Infrastructure as a code, cloud automation

Daniele Spiga

HPC4L - Training. Beirut, Lebanon

August 21th22th October 2020

Outline

Introduction: goal of the session

Definition of infrastructure

Cloud Automation (brief intro)

Main questions we should try to answer

In this session:

- What is the infrastructure, and
- Why I need a infrastructure on top of the Cloud

Ok, but then

- How I can generate the infrastructure
- Why the cloud helps in this context

Introduction: where is my infrastructure ?

We have seen that, through a microservice architecture, we are able to write applications that are (or at least should be) scalable, reliable and maintainable.

However, when it comes to deploying these applications in the Cloud, we naturally need to find and configure the resources that are needed by the application.

For example, we need to provision the VMs where we can run our containers / microservices, exactly like we did when we create VM1 and VM2 on AWS.

In other words, we need to explicitly create our infrastructure.

The infrastructure in this context, from 10 km ...

My definition of infrastructure, in the context of this training, is a “bunch” of **computing resources** made of:

- Storage (volumes…)
- Compute (Virtual Machines...)
- Network

} Resources

Plus a set of **services** such as a simple web service as well as a more complex environment (see later)

- To be configured and deployed on top of the “bunch of storage/compute/network”

} Services

What can be a complex setup

Services

- Batch system
- Big Data Platform
- A ML training facility
- ….
- **A WLCG Site : Example A CMS site integrated**
- **A virtual CMS Site distributed over several clouds**
  - **Hybrid clouds**

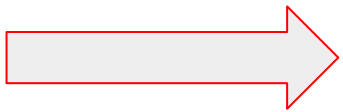
About the “Resources”

Based on our definition of “infrastructure”, getting resources imply to interact with IaaS layer

This can be (over) simplified saying that in the end what we need is some VMs plus some extra configuration

- Such as Storage, networks, ports etc..

Ok that’s all easy. What if I need to do such operation for hundreds of instances, and if this is something I want to do repeatedly?



Automation

Automation and abstraction

Creating VMs is a rather easy operation… ok and if I need hundreds of them and possibly with several software and services configuration?
Ideally one would like

- **To delegate such repetitive (and error prone) operations to a service**
- **To avoid learning Cloud APIs for any IaaS to exploit**
  - As in the case of **Hybrid Cloud** model

Also: in order to make “easy” the **exploitation** of the **underlying hardware**, even specialized (GPU, SSD etc etc…) **fabric level abstraction is a key**

- Remember what matter for users

Our vision: Declarative approach

To focus on a **declarative approach**, driven by a templating engine to specify high-level requirements. Instead of manually setting up every server, you define a configuration script with all the required settings and customizations

Key points:

- Allow users to concentrate on the "**What**" they need to deploy rather than on the "**How**" they should deploy
- Let the underlying system to **abstract providers and automatically instantiate** and setup the computing system(s)

Also, "to do a thing once and use many times"

Infrastructure as a code

With the idea of Infrastructure as Code (IaC), instead of manually creating the infrastructure we need for our applications (e.g. virtual machines, disk volumes, installations, configurations), **we define what we want through machine-readable definition files.**

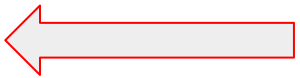
- IaC is based on the realization that "**Complexity kills Productivity**": it therefore aims to simplify how you can realize complex infrastructures and set-ups.

There are many tools that allow us to combine automation with virtualization. With IaC, a**ll the specifications for the infrastructure we are generating should be explicitly written into configuration files.**

About Tools

Some of the most popular tools for IaC are

- Puppet (https://puppet.com)
- **Ansible** (https://www.ansible.com)
- Terraform (https://www.terraform.io)
- Chef (https://www.chef.io/chef/)
- Docker itself provides some form of IaC.



Today we will see this one

Irrespective of your preferred choice, it is important to highlight that it is fundamental that whatever you do with your code and data should be reproducible and manageable.

Why Ansible

Ansible is

- Simple
- Easy to write, read, maintain and evolve -without writing scripts or custom code
- Fast to learn and set up

Support use cases such as

- Remote execution
- Configuration management
- Deployment and Orchestration

Simple connection model:

- No master-slave relationships
- Everything can configure everything, including self-config
- No custom agent to set up
- Simplest mode is push from control node to controlled nodes

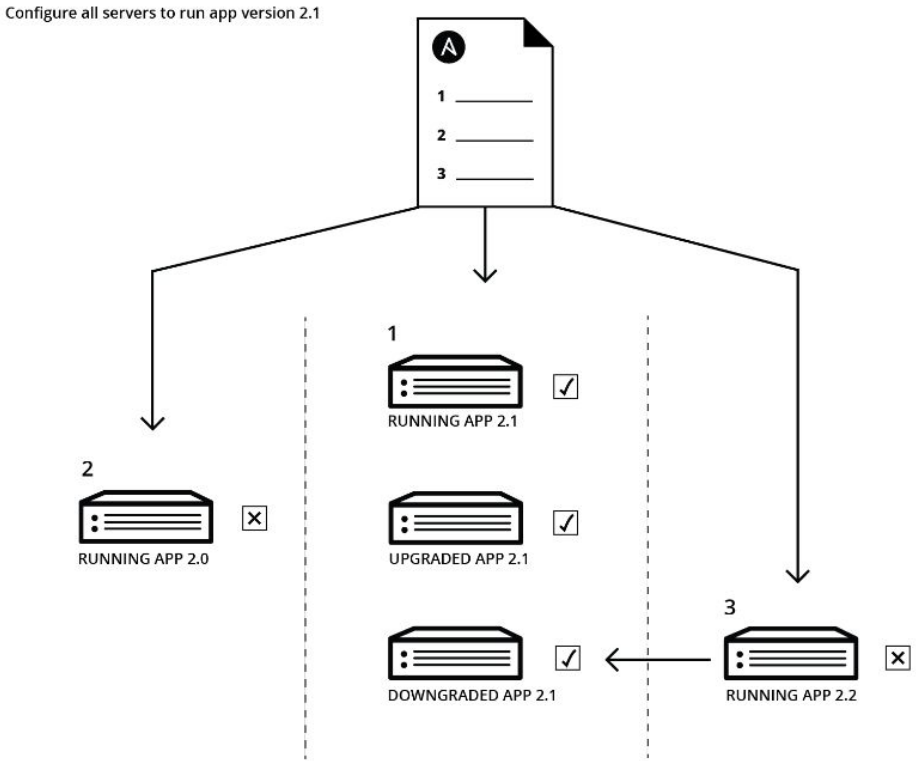
Basic concepts

Control node

Any machine with Ansible installed. Any computer that has a Python installation as a control node - laptops, shared desktops, and servers can all run Ansible. You can have multiple control nodes.

Managed nodes

The network devices (and/or servers) you manage with Ansible. Managed nodes are also sometimes called “**hosts**”. Ansible is not installed on managed nodes.



Basic Concepts (cont)

Inventory

- Fill description of hosts you control. Can nest groups and move hots in and out of groups

Playbook

- How Ansible orchestrates, configures and deploy systems. Ordered lists of tasks. Can also include variables. (written in YAML)

Task

- The units of action in Ansible. You can execute a single task once with an ad-hoc command.

Modules

- The units of code that Ansible ship out to remote machine and executes
- Each module has a particular and specific use
- can invoke a single module with a task, or invoke several different modules in a playbook.
- Once modules are executed on remote machines, they are removed (no long running daemons)

Ansible Roles

Roles are good for

- Managing the complexity: decompose complex jobs into smaller pieces
- Organizing multiple, related tasks and encapsulating data
- Compose reusable ansible content

Roles provide a framework for fully independent, or interdependent, collections of variables, tasks, files, templates, and modules.

- The task file is the main meat of a role. If roles//tasks/main.yaml exists, all the tasks there in and any other files it includes will be embedded and executed. This allows you to split a large number of tasks into separate files, and use other features of task includes

Roles vs Playbooks

Each role is typically limited to a particular theme or desired end result, with all the necessary steps to reach that result either within the role itself or in other roles listed as dependencies.

Roles themselves are not playbooks. There is no way to directly execute a role.

Roles have no setting for which host the role will apply to.

Top-level playbooks are the glue that binds the hosts from your inventory to roles that should be applied to those hosts.

Example: Install and configure Apache

- install the apache2 packages on localhost

```
---
- hosts: localhost
  connection: local
  tasks:
  - name: Apache | Make sure the Apache packages are installed
    apt: name=apache2 update_cache=yes
    when: ansible_os_family == "Debian"

  - name: Apache | Make sure the Apache packages are installed
    yum: name=httpd
    when: ansible_os_family == "RedHat"
```

Execute: ansible-playbook <PlayBook_Name>.yml

- start services after configuration customization

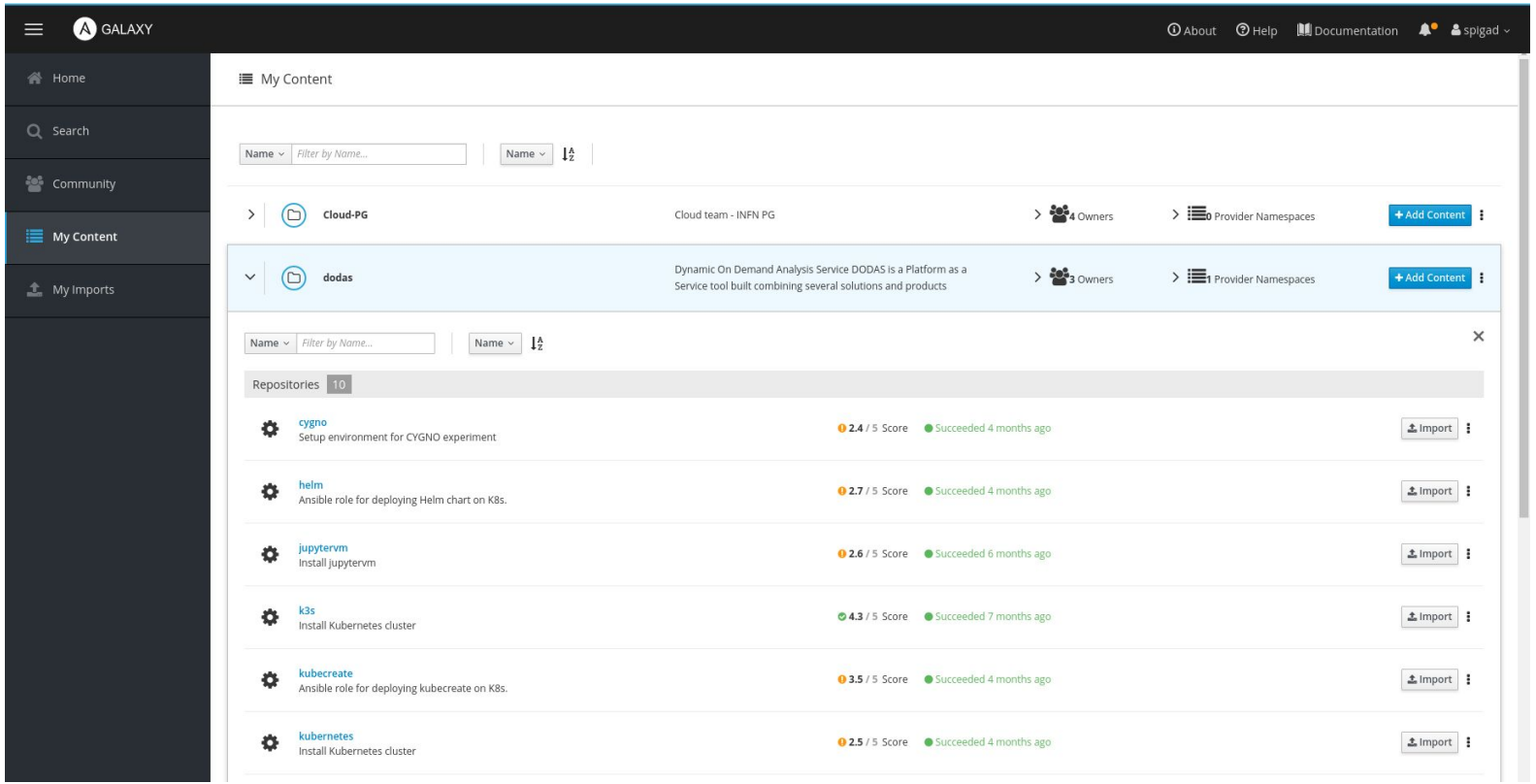
```
---
- hosts: localhost
  connection: local
  tasks:
  - name: Configure apache 1/2
    get_url:
      force: true
      url: https://raw.githubusercontent.com/DODAS-TS/SOSC-2018/master/templates/ha
      dest: /etc/apache2/ports.conf

  - name: Configure apache 2/2
    get_url:
      force: true
      url: https://raw.githubusercontent.com/DODAS-TS/SOSC-2018/master/templates/ha
      dest: /etc/apache2/sites-enabled/000-default.conf

  - name: Start Apache service
    service: name=apache2 state=restarted
    when: ansible_os_family == "Debian"

  - name: Start Apache service
    service: name=httpd state=restarted
    when: ansible_os_family == "RedHat"
```

Ansible Galaxy : Reusing roles



Ok but…

This is handy an useful, but sometimes applications need to have a higher-level description because they have several components.

- All this might not be enough to implement the Declarative model we described

There are several templating mechanisms that can be used to describe and provision resources needed by an application in a Cloud infrastructure.

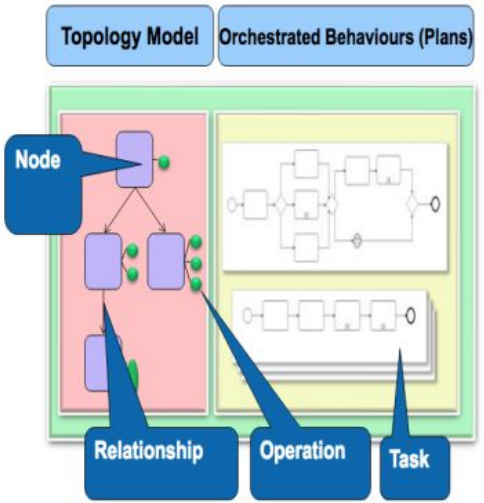
In some sense we need a solution that allow to cover any requirements the applications might have and automatize the deployments in the Cloud.

Example: The AWS CloudFormation

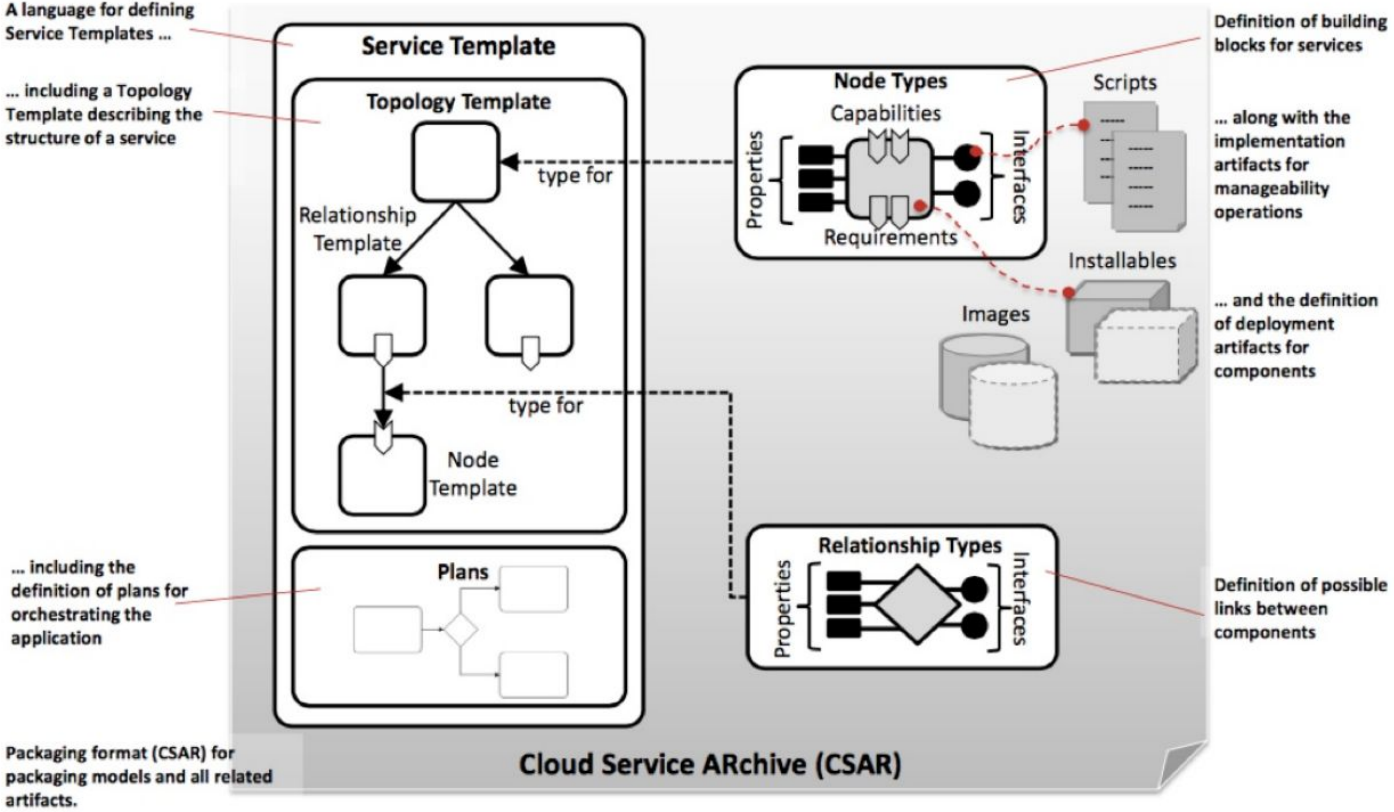
The Amazon way of defining a complete topology for an application is through the CloudFormation language.

TOSCA

- AWS CloudFormation is Amazon -specific. As such, it can only be used with AWS.
- TOSCA (Topology and Orchestration Specification for Cloud Applications) is on the other hand a public standard:
    - **It is an OASIS (https://www.oasis -open.org/) standard language to describe a topology of cloud -based web services, their components, relationships, and the processes that manage them**
- It standardizes the language to describe:
    - The structure of an IT Service (its topology model) .
    - How to orchestrate operational behavior (plans such as build, deploy, patch, shutdown, etc.) .
    - A declarative model that spans applications, virtual and physical infrastructures.



OASIS Topology and Orchestration Specification for Cloud Applications



Key modelling concepts

Topology
TOSCA is used first and foremost to describe the topology of the deployment view for cloud applications and services

Composition
Abstract nodes in one TOSCA topology can be substituted with another topology

Portability
TOSCA applications are portable to different Cloud infrastructures Application

Lifecycle
TOSCA models have a consistent view of state-based lifecycle have Operations (implementations) that can be sequenced against state of any dependent resources

Tosca Topology

- Components in the topology are called Nodes
- Each Node has a Type (e.g. Host, DB, Web server).
  - The Type is abstract and hence portable
  - The Type defines Properties and Interfaces
- An Interface is a set of hooks (named Operations)
- Nodes are connected to one another using Relationships
- Both Node Types and Relationship Types can be derived

Node Template

- An instance of a type
- Has specific properties
- Has artifacts:
  - What to install
  - How to install
- Has requirements and capabilities (or relationship)
  - The basic relationship types are:
    - **dependsOn** – abstract type and its sub types:
    - **hostedOn** – a node is contained within another
    - **connectsTo** – a node has a connection configured to another

Custom Types

TOSCA is highly versatile

- It's possible to define custom types for nodes, relationships, and capabilities —> can be used in different domains

- indigo custom types:
  - https://github.com/indigo-dc/tosca-types

```
node_types:

  tosca.nodes.DODAS.ml_infn:
    derived_from: tosca.nodes.SoftwareComponent
    properties:
        ml_user:
            type: string
            required: no
            default: cloudadm

        jupyter_port:
            type: string
            required: no
            default: 8888
        jupyter_token:
            type: string
            required: yes
            default: testme

        ml_test_url:
            type: string
            required: no
            default: ""

        cvmfs_repositories:
            type: list
            required: no
            default: [cms.cern.ch]

        role_name:
            type: string
            required: no
            default: dodas.ml_infn
```

Ok, but question now is:

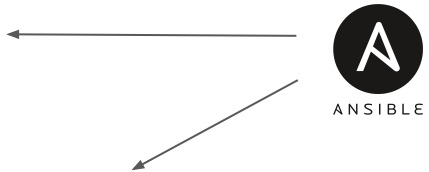
How does Ansible fits with TOSCA ?

- Should I combine the two?
- And how ?
- Is there a smart way to use both? E.g. TOSCA define topology and relationship while Ansible configure software services, dependencies (e.g. “contextualize”) ?

Custom Type

- Galaxy Ansible roles defined as artifacts are automatically installed

```
40      artifacts:
41          jupytervm_config_role:
42              file: dodas.ml_infn
43              type: tosca.artifacts.AnsibleGalaxy.role
44      interfaces:
45        Standard:
46          start:
47            implementation: https://raw.githubusercontent.com/dodas-ts/dodas-apps/master/tosca-types/dodas_artifacts/ansible.yaml
48            inputs:
49              role_name:  { get_property: [ SELF, role_name ] }
50              ml_user: { get_property: [ SELF, ml_user ] }
51              jupyter_port: { get_property: [ SELF, jupyter_port ] }
52              jupyter_token: { get_property: [ SELF, jupyter_token ] }
53              cvmfs_repositories: { get_property: [ SELF, cvmfs_repositories ] }
54
55
```



Example 1

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: Template for deploying a single server with predefined properties.

topology_template:
  inputs:
    cpus:
      type: integer
      description: Number of CPUs for the server.
      constraints:
        - valid_values: [ 1, 2, 4, 8 ]

  node_templates:
    my_server:
      type: tosca.nodes.Compute
      capabilities:
        # Host container properties
        host:
          properties:
            # Compute properties
            num_cpus: { get_input: cpus }
            mem_size: 2048  MB
            disk_size: 10 GB

  outputs:
    server_ip:
      description: The private IP address of the provisioned server.
      value: { get_attribute: [ my_server, private_address ] }
```

Example 2

```
tosca_definitions_version: tosca_simple_yaml_1_0
description: Template for deploying a single server with MySQL software on top.

topology_template:
 inputs:
  # omitted here for brevity

 node_templates:
  mysql:
   type: tosca.nodes.DBMS.MySQL
   properties:
    root_password: { get_input: my_mysql_rootpw }
    port: { get_input: my_mysql_port }
   requirements:
    - host: db_server

  db_server:
   type: tosca.nodes.Compute
   capabilities:
    # omitted here for brevity
```

What can we do in practice with these stuff

TOSCA and other template-driven orchestration mechanisms allow us to realize service composition, i.e. to combine different services to implement complex topologies.

- **An example of service composition developed within INFN is DODAS (Dynamic On-Demand Analysis Service), where we facilitate the deployment of relatively complex set-ups on any cloud provider with almost zero effort.**