Container Orchestration Engines

Container Orchestration support in the Rapid Access Cloud is new and should be considered a Beta feature. As always with the Rapid Access Cloud, ensure you have a backup of any important data.

- Introduction
- Supported COEs
    - Available Templates
- Creating a Swarm Cluster
    - On the Rapid Access Cloud Dashboard
    - On the Command-Line
- Creating a Kubernetes Cluster
    - On the Rapid Access Cloud Dashboard
    - On the Command-Line
- Security Groups
- Using Swarm

Introduction

Container Orchestration Engines (COEs) assist with deploying and managing containers. Popular COEs include Docker Swarm, Kubernetes, and Mesos. The Rapid Access Cloud can assist with deploying COE infrastructure by way of the OpenStack Magnum project.

Supported COEs

The Rapid Access Cloud supports Docker Swarm and Kubernetes.

Available Templates

At this time, we offer the following templates:

Name	Flavor Used	IPv6 Accessible	Floating IP Accessible
swarm-floating-medium	m1.medium	Yes	Yes
swarm-ipv6-medium	m1.medium	Yes	No
swarm-floating-large	m1.large	Yes	Yes
swarm-ipv6-large	m1.large	Yes	No
swarm-floating-xlarge	m1.xlarge	Yes	Yes
swarm-ipv6-xlarge	m1.xlarge	Yes	No
k8s-floating-medium	m1.medium	Yes	Yes
k8s-ipv6-medium	m1.medium	Yes	No
k8s-floating-large	m1.large	Yes	Yes
k8s-ipv6-large	m1.large	Yes	No
k8s-floating-xlarge	m1.xlarge	Yes	Yes
k8s-ipv6-xlarge	m1.xlarge	Yes	No

Creating a Swarm Cluster

Install Docker if it is not already installed.

On the Rapid Access Cloud Dashboard

Coming Soon.

On the Command-Line

Make sure you have the OpenStack command-line tools installed.

- Install OpenStack command-line tools

Next, install the `python-magnumclient` package:

```
# On Linux
$ sudo pip install python-magnumclient


# On Mac
$ pip install --user python-magnumclient
```

Next, choose a Cluster Template. You can view the available templates by doing:

```
$ source /path/to/your/rc/file
$ openstack coe cluster template list

+--------------------------------------+----------------------+
| uuid                                 | name                 |
+--------------------------------------+----------------------+
| 22484703-51e3-468d-b829-64fe19fec7b6 | swarm-floating-medium |
| 416e127b-8377-4d17-bffb-049dd92bba39 | swarm-floating-large  |
| 82151b50-53d6-4a6a-8cb8-7adcf3bd7353 | swarm-floating-xlarge |
| ace2c106-b14d-4cb6-9138-9533fab03453 | swarm-ipv6-medium     |
| 438ed95c-2197-4ea5-b5db-26f2c9d50cea | swarm-ipv6-large      |
| cb442f1d-9ffd-471c-b158-bc2320b0fccb | swarm-ipv6-xlarge     |
+--------------------------------------+----------------------+
```

Next, create a cluster:

```
$ openstack coe cluster create swarm-cluster --cluster-template swarm-floating-medium --master-count 1 --node-
count 3 --keypair mykey --docker-volume-size=10
Request to create cluster a5e0d117-d2cd-4520-8b92-ddcf103ceefb accepted
```

There are a few things to note about this command:

- The above command will create a cluster of 4 total nodes: 1 master and 3 workers. All will be `m1.medium` instances.
- Clusters only support a single master at this time, so you always need to use `--master-count 1`.
- `--keypair` must be an existing Key Pair.
- `--docker-volume-size` is required. The example above will have 4 volumes of 10 gigabytes created. One volume will be attached to each node of your cluster.

You must make sure you have the appropriate quota available to create a cluster. In the above example, you would need to be able to create the following:

- 1 x Floating IP
- 4 x m1.medium instances
- 4 x volumes
- 40gb block storage

The above possible to do with the Rapid Access Cloud's default quota and no existing resources running.

You can watch the status of the cluster creation by taking the printed UUID and doing:

```
$ openstack coe cluster show a5e0d117-d2cd-4520-8b92-ddcf103ceefb


+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| status              | CREATE_IN_PROGRESS                   |
| cluster_template_id | 22484703-51e3-468d-b829-64fe19fec7b6 |
| node_addresses      | []                                   |
| uuid                | a5e0d117-d2cd-4520-8b92-ddcf103ceefb |
| stack_id            | 6ff6833e-04ef-4e59-a12f-9a41936c1304 |
| status_reason       | None                                 |
| created_at          | 2018-07-06T15:55:18+00:00            |
| updated_at          | 2018-07-06T15:55:34+00:00            |
| coe_version         | None                                 |
| labels              | {}                                   |
| faults              |                                      |
| keypair             | cybera                               |
| api_address         | None                                 |
| master_addresses    | []                                   |
| create_timeout      | 60                                   |
| node_count          | 3                                    |
| discovery_url       | None                                 |
| master_count        | 1                                    |
| container_version   | None                                 |
| name                | swarm-cluster                        |
| master_flavor_id    | m1.medium                            |
| flavor_id           | m1.medium                            |
+---------------------+--------------------------------------+
```


Wait until the `status` reads CREATE_COMPLETE before proceeding to the next step.

Next, download the COE authentication information:

```
$ mkdir swarm-cluster
$ openstack coe cluster config --dir swarm-cluster --output-certs swarm-cluster
```

The above command is a little magical in that it has done the following things:

1. Added your Swarm cluster's SSL certificates to the `swarm-cluster` directory.

   ```
   $ ls swarm-cluster/
   ca.pem   cert.pem key.pem
   ```

2. Set some environment variables for you:

   ```
   $ env | grep DOCKER
   DOCKER_HOST=tcp://10.1.2.184:2375
   DOCKER_TLS_VERIFY=True
   DOCKER_CERT_PATH=/Users/jtopjian/clusters/swarm-cluster
   ```

Above, you can see that `DOCKER_HOST` is set to a Private IP address. Private IPs are not accessible unless you use the RAC VPN. Alternatively, you can change `DOCKER_HOST` to the Floating IP or IPv6 address of your master node. To do this, do the following:

```
$ openstack server list -c Name -c Networks
+-----------------------------------------
+-------------------------------------------------------------------------+
| Name                                    |
Networks                                                                  |
+-----------------------------------------
+-------------------------------------------------------------------------+
| swarm-cluster-aq5dpyxwpte5-node-1          | default=2605:fd00:4:1000:f816:3eff:feb5:293d,
10.1.2.187                  |
| swarm-cluster-aq5dpyxwpte5-node-0          | default=2605:fd00:4:1000:f816:3eff:fe09:ca6a,
10.1.2.188                  |
| swarm-cluster-aq5dpyxwpte5-node-2          | default=2605:fd00:4:1000:f816:3eff:fe15:8882,
10.1.2.185                  |
| swarm-cluster-aq5dpyxwpte5-primary-master-0 | default=2605:fd00:4:1000:f816:3eff:fe2a:b323, 10.1.2.184,
162.246.156.5   |
```

Note either the Floating IP (162.246.156.5 in the example above) or the IPv6 address of the Master node. Then do:

```
$ export DOCKER_HOST=tcp://162.246.156.5:2375
```

After that, you now have access to a fully functional Docker Swarm cluster:

```
$ docker node ls
ID                           HOSTNAME                                                STATUS
AVAILABILITY        MANAGER STATUS      ENGINE VERSION
jqm3pmupwxlcdy4bodp3ak7n2     swarm-cluster-aq5dpyxwpte5-node-0.novalocal             Ready
Active                               1.13.1
qseo32fewgiprqqxmf9otg7mb     swarm-cluster-aq5dpyxwpte5-node-1.novalocal             Ready
Active                               1.13.1
vegmr4raoiyq4j5upkvy9o6pi     swarm-cluster-aq5dpyxwpte5-node-2.novalocal             Ready
Active                               1.13.1
zpgho4soqk3qk74dbrny3r4es *   swarm-cluster-aq5dpyxwpte5-primary-master-0.novalocal   Ready
Active              Leader           1.13.1


$ docker run hello-world
Unable to find image 'hello-world:latest' locally
Trying to pull repository docker.io/library/hello-world ...
sha256:3e1764d0f546ceac4565547df2ac4907fe46f007ea229fd7ef2718514bcec35d: Pulling from docker.io/library/hello-
world
9bb5a5d4561a: Pull complete
Digest: sha256:3e1764d0f546ceac4565547df2ac4907fe46f007ea229fd7ef2718514bcec35d
Status: Downloaded newer image for docker.io/hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.


$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
a375d7b710d8        hello-world         "/hello"            25 seconds ago      Exited (0) 23 seconds
ago                       confident_kirch
```

Creating a Kubernetes Cluster

First, install kubectl if it isn't already installed.

On the Rapid Access Cloud Dashboard

Coming Soon.

On the Command-Line

Make sure you have the OpenStack command-line tools installed.

- Install OpenStack command-line tools

Next, install the `python-magnumclient` package:

```
# On Linux
$ sudo pip install python-magnumclient


# On Mac
$ pip install --user python-magnumclient
```

Next, choose a Cluster Template. You can view the available templates by doing:

```
$ source /path/to/your/rc/file
$ openstack coe cluster template list

+--------------------------------------+-----------------------+
| uuid                                 | name                  |
+--------------------------------------+-----------------------+
| e27d8d86-b051-4a8d-98cf-b8cd6afa7df4 | k8s-ipv6-medium       |
| 258b877b-8256-4612-ba2c-1cde512570c8 | k8s-ipv6-large        |
| d397c572-b93b-47ed-8ea0-d824ea90cea9 | k8s-ipv6-xlarge       |
| 6b1af9f0-c073-464b-9552-eb233ab993b2 | k8s-floating-medium   |
| 1e1d58d7-e6b1-4922-a44b-790cc2cc5b66 | k8s-floating-large    |
| fd52ee79-c7a7-43d9-b3a6-fe4080ad7143 | k8s-floating-xlarge   |
+--------------------------------------+-----------------------+
```

Next, create a cluster:

```
$ openstack coe cluster create kubernetes-cluster --cluster-template k8s-floating-medium --master-count 1 --
node-count 3 --keypair mykey --docker-volume-size=10
Request to create cluster 54b4d4e5-1952-415b-b2ac-af0cfcdb9af2 accepted
```

There are a few things to note about this command:

- The above command will create a cluster of 4 total nodes: 1 master and 3 workers. All will be `m1.medium` instances.
- Clusters only support a single master at this time, so you always need to use `--master-count 1`.
- `--keypair` must be an existing Key Pair.
- `--docker-volume-size` is required. The example above will have 4 volumes of 10 gigabytes created. One volume will be attached to each node of your cluster.

You must make sure you have the appropriate quota available to create a cluster. In the above example, you would need to be able to create the following:

- 1 x Floating IP
- 4 x m1.medium instances
- 4 x volumes
- 40gb block storage

The above possible to do with the Rapid Access Cloud's default quota and no existing resources running.

You can watch the status of the cluster creation by taking the printed UUID and doing:

```
$ openstack coe cluster show 54b4d4e5-1952-415b-b2ac-af0cfcdb9af2

+---------------------+------------------------------------------------------------+
| Field               | Value                                                      |
+---------------------+------------------------------------------------------------+
| status              | CREATE_IN_PROGRESS                                         |
| cluster_template_id | 6b1af9f0-c073-464b-9552-eb233ab993b2                       |
| node_addresses      | []                                                         |
| uuid                | 54b4d4e5-1952-415b-b2ac-af0cfcdb9af2                       |
| stack_id            | dae1f0c4-6118-4ef2-9aab-80a5b6958ef6                       |
| status_reason       | None                                                       |
| created_at          | 2018-07-11T05:56:38+00:00                                  |
| updated_at          | 2018-07-11T05:56:54+00:00                                  |
| coe_version         | None                                                       |
| labels              | {u'cert_manager_api': u'true'}                             |
| faults              |                                                            |
| keypair             | cybera                                                     |
| api_address         | None                                                       |
| master_addresses    | []                                                         |
| create_timeout      | 60                                                         |
| node_count          | 2                                                          |
| discovery_url       | https://discovery.etcd.io/56688682e6f32db3f45bfc6b2b82d1a1 |
| master_count        | 1                                                          |
| container_version   | None                                                       |
| name                | kubernetes-cluster                                         |
| master_flavor_id    | m1.medium                                                  |
| flavor_id           | m1.medium                                                  |
+---------------------+------------------------------------------------------------+
```

⚠ Wait until the `status` reads CREATE_COMPLETE before proceeding to the next step.

Next, download the COE authentication information:

```
$ mkdir kubernetes-cluster
$ openstack coe cluster config --dir kubernetes-cluster --output-certs kubernetes-cluster
```

The above command will generate a Kubernetes configuration file and install the SSL certificates for your cluster to the `kubernetes-cluster` directory.

By default, the configuration file is set to communicate with the Kubernetes cluster by its private IP address. Private IPs are not accessible unless you use the RAC VPN. Alternatively, you can change the configuration file to use the Floating IP or IPv6 address of your master node. To do this, do the following:

```
$ openstack server list -c Name -c Networks
+-----------------------------------------
+-------------------------------------------------------------------------+
| Name                                    |
Networks                                                                  |
+-----------------------------------------
+-------------------------------------------------------------------------+
| kubernetes-cluster-brs4edkzdppp-minion-1    | default=2605:fd00:4:1000:f816:3eff:fe8f:a992,
10.1.2.251                  |
| kubernetes-cluster-brs4edkzdppp-minion-0    | default=2605:fd00:4:1000:f816:3eff:fe2e:546d,
10.1.2.250                  |
| kubernetes-cluster-brs4edkzdppp-master-0    | default=2605:fd00:4:1000:f816:3eff:fe46:f51c, 10.1.2.249,
162.246.156.70   |
```

Note either the Floating IP (162.246.156.5 in the example above) or the IPv6 address of the Master node. Then edit the configuration file found under `kubernetes-cluster/config` and find the line that starts with `server`. Change this line to read:

```
server: https://162.246.156.70:6443
```

After that, you now have access to a fully functional Kubernetes cluster:

```
$ kubectl -n kube-system get pods
NAME                                  READY     STATUS    RESTARTS   AGE
coredns-5864cfd79d-86wgm              1/1       Running   0          12m
heapster-68b976dd7-fkckb              1/1       Running   0          12m
kubernetes-dashboard-846b8b6844-5p54j   1/1       Running   0          12m


$ kubectl run nginx --image=nginx --replicas=5
deployment.apps/nginx created


$ kubectl get pods
NAME                    READY     STATUS    RESTARTS   AGE
nginx-8586cf59-2g4bp    1/1       Running   0          30s
nginx-8586cf59-cw4x2    1/1       Running   0          30s
nginx-8586cf59-gnz6m    1/1       Running   0          30s
nginx-8586cf59-hr7x4    1/1       Running   0          30s
nginx-8586cf59-mhcv5    1/1       Running   0          30s
```

Security Groups

By default, your COE cluster has a very strict security group configured. This is to prevent security incident mishaps such as accidentally publishing ElasticSearch, MongoDB, or similar services. When you want to make a service publicly accessible, edit your cluster's security group. You can see the group by doing the following:

```
$ openstack security group list | grep cluster

| c22cd9ba-6098-42b9-8a36-ff00b48924b4 | swarm-cluster-aq5dpyxwpte5-secgroup_swarm_manager-rzqnqkkw4mfd |
| ee5d6145-373e-42bf-9b5d-57372f8f20a3 | swarm-cluster-aq5dpyxwpte5-secgroup_swarm_node-iyghtvtxxg3l    |
```

And then adding a rule to the "manager" group using either the command-line or dashboard.

Using Swarm

General documentation on Docker Swarm can be found here: https://docs.docker.com/engine/swarm/

As a quick demo of Swarm's capabilities, let's deploy a Consul cluster.

This is only for demo purposes and should not be used in any production capacity

First, create a file called `consul.yaml` with the following contents:

```
version: "3"

networks:
  consul:
    driver: overlay

services:
  seed:
    image: consul:latest
    networks:
      - consul
    deploy:
      mode: global
      placement:
        constraints:
          - node.role == manager
    environment:
      - "CONSUL_LOCAL_CONFIG={\"disable_update_check\": true}"
      - "CONSUL_BIND_INTERFACE=eth0"
    entrypoint:
      - consul
      - agent
      - -server
      - -bootstrap-expect=3
      - -data-dir=/tmp/consuldata
      - -bind={{ GetInterfaceIP "eth0" }}

  node:
    image: consul:latest
    networks:
      - consul
    depends_on:
      - seed
    deploy:
      replicas: 3
      placement:
        constraints:
          - node.role != manager
    environment:
      - "CONSUL_LOCAL_CONFIG={\"disable_update_check\": true}"
      - "CONSUL_BIND_INTERFACE=eth0"
      - "CONSUL_HTTP_ADDR=0.0.0.0"
    entrypoint:
      - consul
      - agent
      - -server
      - -data-dir=/tmp/consuldata
      - -bind={{ GetInterfaceIP "eth0" }}
      - -client=0.0.0.0
      - -retry-join=seed:8301
      - -ui
    ports:
      - "8500:8500"
      - "8600:8600"
```

Next, deploy the cluster. This assumes you have set the appropriate environment variables described in the above "Creating a Swarm Cluster" instructions.

```
$ docker stack deploy -c consul.yaml consul

Creating network consul_consul
Creating service consul_node
Creating service consul_seed
```

You can see the status of the cluster by doing:

```
$ docker service list
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
1q2qkzitxqp6        consul_node         replicated          3/3                 consul:latest       *:8500-
>8500/tcp, *:8600->8600/tcp
oxuoch7c5nex        consul_seed         global              1/1                 consul:latest
```

Once all of the replicas are running, the cluster has finished building. Next, wait until the nodes have joined each other. You can view the status by reading the logs:

```
$ docker service logs consul_seed


...
2018/07/23 16:23:29 [INFO] consul: member '7b8eee98fcaf' joined, marking health alive
2018/07/23 16:23:29 [INFO] consul: member '434de12113d3' joined, marking health alive
2018/07/23 16:23:29 [INFO] consul: member '70a8bf94d84a' joined, marking health alive
...
```

Finally, try using the Consul cluster. You need to make sure you add Port 8500 to your security group as mentioned in the "Security Group" section above.

```
$ curl -X PUT -d "Hello World!" http://<floating ip or ipv6>:8500/v1/kv/hello
true


$ curl http://<floating ip or ipv6>:8500/v1/kv/hello
[{"LockIndex":0,"Key":"hello","Flags":0,"Value":"SGVsbG8gV29ybGQh","CreateIndex":43,"ModifyIndex":43}]


$ curl http://<floating ip or ipv6>:8500/v1/kv/hello?raw
Hello World!
```

When finished, you can easily tear down the cluster:

```
$ docker stack rm consul


Removing service consul_node
Removing service consul_seed
Removing network consul_consul
```