INDIGO PAAS Tutorial

- Introductory Concepts
  - TOSCA
  - Ansible
  - Docker
  - Orchestrator APIs
  - INDIGO TOSCA custom types and templates

- Demos
- **Topology and Orchestration Specification for Cloud Applications**
- Standardizes the language to describe
  - The structure of an ITService (its **topology** model)
  - How to orchestrate operational behavior (plans such as build, deploy, patch, shutdown, etc.)
    - Leveraging the BPMN standard
  - Declarative model that spans applications, virtual and physical infrastructure
TOSCA in a nutshell

- A language for defining Service Templates...
- Including a Topology Template describing the structure of a service...
- Including the definition of plans for orchestrating the application...

Cloud Service ARChive (CSAR)

Packaging format (CSAR) for packaging models and all related artifacts.
TOSCA Topology

■ 3 layers
  ● Infrastructure (Cloud or DC objects)
  ● Platform or Middleware (App containers)
  ● Application modules, schemas and configurations

■ Relationships between components:
  ● What’s hosted on what or installed on what
  ● What’s connected to what
Components in the topology are called Nodes

Each Node has a Type (e.g. Host, BD, 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 Type Examples

- Defines properties as YAML maps
- Might define capabilities (What it can provide to other nodes)

```yaml
tosca.nodes.DBMS
  derived_from: tosca.nodes.SoftwareComponent
  properties:
    dbms_root_password:
      type: string
      description: the root password for the DBMS service
    dbms_port:
      type: integer
      description: the port the DBMS service will listen to for data and requests
  capabilities:
    host:
      type: Container
      containee_types: [ tosca.nodes.Database ]
```
Node Type Examples

- Might Define Requirements (what it needs from other nodes)

```yaml
tosca.nodes.Database:
  derived_from: tosca.nodes.Root
  properties:
    db_user:
      type: string
      description: user account name for DB administration
    db_password:
      type: string
      description: the password for the DB user account
    db_port:
      type: integer
      description: the port the underlying database service will listen to data
    db_name:
      type: string
      description: the logical name of the database
  requirements:
    - host: tosca.nodes.DBMS
  capabilities:
    - database_endpoint: tosca.capabilities.DatabaseEndpoint
```
Node Template

- An instance of a type (like Object to Class)
- Has specific properties
- Has artifacts:
  - What to install
  - How to install (mapped to interface hooks)
- Has requirements and capabilities (or relationships)
```
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 ] }
```
INDIGO custom types

- Extend the TOSCA normative types

  E.g.: `tosca.nodes.indigo.GalaxyPortal`,
  `tosca.nodes.indigo.GalaxyWN`
Automated Deployment with Ansible

- Ansible is a configuration management and provisioning tool, similar to Chef, Puppet or Salt.

- Agentless
  - uses SSH by default to make the connection to the target node
  - can be run locally

- Configuration in YAML

- Very easy to learn
  - [http://docs.ansible.com/ansible/index.html](http://docs.ansible.com/ansible/index.html)
Building blocks: Playbook

- Defines sequences of tasks (Plays) to be executed on a group of hosts
  - describes policies machines under management shall enforce
  - contains variables, tasks, handlers, files, templates and roles
  - expressed in YAML
  - Idempotent

---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
    remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
  notify:
  - restart apache
  - name: ensure apache is running (and enable it at boot)
    service: name=httpd state=started enabled=yes
  handlers:
  - name: restart apache
    service: name=httpd state=restarted
Building blocks: Roles

- The best way to organize your playbooks.
  - structure content into related vars, tasks, files, handlers, etc.
  - file structure for automated inclusion of role specific content
  - roles can be shared and pulled from Ansible Galaxy, GitHub, etc.

---

- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  roles:
    - webserver
### Ansible Galaxy: indigo-dc roles

<table>
<thead>
<tr>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ambertools</td>
<td>Ambertools applications</td>
</tr>
<tr>
<td>calico</td>
<td>Configure and run calico</td>
</tr>
<tr>
<td>chronos</td>
<td>Install Chronos</td>
</tr>
<tr>
<td>clues</td>
<td>Install Clues</td>
</tr>
<tr>
<td>consul</td>
<td>Install consul with dnsmasq</td>
</tr>
<tr>
<td>disvis-powerfit</td>
<td>disvis and powerfit applications</td>
</tr>
<tr>
<td>docker</td>
<td>Install docker engine</td>
</tr>
<tr>
<td>etcd</td>
<td>Configure and run dockerized etcd</td>
</tr>
<tr>
<td>galaxy</td>
<td>Install Galaxy portal</td>
</tr>
<tr>
<td>hadoop</td>
<td>Install a Hadoop Cluster</td>
</tr>
<tr>
<td>haproxy-consul</td>
<td>Install haproxy-consul</td>
</tr>
<tr>
<td>keepalived</td>
<td>Install keepalived</td>
</tr>
<tr>
<td>marathon</td>
<td>Install Marathon</td>
</tr>
<tr>
<td>mesos</td>
<td>Install Mesos components</td>
</tr>
<tr>
<td>nfs</td>
<td>NFS server/client</td>
</tr>
<tr>
<td>slurm</td>
<td>Install SLURM cluster</td>
</tr>
<tr>
<td>zabbix-agent</td>
<td>Install Zabbix agent in active mode</td>
</tr>
<tr>
<td>zookeeper</td>
<td>Install zookeeper</td>
</tr>
</tbody>
</table>

To install a role, use the command:

```
ansible-galaxy install indigo-dc.<role>
```
Docker: Build-Ship-Run

- Docker is an open-source engine to easily create lightweight, portable, self-sufficient containers from any application.

- The same container that a developer builds and tests on a laptop can run at scale, in production, on VMs, OpenStack cluster, public clouds and more.

- Docker features:
  - versioning (git-like)
  - component re-use
  - sharing (public repository)
A Dockerfile is a script, composed of various commands (instructions) and arguments listed successively to automatically perform actions on a base image in order to create (or form) a new one.

```
FROM ubuntu
MAINTAINER Romin Irani (email@domain.com)
RUN apt-get update
RUN apt-get install -y nginx
ENTRYPOINT ["/usr/sbin/nginx","-g","daemon off;"]
EXPOSE 80
```
Build - Dockerfile

- docker build -t indigodatacloudapp/ambertools:1.0.0

- Automated build on Docker Hub
Ship and Run

- Ship a Docker Image
  - `docker push indigodatacloudapp/ambertools`

- Fetch a Docker Image
  - `docker pull indigodatacloudapp/ambertools`
    - download docker image from Docker Hub to local Docker repository

- Run a docker container
  - `docker run [...] indigodatacloudapp/ambertools`
    - creates a docker container out of the docker image
Orchestrator APIs

- Create a deployment:
  - POST request to /deployments - parameters:
    - template: string containing a TOSCA YAML-formatted template
    - parameters: the input parameters of the deployment (map of strings)

- Get deployment details
  - GET request to /deployments:
    - curl 'http://localhost:8080/deployments/<uuid>'

- Delete deployment
  - DELETE request
    - curl 'http://localhost:8080/deployments/<uuid>'

Chronos Job definition and APIs

- Submit a Job:
  - POST request to /scheduler/iso8601
    - accepts JSON body
- Add a dependent job
  - POST to /scheduler/dependency
    - accept JSON hash
- List jobs
  - GET to scheduler/jobs
    - filter on name: GET /scheduler/jobs/search?name=<jobId>
- Delete a job
  - POST to /scheduler/job/<jobId>
The Orchestrator interprets the TOSCA template and orchestrates the deployment on Chronos.
■ **Scenario I:** the job data are staged in/out using the user’s onedata spaces (providing proper access token(s))

■ **Scenario II:** the job data are fetched from public URLs and uploaded to a repository (web, swift/S3, etc.) using the credentials specified by the user
  - example template: [https://github.com/indigo-dc/tosca-types/blob/master/examples/indigo_job_output_upload_swift.yaml](https://github.com/indigo-dc/tosca-types/blob/master/examples/indigo_job_output_upload_swift.yaml)
The TOSCA template describes multiple jobs

Each job is run with a specific set of input parameters

The jobs are run in parallel on the Mesos cluster

- the scaling service ensures that new slave nodes are added to the cluster, if needed.
Demos Outline

- **Elastic Galaxy Cluster** (Recorded) Demo:
  - a Galaxy portal is automatically deployed from TOSCA and configured to use a SLURM elastic cluster
  - worker nodes are added as soon as jobs are submitted through the portal

- **High-Available Mesos Cluster** (Recorded) Demo:
  - a complete HA Mesos cluster with Chronos/Marathon framework is automatically deployed from a TOSCA template

- **Application execution** (Live) Demos:
  - Amber
  - EMSO-INGV (Rscript)
  - Lifewatch (barridocker)
Long-Running Service deployment Demo:

- deploy Long-Running Services: nginx (web service), rstudio
- auto-scaling PoC
- load-balancing