MiCADO Documentation

Attila Farkas

Oct 08, 2020

User Documentation

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.

CHAPTER 1

Introduction

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 Template 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 policies for scaling and monitoring both levels of the application.

The format of the Application Description Template 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 and Terraform (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 and CAdvisor as Kubernetes DaemonSets and the 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 in the Kubernetes Dashboard under the `micado-system` and `micado-worker` namespaces, or directly on the 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

A cloud interface supported by MiCADO

- EC2 (tested on Amazon and OpenNebula)
- Nova (tested on OpenStack)
- Azure (tested on Microsoft Azure)
- GCE (tested on Google Cloud)
- CloudSigma
- CloudBroker

MiCADO master (a virtual machine on a supported cloud)

- Ubuntu 16.04 or 18.04
- (Minimum) 2GHz CPU & 3GB RAM & 15GB DISK
- (Recommended) 2GHz CPU & 4GB RAM & 20GB DISK

Ansible Remote (the host where the Ansible Playbook is executed)
this could be the MiCADO Master itself, for a "local" execution of the playbook

- Ansible 2.8 or greater
- curl
- jq (to pretty-format API responses)
- wrk (to load test nginx & wordpress demonstrators)

Ansible

Note: Ansible in the Ubuntu APT repository is outdated and insufficient (at the time of writing this document)

To install Ansible on Ubuntu, 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 16.04, use this command:

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

To install wrk on other operating versions/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-0.9.1.tar.gz -L https://github.com/micado-scale/ansible-
↪micado/releases/download/v0.9.1/ansible-micado-0.9.1.tar.gz
tar -zxvf ansible-micado-0.9.1.tar.gz
cd ansible-micado-0.9.1/
```

Step 2: Specify cloud credential for instantiating MiCADO workers.

MiCADO master will use the credentials 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. It is recommended to fill only the section belonging to your target cloud.

NOTE If you are using Google Cloud, you must replace or fill the *credentials-gce.json* with your downloaded service account key file.

```
cp sample-credentials-gce.json credentials-gce.json
edit credentials-gce.json
```

It is possible to modify cloud credentials after MiCADO has been deployed, see the section titled **Update Cloud Credentials** further down this page

Optional: Added security

Credentials are stored in Kubernetes Secrets on the MiCADO Master. If you wish to keep the credential data in an secure format on the Ansible Remote as well, you can use the Ansible Vault mechanism to to achieve this. Simply create the above file using Vault with the following 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
```

Be sure to see the note about deploying a playbook with vault encrypted files in **Step 7**

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 the subject / CN ('micado-master' by default, but configurable in **micado-master.yml**).

  Two Subject Alternative Name (SAN) entries are also added by the configuration file at `roles/micado_master/start/templates/zorp/san.cnf`:

  - DNS: <*specified hostname*>
  - IP: <*specified IP*>

  The generated certificate file is located at: `/var/lib/micado/zorp/config/ssl.pem`

- 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:

Protocol	Port(s)	Service
TCP	443*	web listening port (configurable*)
TCP	22	SSH
TCP	2379-2380	etcd server
TCP	6443	kube-apiserver
TCP	10250-10252	kubelet, kube-controller, kube-scheduler
UDP	8285 & 8472	flannel overlay network
UDP	500 & 4500	IPSec

> **NOTE:** `[web_listening_port]` should match 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.
>
> **NOTE:** On some network configurations, for example where IPSec protocols **ESP (50)** and **AH (51)** are blocked, important network packets can get dropped in Master-Worker communications. This might be seen as Prometheus scrapes failing with the error **context deadline exceeded**, or Workers failing to join the Kubernetes cluster. To disable the IPSec tunnel securing Master-Worker communications, it can be stopped by appending **ipsec stop** to **runcmd** in the default worker node *cloud-init #cloud-config*.

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.yml for customisation.

```
cp sample-hosts.yml hosts.yml
edit hosts.yml
```

Edit the `hosts.yml` file to set the variables. The following parameters under the key **micado-target** can be updated:

- **ansible_host**: specifies the publicly reachable ip address of the target machine where you intend to build/deploy a MiCADO Master or build a MiCADO Worker. 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: Customize the deployment

A few parameters in *micado-master.yml* can be fine tuned before deployment. They are as follows:

- **disable_optimizer**: Setting this parameter to False enables the deployment of the Optimizer module, to perform more advanced scaling. Default is True.
- **disable_worker_updates**: Setting this parameter to False enables periodic software updates of the worker nodes. Default is True.
- **grafana_admin_pwd**: The string defined here will be the password for Grafana administrator.
- **web_listening_port**: Port number of the dasboard on MiCADO master. Default is 443.
- **web_session_timeout**: Timeout value in seconds for the Dashboard. Default is 600.
- **enable_occopus**: Install and enable Occopus for cloud orchestration. Default is True.
- **enable_terraform**: Install and enable Terraform for cloud orchestration. Default is False.

Note. MiCADO supports running both Occopus & Terraform on the same Master, if desired

Step 7: Start the installation of MiCADO master.

Run the following command to build and initalise a MiCADO master node on the empty VM you launched in Step 4 and pointed to in *hosts.yml* Step 5.

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

Optional: Build & Start Roles

Optionally, you can split the deployment of your MiCADO Master in two. The `build` tags prepare the node will all the necessary dependencies, libraries and images necessary for operation. The `start` tags intialise the cluster and all the MiCADO core components.

You can clone the drive of a **"built"** MiCADO Master (or otherwise make an image from it) to be reused again and again. This will greatly speed up the deployment of future instances of MiCADO.

Running the following command will `build` a MiCADO Master node on an empty Ubuntu VM.

```
ansible-playbook -i hosts.yml micado-master.yml --tags 'build'
```

You can then run the following command to `start` any **"built"** MiCADO Master node which will initialise and launch the core components for operation.

```
ansible-playbook -i hosts.yml micado-master.yml --tags 'start'
```

As a last measure of increasing efficiency, you can also `build` a MiCADO Worker node. You can then clone/snapshot/image the drive of this VM and point to it in your ADT descriptions. Before running this operation, Make sure the *hosts.yml* points to the empty VM where you intend to build the worker image. Adjust the values under the key **micado-target** as needed. The following command will `build` a MiCADO Worker node on an empty Ubuntu VM.

```
ansible-playbook -i hosts.yml build-micado-worker.yml
```

1.1.3 After deployment

Once the deployment has successfully finished, you can proceed with

- visiting the *Dashboard*
- using the restapi
- playing with the *Tutorials*
- creating your *Application Description Template (ADT)*

1.1.4 Update Cloud Credentials

It is possible to modify cloud credentials on an already deployed MiCADO Master. Simply make the necessary changes to the appropriate credentials file (using *ansible-vault* if desired) and then run the following playbook command:

```
ansible-playbook -i hosts.yml micado-master.yml --tags update-auth
```

1.1.5 Check the logs

All logs are now available via the Kubernetes Dashboard on the MiCADO Dashboard. You can navigate to them by changing the **namespace** to `micado-system` or `micado-worker` and then accessing the logs in the **Pods** section You can also 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 components. Scaling decisions, for example, can be inspected under `/var/log/micado/policykeeper`

1.1.6 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 Template (ADT)*

- 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 Application Description Template (ADT)

1.3.1 Overview

MiCADO executes applications described by Application Description Template. The ADT follows the TOSCA Specification and is described in detail in this section.

Main sections of the ADT

Top-level definitions

- **tosca_definitions_version**: `tosca_simple_yaml_1_2`.
- **imports**: 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 section

- **node_templates:** Definitions of the application containers (see **Specification of the Application**) and auxilary components such as a volume (see **Specification of Volumes**) and virtual machines (see **Specification of the Virtual Machine**)
- **policies:** Scaling & metric policies (see **Specification of Policies**)

Types section (optional)

This section is used to optionally define additional detailed types which can be referenced in the **topology_template** section to benefit from abstraction. Under **policy_types:** for example, complex scaling logic can be defined here, then referenced in the **policies** section above

Example of the overall structure of an ADT

```
tosca_definitions_version: tosca_simple_yaml_1_2

imports:
  - https://raw.githubusercontent.com/micado-scale/tosca/v0.9.1/micado_types.yaml

repositories:
  docker_hub: https://hub.docker.com/
  custom_registry: https://my-registry.mydomain.eu/

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

    YOUR-VOLUME:
      type: tosca.nodes.MiCADO.Container.Volume
      properties:
        ...
      interfaces:
        ...

    YOUR-VIRTUAL-MACHINE:
      type: tosca.nodes.MiCADO.<CLOUD_API_TYPE>.Compute
      properties:
        ...
      interfaces:
        ...
      capabilities:
        ...

  policies:
  - monitoring:
      type: tosca.policies.Monitoring.MiCADO
      properties:
        enable_container_metrics: true
        enable_node_metrics: false
  - scalability:
      type: tosca.policies.Scaling.MiCADO
      targets: [ YOUR-VIRTUAL-MACHINE ]
      properties:
        ...
  - scalability:
      type: tosca.policies.Scaling.MiCADO
      targets: [ YOUR-KUBERNETES-APP ]
      properties:
        ...
  - network:
```

(continues on next page)

(continued from previous page)

```
      type: tosca.policies.Security.MiCADO.Network.HttpProxy
      properties:
        encryption: true
        encryption_key: |
          -----BEGIN PRIVATE KEY-----
          ...
  - secret:
      type: tosca.policies.Security.MiCADO.Secret.KubernetesSecretDistribution
      properties:
        ...
```

1.3.2 Application

Please find some example-driven documentation on defining your Kubernetes application here.

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 container is described as a separate named node which references a **type** (more on types below). The definition of the most basic container consists of the following:

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

Properties

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 of the container (*list*)
- **args**: override the default entrypoint of container (*list*)
- **env**: list of required environment variables in format:
  - **name:**
  - **value:**
  - **valueFrom:** for use with ConfigMaps, see below
- **envFrom**: mostly for using ConfigMaps, see below
- **resource:**
  - **requests:**
    - **cpu**: CPU reservation, core components usually require 100m so assume 900m as a maximum
- **ports**: list of published ports to the host machine, you can specify these keywords in the style of a flattened (*Service*, *ServiceSpec* and *ServicePort* can all be defined at the same level - see 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 (generated if not specified)
  - **protocol**: the protocol for the port (defaults to: TCP)
  - **nodePort**: the port (30000-32767) to expose on the host (will create a nodePort Service unless type is explicitly set below)
  - **type**: the type of service for this port (defaults to: ClusterIP unless 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)
  - **hostPort**: the port on the node host to expose the pod at
  - **containerPort**: the port to target if exposing with hostPort

Environment variables can be loaded in from configuration data in Kubernetes ConfigMaps. This can be accomplished by using **envFrom:** with a list of **configMapRef:** to load all data from a ConfigMap into environment variables as seen here , or by using **env:** and **valueFrom:** with **configMapKeyRef:** to load specific values into environment variables as seen here .

Alternatively, ConfigMaps can be mounted as volumes as discussed here , in the same way other volumes are attached to a container, using the **requirements:** notation below. Also see the examples in **Specification of Configuration Data** below.

Artifacts

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.

Requirements

Under the **requirements** section you can define the virtual machine you want to host this particular app, restricting the container to run **only** on that VM. If you do not provide a host requirement, the container will run on any possible virtual machine. You can also attach a volume or ConfigMap to this app - the definition of volumes can be found in the next section. Requirements takes a list of map objects:

- **host:** name of your virtual machine as defined under node_templates
- **volume:**
  - **node:** name of your volume (or ConfigMap) as defined under node_templates
  - **relationship: !!**
    - **type:** `tosca.relationships.AttachesTo`
    - **properties:**
      - **location:** path in container
- **container:** name of a sidecar container defined as a `tosca.nodes.MiCADO.Container.Application.Docker` type under node_templates. The sidecar will share the Kubernetes Pod with the

main container (the sidecar should not be given an interface) **OR** name of an init container defined as a `tosca.nodes.MiCADO.Container.Application.Docker.Init` type under node_templates. The Pod will enter a ready state when the Init Container runs to completion and exits cleanly (ie. with a zero exit code)

If a relationship is not defined for a volume the path on container will be the same as the path defined in the volume (see Specification of Volumes). If no path is defined in the volume, the path defaults to */etc/micado/volumes* for a Volume or */etc/micado/configs* for a ConfigMap

Interfaces

Under the **interfaces** section you can define orchestrator specific options, to instruct MiCADO to use Kubernetes, we use the key **Kubernetes**. Fields under **inputs:** will be translated directly to a Kubernetes manifest so it is possible to use the full range of properties which Kubernetes offers as long as field names and syntax follow the Kubernetes documentation If **inputs:** is omitted a set of defaults will be used to create a Deployment

- **create**: *this key tells MiCADO to create a workload (Deployment/DaemonSet/Job/Pod etc. . .) for this container*
  - **inputs**: *top-level workload and workload spec options go here. . . two examples, for more see* translator documentation
    - **kind:** overwrite the workload type (defaults to Deployment)
    - **spec:**
      - **strategy:**
      - **type:** Recreate (kill pods then update instead of RollingUpdate)

Types

Through abstraction, it is possible to reference a pre-defined parent type and simplify the description of a container. These parent types can hide or reduce the complexity of more complex TOSCA constructs such as **artifacts** and **interfaces** by enforcing defaults or moving them to a simpler construct such as **properties**. Currently MiCADO supports the following types:

- **tosca.nodes.MiCADO.Container.Application.Docker** - The base and most common type for Docker containers in MiCADO. If the desired Docker container image is stored in DockerHub, the property **image:** can be used instead of defining **artifacts:**
- **tosca.nodes.MiCADO.Container.Application.Docker.Deployment** - As above, but orchestrated as a Kubernetes Deployment so that **interfaces:** is not required
- **tosca.nodes.MiCADO.Container.Application.Docker.DaemonSet** - As above, but for a Kubernetes DaemonSet
- **tosca.nodes.MiCADO.Container.Application.Docker.StatefulSet** - As above, but for a Kubernetes StatefulSet
- **tosca.nodes.MiCADO.Container.Application.Pod** - Creates an empty Pod. No properties are available, so to use this type a container must be defined and **assigned no interface** as type `tosca.nodes.MiCADO.Container.Application.Docker` and referenced under **requirements:** (more than one container can be referenced to run multiple containers in a single Pod)
- **tosca.nodes.MiCADO.Container.Application.Pod.Deployment** - As above, but a Kubernetes Deployment

Examples of the definition of a basic application

With *tosca.nodes.MiCADO.Container.Application.Docker* **and the Docker image in a custom repository**

```
YOUR-KUBERNETES-APP:
  type: tosca.nodes.MiCADO.Container.Application.Docker
  properties:
    name:
    command:
    args:
    env:
    ...
  artifacts:
    image:
      type: tosca.artifacts.Deployment.Image.Container.Docker
      file: YOUR_DOCKER_IMAGE
      repository: custom_registry
  requirements:
  - host: YOUR-VIRTUAL-MACHINE
  interfaces:
    Kubernetes:
      create:
        inputs:
        ...
```

With *tosca.nodes.MiCADO.Container.Application.Docker* and the Docker image in DockerHub

```
YOUR-KUBERNETES-APP:
  type: tosca.nodes.MiCADO.Container.Application.Docker
  properties:
    image: YOUR_DOCKER_IMAGE
    name:
    command:
    args:
    env:
    ...
  requirements:
  - host: YOUR-VIRTUAL-MACHINE
  interfaces:
    Kubernetes:
      create:
        inputs:
        ...
```

With *tosca.nodes.MiCADO.Container.Application.Docker.Deployment* and the Docker image in DockerHub

```
YOUR-KUBERNETES-APP:
  type: tosca.nodes.MiCADO.Container.Application.Docker.Deployment
  properties:
    image: YOUR_DOCKER_IMAGE
    name:
    command:
    args:
    env:
    ...
  requirements:
  - host: YOUR-VIRTUAL-MACHINE
```

Multiple containers in a single Pod, images in DockerHub

```
YOUR-KUBERNETES-APP:
```

(continues on next page)

(continued from previous page)

```
  type: tosca.nodes.MiCADO.Container.Application.Docker
  properties:
    image: YOUR_DOCKER_IMAGE
    name:
    command:
    ...

YOUR-OTHER-KUBERNETES-APP:
  type: tosca.nodes.MiCADO.Container.Application.Docker
  properties:
    image: YOUR_OTHER_DOCKER_IMAGE
    name:
    command:
    ...

YOUR-KUBERNETES-POD:
  type: tosca.nodes.MiCADO.Container.Pod.Kubernetes
  requirements:
  - container: YOUR-KUBERNETES-APP
  - container: YOUR-OTHER-KUBERNETES-APP
```

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 here . 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 nested within *topology_template*) you can define an output to retrieve from Kubernetes via the adaptor. Currently, only port info is obtainable.

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

1.3.3 Volume

Volumes are defined at the same level as virtual machines and containers, and are then connected to containers using the **requirements:** notation discussed above in the container spec. Some examples of attaching volumes will follow.

Interfaces

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

- **create**: *this key tells MiCADO to create a persistent volume and claim*
  - **inputs**: persistent volume specific spec options… here are two popular examples, see Kubernetes volumes for more * **spec:**
    - **nfs:**
      - **server:** IP of NFS server
      - **path:** path on NFS share

  * **hostPath:**
    - **path:** path on host

Types

Through abstraction, it is possible to reference a pre-defined parent type and simplify the description of a volume. These parent types can hide or reduce the complexity of more complex TOSCA constructs such as **interfaces** by enforcing defaults or moving them to a simpler construct such as **properties**. Currently MiCADO supports the following volume types:

- **tosca.nodes.MiCADO.Container.Volume** - The base and most common type for volumes in MiCADO. It is necessary to define further fields under **interfaces:**
- **tosca.nodes.MiCADO.Container.Volume.EmptyDir** - Creates a EmptyDir persistent volume (PV) and claim (PVC) in Kubernetes
- **tosca.nodes.MiCADO.Container.Volume.HostPath** - Creates a HostPath PV and PVC. Define the path on host as **path:** under **properties:**
- **tosca.nodes.MiCADO.Container.Volume.NFS** - Creates an NFS PV and PVC. Define the path and server IP as **path:** and **server:** under **properties:**
- **tosca.nodes.MiCADO.Container.Volume.GlusterFS** - Creates a GlusterFS PV and PVC. Define path, endpoint and readOnly flag as **path:**, **endpoints:**, and **readOnly:** under **properties:**

Examples of the definition of a basic volume

With *tosca.nodes.MiCADO.Container.Volume*

```
YOUR-VOLUME:
  type: tosca.nodes.MiCADO.Container.Volume
  interfaces:
    Kubernetes:
      create:
        inputs:
          spec:
            nfs:
              path: /exports
              server: 10.96.0.1

YOUR-KUBERNETES-APP:
  type: tosca.nodes.MiCADO.Container.Application.Docker.Deployment
  properties:
    ...
  requirements:
  - volume:
      node: YOUR-VOLUME
      relationship:
        type: tosca.relationships.AttachesTo
        properties:
          location: /tmp/container/mount/point
```

Another example with *tosca.nodes.MiCADO.Container.Volume*

Here, no **relationship** is defined under **requirements** so the path defined by the volume */etc/mypath* will be used as the container mount point

```
YOUR-VOLUME:
  type: tosca.nodes.MiCADO.Container.Volume
  interfaces:
    Kubernetes:
      create:
        inputs:
          spec:
            hostPath:
              path: /etc/mypath

YOUR-KUBERNETES-APP:
  type: tosca.nodes.MiCADO.Container.Application.Docker.Deployment
  properties:
    ...
  requirements:
  - volume: YOUR-VOLUME
```

With *tosca.nodes.MiCADO.Container.Volume.EmptyDir*

```
YOUR-VOLUME:
  type: tosca.nodes.MiCADO.Container.Volume.EmptyDir

YOUR-KUBERNETES-APP:
  type: tosca.nodes.MiCADO.Container.Application.Docker.Deployment
  properties:
    ...
  requirements:
  - volume:
      node: YOUR-VOLUME
      relationship:
        type: tosca.relationships.AttachesTo
        properties:
          location: /tmp/container/mount/point
```

With *tosca.nodes.MiCADO.Container.Volume.NFS*

```
YOUR-VOLUME:
  type: tosca.nodes.MiCADO.Container.Volume.NFS
  properties:
    path: /exports
    server: 10.96.0.1

YOUR-KUBERNETES-APP:
  type: tosca.nodes.MiCADO.Container.Application.Docker.Deployment
  properties:
    ...
  requirements:
  - volume:
      node: YOUR-VOLUME
      relationship:
        type: tosca.relationships.AttachesTo
        properties:
          location: /tmp/container/mount/point
```

1.3.4 Configuration Data

Configuration data (a Kubernetes **ConfigMap**) are to be defined at the same level as virtual machines, containers and volumes and then loaded into environment variables, or mounted as volumes in the definition of containers as discussed in **Specification of the Application**. Some examples of using configurations will follow at the end of this section.

Interfaces

Currently MiCADO only supports the definition of configuration data as Kubernetes ConfigMaps. Under the **interfaces** section of this type use the key **Kubernetes:** to instruct MiCADO to create a ConfigMap.

- **create**: *this key tells MiCADO to create a ConfigMap*
  - **inputs**: ConfigMap fields to be overwritten, for more detail see ConfigMap
    - **data:** for UTF-8 byte values
    - **binaryData:** for byte values outside of the UTF-8 range

Types

Through abstraction, it is possible to reference a pre-defined parent type and simplify the description of a ConfigMap. These parent types can hide or reduce the complexity of more complex TOSCA constructs such as **interfaces** by enforcing defaults or moving them to a simpler construct such as **properties**. Currently MiCADO supports the following ConfigMap types:

- **tosca.nodes.MiCADO.Container.Config** - The base and most common type for configuration data in MiCADO. It is necessary to define further fields under **interfaces:** as indicated above
- **tosca.nodes.MiCADO.Container.Config.ConfigMap** - Defaults to a Kubernetes interface and abstracts the inputs to properties. Define the data or binary data fields as **data:** and **binaryData:** under **properties:**

Examples of the definition of a simple ConfigMap

Single ENV var with *tosca.nodes.MiCADO.Container.Config*

Here the environment variable MY_COLOUR is assigned a value from the ConfigMap

```
YOUR-CONFIG:
  type: tosca.nodes.MiCADO.Container.Config
  interfaces:
    Kubernetes:
      create:
        inputs:
          data:
            color: purple
            how: fairlyNice
            textmode: "true"

YOUR-KUBERNETES-APP:
  type: tosca.nodes.MiCADO.Container.Application.Docker.Deployment
  properties:
    env:
    - name: MY_COLOUR
      valueFrom:
```

(continues on next page)

(continued from previous page)

```
        configMapKeyRef:
          name: YOUR-CONFIG
          key: color
```

All ENV vars with *tosca.nodes.MiCADO.Container.ConfigMap*

Here an environment variable is created for each key (this becomes the variable name) and value pair in the ConfigMap

```
YOUR-CONFIG:
  type: tosca.nodes.MiCADO.Container.Config.Kubernetes
  properties:
    data:
      color: purple
      how: fairlyNice
      textmode: "true"

YOUR-KUBERNETES-APP:
  type: tosca.nodes.MiCADO.Container.Application.Docker.Deployment
  properties:
    envFrom:
    - configMapRef:
        name: YOUR-CONFIG
```

A volume with *tosca.nodes.MiCADO.Container.Config.Kubernetes*

Here a volume at /etc/config is populated with three files named after the ConfigMap key names and containing the matching values

```
YOUR-CONFIG:
  type: tosca.nodes.MiCADO.Container.Config.Kubernetes
  properties:
    data:
      color: purple
      how: fairlyNice
      textmode: "true"

YOUR-KUBERNETES-APP:
  type: tosca.nodes.MiCADO.Container.Application.Docker.Deployment
  requirements:
  - volume:
      node: YOUR-CONFIG
      relationship:
        type: tosca.relationships.AttachesTo
        properties:
          location: /etc/config
```

1.3.5 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 seven different cloud interfaces: CloudSigma, CloudBroker, EC2, Nova, Azure, OCI and GCE. MiCADO supports multiple virtual machine "sets" which can be restricted to host only specific containers (defined in the requirements section of the container specification). At the moment multi-cloud support is in alpha stage, so only certain combinations of different cloud service providers will work.

NOTE Underscores are not permitted in virtual machine names (ie TOSCA node names). Names should also begin and end with an alphanumeric.

The following ports and protocols should be enabled on the virtual machine acting as MiCADO worker, replacing [exposed_application_ports] with ports you wish to expose on the host:

Protocol	Port(s)	Service
TCP	30000-32767*	exposed application node ports (configurable*)
TCP	22	SSH
TCP	10250	kubelet
UDP	8285 & 8472	flannel overlay network

The following subsections details how to configure them.

General

Here is the basic look of a Virtual Machine node inside an ADT:

```
SAMPLE-VIRTUAL-MACHINE:
  type: tosca.nodes.MiCADO...Compute
    properties:
      <CLOUD-SPECIFIC VM PROPERTIES>
      context:
        insert: true
        cloud_config: |
          runcmd:
          - <some_command_here>

    capabilities:
      host:
        properties:
          num_cpus: 2
          mem_size: 4 GB
      os:
        properties:
          type: linux
          distribution: ubuntu
          version: 18.04

    interfaces:
      Occopus:
        create:
          inputs:
            endpoint: https://mycloud/api/v1
```

The **properties** section is **REQUIRED** and contains the necessary properties to provision the virtual machine and vary from cloud to cloud. Properties for each cloud are detailed further below.

Cloud Contextualisation

It is possible to provide custom configuration of the deployed nodes via cloud-init scripts . MiCADO relies on a cloud-init config to join nodes as workers to the cluster, so it is recommended to only add to the default config, except for certain cases.

The **context** key is supported by all the cloud compute node definitions below. New cloud-init configurations should be defined in **cloud_config** and one of **append** or **insert** should be set to *true* to avoid overwriting the default cloud-init config for MiCADO.

- Setting **append** to true will add the newly defined configurations to the end of the default cloud-init config
- Setting **insert** to true will add the newly defined configurations to the start of the default cloud-init config, before the MiCADO Worker is fully initialised

The **capabilities** sections for all virtual machine definitions that follow are identical and are **ENTIRELY OPTIONAL**. They are ommited in the cloud-specific examples below. They are filled with the following metadata to support human readability:

- **num_cpus** under *host* is an integer specifying number of CPUs for the instance type
- **mem_size** under *host* is a readable string with unit 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 we use either **Occopus** or **Terraform** based on suitability.

- **create**: *this key tells MiCADO to create the VM using Occopus/Terraform*
  - **inputs**: Extra settings to pass to Occopus or Terraform
    - **endpoint:** the endpoint API of the cloud (always required for Occopus, sometimes required for Terraform)

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*.

Currently, only **Occopus** has support for CloudSigma, so Occopus must be enabled as in *Step 6: Customize the deployment*, and the interface must be set to Occopus as in the example below.

```
YOUR-VIRTUAL-MACHINE:
  type: tosca.nodes.MiCADO.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)
        ip_v4_conf:
          conf: dhcp

    interfaces:
      Occopus:
        create:
          inputs:
            endpoint: ADD_YOUR_ENDPOINT (e.g for cloudsigma https://zrh.cloudsigma.
↪com/api/2.0 )
```

(continues on next page)

(continued from previous page)

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*.

Currently, only **Occopus** has support for CloudBroker, so Occopus must be enabled as in *Step 6: Customize the deployment* and the interface must be set to Occopus as in the example below.

```
YOUR-VIRTUAL-MACHINE:
  type: tosca.nodes.MiCADO.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')

    interfaces:
      Occopus:
        create:
          inputs:
            endpoint: 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*.

Terraform supports provisioning on AWS EC2, and **Occopus** supports both AWS EC2 and OpenNebula EC2. To use Terraform, enable it as described in *Step 6: Customize the deployment* and adjust the interfaces section accordingly.

```
YOUR-VIRTUAL-MACHINE:
  type: tosca.nodes.MiCADO.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)

  interfaces:
    Occopus:
      create:
        inputs:
          endpoint: 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.

Under the **interfaces** section of an EC2 virtual machine definition, the **endpoint** input is required by Occopus as seen in the example above.

For Terraform the endpoint is discovered automatically based on region. To customise the endpoint pass the **endpoint** input in interfaces.

```
...
  interfaces:
    Terraform:
      create:
        inputs:
          endpoint: ADD_YOUR_ENDPOINT (e.g https://my-custom-endpoint/api)
```

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*.

Both **Occopus and Terraform** support Nova provisioning. To use Terraform, enable it as described in *Step 6: Customize the deployment* and adjust the interfaces section accordingly.

```
YOUR-VIRTUAL-MACHINE:
  type: tosca.nodes.MiCADO.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)

  interfaces:
    Occopus:
      create:
        inputs:
          endpoint: 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 id of the desired flavor for the VM.
- **tenant_name** is the name of the Tenant or Project to login with.
- **user_domain_name** is the domain name where the user is located.
- **availability_zone** is the availability zone in which to create the VM.
- **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.
- **config_drive** (Terraform only) is a boolean to enable use of a configuration drive for metadata storage.

Under the **interfaces** section of a Nova virtual machine definition, the **endpoint** input (v3 Identity service) is required as seen in the example above.

For Terraform the endpoint should also be passed as **endpoint** in inputs. Depending on the configuration of the OpenStack cluster, it may be necessary to provide **network_name** in addition to the ID.

```
...
  interfaces:
    Terraform:
      create:
        inputs:
          endpoint: ADD_YOUR_ENDPOINT (e.g https://sztaki.cloud.mta.hu:5000/v3)
          network_name: ADD_YOUR_NETWORK_NAME (e.g mynet-default)
```

Authentication in OpenStack is supported by MiCADO in two ways:

The default method is authenticating with the same credentials used to access the OpenStack WebUI by providing the **username** and **password** fields in *credentials-cloud-api.yml* during *Step 2: Specify cloud credential for instantiating MiCADO workers.*

The other option is with Application Credentials For this method, provide **application_credential_id** and **applicaiton_credential_secret** in *credentials-cloud-api.yml*. If these fields are filled, **username** and **password** will be ignored.

Azure

To instantiate MiCADO workers on a cloud through Azure interface, please use the template below. Currently, only **Terraform** has support for Azure, so Terraform must be enabled as in *Step 6: Customize the deployment*, and the interface must be set to Terraform as in the example below.

MiCADO supports Windows VM provisioning in Azure. To force a Windows VM, simply **DO NOT** pass the **public_key** property and **set the image** to a desired WindowsServer Sku (2016-Datacenter). Refer to this Sku list

```
YOUR-VIRTUAL-MACHINE:
  type: tosca.nodes.MiCADO.Azure.Compute
  properties:
    resource_group: ADD_YOUR_RG_HERE (e.g. my-test)
    virtual_network: ADD_YOUR_VNET_HERE (e.g. my-test-vnet)
    subnet: ADD_YOUR_SUBNET_HERE (e.g. default)
    network_security_group: ADD_YOUR_NSG_HERE (e.g. my-test-nsg)
    size: ADD_YOUR_ID_HERE (e.g. Standard_B1ms)
    image: ADD_YOUR_IMAGE_HERE (e.g. 18.04.0-LTS or 2016-Datacenter)
    public_key: ADD_YOUR_MINIMUM_2048_KEY_HERE (e.g. ssh-rsa ASHFF...)
    public_ip: [OPTIONAL] BOOLEAN_ENABLE_PUBLIC_IP (e.g. true)

  interfaces:
    Terraform:
      create:
```

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

- **resource_group** specifies the name of the resource group in which the VM should exist.
- **virtual_network** specifies the virtual network associated with the VM.
- **subnet** specifies the subnet associated with the VM.
- **network_security_group** specifies the security settings for the VM.
- **vm_size** specifies the size of the VM.
- **image** specifies the name of the image.
- **public_ip [OPTIONAL]** Associate a public IP with the VM.
- **key_data** The public SSH key (minimum 2048-bit) to be associated with the instance. **Defining this property forces creation of a Linux VM. If it is not defined, a Windows VM will be created**

Under the **interfaces** section of a Azure virtual machine definition no specific inputs are required, but **Terraform: create:** should be present

Authentication in Azure is supported by MiCADO in two ways:

The first is by setting up a Service Principal and providing the required fields in *credentials-cloud-api.yml* during *Step 2: Specify cloud credential for instantiating MiCADO workers.*

The other option is by enabling a System-Assigned Managed Identity on the **MiCADO Master VM** and then modify access control of the **current subscription** to assign the role of **Contributor** to the **MiCADO Master VM**

GCE

To instantiate MiCADO workers on a cloud through Google interface, please use the template below. Currently, only **Terraform** has support for Google Cloud, so Terraform must be enabled as in *Step 6: Customize the deployment*, and the interface must be set to Terraform as in the example below.

```
YOUR-VIRTUAL-MACHINE:
  type: tosca.nodes.MiCADO.GCE.Compute
  properties:
    region: ADD_YOUR_ID_HERE (e.g. us-west1)
    zone: ADD_YOUR_ID_HERE (e.g. us-west1-a)
    project: ADD_YOUR_ID_HERE (e.g. PGCE)
    machine_type: ADD_YOUR_ID_HERE (e.g. n1-standard-2)
    image: ADD_YOUR_ID_HERE (e.g.  ubuntu-os-cloud/ubuntu-1804-lts)
    network: ADD_YOUR_ID_HERE (e.g. default)
    ssh-keys: ADD_YOUR_ID_HERE (e.g. ssh-rsa AAAB3N...)

  interfaces:
    Terraform:
      create:
```

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

- **project** is the project to manage the resources in.
- **image** specifies the image from which to initialize the VM disk.
- **region** is the region that the resources should be created in.
- **machine_type** specifies the type of machine to create.
- **zone** is the zone that the machine should be created in.
- **network** is the network to attach to the instance.
- **ssh-keys** sets the public SSH key to be associated with the instance.

Under the **interfaces** section of a GCE virtual machine definition no specific inputs are required, but **Terraform: create:** should be present

Authentication in GCE is done using a service account key file in JSON format. You can manage the key files using the Cloud Console. The steps to retrieve the key file is as follows :

- Open the **IAM & Admin** page in the Cloud Console.
- Click **Select a project**, choose a project, and click **Open**.
- In the left nav, click **Service accounts**.
- Find the row of the service account that you want to create a key for. In that row, click the **More** button, and then click **Create key**.
- Select a **Key type** and click **Create**.

OCI

To instantiate MiCADO workers on a cloud through Oracle interface, please use the template below. Currently, only **Terraform** has support for Oracle, so Terraform must be enabled as in *Step 6: Customize the deployment*, and the interface must be set to Terraform as in the example below under `context`.

Note that OCI's Ubuntu VM images feature a number of strict `iptables` rules, which will restrict normal communnication between worker nodes and the MiCADO Master. To resolve this issue, it is important to include the VM contextualisation commands that can be seen in the example below.

```
YOUR-VIRTUAL-MACHINE:
  type: tosca.nodes.MiCADO.OCI.Compute
  properties:
    region: <REGION_NAME> (e.g. uk-london-1)
    availability_domain: <AVAILABILITY_DOMAIN> (e.g. lVvK:UK-LONDON-1-AD-1)
    compartment_id: <COMPARTMENT_OCID> (e.g ocid1.tenancy.oc1..aaa)
    shape: <VM_TYPE_NAME> (e.g. VM.Standard.E2.1)
    source_id: <VM_IMAGE_OCID> (e.g ocid1.image.oc1.uk-london-1.aaa)
    subnet_id: <SUBNET_OCID> (e.g ocid1.subnet.oc1.uk-london-1.aaa)
    network_security_group: <NETWORK_SECURITY_GROUP_OCID> (e.g ocid1.
↪networksecuritygroup.oc1.uk-london-1.aaa)
    ssh_keys: ADD_YOUR_ID_HERE (e.g. ssh-rsa AAAB3N...)
    context:
      insert: true
      cloud_config: |
        runcmd:
        - iptables -D INPUT -j REJECT --reject-with icmp-host-prohibited
        - iptables -D FORWARD -j REJECT --reject-with icmp-host-prohibited

  interfaces:
    Terraform:
      create:
```

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

- **availability_domain** is the availability domain of the instance.
- **source_id** specifies the OCID of an image from which to initialize the VM disk.
- **region** is the region that the resources should be created in.
- **shape** specifies the type of machine to create.
- **compartment_id** is the OCID of the compartment.
- **subnet_id** is the OCID of the subnet to create the VNIC in.
- **network_security_group** specifies the OCID of the network security settings for the VM.
- **ssh_keys** sets the public SSH key to be associated with the instance.

Under the **interfaces** section of an OCI virtual machine definition no specific inputs are required, but **Terraform: create:** should be present.

Authentication in OCI is supported by MiCADO in two ways:

> The first is by setting up an Instance Principal based authentication on the **MiCADO Master VM** by creating suitable ‘Dynamic Group and Policies <https://docs.cloud.oracle.com/en-us/iaas/Content/Identity/Tasks/callingservicesfrominstances.htm>‘__ associated with it.
>
> The other option is by enabling an API Key based authentication on the **MiCADO Master VM** and providing the required fields in *credentials-cloud-api.yml* during *Step 2: Specify cloud credential for*

instantiating MiCADO workers.

Types

Through abstraction, it is possible to reference a pre-defined type and simplify the description of a virtual machine. Currently MiCADO supports these additional types for CloudSigma, but more can be written:

- **tosca.nodes.MiCADO.EC2.Compute.Terra** - Orchestrates with Terraform on eu-west-2, overwrite region_name under **properties** to change region
- **tosca.nodes.MiCADO.CloudSigma.Compute.Occo** - Automatically orchestrates on Zurich with Occopus. There is no need to define further fields under **interfaces:** but Zurich can be changed by overwriting **endpoint** under **properties:**
- **tosca.nodes.MiCADO.CloudSigma.Compute.Occo.small** - As above but creates a 2GHz/2GB node by default
- **tosca.nodes.MiCADO.CloudSigma.Compute.Occo.big** - As above but creates a 4GHz/4GB node by default
- **tosca.nodes.MiCADO.CloudSigma.Compute.Occo.small.NFS** - As *small* above but installs NFS dependencies by default

Example definition of a VM using abstraction

With *tosca.nodes.MiCADO.CloudSigma.Compute.Occo.small* **and omitting capabilities metadata**

```
YOUR-VIRTUAL-MACHINE:
  type: tosca.nodes.MiCADO.CloudSigma.Compute.Occo.small
    properties:
      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)
        ip_v4_conf:
          conf: dhcp
```

1.3.6 Monitoring Policy

Metric collection is now disabled by default. The basic exporters from previous MiCADO versions can be enabled through the monitoring policy below. If the policy is omitted, or if one property is left undefined, then the relevant metric collection will be disabled.

```
policies:
- monitoring:
    type: tosca.policies.Monitoring.MiCADO
    properties:
      enable_container_metrics: true
      enable_node_metrics: true
```

1.3.7 Scaling Policy

Basic scaling

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. You can attach different policies to different containers or virtual machines, though a new policy should exist for each. 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-VIRTUAL-MACHINE:
      type: tosca.nodes.MiCADO.<CLOUD_API_TYPE>.Compute
      ...
    YOUR-OTHER-VIRTUAL-MACHINE:
      type: tosca.nodes.MiCADO.<CLOUD_API_TYPE>.Compute
      ...
    YOUR-KUBERNETES-APP:
      type: tosca.nodes.MiCADO.Container.Application.Docker
      ...
    YOUR-OTHER-KUBERNETES-APP:
      type: tosca.nodes.MiCADO.Container.Application.Docker
      ...

  policies:
  - scalability:
     type: tosca.policies.Scaling.MiCADO
     targets: [ YOUR-VIRTUAL-MACHINE ]
     properties:
       ...
  - scalability:
     type: tosca.policies.Scaling.MiCADO
     targets: [ YOUR-OTHER-VIRTUAL-MACHINE ]
     properties:
       ...
  - scalability:
     type: tosca.policies.Scaling.MiCADO
     targets: [ YOUR-KUBERNETES-APP ]
     properties:
       ...
  - scalability:
     type: tosca.policies.Scaling.MiCADO
     targets: [ YOUR-OTHER-KUBERNETES-APP ]
     properties:
       ...
```

The scaling policies are evaluated periodically. In every turn, the virtual machine level scaling policies are 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. Scaling rule is specified by (a short) Python code. The code can refer to/use the variables. 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 returning a number'
        MYLISTOFSTRING: ['Prometheus query returning a list of strings as tags',
↪'tagname as filter']
        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. 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` or `MYLISTOFSTRING` above)
- **alerts** subsection enables the utilization 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_nodes_todrop: the ids or ip addresses of the nodes to be dropped in case of downscaling **NOTE MiCADO-Terraform supports private IPs on Azure or AWS EC2 only**
  * 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 the virtual machine, the name of the variable to be updated is `m_nodes_todrop`;the variable must be filled with list of ids or ip addresses and as an effect the valid nodes will be dropped. The variable `m_node_count` should not be modified in case of node dropping, MiCADO will update it automatically.
- 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 debugging purposes, the following support is provided:

- `m_dryrun` can be specified in the **constant** as list of components towards which the communication is disabled. It has the following syntax: m_dryrun: ["prometheus","occopus","k8s","optimizer"] Use this feature with caution!
- the standard output of the python code defined by the user under the scaling rule section is collected in a separate log file stored under the policy keeper log directory. It can also be used for debugging purposes.

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

Optimiser-based scaling

For implementing more advanced scaling policies, it is possible to utilize the built-in Optimiser in MiCADO. The role of the Optimiser is to support decision making in calculating the number of worker nodes (virtual machines) i.e. to scale the nodes to the optimal level. Optimiser is implemented using machine learning algorithm aiming to learn the relation between various metrics and the effect of scaling events. Based on this learning, the Optimiser is able to calculate and advise on the necessary number of virtual machines.

Current limitations

- only web based applications are supported
- only one of the node sets can be supported
- no container scaling is supported

Optimiser can be utilised based on the following principles

- User specifies a so-called target metric with its associated minimum and maximum thresholds. The target metric is a monitored Prometheus expression for which the value is tried to be kept between the two thresholds by the Optimiser with scaling advices.
- User specifies several so-called input metrics which represent the state of the system correlating to the target variable
- User specifies several initial settings (see later) for the Optimiser
- User submits the application activating the Optimiser through the ADT
- Optimiser starts with the 'training' phase during which the correlations are learned. During the training phase artificial load must be generated for the web application and scaling activities must be performed (including extreme values) in order to present all possible situations for the Optimiser. During the phase, Optimiser continuously monitors the input/target metrics and learns the correlations.
- When correlations are learnt, Optimiser turns to 'production' phase during which advice can be requested from the Optimiser. During this phase, Optimiser returns advice on request, where the advice contains the number of virtual machines (nodes) to be scaled to. During the production phase, the Optimiser continues its learning activity to adapt to the new situations.

Activation of the Optimiser Optimiser must be enabled at deployment time. By default it is disabled. Once it is enabled and deployed, it can be driven through the scaling policy in subsections "constants" and "queries". Each parameter relating to the Optimiser must start with the "m_opt_" string. In case no variable name with this prefix is found in any sections, Optimiser is not activated.

Initial settings for the Optimiser Parameters for initial settings are defined under the "constants" section and their name must start with the "m_opt_init_" prefix. These parameters are as follows:

- **m_opt_init_knowledge_base** is a parameter which specifies the way how the knowledge base must be built under the Optimiser. When defined as "build_new", Optimiser empties its knowledge base and starts building a new knowledge i.e. starts learning the correlations. When using the "use_existing" value, the knowledge is kept and continued building further. Default is "use_existing".
- **m_opt_init_training_samples_required** defines how many sample of the metrics must be collected by the Optimiser before start learning the correlations. Default is 300.
- **m_opt_init_max_upscale_delta** specifies the maximum change in number of node for an upscaling advice. Default is 6.
- **m_opt_init_max_downscale_delta** specifies the maximum change in number of node for a downscaling advice. Default is 6.
- **m_opt_init_advice_freeze_interval** specifies how many seconds must elapse before the Optimiser advises a different number of node. Can be used to mitigate the frequency of scaling. Defaults to 0.

Definition of input metrics for the Optimizer Input metrics must be specified for the Optimiser under the "queries" subsection to perform the training i.e. learning the correlations. Each parameter must start with the "m_opt_input_" prefix, e.g. m_opt_input_CPU. The following two pieces of variable must be specified for the web application:

- **m_opt_input_AVG_RR** should specify the average request rate of the web server.
- **m_opt_input_SUM_RR** should specify the summary of request rate of the web server.

Definition of the target metric for the Optimizer Target metric is a continuously monitored parameter that must be kept between thresholds. To specify it, together with the thresholds, "m_opt_target_" prefix must be used. These three parameter must be defined under the "queries" sections. They are as follows:

- **m_opt_target_query_MYTARGET** specifies the prometheus query for the target metric called MYTARGET.
- **m_opt_target_minth_MYTARGET** specifies the value above which the target metric must be kept.

- **m_opt_target_maxth_MYTARGET** specifies the value below which the target metric must be kept.

Requesting scaling advice from the Optimizer In order to receive a scaling advice from the Optimiser, the method **m_opt_advice()** must be invoked in the scaling_rule section of the node.

IMPORTANT! Minimum and maximum one node must contain this method invocation in its scaling_rule section for proper operation!

The **m_opt_advice()** method returns a python dictionary containing the following fields:

- **valid** stores True/False value indicating whether the advise can be considered or not.
- **phase** indicates whether the Optimiser is in "training" or "production" phase.
- **vm_number** represents the advise for the target number of nodes to scale to.
- **reliability** represents the goodness of the advice with a number between 0 and 100. The bigger the number is the better/more reliable the advice is.
- **error_msg** contains the error occured in the Optimiser. Filled when valid is False.

1.3.8 Network policy

There are six types of MiCADO network security policy.

- tosca.policies.Security.MiCADO.Network.Passthrough: Pass through network policy. Specifies no additional filtering, no application-level firewall on the nodes.
- tosca.policies.Security.MiCADO.Network.L7Proxy: Apply application-level firewall; can provide TLS control. No protocol enforcement.

```
properties:
  encryption:
    type: boolean
    description: Specifies if encryption should be used
    required: true
  encryption_key:
    type: string
    description: The key file for TLS encryption as unencrypted .PEM
    required: false
  encryption_cert:
    type: string
    description: The cert file for TLS encryption as .PEM
    required: false
  encryption_offload:
    type: string
    description: Controls whether connection should be re-encrypted server side
    required: false
  encryption_cipher:
    type: string
    description: Specifies allowed ciphers client side during TLS handshake
    required: false
```

- tosca.policies.Security.MiCADO.Network.SmtpProxy: Enforce SMTP protocol, can provide TLS control.

```
properties:
  relay_check:
    type: boolean
    description: Toggle relay checking
    required: true
```

(continues on next page)

(continued from previous page)

```
  permit_percent_hack:
    type: boolean
    description: Allow the % symbol in the local part of an email address
    required: false
  error_soft:
    type: boolean
    description: Return a soft error when recipient filter does not match
    required: false
  relay_domains:
    type: list
    description: Domain mails are accepted for use postfix style lists
    required: false
  permit_exclamation_mark:
    type: boolean
    description: Allow the ! symbol in the local part of an email address
    required: false
  relay_domains_matcher_whitelist:
    type: list
    description: Domains mails accepted based on list of regex (precedence)
    required: false
  relay_domains_matcher_blacklist:
    type: list
    description: Domain mails rejected based on list of regular expressions
    required: false
  sender_matcher_whitelist:
    type: list
    description: Sender addresses accepted based on list of regex (precedence)
    required: false
  sender_matcher_blacklist:
    type: list
    description: Sender addresses rejected based on list of regex
    required: false
  recipient_matcher_whitelist:
    type: list
    description: Recipient addresses accepted based on list of regex (precedence)
    required: false
  recipient_matcher_blacklist:
    type: list
    description: Recipient addresses rejected based on list of regex
    required: false
  autodetect_domain_from:
    type: string
    description: Let Zorp autodetect firewall domain name and write to received line
    constraints:
      - valid_values: ["mailname", "fqdn"]
    required: false
  append_domain:
    type: string
    description: Domain to append to email addresses which do not specify a domain
    required: false
  permit_omission_of_angle_brackets:
    type: boolean
    description: Permit MAIL From and RCPT To params without normally required␣
↪brackets
    required: false
  interval_transfer_noop:
    type: integer
```

(continues on next page)

(continued from previous page)

```
    description: Interval between two NOOP commands sent to server while waiting for␣
↪stack proxy results
    required: false
  resolve_host:
    type: boolean
    description: Resolve client host from IP address and write to received line
    required: false
  permit_long_responses:
    type: boolean
    description: Permit overly long responses as some MTAs include variable parts in␣
↪responses
    required: false
  max_auth_request_length:
    type: integer
    description: Maximum allowed length of a request during SASL style authentication
    required: false
  max_response_length:
    type: integer
    description: Maximum allowed line length of server response
    required: false
  unconnected_response_code:
    type: integer
    description: Error code sent to client if connecting to server fails
    required: false
  add_received_header:
    type: boolean
    description: Add a received header into the email messages transferred by proxy
    required: false
  domain_name:
    type: string
    description: Fix a domain name into added receive line. add_received_header must␣
↪be true
    required: false
  tls_passthrough:
    type: boolean
    description: Change to passthrough mode
    required: false
  extensions:
    type: list
    description: Allowed ESMTP extensions, indexed by extension verb
    required: false
  require_crlf:
    type: boolean
    description: Specify whether proxy should enforce valid CRLF line terminations
    required: false
  timeout:
    type: integer
    description: Timeout in ms - if no packet arrives, connection is dropped
    required: false
  max_request_length:
    type: integer
    description: Maximum allowed line length of client requests
    required: false
  permit_unknown_command:
    type: boolean
    description: Enable unknown commands
    required: false
```

- tosca.policies.Security.MiCADO.Network.HttpProxy: Enforce HTTP protocol, can provide TLS control.

```
properties:
  max_keepalive_requests:
    type: integer
    description: Max number of requests allowed in a single session
    required: false
  permit_proxy_requests:
    type: boolean
    description: Allow proxy type requests in transparent mode
    required: false
  reset_on_close:
    type: boolean
    description: If connection is terminated without a proxy generated error, send an␣
↪RST instead of a normal close
    required: false
  permit_unicode_url:
    type: boolean
    description: Allow unicode characters in URLs encoded as u'
    required: false
  permit_server_requests:
    type: boolean
    description: Allow server type requests in non transparent mode
    required: false
  max_hostname_length:
    type: integer
    description: Maximum allowed length of hostname field in URLs
    required: false
  parent_proxy:
    type: string
    description: Address or hostname of parent proxy to be connected
    required: false
  permit_ftp_over_http:
    type: boolean
    description: Allow processing FTP URLs in non transparent mode
    required: false
  parent_proxy_port:
    type: integer
    description: Port of parent proxy to be connected
    required: false
  permit_http09_responses:
    type: boolean
    description: Allow server responses to use limited HTTP 0 9 protocol
    required: false
  rewrite_host_header:
    type: boolean
    description: Rewrite host header in requests when URL redirection occurs
    required: false
  max_line_length:
    type: integer
    description: Maximum allowed length of lines in requests and responses
    required: false
  max_chunk_length:
    type: integer
    description: Maximum allowed length of a single chunk when using chunked transer␣
↪encoding
    required: false
  strict_header_checking_action:
```

(continues on next page)

(continued from previous page)

```
    type: string
    description: Specify Zorp action if non rfc or unknown header in communication
    constraints:
      - valid_values: ["accept", "drop", "abort"]
    required: false
  non_transparent_ports:
    type: list
    description: List of ports that non transparent requests may use
    required: false
  strict_header_checking:
    type: boolean
    description: Require RFC conformant HTTP headers
    required: false
  max_auth_time:
    type: integer
    description: Force new auth request from client browser after time in seconds
    required: false
  max_url_length:
    type: integer
    description: Maximum allowed length of URL in a request
    required: false
  timeout_request:
    type: integer
    description: Time to wait for a request to arrive from client
    required: false
  rerequest_attempts:
    type: integer
    description: Control number of attempts proxy takes to send request to server
    required: false
  error_status:
    type: integer
    description: On error, Zorp uses this as status code of HTTP response
    required: false
  keep_persistent:
    type: boolean
    description: Try to keep connection to client persistent, even if unsupported
    required: false
  error_files_directory:
    type: string
    description: Location of HTTP error messages
    required: false
  max_header_lines:
    type: integer
    description: Maximum number of eader lines allowed in requests and responses
    required: false
  use_canonicalized_urls:
    type: boolean
    description: Enable canonicalization - converts URLs to canonical form
    required: false
  max_body_length:
    type: integer
    description: Maximum allowed length of HTTP request or response body
    required: false
  require_host_header:
    type: boolean
    description: Require presence of host header
    required: false
```

(continues on next page)

(continued from previous page)

```
buffer_size:
  type: integer
  description: Size of I O buffer used to transfer entity bodies
  required: false
permitted_responses:
  type: list
  description: Normative policy hash for HTTP responses indexed by HTTP method and␣
↪response code
  entry_schema:
    description: dictionary (string/int)
    type: map
  required: false
transparent_mode:
  type: boolean
  description: Enable transparent mode for the proxy
  required: false
permit_null_response:
  type: boolean
  description: Permit RFC incompliant responses with headers not terminated by CRLF,
↪ and not containing entity body
  required: false
language:
  type: string
  description: Specify language of HTTP error pages displayed to client
  required: false
  default: English
error_silent:
  type: boolean
  description: Turns off verbose error reporting to HTTP client, making firewall␣
↪fingerprinting more difficult
  required: false
permitted_requests:
  type: list
  description: List of permitted HTTP methods indexed by verb
  required: false
use_default_port_in_transparent_mode:
  type: boolean
  description: Enable use of default port in transparent mode
  required: false
timeout_response:
  type: integer
  description: Time to wait for the HTTP status line to arrive from the server
  required: false
permit_invalid_hex_escape:
  type: boolean
  description: Allow invalid hexadecimal escaping in URLs
  required: false
auth_cache_time:
  type: integer
  description: Caching authentication information time in seconds
  required: false
timeout:
  type: integer
  description: General I O timeout in ms
  required: false
default_port:
  type: integer
```

(continues on next page)

(continued from previous page)

```
    description: Used in non transparent mode when URL does not contain a port number
    required: false
    default: 80
```

- tosca.policies.Security.MiCADO.Network.HttpURIFilterProxy: Enforce HTTP protocol with regex URL filtering capabilities

```
properties:
  matcher_whitelist:
    type: list
    description: List of regex determining permitted access to a URL (precedence)
    required: true
  matcher_blacklist:
    type: list
    description: List of regex determining prohibited access to a URL
    required: true
```

- tosca.policies.Security.MiCADO.Network.HttpWebdavProxy: Enforce HTTP protocol with request methods for WebDAV.

This proxy has no additional properties.

1.3.9 Secret policy

There is a way to define application-level secrets in the MiCADO application description. These secrets are managed by Security Policy Manager and stored and distributed as a single secret called **micado.appsecret** by Kubernetes.

See an example below for creating a secret using policies, and assigning it to an environment variable (ENV_SALT in the example) in a container:

```
topology_template:
  node_templates:
    my-app-container:
      type: tosca.nodes.MiCADO.Container.Application.Docker
      properties:
        ...
        env:
        - name: ENV_SALT
          valueFrom:
            secretKeyRef:
              name: micado.appsecret
              key: salt_value

  policies:
  - secret:
      type: tosca.policies.Security.MiCADO.Secret.KubernetesSecretDistribution
      properties:
        text_secrets:
          salt_value: "123456qwerty"
```

1.4 Tutorials

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

1.4.1 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

1.4.2 cqueue

This application demonstrates a deadline policy with CQueue. CQueue provides a lightweight queueing service for executing containers. The entire infrastructure will be deployed by a single ADT as a microservices architecture. CQueue server (implemented by RabbitMQ, Redis and a web-based frontend) will be run on a static VM in the cluster. The server stores items in a queue where each item represents a container execution. CQueue workers will run on a separate set of scalable VMs, and are responsible for fetching items and preforming the execution of the container locally. The demonstration below shows that the items can be consumed before a deadline using MiCADO for scaling the CQueue worker and its VM nodes.

If you prefer to launch your own cQueue server externally, use the docker-compose file `docker-compose-cqueue-server.yaml` and edit the relevant shell scripts to point to your server and to launch `micado-cqworker.yaml` instead.

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

- Step1: Update the file cq-microservice.yaml with the CloudSigma ID details necessary to launch your **two** VM sets
  - Update each 'ADD_YOUR_ID_HERE' string with the proper value retrieved under your CloudSigma account.
  - Make sure port 30888 is open on the `cq-server` virtual machine set
- Step2: Update the parameter file, called `_settings` . You need the ip address for the MiCADO master and, once your worker nodes are running, you should enter the IP for the CQueue server which is about to be deployed. Setting the IP of the CQueue server is a required step if your MiCADO Master does not have the appropriate port open.
- Step3: Run `./1-deploy-cq-microservices.sh` to deploy the cQueue server and worker components to separate virtual machine nodes
- Step4: Use your Cloud WebUI and find the public IP of the VM hosting the cQueue server (in fact, this can be **any** VM in your cluster with port 30888 open)
- Step5: Run `./3-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: Run `./4-update-cqueue-deadline.sh xxxxxxx` where **xxxxxxx** is the unix timestamp taken from the previous step.
- Step7: Run `./5-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.
- Step8a: You can run `./2-list-running-apps.sh` to list the apps running under MiCADO.
- Step8b: You can run `query-services.sh` to see the details of docker services of your application
- Step8c: You can run `query-nodes.sh` to see the details of docker nodes hosting your application
- Step9: Run `./6-undeploy-cq-microservices.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.

1.4.3 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` (port 30012 must be open) 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, which should now be accessible on <micado_worker_ip>:30012
- 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. **NOTE:** In some cases, depending on your cloud, the pre-configured load test may be too weak to trigger a scaling response from MiCADO. If this is the case, edit the file `3-generate-traffic.sh` and increase the load options in the command on the very last line, for example `wrk -t4 -c40 -d8m http://.....` On the other hand, a load test too powerful will be like launching a denial-of-service attack on yourself.
- 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

1.4.4 wordpress

This application deploys a wordpress blog, complete with MySQL server and a Network File Share for peristent 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 (port 30010 must be open) 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 <worker_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. **NOTE:** In some cases, depending on your cloud, the pre-configured load test may be too weak to trigger a scaling response from MiCADO. If this is the case, edit the file `3-generate-traffic.sh` and increase the load options in the command on the very last line, for example `wrk -t4 -c40 -d8m http://....`. On the other hand, a load test too powerful will be like launching a denial-of-service attack on yourself.
- 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

1.5 MiCADO Client Library

1.5.1 Overview

MiCADO client library extends the MiCADO functionality with MiCADO master deployment capabilities and application management. The library aims to provide a basic API from Python environment and support the following:

- **Deploy MiCADO service**
  - Create, Destroy MiCADO master VM
- **Manage application**
  - Create, Update, Delete MiCADO applications

Currently, client library supports only NOVA interface. We plan to extend with additional interfaces later.

1.5.2 Getting Started

Install requirements

The required Python packages are defined under the `requirements.txt`. Make sure to install those before using MiCADO client library. For reference, they are:

- requests==2.24.0
- ruamel.yaml==0.16.10
- pycryptodome==3.9.8
- python-novaclient==17.2.0
- openstacksdk==0.48.0
- ansible==2.10.0

Get the MiCADO Client Library

Currently, the library is available directly from github.

Simply clone the respository and add the location to your PYTHONPATH

```
$ MC_PATH="/usr/local/lib/micado-client"
$ git clone https://github.com/micado-scale/micado-client $MC_PATH
$ export PYTHONPATH="$PYTHONPATH:$MC_PATH"
$ mkdir -p ~/.micado-cli
$ touch ~/.micado-cli/credentials-cloud-api.yml
```

Specify cloud credentials

Specify cloud credential for MiCADO master VM creation. Please, edit `~/.micado-cli/credentials-cloud-api.yml`

```
resource:
-
    type: nova
    auth_data:
        # Select your authentication method
        # Option #1
        username:
        password:
        # Option #2
        application_credential_id:
        application_credential_secret:
```

1.5.3 Example

Note: Before you start testing, make sure the authentication data in the correct place.

For more details, see the Documentation Reference section below. There are three use-cases identified for using micado-client.

Use-case 1

MiCADO master is created with the help of MiCADO client library. The create and destroy methods are invoked in the same program i.e. storing and retrieving the `client.master` object is not needed.

```
from micado import MicadoClient

client = MicadoClient(launcher="openstack")
client.master.create(
    auth_url='yourendpoint',
    project_id='project_id',
    image='image_name or image_id',
    flavor='flavor_name or flavor_id',
    network='network_name or network_id',
    keypair='keypair_name or keypair_id',
    security_group='security_group_name or security_group_id'
    )
```

(continues on next page)

(continued from previous page)

```
client.applications.list()
client.master.destroy()
```

Use-case 2

MiCADO master is created with the help of MiCADO client library. The create and destroy methods are invoked in seperate programs i.e. storing and retrieving the `client.master` object is needed.

```
from micado import MicadoClient

client = MicadoClient(launcher="openstack")
master_id = client.master.create(
    auth_url='yourendpoint',
    project_id='project_id',
    image='image_name or image_id',
    flavor='flavor_name or flavor_id',
    network='network_name or network_id',
    keypair='keypair_name or keypair_id',
    security_group='security_group_name or security_group_id'
    )
client.applications.list()
<< store your master_id >>
<< exiting... >>
-------------------------------------------------------------
<< start >>
...
master_id = << retrieve master_id >>
client = MicadoClient(launcher="openstack")
client.master.attach(master_id = master_id)
client.applications.list()
client.master.destroy()
```

Use-case 3

MiCADO master is created independently from the MiCADO client library. The create and destroy methods are not invoked since the client library used only for handling the applications.

```
from micado import MicadoClient
client = MicadoClient(
    endpoint="https://micado/toscasubmitter/",
    version="v2.0",
    verify=False,
    auth=("ssl_user", "ssl_pass")
    )
client.applications.list()
```

1.5.4 Documentation Reference

MiCADO Client

Higher-level client for both submitter interaction and launching ability

class micado.client.**MicadoClient**(**args*, ***kwargs*)

Bases: `object`

The MiCADO Client

Builds and communicates with a MiCADO Master node

Usage with a launcher:

a)
```
>>> from micado import MicadoClient
>>> client = MicadoClient(launcher="openstack")
>>> client.master.create(
...     auth_url='yourendpoint',
...     project_id='project_id',
...     image='image_name or image_id',
...     flavor='flavor_name or flavor_id',
...     network='network_name or network_id',
...     keypair='keypair_name or keypair_id',
...     security_group='security_group_name or security_group_id'
... )
>>> client.applications.list()
>>> client.master.destroy()
```

b)
```
>>> from micado import MicadoClient
>>> client = MicadoClient(launcher="openstack")
>>> master_id = client.master.create(
...     auth_url='yourendpoint',
...     project_id='project_id',
...     image='image_name or image_id',
...     flavor='flavor_name or flavor_id',
...     network='network_name or network_id',
...     keypair='keypair_name or keypair_id',
...     security_group='security_group_name or security_group_id'
... )
>>> client.applications.list()
>>> << store your master_id >>
>>> << exit >>
>>> -------------------------------------------------------------
>>> << start >>
>>> ...
>>> master_id = << retrieve master_id >>
>>> client = MicadoClient(launcher="openstack")
>>> client.master.attach(master_id = master_id)
>>> client.applications.list()
>>> client.master.destroy()
```

Usage without a launcher i.e. MiCADO master is already created independently from the client library.

```
>>> from micado import MicadoClient
>>> client = MicadoClient(
...     endpoint="https://micado/toscasubmitter/",
...     version="v2.0",
...     verify=False,
...     auth=("ssl_user", "ssl_pass")
... )
>>> client.applications.list()
```

Parameters

- **auth_url** (*string*) – Authentication URL for the NOVA resource.
- **image** (*string*) – Name or ID of the image resource.
- **flavor** (*string*) – Name or ID of the flavor resource.

- **network** (*string*) – Name or ID of the network resource.
- **keypair** (*string*) – Name or ID of the keypair resource.
- **security_group** (*string, optional*) – name or ID of the security_group resource. Defaults to 'all'.
- **region** (*string, optional*) – Name of the region resource. Defaults to None.
- **user_domain_name** (*string, optional*) – Define the user_domain_name. Defaults to 'Default'
- **project_id** (*string, optional*) – ID of the project resource. Defaults to None.
- **micado_user** (*string, optional*) – MiCADO username. Defaults to admin.
- **micado_password** (*string, optional*) – MiCADO password. Defaults to admin.
- **endpoint** (*string*) – Full URL to API endpoint (omit version). Required.
- **version** (*string, optional*) – MiCADO API Version (minimum v2.0). Defaults to 'v2.0'.
- **verify** (*bool, optional*) – Verify certificate on the client-side. OR (str): Path to cert bundle (.pem) to verfiy against. Defaults to True.
- **auth** (*tuple, optional*) – Basic auth credentials (<user>, <pass>). Defaults to None.

applications

classmethod from_master()

Usage: Ensure MICADO_API_ENDPOINT and MICADO_API_VERSION environment variables are set, then:

```
>>> from micado import MicadoClient
>>> client = MicadoClient.from_master()
```

master

MiCADO Application

Module for managing and representing applications in MiCADO

class micado.models.application.**Application**(*client*, *id=None*, *info=None*, *resource=None*)

Bases: micado.models.base.Model

Representation of an application deployed in MiCADO

The state of an application can be refreshed with reload()

adaptors

Adaptor info for this application

class micado.models.application.**Applications**(*client*)

Bases: micado.models.base.Resource

Model for managing applications in MiCADO

Basic CRUD functionality is implemented here with create(), list()/get(), update() and delete()

create(*app_id=None*, ***kwargs*)

Creates a new application in MiCADO

Parameters

- **app_id** (*string, optional*) – Application ID. Generated if None.
- **adt** (*dict, optional*) – YAML dict of Application Description Template. Required if URL is empty. Defaults to None.
- **url** (*string, optional*) – URL of YAML ADT. Required if ADT is empty. Defaults to None.
- **params** (*dict, optional*) – TOSCA input parameters. Defaults to {}.
- **dryrun** (*bool, optional*) – Flag to skip execution of components. Defaults to False.

Usage:

```
>>> client.applications.create(app_id="stresstest",
                               url="example.com/repo/adt.yaml")
"stresstest created successfully"
```

Returns ID and status of deployment

Return type dict

delete(*app_id*)

Deletes an application in MiCADO given its ID.

Parameters **app_id** (*string*) – Application ID to delete

Usage:

```
>>> client.applications.delete("stresstest")
"stresstest deleted successfully"
```

Returns ID and status of deletion

Return type dict

get(*app_id*)

Retrieves info on a specific application, given its ID

Parameters **app_id** (*string*) – Application ID to fetch. Required

Usage:

```
>>> my_app = client.applications.get("stresstest")
>>> my_app.id
"stresstest"
>>> my_app.adaptors
{'KubernetesAdaptor': 'Executed', 'OccopusAdaptor': 'Skipped'}
```

Returns Relevant information for a single application

Return type Application object

list()

Retrieves the available list of applications in MiCADO

Usage:

```
>>> running_apps = client.applications.list()
>>> [app.id for app in running_apps]
["stresstest"]
```

Returns Relevant info for all applications

Return type list of Application objects

model

alias of *Application*

MiCADO Master

Higher-level methods to manage the MiCADO master

class micado.models.master.**MicadoMaster**(***kwargs*)

Bases: micado.models.base.Model

api

attach(*master_id*)

Configure the master object to handle the instance created by the def:create()

Parameters **master_id** (*string*) – master ID returned by def:create()

create(***kwargs*)

Creates a new MiCADO master VM and deploy MiCADO service on it.

Parameters

- **auth_url** (*string*) – Authentication URL for the NOVA resource.
- **image** (*string*) – Name or ID of the image resource.
- **flavor** (*string*) – Name or ID of the flavor resource.
- **network** (*string*) – Name or ID of the network resource.
- **keypair** (*string*) – Name or ID of the keypair resource.
- **security_group** (*string, optional*) – name or ID of the security_group resource. Defaults to ‘all’.
- **region** (*string, optional*) – Name of the region resource. Defaults to None.
- **user_domain_name** (*string, optional*) – Define the user_domain_name. Defaults to ‘Default’
- **project_id** (*string, optional*) – ID of the project resource. Defaults to None.
- **micado_user** (*string, optional*) – MiCADO username. Defaults to admin.
- **micado_password** (*string, optional*) – MiCADO password. Defaults to admin.

Usage:

```
>>> client.master.create(
...     auth_url='yourendpoint',
...     project_id='project_id',
...     image='image_name or image_id',
...     flavor='flavor_name or flavor_id',
...     network='network_name or network_id',
...     keypair='keypair_name or keypair_id',
...     security_group='security_group_name or security_group_id'
... )
```

Returns ID of MiCADO master

Return type string

`destroy`()

Destroy running applications and the existing MiCADO master VM.

Usage:

```
>>> client.master.destroy()
```

`init_api`()

Configure Submitter API

Returns return SubmitterClient

Return type SubmitterClient

`launcher`

`master_id`

1.5.5 Roadmap

- Support additional Cloud interfaces (e.g EC2)

1.6 Release Notes

Changelog since v0.5.x of MiCADO

See more detailed notes about upgrading in *What's New*

1.6.1 v0.9.1 (1 October 2020)

- Add support for Oracle Cloud Infrastructure
- Add support for Ubuntu 20.04 LTS
- Improve RESTful nature of Submitter with v2.0 API
- Base component images on `alpine` for a smaller footprint
- Bump Kubernetes to v1.19
- Support TOSCA v1.2 template files
- Refactor custom TOSCA type definitions

- Refactor Submitter parsing modules to improve parsing times
- Refactor KubernetesAdaptor for more customisable resources
- Improve validation of translated Kubernetes manifests
- Support config_drive flag in OpenStack (Terraform only)
- Port PolicyKeeper to Python3
- Increase timeout for MiCADO component deployment (for slower machines)
- Increase timeout for inactive worker node removal (for poor networks)
- Reduce Prometheus default scrape interval (for custom exporters)
- Add `mode` to Ansible tasks for CVE-2020-1736
- Include hostname and IP as SANs in self-signed certs
- Fix: enable secret distribution via ADT policy

1.6.2 v0.9.0 (9 April 2020)

- Refactor playbook tasks to be more component-specific
- Add playbook tasks for configuring and installing Terraform
- Use the Ansible *k8s module* for managing Kubernetes resources
- Optimise cloud-init scripts by reducing *apt-get update*
- Fix Master-Worker Ubuntu mismatch bug
- Handle undefined credential file path
- Store credential data in Kubernetes Secrets
- Support updates of credentials on a deployed MiCADO Master
- Add demo ADTs for Azure & GCE
- Update QuickStart docs in README
- Bump Grafana to v6.6.2
- Bump Prometheus to v2.16.0
- Bump Kubernetes-Dashboard to v2.0.0 (rc7)
- Hide Kubernetes Secrets on Kubernetes-Dashboard
- Refactor PK main loop to support multiple cloud orchestrators
- Add Terraform handler to PK for scaling (up/down and dropping specific nodes)
- Switch to the *pykube* package in PK instead of *kubernetes*
- Add the TerraformAdaptor to the TOSCASubmitter
- Bump TOSCASubmitter package versions
- Discover cloud from TOSCA ADT type and deprecate *interface_cloud*
- Rename ADT compute property *endpoint_cloud* to *endpoint*
- Support *insert* in ADT to modify cloud-init cloud-config
- Support authentication with OpenStack application credential

- Pass orchestrator info to PK during PKAdaptor translation
- Lower reserved CPU and Memory for Zorp Ingress on workers
- Only deploy Zorp Ingress to workers with matching ADT policy
- Bump Kubernetes to v1.18
- Bump Flannel to v0.12
- Bump containerd.io to v.1.2.13
- Bump Occopus to v1.7 (rc6)
- Bump cAdvisor to v0.34.0
- Bump AlertManager to v0.20.0

1.6.3 v0.8.0 (30 September 2019)

- simplify ADTs by introducing pre-defined TOSCA node types
- add support for Kubernetes ConfigMaps, Namespaces and multi-container Pods
- metric collection (disabled by default) is now enabled with "monitoring" policy
- upgrade all components (Docker, Kubernetes, Grafana, Prometheus, etc. . .)
- introduce new Optimizer supported scaling
- add MiCADO version on dashboard and Grafana
- introduce log rotate for Docker and components
- introduce node downscale mechanism with node selection
- redirect stdout of scaling_rule usercode to different log file
- add support of keystone V3 for OpenStack in Occopus
- improve cloud API handling in Occopus
- make the master node web authentication timeout configurable
- make master-worker node VPN connection more restrictive
- implement ADT-based application secret distribution
- push cloud secrets to Credential Store at deploy time
- implement Security Policy Manager adaptor in the TOSCA Submitter
- add support for configuring application-level firewalling rules for the application through the ADT (FWaaS)
- generate node certificate with the right common name for the master node
- make the micadoctl command line utility to work after the transition to Kubernetes pods
- fix keypair distribution to worker nodes
- update TOSCA template for Kubernetes application-level secret distribution
- refactor Kubernetes translation
- fix Policy Keeper Kubernetes node maintenance
- propagate Kubelet configuration to woker nodes
- support system cGroup driver by Docker & Kubernetes

- fix Kubernetes node objects to be deleted on "undeploy"
- fix Occopus create & import actions to correctly raise exceptions
- fix Occopus updates not to kill unrelated nodes
- support updates of an ADT with no Occopus nodes
- support updates of an ADT with no Kubernetes nodes
- add a timeout to Kubernetes undeploy
- simplify hosts.yml file

1.6.4 v0.7.3 (14 Jun 2019)

- update MiCADO internal core services to run in Kubernetes pods
- remove Consul and replace it with Prometheus' Kubernetes Service Discovery
- update cAdvisor and NodeExporter to run as Kubernetes DaemonSets
- introduce the support for creating prepared image for the MiCADO master and the MiCADO worker
- introduce the support for deploying unique "sets" of virtual machines scaling independently
- update Grafana to track the independently scaling VMs from the drop-down Node ID
- update scrape interval between Prometheus and cAdvisor to be less frequent
- fix the Occopus Adaptor to correctly raise exceptions for the submitter
- update Kubernetes Dashboard to improve RBAC permissions
- update the Flannel Overlay deployment
- update the Kubernetes eviction thresholds on the Master node to be lowered
- remove Docker-Compose from Master & Workers
- fix dependencies and vulnerabilities
- add dry-run support for the Submitter upon launch of TOSCA ADT
- add new api call for the Submitter to validate TOSCA template
- improve Submitter logs
- improve Submitter responses to users
- improve handling of wrong template by Submitter
- add support for hv_relaxed and hv_tsc CloudSigma specific properties
- add support for tagging EC2 type resources
- add disk and free space checking to the deployment playbook
- update the Wordpress demo to demonstrate "virtual machine sets"
- update the cQueue demo to demonstrate "virtual machine sets"
- fix and improve the NGINX demo

1.6.5 v0.7.2-rev1 (01 Apr 2019)

- fix dependency issue for Kubernetes 1.13.1 (kubernetes/kubernetes#75683)

1.6.6 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 kubectl 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

1.6.7 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)

1.6.8 v0.7.0 (12 Dec 2018)

- Introduce Kubernetes as the primary container orchestration engine
- Replace the swarm-visualiser with the Kubernetes Dashboard

1.6.9 Older MiCADO Versions

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

1.7 What's New

This section contains detailed upgrade notes for recent versions

1.7.1 v0.9.1

Major Enhancements

Support for Oracle Cloud Infrastructure

MiCADO now includes support for Oracle Cloud Infrastructure !

Provisioning virtual machines in OCI is currently supported by Terraform in MiCADO. Details on preparing your ADT for use with Oracle can be found in the *OCI* section of the documentation.

Submitter API v2.0

The MiCADO Submitter RESTful API has been updated to v2.0, to facilitate better intergration with other tools and platforms. The functionality of the previous API version has been preserved and limited backwards compatibility with v1.0 of the API is still maintained in this version.

Expect v1.0 of the API to be deprecated in a future version.

Fixes

Kubernetes Secret Distribution via ADT Policies

A bug which prevented MiCADO from distributing application secrets defined in policies within the ADT has been resolved. See the *Secret policy* section of the documentation for details on how to define and assign secrets inside an ADT.

Known Issues & Deprecations

Ubuntu 16.04

This version of MiCADO adds support for the latest Ubuntu 20.04 LTS. Going forward, MiCADO will deprecate support for Ubuntu 16.04 LTS and focus on supporting the current long-term support releases of Ubuntu: 18.04 and 20.04

1.7.2 v0.9.0

Major Enhancements

Terraform for Cloud Orchestration

Support for Terraform has been added to MiCADO!
The TerraformAdaptor currently supports the following cloud resources:

- OpenStack Nova Compute
- Amazon EC2 Compute
- Microsoft Azure Compute
- Google Compute Engine

To use Terraform with MiCADO it must be **enabled** during deployment of the MiCADO Master, and an appropriate **ADT** should be used.

Improved Credential File Handling

Cloud credentials are now stored in Kubernetes Secrets on the MiCADO Master. Additionally, credentials on an already deployed MiCADO can now be updated or modified using Ansible.

Improved Node Contextualisation

It is now possible to **insert** contextualisation configurations earlier in the default *cloud-init #cloud-config* for worker nodes. This extends the existing **append** functionality to support configuration tasks which should precede the initialisation of the worker node (joining the Kubernetes cluster, bringing up the IPSec tunnel, etc…)

Fixes

Zorp Ingress

The Zorp Ingress Controllers in v0.8.0 were incorrectly being deployed alongside *every* application, even if the policy did not call for it. This has now been resolved.

Additionally, these workers were requesting a large amount of CPU and Memory, which could limit scheduling on the node. Those requests have been lowered to more reasonable values.

Different Versioned Workers

In previous versions of MiCADO, deployed worker nodes which did not match the Ubuntu version of the MiCADO Master would be unable to join the MiCADO cluster. This has now been resolved.

Known Issues & Deprecations

IPSec and Dropped Network Packets

On some network configurations, for example where IPSec protocols ESP (50) and AH (51) are blocked, important network packets can get dropped in Master-Worker communications. This might be seen as Prometheus scrapes failing with the error **context deadline exceeded**, or Workers failing to join the Kubernetes cluster. To disable the IPSec tunnel securing Master-Worker communications, it can be stopped by appending **ipsec stop** to **runcmd** in the default worker node *cloud-init #cloud-config*.

Compute Node Inputs in ADTs

The Occopus **input** *interface_cloud* has been deprecated and removed, as cloud discovery is now based on TOSCA type. It will continue to be supported (ignored) in this version of MiCADO but may raise warnings or errors in future versions.

The **input** *endpoint_cloud* has been deprecated in favour of *endpoint*. Both Terraform and Occopus will support *endpoint_cloud* in this version of MiCADO but a future version will drop support.

With the above changes in mind, Terraform will support v0.8.0 ADTs which only include EC2 or Nova Compute nodes. This can be acheieved simply by changing **interfaces** from *Occopus* to *Terraform*, though it should be noted:

- Terraform will auto-discover the EC2 endpoint based on the *region_name* property, making the *endpoint* input no longer required. The *endpoint* input can still be passed in to provide a custom endpoint.

- For some OpenStack configurations, Terraform requires a *network_name* as well as *network_id* to correctly identify networks. The *network_name* property can be passed in as **properties** or **inputs**

Python Module Index

Index