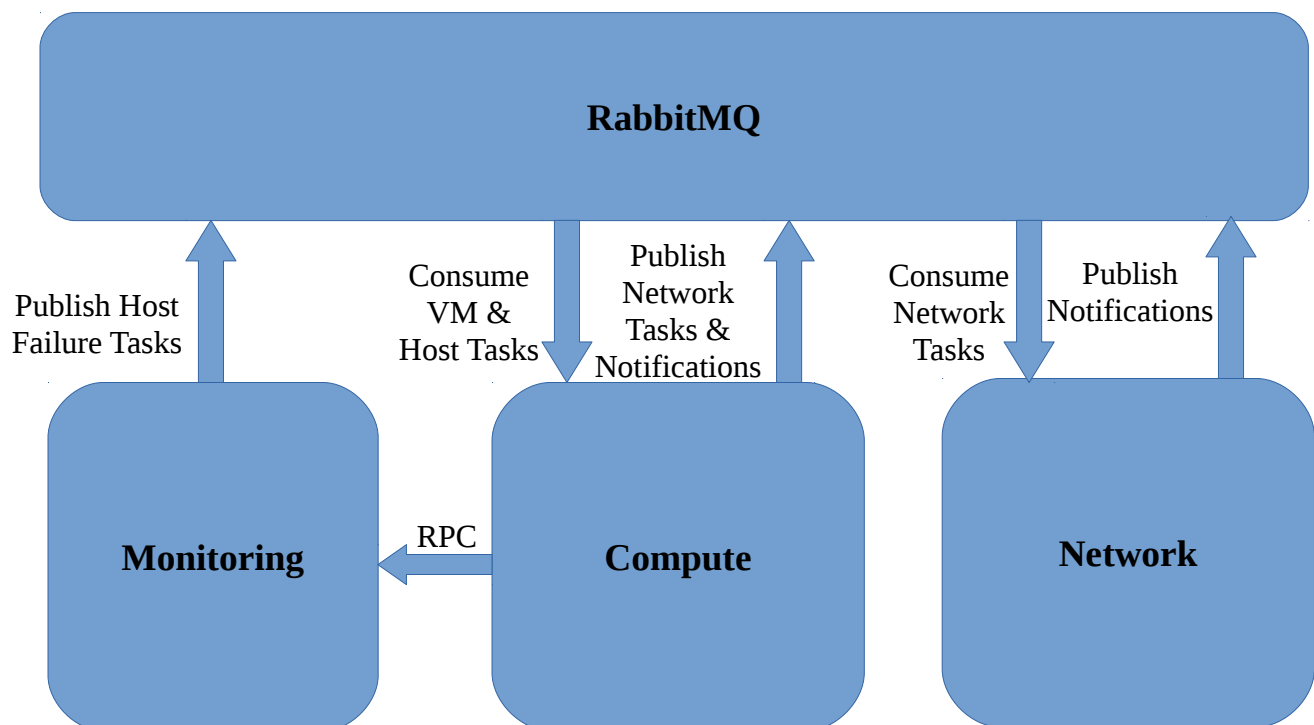
Finally, start the server:

```
$ mysite/start.sh
```

Automation Services:

Back-end services are written in Python 3.6 and they are the layer that interacts with the infrastructure to create and manage Virtual Machines and networks lifecycle as well as monitoring the infrastructure.

The following diagram describes the relation between them:



1- Monitoring:

The next commands are executed inside Monitoring directory in Kloudak source code except for database creation commands are executed on PostgreSQL server.

First, install the following packages:

```
$ sudo dnf install libvirt libvirt-devel
```

Second, install the dependencies:

```
$ sudo pip3.6 install -r requirements.txt
```

Third, configure the database:

```
$ sudo su postgres
```

```
$ psql
```

```
$ CREATE DATABASE monitor;
```

```
$ CREATE USER mon_admin WITH PASSWORD 'Maglab123!';
```

```
$ GRANT ALL ON DATABASE monitor TO mon_admin;
```

Fourth, edit conf.json file with the following:

```
{  
  
  "broker": "172.17.0.1",  
  
  "database": "172.17.0.1",  
  
    "time":10  
  
}
```

Fifth, create the database schema:

```
$ python3.6 db_setup.py
```

Sixth, edit pop_db.py file with the following:

```
#!/usr/bin/python3.6
from orm_schema import Host, Area, Pool
from orm_io import dbIO
io = dbIO('localhost')
a = Area(area_name='Area-01')
io.add([a])

a = io.query(Area, area_name='Area-01')[0]
h1 = Host(
    host_name='kvm-1',
    host_ip='192.168.1.7',
    host_cpus=2,
    host_memory=8,
    host_free_memory=8,
    state=True,
    area_id=a.area_id)
h2 = Host(
    host_name='kvm-2',
    host_ip='192.168.1.8',
    host_cpus=2,
    host_memory=8,
    host_free_memory=8,
    state=True,
    area_id=a.area_id
)
io.add([h1, h2])

p1 = Pool(
    pool_name='pool-01',
    pool_path='/var/lib/pool-01/',
    pool_size=20,
    pool_free_size=20,
    area_id=a.area_id
)
io.add([p1])
```

Seventh, populate the database with the previous information:

```
$ python3.6 pop_db.py
```

Eighth, install and start a zookeeper client

```
$ sudo dnf install zookeeper
```

```
$ sudo systemctl start zookeeper
```

```
$ sudo zkCli.sh -server localhost:2181
```

Ninth, start a celery worker:

```
$ celery -A tasks worker --loglevel=info --hostname=h1
```

Tenth, start the RPC server:

```
$ python3.6 rpcServer.py
```

Eleventh, add monitoring ssh key to trusted keys on both hypervisors:

```
$ ssh-copy-id root@kvm-1
```

```
$ ssh-copy-id root@kvm-2
```

Finally, start the monitor service:

```
$ python3.6 monitor.py
```

2- Compute:

The next commands are executed inside Compute directory in Kloudak source code except for database creation commands are executed on PostgreSQL server.

First, install the following packages:

```
$ sudo dnf install libvirt libvirt-devel
```


Second, install the dependencies:

```
$ sudo pip3.6 install -r Worker/requirements.txt
```

Third, configure the database:

```
$ sudo su postgres
```

```
$ psql
```

```
$ CREATE DATABASE compute;
```

```
$ CREATE USER comp_admin WITH PASSWORD 'Maglab123!';
```

```
$ GRANT ALL ON DATABASE compute TO comp_admin;
```

Fourth, edit Worker/conf.json file with the following:

```
{  
  
    "broker": "172.17.0.1",  
  
    "database": "172.17.0.1",  
  
}
```

Fifth, create the database schema:

```
$ python3.6 Worker/db_setup.py
```

Sixth, edit Worker/pop_db.py with the following:

```
#!/usr/bin/python3.6  
  
from lib2.orm_schema import Host, Area, Pool, Template  
from lib2.base import dbIO  
io = dbIO('localhost')  
a = Area(  
    area_name='Area-01',
```

```

area_gw='163.12.0.1'
)
io.add([a])
t = Template(
template_name='Template-01',
template_path='/var/lib/pool-01/',
template_ifname='eth0'
)
a = io.query(Area, area_name='Area-01')[0]
h1 = Host(
host_name='kvm-1',
host_ip='192.168.1.7',
host_cpus=2,
host_memory=8,
host_free_memory=8,
state=True,
area_id=a.area_id
)
h2 = Host(
host_name='kvm-2',
host_ip='192.168.1.8',
host_cpus=2,
host_memory=8,
host_free_memory=8,
state=True,
area_id=a.area_id
)
io.add([h1, h2])
p1 = Pool(
pool_name='pool-01',
pool_path='/var/lib/pool-01/',
pool_size=20,
pool_free_size=20,
area_id=a.area_id
)
io.add([p1])

```

Fifth, create the database schema:

```
$ python3.6 Worker/pop_db.py
```

Sixth, download and extract Fedora cloud image to the pool and name it Template-01.raw from the following url:

https://download.fedoraproject.org/pub/fedora/linux/releases/28/Cloud/x86_64/images/Fedora-Cloud-Base-28-1.1.x86_64.raw.xz

Seventh, add the ssh key to trusted keys on both hypervisors:

```
$ ssh-copy-id root@kvm-1
```

```
$ ssh-copy-id root@kvm-2
```

Eighth, run the rollbackWorker:

```
$ python3.6 Worker/rollbackWorker.py
```

Finally, start the main worker:

```
$ python3.6 Worker/worker.py
```

3- Network:

The next commands are executed inside Network directory in Kloudak source code except for database creation commands are executed on PostgreSQL server.

First, install the dependencies:

```
$ sudo pip3.6 install -r VLAN/requirements.txt
```

Second, configure the database:

```
$ sudo su postgres
```

```
$ psql
```

```
$ CREATE DATABASE network;
```

```
$ CREATE USER net_admin WITH PASSWORD 'Maglab123!';
```

```
$ GRANT ALL ON DATABASE network TO net_admin;
```

Third, edit Worker/conf.json file with the following:

```
{  
  
  "broker": "172.17.0.1",  
  
  "database": "172.17.0.1",  
  
}
```

Fourth, create the database schema:

```
$ python3.6 VLAN/db_setup.py
```

Fifth, edit VLAN/pop_db.py with the following:

```
#!/usr/bin/python3.6  
  
from lib.orm_schema import base, Area, Host  
from lib.orm_io import dbIO  
from config import get_config  
conf_dict = get_config('conf.json')  
io = dbIO(conf_dict['database'])  
a = io.query(Area, area_name='Area-01')[0]  
h1 = Host(  
  host_name='kvm-1',  
  host_ip='192.168.1.7',  
  state=True,  
  area_id=a.area_id  
)  
h2 = Host(  
  host_name='kvm-2',  
  host_ip='192.168.1.8',  
  state=True,  
  area_id=a.area_id  
)  
io.add([h1, h2])
```

Sixth, populate the database:

```
$ python3.6 VLAN/pop_db.py
```

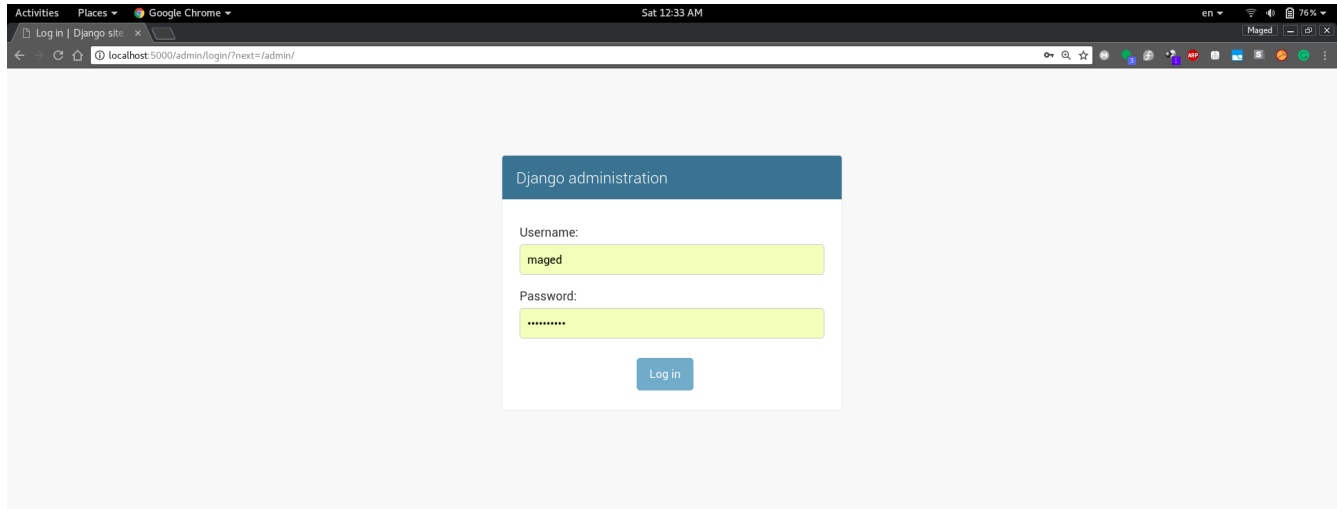
Seventh, run the rollback worker:

```
$ python3.6 VLAN/rollbackWorker.py
```

Finally, start the main service worker:

```
$ python3.6 VLAN/worker.py
```

Now, open a web browser and go to inventory administration page and login with superuser credentials.



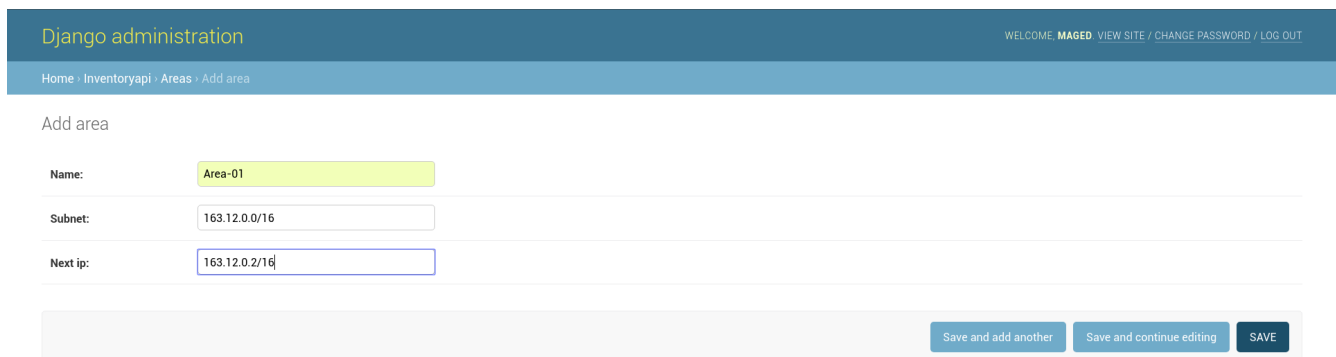
Django administration

Username:
maged

Password:

Log in

Choose add next to Areas to create a new one with following data:



Django administration

WELCOME, **MAGED** / VIEW SITE / CHANGE PASSWORD / LOG OUT

Home / Inventoryapi / Areas / Add area

Add area

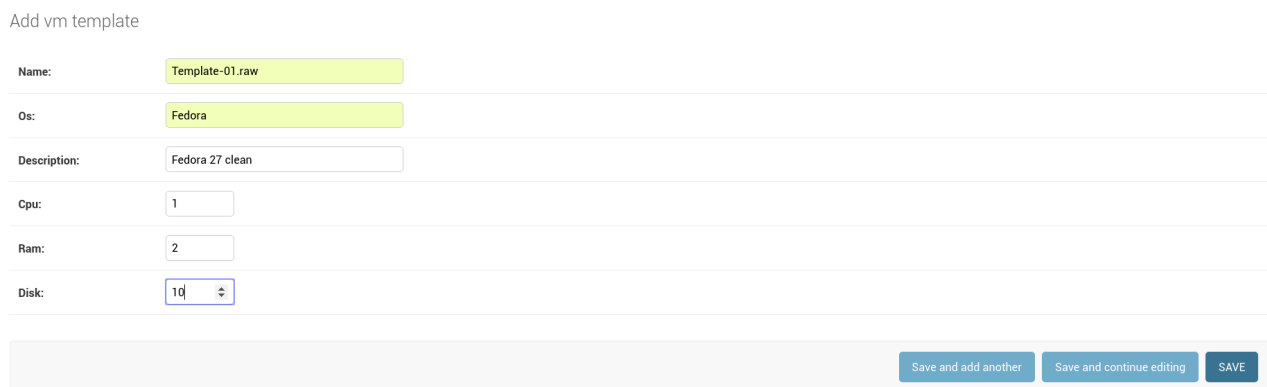
Name: Area-01

Subnet: 163.12.0.0/16

Next ip: 163.12.0.2/16

Save and add another Save and continue editing SAVE

Choose add next to Templates to create a new one with the following data:



Add vm template

Name: Template-01.raw

Os: Fedora

Description: Fedora 27 clean

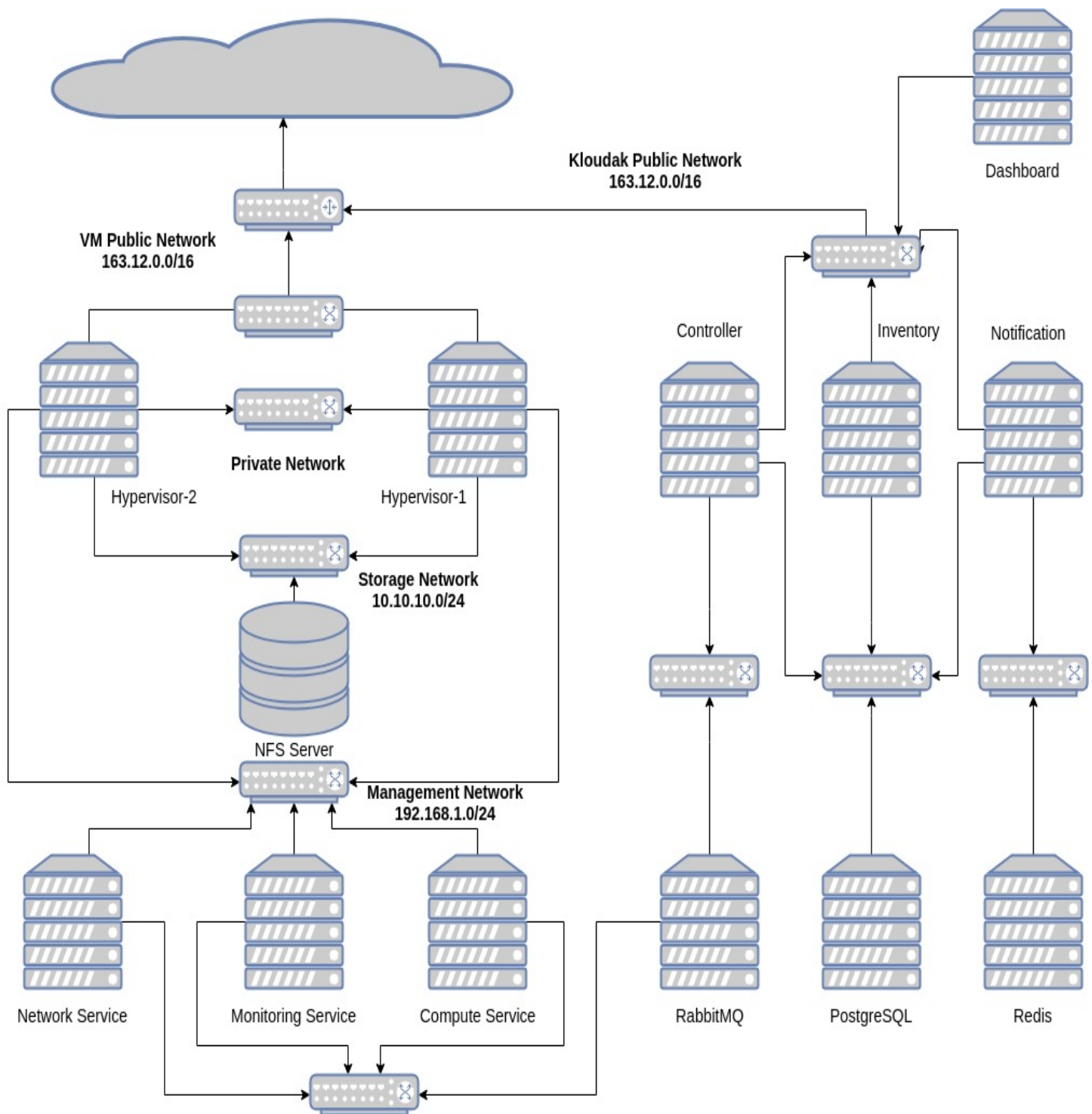
Cpu: 1

Ram: 2

Disk: 10

Save and add another Save and continue editing SAVE

Your deployment should look like this:



6- Inventory

Roles:

1- User Login/Sign up:

Inventory service is considered the identity provider of Kloudak platform. It handles all user related operation.

For login users use POST to send username and password with the same ids to the inventory URL (/login/).

For sign up use POST to send username, password and email with the same ids to the inventory URL (/signup/).

2- Generating Tokens:

Once a user is logged in, he is redirected to a workspace which he is part of or, to create a new workspace if he is not part of any workspaces.

To start interact with the objects in a workspace, a user needs an authentication token to use it as part of his requests to controller and notification services.

To obtain the token use GET to the inventory URL (/get_token/).

3- Data Storage:

Inventory is mainly a data storage as the name suggests. It stores all information used by End-users which is:

- Templates:

Templates describe the SW & HW which would be the same in its children VMs.

name: (template name)

OS: (Operating system)

description: (small description of the template)

CPU: (number of CPUs)

RAM: (memory size in gigabytes)

disk: (hard disk size in gigabytes)

- Areas:

name: (area name)

subnet: (area subnet)

next available IP

- Workspaces:

A collection of users and resources which functions in a similar way to an organization.

name: (workspace name)

- CustomUsers:

It is basically a user profile in a workspace which contains his permissions

user: (a foreign key to the user account associated with this profile)

workspace: (a foreign key to the parent workspace)

vm_can_add: (boolean which describes the permission)

vm_can_edit: (boolean which describes the permission)

vm_can_delete: (boolean which describes the permission)

... the same goes for network and user permissions

- Networks:

name: (network name)

owner: (a foreign key to the parent workspace)

description: (small description for the network)

- VMs:

name: (vm name)

owner: (a foreign key to the parent workspace)

description: (small description for the network)

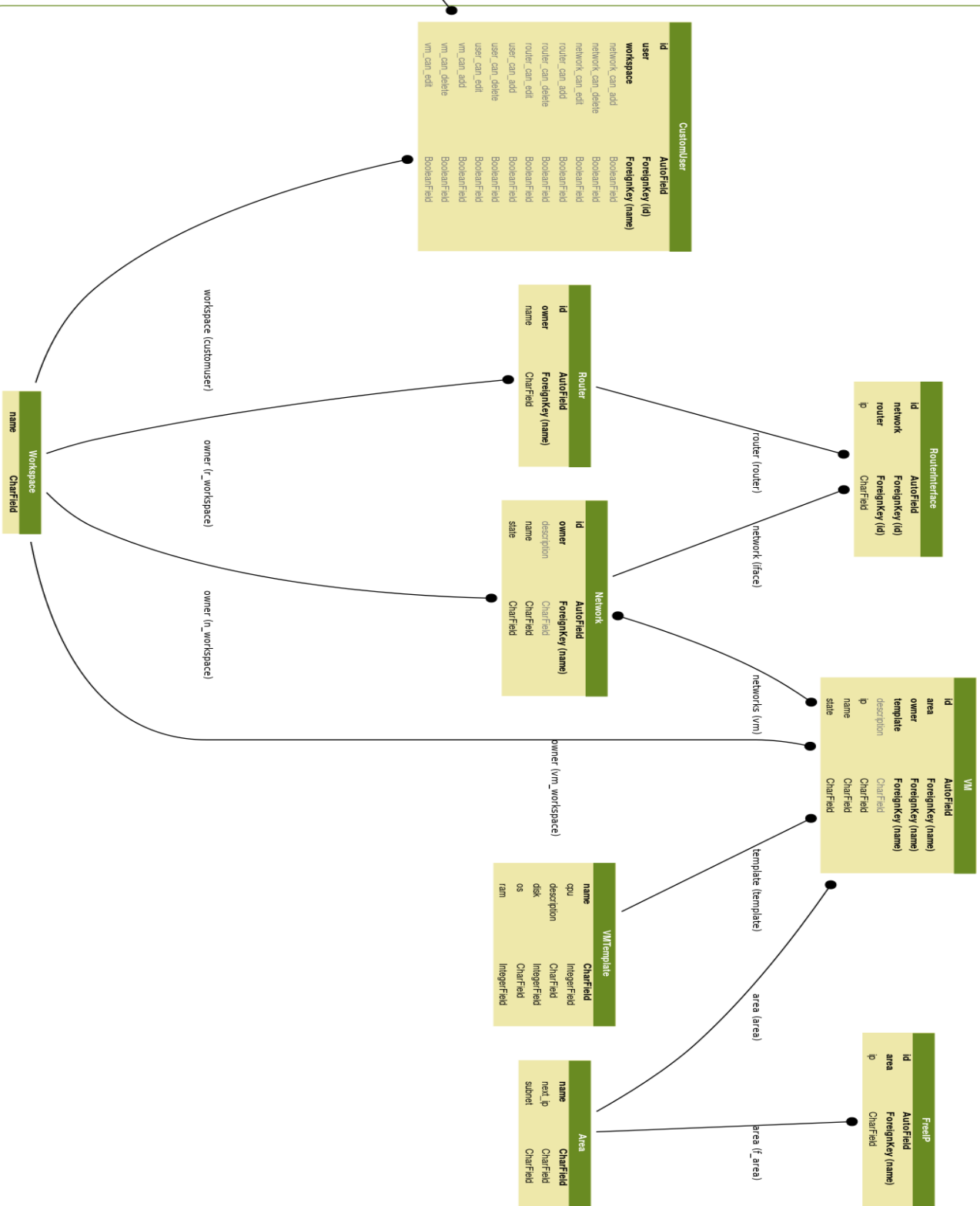
ip: (vm ip address)

area: (a foreign key to the area which the vm is part of)

template: (a foreign key to the base template for the vm)

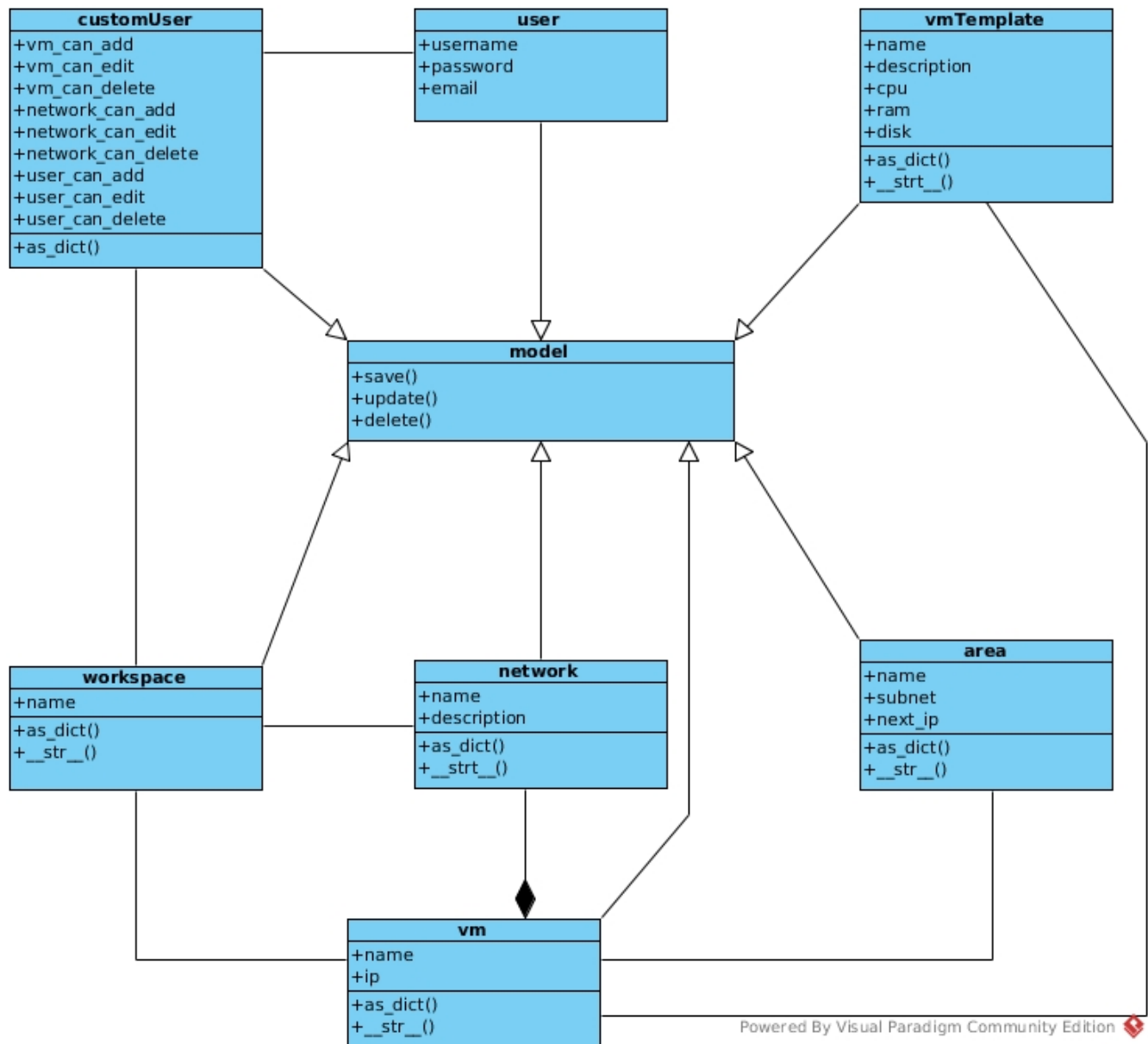
networks: (many-to-many field refers to networks connected to the vm)

InventoryAPI



ERD:

Class Diagram:



Powered By Visual Paradigm Community Edition

7- Controller

Roles:

1- Object Manipulation:

Users can only view information stored in inventory (except for workspaces and users).

To create, modify or delete any object they must use the controller API.

Controller service provides the following APIs to allow users to manipulate objects:

- network: URL (/networks/) for actions related to network objects.
- vm: URL (/vms/) for actions related to vm objects.

When a request is received, the controller first validates the token to make sure the user is part of the specified workspace and has the permissions required to perform the specified action.

After token validation, the controller validates the request information for example, if the request is to create a new network, the controller first validates that there is no networks with same name in the specified workspace. The validation is performed using GET requests on the URL of the specified object and checking the status code.

Finally, the controller dispatches a task to the corresponding queue (<vm/network>).

2- Queue Monitoring:

Automation services which consume the controller generated tasks, respond with notification message after finishing the task (whether it was successful or not) on a queue named (<vm/network>_notification).

Depending on the content of the notification message, specifically (status: <failed/successful>, retires: <int>), the controller opens a websocket connection to the workspace notification room with superuser token and sends a notification.

3- Retry & Rollback:

If the received notification message has a 'failed' status, the controller checks the value of 'retries' of that message. If the value is larger than 0, then the controller decrements that value and resubmits the task to the queue. Else, the controller initiates a rollback for that task and sends a notification to the dashboard with task failure.

Components:

1- ControllerAPI:

a REST API which handles user requests from the dashboard or any scripting/automation tool.

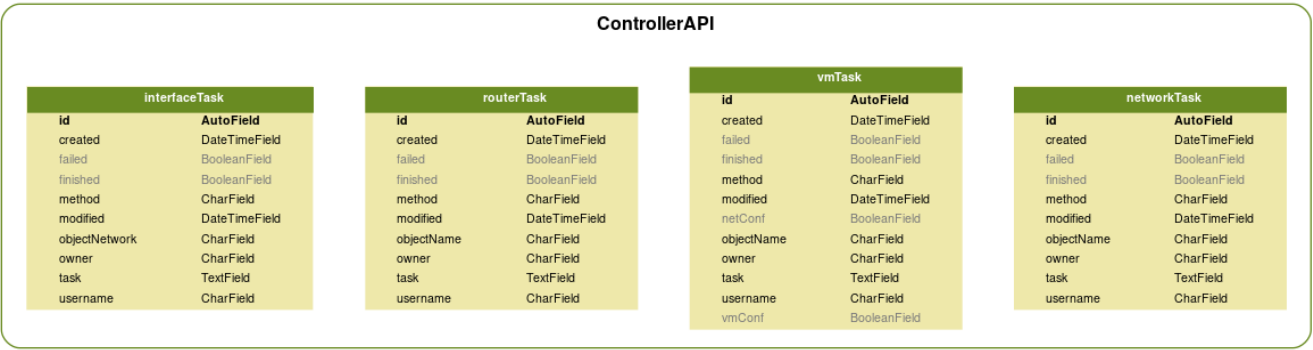
2- Network Notification Consumer:

a daemon process which consumes notification messages generated by network services, and sends notifications.

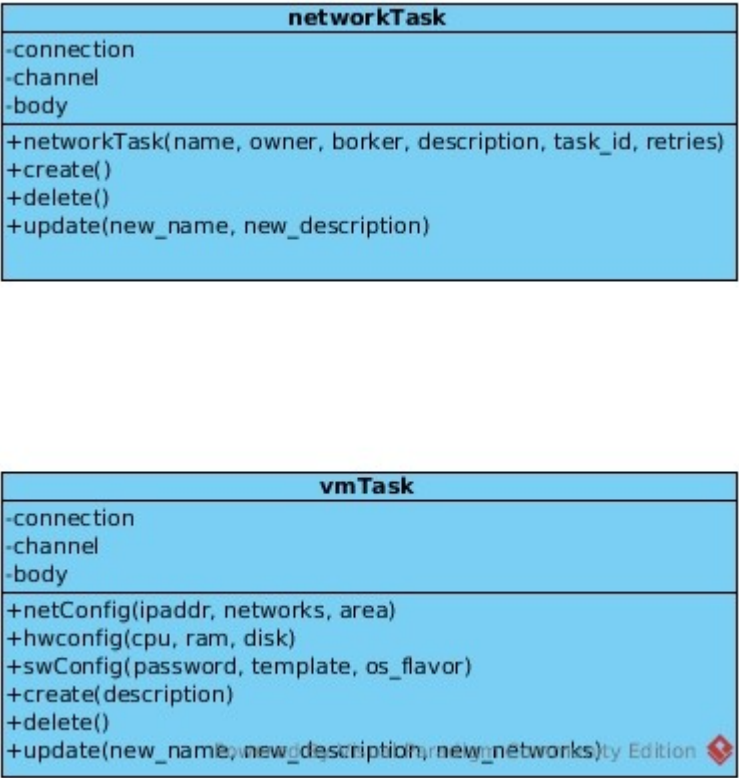
3- VM Notification Consumer:

a daemon process which consumes notification messages generated by compute services, and sends notifications.

ERD:



Class Diagram:



8- Notification

Roles:

1- Workspace Notification Rooms:

The notification service is basically a multi-room chat application built using django channels with redis as channels backend. With each workspace represented as a room.

2- Handling End-Users Connections:

Users connect to their workspace room to subscribe for notifications on the URL (ws://<notification>:3000/<workspace name>/<token>).

3- Handling Superuser Connections:

To send notifications, the controller connects to a room with the superuser token and then sends the message.

Only connections with superuser token can send messages to the members of the room.

