Infrastructure as code

Introduction by example

Outline

- Introduction
- What we want
- Automate it all
- Target service
- Tosca template
- Make it real
- See it in action
- Summary

Introduction

Cloud Automation is a set of processes and technologies that allow to automate several operations.

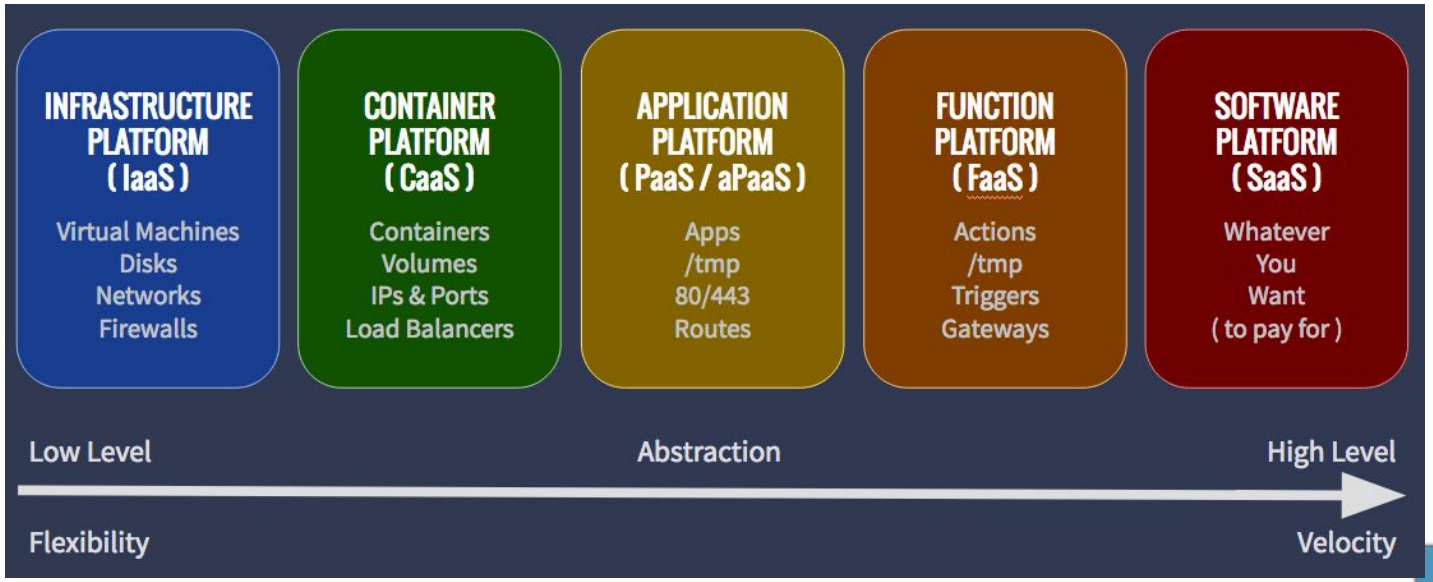
Doing things **by hand is rarely a good idea** when complexity increases, and we have already seen several relatively complex technologies.

We have seen that, through a **microservice architecture** and some related processes and tools such as **DevOps**, 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.

Introduction

When working with **cloud resources**, depending on the user needs, different layers of underlying abstraction can be needed and, depending on how many layers and their composition, one can define different categories.



What we want

- **Not care** about the **infrastructure**
- **Not care** about the **volatility** of the **resources**
- A platform with the **frameworks we want to use**
- **A custom environment** for the end users
- A **scalable** solution
- A **replicable** solution

We can **automate** these steps **at different levels**, such as:

- ***Infrastructure***: you need and infrastructure manager
- ***Cluster***: you have to describe how your resources are organized
- ***Software***: you need to specify your customizations

Missing link

- **TOSCA**

TOSCA means Topology and Orchestration Specification for Cloud Applications. It is an **open standard to describe your cloud composition** and applications in a way independent from the infrastructure and the resource manager.

- **Ansible**

It is an **open source software** that **automates software deployment**. With Ansible you can create recipe that customize an environment with your software and your specifications.

Automatize it all

The best solution would be something that will do **automatically** all the wanted stuff for us but, we still need something that interacts with the resources.

This part is taken over by the **Infrastructure Manager**, that controls the resources for us and that we can instruct with **TOSCA** and **Ansible**.

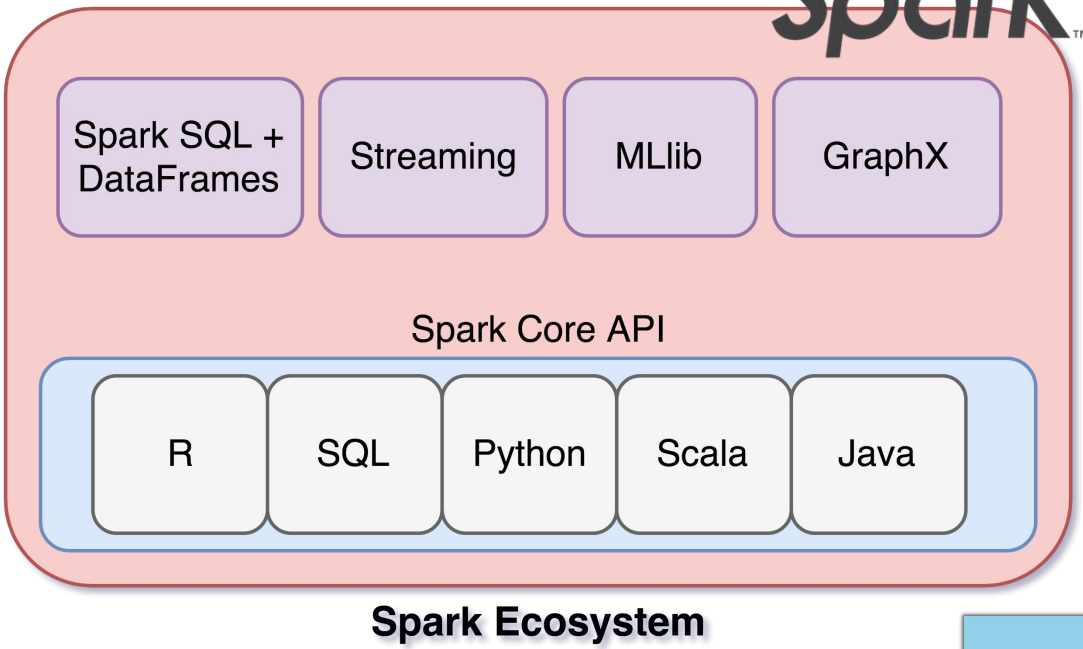
Example context

- **Cluster**: A group of coupled computers that work together. We need to increase our computational power. It's not trivial to configure and manage.
- **Cloud**: An high-level view of services provided by a cluster of computers. It gives a more user-friendly approach to interact with calculus computing resources.
- **Big Data**: The nightmare of data analysts and treasure for Machine Learning people. They need an appropriate environment to be managed and it's also a problem the storing of the data themselves.

Target service

A common framework used in this field is Apache Spark.

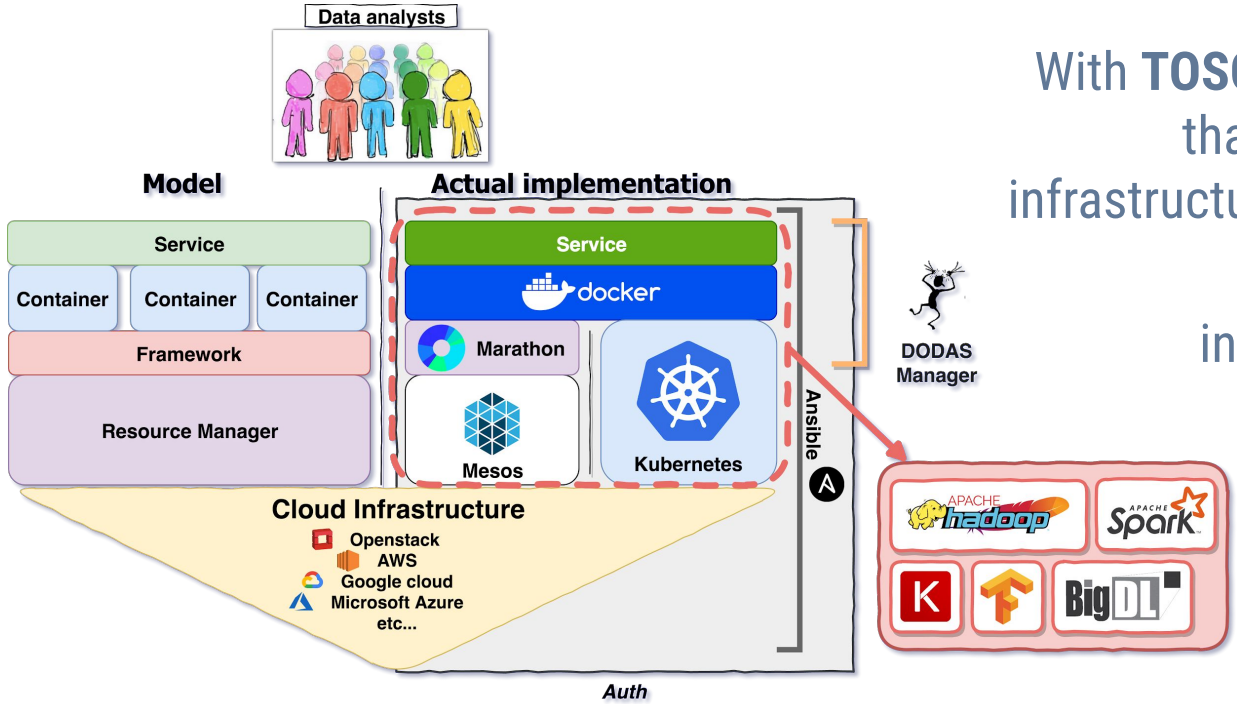
Apache Spark is an open-source powerful distributed querying and processing engine. It's quite easy to write a task for this engine (using its API) and it allows you to process a large amount of data.



Spark Ecosystem

Tosca template

We have to describe the service with the declarative language named **TOSCA**. The idea is to reproduce the **schema** of the target configuration.



With **TOSCA** we can define **nodes** that represents part of our infrastructure. Plus we can specify configurations,

inputs and variable of our environment.

Make it real

The basic nodes we need are:

- Master
- Slave
- Load balancer

```
mesos_master:
    type: tosca.nodes.indigo.MesosMaster
    [...]

mesos_slave:
    type: tosca.nodes.indigo.MesosSlave
    [...]

mesos_load_balancer:
    type: tosca.nodes.indigo.MesosLoadBalancer
    [...]
```

```
mesos_master_server:
    type: tosca.nodes.indigo.Compute
    [...]

mesos_slave_server:
    type: tosca.nodes.indigo.Compute
    [...]

mesos_lb_server:
    type: tosca.nodes.indigo.Compute
    [...]
```

Make it real

Also, a role for the Spark environment is necessary:

```
spark_application:
    type: tosca.nodes.indigo.SparkMesos
    [...]
```

This last part involves also the customization for the end users, that can change the service due to the frameworks they want to use.

Make it real

To customize the service we prepared a specific ansible recipe that uses also Docker containers.

The container allow us to encapsulate all the requirements we want.

```
[...]

- name: "[Spark-Mesos] deploy app container on Marathon"
  run_once: true
  uri:
     url: "://marathon.service.consul:/v2/apps"
     user: ""
     password: ""
     validate_certs: "no"
     method: POST
     HEADER_Content-Type: "application/json"
     body: ".json') }}"
     body_format: json
     status_code: 201
  register: post_result
  when: get_result.status == 404
  tags:
    - spark

[...]
```

Complete TOSCA template

Link to TOSCA file:

https://raw.githubusercontent.com/DODAS-TS/SOSC-2018/master/templates/hands-on-2/spark-cluster.yaml

Complete TOSCA template

```
tosca_definitions_version: tosca_simple_yaml_1_0


imports:
  - indigo_custom_types:
https://raw.githubusercontent.com/DODAS-TS/SOSC-2018/master/templates/common/types
.yml


description: >
     TOSCA template for deploying Spark + Mesos cluster. The following flavors can
be selected setting the input spark_mesos_flavor
```

Complete TOSCA template

```
topology_template:

  inputs:

     marathon_username:
       type: string
       default: "admin"

     marathon_password:
       type: string
       description: Admin password for accessing Marathon HTTP service
       default: 'passwd'
       required: yes
[...]
```

Complete TOSCA template

```
node_templates:

     spark_application:
       type: tosca.nodes.indigo.SparkMesos
       properties:
         marathon_password: { get_input: marathon_password }
         zookeeper_peers: { get_attribute: [ mesos_master_server, private_address ]
}
         spark_hdfs_uri: { get_input: hdfs_uri }
         spark_cores_max: { get_input: spark_cores_max }
         [...]
       requirements:
       - host: mesos_master
       - dependency: mesos_master
[...]
```

Complete TOSCA template

```
mesos_master:
     type: tosca.nodes.indigo.MesosMaster
     properties:
       mesos_username: { get_input: mesos_username }
       mesos_password: { get_input: mesos_password }
       marathon_username: { get_input: marathon_username }
       marathon_password: { get_input: marathon_password }
       mesos_masters_list: { get_attribute: [ mesos_master_server, private_address
] }
       chronos_password: 's3cret'
     requirements:
     - host: mesos_master_server

[...]
```

Complete TOSCA template

```
mesos_master_server:
     type: tosca.nodes.indigo.Compute
     capabilities:
     endpoint:
          properties:
          dns_name: mesosserverpublic
          network_name: PUBLIC
          ports:
               mesos_port:
                 protocol: tcp
                 source: 5050
     scalable:
          properties:
               count: { get_input:
master_num }
     host:
          properties:
          num_cpus: { get_input:
master_cpus }
          mem_size: { get_input:
master_mem }
     os:
          properties:
          image: { get_input:
server_image }
```

Complete TOSCA template

```
outputs:
     spark_bastion:
     value: { concat: [ 'ssh admin@', get_attribute: [ mesos_lb_server,
public_address, 0 ], ' -p 31042' ] }
     mesos_endpoint:
     value: { concat: [ 'http://', get_attribute: [ mesos_master_server,
public_address, 0 ], ':5050' ] }
     marathon_endpoint:
     value: { concat: [ 'https://', get_attribute: [ mesos_master_server,
public_address, 0 ], ':8443' ] }
     master_creds:
     value: { get_attribute: [ mesos_master_server, endpoint, credential, 0 ] }
```

Summary

- We can have tools that automate the platform creation
- We can customize our environment to reflect the workflow requirements
- We can reuse our configuration on several infrastructures
- We can maintain and update easily our frameworks
- We can reuse resources in base of the purpose

And now?

We can use all those stuff?

Yes, let's move to the **next hands-on**...

continue…