Virtualized Web Portals in EGI Federated Cloud

Aleš Křenek, Radim Peša, Tomáš Raček, Vlastimil Holer, Daniel Kouřil, Ĺubomír Ontkoc

MUSTweek, Brno, March 5–10

Why to virtualize web portals

- Web portal advantages
  - the user is scientist, not IT enthusiast
  - shield him/her from complexity of application and infrastructure
  - easy use, reproducible results
- Drawbacks
  - application and infra are complex, the portal is twice more
  - hand-crafted, “don’t touch and run for ever”
- Go to cloud
  - reproducible, automated deployment
  - for user: more flexible and scalable setup
  - for portal manager: more initial work but it pays

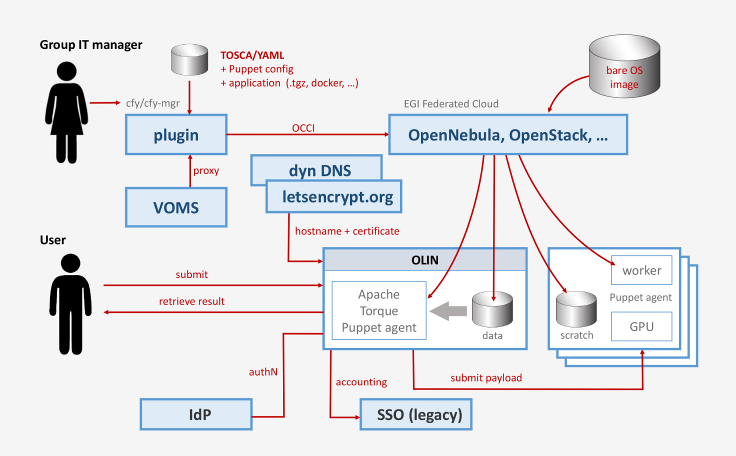
Available software solutions

- Many cloud orchestration and configuration management tools exist
  - brief overview in West-life D4.1
  - thorough survey in INDIGO-Datacloud deliverables
- Pragmatic choice for initial West-life solutions
  - **Cloudify** – cloud orchestration (before 1st INDIGO release was available)
  - **Puppet** – configuration management (long term experience with us)

Typical portal architecture

- Web front-end
- Spool storage – one folder per job
  - may have complex internal structure, long or short lived
- Machinery to handle computation
  - triggered by changes in spool directory
  - either "local" lightweight calculation or remote jobs
- interface to AAI (user AuthN/Z, accouting)
- interface to batch system or grid

Final picture



Deployment bottom-up

- cloud nodes providers – EGI FedCloud sites
- cloud management systems – OpenStack, OpenNebula, . . . (mostly hidden)
- access interface – OCCI, standard, hides management systems
- orchestration (coordinated deployent) – Cloudify (local, touches of CFM), Indigo solutions

Deployment top-down

- blueprint and node types
  - node can be VM, installed software, specific configuration action, . . .
  - relationships among them (inclusion, dependencies, . . .)
  - lifecycle phases (create, configure, start, stop, . . .)
- inputs – specific parameters for one deployment
  - to keep the same blueprint
- scripts to implement non-default lifecycle phases
- resources – any data used in at any stage
  - ssh keys, configuration files, tarballs to expand, . . .
- plugins
  - highly modular architecture, anything can be (re)implemented by plugin
  - **fabric** – execute remote commands
  - **occi** – create VMs
- software install and configuration
  - Puppet – the real way, used as blackbox today
  - hand-made scripts – manageable in tutorial

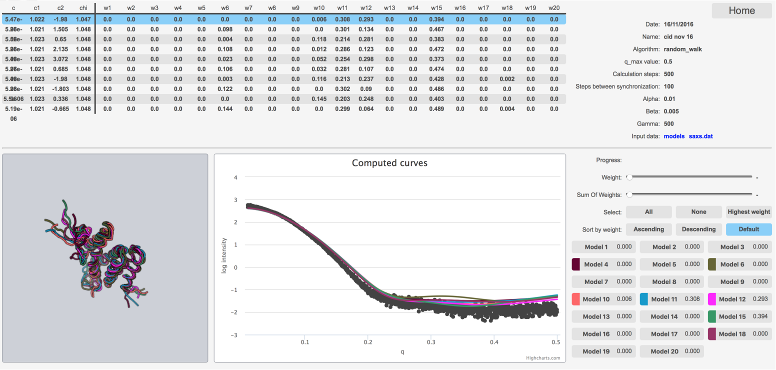
Tutorial overview

- Understand the homework
  - obtain X.509 certificate and register it with VO
  - setup client environment – software, CA certificates, VO servers, . . . (docker container)
  - check that occi works (interact with FedCloud site)
  - do the magic deployment out of blackbox
  - **let's understand it**
- Deploy web application
  - start with non-claudified (but cleaned up) application code
  - **extend Tosca description**
  - **provide specific configuration scripts**

Tutorial overview

- Add worker node
  - start with working web front end
  - pick another example – Torque server + worker node
  - **merge two Tosca specifications**
  - **configure multi-node interaction**
- Real-world user authentication
  - start with working application with fake user
  - **set up service provider and connect with IdP proxy**

The application – SAXS ensemble fit



c	c1	c2	chi	w1	w2	w3	w4	w5	w6	w7	w8	w9	w10	w11	w12	w13	w14	w15	w16	w17	w18	w19	w20
5.47e-	1.022	-1.98	1.047	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.006	0.308	0.293	0.0	0.0	0.394	0.0	0.0	0.0	0.0	0.0
5.26e-	1.021	1.505	1.048	0.0	0.0	0.0	0.0	0.0	0.098	0.0	0.0	0.0	0.0	0.301	0.134	0.0	0.0	0.467	0.0	0.0	0.0	0.0	0.0
5.02e-	1.023	0.65	1.048	0.0	0.0	0.0	0.0	0.0	0.004	0.0	0.0	0.0	0.118	0.214	0.281	0.0	0.0	0.383	0.0	0.0	0.0	0.0	0.0
5.26e-	1.021	2.135	1.048	0.0	0.0	0.0	0.0	0.0	0.108	0.0	0.0	0.0	0.012	0.286	0.123	0.0	0.0	0.472	0.0	0.0	0.0	0.0	0.0
5.08e-	1.023	3.072	1.048	0.0	0.0	0.0	0.0	0.0	0.023	0.0	0.0	0.0	0.052	0.254	0.298	0.0	0.0	0.373	0.0	0.0	0.0	0.0	0.0
5.27e-	1.021	0.685	1.048	0.0	0.0	0.0	0.0	0.0	0.106	0.0	0.0	0.0	0.032	0.281	0.107	0.0	0.0	0.474	0.0	0.0	0.0	0.0	0.0
5.08e-	1.023	-1.98	1.048	0.0	0.0	0.0	0.0	0.0	0.003	0.0	0.0	0.0	0.116	0.213	0.237	0.0	0.0	0.428	0.0	0.0	0.002	0.0	0.0
5.26e-	1.021	-1.803	1.048	0.0	0.0	0.0	0.0	0.0	0.122	0.0	0.0	0.0	0.0	0.302	0.09	0.0	0.0	0.486	0.0	0.0	0.0	0.0	0.0
5.5606	1.023	0.336	1.048	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.145	0.203	0.248	0.0	0.0	0.403	0.0	0.0	0.0	0.0	0.0
5.19e- 06	1.021	-0.665	1.048	0.0	0.0	0.0	0.0	0.0	0.144	0.0	0.0	0.0	0.0	0.299	0.064	0.0	0.0	0.489	0.0	0.0	0.004	0.0	0.0

Date: 16/11/2016
Name: cid nov 16
Algorithm: random_walk
q_max value: 0.5
Calculation steps: 500
Steps between synchronization: 100
Alpha: 0.01
Beta: 0.005
Gamma: 500
Input data: models saxs.dat

Progress:
Weight: -
Sum Of Weights: -
Select: All None Highest weight
Sort by weight: Ascending Descending Default

Model 1	0.000	Model 2	0.000	Model 3	0.000
Model 4	0.000	Model 5	0.000	Model 6	0.000
Model 7	0.000	Model 8	0.000	Model 9	0.000
Model 10	0.006	Model 11	0.308	Model 12	0.293
Model 13	0.000	Model 14	0.000	Model 15	0.394
Model 16	0.000	Model 17	0.000	Model 18	0.000
Model 19	0.000	Model 20	0.000		

Bricks to be used

- Apache server
  - single node deployment
  - set up a VM using bare OS image (CentOS 7) using OCCI
  - use Puppet to configure Apache web server with "Hello, world!" CGI script
  - we will use it "as is", not touching internals (deployment scripts, Puppet recipes, . . .)
- Torque server + worker node
  - two node deployment
  - standalone, independent on the Apache one
  - complex Puppet configuration again

Don't panic!

- It is rather complex work, we know
- Many things can go wrong
- We will do the work step by step
- Use local git commits to preserve work
- Emergency checkpoints
  - working implementations of the major steps
  - you can pick them if you get really lost

Understand the homework

- In your Docker container (radimpesa/mustweek2017)
    - do a fresh clone of git@github.com:ICS-MU/westlife-mustweek2017.git
    - look into apache/ folder
- These slides in talks/ folder
- M4 preprocessing to distinguish local vs. CFM deployment
    - ignore today, just don't edit the generated .yaml files
- browse the .yaml files and ask about their meaning
    - blueprint and inputs in the main
    - types/ folder
- briefly look into the deployment script
    - scripts/puppet/runner.sh
    - prepares and invokes Puppet
    - this is the real stuff, no need to understand details now

Understand the homework

- Initialize Cloudify:

  ```
  # source $HOME/cfy/bin/activate
  ```

- Put something unique into:

  ```
  resources/puppet/site/helloworld/files/index.py
  ```

- Deploy:

  ```
  # make clean && make cfy-deploy
  ```

  - check the result, see:

    ```
    # cfy local outputs
    ```

  - ssh to the deployed node:

    ```
    # ssh -i resources/ssh/id_rsa cfy@the_endpoint_IP
    ```

  - point you web browser to:

    ```
    http://the_endpoint_IP/cgi-bin/index.py
    ```

- Cleanup:

  ```
  # make cfy-undeploy
  ```

Deploy web application

- To speed up, start with the `apache/` example
  - copy `Makefile`, blueprint and inputs, `types/`, and `{scripts,resources}/puppet`
- add “software” node to the blueprint
  - contained in `apacheNode` (see `relationships` section)
  - started after `apache` node (`depens_on` relationship)
  - use fabric plugin to start scripts
- Installation, configuration, and start scripts
  - “poor-man” quick solution (professional would use puppet . . .)
  - put them to `scripts/saxs-portal/`
  - runs unpriviledged – use `sudo`
  - adapt (and break up) simple installation script and tarballs from `saxs/`
  - use `ctx` “shell API” to suck in cloudify resources (tarballs etc.)

    ```
    ctx download-resource resources/your/path/to/file
    @{"target_path": "/tmp/destfile"}'
    ```

Add worker node

- Pick the other example in torque/
  - appropriate pieces of blueprint and inputs
  - puppet resources (manifests/ and site/*) – just copy, no need to touch them
  - merge into results of previous step
- Deploy application sofware to the worker node
  - get inspiration from the web application deployment
- generate SSH keys for the saxs user
  - add key generation to Makefile
  - access it via ctx API in the installation script
- Add saxsWorker node
  - similar to saxsPortal
  - use saxs/worker_node_setup.sh to start with
  - copy in and install ensamle-fit binary
- Enable job management (/usr/local/saxs/saxsd.sh)

Add worker node

- Install IMP library – quick, dirty way:
  `wget https://integrativemodeling.org/2.6.2/download/IMP-2.6.2-1.e`
  into the `create.sh` script
- Clean way
  - extend `example.nodes.WebServer` to a new node type with `imp_url`
  - provide the value in inputs file
  - declare and use with `get_input` in blueprint
  - use `ctx node properties imp_url` in `create.sh` to retrieve the value
- **It should work end-to-end now**
  - test with sample data from `saxs/`