DevOps – Lecture 10

Application Deployment Modelling

8 Nov 2022

Chinmaya Dehury

Chinmaya.Dehury@ut.ee
Where are we now?

1. SDLC + Why DevOps
2. DevOps Phases + Cloud Computing – Basics
3. VCS – GitLab
4. Containerization – Kubernetes
5. Microservices
6. Automation – Ansible
7. CI/CD – GitLab
8. Continuous Testing
9. Monitoring

10. **Application Deployment Modeling**

11. DataOps
12. DevOps + Serverless
13. Challenges + Future Scope
OUTLINE

• Application Deployment
• Deployment phase in DevOps
• Deployment Methods
  • Blue-Green
  • Canary
• TOSCA: Application Modelling standard
• Winery: Graphical Modelling Tool for TOSCA
• Opera: TOSCA Orchestrator
Updates

- **Week-14**: Lec-13: 29 Nov: Industrial and Research Challenges and Future Scope
  - No Lab. Catch up time.

- **Week-11**: Lec-10: 08 Nov: Application Deployment Modelling
  - Lab: TOSCA Modelling & Orchestration
    - Updated Room number for Group 1: 2040

*HPC-UT Notice*: Update your VM, OpenSSL
What is Application Deployment

Process of installing, configuring, accessing using a URL

Deployment Activities:

- Create/Delete
- Installation /Uninstallation
- Configuration
- Activation/deactivation
- Update
Deployment phase in DevOps

You basically deploy the software product on a production environment

• Product is ready for
  • different deployment contexts
  • different deployment environments
  • different deployment platforms

• If the release process for deployment to the production environment is also automated, it is referred to as continuous deployment.

• In a continuous deployment process, every validated change is automatically released to users.
Continuous Deployment* – Benefits

• **Eliminate manual processes** for Continuous Delivery
• Improve the **focus on the product**.
• Automate the repetitive tasks and focus on actual testing.
• **Scale** from a single application to an Enterprise IT portfolio.
• **Quick response** to customer feedback

*Discussed in previous lectures*
Points to know, while deploying an application

• How to minimize existing application **downtime**, if any.
• How to manage and resolve incidents with **minimal impact on users**.
• How to **address failed deployments** in a reliable, effective way.
• How to minimize people and process errors to achieve predictable, repeatable deployments.

Src: [https://cloud.google.com/architecture/application-deployment-and-testing-strategies](https://cloud.google.com/architecture/application-deployment-and-testing-strategies)
Some popular deployment methods

- Blue-Green Deployment
- Canary Deployment
- Atomic Deployment
- Shadow deployment
- Rolling deployment
Blue-Green Deployment Methods

- Also known as red/black deployment
- blue represents the current version
- green represents the new version

LTAT.06.015 - Application Deployment Modelling
Canary Deployment Methods (1/3)

• Deploy the change to a small subset of servers
• Test, or wait until satisfied
• Less impact if fail
• serves as an early warning indicator
  • with less impact on downtime
Canary Deployment Methods (2/3)

- Reduces the risk of deploying new version
- Slow rolling out
- Rolling update deployment pattern

Canary Deployment Methods (3/3)

- Reduces the risk of deploying new version
- Slow rolling out
- Side-by-Side Deployments

# Blue-Green vs Canary Deployment Methods

<table>
<thead>
<tr>
<th>Key Benefits</th>
<th>Blue-Green</th>
<th>Canary Deployment Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Requirement</td>
<td>Huge</td>
<td>Less compared to B/G</td>
</tr>
<tr>
<td>Rollout time</td>
<td>Negligible</td>
<td>Slow</td>
</tr>
<tr>
<td>Rollback time</td>
<td>Instant</td>
<td>Compared to B/G, it is more.</td>
</tr>
<tr>
<td>Downtime (for mission critical services, affect SLA)</td>
<td>Zero Downtime</td>
<td>Zero Downtime</td>
</tr>
<tr>
<td>Failure impact</td>
<td>Less</td>
<td>Less</td>
</tr>
<tr>
<td>Time-to-market</td>
<td>Quick</td>
<td>Quick</td>
</tr>
</tbody>
</table>

**Diagram:**

The diagrams illustrate the comparison between Blue-Green and Canary deployment methods. The Blue-Green method involves simultaneously deploying a new version of the application while the current version continues to run, allowing for a fast and controlled rollback in case of issues. The Canary method, on the other hand, deploys the new version to a small portion of users initially and gradually increases the deployment as it is deemed successful, promoting a more gradual and measured roll-out.
Modelling the applications before deployment

How would you model your application if you have hundreds of components?
Modelling the applications before deployment

How would you model your application if you have hundreds of components?

- Dependencies
- Lifecycle of each component
- Different types of component
- Workflow
- Abstraction
Modelling the applications before deployment

How would you model your application if you have hundreds of components?

TOSCA

- Dependencies
- Lifecycle of each component
- Different types of component
- Workflow
- Abstraction
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 (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 standard overview (contd...)

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

• 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
Modelling with TOSCA – an example
TOSCA modelling

Let’s start with an example
TOSCA modelling

Let’s start with an example

```
Nginx
Container

Hosted on

Docker
Engine

Hosted on

ETAIS
Virtual
Machine
```
Let’s start with an example
TOSCA modelling

Let’s start with an example

Edges Or Relationship

Nodes

Hosted on

Hosted on

Nginx Container

Docker Engine

Virtual Machine

LTAT.06.015 - Application Deployment Modelling
TOSCA modelling

Let’s start with an example

A graph (Nodes + Edges)

- **Nodes**
  - Nginx Container
  - Docker Engine
  - ETAIS (Virtual Machine)

- **Edges**
  - Hosted on

- **Relationship**
  - Edges or Relationship
TOSCA modelling – Topology template

Let’s start with an example

In TOSCA topology_template

Nodes

- Nginx Container
  - Hosted on

- Docker Engine
  - Hosted on

- Virtual Machine

In TOSCA Node_template

Edges

In TOSCA relationship_template
TOSCA representation – Node_template

TOSCA representation of existing VM

Assumption: VM (remoteworkstation) is already created
TOSCA representation– Node_template

A graph (Nodes + Edges)

In TOSCA
relationship_template

In TOSCA
topology_template

TOSCA representation of Docker Engine software component

TOSCA representation of Virtual Machine

Assumption: VM (remoteworkstation) is already created
TOSCA representation – Node_template

A graph (Nodes + Edges) in TOSCA:

- **Nginx Container**
  - Hosted on **Docker Engine**
  - Host: RemoteWorkstation

**TOSCA representation of Nginx container software component**

**TOSCA representation of Docker Engine software component**

**TOSCA representation of existing VM**

Assumption: VM (remoteworkstation) is already created
TOSCA representation – Node_template

Virtual Machine

In TOSCA topology_template

Docker Engine

Hosted on

Nginx Container

Hosted on
TOSCA representation – relationship_templates

In TOSCA topology_template

```
topology_template:
  node_templates:
    containerNginx 0:
      type: radon.nodes.docker.containerNginx
      metadata:
      properties:
      requirements:
        - host:
          relationship: con_HostedOn_1
          capability: host
    RemoteWorkstation 0:
      type: radon.nodes.docker.DockerEngineUT
      metadata:
      requirements:
        - host:
          relationship: con_HostedOn_0
          capability: host
          relationship_templates:
            con_HostedOn_0:
              type: tosca.relationships.HostedOn
            con_HostedOn_1:
              type: tosca.relationships.HostedOn
```
TOSCA representation – Node + relationship template

In TOSCA

**topology_template**

```yaml
1  topology_template:
2    node_templates:
3      containerNginx_0:
4      RemoteWorkstation_0:
5      DockerEngineUT_0:
6    relationship_templates:
7      con_HostedOn_0:
8      con_HostedOn_1:
```

- **Nginx Container**
- **Docker Engine**
- **ETAIS Virtual Machine**

Hosted on relationships:
- Nginx Container is hosted on Docker Engine
- Docker Engine is hosted on ETAIS Virtual Machine
TOSCA representation – Node + relationship template

Graphical representation of topology_template using Winery
TOSCA – In general
TOSCA modelling - Basic blocks

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

Basic blocks:
• Topology
  • Nodes
  • Relationship
• Composition
• Requirements – Capabilities
• State (nodes, relationship)
• Lifecycle
• Policy

Graphical representation of topology_template using Winery
• Templates define the structure of a cloud service/application.
• Usually described in YAML or in XML format
• Structure of service template
  • Node Type
  • Relationship type
  • Node template
  • Relationship template
TOSCA modelling – Application/Service Template

Service Template

```
tosca_definitions_version: tosca_simple_yaml_1_3
metadata:
  targetNamespace: "radon.blueprints.examples"
  name: "myfirstToscaApp"
  LTAT.06.015 - Application Deployment Modelling
imports:
  - file: radonnodesVM_RemoteWorkstation.tosca
    namespace_url: radon.nodes.VM
    namespace_prefix: radonnodesVM
  - file: radonnodesdocker_containerNginx.tosca
    namespace_url: radon.nodes.docker
    namespace_prefix: radonnodesdocker
  - file: radonnodesdocker_DockerEngineUT.tosca
    namespace_url: radon.nodes.docker
    namespace_prefix: radonnodesdocker

topology_template:
  node_templates:
    containerNginx_0:
      type: radon.nodes.VM.RemoteWorkstation
      properties:
        ssh_username: "centos"
        Keyfile: "/home/ubuntu/chinmayadheury"
        external_ip: "172.17.91.1"
    RemoteWorkstation_0:
      type: radon.nodes.docker.DockerEngineUT
      properties:
        x: "G48"
        y: "91"
        display_name: "DockerEngineUT"
    requirements:
      - host:
          node: RemoteWorkstation_0
          relationship: con_HostedOn_0
          capability: host
          relationship_templates:
            con_HostedOn_0:
              type: tosca.relationships.HostedOn
            con_HostedOn_1:
              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.
TOSCA modelling – Node Type

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

---

topology_template:
node_templates:
  containerNginx_0:
    type: radon.nodes.docker.containerNginx
    metadata:
    properties:
      container_port: "80"
      image_name: "nginx:1.13"
      container_name: "mynginxserver"
      host_port: "8082"
    requirements:
      - host:

RemoteWorkstation_0:
  type: radon.nodes.VM.RemoteWorkstation
  metadata:
  properties:
    ssh_username: "centos"
    KeyFile: "/home/ubuntu/chinmayadehury"
    external_ip: "172.17.191.11"
  requirements:
    - host:

DockerEngineUT_0:
  type: radon.nodes.docker.DockerEngineUT
  metadata:
  requirements:
    - host:

---

LTAT.06.015 - Application Deployment Modelling
TOSCA - Relationships

Relationship types

defaultNginxServer_chinnm
(containerNginx)

HostedOn

DockerEngineUT
(DockerEngineUT)

myWorkerVM_chinnmaya
(RemoteWorkstation)

Relationships
TOSCA Relationship types

**AttachesTo:**
- represents an attachment relationship between two nodes.
- *For example*,
  - attaching a storage node to a Compute node
Relationship types

- **HostedOn**: represents a hosting relationship between two nodes.
- **ConnectsTo**: represents a network connection relationship between two nodes.
- **DependsOn**: represents a general dependency relationship between two nodes.
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 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 modelling – Properties & Attributes

- Properties:
  - *What you provide during modelling*

- Attribute:
  - *Runtime variable*

- Example:
  - *EC2 node type*
    - *To create an EC2 instance*
TOSCA modelling – Requirements and Capability

LTAT.06.015 - Application Deployment Modelling 52
TOSCA modelling – Requirements and Capability

• Typically, *Requirements* are described against a known *Capability Type*

• **Capabilities Types**
  - `tosca.capabilities.Compute`
  - `tosca.capabilities.Network`
  - `tosca.capabilities.Storage`
  - `tosca.capabilities.Container`
  - ...
  - ...

![Diagram of TOSCA modelling](image)
TOSCA Interface types

• Interfaces are **reusable** entities that define a set of **operation**
• can be included as part of a **Node type** or **Relationship Type** definition
• An **operations** may have code or scripts associated with them that orchestrators can execute
• **Possible Operations**
  • Create
  • Configure
  • Start
  • Stop
  • Delete
How would you model graphically?
Winery: Graphical Modelling Tool for TOSCA
Winery – Graphical Modelling tool

Winery – Graphical Modelling tool for TOSCA-based application

Service Templates  Node Types  Relationship Types  Other Elements

Add new
Import CSAR
Group by Namespace
Winery – Topology Modeler

Winery – Give a set of

Functionalities

List of node types

Canvas area to model your application/service

- Winery
- Topology Modeler
- Give a set of

- LTAT.06.015
  - Application Deployment Modelling
Winery – Topology Modeler – node types

- List of node types available in Winery
- Below are some examples
- Note: All these node types are developer under RADON project ([https://radon-h2020.eu/](https://radon-h2020.eu/))

**List of node types under**
- `radon.nodes.VM`
  - AWS
  - EC2
  - Rn
  - OpenStack
  - RemoteWorkstation
  - waldurOS

**List of node types under**
- `radon.nodes.docker`
  - Rn
  - DockerApplication
  - DockerEngine
  - DockerEngineUT
  - container
  - containerNginx
Winery – Node details - Properties

LTAT.06.015 - Application Deployment Modelling
CSAR : Packaging the service template

- TOSCA YAML CSAR file is an archive file using the ZIP file format
- The structure of CSAR complies with the TOSCA Simple Profile Specification.

CSAR: Packaging the service template

CSAR : Packaging the service template

• Example: `myfirstToscaApp.csar`

• csar directory structure:
CSAR: Packaging the service template

- Example: `myfirstToscaApp.csar`
- csar directory structure:
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/radon-h2020/xopera-opera
• OpenTOSCA - http://www.opentosca.org/
• see full list...
TOSCA Implementation

- Cloudify - http://getcloudify.org/
- DICER - https://github.com/DICERs/DICER
- Eclipse Winery - https://projects.eclipse.org/projects/soa.winery
- xOpera - https://github.com/radon-h2020/xopera-opera
- OpenTOSCA - http://www.opentosca.org/
- see full list...

Examination and Comparison of TOSCA Orchestration Tools:
xOpera orchestrator
TOSCA implementation - xOpera

• 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

- Recall Automation Lecture;
  - https://courses.cs.ut.ee/2022/DevOps/fall/Main/Lectures

- 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.
End-to-End Service Modelling and Orchestration

Plan your Application

Model the service using Winery

myfirstToscaApp

export csar

myfirstToscaApp.csar

xOpera orchestrator

Automation using Ansible

Implementation using

• Python
• Sh
• Java
• ...

myfirstToscaApp
Service deployment solutions – A Summary

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

**Graphical modelling tool**
- Winery

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

**Automation Engine**
- To implement the necessary service deployment and lifecycle management tasks: Ansible
Lab Sessions

- Winery
- xOpera
- Implement below application
References

1. https://www.oasis-open.org/
7. https://www.slideshare.net/melsatar/cloud-deployments-models
9. https://www.slideshare.net/knoldus/introduction-to-ansible-81369741
Any Question?

THANK YOU