MiCADO
scale

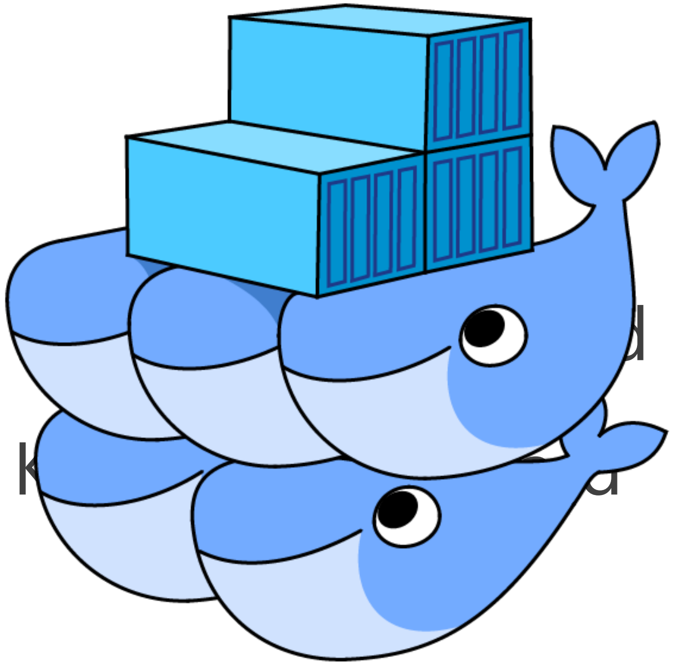
Konténer orkesztráció és autóskálázás MiCADO referencia architektúrával

Container orchestration and autoscaling by MiCADO

Kovács József

HUN REN SZTAKI

Motivation



- □ d
- □ K d

Kubernetes is good

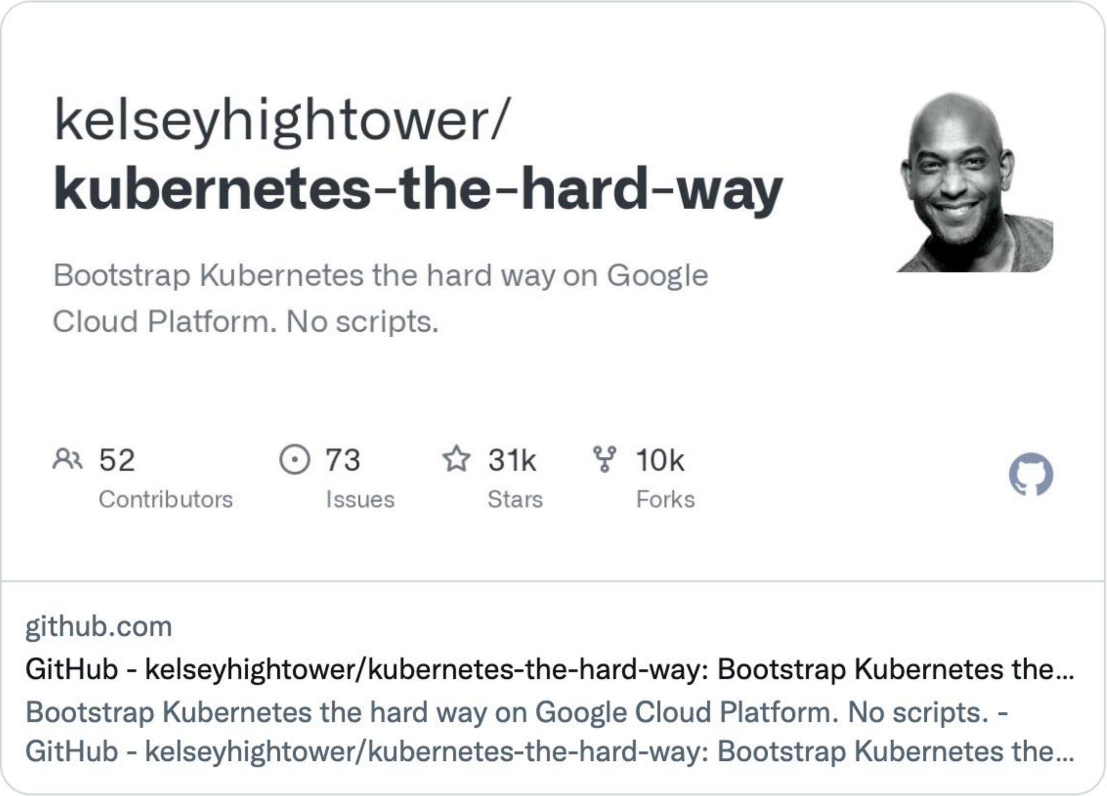
for Cloud Native Microservices Architectures

- □ Self-healing
- □ Auto-scaling
- □ Health-checking
- □ Rolling updates
- □ Networking
- □ Security

Kubernetes is hard

(even when using a managed service)

- □ Deploying/managing a cluster
- □ Configuring a cluster
- □ Understanding abstractions
  - □ pod, job, deployment, replica set
- □ Writing templates
  - □ manifest files
- □ Debugging



kelseyhightower/
kubernetes-the-hard-way

Bootstrap Kubernetes the hard way on Google Cloud Platform. No scripts.

52 Contributors 73 Issues 31k Stars 10k Forks

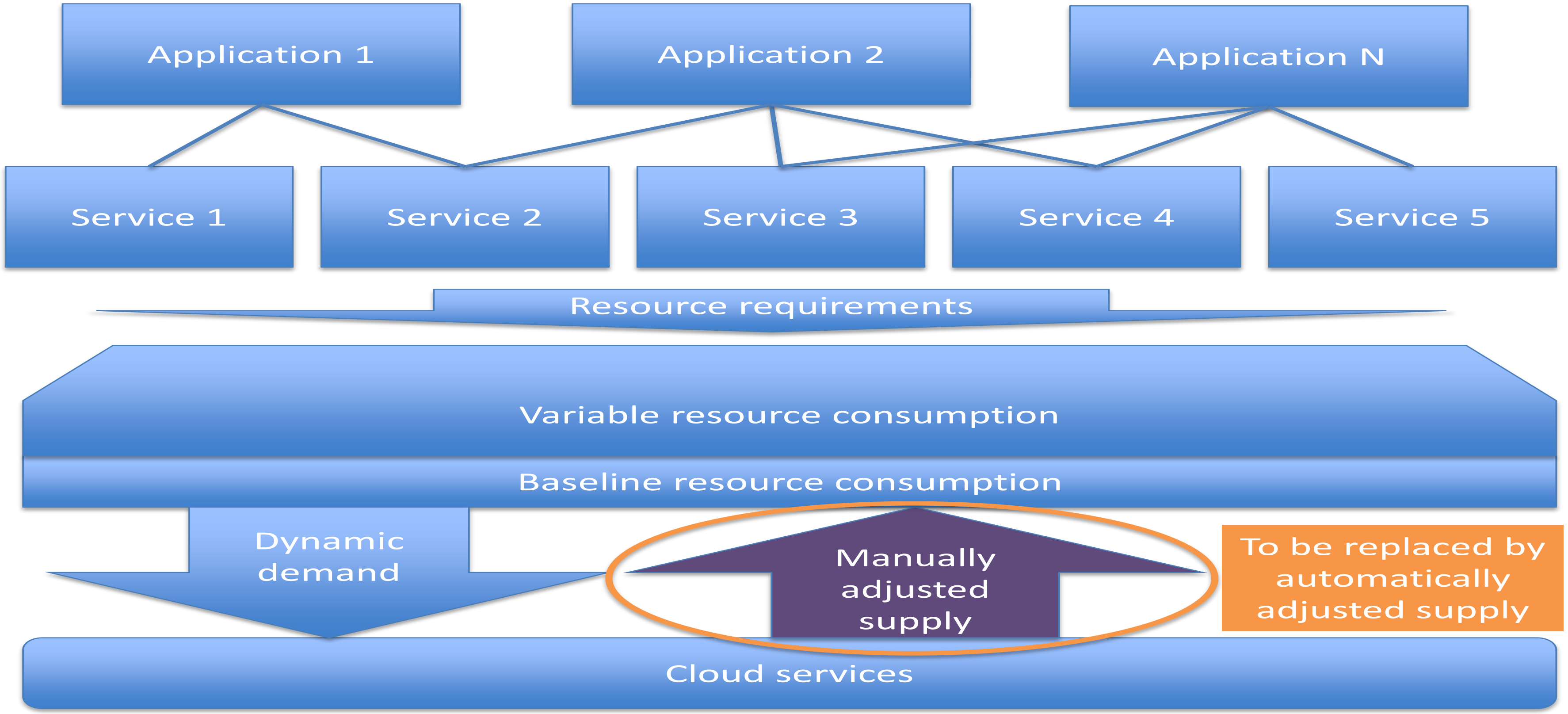
github.com

GitHub - kelseyhightower/kubernetes-the-hard-way: Bootstrap Kubernetes the...

Bootstrap Kubernetes the hard way on Google Cloud Platform. No scripts. - GitHub - kelseyhightower/kubernetes-the-hard-way: Bootstrap Kubernetes the...

Application level orchestration

- multiple heterogeneous clouds
- wide range of scaling policies
- wide range of monitoring parameters
- advanced security solutions



To achieve resource scalability and efficient resource utilisation supporting

Solution

Dynamic Cloud Orchestrator

- "One-click" deployment of an enhanced Kubernetes cluster
- Deploys, provisions, manages (auto-scaling, self-healing):
  - Applications (containers)
  - Cloud resources (virtual machines)
- Improved security
- Metrics dashboard

MiCADO scale

MiCADO – Microservices-based Cloud Application-level Dynamic Orchestrator

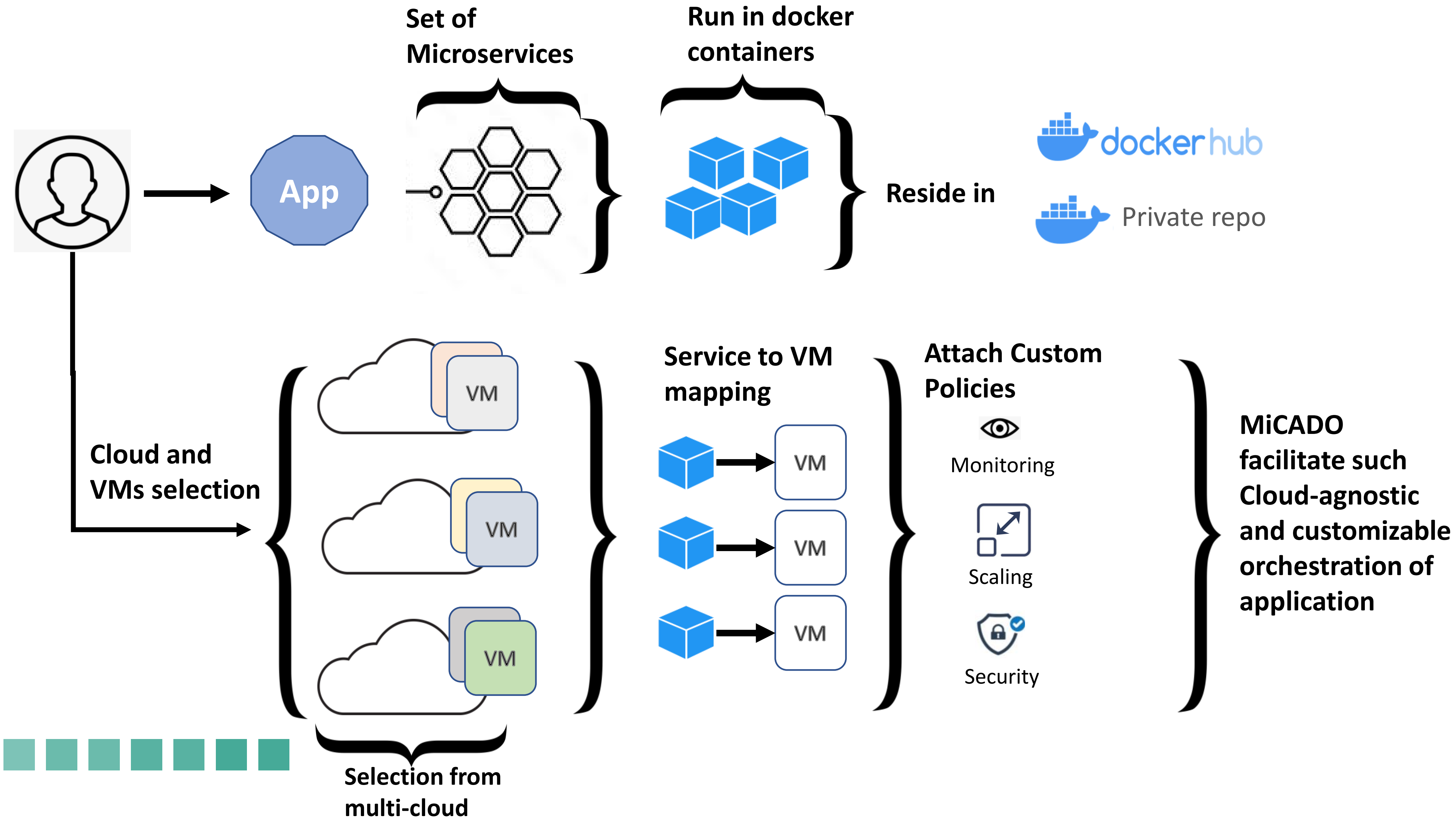
- History
  - Result of the **H2020 COLA** (Cloud Orchestration at the Level of Application) project (2017-2019)
  - Based on cooperation between Westminster University and SZTAKI
  - Since 2019, Westminster University has taken over the maintenance and development, SZTAKI contribution
  - Further developed in many European projects since the first prototype
    - currently actively developed in the H2020 DIGITbrain project
    - used in PITHIA-NRF, CO-VERSATILE (and Harpocrates and ARCAFF from October 2022)

MiCADO scale

MiCADO – Microservices-based Cloud Application-level Dynamic Orchestrator

- Main features
  - Automated application **deployment** based on TOSCA-based application description templates
  - **TOSCA** - Topology and Orchestration Specification for **Cloud** Applications
  - Automated **scaling** based on highly customisable scaling policies
    - scaling at both **container and virtual machine** levels
  - **Multi-cloud** support – application portability
  - Policy driven **security** settings
  - **Open source** and fully managed distributions
  - **Job queue** management with the extension of JQueuer
  - **Edge support** introduced for IoT-based applications

The overall concept



Set of Microservices

Run in docker containers

Reside in

Cloud and VMs selection

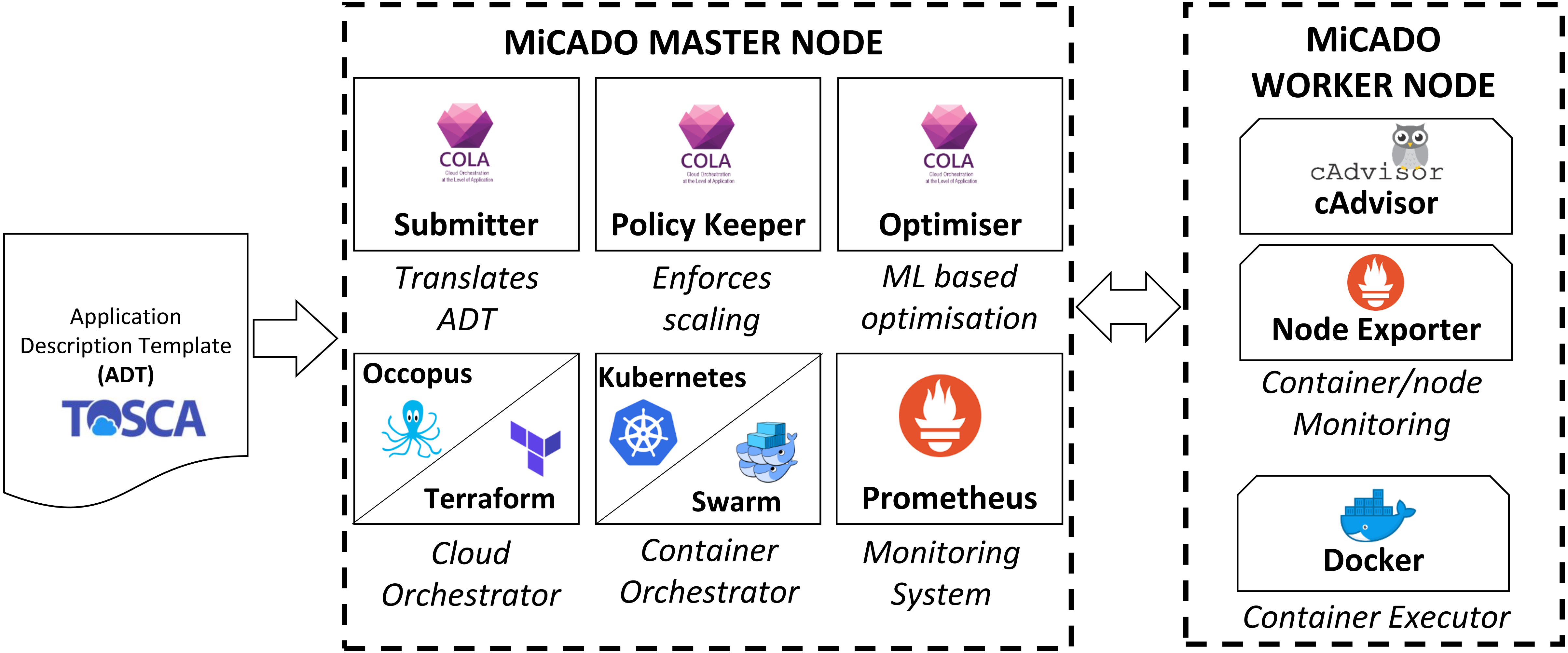
Selection from multi-cloud

Service to VM mapping

Attach Custom Policies

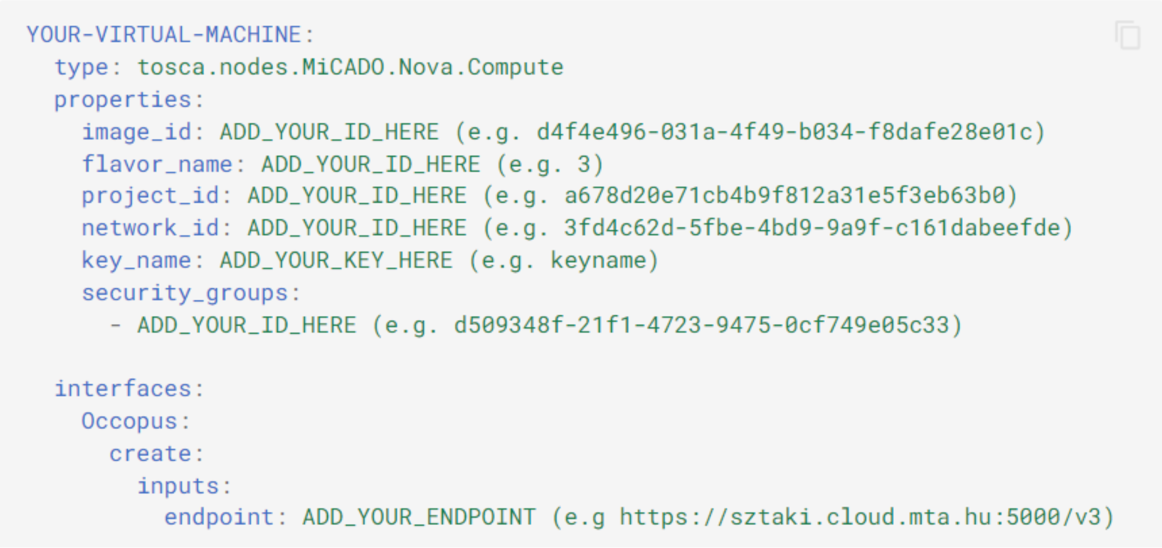
MiCADO facilitate such Cloud-agnostic and customizable orchestration of application

High-level architecture



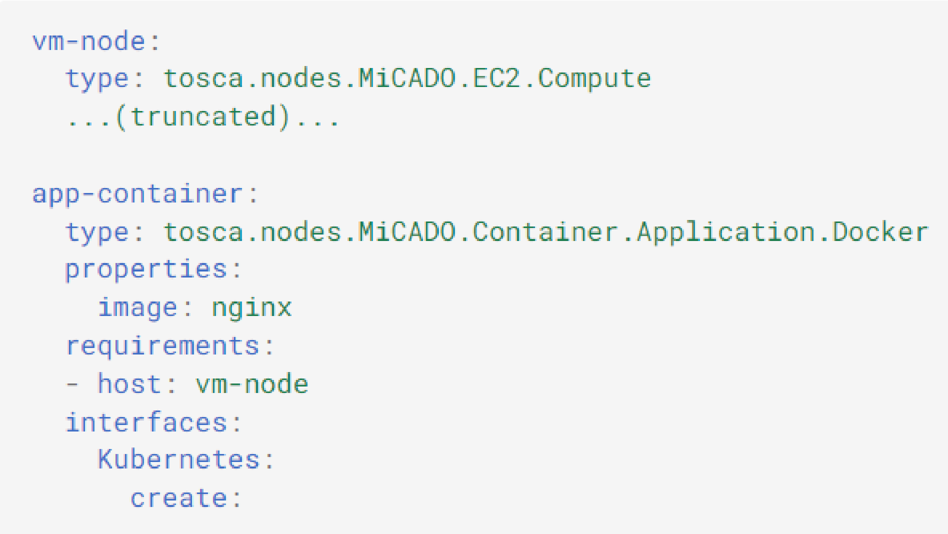
Application Description - Basics

Cloud Infrastructure (Instance size, SSH keys, opened ports, VM image)



```
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)
```

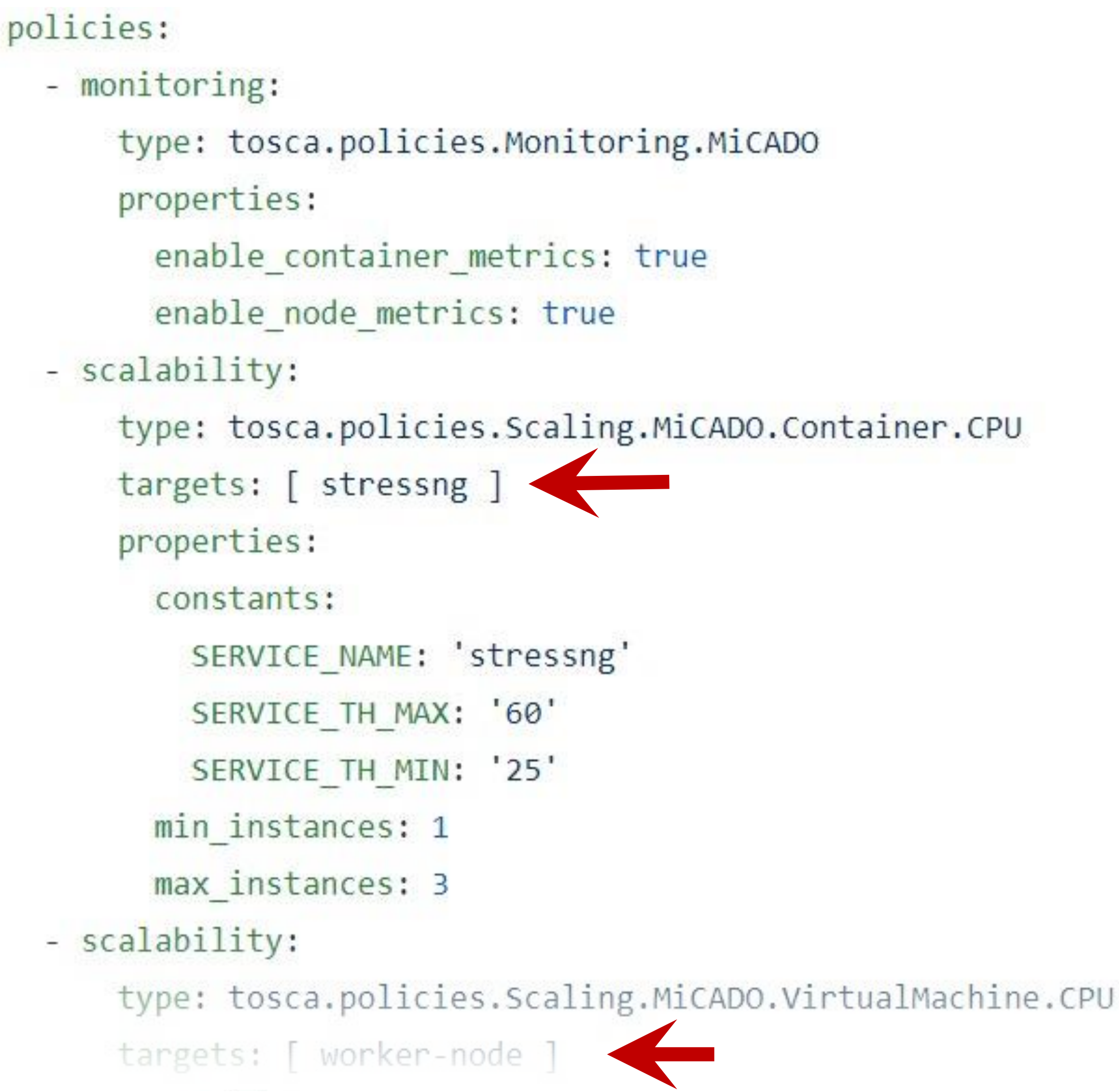
Container Infrastructure (Container, volumes, configurations)



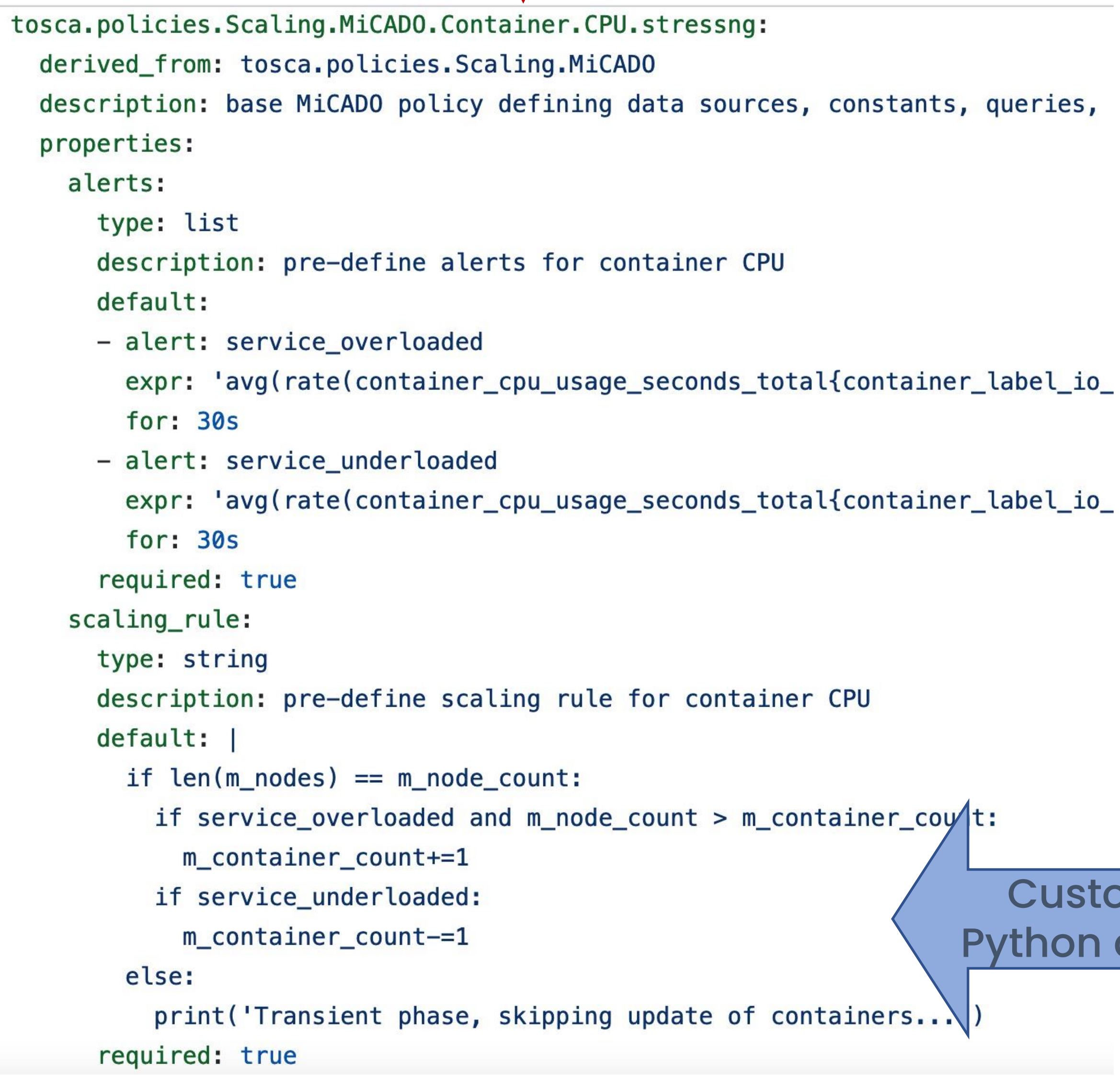
```
vm-node:
  type: tosca.nodes.MiCADO.EC2.Compute
  ...(truncated)...

app-container:
  type: tosca.nodes.MiCADO.Container.Application.Docker
  properties:
    image: nginx
  requirements:
  - host: vm-node
  interfaces:
    Kubernetes:
      create:
```

Application Description - Policies

- Monitoring subsystem
  - Monitoring metrics are collected by dynamically attachable data collectors (Prometheus exporters)
  - System and application metrics
- Highly customisable scaling system
  - Scaling of BOTH containers and virtual machines are supported
  - Scaling logic is fully programmable (using Python)
  - Various strategies (load-based, deadline-based, event-based, Scheduled)

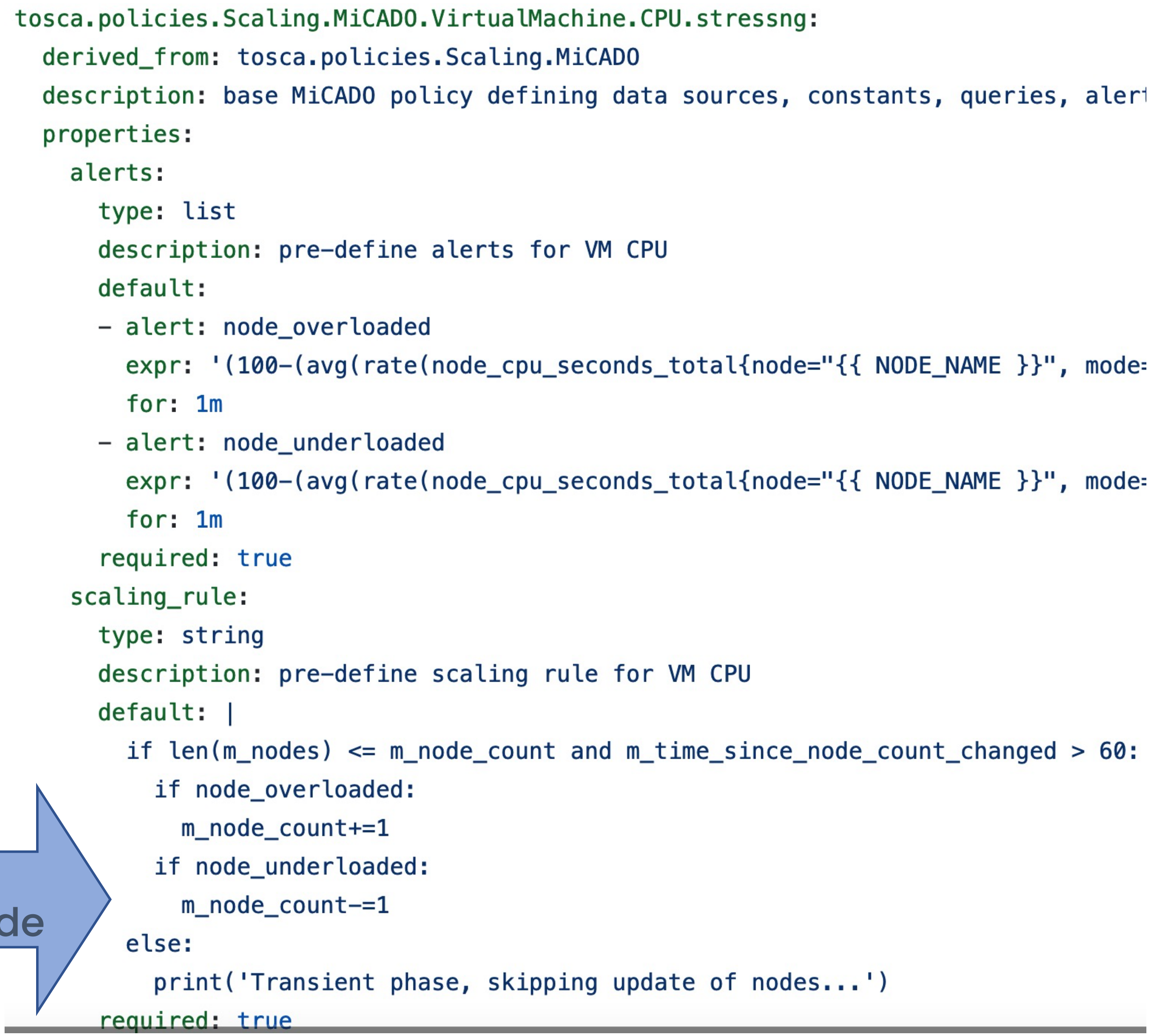


```
policies:
  - monitoring:
    type: tosca.policies.Monitoring.MiCADO
    properties:
      enable_container_metrics: true
      enable_node_metrics: true
  - scalability:
    type: tosca.policies.Scaling.MiCADO.Container.CPU
    targets: [ stressng ]
    properties:
      constants:
        SERVICE_NAME: 'stressng'
        SERVICE_TH_MAX: '60'
        SERVICE_TH_MIN: '25'
      min_instances: 1
      max_instances: 3
  - scalability:
    type: tosca.policies.Scaling.MiCADO.VirtualMachine.CPU
    targets: [ worker-node ]
```



```
tosca.policies.Scaling.MiCADO.Container.CPU.stressng:
  derived_from: tosca.policies.Scaling.MiCADO
  description: base MiCADO policy defining data sources, constants, queries,
  properties:
    alerts:
      type: list
      description: pre-define alerts for container CPU
      default:
      - alert: service_overloaded
        expr: 'avg(rate(container_cpu_usage_seconds_total{container_label_io_
        for: 30s
      - alert: service_underloaded
        expr: 'avg(rate(container_cpu_usage_seconds_total{container_label_io_
        for: 30s
      required: true
    scaling_rule:
      type: string
      description: pre-define scaling rule for container CPU
      default: |
        if len(m_nodes) == m_node_count:
          if service_overloaded and m_node_count > m_container_count:
            m_container_count+=1
          if service_underloaded:
            m_container_count-=1
        else:
          print('Transient phase, skipping update of containers...')
      required: true
```

Custom Python code



```
tosca.policies.Scaling.MiCADO.VirtualMachine.CPU.stressng:
  derived_from: tosca.policies.Scaling.MiCADO
  description: base MiCADO policy defining data sources, constants, queries, alert
  properties:
    alerts:
      type: list
      description: pre-define alerts for VM CPU
      default:
      - alert: node_overloaded
        expr: '(100-(avg(rate(node_cpu_seconds_total{node="{{ NODE_NAME }}", mode=
        for: 1m
      - alert: node_underloaded
        expr: '(100-(avg(rate(node_cpu_seconds_total{node="{{ NODE_NAME }}", mode=
        for: 1m
      required: true
    scaling_rule:
      type: string
      description: pre-define scaling rule for VM CPU
      default: |
        if len(m_nodes) <= m_node_count and m_time_since_node_count_changed > 60:
          if node_overloaded:
            m_node_count+=1
          if node_underloaded:
            m_node_count-=1
        else:
          print('Transient phase, skipping update of nodes...')
      required: true
```

Support for a large variety of clouds

ORACLE

amazon web services

Azure

egi

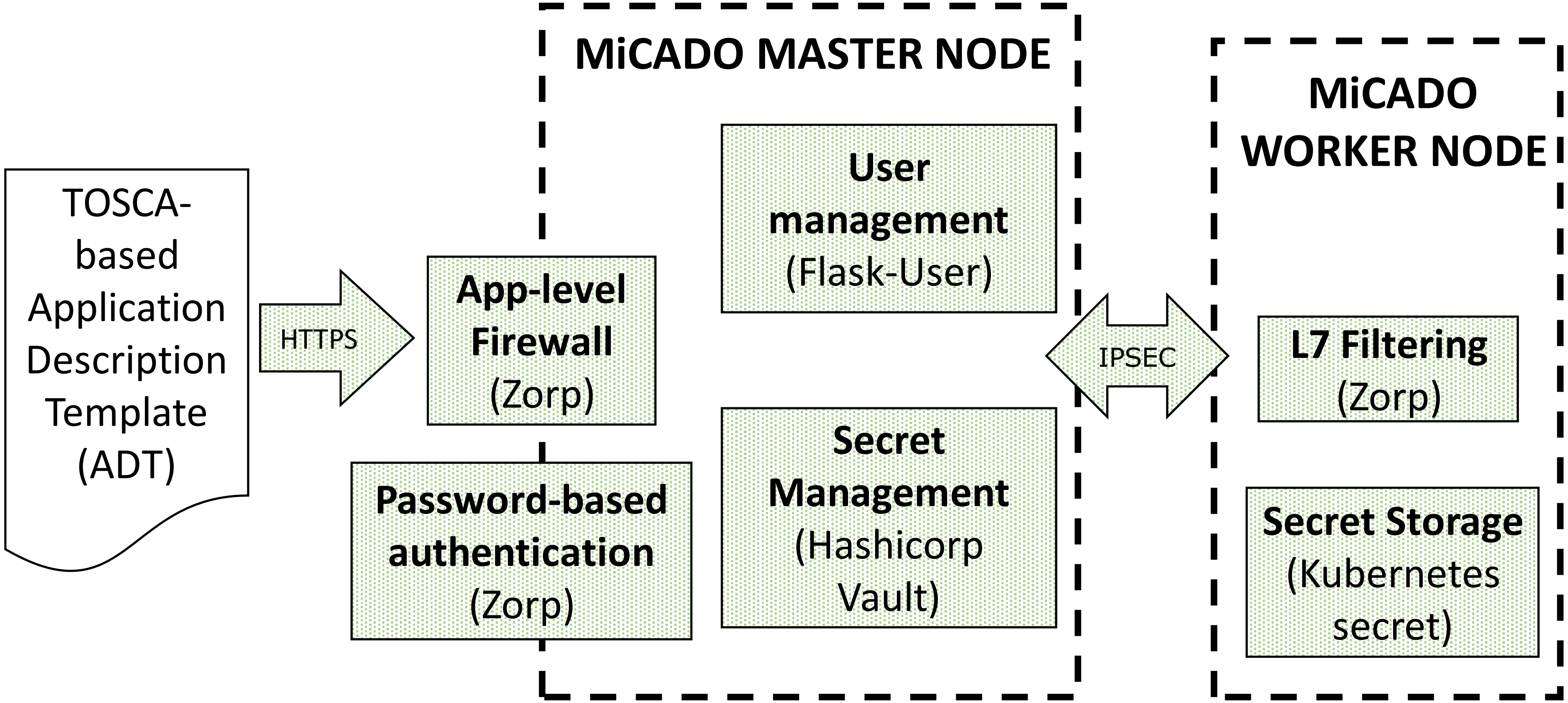
openstack®

Google Cloud

CloudSigma

CloudBroker

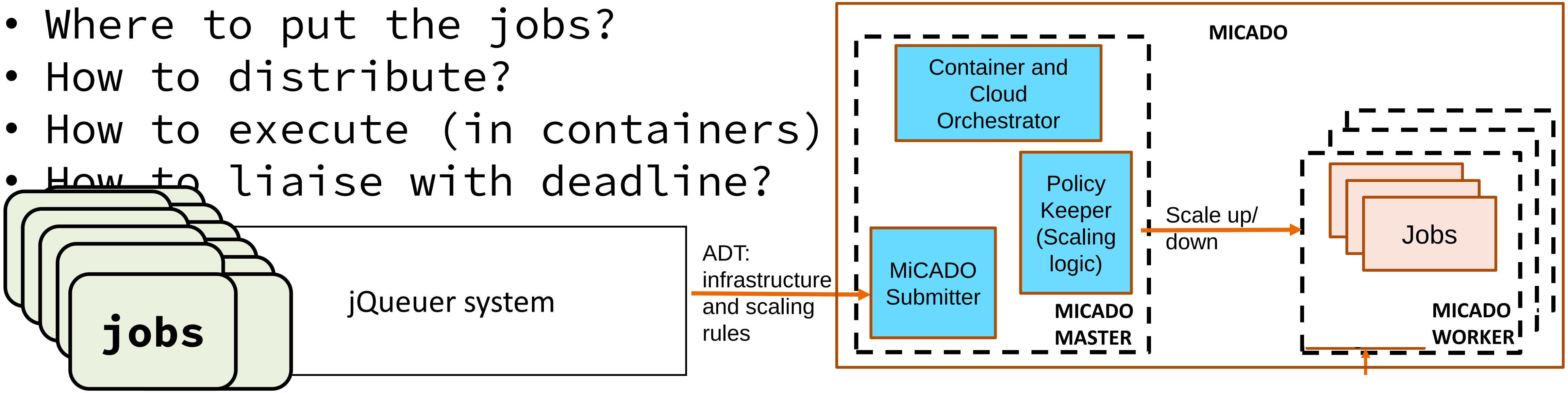
Advanced security features


TOSCA-based Application Description Template (ADT)
HTTPS
MiCADO MASTER NODE
App-level Firewall (Zorp)
User management (Flask-User)
Password-based authentication (Zorp)
Secret Management (Hashicorp Vault)
IPSEC
MiCADO WORKER NODE
L7 Filtering (Zorp)
Secret Storage (Kubernetes secret)

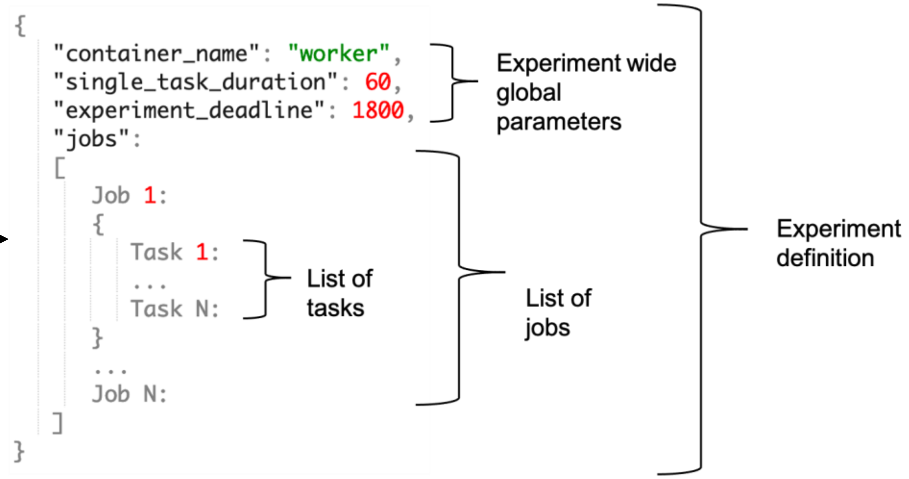
- implements **industry-standard** best practices
- provides **security functions lacking** in most cloud environments
- minimize the need of **user-supplied configuration**
- **pluggable** architecture
- validated by **penetration** testing

MiCADO and job execution

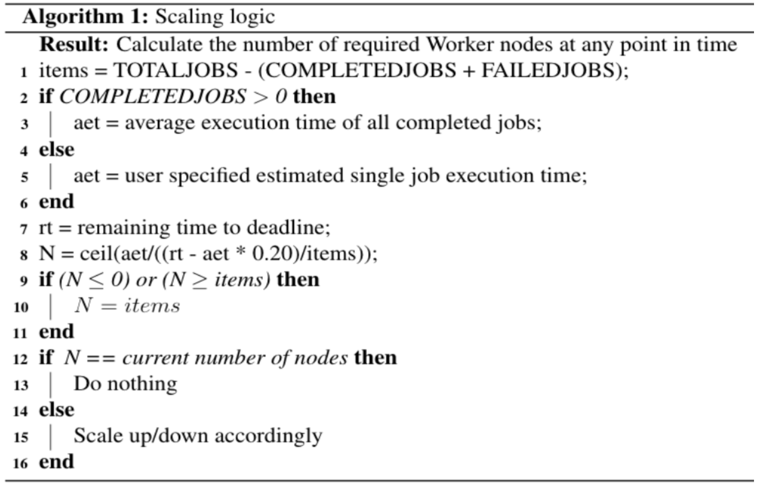
- Large number of jobs results in significant overall execution time
- Usually Restricted to complete all jobs by a deadline
  - Where to put the jobs?
  - How to distribute?
  - How to execute (in containers)
  - How to liaise with deadline?


jobs
jQueuer system
ADT: infrastructure and scaling rules
MICADO
Container and Cloud Orchestrator
Policy Keeper (Scaling logic)
MiCADO Submitter
MICADO MASTER
Scale up/ down
Jobs
MICADO WORKER

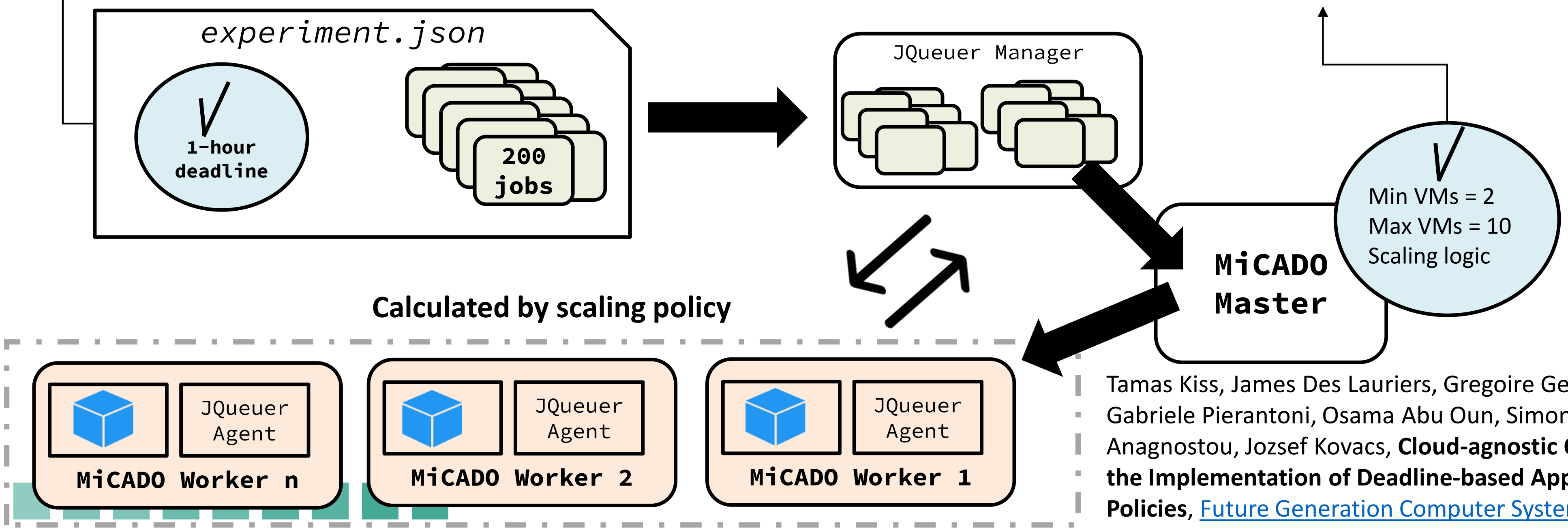
MiCADO and job execution



```
{
  "container_name": "worker",
  "single_task_duration": 60,
  "experiment_deadline": 1800,
  "jobs":
  [
     Job 1:
     {
        Task 1:
        ...
        Task N:
     }
     ...
     Job N:
  ]
}
```



```
Algorithm 1: Scaling logic
Result: Calculate the number of required Worker nodes at any point in time
items = TOTALJOBS - (COMPLETEDJOBS + FAILEDJOBS);
if COMPLETEDJOBS > 0 then
    aet = average execution time of all completed jobs;
else
    aet = user specified estimated single job execution time;
end
rt = remaining time to deadline;
N = ceil(aet/((rt - aet * 0.20)/items));
if (N ≤ 0) or (N ≥ items) then
    N = items
end
if N == current number of nodes then
    Do nothing
else
    Scale up/down accordingly
end
```

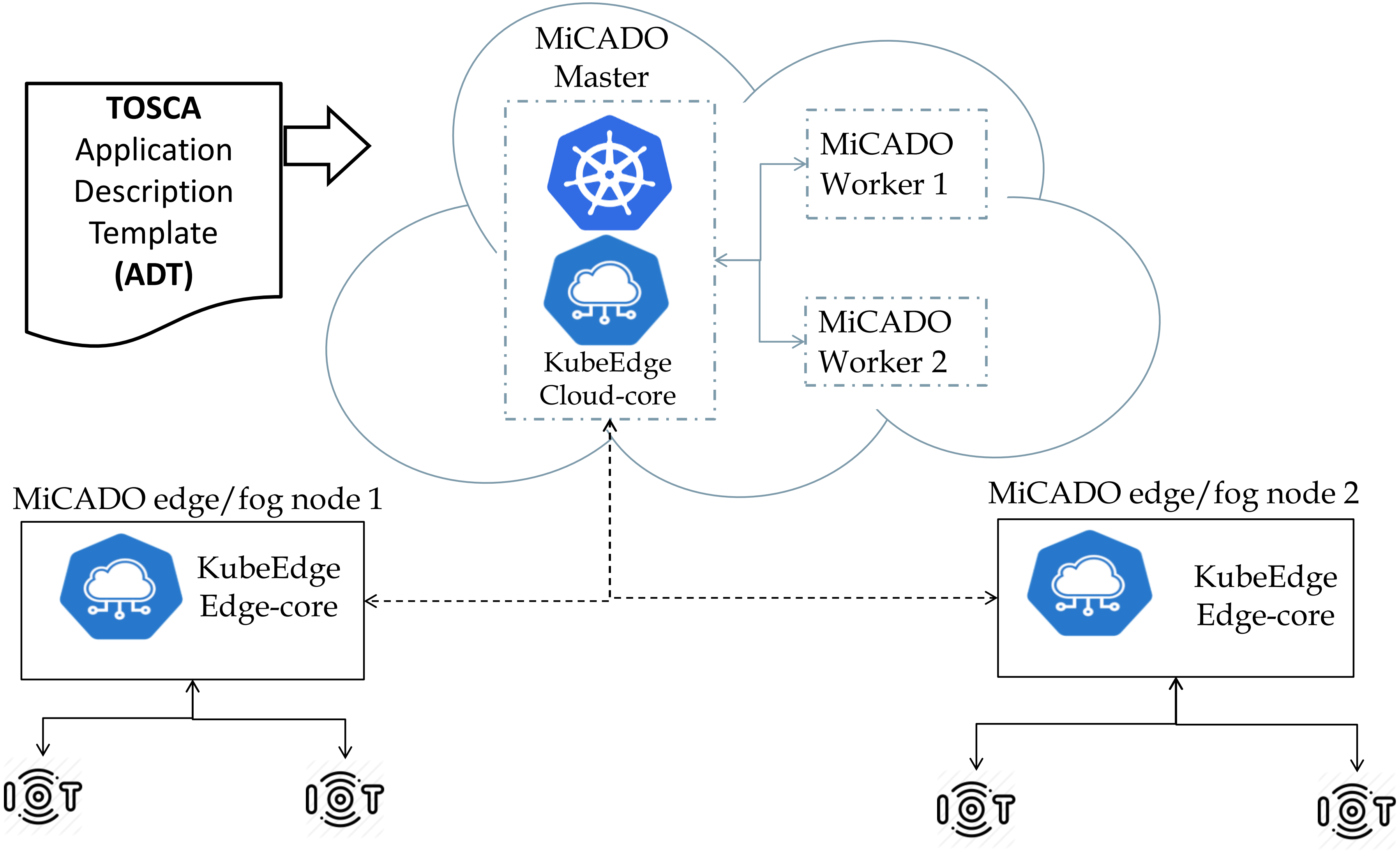


Calculated by scaling policy

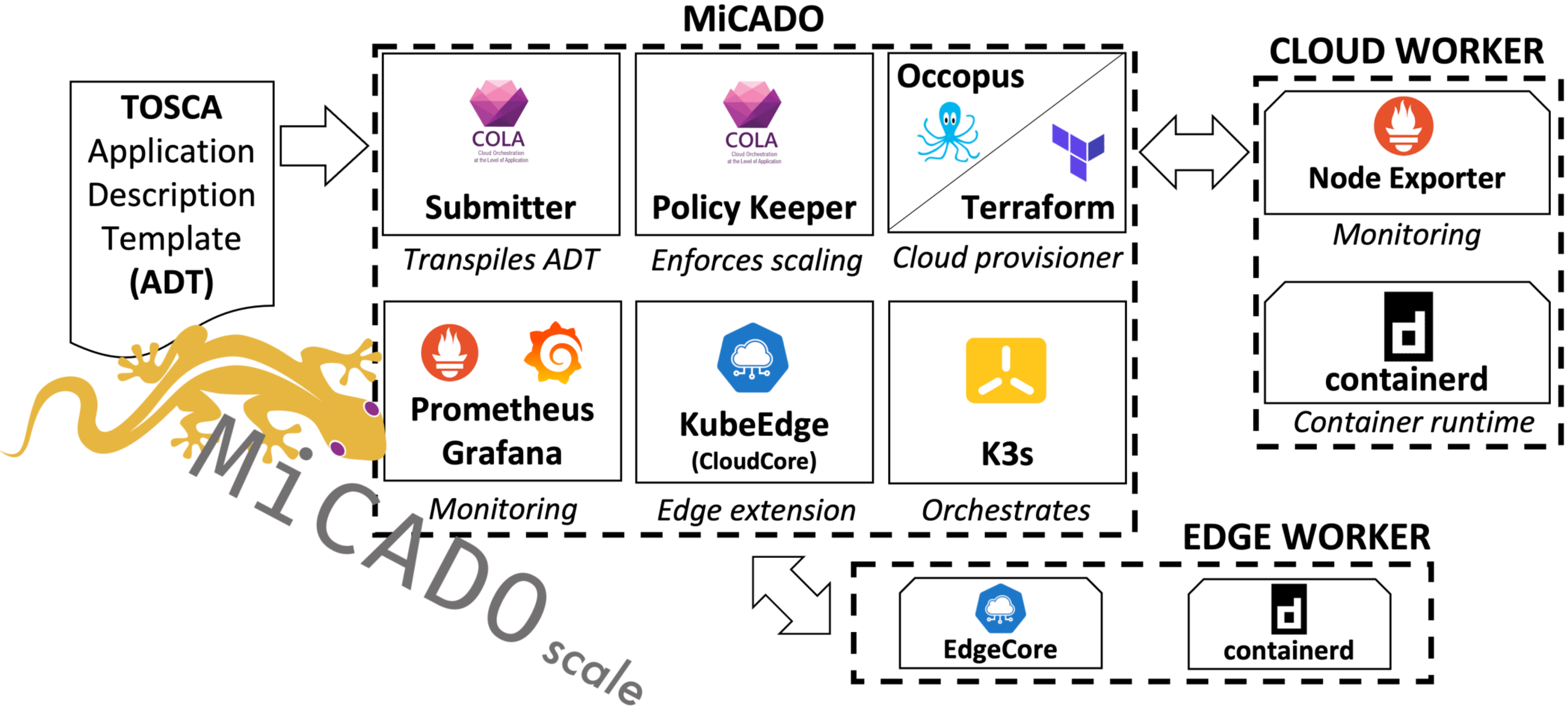
Tamas Kiss, James Des Lauriers, Gregoire Gesmier, Gabor Terstyanszky, Gabriele Pierantoni, Osama Abu Oun, Simon JE Taylor, Anastasia Anagnostou, Jozsef Kovacs, **Cloud-agnostic Queuing System to Support the Implementation of Deadline-based Application Execution Policies**, Future Generation Computer Systems, Elsevier, Vol 101, December, 2019, pp 99-111

MiCADO EDGE/FOG Extension

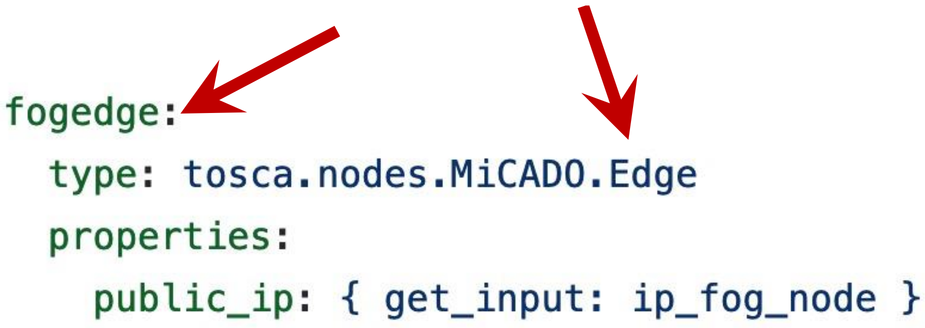
- □ Solution using KubeEdge
- □ Automated deployment of microservices extended to edge nodes
- □ Monitoring information collected from edge workers
- □ Scaling/reconfiguration policies extended towards edge



Architecture with Edge extension



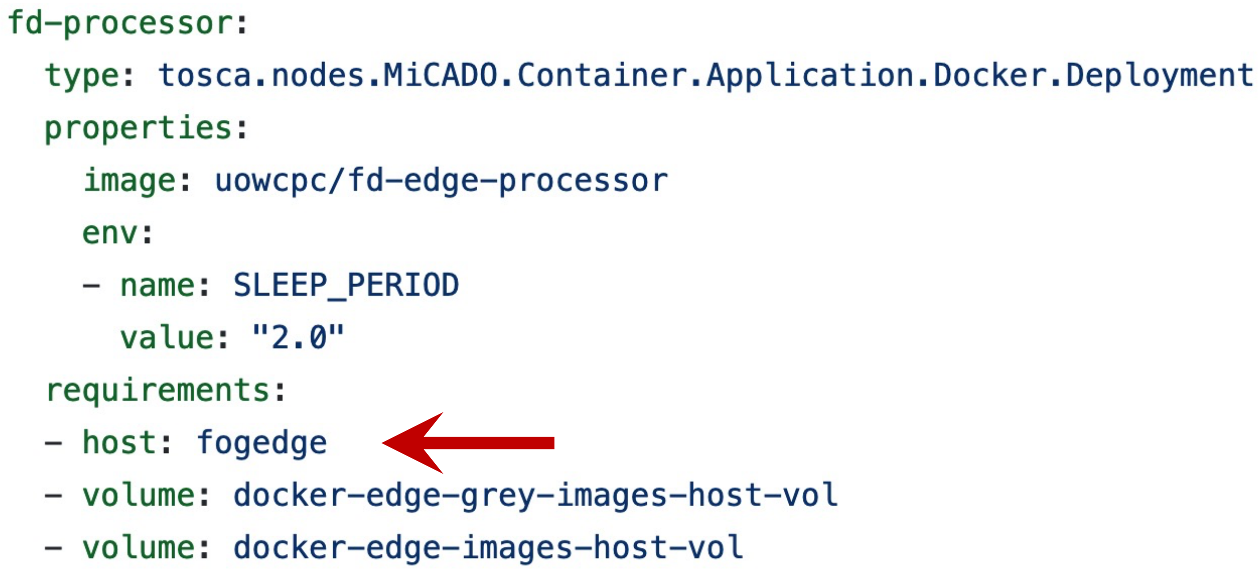
Now with a single uniform descriptor (ADT), the entire Cloud-to-Edge application can be described, E.g.



```
fogedge:
  type: tosca.nodes.MiCADO.Edge
  properties:
    public_ip: { get_input: ip_fog_node }
```

Edge Node ↑

auto-deployment of edge through ADT

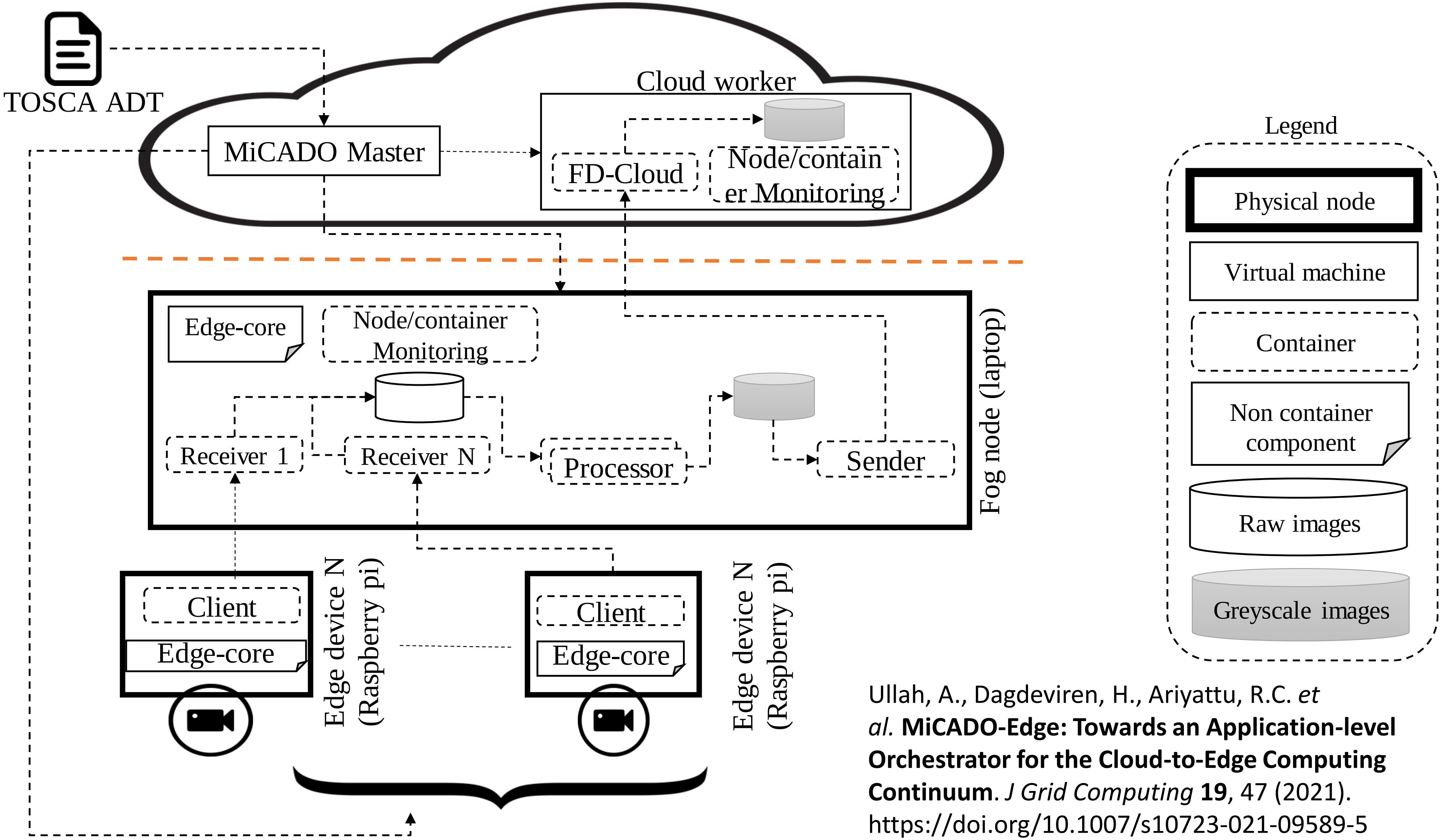


```
fd-processor:
  type: tosca.nodes.MiCADO.Container.Application.Docker.Deployment
  properties:
    image: uowcpc/fd-edge-processor
    env:
    - name: SLEEP_PERIOD
      value: "2.0"
  requirements:
  - host: fogedge
  - volume: docker-edge-grey-images-host-vol
  - volume: docker-edge-images-host-vol
```

Service Container ↑

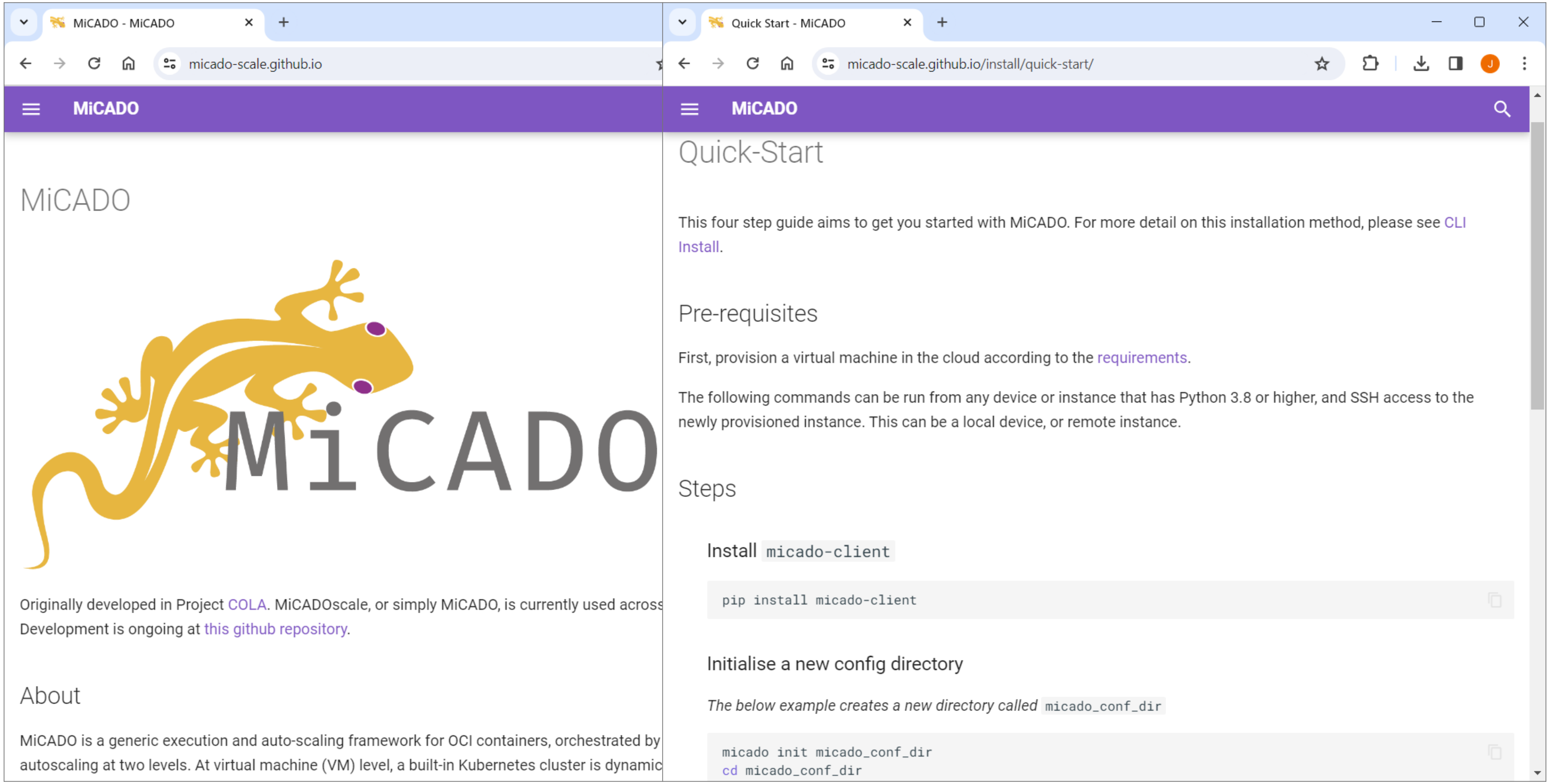
MiCADO edge/fog extension – face detection application

- Cloud server stores images with faces
- Fog node recognises faces in images
- Edge device captures video stream



Ullah, A., Dagdeviren, H., Ariyattu, R.C. *et al.* **MiCADO-Edge: Towards an Application-level Orchestrator for the Cloud-to-Edge Computing Continuum**. *J Grid Computing* **19**, 47 (2021). https://doi.org/10.1007/s10723-021-09589-5

Deployment: Step 1: install micado-client



MiCADO

Originally developed in Project COLA. MiCADOscale, or simply MiCADO, is currently used across
Development is ongoing at this github repository.

About

MiCADO is a generic execution and auto-scaling framework for OCI containers, orchestrated by
autoscaling at two levels. At virtual machine (VM) level, a built-in Kubernetes cluster is dynamic

Quick-Start

This four step guide aims to get you started with MiCADO. For more detail on this installation method, please see CLI Install.

Pre-requisites

First, provision a virtual machine in the cloud according to the requirements.

The following commands can be run from any device or instance that has Python 3.8 or higher, and SSH access to the newly provisioned instance. This can be a local device, or remote instance.

Steps

Install `micado-client`

```
pip install micado-client
```

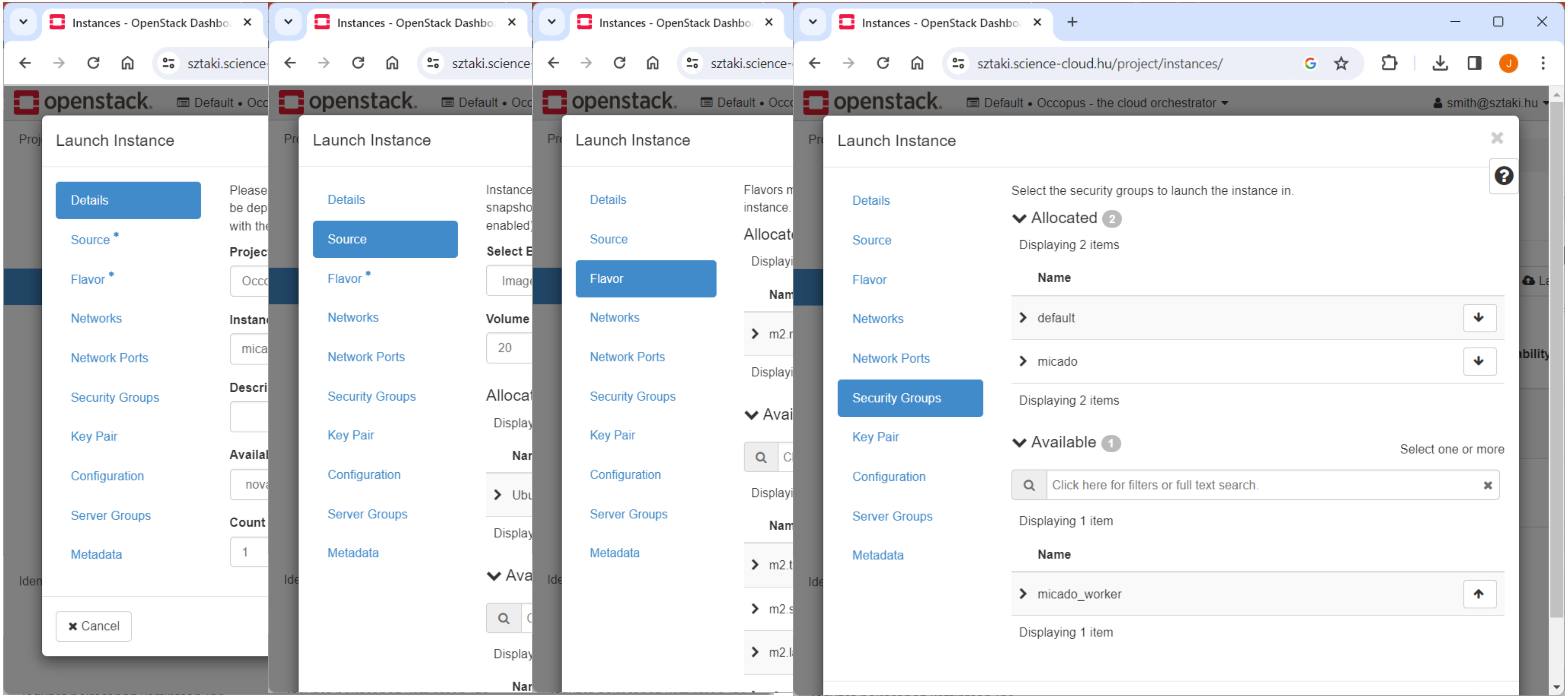
Initialise a new config directory

The below example creates a new directory called `micado_conf_dir`

```
micado init micado_conf_dir
cd micado_conf_dir
```

HUN-REN Cloud

Step 2: launch a new VM for MiCADO



Select the security groups to launch the instance in.

Allocated 2

Displaying 2 items

Name
default
micado

Displaying 2 items

Available 1

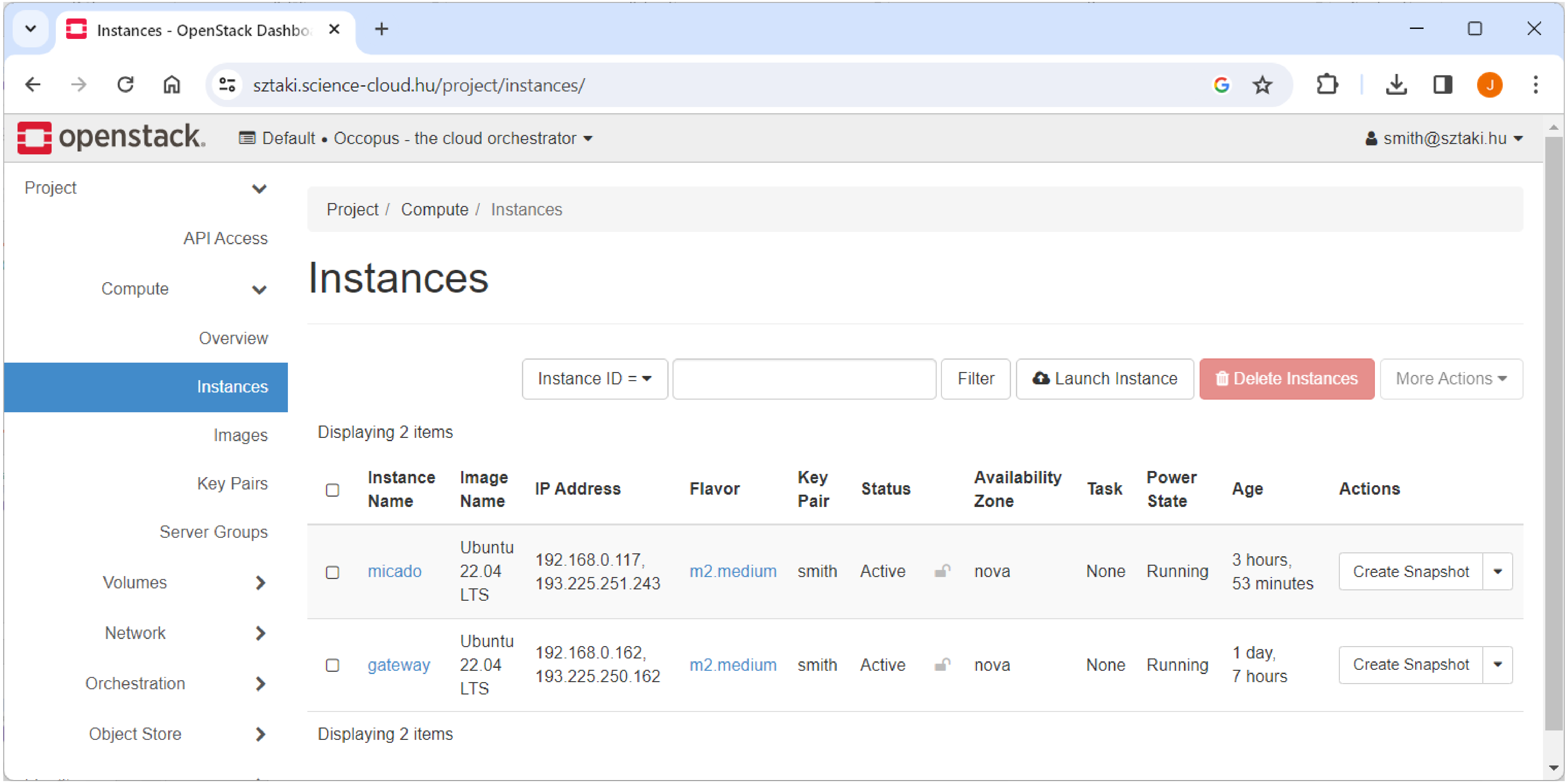
Select one or more

Click here for filters or full text search.

Displaying 1 item

Name
micado_worker

Displaying 1 item



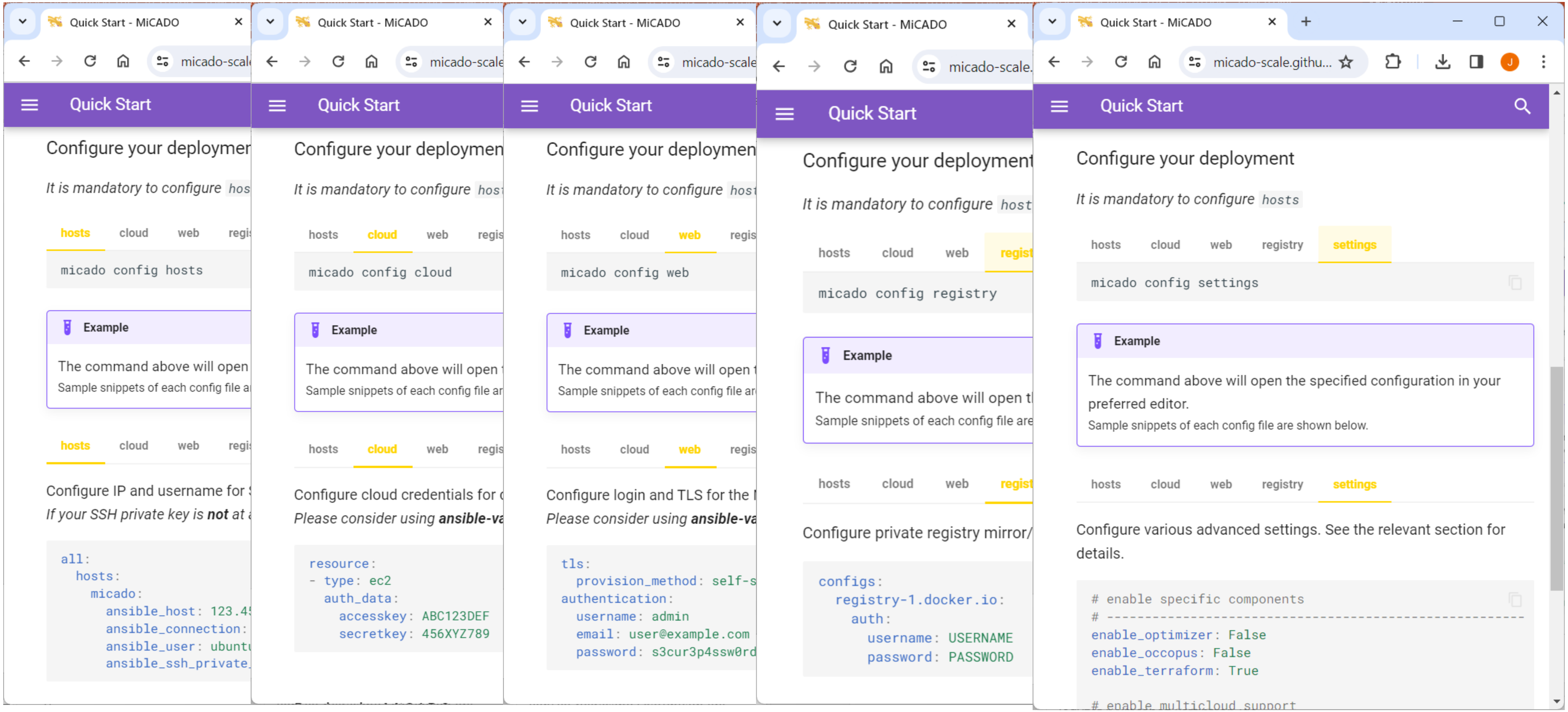
Instances

Displaying 2 items

□	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status		Availability Zone	Task	Power State	Age	Actions
□	micado	Ubuntu 22.04 LTS	192.168.0.117, 193.225.251.243	m2.medium	smith	Active	🔓	nova	None	Running	3 hours, 53 minutes	Create Snapshot ▾
□	gateway	Ubuntu 22.04 LTS	192.168.0.162, 193.225.250.162	m2.medium	smith	Active	🔓	nova	None	Running	1 day, 7 hours	Create Snapshot ▾

Displaying 2 items

Step 3: configure 5 groups of details



Configure your deploymer

It is mandatory to configure `hos`

```
micado config hosts
```

The command above will open
Sample snippets of each config file a

Configure IP and username for !
If your SSH private key is ***not*** *at a*

```
all:
  hosts:
    micado:
      ansible_host: 123.45
      ansible_connection:
      ansible_user: ubuntu
      ansible_ssh_private_
```

Configure your deploymen

It is mandatory to configure `host`

```
micado config cloud
```

The command above will open
Sample snippets of each config file ar

Configure cloud credentials for c
Please consider using ***ansible-va***

```
resource:
- type: ec2
  auth_data:
    accesskey: ABC123DEF
    secretkey: 456XYZ789
```

Configure your deploymen

It is mandatory to configure `host`

```
micado config web
```

The command above will open
Sample snippets of each config file ar

Configure login and TLS for the l
Please consider using ***ansible-va***

```
tls:
  provision_method: self-s
authentication:
  username: admin
  email: user@example.com
  password: s3cur3p4ssw0rd
```

Configure your deployment

It is mandatory to configure `host`

```
micado config registry
```

The command above will open t
Sample snippets of each config file are

Configure private registry mirror/

```
configs:
  registry-1.docker.io:
    auth:
      username: USERNAME
      password: PASSWORD
```

Configure your deployment

It is mandatory to configure `hosts`

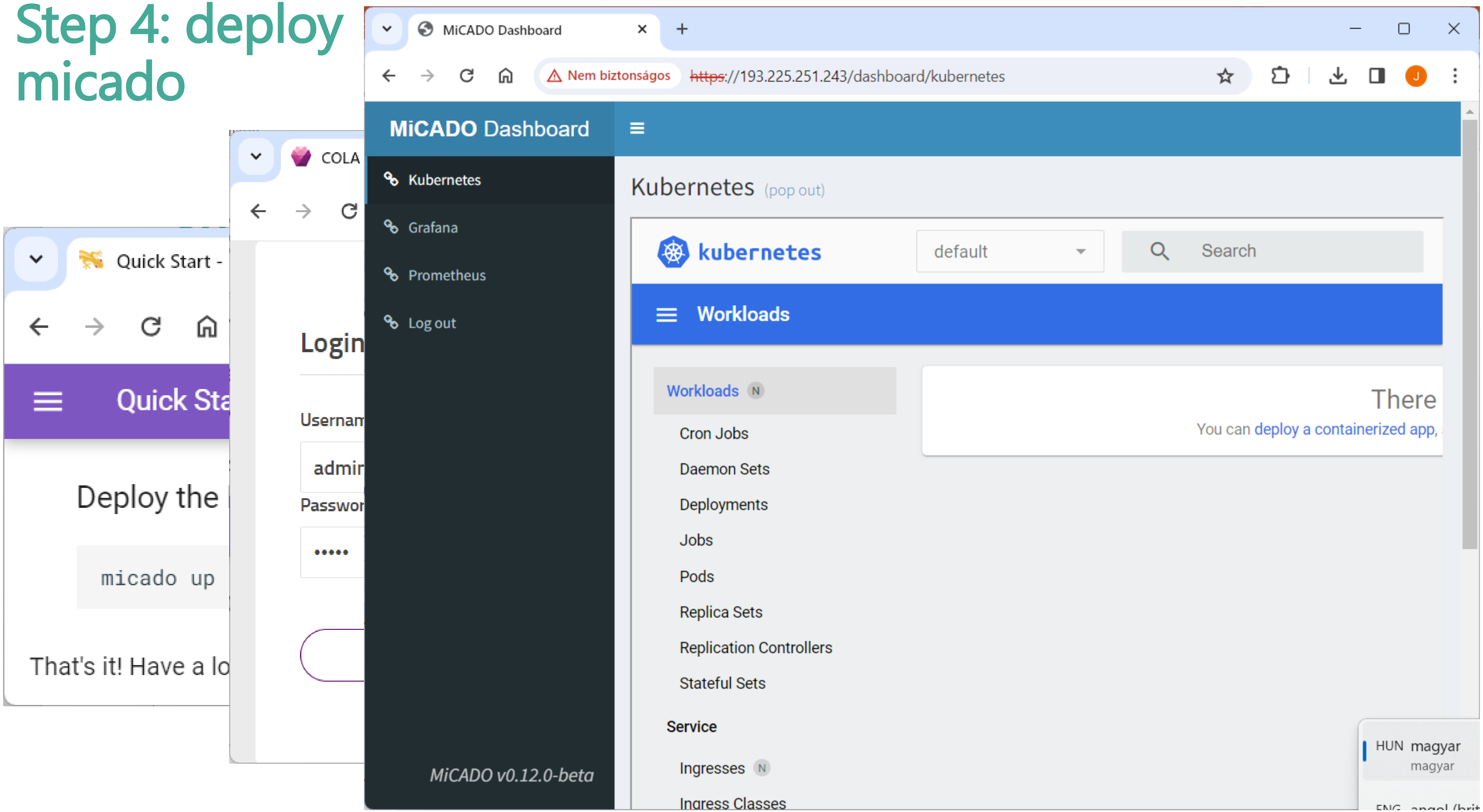
```
micado config settings
```

The command above will open the specified configuration in your preferred editor.
Sample snippets of each config file are shown below.

Configure various advanced settings. See the relevant section for details.

```
# enable specific components
# ------------------------------------------------------
enable_optimizer: False
enable_occopus: False
enable_terraform: True

# enable multicloud support
```

Step 4: deploy micado

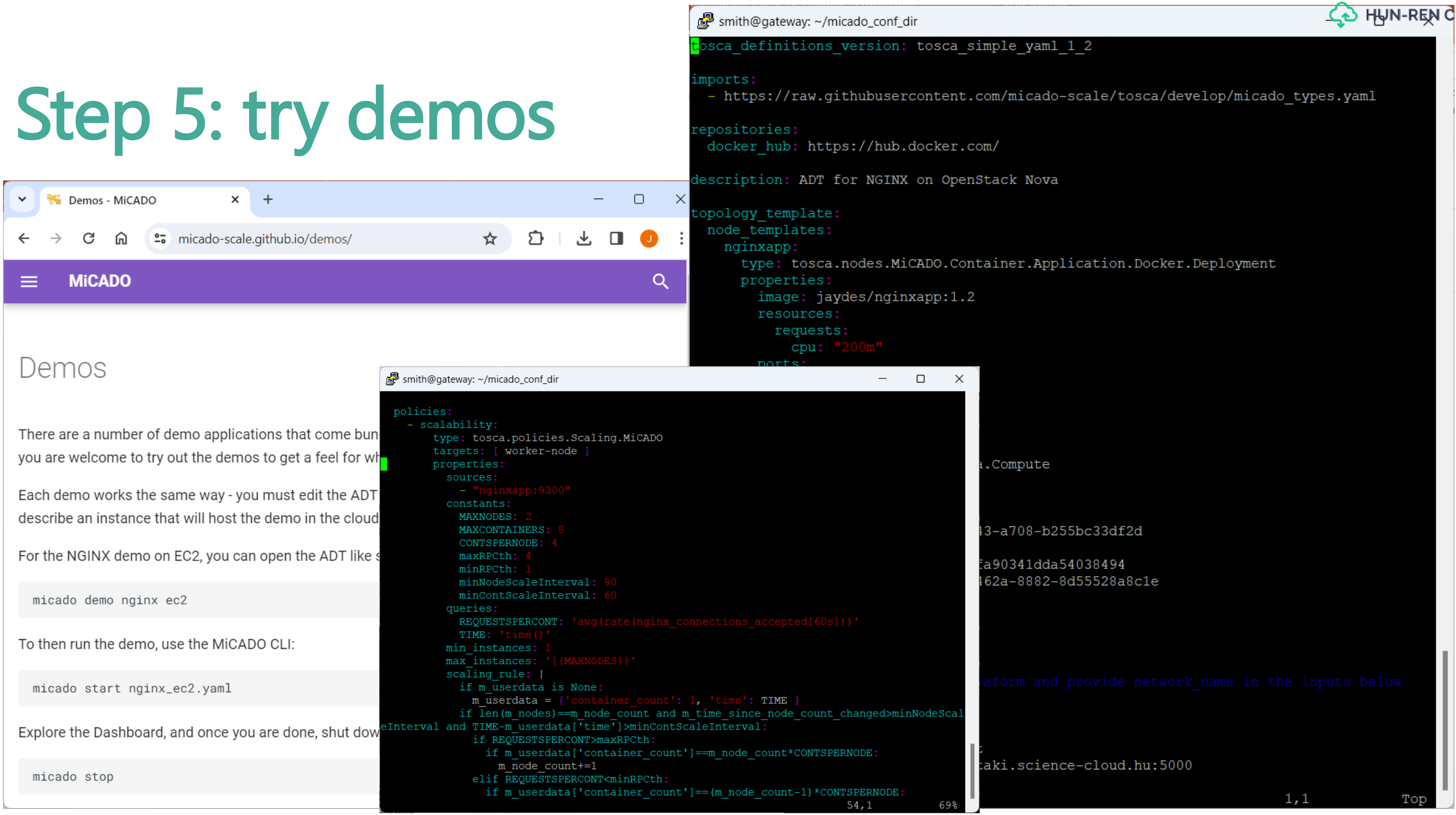


Deploy the

```
micado up
```

That's it! Have a lo

Step 5: try demos



Demos

There are a number of demo applications that come bun
you are welcome to try out the demos to get a feel for wh

Each demo works the same way - you must edit the ADT
describe an instance that will host the demo in the cloud

For the NGINX demo on EC2, you can open the ADT like s

```
micado demo nginx ec2
```

To then run the demo, use the MiCADO CLI:

```
micado start nginx_ec2.yaml
```

Explore the Dashboard, and once you are done, shut dow

```
micado stop
```

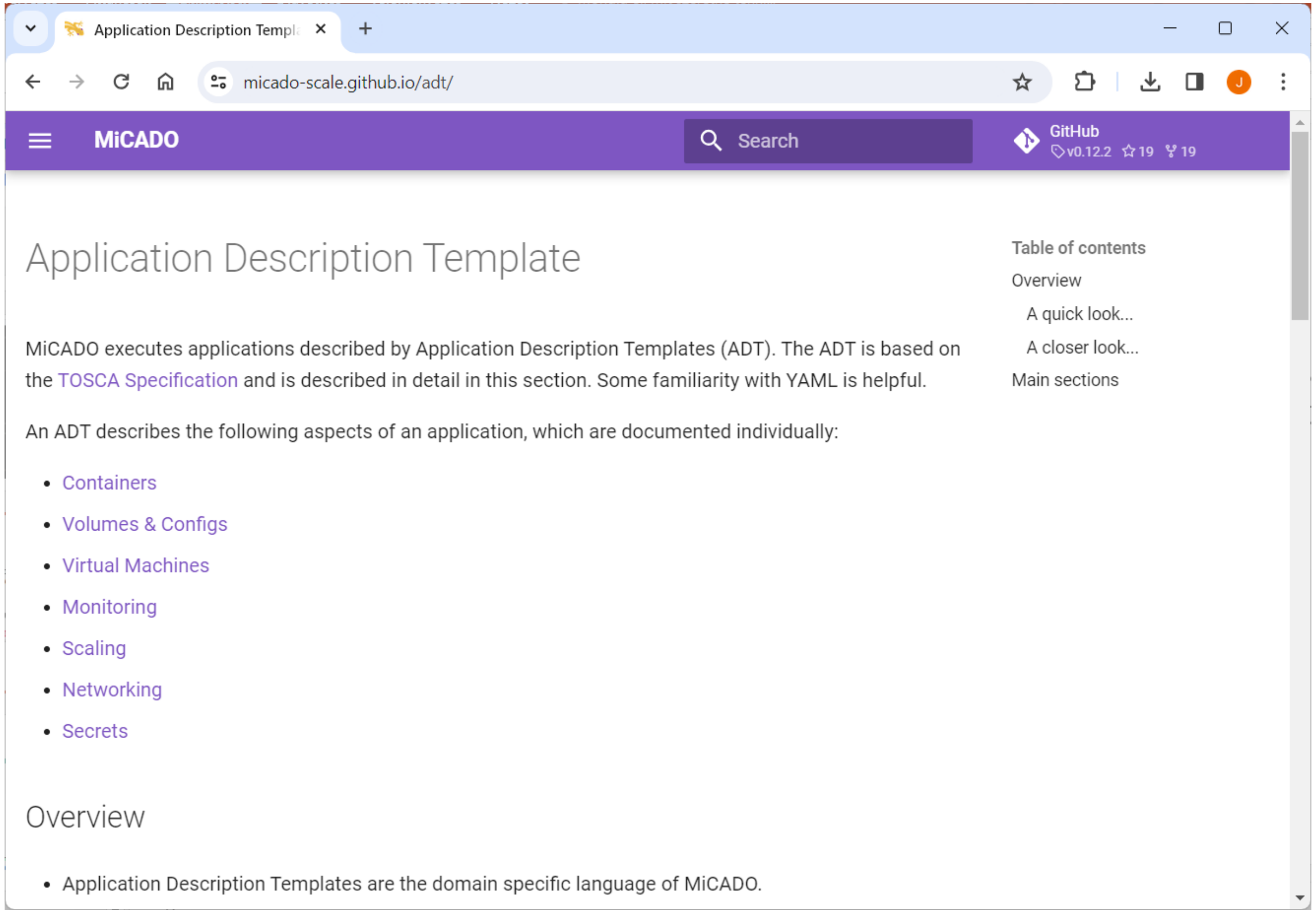
smith@gateway: ~/micado_conf_dir

```
tosca_definitions_version: tosca_simple_yaml_1_2

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

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

description: ADT for NGINX on OpenStack Nova

topology_template:
  node_templates:
    nginxapp:
      type: tosca.nodes.MiCADO.Container.Application.Docker.Deployment
      properties:
        image: jaydes/nginxapp:1.2
        resources:
          requests:
            cpu: "200m"
        ports:

                            a.Compute

                            43-a708-b255bc33df2d

                            fa90341dda54038494
                            462a-8882-8d55528a8c1e

                            aform and provide network name in the inputs below

                            aki.science-cloud.hu:5000
                                                                 1,1           Top
```

smith@gateway: ~/micado_conf_dir

```
policies:
  - scalability:
      type: tosca.policies.Scaling.MiCADO
      targets: [ worker-node ]
      properties:
        sources:
          - "nginxapp:9300"
        constants:
          MAXNODES: 2
          MAXCONTAINERS: 8
          CONTSPERNODE: 4
          maxRPCth: 4
          minRPCth: 1
          minNodeScaleInterval: 90
          minContScaleInterval: 60
        queries:
          REQUESTSPERCONT: 'avg(rate(nginx_connections_accepted[60s]))'
          TIME: 'time()'
        min_instances: 1
        max_instances: '{{MAXNODES}}'
        scaling_rule: |
          if m_userdata is None:
            m_userdata = {'container_count': 1, 'time': TIME }
          if len(m_nodes)==m_node_count and m_time_since_node_count_changed>minNodeScal
eInterval and TIME-m_userdata['time']>minContScaleInterval:
            if REQUESTSPERCONT>maxRPCth:
              if m_userdata['container_count']==m_node_count*CONTSPERNODE:
                m_node_count+=1
            elif REQUESTSPERCONT<minRPCth:
              if m_userdata['container_count']==(m_node_count-1)*CONTSPERNODE:
                                                                54,1          69%
```

Step 6: build your own application



Application Description Template

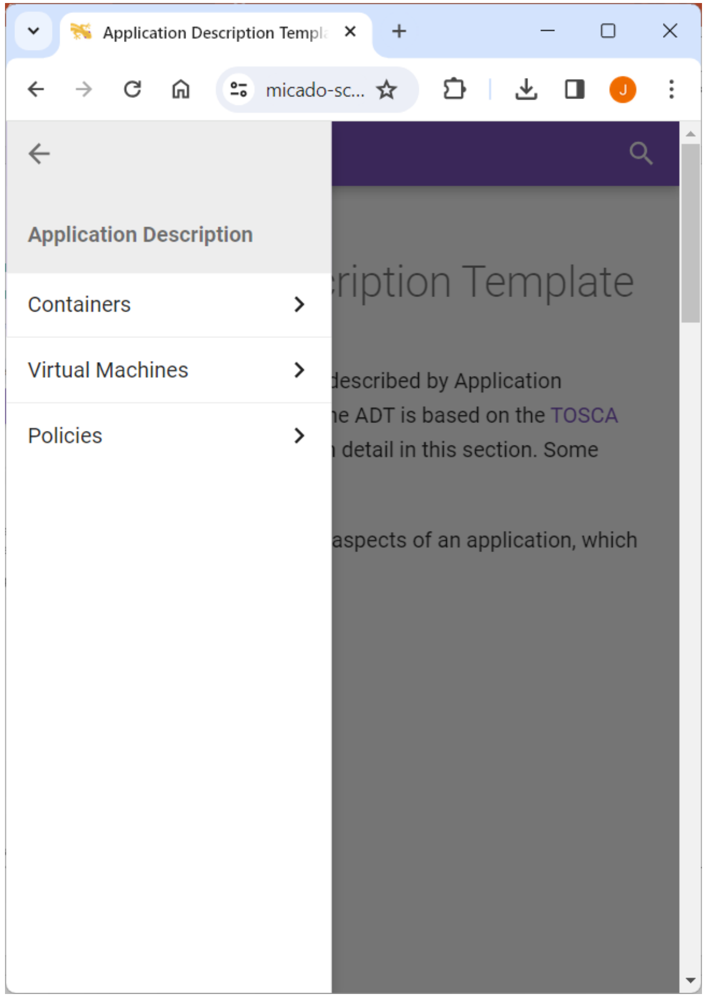
MiCADO executes applications described by Application Description Templates (ADT). The ADT is based on the TOSCA Specification and is described in detail in this section. Some familiarity with YAML is helpful.

An ADT describes the following aspects of an application, which are documented individually:

- Containers
- Volumes & Configs
- Virtual Machines
- Monitoring
- Scaling
- Networking
- Secrets

Overview

- Application Description Templates are the domain specific language of MiCADO.


Application Description Templ
micado-sc...
Application Description
Containers
Virtual Machines
Policies
ription Template
described by Application
he ADT is based on the TOSCA
detail in this section. Some
aspects of an application, which

https://micado-scale.github.io

Summary

- Cloud-agnostic orchestration solution
- Pluggable architecture based on open-source components
- Standardised TOSCA-based application and policy description
- Automated application deployment in clouds
- Support for highly customisable scaling policies
- Support for large variety of clouds

CloudSigma

ORACLE

amazon web services

Google Cloud

Azure

egi

CloudBroker

openstack.

https://micado-scale.github.io

MiCADO
scale

Thank you for your interest!

https://micado-scale.github.io

presenter, designer, contributor: Jozsef Kovacs jozsef.kovacs@sztaki.hun-ren.hu
project leader: Tamas Kiss t.kiss@westminster.ac.uk
main developer: Jay Deslauriers j.deslauriers@westminster.ac.uk

Acknowledgement:
the work and slides presented were provided by the members of Centre of Parallel Computing at The University of Westminster