Cloud Computing – Lecture 10

Deployment models in cloud

14 April 2020

Chinmaya Dehury
Satish Srirama
Outlines

• Introduction to deployment models in cloud
• Deployment concerns and selection criteria
• Cloud deployment solutions
• Service deployment in cloud
• Service deployment standard
• Orchestration of cloud services
• Cloud service management solutions
Cloud deployment model

What do you mean by deployment model?

• Who is the targeted user?
• Each cloud model is a representative of a specified environment and is distinguished by
  • Size
  • Access
  • Storage and
  • Ownership.
Cloud deployment model
Cloud deployment models

1. Private Cloud
2. Public Cloud
3. Hybrid Cloud
Cloud deployment models comparison

<table>
<thead>
<tr>
<th>Description</th>
<th>Public Cloud</th>
<th>Private Cloud</th>
<th>Hybrid Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud environment</td>
<td>Multi-Tenancy-Shared environment.</td>
<td>Single tenancy-only for single use of an organization.</td>
<td>Both single tenancy and multi-tenancy.</td>
</tr>
<tr>
<td>Data center location</td>
<td>Anywhere</td>
<td>Inside the organization’s network.</td>
<td>Both</td>
</tr>
<tr>
<td>Resource sharing (Server hardware, network and storage)</td>
<td>shared by multiple users in the cloud.</td>
<td>No sharing of resources.</td>
<td>Very secure; integration options add an additional layer of security.</td>
</tr>
</tbody>
</table>
Cloud Computing - Lecture 10: Deployment models in cloud

<table>
<thead>
<tr>
<th>Description</th>
<th>Public Cloud</th>
<th>Private Cloud</th>
<th>Hybrid Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud storage</td>
<td>storage as a service on a pay per use basis.</td>
<td>Internal cloud storage</td>
<td>Streamlined storage in local and public cloud storage.</td>
</tr>
<tr>
<td>Scalability</td>
<td>Instant and unlimited.</td>
<td>Sacrifices scalability but provides greater control and security.</td>
<td>On demand unlimited resources.</td>
</tr>
<tr>
<td>Pricing structure</td>
<td>On usage basis.</td>
<td>Comparatively expensive.</td>
<td>High</td>
</tr>
<tr>
<td>Cloud Security</td>
<td>Depends on service provider</td>
<td>Most secure</td>
<td>Secure</td>
</tr>
<tr>
<td>Performance</td>
<td>Low to medium.</td>
<td>Very High.</td>
<td>Very High.</td>
</tr>
</tbody>
</table>
More deployment models...

• Managed Private Cloud
• Hosted Private Cloud
• Community Cloud Services
Deployment Concerns

• Security concern
• Data governance
• Audit
• Management
• Multi-tenancy
• Vendor lock-in concern
• Standard concern
Deployment selection criteria

• Workload Patterns
• Security concerns
• Regulations
• Availability
• Hosted data classification
• Type of requested services
• Available budget
• Time to operate
• Skills
Cloud Deployment solutions

• OpenStack
• CloudStack
• Cloudify
• Vmware
• IBM Bluemix Cloud
• SAP HANA Cloud platform
• Red Hat Cloud
• Cisco System
• Dell Cloud
• Platform.sh
• Citrix Cloud
• HP Helion
• Oracle Cloud
• NetApp
Cloud Deployment solutions

- OpenStack
- CloudStack
- Cloudify
- Vmware
- IBM Bluemix Cloud
- SAP HANA Cloud platform
- Red Hat Cloud
- Cisco System
- Dell Cloud
- Platform.sh
- Citrix Cloud
- HP Helion
- Oracle Cloud
- NetApp

This is more about how to setup your own cloud !!!
Cloud Deployment solutions

But how about the service deployment?
But how about the service deployment?

How would you deploy your own service, e.g. a messaging platform, your own website, etc.
TOSCA standard

• Topology and Orchestration Specification for Cloud Applications (TOSCA)
• By OASIS, a not-for-profit consortium
• Enhance the portability and operational management of cloud and other types of applications and services across their entire lifecycle.

• User will be able to provide
  • Description of interoperability of the application and infrastructure cloud services
  • Relationships between parts of the services
  • operational behavior of the services
    • Deploy, patch, shutdown
TOSCA standard (contd...)

• Facilitate higher level of Solution portability:
  • Portable deployment
  • Simplify migration
  • Dynamic, flexible scaling

• Model-driven creation of cloud services

• Standardize deployment

• Process-driven service lifecycle management

• Latest version:
  • TOSCA Simple Profile in YAML Version 1.3: https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/TOSCA-Simple-Profile-YAML-v1.3.html
TOSCA modelling

**GOAL:**
- Cross cloud, cross tools, orchestration of application on the cloud

**Basic blocks:**
- Topology
- Composition
- Requirements – Capabilities
- State (nodes, relationship)
- Lifecycle
- Policy
TOSCA modelling – Service Template

• Templates define the structure of a cloud service.
• Usually described in YAML or in XML format
• Structure of service template
  • Node Type
  • Relationship type
  • Node template
  • Relationship template
TOSCA modelling – Node Type

• Describe the type of a resource, component. E.g. type of cloud, type of software etc.
• This is similar to declaring a class in different Object Oriented Programming languages.
• Describe the lifecycle of the node type.

```python
tosca.interfaces.node.Lifecycle:
  create:
    description: Basic lifecycle create operation.
  configure:
    description: Basic lifecycle configure operation.
  start:
    description: Basic lifecycle start operation.
  stop:
    description: Basic lifecycle stop operation.
  delete:
    description: Basic lifecycle delete operation.
```
TOSCA modelling – Node Type

• Describe Properties, attributes, capabilities, requirements of a node

tosca.nodes.Root:
  derived_from: tosca.entity.Root
description: The TOSCA Node Type all other TOSCA base Node Types derive from
attributes:
tosca_id:
  type: string
capabilities:
  feature:
    type: tosca.capabilities.Node
requirements:
  - dependency:
    capability: tosca.capabilities.Node
    node: tosca.nodes.Root
    relationship: tosca.relationships.DependsOn
    occurrences: [ 0, UNBOUNDED ]

interfaces:
  Standard:
    type: tosca.interfaces.node.lifecycle.Standard
TOSCA modelling – Node template

• Define topology with
  • Nodes (*node_templates*)
  • Relationship (*relationship_templates*)

• Node template
  • To describe components in the topology structure.
  • Set of nodes (resources/components)
  • Concrete Node:
    • Have deployment and implementation artifacts
  • Abstract Node:
    • Describe only node type, and mandatory capabilities and properties
TOSCA modelling - Node template

• Node template - Example

tosca_definitions_version: tosca_simple_yaml_1_3
description: Template with a generic dependency between two nodes.
topology_template:
  inputs:
  # omitted here for brevity

node_templates:
  my_app:
    type: my.types.MyApplication
    properties:
    # omitted here for brevity
    requirements:
    - dependency: some_service

some_service:
  type: some.nodetype.SomeService
  properties:
  # omitted here for brevity
TOSCA modelling - Relationship template

- Relationship template
  - To describe connections, dependencies, deployment ordering
- Requirements and capabilities are implicit way to describe the relationship

- Some basic explicit relationship types:
  - DependsOn
  - ConnectsTo
  - AttachesTo
  - RoutesTo
  - HostedOn
TOSCA modelling - Relationship template

Relationship types example:
TOSCA modelling - Relationship template

Relationship types example:
TOSCA modelling - Relationship template

Relationship types example:
TOSCA modelling - Relationship template

Relationship Example

node_templates:

my_block_storage:
  type: BlockStorage
  properties:
    size: 10

my_web_app_tier_1:
  derived_from: Compute
  requirements:
    - local_storage:
      node: my_block_storage
      relationship: storage_attachesto_1

my_web_app_tier_2:
  derived_from: Compute
  requirements:
    - local_storage:
      node: my_block_storage
      relationship: storage_attachesto_2

relationship_templates:

storage_attachesto_1:
  type: MyAttachesTo
  properties:
    location: /my_data_location

storage_attachesto_2:
  type: MyAttachesTo
  properties:
    location: /some_other_data_location

relationship_types:

MyAttachesTo:
  derived_from: AttachesTo
  interfaces:
    some_interface_name:
      some_operation:
        implementation: default_script.sh
TOSCA modelling - Relationship template

• Relationships have interfaces as well

• **configure** : the basic interface
  • `pre_configure_source`
  • `pre_configure_target`
  • `post_configure_source`
  • `post_configure_target`
  • `add_source, add_target`
  • `remove_source, remove_target`
TOSCA Implementation

- Cloudify - [http://getcloudify.org/](http://getcloudify.org/)
- DICER - [https://github.com/DICERs/DICER](https://github.com/DICERs/DICER)
- Eclipse Winery - [https://projects.eclipse.org/projects/soa.winery](https://projects.eclipse.org/projects/soa.winery)
- Opera - [https://github.com/radon-h2020/xopera-opera](https://github.com/radon-h2020/xopera-opera)
- OpenTOSCA - [http://www.opentosca.org/](http://www.opentosca.org/)
- see full list...
TOSCA implementation

We will discuss following two

• Cloudify
• Opera
TOSCA implementation - Cloudify

- open source cloud orchestration framework.
- written in the Python programming language.
- Model applications and services and automate their entire life cycle.
- Supports configuration management tools like Chef, Puppet, Ansible for the application deployment phase.
- Each application is describe in a YAML file called blueprint.
- GUI to model the service blueprint.
TOSCA implementation - Cloudify
TOSCA implementation - Cloudify
Cloudify Manager primarily is built with **open-source components**:

- **Nginx**: high-performing Web server. For the Cloudify REST service, file server to host Cloudify-specific resources
- **Gunicorn**: Web server gateway interface HTTP server
- **Flask**: Web framework
- **PostgreSQL**: object-relational database that stores the application’s model, indexing, and logs’ and events’ storage
- **Logstash**: to pull log and event messages from RabbitMQ and index them in PostgreSQL.
Cloudify Manager primarily is built with open-source components (contd...):

- **RabbitMQ**: Queueing deployment tasks, logs and events, Queueing metrics
- **Riemann**: policy-based decision maker (an experimental feature)
- **Celery**: distributed task queue. Cloudify management worker, the deployment-specific agents and the host agents are based on Celery
- **InfluxDB**: time-series database; to store metrics submitted by the application’s hosts
TOSCA implementation - Opera

- Lightweight orchestrator
- Compliant with OASIS TOSCA
- Support TOSCA Simple Profile YAML v1.3
- Red Hat *Ansible Automation* for interface implementation
- Supported by European Union’s Horizon 2020 research and innovation program “RADON”.
- [https://github.com/radon-h2020/xopera-opera](https://github.com/radon-h2020/xopera-opera)
Ansible Automation

- IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs.
- Founded in 2013
- Bought by Red Hat in 2015.
- Similar technology: Chef, Puppet
- Heavily use SSH infrastructure to connect and control other remote servers.
Ansible

**Advantage:**

- Open-source tool
- **Agentless:** No extra agent is required on the remote machine.
- Flexible
- Quite easy as it follow YAML format.
- User/developer focus only on the task.
Ansible Architecture: **Modules**

- Over 1000 modules to automate the tasks
- Do the actual work
- Most of modules are standalone
- Users can also write their own modules

- Here ‘service’ is a module.
- Some other modules: `read_csv`, `file`, `copy`, `lineinfile`, etc
Ansible Architecture: **Inventories**

- List of nodes along with related info.:
  - Address
  - Username
  - Password
  - Connection type
  - Other configuration for ssh connection

- Default location of inventory file: `/etc/ansible/hosts`

- Inventory can be in *YAML* or in *ini* format
Ansible Architecture: Inventories

Example

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mail.example.com</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[webservers]</td>
<td>foo.example.com</td>
<td>bar.example.com</td>
</tr>
<tr>
<td>[dbservers]</td>
<td>one.example.com</td>
<td>two.example.com</td>
</tr>
<tr>
<td></td>
<td>three.example.com</td>
<td></td>
</tr>
<tr>
<td>[targets]</td>
<td>localhost</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ansible_connection=local</td>
<td></td>
</tr>
<tr>
<td></td>
<td>other1.example.com</td>
<td>ansible_connection=ssh</td>
</tr>
<tr>
<td></td>
<td>other2.example.com</td>
<td>ansible_connection=ssh</td>
</tr>
<tr>
<td></td>
<td>ansible_user=myuser</td>
<td>ansible_user=myotheruser</td>
</tr>
</tbody>
</table>
Ansible Architecture: **Playbooks**

- It’s in YAML format
  - Playbook contains Plays;
  - Plays contain tasks;
  - Tasks call module.

- Simple command to run a playbook:
  
  ansible-playbook playbook.yml
Ansible Architecture: **Playbooks**

**Example:**

---

- 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
      service:
        name: httpd
        state: started
Puppet

• An open-source configuration management technology.
• Manages the infrastructure on physical or virtual machines.
• Written in Ruby, C++, and Clojure
• Usually follows client-server or agent-master architecture.

• **Puppet Server**: on one or more servers
• **Puppet Agent**: on all the machines that we want to manage
Puppet Blocks

• Different building blocks:
  • Modules
  • Classes
  • Resources
  • Manifest
Puppet Blocks - Modules

- A collection of files and directories, such as Manifests, Class definitions.
- Each module manages a specific task in the infrastructure.
- Contains:
  - classes, defined types, tasks, task plans, functions, resource types and providers, etc.
- Reusable and shareable.
Puppet Blocks - Modules
Examples

class autofs {
  package { autofs: ensure => latest }
  service { autofs: ensure => running }

  file { "/etc/auto.homes":
    source => "puppet://$servername/modules/autofs/auto.homes"
  }
  file { "/etc/auto.master":
    content => template("autofs/auto.master.erb")
  }
}
Puppet Blocks - **Classes**

- Defined as a collection of resources
- Classes are defined inside Puppet manifest files.
- Created by writing a class definition in a manifest (`.pp`) file.
Example:

```puppet
class f3backup (  
  $backup_home = '/backup',  
  $backup_server = 'default',  
  $myname = $::fqdn,  
  $ensure = 'directory',  
) {  
  include '::f3backup::common'  
  if ( $myname == '' or $myname == undef ) {  
    fail('myname must not be empty')  
  }  
  @@file { "${backup_home}/f3backup/${myname}":
    # To support 'absent', though force will be needed
    ensure => $ensure,
    owner  => 'backup',
    group  => 'backup',
    mode   => '0644',
    tag    => "f3backup-{$backup_server}",
    }
  }
}
```
Puppet Blocks - Resources

- Resources are the fundamental unit for modeling system configurations.
- Puppet code in manifest file or any other file is called a resource declaration.
- Expressed in Declarative Modelling Language (DML)

Resource Declaration:
- Resource Type
- Resource Parameter
- Attributes
- values
Puppet Blocks - Resources

• Example

```ruby
user { 'student1':
  ensure => present,
  uid    => '552',
  shell  => '/bin/bash',
  home   => '/home/student1',
}
```

**Resource Type** – `user`


**Attributes** – `ensure`, `uid`, `shell`, `home`.

**Values** – values of `ensure`, `uid`, `shell`, `home`. 
Resource Type:
Some more types exec, file, cron, mount, package, service, user, etc

Attributes – Values pairs:
For each resource type, Puppet offers a set of Attributes. 
  e.g. For User, attributes can be
    home, groups, gid, password_max_age, password_min_age, shell, etc.
Puppet Blocks - **Manifest**

- a directory containing puppet DSL files.
- .pp extension (Puppet Program)
- Puppet manifest consists of the following important components –
  - **Resources**
  - **Templates** (used to construct configuration files on the node).
  - **Nodes** (contain all the definition related to a client node)
  - **Classes**
Service deployment solutions – A Summary

Standard Specification
- A standard to follow for service deployment and management: TOSCA

Orchestrator
- A tool to understand the standard: Cloudify, Opera

Automation Engine
- To implement the necessary service deployment and lifecycle management tasks: Ansible and Puppet
Research topics (contd...)

• Distributed file systems and data management
• Legacy System Migration
• Cloud SLAs
• Performance/scale/price estimation
• Mobile service deployment
• Deployment Model Selection
• Cost & Risk Analysis
What next ???
Let’s move to lab session...
References

1. https://www.oasis-open.org/
7. https://www.slideshare.net/melsatar/cloud-deployments-models
References

9. https://docs.cloudify.co/4.3.0/about/manager_architecture/components/
10. https://www.slideshare.net/knoldus/introduction-to-ansible-81369741
Thank you