Cloudify integration document

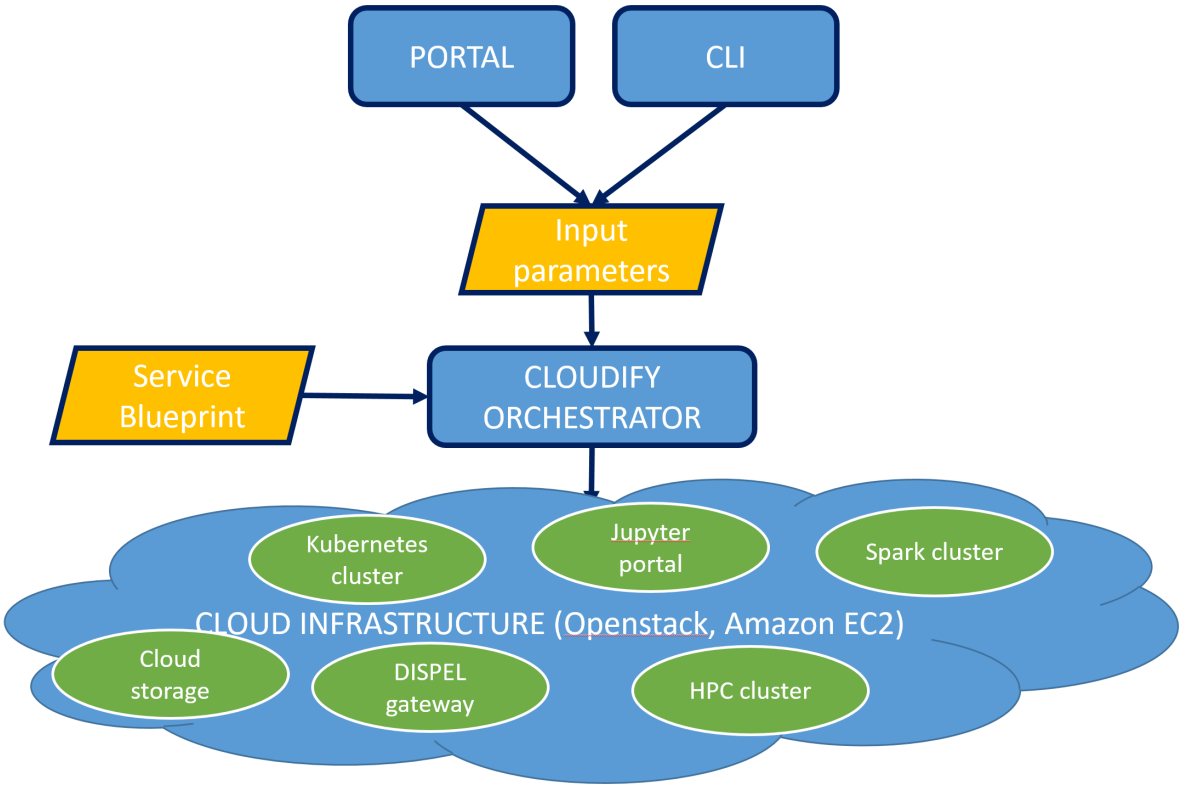
1. Basic terms of Cloudify

- Cloudify only manage/orchestrate services, not jobs. One of the main differences between services and jobs (e.g. on HPC platforms) are their lifetime. Jobs have short lifetime, they terminate when their computation finishes. Services have “unlimited” lifetime, they terminate only on explicit commands from owners. During their lifetime, services accept requests from users and make corresponding computations.
  Examples of services: Jupyter portal, DISPEL gateway, cloud storages, virtual clusters (HPC/Spark/Kubernetes)

- Blueprint: A package containing TOSCA template + corresponding scripts/data of service. Similar to application code (binaries/scripts) of jobs. Blueprints are uploaded to Cloudify manager server before service deployment.

- Deployment: An instance of blueprint. Deployments are NOT instantiated (installed/deployed) immediately after creation, only an abstractions /handles of the deployments are created.

- Execution (workflow execution): An action of a deployment. Typical workflows for executions are “install” which install/deploy the service and “uninstall”, that undeploy service. The workflows are created by default in every TOSCA template.



2. Typical lifecycle of a service in Cloudify

There are many advanced features for service orchestration, e.g. monitoring, auto-healing, auto-scaling, however, in the first step of integration, we would use the simplified lifecycle of the service as follows:

1. Implementing service blueprints: preparing TOSCA templates and service codes/data. Service-specific initialization/configuration scripts must be provided by service owners. See Section 5 "How to create blueprints for services" for more details.

2. Uploading blueprints to Cloudify manager: input: blueprint tar.gz package, output: blueprint ID.
https://docs.cloudify.co/api/v3.1/?shell#upload-blueprint

3. Creating deployments from blueprints: input: blueprint ID, blueprint input parameters, output: deployment ID.
https://docs.cloudify.co/api/v3.1/?shell#create-deployment

Example of creating an instance (deployment) of Jupyter portal at Cloudify server at IISAS via Cloudify REST API:

```
curl -X PUT --header "Tenant:default_tenant" --header "Content-Type: application/json" -u "USER:PASS"  -d
'{"blueprint_id": "jupyter-blueprint"}' "https://cloudify.fedcloud.eu/api/v3.1/deployments/jupyter-portal?
_include=id"
```

Using Python client (including client initialization):

```
#Client initialization

from cloudify_rest_client import CloudifyClient

client = CloudifyClient(
   host='cloudify.fedcloud.eu',
   protocol='https',
   username='USER',
   password='PASS',
   tenant='default_tenant')

#Creating deployment

client.deployments.create(blueprint_id='jupyter-blueprint', deployment_id='jupyter-portal')
```

Note that creating deployment means only creation of a handle of the deployment (and some preparation), not deploying/installing services. Installing services is done in the next step.

4. Deploying services by executing "install" workflow: input: deployment ID, workflow ID="install", output: execution ID (and optional service URL).
https://docs.cloudify.co/api/v3.1/?shell#start-execution

Example of installing Jupyter portal via Cloudify server on IISAS-FedCloud Openstack site at IISAS via Cloudify REST API:

```
curl -X POST --header "Tenant:default_tenant" --header "Content-Type: application/json" -u "USER:PASS"  -d
'{"deployment_id": "jupyter-portal", "workflow_id":"install"}' "https://cloudify.fedcloud.eu/api/v3.1
/executions?_include=id"
```

Using Python client:

```
client.executions.start(deployment_id='jupyter-portal', workflow_id='install')
```

5. Using the deployed services: there are two possible approaches:

a. **Using services via Cloudify**: This approach is suitable for services without their own API/GUI, e.g. generic computing servers. Users can perform predefined operations on the servers using workflow "execute_operation". The workflow is started via Cloudify REST API, with parameters: deployment ID, workflow ID="execute_operation", name of operation to execute, and the parameters for the operation
https://docs.cloudify.co/4.6/working_with/workflows/built-in-workflows/

Example of executing a command on computing server deployed on IISAS-FedCloud via Cloudify REST API:

```
curl -X POST --header "Tenant:default_tenant" --header "Content-Type: application/json" -u "USER:PASS"  -d
'{"deployment_id": "compute-server", "workflow_id":"execute_operation", "parameters":{"operation":"exec",
"node_ids":"compute_server", "operation_kwargs":{"cmd":"echo", "arg1":"Hello world"}}}' "https://cloudify.
fedcloud.eu/api/v3.1/executions?_include=id"
```

Example of executing UC4 job on computing server deployed on IISAS-FedCloud via Cloudify REST API:

```
curl -X POST --header "Tenant:default_tenant" --header "Content-Type: application/json" -u "USER:PASS" -d
'{"deployment_id": "computing-server", "workflow_id":"execute_operation", "parameters":{"operation":"exec",
"node_ids":"compute_server", "operation_kwargs":{"cmd":"./uc4-job.sh", "arg1":"hdfs://147.213.75.180:8020/user
/lufthansa/", "arg2":"hdfs://147.213.75.180:8020/user/lufthansa/"}}}' "https://cloudify.fedcloud.eu/api/v3.1
/executions?_include=id"
```

where "`uc4-job.sh`" is the UC4-specific script for executing UC4 jobs in cloud and "`arg1`" and "`arg2`" are the input and output data of the job

b. Using service directly without Cloudify, using the service URL from the deployment: This approach is suitable for services with their own API/GUI (DISPEL, LOBCDER, Jupyter portal, …). Users get their service URLs from deployments and access the services directly via the URLs without the help of Cloudify. As the services have already their own API/GUI, using services via Cloudify would add unnecessary overheads.

Example of Jupyter portal deployed on IISAS-FedCloud, after installation, users receive URL http://147.213.76.100:8080, that they can put to browser and use Jupyter interactively

6. Undeploying services by executing "uninstall" workflow: input: deployment ID, workflow ID="uninstall", output: execution ID (and optional service URL).
https://docs.cloudify.co/api/v3.1/?shell#start-execution

Example of uninstalling Jupyter portal via Cloudify server on IISAS-FedCloud Openstack site at IISAS via Cloudify REST API:

```
curl -X POST --header "Tenant:default_tenant" --header "Content-Type: application/json" -u "USER:PASS"  -d
'{"deployment_id": "jupyter-portal", "workflow_id":"uninstall"}' "https://cloudify.fedcloud.eu/api/v3.1
/executions?_include=id"
```

Using Python client

```
client.executions.start(deployment_id='jupyter-portal', workflow_id='uninstall')
```

3. Cloudify API and Python client library

The REST API of Cloudify is described at https://docs.cloudify.co/api/v3.1/?shell#cloudify-rest-api-v3-1

The corresponding Python library is available at https://github.com/cloudify-cosmo/cloudify-rest-client

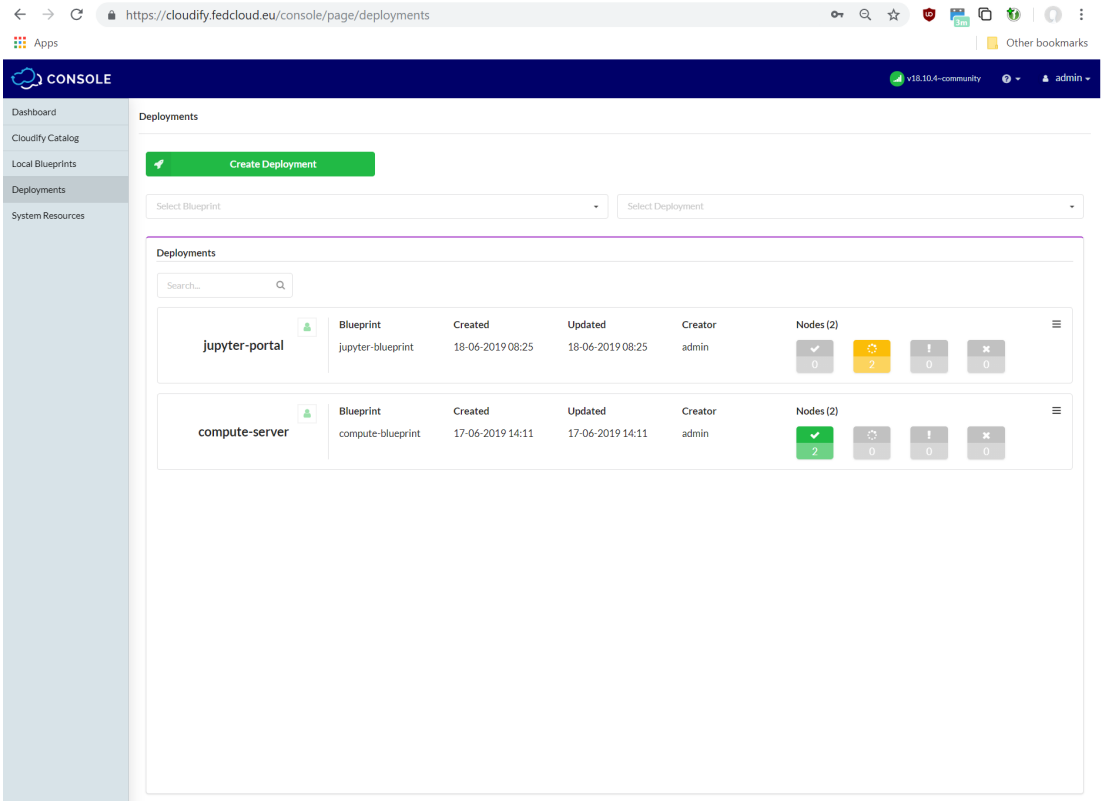
4. Cloudify deployment at IISAS

The Cloudify manager is deployed at https://cloudify.fedcloud.eu/.

Users can access Cloudify via REST API, Python client or Cloudify GUI portal. All methods of accessing Cloudify are equivalent and will provide the same results.

The blueprints for computing server, Jupyter portal and DISPEL portal have been already created and uploaded to the Cloudify server.

Credentials for accessing the server have been sent to corresponding partners for integration.



5. How to create blueprints for services

Blueprints contain TOSCA templates + corresponding scripts/data of services. The TOSCA templates can be created by Cloudify experts, however, **scripts /configurations files of services must be provided by service developers/owners**.

Taking the example of the blueprint of Jupyter portal that is uploaded in Cloudify server at IISAS. Without additional configuration specific to the service, the Jupyter portal will be unusable because it is open to all in the world for reading/writing. Therefore, beside installing/starting Jupyter portal, the blueprint also does the following things:

- Setting token/password for logging into the Jupyter portal. The token/password is specified in the input parameters and we need to understand how to configure the password for Jupyter portal and add the corresponding setting to the initialization script.
- Importing user's notebooks into Jupyter portal. The notebook is specified in the input parameters (as URL to github), and the initialization script will download the user-specified notebook, import the notebook to Jupyter portal.

Now after deployment, users receive URL of the Jupyter portal securely, with his own notebook imported and can start his work immediately without additional configuration.

Similar to the Jupyter portal, for other services (LOBCDER, DISPEL), the owners/developers of the services must provide scripts/configs for service specific configuration, so Cloudify experts can compose them into blueprints.