TOSCA-based Container Orchestration on Mesos
Two-Phase Deployment of Cloud Applications using Container-based Artifacts

11th Symposium and Summer School On Service-Oriented Computing,
June 25 – June 30, 2017 in Crete, Greece

Stefan Kehrer, Wolfgang Blochinger
Reutlingen University
TOSCA-based Container Orchestration on Mesos

Agenda

» Introduction
» TOSCA at a glance
» Two-phase deployment
» TOSCA-based Integration
» Conclusion
TOSCA-based Container Orchestration on Mesos

Agenda

» Introduction
» TOSCA at a glance
» Two-phase deployment
» TOSCA-based Integration
» Conclusion
Introduction

- Fast software release cycles are an essential business requirement
- DevOps proposed to foster collaboration of development and operations personnel
- Deployment automation is key to enable fast release cycles
  - DevOps artifacts (e.g., scripts or templates) encapsulate deployment logic
  - Two classes of DevOps artifacts (Wetinger et al. [3]):
    - **Node-centric artifacts**
      - Deployment logic of a single node
    - **Environment-centric artifacts**
      - Deployment logic of multi-node application topologies

---

Introduction

- Application components are packaged using containers
- Node-centric deployment logic is specified (e.g., in a Dockerfile) and employed to build a container-based artifact (e.g., Docker image)
- An application topology is comprised of multiple container-based artifacts
- Templates are used for environment-centric deployment logic
- Several container management systems evolved to deploy container-based applications:
Introduction

» Problems

– Heterogeneous orchestration solutions lead to vendor-lock-in [4]
– Current approaches do not integrate node-centric and environment-centric deployment logic, e.g., components of a node cannot be configured after node creation

» Contributions

– Two-phase deployment process to integrate node-centric and environment-centric deployment
  – TOSCA-based modeling constructs
– TOSCA-based container management system on top of Apache Mesos

TOSCA-based Container Orchestration on Mesos

Agenda

» Introduction
» **TOSCA at a glance**
» Two-phase deployment
» TOSCA-based Integration
» Conclusion
TOSCA at a glance

» Topology and Orchestration Specification for Cloud Applications

- Standardized language for portable cloud applications (OASIS)
- Applications are described as topology graphs and management plans
- Topology model describes a topology graph of typed nodes and relationships
- Deployment artifacts to instantiate nodes
- Implementation artifacts to execute lifecycle operations
- Application description captured in service template / Cloud service archive (CSAR)
TOSCA-based Container Orchestration on Mesos

Agenda

» Introduction
» TOSCA at a glance
» **Two-phase deployment**
» TOSCA-based Integration
» Conclusion
TOSCA-based Container Orchestration on Mesos

Two-phase deployment

Node-centric deployment

- Node-related configurations are applied
- Requires a fine-granular topology model
- Environment-related dependencies have to be considered
- Results in a container-based artifact

Environment-centric deployment

- Deployment based on a container management system
- Requires coarse-granular topology model enriched with container-based artifacts

Node-centric topology model

Environment-centric topology model
TOSCA-based Container Orchestration on Mesos

Two-phase deployment

Two-phase deployment process

- Allows two views on the application topology
- Integrates node-centric and environment-centric deployment logic
- Standards-based service template ensures portability

Process fragment supporting two-phase deployment
TOSCA-based Container Orchestration on Mesos

Agenda

» Introduction

» TOSCA at a glance

» Two-phase deployment

» TOSCA-based Integration
  » Environment-centric deployment phase
  » Node-centric deployment phase

» Conclusion
TOSCA-based Container Orchestration on Mesos

Agenda

» Introduction
» TOSCA at a glance
» Two-phase deployment
» TOSCA-based Integration
  » Environment-centric deployment phase
  » Node-centric deployment phase
» Conclusion
TOSCA-based Container Orchestration on Mesos

TOSCA-based Integration

» Modeling containers
– Resource properties for CPU shares, memory size, and disk size
– Docker image deployment artifact
– Repository for container image provisioning
– Create operation requires deployment artifact and several input values

Environment-centric topology model

```
1  mysql_container:
2      type: cst.nodes.Docker.MySQL
3  properties:
4      cpu_shares: 0.5
5      mem_size: 512 MB
6      disk_size: 500 MB
7  capabilities:
8      ...
9  artifacts:
10     my_image:
11         file: mysql/mysql-server
13             --> Container.Docker
14         repository: docker_hub
15  interfaces:
16     Standard:
17         create:
18             implementation: my_image
19             inputs:
20                 MYSQL_ROOT_PASSWORD: my-root-pw
21                 MYSQL_USER: my-user
22                 MYSQL_PASSWORD: my-user-pw
23                 MYSQL_DATABASE: my-db
```

MySQL node template in YAML
TOSCA-based Container Orchestration on Mesos

TOSCA-based Integration

» Modeling relationships

– Capability-Requirement pair in line with TOSCA endpoints concept instead of Docker links

– IP address is assigned during deployment ⇒ No port mapping!

– Relationship `connect_to_db` requires configuration, e.g., set IP address in configuration files

```yaml
mysql_container:  
type: cst.nodes.Docker.MySQL  
properties:  
  ...
  capabilities:  
    db_endpoint:  
      properties:  
        protocol: tcp  
        port: 3306  
  ...
```

MySQL node template in YAML

```yaml
wordpress_container:  
type: cst.nodes.Docker.WordPress  
properties:  
  ...
  capabilities:  
    ...
  requirements:  
    - db_endpoint:  
        node: mysql_container  
        relationship: connect_to_db  
  ...
```

Wordpress node template in YAML
TOSCA-based Container Orchestration on Mesos

TOSCA-based Integration

» Embedded implementation artifacts
  – Implementation artifacts provided by container-based deployment artifacts
  – Benefits:
    – Single artifact per node
    – Using repositories instead of CSAR
    – Additional distribution of implementation artifacts, e.g., by using SSH, is not necessary
    – Runtime dependencies are part of the container

Artifacts in TOSCA

Embedded implementation artifacts

Deployment Artifact

Node Type

VM Image

realizes

Shell Script

Docker Image

provides

Node Type

realizes

Implementation Artifacts

Implementation Artifacts
**TOSCA-based Integration**

Node-triggered implementation artifacts

- Triggered by the container itself during the creation
- Configure a container directly after its instantiation
- Input values are static or can be resolved before node instantiation

Environment-triggered implementation artifacts

- Triggered by orchestration solutions
- Configure a node instance
- Input values are dynamic and dependent on runtime information

» How to expose these artifacts?

```
1  mysql_container:
2    type: cst.nodes.Docker.MySQL
3    ...
17   interfaces:
18    Standard:
19      create:
20        implementation: my_image
21        inputs:
22          MYSQL_ROOT_PASSWORD: my-root-pw
23          MYSQL_USER: my-user
24          MYSQL_PASSWORD: my-user-pw
25          MYSQL_DATABASE: my-db
```

MySQL node template in YAML
TOSCA-based Container Orchestration on Mesos

TOSCA-based Integration

» Management APIs

- Standards-based API to wrap environment-triggered implementation artifacts
- Container-based artifacts encapsulate implementation artifacts and management APIs
- Added keyname api for modeling management APIs

Management API concept

connect_to_db relationship template in YAML

```yaml
connect_to_db:
  type: ConnectsTo
  interfaces:
  - Configure:
      pre_configure_source:
        api:
          type: REST/HTTP
          protocol: http
          method: POST
          format: json
          path: /api/configure
          port: 8080
      inputs:
        WORDPRESS_DB_HOST: { get_attribute: -> [TARGET, ip_address] }
        WORDPRESS_DB_USER: my-user
        WORDPRESS_DB_PASSWORD: my-user-pw
        WORDPRESS_DB_NAME: my-db
```
TOSCA-based Container Orchestration on Mesos

Agenda

» Introduction
» TOSCA at a glance
» Two-phase deployment
» **TOSCA-based Integration**
  » Environment-centric deployment phase
  » **Node-centric deployment phase**
» Conclusion
TOSCA-based Container Orchestration on Mesos

TOSCA-based Integration

- Modeling construct to model “internal” structure of containers

Topology of a container-based example application

```
1  wordpress_container:
2      type: cst.nodes.Docker.WordPress
3      children: [wordpress, apache, php_module]
4     ...
```

Modeling child nodes for wordpress_container
TOSCA-based Container Orchestration on Mesos

TOSCA-based Integration

- Modeling construct to model “internal” structure of containers

![Topology of a container-based example application](image)

```yaml
1  wordpress_container:
2      type: cst.nodes.Docker.WordPress
3      children: [wordpress, apache_php]
4      ...
```

Modeling child nodes for `wordpress_container`
TOSCA-based Container Orchestration on Mesos

TOSCA-based Integration

```yaml
1  wordpress:
2     type: cst.nodes.DockerInternal.WordPress
3     ...
4  artifacts:
5     build_spec:
6        file: wp/Dockerfile
8     ...
9  interfaces:
10     Standard:
11     create:
12        implementation:
13        primary: build_spec
14        dependencies:
15        - wp/pre_configure.sh
16        - wp/managementAPI-rest.jar
```

Node-centric topology

Node-centric deployment phase

Implement / Gather Artifacts → Construct Build Specification → Build Container-based Artifact → Test Container-based Artifact

Node-centric Deployment Phase

wordpress_node_template_in_YAML
TOSCA-based Container Orchestration on Mesos

TOSCA-based Integration

Node-centric topology

Wordpress node template in YAML

```yaml
wordpress:
  type: cst.nodes.DockerInternal.WordPress
  ...
  artifacts:
    build_spec:
      file: wp/Dockerfile
    ...
  interfaces:
    Standard:
      create:
        implementation:
          primary: build_spec
          dependencies:
            - wp/pre_configure.sh
            - wp/managementAPI-rest.jar
```

Node-centric Deployment Phase

Implement/Gather Artifacts → Construct Build Specification → Build Container-based Artifact → Test Container-based Artifact
TOSCA-based Container Orchestration on Mesos

TOSCA-based Integration

Node-centric topology

wordpress_container

wordpress

HostedOn

apache_php

Node-centric topology

wordpress node template in YAML

```yaml
1  wordpress:
2    type: cst.nodes.DockerInternal.WordPress
3      ...
4  artifacts:
5    build_spec:
6      file: wp/Dockerfile
8      ...
9  interfaces:
10     Standard:
11     create:
12        implementation:
13        primary: build_spec
14        dependencies:
15          - wp/pre_configure.sh
16          - wp/managementAPI-rest.jar
```

Node-centric Deployment Phase

Node-centric deployment phase
TOSCA-based Container Orchestration on Mesos

Agenda

» Introduction
» TOSCA at a glance
» Two-phase deployment
» TOSCA-based Integration
» Conclusion
Conclusion

- TOSCA-based orchestration ensures a uniform interface for container management systems and addresses an open research topic [4]
- CSAR captures all required deployment logic
- Environment-triggered implementation artifacts support dynamic runtime management
- Two-phase deployment process for creating and maintaining multi-node application topologies
- Developer is responsible for exposing management APIs

Future work

- Tool support for node-centric deployment phase
- Higher degree of automation for specific application classes

Thank You

Stefan Kehrer
Reutlingen University
stefan.kehrer@reutlingen-university.de