Cloud Computing

Final project options

Project for working students (1 student per group)

Option #0 - Cloud Application

Design and develop a simple cloud computing application based on containers. You can select any use-case you like. The cloud computing application will need the following components:

- A frontend layer Exposing a REST interface that receives and dispatches requests
- A backend layer that processes the requests coming from the frontend
- A database layer that stores the data

The user can interact with the application through the REST plugin installed on the browser to perform the retrieval of some data stored in the system. The backend must process the requests from users by interacting with the database.

In order to ensure scalability, a message-based queue system must be used for the communication between the frontend and the backend.

An additional module that produces and stores in the database new data periodically should be included in the backend. Fake data could be generated by using the Faker python library (https://pypi.org/project/Faker/).

Regular projects (4 people groups)

Option #1 – OpenStack: Timeseries as a service

Gnocchi is an open-source, multi-tenant timeseries, metrics and resources database. It provides an HTTP REST interface to create and manipulate the data. It is designed to store metrics at a very large scale on CEPH while providing access to metrics and resources information and history.

Install in an OpenStack instance the Gnocchi database and create two simple applications, one consumer and one producer, that exploit the REST interface exposed by Gnocchi to store/retrieve the data. The producer must mimic the periodic production of measurement data (random data is OK) to be stored in the timeseries database, the consumer must retrieve periodically some aggregated values like average, max, min, etc.

References:

- Gnocchi installation: https://jaas.ai/gnocchi/37
- Gnocchi REST interface: https://gnocchi.xyz/rest.html

Option #2 – OpenStack: Python SDK

Use the OpenStack SDK to create an external software to schedule the creation of an additional set of VMs of a certain type. This solution could be used to automatically schedule the creation (and the following destruction) of VMs at peak hours.

The external software will have to implement the following functionalities:

- At initialization, the application should create two new flavors on the platform, one *standard* flavor and a *large* flavor with more resources
- At scheduled times, the application should trigger the creation of new VMs, based on an existing image
- At the end of the scheduled peak period, the application should destroy the additional VMs

In order to allow system administrator to schedule for the creation of new VMs, the application must expose a REST interface to schedule the peak period and select the flavor and the image for the VMs to be created.

References:

- OpenStack SDK examples: https://github.com/openstack/openstacksdk/tree/master/examples

Option #3 - Cloud Application Project

Design and develop a multi-tier cloud computing application based on containers. The application should mimic a certain application implementing some simple functionalities. The cloud computing application will need the following components:

- A load balancer layer to load-balance ingress traffic implemented using the software HAproxy
- A frontend layer exposing a REST interface that receives and dispatches requests
- A backend layer that processes the requests coming from the frontend
- A database layer that stores the data

The user can interact with the application through the REST plugin installed on the browser.

The user will be able to perform the retrieval of some data. The backend must process the requests from users by interacting with the database.

In order to ensure scalability a message-based queue system must be used for the communication between the frontend and the backend.

Configuration parameters shared across all the components of the application should be stored in a Zookeeper instance running in a container.

References:

- HAproxy docker image: https://hub.docker.com/_/haproxy
- HAproxy configuration: https://dzone.com/articles/how-to-configure-ha-proxy-as-a-proxy-and-loadbalan
- Zookeeper container: https://hub.docker.com/_/zookeeper
- Python Kazoo library to interact with Zookeeper: https://kazoo.readthedocs.io/en/latest/basic_usage.html#creating-nodes

Workbook

Gnocchi

Gnocchi installation can be performed in a similar manner as the other OpenStack services:

Install the components. On the architecture manager:

```
juju deploy --to lxd:0 gnocchi

juju deploy --to lxd:0 memcached


juju add-relation gnocchi mysql

juju add-relation gnocchi memcached

juju add-relation gnocchi keystone

juju add-relation gnocchi ceph-mon
```

Install the OpenStack command line tools required to retrieve a token to be used to interact with Gnocchi. On the controller:

```
snap install --classic openstackclients
```

Gnocchi exposes a REST interface to create metrics, push the data and retrieve the data. Gnocchi REST interface requires a token that can be retrieved through the following command:

```
source admin.sh

openstack token issue
```

The REST interface exposed by Gnocchi is available on the IP address of the container in which Gnocchi runs (it can be retrieved from juju status).

Some operations:

Retrieve Gnocchi current status

```
curl -H "X-AUTH-TOKEN: AUTH_TOKEN" http://252.3.26.90:8041/v1/status?details=False
```

Create a new metric

```
curl -d "{\"archive_policy_name\": \"name_metric\"}" -X POST -H "Content-Type: application/json" -H "X-AUTH-TOKEN: AUTH_TOKEN" http://252.3.26.90:8041/v1/metric
```

Retrieve the status of a metric and its ID

```
curl -H "X-AUTH-TOKEN: AUTH_TOKEN" http://252.3.26.90:8041/v1/metric
```

Push some measurements

```
curl -d "[ { \"timestamp\": \"2014-10-06T14:33:57\", \"value\": 43.1 }] " -X POST -H "Content-Type: application/json" -H "X-AUTH-TOKEN: AUTH_TOKEN" http://252.3.26.90:8041/v1/metric/ID_METRIC/measures
```

Read measurements

```
curl -H "X-AUTH-TOKEN: AUTH_TOKEN" http://252.3.26.90:8041/v1/metric/ID_METRIC/measures
```

Python SDK

See the last slides in the OpenStack operations LAB.

HAProxy

Example of a haproxy.cfg to load balance across two different servers:

```
global
 maxconn 256
 debug
 log /dev/log local0 debug

defaults
 mode http
 timeout connect 5000ms
 timeout client 50000ms
 timeout server 50000ms
 log global

listen stats
 bind *:9999
 stats enable
 stats hide-version
 stats uri /stats
 stats auth admin:admin@123

frontend myApp
 bind *:80
 default_backend myAppBackEnd


backend myAppBackEnd

 balance roundrobin

 mode http

 server myAppServer1 172.17.0.3:8080 check

 server myAppServer2 172.17.0.2:8080 check
```

Zookeeper

Some code to connect to a zookeeper node, store a data and retrieve it:

```
zk = KazooClient(hosts='172.17.0.5:2181', read_only=True)

zk.start()

#Store the data

zk.ensure_path("/my/favorite")

zk.create("/my/favorite/node", b"a value")

#Retrieve the data

data, stat = zk.get("/my/favorite/node")

print("Version: %s, data: %s" % (stat.version, data.decode("utf-8")))
```