OCCI-compliant Occopus orchestrator and experiences with using it with the EGI Federated Cloud

Jozsef Kovacs and Peter Kacsuk
MTA SZTAKI
Motivations for Occopus – End user view

• There are many use cases where the goal is to set up a complete infrastructure or a set of services (called as virtual infrastructures) on demand in the cloud.

• Goal is to enable end-user scientists to easily (ideally by 1 click) deploy such a VI in the cloud either temporarily or for a longer period of time.

• Example:
  o biologist would like to run autodock in-silico simulation between one receptor molecule and 100.000 ligand molecules
  o The biologist takes the autodock VI from EGI AppDB and by 1 click deploys it in a target cloud and uses it.
Motivations for Occopus – VI developer view

• Enable virtual infrastructure developers to develop the required VIs and VI descriptors for various cloud types

• Requirements:
  o Easy and fast development of VIs without very deep cloud expertise
  o Easy specification/description of the required VI
  o Fast and automatic deployment of the VI specified by the VI developer
  o Fault-tolerant life-cycle management of deployed virtual machines
  o Cloud type independent solution
**Example: Creating docker cluster in cloud**

**Step 1:** Take VI descriptor of docker cluster VI from EGI AppDB

**Step 2:** Deploy docker cluster VI by 1 click in EGI Cloud

**Step 3:** Deploy docker containers in the docker cluster
Main features of Occopus

- Command line tool and REST API service
- Multi-cloud support
- Pluggable architecture → e.g. connecting to a new cloud requires only a new plugin
- Error-detection (fatal/transient) and recovery
- Support for configuration management tools (like Chef)
- Garbage collection at VM cancellation
- Manual scaling
- Serial and parallel synchronization strategies among nodes of a VI
How does Occopus work?

- At deployment time the dependencies among nodes are managed.
- Required communication among nodes are handled to enable their interaction and collaboration.
- If Occopus works as a service it manages the whole life-cycle of the nodes.
How to describe your infrastructure

• Occopus requires 2 descriptions:
  o **Virtual infrastructure description:**
    • Specifies the **nodes** (services) to be deployed and all **cloud-independent** attributes e.g. input values for a service.
    • Specifies the **dependencies** among the nodes, to decide the order of deployment
    • Specifies **scaling** related attributes like min, max number of instances
  o **Node definition:**
    • Defines **how to construct the node** on a target cloud. This contains all **cloud dependent** settings, e.g. image id, flavour, contextualization (complete: could be deployed as a 1-node VI)

• See detailed tutorials at the Occopus web page:
  o [http://occopus.lpds.sztaki.hu/tutorials](http://occopus.lpds.sztaki.hu/tutorials)
Infrastructure description in Occopus

List of nodes

Type refers to an implementation in node definition

Dependencies specifies the required order of deployment

Infrastructure description:

```yaml
nodes:
  - &DBS_Node
ame: mysql_server
type: ec2_chef_mysql_server_node
  - &WP_Node
name: wordpress
type: ec2_chef_wordpress_node
dependencies:
  - connection: [ *WP_Node, *DBS_Node ]
```
Node definition:

```
'node_def:ec2_chef_mysql_server_node':

resource:
  type: ec2
  endpoint: http://cfe2.lpds.sztaki.hu:4567
  regionname: ROOT
  image_id: ami-00001441
  instance_type: m1.medium

contextualisation:
  type: cloudinit
  context_template: !text_import
    url: file://cloud_init_wordpress.yaml
    attributes:
      mysql:
        server_root_password: '{{ variables.mysql_root_password }}'

config_management:
  type: chef
  endpoint: https://c155-14.localcloud
  run_list:
    - recipe[database-setup::db]

health_check:
  mysql dbs:
    - {name: my_DB,
       user: my_user,
       pass: '{{ variables.mysql_dbuser_password }}'}
```

‘resource’ section **obligatory**: details the attributes needed for instantiation of a node

‘contextualization’ section specifies cloud-init config file and attributes for chef recepies

‘config_management’ section specifies chef server and recepies to run

‘health_check’ section specifies how to check node health
• Occopus can handle OCCI cloud interface to utilize EGI FedCloud resources
• Nodes of deployable infrastructure are instantiated on the target cloud through OCCI and used Cloud-init for contextualisation.
Occopus can utilise multiple clouds in a federation like **EGI FedCloud**

- Nodes of deployable VI are instantiated on different FedCloud sites
- Connection is based on public ips
Pluggable Occopus architecture

**Occopus**

- **Resource (handlers)**: EC2, NOVA, OCCI, CloudBroker, Docker
- **Contextualisation**: Cloudinit, CloudBroker, Docker
- **Config managers**: Chef
- **Health Checking**: Ping, Port, Url, Mysql
OCCI resource attributes in Occopus

**type: occi**
Selects the occi resource handler. It requires the occi client to be installed locally.

**endpoint**
Occi site endpoint.

**resource_tpl**
Template ID.

**os_tpl**
OCCI ID.

**public_key**
Optional. The public ssh key to be deployed on the target virtual machine.

**link**
Optional. List of compute or network resources to be attached to the VM. Using this option enables one to attach additional disk images or public networks to the VM.

**name**
Optional. A user-defined name for this resource. Used in logging and can be referred to in the authorisation file.
In this example, we create a swarm/docker head (H) node and two swarm/docker worker (W) nodes in the CESNET EGI FedCloud.
Inside the Docker cluster nodes

**Head node**
- Consul agent (on port 8500)
- Swarm manage (on port 3375)
- Docker (on port 2375)

**Worker node(s)**
- Consul agent (on port 8500) joins to head:8500
- Swarm join (on port 3375) joins to head:3375
- Docker (on port 2375)
- Containers
Creating a Docker Swarm cluster

**Infrastructure description:**

```yaml
nodes:
  - &H
    name: head
    type: occi_dockerswarm_head_node
  - &W
    name: worker
    type: occi_dockerswarm_worker_node
  scaling:
    min: 2

dependencies:
  - [*W, *H]
```

**Node definition for the head node:**

```yaml
'node_def:occi_dockerswarm_head_node':
  resource:
    type: occi
    endpoint: https://carach5.ics.muni.cz:11443
    os_tpl: os_tpl#uuid_egiUbuntu_server_14_04_lts_fedcloud_warg_131
    resource_tpl: http://fedcloud.egi.eu/occi/compute/flavour/1.0#medium
    link:
    public_key: /home/myaccount/.ssh/authorized_keys
  contextualisation:
    type: cloudinit
    context_template: !text_import
    url: file:///cloud_init_head_node.yaml

health_check:
  ports:
    - 2375
```

**Requirements:**

- 1 head and any (currently 2) number of worker nodes
- Scaling min parameter is 2 means 2 instances will be deployed at startup
- Workers started after head

**Node definition for the head node:**

- **OCCI IDs for a base Ubuntu image**
- **Attach a network to get a public IP**
- **Add public ssh key**
- **Cloud init script to setup Swarm Head node**
- **Ready when Swarm port is available**
Creating a Docker Swarm cluster

Infrastructure description:

- **name**: head
  - **type**: occi_dockerswarm_head_node
- **name**: worker
  - **type**: occi_dockerswarm_worker_node
  - **scaling**: min: 2
  - **dependencies**: [*W, *H]

Require 1 head and any (currently 2) number of worker nodes.

Scaling min parameter is 2 means 2 instances will be deployed at startup.

Workers started after head.
Creating a Docker Swarm cluster

Node definition for the head node:

```
'node_def:occi_dockerswarm_head_node':
  resource:
    type: occi
    endpoint: https://carach5.ics.muni.cz:11443
    os_tpl:
      os_tpl#uuid_egi_ubuntu_server_14_04_lts_fedcloud_warg_131
      resource_tpl:
        http://fedcloud.egi.eu/occi/compute/flavour/1.0#medium
        link:
        public_key: /home/myaccount/.ssh/authorized_keys

  contextualisation:
    type: cloudinit
    context_template: !text_import
      url: file://cloud_init_head_node.yaml

  health_check:
    ports:
      - 2375
```

- **OCCI IDs for a base Ubuntu image**
- **Attach a network to get a public IP**
- **Add public ssh key**
- **Cloud init script to setup Swarm Head node**
- **Ready when Swarm port is available**
The resulted docker cluster

- Dynamically scaled up or down with Occopus
- Docker containers can be started
  - by docker commands
  - by Occopus (which can start containers by its docker plugin)
Example: Docker ping

Builds 1 receiver and 1 sender

Sender starts after Receiver

Node definition for the sender node:

```
'node_def:docker_ping_sender_node':
  resource:
    type: docker
    endpoint: tcp://192.168.153.52:2375
    origin: https://s3.lpds.sztaki.hu/docker/busybox_ping.tar
    image: busybox_ping
    network_mode: overlaynet
    tag: latest
  contextualisation:
    type: docker
    env: ["target_ip={getip("ping_receiver")}"]
    command: "sh -c /root/start.sh"
  health_check:
    ping: False
```

Infrastructure description:

```
nodes:
  - &R
    name: ping_receiver
    type: docker_ping_receiver_node
    variables:
      message: "Hello World! I am the receiver node."
  - &S
    name: ping_sender
    type: docker_ping_sender_node
    dependencies:
      - [ *S, *R ]
```

Attributes of previously built docker/swarm cluster and the container to be instantiated

Docker contextualization specifies command and environment variables

Docker container is not accessible from outside (in this example)
Nunzio Andrea Galante from d4science:

• “As far as the OCCI experimentation concerns, we have tested the functionalities proposed by Occopus.
• Indeed now we can
  o authenticate to the FedCloud VO,
  o create infra/VMs on OCCI-based sites,
  o specify properties (e.g., os templates, resource templates).
• By using the context script then, we are actually able to monitor the instances on d4Science infrastructure.”
Current state of Occopus

• Open-source (License: Apache v2)
• 3 beta releases previously
• **Now: Release v1.0 (first production release)**
• Python 2.7
• Base webpage: [http://occopus.lpds.sztaki.hu](http://occopus.lpds.sztaki.hu)
• Git: [https://github.com/occopus](https://github.com/occopus)
• Documentation:
  o Users’ Guide
  o Developers’ Guide
  o Tutorials
• Testing: Python *nose*tests*
• Deployment: *setuptools*
  o Package repository: [http://pip.lpds.sztaki.hu/packages](http://pip.lpds.sztaki.hu/packages)