MiCADO Documentation

Attila Farkas

Jan 17, 2019

User Documentation

This software is developed by the COLA project.

CHAPTER 1

Introduction

MiCADO is an auto-scaling framework for Docker applications. It supports autoscaling at two levels. At virtual machine (VM) level, a built-in Docker Swarm cluster is dynamically extended or reduced by adding/removing cloud virtual machines. At docker service level, the number of replicas implementing a Docker Service can be increased/decreased.

MiCADO requires a TOSCA based Application Description to be submitted containing three sections: 1) the definition of the interconnected Docker services, 2) the specification of the virtual machine and 3) the implementation of scaling policy for both scaling levels. The format of the Application Description for MiCADO is detailed later.

To use MiCADO, first the MiCADO core services must be deployed on a virtual machine (called MiCADO Master) by an Ansible playbook. MiCADO Master contains Docker engine (configured as Swarm manager), Occopus (to scale VMs), Prometheus (for monitoring), Policy Keeper (to perform decision on scaling) and Submitter (to provide submission endpoint) microservices to realize the autoscaling control loops. During operation MiCADO workers (realised on new VMs) are instantiated on demand which deploy Prometheus Node Exporter, CAdvisor and Docker engine through contextualisation. The Docker engine of the newly instantiated MiCADO workers joins the Swarm manager on the MiCADO Master.

In the current release, the status of the system can be inspected through the following ways: REST API provides interface for submission, update and list functionalities over applications. Dashboard provides three graphical view to inspect the VMs and Docker services. They are Docker Visualizer, Grafana and Prometheus. Finally, advanced users may find the logs of the MiCADO core services useful on MiCADO master.

1.1 Deployment

As stated in the above section, to use MiCADO, you need to deploy the MiCADO services on a (separate) virtual machine, called MiCADO master. We recommend doing the installation remotely i.e. to download the Ansible playbook on your local machine and run the deployment on an empty virtual machine dedicated for this purpose on your preferred cloud.

1.1.1 Prerequisites

Git & Ansible 2.4 or greater are needed on your (local) machine to run the Ansible playbook.

The version of Ansible in the Ubuntu 16.04 APT repository is outdated and insufficient

Ansible

Install Ansible on Ubuntu 16.04.

```
sudo apt-get update
sudo apt-get install software-properties-common
sudo apt-add-repository ppa:ansible/ansible
sudo apt-get update
sudo apt-get install ansible
```

To install Ansible on other operation system follow the *official installation guide*.

1.1.2 Installation

Perform the following steps on your local machine.

Step 1: Download the ansible playbook.

Currently, MiCADO v5 version is available.

```
git clone https://github.com/micado-scale/ansible-micado.git ansible-micado
cd ansible-micado
git checkout v0.5.0
```

Step 2: Specify credential for instantiating MiCADO workers.

MiCADO master will use this credential to start/stop VM instances (MiCADO workers) to realize scaling. Credentials here should belong to the same cloud as where MiCADO master is running. We recommend making a copy of our predefined template and edit it. The ansible playbook expects the credential in a file, called credentials.yml. Please, do not modify the structure of the template!

```
cp sample-credentials.yml credentials.yml
vi credentials.yml
```

Edit credentials.yml to add cloud credentials. You will find predefined sections in the template for each cloud interface type MiCADO supports. Fill only the section belonging to your target cloud.

Step 3: (Optional) Specify details of your private Docker repository.

Set the Docker login credentials of your private Docker registries in which your personal containers are stored. We recommend making a copy of our predefined template and edit it. The ansible playbook expects the docker registry details in a file, called docker-cred.yml. Please, do not modify the structure of the template!

```
cp sample-docker-cred.yml docker-cred.yml
vi docker-cred.yml
```

Edit docker-cred.yml and add username, password, and repository url. To login to the default docker_hub, leave DOCKER_REPO as is (a blank string).

Step 4: Launch an empty cloud VM instance for MiCADO master.

This new VM will host the MiCADO master core services. Use any of aws, ec2, nova, etc command-line tools or web interface of your target cloud to launch a new VM. We recommend a VM with 2 cores, 4GB RAM, 20GB disk. Make sure you can ssh to it (password-free i.e. ssh public key is deployed) and your user is able to sudo (to install MiCADO as root). Store its IP address which will be referred as `IP` in the following steps. The following ports should be open on the virtual machine:

```
TCP: 22,2377,3000,4000,5000,5050,7946,8080,8300,8301,8302,8500,8600,9090,9093,12345
UDP: 4789,7946,8301,8302,8600
```

Step 5: Customize the inventory file for the MiCADO master.

We recommend making a copy of our predefined template and edit it. Use the template inventory file, called sample-hosts for customisation.

```
cp sample-hosts hosts
vi hosts
```

Edit the `hosts` file to set ansible variables for MiCADO master machine. Update the following parameters: ansible_host=*IP*, ansible_connection=*ssh* and ansible_user=*YOUR SUDOER ACCOUNT*. Please, revise the other parameters as well, however in most cases the default values are correct.

Step 6: Start the installation of MiCADO master.

```
ansible-playbook -i hosts micado-master.yml
```

1.1.3 Health checking

At the end of the deployment, core MiCADO services will be running on the MiCADO master machine. Here are the commands to test the operation of some of the core MiCADO services:

- Occopus:

```
curl -s -X GET http://IP:5000/infrastructures/
```

- Prometheus:

```
curl -s http://IP:9090/api/v1/status/config | jq '.status'
```

1.1.4 Check the logs

Alternatively, you can SSH into MiCADO master and check the logs at any point after MiCADO is succesfully deployed. All logs are kept under `/var/log/micado` and are organised by component. Scaling decisions, for example, can be inspected under `/var/log/micado/policykeeper`

1.2 Dashboard

MiCADO has a simple dashboard that collects web-based user interfaces into a single view. To access the Dashboard, visit `http://IP:4000`.

The following webpages are currently exposed:

- Docker visualizer: it graphically visualizes the Swarm nodes and the containers running on them.
- Grafana: graphically visualize the resources (nodes, containers) in time.
- Prometheus: monitoring subsystem. Recommended for developers, experts.

1.3 REST API

MiCADO has a TOSCA compliant submitter to submit, update, list and remove MiCADO applications. The submitter exposes the following REST API:

- To launch an application specified by a TOSCA description stored locally, use this command:

```
curl -F file=@[path to the TOSCA description] -X POST http://[IP]:5050/v1.0/app/
↪launch/file/
```

- To launch an application specified by a TOSCA description stored locally and specify an application id, use this command:

```
curl -F file=@[path to the TOSCA description] -F id=[APPLICATION_ID]  -X POST http://
↪[IP]:5050/v1.0/app/launch/file/
```

- To launch an application specified by a TOSCA description stored behind a url, use this command:

```
curl -d input="[url to TOSCA description]" -X POST http://[IP]:5050/v1.0/app/launch/
↪url/
```

- To launch an application specified by a TOSCA description stored behind an url and specify an application id, use this command:

```
curl -d input="[url to TOSCA description]" -d id=[ID] -X POST http://[IP]:5050/v1.0/
↪app/launch/url/
```

- To update a running MiCADO application using a TOSCA description stored locally, use this command:

```
curl -F file=@"[path to the TOSCA description]" -X PUT http://[IP]:5050/v1.0/app/
↪update/file/[APPLICATION_ID]
```

- To update a running MiCADO application using a TOSCA description stored behind a url, use this command:

```
curl -d input="[url to TOSCA description]" -X PUT http://[IP]:5050/v1.0/app/update/
↪file/[APPLICATION_ID]
```

- To undeploy a running MiCADO application, use this command:

```
curl -X DELETE http://[IP]:5050/v1.0/app/undeploy/[APPLICATION_ID]
```

- To query all the running MiCADO applications, use this command:

```
curl -X GET http://[IP]:5050/v1.0/list_app/
```

- To query one running MiCADO application, use this command:

```
curl -X GET http://[IP]:5050/v1.0/app/[APPLICATION_ID]
```

1.4 Application description

MiCADO executes applications described by the Application Descriptions following the TOSCA format. This section details the structure of the application description.

Application description has four main sections:

- **tosca_definitions_version**: `tosca_simple_yaml_1_0`.
- **imports**: a list of urls pointing to custom TOSCA types. The default url points to the custom types defined for MiCADO. Please, do not modify this url.
- **repositories**: docker repositories with their addresses.
- **topology_template**: the main part of the application description to define 1) docker services, 2) virtual machine (under the **node_templates** section) and 3) the scaling policy under the **policies** subsection. These sections will be detailed in subsections below.

Here is an overview example for the structure of the MiCADO application description:

```
tosca_definitions_version: tosca_simple_yaml_1_0

imports:
  - https://raw.githubusercontent.com/micado-scale/tosca/v0.5.0/micado_types.yaml

repositories:
  docker_hub: https://hub.docker.com/

topology_template:
  node_templates:
    YOUR_DOCKER_SERVICE:
      type: tosca.nodes.MiCADO.Container.Application.Docker
      properties:
        ...
      artifacts:
        ...
    ...
    YOUR_OTHER_DOCKER_SERVICE:
      type: tosca.nodes.MiCADO.Container.Application.Docker
      properties:
        ...
      artifacts:
        ...
    YOUR_DOCKER_NETWORK:
      type: tosca.nodes.MiCADO.network.Network.Docker
      properties:
        ...

    YOUR_VIRTUAL_MACHINE:
      type: tosca.nodes.MiCADO.Occopus.<CLOUD_API_TYPE>.Compute
```

(continues on next page)

(continued from previous page)

```
      properties:
        cloud:
          interface_cloud: ...
          endpoint_cloud: ...
      capabilities:
        host:
          properties:
            ...

  policies:
  - scalability:
    type: tosca.policies.Scaling.MiCADO
    targets: [ YOUR_VIRTUAL_MACHINE ]
    properties:
      ...
  - scalability:
    type: tosca.policies.Scaling.MiCADO
    targets: [ YOUR_DOCKER_SERVICE ]
    properties:
      ...
  - scalability:
    type: tosca.policies.Scaling.MiCADO
    targets: [ YOUR_OTHER_DOCKER_SERVICE ]
    properties:
      ...
```

1.4.1 Specification of Docker services

Under the node_templates section you can define any number of interconnected Docker service (see **YOUR_DOCKER_SERVICE**) similarly as in a docker-compose file. Each docker service definition consists of three main parts: type, properties and artifacts. The value of the **type** keyword for a Docker service must always be `tosca.nodes.MiCADO.Container.Application.Docker`. The **properties** section will contain most of the setting of the Docker service. Under the **artifacts** section the Docker image (see **YOUR_DOCKER_IMAGE**) must be defined. Optionally, Docker networks can be defined in the same way as in a docker-compose file (see **YOUR_DOCKER_NETWORK**).

```
topology_template:
  node_templates:
    YOUR_DOCKER_SERVICE:
      type: tosca.nodes.MiCADO.Container.Application.Docker
      properties:
          ...
      artifacts:
       image:
         type: tosca.artifacts.Deployment.Image.Container.Docker
         file: YOUR_DOCKER_IMAGE
         repository: docker_hub
    YOUR_DOCKER_NETWORK:
      type: tosca.nodes.MiCADO.network.Network.Docker
      properties:
        ...
```

The fields under the **properties** section of the Docker service are derived from the docker-compose file. Therefore, you can additional information about the properties in the docker compose documentation. The syntax of the property values is the same as in the docker-compose file.

Under the **properties** section of a Docker service (see **YOUR_DOCKER_SERVICE**) you can specify the following keywords:

- **command**: command line expression to be executed by the container.
- **deploy**: Swarm specific deployment options.
- **entrypoint**: override the default entrypoint of container.
- **environment**: map of all required environment variables.
- **expose**: expose ports without publishing them to the host machine.
- **labels**: map of metadata like Docker labels.
- **logging**: map of the logging configuration.
- **networks**: list of connected networks for the service.
- **volumes**: list of connected volumes for the service.
- **ports**: list of published ports to the host machine.
- **secrets**: list of per-service secrets to grant access for the service.

Under the **artifacts** section you can define the docker image for the docker service. Three fileds must be defined:

- **type**: `tosca.artifacts.Deployment.Image.Container.Docker`
- **file**: docker image for the docker service(e.g. sztakilpds/cqueue_frontend:latest)
- **repository**: name of the repository where the image is located. The name used here (e.g. docker_hub), must be defined at the top of the description under the **repositories** section.

To define a Docker network (see **YOUR_DOCKER_NETWORK**) the following fields must be specified:

- **attachable**: if set to true, then standalone containers can attach to this network, in addition to services
- **driver**: specify which driver should be used for this network. (overlay, bridge, etc.)

1.4.2 Specification of the Virtual Machine

The network of Docker services specified in the previous section is executed under Docker Swarm. This section introduces how the parameters of the virtual machine can be configured which will be hosts the Docker worker node. During operation MiCADO will instantiate as many virtual machines with the parameters defined here as required during scaling. MiCADO currently supports four different cloud interfaces: CloudSigma, CloudBroker, EC2, Nova. The following ports and protocols should be enabled on the virtual machine:

```
ICMP
TCP: 22,2377,7946,8300,8301,8302,8500,8600,9100,9200
UDP: 4789,7946,8301,8302,8600
```

The following subsections details how to configure them.

CloudSigma

To instantiate MiCADO workers on CloudSigma, please use the template below. MiCADO **requires** num_cpus, mem_size, vnc_password, libdrive_id and public_key_id to instantiate VM on *CloudSigma*.

```
topology_template:
  node_templates:
    worker_node:
      type: tosca.nodes.MiCADO.Occopus.CloudSigma.Compute
      properties:
        cloud:
          interface_cloud: cloudsigma
          endpoint_cloud: ADD_YOUR_ENDPOINT (e.g for cloudsigma https://zrh.
↪cloudsigma.com/api/2.0 )
      capabilities:
        host:
          properties:
            num_cpus: ADD_NUM_CPUS_FREQ (e.g. 4096)
            mem_size: ADD_MEM_SIZE (e.g. 4294967296)
            vnc_password: ADD_YOUR_PW (e.g. secret)
            libdrive_id: ADD_YOUR_ID_HERE (eg. 87ce928e-e0bc-4cab-9502-514e523783e3)
            public_key_id: ADD_YOUR_ID_HERE (e.g. d7c0f1ee-40df-4029-8d95-
↪ec35b34dae1e)
            firewall_policy: ADD_YOUR_ID_HERE (e.g. fd97e326-83c8-44d8-90f7-
↪0a19110f3c9d)
```

- **num_cpu** is the speed of CPU (e.g. 4096) in terms of MHz of your VM to be instantiated. The CPU frequency required to be between 250 and 100000
- **mem_size** is the amount of RAM (e.g. 4294967296) in terms of bytes to be allocated for your VM. The memory required to be between 268435456 and 137438953472
- **vnc_password** set the password for your VNC session (e.g. secret).
- **libdrive_id** is the image id (e.g. 87ce928e-e0bc-4cab-9502-514e523783e3) on your CloudSigma cloud. Select an image containing a base os installation with cloud-init support!
- **public_key_id** specifies the keypairs (e.g. d7c0f1ee-40df-4029-8d95-ec35b34dae1e) to be assigned to your VM.
- **firewall_policy** optionally specifies network policies (you can define multiple security groups in the form of a list, e.g. fd97e326-83c8-44d8-90f7-0a19110f3c9d) of your VM.

CloudBroker

To instantiate MiCADO workers on CloudBroker, please use the template below. MiCADO **requires** deployment_id and instance_type_id to instantiate a VM on *CloudBroker*.

```
topology_template:
  node_templates:
    worker_node:
      type: tosca.nodes.MiCADO.Occopus.CloudBroker.Compute
      properties:
        cloud:
          interface_cloud: cloudbroker
          endpoint_cloud: ADD_YOUR_ENDPOINT (e.g https://cola-prototype.cloudbroker.
↪com )
      capabilities:
        host:
          properties:
            deployment_id: ADD_YOUR_ID_HERE (e.g. e7491688-599d-4344-95ef-
↪aff79a60890e)
            instance_type_id: ADD_YOUR_ID_HERE (e.g. 9b2028be-9287-4bf6-bbfe-
↪bcbc92f065c0)
```

(continues on next page)

(continued from previous page)

```
            key_pair_id: ADD_YOUR_ID_HERE (e.g. d865f75f-d32b-4444-9fbb-3332bcedeb75)
            opened_port: ADD_YOUR_PORTS_HERE (e.g. '22,2377,7946,8300,8301,8302,8500,
↪8600,9100,9200,4789')
```

- **deployment_id** is the id of a preregistered deployment in CloudBroker referring to a cloud, image, region, etc. Make sure the image contains a base OS (preferably Ubuntu) installation with cloud-init support! The id is the UUID of the deployment which can be seen in the address bar of your browser when inspecting the details of the deployment.
- **instance_type_id** is the id of a preregistered instance type in CloudBroker referring to the capacity of the virtual machine to be deployed. The id is the UUID of the instance type which can be seen in the address bar of your browser when inspecting the details of the instance type.
- **key_pair_id** is the id of a preregistered ssh public key in CloudBroker which will be deployed on the virtual machine. The id is the UUID of the key pair which can be seen in the address bar of your browser when inspecting the details of the key pair.
- **opened_port** is one or more ports to be opened to the world. This is a string containing numbers separated by a comma.

EC2

To instantiate MiCADO workers on a cloud through EC2 interface, please use the template below. MiCADO **requires** region_name, image_id and instance_type to instantiate a VM through *EC2*.

```
topology_template:
  node_templates:
    worker_node:
      type: tosca.nodes.MiCADO.Occopus.EC2.Compute
      properties:
        cloud:
          interface_cloud: ec2
          endpoint_cloud: ADD_YOUR_ENDPOINT (e.g ec2.eu-west-1.amazonaws.com )
      capabilities:
        host:
          properties:
            region_name: ADD_YOUR_REGION_NAME_HERE (e.g. eu-west-1)
            image_id: ADD_YOUR_ID_HERE (e.g. ami-12345678)
            instance_type: ADD_YOUR_INSTANCE_TYPE_HERE (e.g. t1.small)
```

- **region_name** is the region name within an EC2 cloud (e.g. eu-west-1).
- **image_id** is the image id (e.g. ami-12345678) on your EC2 cloud. Select an image containing a base os installation with cloud-init support!
- **instance_type** is the instance type (e.g. t1.small) of your VM to be instantiated.
- **key_name** optionally specifies the keypair (e.g. my_ssh_keypair) to be deployed on your VM.
- **security_group_ids** optionally specify security settings (you can define multiple security groups or just one, but this property must be formatted as a list, e.g. [sg-93d46bf7]) of your VM.
- **subnet_id** optionally specifies subnet identifier (e.g. subnet-644e1e13) to be attached to the VM.

Nova

To instantiate MiCADO workers on a cloud through Nova interface, please use the template below. MiCADO **requires** image_id flavor_name, project_id and network_id to instantiate a VM through *Nova*.

```
topology_template:
  node_templates:
    worker_node:
      type: tosca.nodes.MiCADO.Occopus.Nova.Compute
      properties:
        cloud:
          interface_cloud: nova
          endpoint_cloud: ADD_YOUR_ENDPOINT (e.g https://sztaki.cloud.mta.hu:5000/v3)
      capabilities:
        host:
          properties:
            image_id: ADD_YOUR_ID_HERE (e.g. d4f4e496-031a-4f49-b034-f8dafe28e01c)
            flavor_name: ADD_YOUR_ID_HERE (e.g. 3)
            project_id: ADD_YOUR_ID_HERE (e.g. a678d20e71cb4b9f812a31e5f3eb63b0)
            network_id: ADD_YOUR_ID_HERE (e.g. 3fd4c62d-5fbe-4bd9-9a9f-c161dabeefde)
            key_name: ADD_YOUR_KEY_HERE (e.g. keyname)
            security_groups:
              - ADD_YOUR_ID_HERE (e.g. d509348f-21f1-4723-9475-0cf749e05c33)
```

- **project_id** is the id of project you would like to use on your target Nova cloud.
- **image_id** is the image id on your Nova cloud. Select an image containing a base os installation with cloud-init support!
- **flavor_name** is the name of flavor to be instantiated on your Nova cloud.
- **server_name** optionally defines the hostname of VM (e.g.:"helloworld").
- **key_name** optionally sets the name of the keypair to be associated to the instance. Keypair name must be defined on the target nova cloud before launching the VM.
- **security_groups** optionally specify security settings (you can define multiple security groups in the form of a list) for your VM.
- **network_id** is the id of the network you would like to use on your target Nova cloud.

1.4.3 Description of the scaling policy

To utilize the autoscaling functionality of MiCADO, scaling policies can be defined on virtual machine and on docker service level. Scaling policies can be listed under the **policies** section. Each **scalability** subsection must have the **type** set to the value of `tosca.policies.Scaling.MiCADO` and must be linked to a node defined under **node_template**. The link can be implemented by specifying the name of the node under the **targets** subsection. The details of the scaling policy can be defined under the **properties** subsection. The structure of the **policies** section can be seen below.

```
topology_template:
  node_templates:
    YOUR_DOCKER_SERVICE:
      type: tosca.nodes.MiCADO.Container.Application.Docker
      ...
    ...
    YOUR_OTHER_DOCKER_SERVICE:
      type: tosca.nodes.MiCADO.Container.Application.Docker
```

(continues on next page)

(continued from previous page)

```
      ...
    YOUR_VIRTUAL_MACHINE:
      type: tosca.nodes.MiCADO.Occopus.<CLOUD_API_TYPE>.Compute
      ...

  policies:
  - scalability:
    type: tosca.policies.Scaling.MiCADO
    targets: [ YOUR_VIRTUAL_MACHINE ]
    properties:
      ...
  - scalability:
    type: tosca.policies.Scaling.MiCADO
    targets: [ YOUR_DOCKER_SERVICE ]
    properties:
      ...
  - scalability:
    type: tosca.policies.Scaling.MiCADO
    targets: [ YOUR_OTHER_DOCKER_SERVICE ]
    properties:
      ...
```

The scaling policies are evaluated periodically. In every turn, the virtual machine level scaling is evaluated, followed by the evaluation of each scaling policies belonging to Docker services.

The **properties** subsection defines the scaling policy itself. For monitoring purposes, MiCADO integrates the Prometheus monitoring tool with two built-in exporters on each worker node: Node exporter (to collect data on nodes) and CAdvisor (to collect data on containers). Based on Prometheus, any monitored information can be extracted using the Prometheus query language and the returned value can be associated to a user-defined variable. Once variables are updated, scaling rule is evaluated. It can be specified by a short Python code which can refer to the monitored information. The structure of the scaling policy can be seen below.

```
- scalability:
    ...
    properties:
      sources:
        - 'myprometheus.exporter.ip.address:portnumber'
      constants:
        LOWER_THRESHOLD: 50
        UPPER_THRESHOLD: 90
        MYCONST: 'any string'
      queries:
        THELOAD: 'Prometheus query expression'
        MYEXPR: 'something refering to {{MYCONST}}'
      alerts:
        - alert: myalert
          expr: 'Prometheus expression for an event important for scaling'
          for: 1m
      min_instances: 1
      max_instances: 5
      scaling_rule: |
        if myalert:
          m_node_count=5
        if THELOAD>UPPER_THRESHOLD:
          m_node_count+=1
        if THELOAD<LOWER_THRESHOLD:
          m_node_count-=1
```

The subsections have the following roles:

- **sources** supports the dynamic attachment of an external exporter by specifying a list endpoints of exporters (see example above). Each item found under this subsection is configured under Prometheus to start collecting the information provided/exported by the exporters. Once done, the values of the parameters provided by the exporters become available.
- **constants** subsection is used to predefined fixed parameters. Values associated to the parameters can be referred by the scaling rule as variable (see `LOWER_THRESHOLD` above) or in any other sections referred as Jinja2 variable (see `MYEXPR` above).
- **queries** contains the list of Prometheus query expressions to be executed and their variable name associated (see `THELOAD` above)
- **alerts** subsection enables the utilisation of the alerting system of Prometheus. Each alert defined here is registered under Prometheus and fired alerts are represented with a variable of their name set to True during the evaluation of the scaling rule (see `myalert` above).
- **min_instances** keyword specifies the lowest number of instances valid for the node.
- **max_instances** keyword specifies the highest number of instances valid for the node.
- **scaling_rule** specifies Python code to be evaluated periodically to decide on the number of instances. The Python expression must be formalized with the following conditions:
  - Each constant defined under the 'constants' section can be referred; its value is the one defined by the user.
  - Each variable defined under the 'queries' section can be referred; its value is the result returned by Prometheus in response to the query string.
  - Each alert name defined under the 'alerts' section can be referred, its value is a logical True in case the alert is firing, False otherwise
  - Expression must follow the syntax of the Python language
  - Expression can be multiline
  - The following predefined variables can be referred; their values are defined and updated before the evaluation of the scaling rule
    - m_nodes: python list of nodes belonging to the docker swarm cluster
    - m_node_count: the target number of nodes
    - m_container_count: the target number of containers for the service the evaluation belongs to
    - m_time_since_node_count_changed: time in seconds elapsed since the number of nodes changed
  - In a scaling rule belonging to the virtual machine, the name of the variable to be updated is `m_node_count`; as an effect the number stored in this variable will be set as target instance number for the virtual machines.
  - In a scaling rule belonging to a docker service, the name of the variable to be set is `m_container_count`; as an effect the number stored in this variable will be set as target instance number for the docker service.

For further examples, inspect the scaling policies of the demo examples detailed in the next section.

1.5 Tutorials

You can find test application(s) under the subdirectories of the 'testing' directory. The current tests are configured for CloudSigma.

1.5.1 stressng

This application contains a single service, performing constant load. Policy defined for this application scales up/down both nodes and the stressng service based on cpu consumption. Helper scripts has been added to the directory to ease application handling.

- Step1: add your `public_key_id` to both the `stressng.yaml` and `stressng-update.yaml` files. Without this CloudSigma does not execute the contextualisation on the MiCADO worker nodes. The ID must point to your public ssh key under your account in CloudSigma. You can find it on the CloudSigma Web UI under the "Access & Security/Keys Management" menu.
- Step2: add a proper `firewall_policy` to both the `stressng.yaml` and `stressng-update.yaml` files. Without this MiCADO master will not reach MiCADO worker nodes. Firewall policy ID can be retrieved from a rule defined under the "Networking/Policies" menu. The following ports must be opened for MiCADO workers: all inbound connections from MiCADO master
- Step3: set the MICADO_MASTER variable to contain the IP of the MiCADO master node with `export MICADO_MASTER=a.b.c.d`
- Step4: run `1-submit-tosca-stressng.sh` to create the minimum number of MiCADO worker nodes and to deploy the docker stack including the stressng service defined in the `stressng.yaml` TOSCA description. Optionally, add as an argument a user-defined application id (ie. `1-submit-tosca-stressng.sh stressng`). The system should respond by scaling up virtual machines and containers to the maximum specified.
- Step4a: run `2-list-apps.sh` to see currently running applications and their IDs
- Step5: run `3-update-tosca-stressng.sh <ID>` with the appropriate ID to update the service and reduce the CPU load. The system should respond by scaling down virtual machines and containers to the minimum specified.
- Step6: run `4-undeploy-with-id.sh <ID>` with the appropriate ID to remove the stressng stack and all the MiCADO worker nodes

1.5.2 cqueue

This application demonstrates a deadline policy using CQueue. CQueue provides a lightweight queueing service for executing containers. CQueue server (implemented by RabbitMQ, Redis and a web-based frontend) stores items where each represents a container execution. CQueue worker fetches an item and preform the execution of the container locally. The demonstration below shows that the items can be consumed by deadline using MiCADO for scaling the CQueue worker. The demonstration requires the deployment of a CQueue server separately, then the submission of the CQueue worker to MiCADO with the appropriate (predefined) scaling policy.

- Step1: Launch a separate VM and deploy CQueue server using the compose file, called `docker-compose-cqueue-server.yaml`. You need to install docker and docker-compose to use the compose file. This will be your cqueue server to store items representing container execution requests. Important: you have to open ports defined under the 'ports' section for each of the four services defined in the compose file.
- Step2: Update the parameter file, called `_settings` . You need the ip address for the MiCADO master and for the CQueue server.
- Step3: Run `./1-submit-jobs.sh 50` to generate and send 50 jobs to CQueue server. Each item will be a simple Hello World app (combined with some sleep) realized in a container. You can later override this with your own container.
- Step4: Edit the TOSCA description file, called `micado-cqworker.yaml`.
  - Replace each 'cqueue.server.ip.address' string with the real ip of CQueue server.

- Update each 'ADD_YOUR_ID_HERE' string with the proper value retrieved under your CloudSigma account.

- Step5: Run `./2-get_date_in_epoch_plus_seconds.sh 600` to calculate the unix timestamp representing the deadline by which the items (containers) must be finished. Take the value from the last line of the output produced by the script. The value is 600 seconds from now.
- Step6: Edit the TOSCA description file, called `micado-cqworker.yaml`.
  - Update the value for the 'DEADLINE' which is under the 'policies/scalability/properties/constants' section. The value has been extracted in the previous step. Please, note that defining a deadline in the past results in scaling the application to the maximum (2 nodes and 10 containers).
- Step7: Run `./3-deploy-cq-worker-to-micado.sh` to deploy the CQworker service, which will consume the items from the CQueue server i.e. execute the containers specified by the items.
- Step8: Monitor the application:
  - visit http://micado.master.ip:4000/docker-visualizer to see the number of nodes and containers.
  - watch the log of policy keeper on the MiCADO master (/var/log/micado/policykeeper).
- Step9: Run `./4-list-running-apps.sh` to list the apps you are running under MiCADO.
- Step10: Run `./5-undeploy-cq-worker-from-micado.sh` to remove your application from MiCADO when all items are consumed.
- Step11: You can have a look at the state `./cqueue-get-job-status.sh <task_id>` or stdout of container executions `./cqueue-get-job-status.sh <task_id>` using one of the task id values printed during Step 3.

1.6 Release Notes

v0.5.0 (12 July 2018)

- Introduce supporting TOSCA
- Introduce supporting user-defined scaling policy
- Dashboard added with Docker Visualizer, Grafana, Prometheus
- Deployment with Ansible playbook
- Support private docker registry
- Improve persistence of MiCADO master services