
MiCADO Documentation

Attila Farkas

Feb 25, 2019

1	Introduction	3
1.1	Deployment	3
1.2	Dashboard	8
1.3	REST API	8
1.4	Application description	9
1.5	Tutorials	19
1.6	Release Notes	23

This software is developed by the [COLA project](#) and is hosted at the [MiCADO-scale github repository](#). Please, visit the [MiCADO homepage](#) for general information about the product.

MiCADO is an auto-scaling framework for Docker containers, orchestrated by Kubernetes. It supports autoscaling at two levels. At virtual machine (VM) level, a built-in Kubernetes cluster is dynamically extended or reduced by adding/removing cloud virtual machines. At Kubernetes level, the number of replicas tied to a specific Kubernetes Deployment can be increased/decreased.

MiCADO requires a TOSCA based Application Description to be submitted containing three sections: 1) the definition of the individual applications making up a Kubernetes Deployment, 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 is configured as the Kubernetes Master Node and has installed the Docker Engine, 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 newly instantiated MiCADO workers join the Kubernetes cluster managed by 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 Kubernetes Deployments. They are the Kubernetes Dashboard, Grafana and Prometheus. Finally, advanced users may find the logs of the MiCADO core services useful on MiCADO master.

1.1 Deployment

To deploy MiCADO you need a (separate) virtual machine, called MiCADO master. There are two ways of deployment:

- remote: download the Ansible playbook on your local machine, configure the MiCADO master as target machine and run the playbook to perform the deployment remotely.
- local: login to the MiCADO master, download the Ansible playbook, configure the localhost as target machine and run the playbook to perform the deployment locally.

We recommend to perform the installation remotely as all your configuration files are preserved on your machine, i.e. it is easier to repeat the deployment if needed.

1.1.1 Prerequisites

For cloud interfaces supported by MiCADO:

- EC2 (tested on Amazon and OpenNebula)
- Nova (tested on OpenStack)
- CloudSigma
- CloudBroker

For the MiCADO master:

- Ubuntu 16.04
- (Minimum) 2GHz CPU & 2GB RAM
- (Recommended) 2GHz CPU & 4GB RAM

For the host where the Ansible playbook is executed (differs depending on local or remote):

- Ansible 2.4 or greater
- curl
- jq (to pretty-format API responses)
- wrk (to load test nginx & wordpress demonstrators)

Ansible

Note: Ansible in the Ubuntu 16.04 APT repository is outdated and insufficient (at the time of writing this document)

To install Ansible on Ubuntu 16.04, use these commands:

```
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 systems follow the [official installation guide](#).

curl

To install curl on Ubuntu, use this command:

```
sudo apt-get install curl
```

To install curl on other operating systems follow the [official installation guide](#).

jq

To install jq on Ubuntu, use this command:

```
sudo apt-get install jq
```

To install jq on other operating systems follow the [official installation guide](#).

wrk

To install wrk on Ubuntu, use this command:

```
sudo apt-get install wrk
```

To install wrk on other operating systems check the sidebar on the [github wiki](#).

1.1.2 Installation

Perform the following steps either on your local machine or on MiCADO master depending on the installation method.

Step 1: Download the ansible playbook.

```
curl --output ansible-micado-0.7.2.tar.gz -L https://github.com/micado-scale/ansible-  
↪micado/releases/download/v0.7.2/ansible-micado-0.7.2.tar.gz  
tar -zxvf ansible-micado-0.7.2.tar.gz  
cd ansible-micado-0.7.2/
```

Step 2: Specify cloud credential for instantiating MiCADO workers.

MiCADO master will use this credential against the cloud API to start/stop VM instances (MiCADO workers) to host the application and 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. MiCADO expects the credential in a file, called `credentials-cloud-api.yml` before deployment. Please, do not modify the structure of the template!

```
cp sample-credentials-cloud-api.yml credentials-cloud-api.yml  
edit credentials-cloud-api.yml
```

Edit `credentials-cloud-api.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.

Optionally you can use the [Ansible Vault](#) mechanism to keep the credential data in an encrypted format. To achieve this, create the above file using Vault with the command

```
ansible-vault create credentials-cloud-api.yml
```

This will launch the editor defined in the `$EDITOR` environment variable to make changes to the file. If you wish to make any changes to the previously encrypted file, you can use the command

```
ansible-vault edit credentials-cloud-api.yml
```

Step 3a: Specify security settings and credentials to access MiCADO.

MiCADO master will use these security-related settings and credentials to authenticate its users for accessing the REST API and Dashboard.

```
cp sample-credentials-micado.yml credentials-micado.yml
edit credentials-micado.yml
```

Specify the provisioning method for the x509 keypair used for TLS encryption of the management interface in the `tls` subtree:

- The ‘self-signed’ option generates a new keypair with the specified hostname as subject (or ‘micado-master’ if omitted).
- The ‘user-supplied’ option lets the user add the keypair as plain multiline strings (in unencrypted format) in the `ansible_user_data.yml` file under the ‘cert’ and ‘key’ subkeys respectively.

Specify the default username and password for the administrative user in the `authentication` subtree.

Optionally you may use the Ansible Vault mechanism as described in Step 2 to protect the confidentiality and integrity of this file as well.

Step 3b: (Optional) Specify credentials to use private Docker registries.

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

```
cp sample-credentials-docker-registry.yml credentials-docker-registry.yml
edit credentials-docker-registry.yml
```

Edit `credentials-docker-registry.yml` and add username, password, and registry url. To login to the default `docker_hub`, leave `DOCKER_REPO` as is (<https://index.docker.io/v1/>).

Optionally you may use the Ansible Vault mechanism as described in Step 2 to protect the confidentiality and integrity of this file as well.

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

This new VM will host the MiCADO core services.

a) Default port number for MiCADO service is 443. Optionally, you can modify the port number stored by the variable called `web_listening_port` defined in the ansible playbook file called `micado-master.yml`.

b) Configure a cloud firewall settings which opens the following ports on the MiCADO master virtual machine:

```
TCP: 22, 2380, 6443, 8300, 8301, 8302, 8500, 8600, 10252, [web_listening_port]
UDP: 2379, 8301, 8302, 8472, 8600, 10250, 10251
```

NOTE: replace `[web_listening_port]` with the actual value specified in Step 4a.

NOTE: MiCADO master has built-in firewall, therefore you can leave all ports open at cloud level.

c) Finally, launch the virtual machine with the proper settings (capacity, ssh keys, firewall): 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.

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
edit hosts
```

Edit the `hosts` file to set ansible variables for MiCADO master machine. Update the following parameters:

- **ansible_host**: specifies the publicly reachable ip address of MiCADO master. Set the public or floating IP of the master regardless the deployment method is remote or local. The ip specified here is used by the Dashboard for webpage redirection as well
- **ansible_connection**: specifies how the target host can be reached. Use “ssh” for remote or “local” for local installation. In case of remote installation, make sure you can authenticate yourself against MiCADO master. We recommend to deploy your public ssh key on MiCADO master before starting the deployment
- **ansible_user**: specifies the name of your sudoer account, defaults to “ubuntu”
- **ansible_become**: specifies if account change is needed to become root, defaults to “True”
- **ansible_become_method**: specifies which command to use to become superuser, defaults to “sudo”
- **ansible_python_interpreter**: specifies the interpreter to be used for ansible on the target host, defaults to “usr/bin/python3”

Please, revise all the parameters, however in most cases the default values are correct.

Step 6: Start the installation of MiCADO master.

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

If you have used Vault to encrypt your credentials, you have to add the path to your vault credentials to the command line as described in the [Ansible Vault documentation](#) or provide it via command line using the command

```
ansible-playbook -i hosts micado-master.yml --ask-vault-pass
```

1.1.3 After deployment

Once the deployment has successfully finished, you can proceed with

- visiting the *Dashboard*
- using the *REST API*
- playing with the *Tutorials*
- creating your *Application description*

1.1.4 Check the logs

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

1.1.5 Accessing user-defined service

In case your application contains a container exposing a service, you will have to ensure the following to access it.

- First set **nodePort: xxxxx** (where xxxxx is a port in range 30000-32767) in the **properties: ports:** TOSCA description of your docker container. More information on this in the [Application description](#)
- The container will be accessible at `<IP>:<port>` . Both, the IP and the port values can be extracted from the Kubernetes Dashboard (in case you forget it). The **IP** can be found under *Nodes > my_micado_vm > Addresses* menu, while the **port** can be found under *Discovery and load balancing > Services > my_app > Internal endpoints* menu.

1.2 Dashboard

MiCADO has a simple dashboard that collects web-based user interfaces into a single view. To access the Dashboard, visit `https://[IP]:[PORT]`, where

- [IP] is the ip address of MiCADO master, the virtual machine you have launched in Step 4 of [Deployment](#)
- [PORT] is the port number configured during Step 4 of [Deployment](#), its value is held by the `web_listening_port` variable specified in the `micado-master.yml` ansible file.

The following webpages are currently exposed:

- Kubernetes Dashboard: A read-only instance of the Kubernetes WebUI providing a full overview of the infrastructure. Simply *SKIP* the authentication pop-up to gain read-only access to the dashboard.
- Grafana: graphically visualize the resources (nodes, containers) in time. After deploying your application, you can select the service whose metrics you want using the ‘Service’ drop down running above the graphs area.
- 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 (with an option in bold to specify an ID):

```
curl --insecure -s -F file=@[path to the TOSCA template] **-F id=[APPLICATION_ID]** -  
↪X POST https://[username]:[password]@[IP]:[port]/toscasubmitter/v1.0/app/launch/
```

- To launch an application specified by a TOSCA description stored behind a url (with an option in bold to specify an ID):

```
curl --insecure -s -d input="[url to TOSCA description]" **-d id=[APPLICATION_ID]** -  
↪X POST https://[username]:[password]@[IP]:[port]/toscasubmitter/v1.0/app/launch/
```

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

```
curl --insecure -s -F file=@"[path to the TOSCA description]" -X PUT https://  
↪[username]:[password]@[IP]:[port]/toscasubmitter/v1.0/app/update/[APPLICATION_ID]
```

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

```
curl --insecure -s -d input="[url to TOSCA description]" -X PUT https://
↳ [username]:[password]@[IP]:[port]/toscasubmitter/v1.0/app/update/[APPLICATION_ID]
```

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

```
curl --insecure -s -X DELETE https://[username]:[password]@[IP]:[port]/toscasubmitter/
↳ v1.0/app/undeploy/[APPLICATION_ID]
```

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

```
curl --insecure -s -X GET https://[username]:[password]@[IP]:[port]/toscasubmitter/v1.
↳ 0/list_app/
```

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

```
curl --insecure -s -X GET https://[username]:[password]@[IP]:[port]/toscasubmitter/v1.
↳ 0/[APPLICATION_ID]/status
```

- To query the full execution status of MiCADO, use this command:

```
curl --insecure -s -X GET https://[username]:[password]@[IP]:[port]/toscasubmitter/v1.
↳ 0/info_threads
```

- To query the services of a running MiCADO application, use this command:

```
curl --insecure -s -d query='services' -X GET https://
↳ [username]:[password]@[IP]:[port]/toscasubmitter/v1.0/app/query/[APPLICATION_ID]
```

- To query the nodes hosting a running MiCADO application, use this command:

```
curl --insecure -s -d query='nodes' -X GET https://[username]:[password]@[IP]:[port]/
↳ toscasubmitter/v1.0/app/query/[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 goal of the application description is to define 1) kubernetes deployments (of docker containers), 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 example for the structure of the MiCADO application description:

```
tosca_definitions_version: toscasimpleyaml_1_0
imports:
```

(continues on next page)

```
- https://raw.githubusercontent.com/micado-scale/tosca/v0.x.2/micado_types.yaml

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

topology_template:
  node_templates:
    YOUR-KUBERNETES-APP:
      type: toasca.nodes.MiCADO.Container.Application.Docker
      properties:
        ...
      artifacts:
        ...
      interfaces:
        ...
    ...
    YOUR-OTHER-KUBERNETES-APP:
      type: toasca.nodes.MiCADO.Container.Application.Docker
      properties:
        ...
      artifacts:
        ...
      interfaces:
        ...

    YOUR_VIRTUAL_MACHINE:
      type: toasca.nodes.MiCADO.<CLOUD_API_TYPE>.Compute
      properties:
        ...
      interfaces:
        ...
      capabilities:
        host:
          properties:
            ...

  outputs:
    ports:
      value: { get_attribute: [ YOUR-KUBERNETES-APP, port ] }

  policies:
    - scalability:
      type: toasca.policies.Scaling.MiCADO
      targets: [ YOUR_VIRTUAL_MACHINE ]
      properties:
        ...
    - scalability:
      type: toasca.policies.Scaling.MiCADO
      targets: [ YOUR-KUBERNETES-APP ]
      properties:
        ...
    - scalability:
      type: toasca.policies.Scaling.MiCADO
      targets: [ YOUR-OTHER-KUBERNETES-APP ]
      properties:
        ...
```

1.4.1 Specification of Docker containers (to be orchestrated by Kubernetes)

NOTE Kubernetes does not allow for underscores in any resource names (read: TOSCA node names). Names must also begin and end with an alphanumeric.

Under the `node_templates` section you can define one or more Docker containers and choose to orchestrate them with Kubernetes (see **YOUR-KUBERNETES-APP**). Each app is described as a separate node with its own definition consisting of four main parts: type, properties, artifacts and interfaces.

The **type** keyword for Docker containers must always be `tosca.nodes.MiCADO.Container.Application.Docker`.

The **properties** section will contain the options specific to the Docker container runtime

The **artifacts** section must define the Docker image (see **YOUR_DOCKER_IMAGE**)

The **interfaces** section tells MiCADO how to orchestrate the container.

The `create` field *inputs* will override the **workload** metadata & spec of a bare Kubernetes Deployment manifest.

The `configure` field *inputs* will override the **pod** metadata & spec of that workload.

A stripped back definition of a node_template looks like this:

```

topology_template:
  node_templates:
    YOUR-KUBERNETES-APP:
      type: toasca.nodes.MiCADO.Container.Application.Docker
      properties:
        name:
        command:
        args:
        env:
        ...
      artifacts:
        image:
          type: toasca.artifacts.Deployment.Image.Container.Docker
          file: YOUR_DOCKER_IMAGE
          repository: docker_hub
      interfaces:
        Kubernetes:
          create:
            implementation: image
            inputs:
              ...
          configure:
            inputs:
              ...
      outputs:
        ports:
          value: { get_attribute: [ YOUR-KUBERNETES-APP, port ]}

```

The fields under the **properties** section of the Kubernetes app are a collection of options specific to all iterations of Docker containers. The translator understands both Docker-Compose style naming and Kubernetes style naming, though the Kubernetes style is recommended. You can find additional information about properties in the [translator documentation](#). These properties will be translated into Kubernetes manifests on deployment.

Under the **properties** section of an app (see **YOUR-KUBERNETES-APP**) here are a few common keywords.:

- **name**: name for the container (defaults to the TOSCA node name)
- **command**: override the default command line expression to be executed by the container.

- **args**: override the default entrypoint of container.
- **env**: list of *name*: & *value*: of all required environment variables.
- **resource.requests.cpu**: CPU reservation, should be set 100m lower than max (900m == 1000m)
- **ports**: list of published ports to the host machine, you can specify these keywords in the style of a [Kubernetes Service](#)
 - **targetPort**: the port to target (assumes port if not specified)
 - **port**: the port to publish (assumes targetPort if not specified)
 - **name**: the name of this port in the service (will be generated if not specified)
 - **protocol**: the protocol for the port (defaults to: TCP)
 - **nodePort**: the port (30000-32767) to expose on the host (this will create a nodePort Service unless type is explicitly set below)
 - **type**: the type of service for this port (defaults to: ClusterIP except if nodePort is defined above)
 - **clusterIP**: the desired (internal) IP (10.0.0.0/24) for this service (defaults to next available)
 - **metadata**: service metadata, giving the option to set a name for the service. Explicit naming can be used to group different ports together (default grouping is by type)

Under the **artifacts** section you can define the docker image for the kubernetes app. Three fields must be defined:

- **type**: `tosca.artifacts.Deployment.Image.Container.Docker`
- **file**: docker image for the kubernetes app (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.

Under the **interfaces** section you can define orchestrator specific options, here we use the key **Kubernetes**:

- **create**: *this key tells MiCADO to create a workload (Deployment/DaemonSet/Job/Pod etc. . .) for this container*
 - **implementation**: this should always point to your image artifact
 - **inputs**: top-level workload and workload spec options follow here. . . Some examples, see [translator documentation](#)
 - * **kind**: overwrite the workload type (defaults to Deployment)
 - * **strategy.type**: change to Recreate to kill pods then update (defaults to RollingUpdate)
- **configure**: *this key configures the Pod for this workload*
 - **inputs**: `PodTemplateSpec` options follow here. . . For example
 - * **restartPolicy**: change the restart policy (defaults to Always)

A word on networking in Kubernetes

Kubernetes networking is inherently different to the approach taken by Docker/Swarm. This is a complex subject which is worth a read: <https://kubernetes.io/docs/concepts/cluster-administration/networking/> . Since every pod gets its own IP, which any pod can by default use to communicate with any other pod, this means there is no network to explicitly define. If the **ports** keyword is defined in the definition above, pods can reach each other over CoreDNS via their hostname (container name).

Under the **outputs** section (this key is **NOT** nested within *node_templates*) you can define an output to retrieve from Kubernetes via the adaptor. Currently, only port info is obtainable.

1.4.2 Specification of the Virtual Machine

The collection of docker containers (kubernetes applications) specified in the previous section is orchestrated by Kubernetes. This section introduces how the parameters of the virtual machine can be configured which will host the Kubernetes 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 acting as MiCADO worker:

```
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.

General

The **capabilities** sections for all virtual machine definitions that follow are identical and are **ENTIRELY OPTIONAL**. They are filled with metadata to support human readability.:

- **num_cpus** under *host* is a readable string specifying clock speed of the instance type
- **mem_size** under *host* is a readable string specifying RAM of the instance type
- **type** under *os* is a readable string specifying the operating system type of the image
- **distribution** under *os* is a readable string specifying the OS distro of the image
- **version** under *os* is a readable string specifying the OS version of the image

The **interfaces** section of all virtual machine definitions that follow are **REQUIRED**, and allow you to provide orchestrator specific inputs, in the examples below we use **Occopus**.

- **create**: *this key tells MiCADO to create the VM using Occopus*
 - **inputs**: Specific settings for Occopus follow here
 - * **interface_cloud**: tells Occopus which cloud to interface with
 - * **endpoint_cloud**: tells Occopus the endpoint API of the cloud

CloudSigma

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

```
topology_template:
  node_templates:
    worker_node:
      type: toasca.nodes.MiCADO.Occopus.CloudSigma.Compute
      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)
        nics:
          - firewall_policy: ADD_YOUR_FIREWALL_POLICY_ID_HERE (e.g. fd97e326-83c8-
↪44d8-90f7-0a19110f3c9d)
```

(continues on next page)

(continued from previous page)

```

    ip_v4_conf:
      conf: dhcp
  capabilities:
    # OPTIONAL METADATA
  host:
    properties:
      num_cpus: 2GHz
      mem_size: 2GB
  os:
    properties:
      type: linux
      distribution: ubuntu
      version: 16.04
  interfaces:
    Occopus:
      create:
        inputs:
          interface_cloud: cloudsigma
          endpoint_cloud: ADD_YOUR_ENDPOINT (e.g for cloudsigma https://zrh.
↪cloudsigma.com/api/2.0 )

```

Under the **properties** section of a CloudSigma virtual machine definition these inputs are available.:

- **num_cpus** 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.
- **nics[.firewall_policy | .ip_v4_conf.conf]** specifies network policies (you can define multiple security groups in the form of a list for 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:
        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)
        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')
      capabilities:
        # OPTIONAL METADATA

```

(continues on next page)

(continued from previous page)

```

host:
  properties:
    num_cpus: 2GHz
    mem_size: 2GB
os:
  properties:
    type: linux
    distribution: ubuntu
    version: 16.04
interfaces:
  Occopus:
    create:
      inputs:
        interface_cloud: cloudbroker
        endpoint_cloud: ADD_YOUR_ENDPOINT (e.g https://cola-prototype.
↪cloudbroker.com )

```

Under the **properties** section of a CloudBroker virtual machine definition these inputs are available.:

- **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: toasca.nodes.MiCADO.Occopus.EC2.Compute
      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)
      capabilities:
        # OPTIONAL METADATA
      host:
        properties:
          num_cpus: 2GHz
          mem_size: 2GB
      os:
        properties:

```

(continues on next page)

(continued from previous page)

```

    type: linux
    distribution: ubuntu
    version: 16.04
  interfaces:
    Occopus:
      create:
        inputs:
          interface_cloud: ec2
          endpoint_cloud: ADD_YOUR_ENDPOINT (e.g https://ec2.eu-west-1.amazonaws.
↪com)

```

Under the **properties** section of an EC2 virtual machine definition these inputs are available.:

- **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: toasca.nodes.MiCADO.Occopus.Nova.Compute
      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)
      capabilities:
        # OPTIONAL METADATA
        host:
          properties:
            num_cpus: 2GHz
            mem_size: 2GB
        os:
          properties:
            type: linux
            distribution: ubuntu
            version: 16.04
      interfaces:
        Occopus:

```

(continues on next page)

(continued from previous page)

```

    create:
      inputs:
        interface_cloud: nova
        endpoint_cloud: ADD_YOUR_ENDPOINT (e.g https://sztaki.cloud.mta.hu:5000/
↪v3)

```

Under the **properties** section of a Nova virtual machine definition these inputs are available.:

- **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 the application 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-KUBERNETES-APP:
      type: toasca.nodes.MiCADO.Container.Application.Docker
      ...
    YOUR-OTHER-KUBERNETES-APP:
      type: toasca.nodes.MiCADO.Container.Application.Docker
      ...
    YOUR_VIRTUAL_MACHINE:
      type: toasca.nodes.MiCADO.Occopus.<CLOUD_API_TYPE>.Compute
      ...

  policies:
  - scalability:
      type: toasca.policies.Scaling.MiCADO
      targets: [ YOUR_VIRTUAL_MACHINE ]
      properties:
        ...
  - scalability:
      type: toasca.policies.Scaling.MiCADO
      targets: [ YOUR-KUBERNETES-APP ]
      properties:

```

(continues on next page)

```

...
- scalability:
  type: toska.policies.Scaling.MiCADO
  targets: [ YOUR-OTHER-KUBERNETES-APP ]
  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 kubernetes-deployed applications.

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 referring 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. MiCADO supports Kubernetes service discovery to define such a source, simply pass the name of the app as defined in TOSCA and do not specify any port number
- **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 kubernetes 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 kubernetes deployment, 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 kubernetes service.

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

1.5 Tutorials

You can find some demo applications under the subdirectories of the ‘testing’ directory in the downloaded (and unzipped) installation package of MiCADO.

1.5.1 stressng

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

Note: make sure you have the `jq` tool installed required by the helper scripts.

- Step1: make a copy of the TOSCA file which is appropriate for your cloud - `stressng_<your_cloud>.yaml` - and name it `stressng.yaml` (ie. by issuing the command `cp stressng_cloudsigma.yaml stressng.yaml`)

- Step2: fill in the requested fields beginning with `ADD_YOUR_...`. These will differ depending on which cloud you are using.

Important: Make sure you create the appropriate firewall policy for the MiCADO workers as described [here!](#)

- In CloudSigma, for example, the `libdrive_id`, `public_key_id` and `firewall_policy` fields must be completed. Without these, CloudSigma does not have enough information to launch your worker nodes. All information is found on the CloudSigma Web UI. `libdrive_id` is the long alphanumeric string in the URL when a drive is selected under “Storage/Library”. `public_key_id` is under the “Access & Security/Keys Management” menu as **Uuid**. `firewall_policy` can be found when selecting a rule defined under the “Networking/Policies” menu. The following ports must be opened for MiCADO workers: *all inbound connections from MiCADO master*
- Step3: Update the parameter file, called `_settings`. You need the ip address for the MiCADO master and should name the application by setting the `APP_ID` ***the application ID can not contain any underscores (_)** You should also change the SSL user/password/port information if they are different from the default.
- Step4: run `1-submit-tosca-stressng.sh` to create the minimum number of MiCADO worker nodes and to deploy the Kubernetes Deployment including the stressng app defined in the `stressng.yaml` TOSCA description.
- Step4a: run `2-list-apps.sh` to see currently running applications and their IDs
- Step5: run `3-stress-cpu-stressng.sh 85` to stress the service and increase the CPU load. After a few minutes, you will see the system respond by scaling up virtual machines and containers to the maximum specified.
- Step6: run `3-stress-cpu-stressng.sh 10` to update the service and decrease the CPU load. After a few moments the system should respond by scaling down virtual machines and containers to the minimum specified.
- Step7: run `4-undeploy-stressng.sh` 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 queuing 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 perform 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.

Note: make sure you have the `jq` tool installed required by the helper scripts.

- 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.

Important: Make sure you create the appropriate firewall policy for the MiCADO workers as described [here!](#)

- 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.
- Step8a: Run `./4-list-running-apps.sh` to list the apps running under MiCADO.
- Step8b: run `query-services.sh` to see the details of docker services of your application
- Step8c: run `query-nodes.sh` to see the details of docker nodes hosting your application
- Step9: Run `./5-undeploy-cq-worker-from-micado.sh` to remove your application from MiCADO when all items are consumed.
- Step10: 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.

1.5.3 nginx

This application deploys a http server with nginx. The container features a built-in prometheus exporter for HTTP request metrics. The policy defined for this application scales up/down both nodes and the nginx service based on active http connections. wrk (`apt-get install wrk` | <https://github.com/wg/wrk>) is recommended for HTTP load testing.

Note: make sure you have the `jq` tool and `wrk` benchmarking app installed as these are required by the helper scripts. Best results for `wrk` are seen on multi-core systems.

- Step1: make a copy of the TOSCA file which is appropriate for your cloud - `nginx_<your_cloud>.yaml` - and name it `nginx.yaml`
- Step2: fill in the requested fields beginning with `ADD_YOUR_...` These will differ depending on which cloud you are using.

Important: Make sure you create the appropriate firewall policy for the MiCADO workers as described [here!](#)

- In CloudSigma, for example, the `libdrive_id`, `public_key_id` and `firewall_policy` fields must be completed. Without these, CloudSigma does not have enough information to launch your worker nodes. All information is found on the CloudSigma Web UI. `libdrive_id` is the long alphanumeric string in the URL when a drive is selected under “Storage/Library”. `public_key_id` is under the “Access & Security/Keys Management” menu as **Uuid**. `firewall_policy` can be found when selecting a rule defined under the “Networking/Policies” menu. The following ports must be opened for MiCADO workers: *all inbound connections from MiCADO master*
- Step3: Update the parameter file, called `_settings`. You need the ip address for the MiCADO master and should name the deployment by setting the `APP_ID`. ***the application ID can not contain any underscores (_)** The `APP_NAME` must match the name given to the application in TOSCA (default: **nginxapp**) You should also change the SSL user/password/port information if they are different from the default.

- Step4: run `1-submit-tosca-nginx.sh` to create the minimum number of MiCADO worker nodes and to deploy the Kubernetes Deployment including the nginx app defined in the `nginx.yaml` TOSCA description.
- Step4a: run `2-list-apps.sh` to see currently running applications and their IDs, as well as the ports forwarded to 8080 for accessing the HTTP service
- Step5: run `3-generate-traffic.sh` to generate some HTTP traffic. After thirty seconds or so, you will see the system respond by scaling up containers, and eventually virtual machines to the maximum specified.
- Step5a: the load test will finish after 10 minutes and the infrastructure will scale back down
- Step6: run `4-undeploy-nginx.sh` to remove the nginx deployment and all the MiCADO worker nodes

1.5.4 wordpress

This application deploys a wordpress blog, complete with MySQL server and a Network File Share for persistent data storage. It is a proof-of-concept and is **NOT** production ready. The policy defined for this application scales up/down both nodes and the wordpress frontend container based on network load. `wrk` (`apt-get install wrk` | <https://github.com/wg/wrk>) is recommended for HTTP load testing, but you can use any load generator you wish.

Note: make sure you have the `jq` tool and `wrk` benchmarking app installed as these are required by the helper scripts to force scaling. Best results for `wrk` are seen on multi-core systems.

- Step1: make a copy of the TOSCA file which is appropriate for your cloud - `wordpress_<your_cloud>.yaml` - and name it `wordpress.yaml`
- Step2: fill in the requested fields beginning with `ADD_YOUR_...` These will differ depending on which cloud you are using.

Important: Make sure you create the appropriate firewall policy for the MiCADO workers as described [here!](#)

- In CloudSigma, for example, the `libdrive_id`, `public_key_id` and `firewall_policy` fields must be completed. Without these, CloudSigma does not have enough information to launch your worker nodes. All information is found on the CloudSigma Web UI. `libdrive_id` is the long alphanumeric string in the URL when a drive is selected under “Storage/Library”. `public_key_id` is under the “Access & Security/Keys Management” menu as **Uuid**. `firewall_policy` can be found when selecting a rule defined under the “Networking/Policies” menu. The following ports must be opened for MiCADO workers: *all inbound connections from MiCADO master*
- Step3: Update the parameter file, called `_settings`. You need the ip address for the MiCADO master and should name the deployment by setting the `APP_ID`. ***the application ID can not contain any underscores (_)** The `FRONTEND_NAME`: must match the name given to the application in TOSCA (default: **wordpress**) You should also change the SSL user/password/port information if they are different from the default.
- Step4: run `1-submit-tosca-wordpress.sh` to create the minimum number of MiCADO worker nodes and to deploy the Kubernetes Deployments for the NFS and MySQL servers and the Wordpress frontend.
- Step4a: run `2-list-apps.sh` to see currently running applications and their IDs, as well as the nodePort open on the host for accessing the HTTP service (defaults to 30010)
- Step5: navigate to your wordpress blog (generally at `<master_node_ip>:30010`) and go through the setup tasks until you can see the front page of your blog
- Step6: run `3-generate-traffic.sh` to generate some HTTP traffic. After thirty seconds or so, you will see the system respond by scaling up a VM and containers to the maximum specified.
- Step6a: the load test will stop after 10minutes and the infrastructure will scale back down
- Step7: run `4-undeploy-wordpress.sh` to remove the wordpress deployment and all the MiCADO worker nodes

1.6 Release Notes

v0.7.2 (25 Feb 2019)

- add checking for minimal memory on micado master at deployment
- support private networks on cloudsigma
- support user-defined contextualisation
- support re-use across other container & cloud orchestrators in ADT
- new TOSCA to Kubernetes Manifest Adaptor
- add support for creating DaemonSets, Jobs, StatefulSets (with limited functionality) and standalone Pods
- add support for creating PersistentVolumes & PVClaims
- add support for specifying custom service details (NodePort, ClusterIP, etc.)
- minor improvements to Grafana dashboard
- support asynchronous calls through TOSCASubmitter API
- fix kubectrl error on MiCADO Master restart
- fix TOSCASubmitter rollback on errors
- fix TOSCASubmitter status & output display
- add support for encrypting master-worker communication
- automatically provision and revoke security credentials for worker nodes
- update default MTU to 1400 to ensure compatibility with OpenStack and AWS
- add Credential Store security enabler
- add Security Policy Manager security enabler
- add Image Integrity Verifier Security enabler
- add Crypto Engine security enabler
- add support for kubernetes secrets
- reimplement Credential Manager using the flask-users library

v0.7.1 (10 Jan 2019)

- Fix: Add SKIP back to Dashboard (defaults changed in v1.13.1)
- Fix: URL not found for Kubernetes manifest files
- Fix: Make sure worker node sets hostname correctly
- Fix: Don't update Kubernetes if template not changed
- Fix: Make playbook more idempotent
- Add Support for outputs via TOSCA ADT
- Add Kubernetes service discovery support to Prometheus
- Add new demo: nginx (HTTP request scaling)

v0.7.0 (12 Dec 2018)

- Introduce Kubernetes as the primary container orchestration engine

- Replace the swarm-visualiser with the Kubernetes Dashboard

v0.6.1 (15 Oct 2018)

- enable VM-only deployments
- add support for special characters in SSL credentials
- fix missing vm instance number reset at undeployment
- add option to disable auto-updates on worker nodes
- modify default launch-order of TOSCA adaptors
- add cloud-specific TOSCA templates and improve helper scripts for stressng
- flatten CPU scaling policies
- improve virtual machine build time
- fix Zorp starting dependency
- fix Docker login timing issue
- remove unnecessary port from docker compose file
- enable Prometheus DB export

v0.6.0 (10 Sept 2018)

- introduce documentation repository and host its content at <http://micado-scale.readthedocs.io>
- improve MiCADO master containers restart policy
- fix MTU issue in relation to Docker
- fix Occopus restart issue
- fix health-checking for Cloudbroker-AWS platform
- update host naming convention for worker and master nodes
- make wait-update task idempotent in ansible playbook
- fix issue with worker node deployment in EC2 clouds
- fix issue with user-defined Docker networks in OpenStack clouds
- make Submitter response message structure uniform
- add 'nodes' and 'services' query methods to REST API
- improve 'stressng' and 'cqueue' test helper scripts
- add more compose properties to custom TOSCA definition
- fix floating ip issues in the Dashboard component
- add new links to Dashboard to reflect the changes introduced by reverse proxying
- fix Dashboard to generate links based on the contents of the Host header to find the frontend URL automatically
- make consul security encryption based on generated random key instead of static key
- add reverse proxy, TLS encryption and application-level firewalling capabilities to the web interfaces exposed by the MiCADO master node
- add packet filtering for closing down non-public ports
- add systemd unit for MiCADO services

- update the ansible playbook to use the built-in service module for installing and handling MiCADO services
- update the documentation to reflect the changes after the introduction of reverse proxying
- add support for form-based authentication of exposed web services
- add COLA-themed login page
- add the Credential Manager component to store and handle web service users and passwords securely
- add support for provisioning a user to the Credential Manager via Ansible
- add support for user and admin roles in the Credential Manager
- add support for authorization of the web services based on user role
- add documentation about the Ansible Vault mechanism to protect sensitive deployment details
- add support for HTTP basic authentication for APIs
- add support for making the web interface's listening port configurable
- update the documentation of API calls in terms of authentication, encryption and reverse proxying
- add micadoctl tool for user and service management
- add HTTP method filter to firewall in order to control requests directed to containers
- add support for IPv6 exposure of services
- add IPv6 packet filtering

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