MICADO V3 advanced

This page is intended to use for more advanced user. If you want to customize MICADO, for example run your own application or find out problems in your installation then here you can find more information.

# 1. Overview of the Components

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.

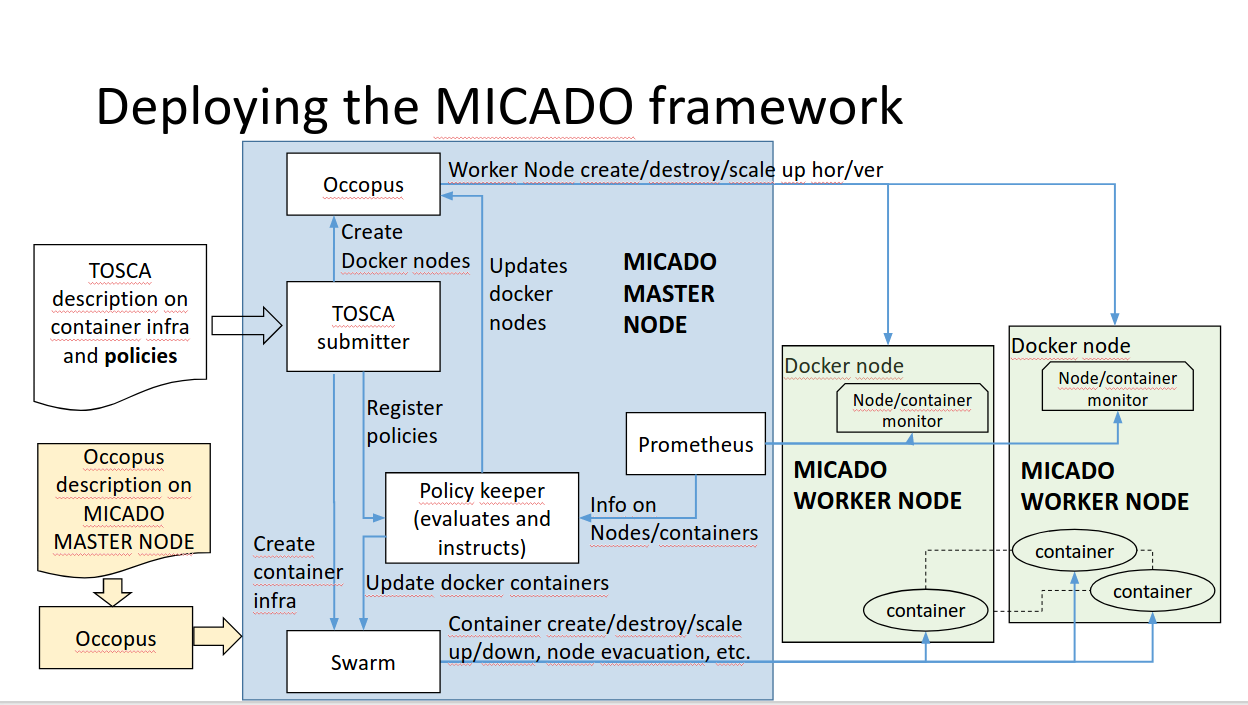


Figure 1. Docker based auto-scaling infrastructure with a separate load balancing layer

The scalable architecture framework consists of the following services:

1. Cloud orchestrator and manager: [Occopus](http://occopus.lpds.sztaki.hu/)
2. Service discovery: [Consul](https://www.consul.io/)
3. Docker engine: [Docker Swarm](https://docs.docker.com/engine/swarm/)
4. Monitoring tool: [Prometheus](https://prometheus.io/)

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 demonstration architecture, a stress testing application was selected. Notice that other applications can easily replace the example application.

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.

**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 extremely easy. Users don’t have to modify the node description files at all. After logging in to 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.
* use an alert executor that communicates with Swarm and Occopus to scale running applications and host machines.
* using a docker cluster that is running the applications

Prerequisites

* accessing a cloud through an Occopus-compatible interface (e.g. EC2, OCCI, Nova, cloudsigma, cloudbroker)
* target cloud contains a base 16.04 ubuntu LTS OS image with cloud-init support (image id, instance type)
* 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)

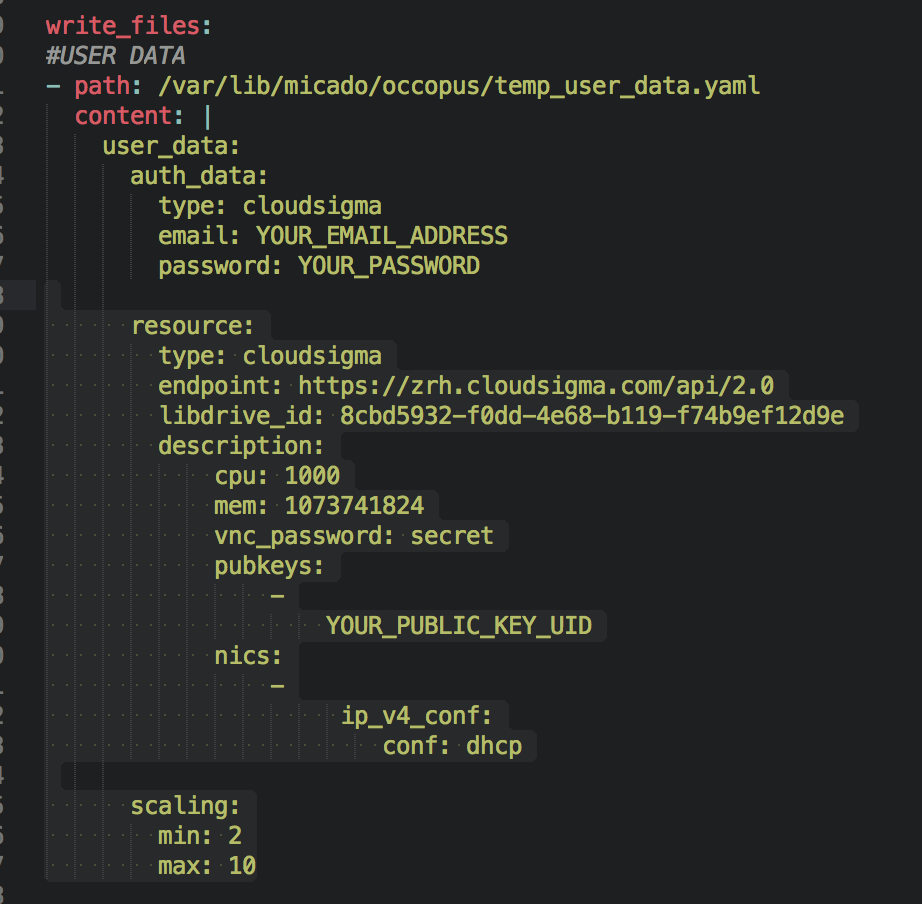
Download base project files

Download the MICADO input file from here: <https://goo.gl/ZhgvUm>

# 2. Deployment of MiCADO

## 2.1 Insert your user imputs

Now you have to modify the file that you downloaded, which is a script which will install MICADO on a cloud. In the beginning of the file you will see a section called “USER DATA” as shown in this picture:

 This file specifies the user credentials for the target cloud, the resource IDs that will be used for the Virtual machines and a scaling section which specifies the scaling ranges. We provided you an already filled out configuration for the CloudSigma cloud. The only thing you have to change is your email and password credentials the UID of your public key in the “pubkeys” section. (You can find or create your key on Cloudsigma by clicking on the left-hand side on “Access and Security” and then select “key management on the CloudSigma website. You will find the uuid of you key there.). When you are ready save the file and exit.

## 2.2 Change user data to deploy MICADO into another cloud

If you decide to use another cloud for the deployment you should:

1. Change the user\_data section, since some plugins call user credentials in other way (like username instead of email).
2. Change the resource section and find out the different resource ID-s on your target cloud.
3. Change the scaling ranges depending on which ranges you want to scale the number of virtual machiens.

**NOTE:** You can find already filled out resource descriptions and more information about user\_data on the Occopus webpage: <http://occopus.lpds.sztaki.hu/user-guide> under the “resource section” and under the “authentication” section.

## 2.3 Check the syntax

Before deploying MICADO we advise you to check the syntax of your file. Since it is a yaml formatted file you should make sure of the syntax. To do so just copy paste your MICADO file to an online yaml checker: <http://www.yamllint.com/>

## 2.4 Start MICADO

To start MICADO user the VM creation wizard on your target cloud.

* Choose a favour type
* An Ubuntu 16.04 LTS Image
* Attach your ssh key
* Paste the previously downloaded file to the cloud-init box and activate it. (sometimes its under “advanced setting”.
* If appropriate open ports or attach security groups.
* The click on “Create”

# 3. Deploy the Application

The infrastructure itself won’t start any application. To start the example stress testing application or implement your own application follow these steps. First you have to SSH into the MiCADO VM on your cloud.

After you logged in, you have to start the application as a Docker service. To start the service type the following command. Look for the place holder!

## 3.1 Deploy an example stress testing application

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

$ docker service create progrium/stress --cpu 2 --io 1 --vm 2 --vm-bytes 128M

--name: Optional. Gives a name to the service. Otherwise will get a random name.

-- publish port: Optional. Routes http requests to the applications, so you will reach it from outside.

progium/stress: name of the docker image

--limit-cpu: Optional. A value between 0-1 that limits the cpu usage of the containers.

## 3.2 To start your own application

$ 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.

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/>

# 4. Testing

This section is for testing purposes. If you experience some problems you can find out how to solve them here.

## 4.1 Test if the system is operational

On your browser enter the following URL:

http://ip\_address\_of\_MICADO\_VM:8500

You should see the web page of Consul. If you see every running service with a green box, you are good to go but if some of them are red, there are some problem. Also on the “nodes” tab you should see at least 3 nodes (MICADO + minimum number of scaling ranges you specified).

## 4.2 Test if scaling working properly

The stress testing application should automatically overload and stress down the worker cluster in a few minutes.

If everything went well in a few minutes you could see VMs booting on your cloud.

To check out the number of nodes after the scale up event in MICADO, check Prometheus on the following link:

<http://ip_address_of_MICADO_VM:9090/targets>

If you have more than 2, then it means that MICADO successfully scaled up the application nodes from 2.

Now we also would like to test if it scales down if there is no load on the cluster. To do so, stop delete the application and this way delete the load on the cluster After a few minutes, the number of nodes in the cluster should be go back to its minimum value (specified in the user\_data, scaling part).

## 4.3 Query running nodes and applications

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

$ docker service ls ( shows the number of application containers of the running applications)

$ docker node ls (gives back the nodes in connected to MICADO)

## 4.4 Check Prometheus alerts

You can also check the status of your alerts during the testing on Prometheus’s webpage. You should see two alerts always that scales the VMs (worker\*\_loaded) and depending on the number of deployed application more alerts that are automatically generated when you start an application. They are deleted at the moment when you delete your application service.

<http://ip_address_of_MICADO_VM:9090/alerts>

**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 (up to 5 minutes). The connected nodes are present on prometheus’s targets page.

# 5. Delete your infrastructure

When you finished with MICADO and wish to delete everything, you just need run the following command:

$ curl -X DELETE http://[micado\_master\_ip]:5000/infrastructures/micado\_worker\_infra

This will delete all the worker nodes. To delete MICADO itself you have to do it by hand, and delete it on your cloud.

Contributing to MICADO can be done on github. Just create a new pull request the one of the admins will merge you changes if everything is fine.

<https://github.com/UniversityOfWestminster/MICADO>

The following video will show how to build up, manage and scale the example application:

<https://youtu.be/DsNBLQTvQFY>