MICADO V2

This tutorial will show how to deploy application to MICADO V2. With the help of MICADO, applications can be auto-scaled without user intervention in a predefined scaling ranges to guarantee that the application always runs at the optimum level of resources. As a user you may not find major differences in this version compared to the previous version of MICADO, but in the background we made some major modification. We implemented container monitoring, and with the help of the collected information, now the deployed application can be scaled in the container level also, which means changing the number of containers of the application and scaling the virtual machines only if there is no resource left on the host machines. This gives us faster feedback in the control loop and with container level scaling we can fit the demand curve better, in real time. You will also find out in this tutorial that deploying multiple applications into the same infrastructure have improved as well and you can limit resource usage of your applications. You don’t have to use Docker’s global mode services any more. You will also find application specific Alerts in Prometheus which will be generated automatically when you start you application and which will be deleted when you delete the application.

The tutorial builds a scalable architecture framework with the help of Occopus and performs the automatic scaling of the application based on Occopus, Docker Swarm and Prometheus (a monitoring tool). The scalable architecture framework can be seen in Figure 1.

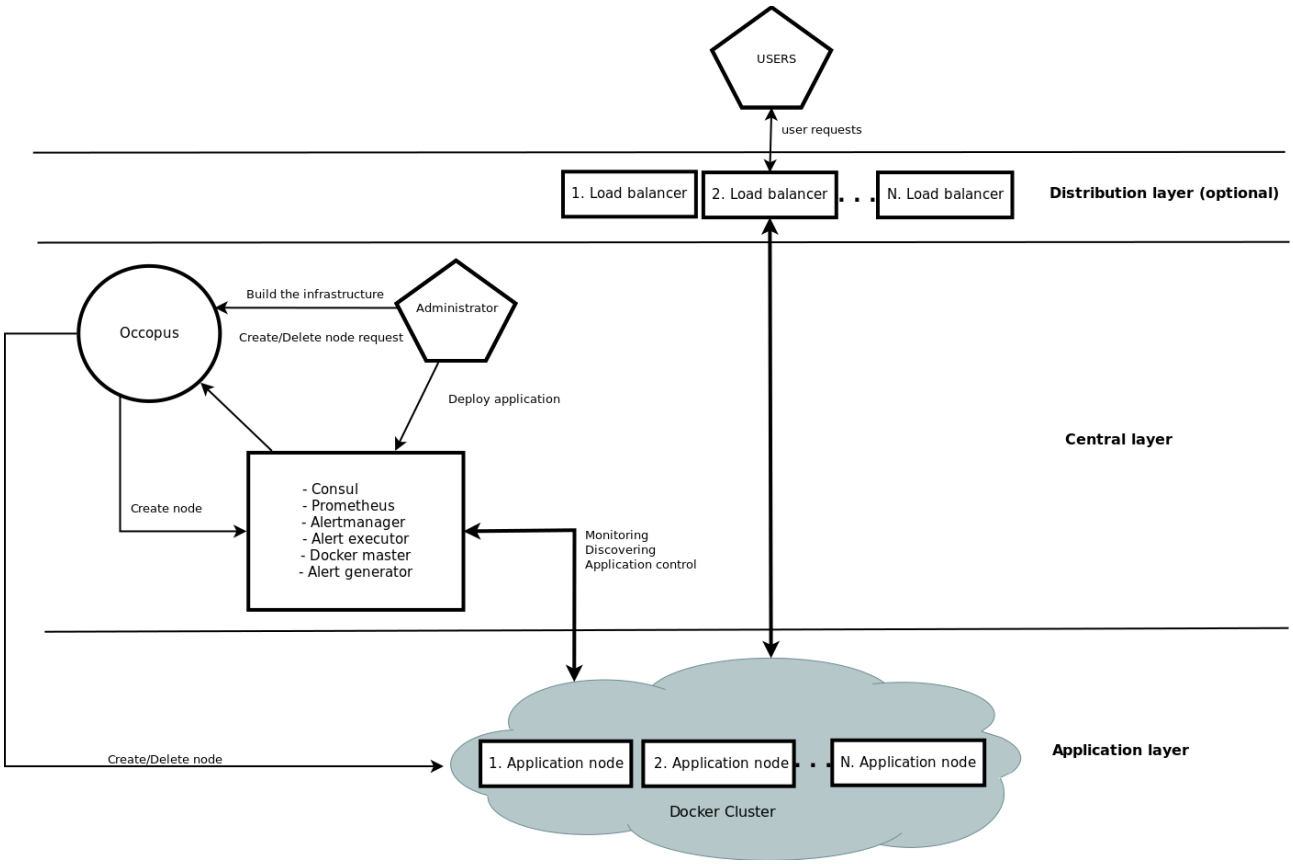

Figure 1. Docker based auto-scaling infrastructure with a separate load balancing layer

The scalable architecture framework consists of the following services:

1. Cloud orchestrator and manager: Occopus
2. Application node: Data Avenue (DA)
3. Service discovery: Consul
4. Load balancer: Haproxy or Docker Swarm
5. Monitoring tool: Prometheus

In this infrastructure you will see how can you create a docker service in Docker Swarm. This service will be the user application. You can also deploy multiple application to the same infrastructure later if you wish or delete them. In this infrastructure nodes are discovered by Consul, which is a service discovery tool also providing DNS service and are monitored by Prometheus, a monitoring software.

In this auto-scaling example we implemented a multi-layered traditional load-balancing schema. On the upper layer, there are load balancer nodes organised into a cluster and prepared for scaling. The load balancer cluster is handling the load generated by secured http transfer (http(s)) between the client and the underlying application. Also we use Swarm’s inner routing mesh and if you

are using computational heavy applications instead of user heavy applications you are advised to use MICADO without the scalable load balancing layer which will decrees the number of VMs for you infrastructure. The application is also organised inside a scalable cluster to distribute the load generated by serving the client requests. In this demonstration architecture, the Data Avenue (DA) service was selected to be the concrete application. Notice that other applications can easily replace the DA service. The DA service here implements data transfer between the client and a remote storage using various protocols (http, sftp, s3, ...). For further details about the software, please visit the Data Avenue website. Finally, in the lowest layer there is a Database node (not shown in Figure 1) required by the instances of Data Avenue to store and retrieve information (authentication, statistics) for their operation. If you use your own application feel free to delete the database later on.

The monitor service Prometheus collects runtime information about virtual machines and also about the running containers on these machines. The VMs are connected to Docker Swarm. When an application is overloaded, Prometheus instructs Swarm to increment the number of containers for that application. When there is no more resource available in the docker cluster to create new containers, Prometheus calls Occopus to scale up, and create a new virtual machines. The docker based applications can have unlimited resources of the host machines or you can limit the available resource for the containers, which is advised. This will ensure that different applications can work next to each other. If the application service is under loaded Prometheus instructs first Swarm to decrease the number of containers of that application and if one of the hosts in the cluster gets under loaded too, it will call Occopus to scale down the number of the host machines. If you which to use the scalable load balancing layer shown on Fig. 1., the same scaling event will be applied to them by Occopus.

Advantages of Docker

- encapsulation

- shorter node description files

- easier maintenance

- OS independent

The biggest advantage of Docker what we will see, is that changing user application is easy. Users don’t have to modify the node description files at all. After logging in the virtual machine which is running Docker Swarm, you can start your own Docker based application in one command.

Features

- using Prometheus to monitor nodes and containers.
- generate application specific alerting rules automatically.
- using load balancers to share system load between application nodes.
- using Consul as a DNS service discovery agent.
- using a docker cluster that is running the applications

- use an alert executor that communicates with Swarm and Occopus to scale running applications and host machines.

Prerequisites

- accessing a cloud through an Occopus-compatible interface (e.g. EC2, OCCI, Nova, etc.)
- target cloud contains a base 16.04 ubuntu OS image with cloud-init support (image id, instance type) for the App, Central and LB node type, and for 'db' node use 14.04 ubuntu OS image with cloud-init support.
- start Occopus in Rest-API mode (Occopus-rest-service)

Download

Before proceed further, please decide if you need a separate load balancing layer or not. We provide you two options to choose from. Either you can create the MICADO infrastructure with a separate load-balancing layer or without it as you can see of Fig. 1. At the moment the separate load-balancing based version (B) support only one application/infrastructure! If you are planning to deploy more applications, please don't you version (B). Also, if you are not sure, choose "A" instead, which will not implement it.

A. https://github.com/rabotond/MICADO/raw/master/MICADOV2A/MICADOV2A.tar.gz

B. https://github.com/rabotond/MICADO/raw/master/MICADOV2B/MICADOV2B.tar.gz

Steps

Open the file `nodes/node_definitions.yaml` and edit the resource section of the nodes labelled by `node_def`:

- you must select an Occopus compatible resource plugin
- you can find and specify the relevant: list of attributes for the plugin
- you may follow the help on collecting the values of the attributes for the plugin
- you may find a resource template for the plugin in the resource plugin tutorials

The downloadable package for this example contains a resource template for the EC2 plugin.

Edit the: infrastructure_descriptor.yaml infrastructure descriptor file. Set the following attributes:

- scaling is the interval in which the number of nodes can change (min, max).

```
- &APP_cluster # Node Running your application
   name: app
   type: app
   scaling:
      min: 1
      max: 10
```

Important

Keep in mind that Occopus has to start at least one node from each node type to work properly! Also Occopus will ensure that one instance will always run from the node types. Auto scaling events (scale up, scale down) are based on Prometheus rules which act as thresholds, let's say scale up if cpu usage > 80%. In this example you can see the implementation of a cpu utilization in your app-lb cluster with some threshold values. If you are not happy with the current threshold values, feel free to change them in the /etc/prometheus/prometheus.rules file.

Edit the "variables" section of the infrastructure_descriptor.yaml file. Set the following attributes:

- Occopus_restservice_ip is the ip address of the host where you will start the Occopus-rest-service
- Occopus_restservice_port is the port you will bind the Occopus-rest-service to

```
Occopus_restservice_ip: "127.0.0.1"
Occopus_restservice_port: "5000"
```

- applicationport: Only edit this field if you chose to use the separate load-balancing layer, version "B". It will tell Haproxy on which port you application is running. We set it to 8080 since our example application will listen on this port.

Components in the infrastructure connect to each other, therefore several port ranges must be opened for the VMs executing the components. Clouds implement port opening various way (e.g. security groups for OpenStack, etc.). Make sure you implement port opening in your cloud for the following port ranges:

```
TCP 22 (SSH)
TCP 80 (HTTP)
TCP 443 (HTTPS)
TCP 8300 (Consul) TCP Server RPC. This is used by servers to handle incoming requests from other agents.
TCP and UDP 8301 (Consul) This is used to handle gossip in the LAN. Required by all agents.
TCP and UDP 8302 (Consul) This is used by servers to gossip over the WAN to other servers.
TCP 8400 (Consul) CLI RPC. This is used by all agents to handle RPC from the CLI.
```

```
TCP 8500 (Consul) HTTP API. This is used by clients to talk to the HTTP API.
TCP and UDP 8600 (Consul) DNS Interface. Used to resolve DNS queries.
TCP 9090 (Prometheus)
TCP 8080 (Data Avenue)
TCP 9093 (Alertmanager)

TCP 9095 (Alert executor)
```

Make sure your authentication information is set correctly in your authentication file. You must set your authentication data for the resource you would like to use. Setting authentication information is described here.

Load the node definitions into the database.

Important

Occopus takes node definitions from its database when builds up the infrastructure, so importing is necessary whenever the node definition or any imported (e.g. contextualisation) file changes!

```
Occopus-import nodes/node_definitions.yaml
```

Start Occopus in REST service mode:

```
Occopus-rest-service --host [Occopus_restservice_ip] --port [Occopus_restservice_port]
```

Use ip and port values as defined in the infrastructure description! Alternatively, use 0.0.0.0 for the host IP.

Start deploying the infrastructure through the Occopus service:

```
curl -X POST http://[Occopus_restservice_ip]:[Occopus_restservice_port]/infrastructures/ --data-binary @infrastructure_descriptor.yaml
```

Start the application.

The infrastructure itself won't start any application. To start the example DA application or implement your own application follow these steps. First you have to SSH into the VM that runs Docker Swarm which node called Central.
After you logged in, you have to start the application as a Docker service. If you don't specify the "--replicas" tag it will allocate 1 container automatically which should be fine for you. We also advice you to use cpu limitation "--limit-cpu" if you are planning to deploy multiple application. In case of one application, you shouldn't bother with it. To start the service type the following command. Look for the place holder!

To start the example Data avenue application (requires sudo privileges!):

```
docker service create --name dataavenue --limit-cpu=0.6 --publish 80:8080 -p 8080:8080 micado/dataavenue "database IP address."
```

--name: gives a name to the service which you will find in all the nodes in the Swarm cluster

– mode global: runs the Docker image in every node in the Docker cluster

– publish port: routes http requests to the applications

– “database IP address”: points the example DA application containers to the database node

– -p: expose container ports to host ports

– micado/dataavenue: name of the image

– --limit-cpu: a value between 0-1 that limits the cpu usage of the containers.

Note: Deploying the DA application takes around 2 minutes. You will be able to reach the application on any of the deployed virtual machine IP address/blacktop3, like: http://”one of you hosts IP address”/blacktop3

To start any other application (requires sudo privileges!):

```
Docker service create --name [name_of_the_application] – limit-cpu=”value” --publish [port_number_where_your_application_listen]:[port_on_which_you_want_to_reach_it] -p [host_port]:[container_port] [docker_image]
```

What you have to keep in mind that Docker containers have a separate network, and you should make sure that you expose the ports on which your application is listening to the virtual machine port. Otherwise you won’t be able to reach your running application.

Note: To expose container ports use the “-p [host port]:[container port]” as in the above command

After you started your service, Docker Swarm will share the load in the Swarm cluster by creating a routing mash. It is created automatically, when you specified the –publish argument. Your applications will be reachable on all of your hosts machines interfaces, and on the port number you published.

Note: You can find more info about Docker’s built in load balancer in the following link: https://docs.Docker.com/engine/swarm/ingress/

Note: If you are using the separate load-balancing layer based version, also don’t forget to publish you application since Haproxy will proxy to this port number.

To query the running services and the available nodes in the Swarm cluster run (requires sudo privileges!):

```
docker service list

docker node ls
```

You can scale up manually the application nodes through the Occopus REST interface and after a few minutes you can observe that the newly connected nodes will be automatically removed because the underloaded alert will fire. You can also check the status of your alerts during the testing at http://[Central node IP address]:9090/alerts

```
curl -X POST http://[Occopus_restservice_ip]:[Occopus_restservice_port]/infrastructures/[infrastructure_id]/scaleup/app
```

Important

Depending on the cloud you are using for you virtual machines it can take a few minutes to start a new node and connect it to your infrastructure. The connected nodes are present on prometheus's targets page.

To test the scaling mechanisms with the example DA application put some load on the application nodes with the command below. Just select you central node (or LB node if you have) IP address. The incoming user requests will be shared evenly between the application nodes. Start around 10 copy of the command to generate enough load. After a few minutes the cluster will be overloaded, and the overloaded alerts will fire in Prometheus for the DA application. Swarm will create a new container first, if there is enough resource to do so, and if there is no available resource, a new host machine will be started and connected to your cluster. Also, if you stop sending files for a while, the under loaded alert will fire in Prometheus and one (or more) of the application/load balancer nodes will be shut (scaled) down while Swarm will also decrease the number of containers as well.

To query the nodes and their IP addresses, use this command:

```
curl -X GET http://[Occopus_restservice_ip]:[Occopus_restservice_port]/infrastructures/[infrastructure_id]
```

Once, you have the IP of the selected node, generate load by transferring a 1GB file using the command below. Do not forget to update the placeholder in the command bellow!

```
curl -k -o /dev/null -H "X-Key: 1a7e159a-ffd8-49c8-8b40-549870c70e73" -H "X-URI:https://autoscale.s3.lpds.sztaki.hu/files_for_autoscale/1GB.dat" http://[ central node or LB node IP address]/blacktop3/rest/file
```

To check the status of alerts under Prometheus during the testing, keep watching the following url in your browser:

```
http://[central node ip]:9090/alerts
```

To query the running applications and the available node in the cluster on the Central node by Swarm (requires root privileges!)

```
docker service ls

docker node ls
```

Delete applications

If you decided to delete on of you application of the example DA application, you can do it by logging in to the Central node, first check the name of that specific application and then delete it.

```
docker service ls

docker service rm “service name”
```

Finally, you may destroy the infrastructure using the infrastructure id.

```
curl -X DELETE http://[Occopus_restservice_ip]:[Occopus_restservice_port]/infrastructures/[infra id]
```

The following video will show how to build up, manage and scale the example application: https://youtu.be/DsNBLQTvQFY