Cloud Computing – Lecture 13

Cloud Service Deployment Models

03 May 2022

Chinmaya Dehury

Chinmaya.Dehury@ut.ee
Outlines

• Deployment models in cloud
• Service deployment in cloud
• Service deployment standard - TOSCA
• Graphical modeling tools
  • Cloudify
  • Winery GMT – Eclipse Winery
Cloud deployment models

1. Private Cloud
2. Public Cloud
3. Hybrid Cloud
4. Managed Private Cloud
5. Hosted Private Cloud
6. Community Cloud Services
Cloud deployment models

1. Private Cloud
2. Public Cloud
3. Hybrid Cloud
4. Managed Private Cloud
5. Hosted Private Cloud
6. Community Cloud Services
Cloud Deployment solutions

How about the service deployment?

How would you deploy your own service, e.g. a messaging platform, your own website, etc.
Cloud Deployment solutions

Some proprietary solutions

AWS CloudFormation
AWS Serverless Application Model
Azure resource manager

...
AWS CloudFormation

- Speed up cloud provisioning with infrastructure as code
- Create templates to describe your AWS resources and their properties.
- Simplify infrastructure management
- Cross account & cross-region management
- Dependency management
- JSON or YAML formatted text file

- [https://aws.amazon.com/cloudformation/](https://aws.amazon.com/cloudformation/)
AWS Serverless Application Model (SAM)

• Src: https://aws.amazon.com/serverless/sam/
• An open-source framework for building serverless applications
• SAM is built atop AWS CloudFormation
• Build serverless applications in simple and clean syntax
• Shorthand syntax to express
  • functions,
  • APIs,
  • databases, and
  • event source mappings.
• During deployment,
  • SAM transforms and expands the SAM syntax → AWS CloudFormation syntax
  • enabling you to build serverless applications faster.
• Use YAML
Azure Resource Manager (ARM) templates

- Create declarative templates to provision the resources, define their dependencies
- Faster deployment of the resource with parallelism deployment
- Use of infrastructure as code (in JSON file)
- Some list of templates:
  - [http://armviz.io](http://armviz.io) for visual representation of your template.
Azure Resource Manager (ARM) templates

- Create declarative templates to provision the resources, define their dependencies
- Faster deployment of the resource with parallelism deployment
- Use of infrastructure as code (in JSON file)
- Some list of templates:
  - [http://armviz.io](http://armviz.io) for visual representation of your template.
Azure Resource Manager (ARM) features

- Declarative syntax: in JSON
- Orchestration:
  - Resource Manager (RM) takes the dependencies into account
  - RM may deploy resources in parallel
- Modular files:
  - Import external templates
  - Foster reusability of templates
- Built-in validation:
  - Validator makes sure that the template is workable
- Tracked deployments:
  - review the deployment history
- CI/CD integration
  - Integration with existing CI/CD tools (Github, Azure DevOps, Azure Pipelines)
Modelling Cloud Services — Expected features

- Declarative syntax: in human-readable form: JSON or YAML
- Orchestration
- Modular files: Nested template, import external templates. Grouping the service component.
- Reusability
- Extensibility
- Built-in validation
- Template versioning
- Tracked deployments
- CI/CD integration
TOSCA
TOSCA standard overview

• Topology and Orchestration Specification for Cloud Applications (TOSCA)
  • TOSCA is developed under the OASIS Open Consortium.
  • Members include Cisco, Fujitsu, Hewlett Packard Enterprise (HPE), Huawei, IBM, NEC Corporation, NetApp, NetCracker, U.S. NIST, Red Hat, SAP and many more.

• 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 overview (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:
Some imp links

- 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
  - https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/os/TOSCA-Simple-Profile-YAML-v1.3-os.pdf

- TOSCA Implementations

- TOSCA Implementation Stories
  - https://www.oasis-open.org/tosca-implementation-stories/

- OASIS TOSCA YouTube Video Playlist
  - https://www.youtube.com/c/Oasis-openOrg
TOSCA modelling

Let’s start with an example

1. Create a VM in OpenStack Cloud
2. Start the VM
3. Install Nifi on that VM
4. Start Nifi Service
TOSCA modelling

Let’s start with an example

```yaml
tosca_definitions_version: tosca_simple_yaml_1_3
metadata:
  targetNamespace: "radon.blueprints.examples"
  name: "DataPipelineExample"
  imports: 

  topology_template:
    node_templates:
      OpenStack_0:
        type: radon.nodes.VM.OpenStack
        metadata:
          x: "523"
          y: "828"
          displayName: "OpenStack"
        properties:
          flavor: "m1.medium"
          key_name: "chinmayadehury"
          image: "13a94b11-98ee-43a4-ad29-00ae97e8f790"
          ssh_username: "centos"
          name: "NifiHost2-temp"
          network: "provider_64_net"

Nifi_0:
  type: radon.nodes.nifi.Nifi
  metadata:
    x: "522"
    y: "619"
    displayName: "Nifi"
  properties:
    port: 8080
    component_version: "1.12.1"
  requirements:
    - host:
        node: OpenStack_0
        relationship: con_HostedOn_2
        capability: host
```

A VM in private OpenStack Cloud

Install Apache Nifi atop openstack VM
TOSCA modelling

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

**Basic blocks:**
- Topology
- State
  - Nodes
  - relationship
TOSCA modelling

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

Basic blocks:
• Topology
• State
  • Nodes
  • Relationship
• Requirements – Capabilities
• Lifecycle
• 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 – Service Template

Service Template Example

tosca_definitions_version: tosca_simple_yaml_1_3

metadata:
  targetNamespace: "radon.blueprints.examples"
  name: "DataPipelineExample"

imports:
  - file: radonnodesVM___OpenStack.tosca

namespace_uri: radon.nodes.VM
namespace_prefix: radonnodesVM


topology_template:
  node_templates:
    OpenStack_0:
      type: radon.nodes.VM.OpenStack
      properties:
        flavor: "m1.medium"
        key_name: "chinmaya"
        image: "13a94b11-98e-ad29-00ae97e8f790"
        ssh_username: "centos"
        name: "NifiHost"
        network: "provider_64_net"

    Nifi_0:
      type: radon.nodes.nifi.Nifi
      properties:
        port: 8080
        component_version: "1.12.1"
      requirements:
        - host:
          node: OpenStack_0
          relationship: con_HostedOn_2
          capability: host

relationship_templates:
  con_HostedOn_2:
    type: tosca.relationships.HostedOn
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.

```plaintext
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 - Example

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

```yaml
tosca_definitions_version: tosca_simple_yaml_1_3

node_types:
  radon.nodes.VM.OpenStack:
    derived_from: tosca.nodes.Compute
    attributes:
      public_address:
    properties:
      flavor:
        type: string
        description: OpenStack flavor id (flavor names are not accepted)
      key_name: ....
    interfaces:
      Standard:
        type: tosca.interfaces.node.lifecycle.Standard
        operations:
          create:
            inputs:
              key_name: ....
              image: ....
              ....
            implementation:
              primary: /nodetypes/radon.nodes.VM/OpenStack/files/create/create.yml
          delete: ....
    artifacts:
      create:
        type: radon.artifacts.Ansible
        file: /nodetypes/radon.nodes.VM/OpenStack/files/create/create.yml
        delete: ....
```

TOSCA modelling – Node template

- Define topology with
  - Nodes (*node_templates*)
  - Relationship (*relationship_templates*)
TOSCA modelling – Node template

• 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

```yaml
tosca_definitions_version: tosca_simple_yaml_1_3
metadata:
  targetNamespace: "radon.blueprints.examples"
  name: "DataPipelineExample"
imports:
  topology_template:
    node_templates:
      ConsGCSBucket_0:
      PubGCS_0:
      ConsS3Bucket_0:
      PubsAzureBlob_0:
      OpenStack_0:
      Nifi_0:
```
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 template
  • To describe connections, dependencies, deployment ordering

• Requirements and capabilities are implicit way to describe the relationship

• Some basic explicit relationship types:
  • DependsOn: represents a general dependency relationship between two nodes.
  • ConnectsTo: represents a network connection relationship between two nodes.
  • AttachesTo: represents an attachment relationship between two nodes. For example, attaching a storage node to a Compute node.
  • RoutesTo: represents an intentional network routing between two Endpoints in different networks.
  • HostedOn: represents a hosting relationship between two nodes.
TOSCA modelling - **Interfaces**

- Interfaces are *reusable entities*
- Define a *set of operations*
- Can be included as part of a *Node type* or *Relationship Type* definition.

```plaintext
tosca.interfaces.node.lifecycle.Standard:
  derived_from: tosca.interfaces.Root
  create:
    description: Standard lifecycle create operation.
  configure:
    description: Standard lifecycle configure operation.
  start:
    description: Standard lifecycle start operation.
  stop:
    description: Standard lifecycle stop operation.
  delete:
    description: Standard lifecycle delete operation.
```

**Operations**
TOSCA modelling – Interfaces operations

tosca_definitions_version: tosca_simple_yaml_1_3
imports: 
node_types:
  radon.nodes.nifi.Nifi:
    derived_from: tosca.nodes.SoftwareComponent
    metadata: 
    properties: 
    capabilities: 
    interfaces:
      Standard:
        type: tosca.interfaces.node.lifecycle.Standard
        operations:
          stop:
            implementation:
              primary: /nodeltypes/radon.nodes.nifi/Nifi/files/stop/stop.yml
          start:
            implementation:
              primary: /nodeltypes/radon.nodes.nifi/Nifi/files/start/start.yml
          create:
            inputs:
              tarball_version:
                type: string
                required: true
                default: { get_property: [ SELF, component_version ] }
            implementation:
              primary: /nodeltypes/radon.nodes.nifi/Nifi/files/create/create.yml
          configure:
            inputs: 
            implementation:
              primary: /nodeltypes/radon.nodes.nifi/Nifi/files/configure/configure.yml
          delete:
            inputs:
              tarball_version: 
            implementation:
              primary: /nodeltypes/radon.nodes.nifi/Nifi/files/delete/delete.yml
TOSCA modelling - Interfaces

• 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 modelling – Requirements & Capabilities

tosca_definitions_version: tosca_simple_yaml_1_3

node_types:
  radon.nodes.datapipeline.MidwayPB:
    derived_from: radon.nodes.datapipeline.PipelineBlock
    requirements:
      - ConnectToPipeline:
        capability: radon.capabilities.datapipeline.ConnectToPipeline
        node: radon.nodes.datapipeline.PipelineBlock
        relationship: radon.relationships.datapipeline.ConnectNifiLocal
        occurrences: [ 1, UNBOUNDED ]
      - host:
        capability: tosca.capabilities.Container
        node: radon.nodes.nifi.Nifi
        relationship: tosca.relationships.HostedOn
        occurrences: [ 1, 1 ]

Capabilities:
  ConnectToPipelineRemote:
    occurrences: [ 1, UNBOUNDED ]
    valid_source_types: [ radon.nodes.datapipeline.SourcePB, radon.nodes.datapipeline.MidwayPB ]
    type: radon.capabilities.datapipeline.ConnectToPipeline
  ConnectToPipeline:
    occurrences: [ 1, UNBOUNDED ]
    valid_source_types: [ radon.nodes.datapipeline.SourcePB, radon.nodes.datapipeline.MidwayPB ]
    type: radon.capabilities.datapipeline.ConnectToPipeline
TOSCA modelling – Requirements & Capabilities

tosca_definitions_version: tosca_simple_yaml_1_3

node_types:
  radon.nodes.datapipeline.MidwayPB:
    derived_from: radon.nodes.datapipeline.PipelineBlock
    requirements:
      - ConnectToPipeline:
          capability: radon.capabilities.datapipeline.ConnectToPipeline
          node: radon.nodes.datapipeline.PipelineBlock
          relationship: radon.relationships.datapipeline.ConnectNifiLocal
          occurrences: [ 1, UNBOUNDED ]
          - host:
              capability: tosca.capabilities.Container
              node: radon.nodes.nifi.Nifi
              relationship: tosca.relationships.HostedOn
              occurrences: [ 1, 1 ]

capabilities:
  ConnectToPipelineRemote:
    occurrences: [ 1, UNBOUNDED ]
    valid_source_types: [ radon.capabilities.datapipeline.PipelineBlock ]
  ConnectToPipeline:
    occurrences: [ 1, UNBOUNDED ]
    valid_source_types: [ radon.capabilities.datapipeline.PipelineBlock ]

Cloud Computing - Lec 13: Cloud Service Deployment models
TOSCA Implementation

• Cloudify - http://getcloudify.org/
• DICER - https://github.com/DICERs/DICER
• Eclipse Winery - https://projects.eclipse.org/projects/soa.winery
• Opera - https://github.com/xlab-si/xopera-opera
• OpenTOSCA - http://www.opentosca.org/
• see full list...
TOSCA implementation

We will discuss following two

• Cloudify

• RADON Graphical Modelling Tool (GMT)
TOSCA implementation - Cloudify
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 YMAL file called blueprint.
- GUI to model the service blueprint.
TOSCA implementation - Cloudify
Cloudify - Architecture

[Image: https://docs.cloudify.co/4.3.0/images/architecture/cloudify_advanced_architecture.png]

Cloud Computing - Lec 13: Cloud Service Deployment models
Cloudify – some components

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 PostGresSQL.
Cloudify – some components

Cloudify Manager primarily is built with open-source components (contd...):

• **RabbitMQ**: Queueing deployment tasks, logs and events, Queueing metrics

• **Riemann**: policy-based decision maker

• **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
Cloudify – further resources

https://vimeo.com/677206773

Try Cloudify Self-Managed Environments:
https://cloudify.co/download/
TOSCA Graphical Modelling Tool (GMT)
- Eclipse Winery
The Graphical Modeling Tool (GMT) is developed based on Eclipse Winery.

- A web-based environment to graphically model TOSCA-based application topologies.
- Support latest version of TOSCA standard.

It includes:
- A component to manage TOSCA types and templates,
- A Topology Modeler that enables to graphically compose application topologies and specify configuration properties,
- A file-based backend to store, import, and export TOSCA entities.
Graphical Modelling Tool (GMT) - Overview

- Version control of model development
- Development of
  - Node type
  - Relationship type
  - Capability type
  - Service templates
  - Artifacts type
- Export and importing the service template
  - Export in TOSCA exchange format is a Cloud Service Archive (CSAR)
  - CSAR is essentially a ZIP file
- Commit to remote GitHub repository
Node type development

- What you can do while developing a node type
  - Define properties and attributes
  - Define requirements and capabilities
  - Define Interface, operations and their inputs
  - Readme file, documentations, appearance etc

- What you can’t do
  - Prepare the implementation file (e.g. Ansible files, script files)
Eclipse Winery - Component

HTTP REST API

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOSCA XML Model Importer &amp; Exporter</td>
<td>TOSCA XML Model Importer &amp; Exporter</td>
</tr>
<tr>
<td>XaaS Packager</td>
<td>XaaS Packager</td>
</tr>
<tr>
<td>Versioning &amp; Difference Calculation</td>
<td>Versioning &amp; Difference Calculation</td>
</tr>
<tr>
<td>TOSCA YAML Model Importer &amp; Exporter</td>
<td>TOSCA YAML Model Importer &amp; Exporter</td>
</tr>
<tr>
<td>BPMN4TOSCA Management Plan Importer</td>
<td>BPMN4TOSCA Management Plan Importer</td>
</tr>
<tr>
<td>CSAR Packager</td>
<td>CSAR Packager</td>
</tr>
<tr>
<td>Topology Completion</td>
<td>Topology Completion</td>
</tr>
<tr>
<td>Accountability</td>
<td>Accountability</td>
</tr>
<tr>
<td>TOSCA YAML Model to TOSCA XML Model Transformer</td>
<td>TOSCA YAML Model to TOSCA XML Model Transformer</td>
</tr>
<tr>
<td>BPMN4TOSCA to BPEL Transformer</td>
<td>BPMN4TOSCA to BPEL Transformer</td>
</tr>
<tr>
<td>Consistency Check</td>
<td>Consistency Check</td>
</tr>
<tr>
<td>Splitting &amp; Matching</td>
<td>Splitting &amp; Matching</td>
</tr>
<tr>
<td>Compliance Checker</td>
<td>Compliance Checker</td>
</tr>
<tr>
<td>Key-based Policy Template Generator</td>
<td>Key-based Policy Template Generator</td>
</tr>
<tr>
<td>Key &amp; Access Control List (ACL) Management</td>
<td>Key &amp; Access Control List (ACL) Management</td>
</tr>
<tr>
<td>Implementation Artifact Generator</td>
<td>Implementation Artifact Generator</td>
</tr>
<tr>
<td>Templates, Types, Plans &amp; CSARs Repository</td>
<td>Templates, Types, Plans &amp; CSARs Repository</td>
</tr>
<tr>
<td>Matching Templates Repository</td>
<td>Matching Templates Repository</td>
</tr>
<tr>
<td>Compliance Rules Repository</td>
<td>Compliance Rules Repository</td>
</tr>
<tr>
<td>Winery Backend System Components</td>
<td>Winery Backend System Components</td>
</tr>
</tbody>
</table>

Functionality provided by the OpenTOSCA Container and usable in the Winery if a Container instance is running

Img src: [https://winery.readthedocs.io/en/latest/_images/components.png](https://winery.readthedocs.io/en/latest/_images/components.png)
Eclipse Winery - Component

Four parts:
- the templates, types, plans, and CSARs management,
- the TOSCA topology model editor,
- the BPMN4TOSCA management plan editor, and
- the repository to store templates, types, plans, etc

![Diagram of Winery UI Components and HTTP REST API](https://winery.readthedocs.io/en/latest/_images/components.png)

Img src: [https://winery.readthedocs.io/en/latest/_images/components.png](https://winery.readthedocs.io/en/latest/_images/components.png)
Eclipse Winery - Component

Four parts:

- **The templates, types, plans, and CSARs management**
  - *Provides* functionality to access, store, or delete TOSCA elements in the *Repository*
  - Enables *managing* all TOSCA types, templates, node types, relationship types, policy types, artifact types, artifact templates, and artifacts such as virtual machine images.

![Diagram showing Winery UI Components and HTTP REST API](https://winery.readthedocs.io/en/latest/_images/components.png)

Eclipse Winery - Component

Four parts:

- **The TOSCA topology model editor,**
  - Enables the creation of service templates as directed graphs.
  - Modeled service templates can be exported in YAML format

![Diagram of Eclipse Winery components](https://winery.readthedocs.io/en/latest/_images/components.png)
Eclipse Winery – Installation & web interface overview

docker run -it -p 8080:8080 \
  -e PUBLIC_HOSTNAME=localhost \
  -e WINERY_FEATURE_RADON=true \
  -e WINERY_REPOSITORY_PROVIDER=yaml \
  -e WINERY_REPOSITORY_URL=https://github.com/radon-h2020.radon-particles \n  opentosca/radon-gmt

Web interface of Winery GMT
Eclipse Winery – Installation & web interface overview
Eclipse Winery – Installation & web interface overview
Service template Orchestration – 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 programme.
• [https://github.com/xlab-si/xopera-opera](https://github.com/xlab-si/xopera-opera)
Automation Engine – Ansible*  

- 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.  

Example of Ansible code:
```
- name: Install the latest version of Apache  
yum:  
  name: httpd  
  state: latest
```
Automation Engine – 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.

```
- name: Install the latest version of Apache
  yum:
    name: httpd
    state: latest
```
Service deployment solutions – A Summary

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

**Graphical Modelling Tool**
A tool to model the TOSCA-based cloud services graphically: **RADON GMT- Eclipse Winery, Cloudify**

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

**Automation Engine**
To implement the necessary service deployment and lifecycle management tasks: **Ansible and Puppet**
What next ???
Let’s move to lab session...
(Use of Eclipse Winery, TOSCA, Opera)
References

5. https://docs.cloudify.co/4.3.0/about/manager_architecture/components/
6. https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/os/TOSCA-Simple-Profile-YAML-v1.3-os.html#Toc26969470
Thank you