Hochschule Reutlingen
Reutlingen University

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

Universität Stuttgart   Herman Hollerith Zentrum   Hochschule Reutlingen Reutlingen University

Agenda

Hochschule Reutlingen
Reutlingen University

» Introduction

» TOSCA at a glance

» Two-phase deployment

» TOSCA-based Integration

» Conclusion

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 (Wettinger et al. [3]):

Puppet Module

Chef Cookbook

Docker Image

Node-centric artifacts
Deployment logic of a single node

Heat Orchestration Template (HOT)

Juju Bundle

CloudFormation Template

Environment-centric artifacts
Deployment logic of multi-node application topologies

[3] Wettinger J, Breitenbücher U, Kopp O, Leymann F (2016) Streamlining devops automation for cloud applications using tosca as standardized metamodel. Future Generation Computer Systems 56(C):317-332

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:

Marathon & Apache Mesos

Docker Swarm

Amazon EC2 Container Service

Kubernetes

Google Container Engine

Hochschule Reutlingen
Reutlingen University

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

[4] Kratzke N, Quint PC (2017) Understanding cloudnative applications after 10 years of cloud computing - a systematic mapping study. Journal of Systems and Software 126:1-16

Agenda

Hochschule Reutlingen
Reutlingen University

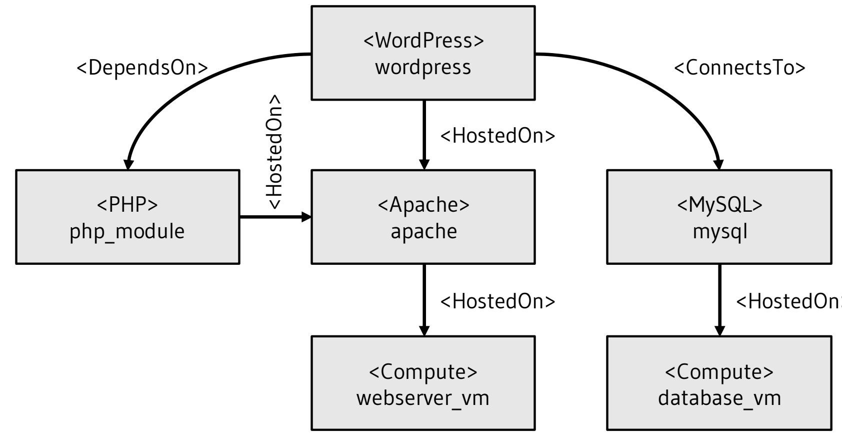
- Introduction
- **TOSCA at a glance**
- Two-phase deployment
- TOSCA-based Integration
- Conclusion

TOSCA at a glance

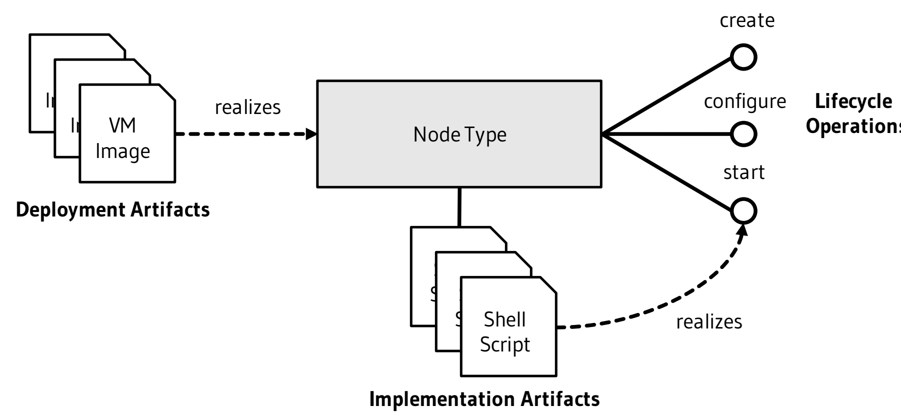
Hochschule Reutlingen
Reutlingen University

» 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)



Topology graph of example application



Deployment and implementation artifacts

Agenda

Hochschule Reutlingen
Reutlingen University

» Introduction

» TOSCA at a glance

» **Two-phase deployment**

» TOSCA-based Integration

» Conclusion

Two-phase deployment

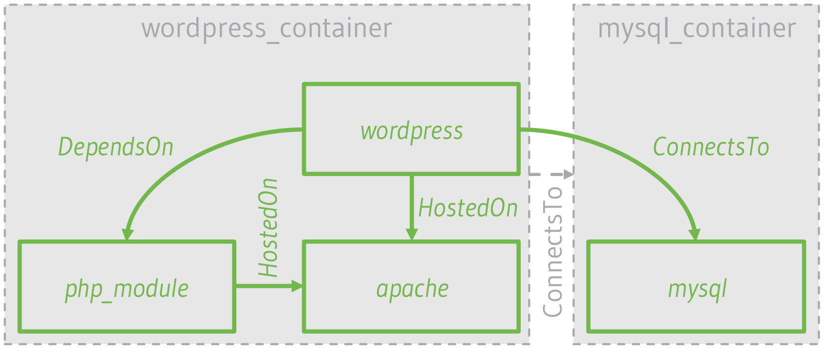
Hochschule Reutlingen
Reutlingen University

Node-centric deployment

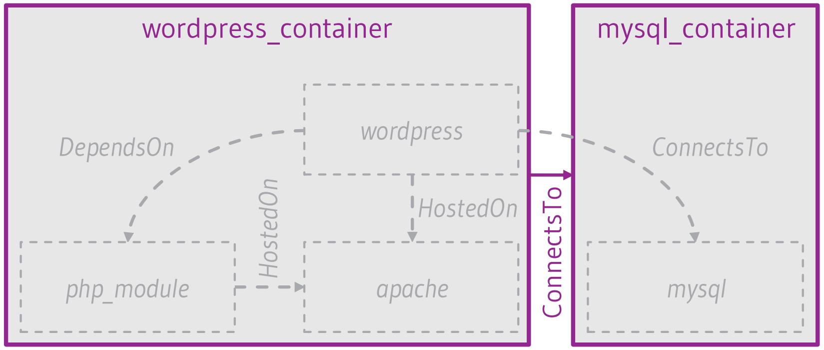
- Node-related configurations are applied
- Requires a fine-granular topology model
- Environment-related dependencies have to be considered
- Results in a container-based artifact

Environment-centric deployment

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



Node-centric topology model

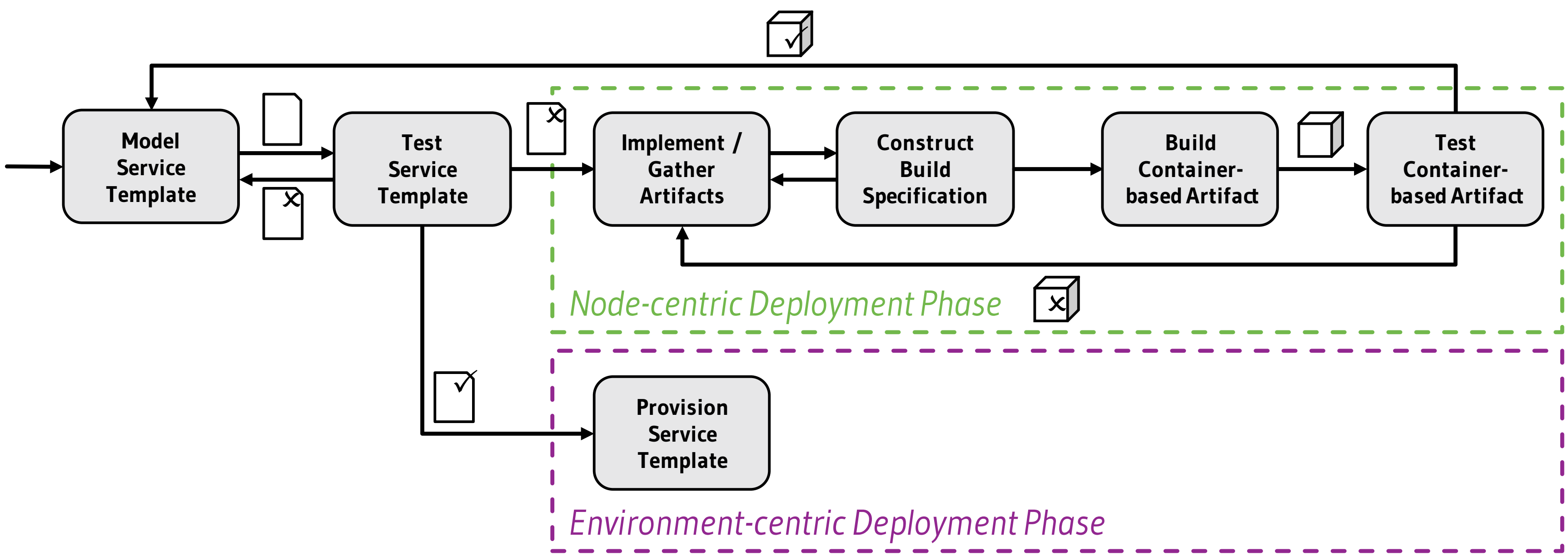


Environment-centric topology model

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

Hochschule Reutlingen
Reutlingen University

Agenda

- Introduction
- TOSCA at a glance
- Two-phase deployment
- **TOSCA-based Integration**
  - Environment-centric deployment phase
  - Node-centric deployment phase
- Conclusion

Agenda

- Introduction
- TOSCA at a glance
- Two-phase deployment
- **TOSCA-based Integration**
  - **Environment-centric deployment phase**
  - Node-centric deployment phase
- Conclusion

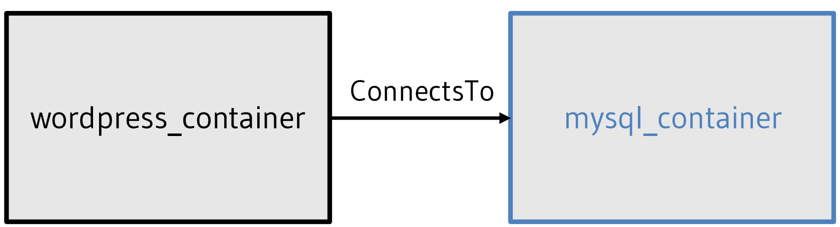
TOSCA-based Integration

» 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

```
mysql_container:
  type: cst.nodes.Docker.MySQL
  properties:
    cpu_shares: 0.5
    mem_size: 512 MB
    disk_size: 500 MB
  capabilities:
    ...
  artifacts:
    my_image:
      file: mysql/mysql-server
      type: tosca.artifacts.Deployment.Image.
         -> Container.Docker
      repository: docker_hub
  interfaces:
    Standard:
      create:
        implementation: my_image
        inputs:
          MYSQL_ROOT_PASSWORD: my-root-pw
          MYSQL_USER: my-user
          MYSQL_PASSWORD: my-user-pw
          MYSQL_DATABASE: my-db
```

MySQL node template in YAML



Environment-centric topology model

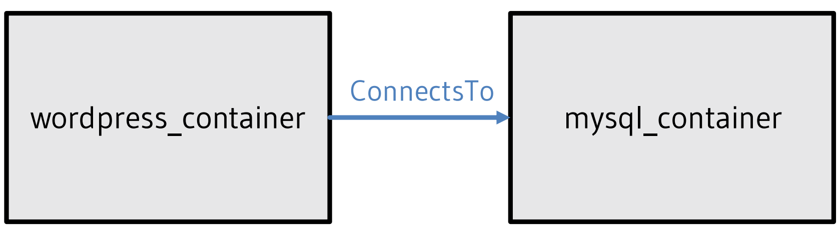
TOSCA-based Integration

» 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



Environment-centric topology model

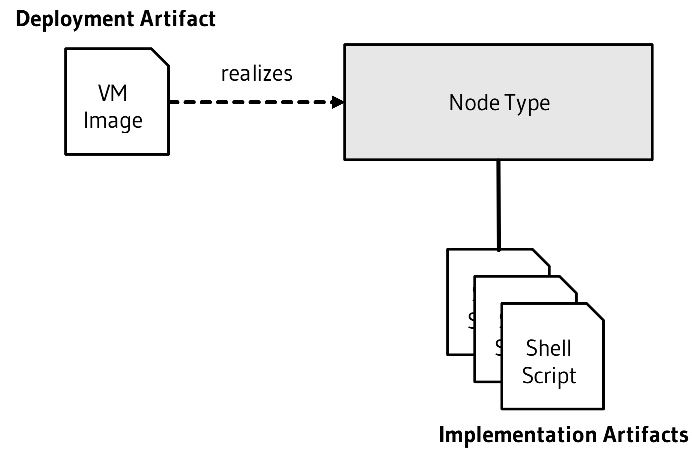
```yaml
wordpress_container:
  type: cst.nodes.Docker.WordPress
  properties:
    ...
  capabilities:
    ...
  requirements:
    - db_endpoint:
        node: mysql_container
        relationship: connect_to_db
  ...
```

Wordpress node template in YAML

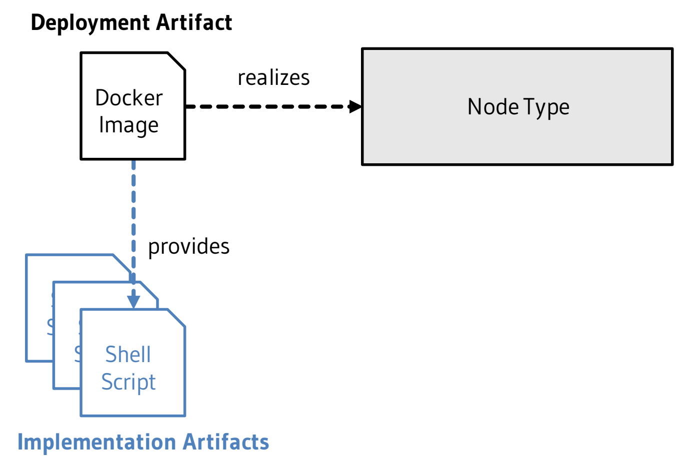
TOSCA-based Integration

» 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

TOSCA-based Integration

Hochschule Reutlingen
Reutlingen University

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

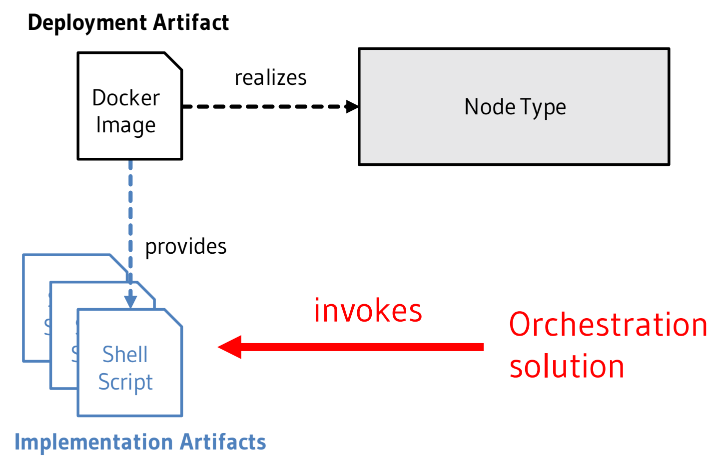
```yaml
mysql_container:
  type: cst.nodes.Docker.MySQL
  ...
  interfaces:
    Standard:
      create:
        implementation: my_image
        inputs:
          MYSQL_ROOT_PASSWORD: my-root-pw
          MYSQL_USER: my-user
          MYSQL_PASSWORD: my-user-pw
          MYSQL_DATABASE: my-db
```

MySQL node template in YAML

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?

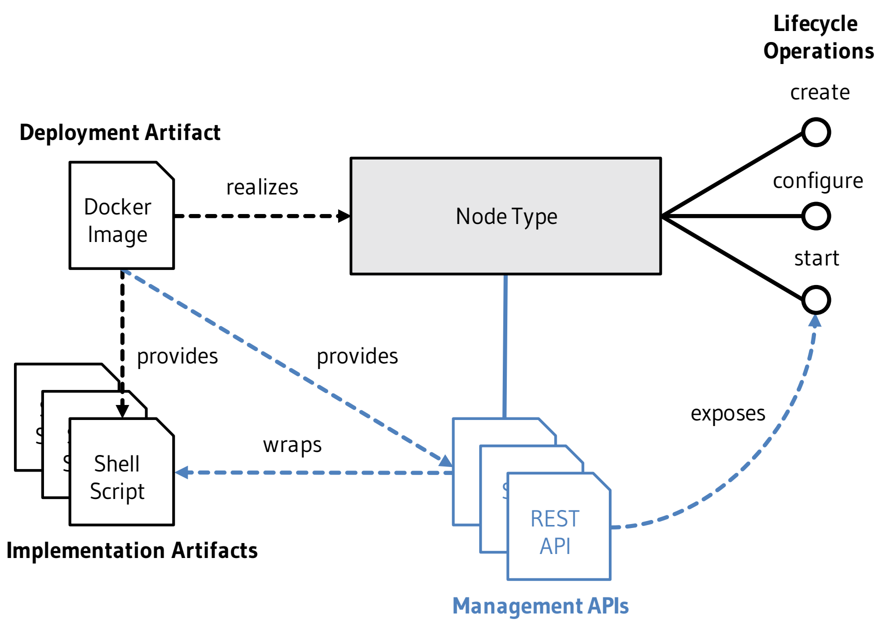


Embedded implementation artifacts

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

```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
```

connect_to_db relationship template in YAML

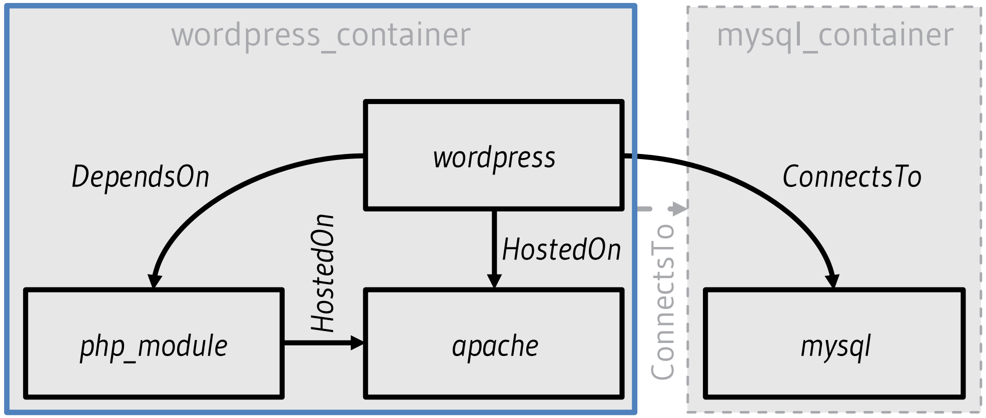
Agenda

Hochschule Reutlingen
Reutlingen University

- Introduction
- TOSCA at a glance
- Two-phase deployment
- **TOSCA-based Integration**
  - Environment-centric deployment phase
  - **Node-centric deployment phase**
- Conclusion

TOSCA-based Integration

- Modeling construct to model "internal" structure of containers



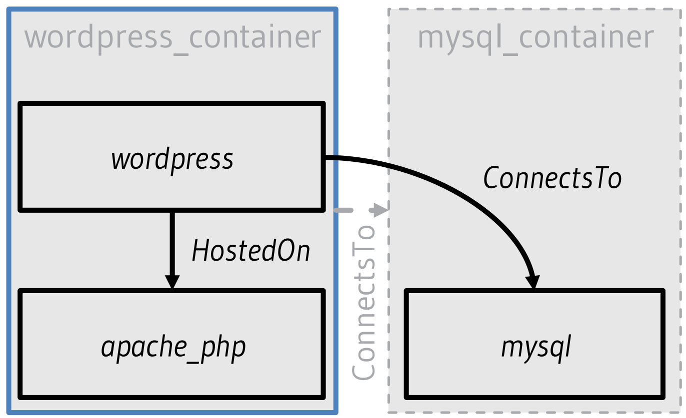
Topology of a container-based example application

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

Modeling child nodes for wordpress_container

TOSCA-based Integration

- Modeling construct to model “internal” structure of containers

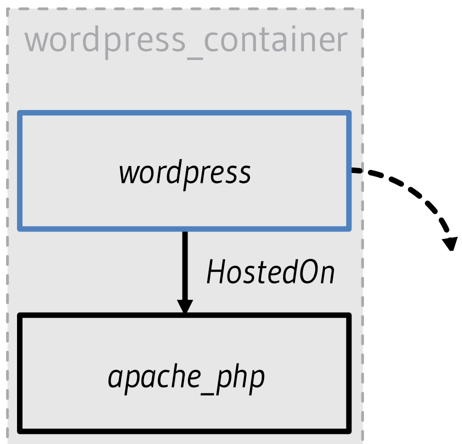


Topology of a container-based example application

```
wordpress_container:
  type: cst.nodes.Docker.WordPress
  children: [wordpress, apache_php]
  ...
```

Modeling child nodes for wordpress_container

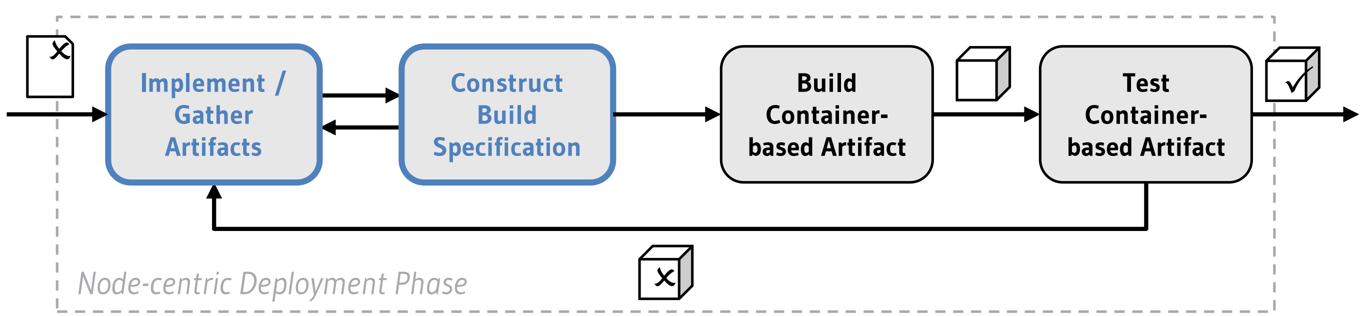
TOSCA-based Integration



Node-centric topology

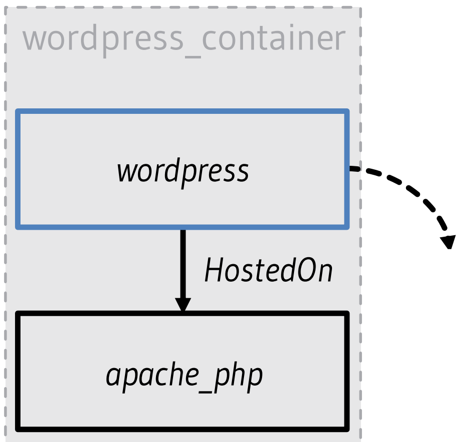
```yaml
wordpress:
  type: cst.nodes.DockerInternal.WordPress
  ...
  artifacts:
    build_spec:
      file: wp/Dockerfile
      type: cst.artifacts.Deployment.ImageSpec.Docker
    ...
  interfaces:
    Standard:
      create:
        implementation:
          primary: build_spec
          dependencies: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

wordpress node template in YAML



Node-centric deployment phase

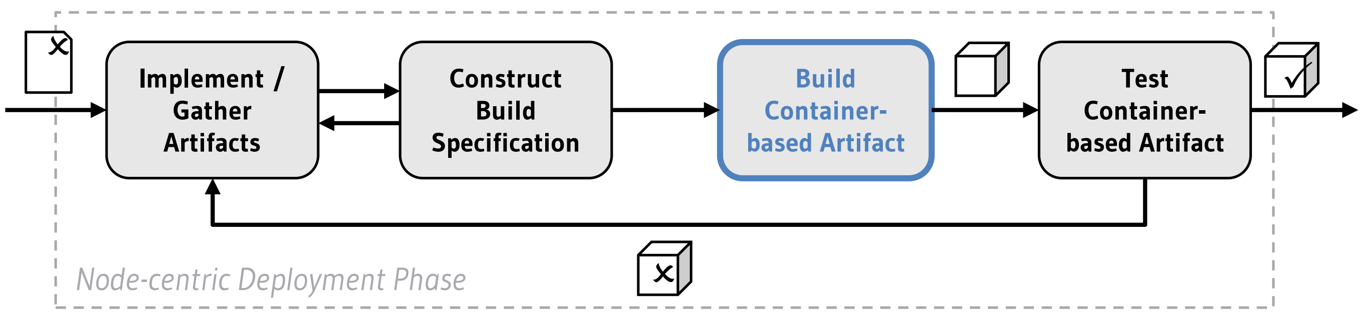
TOSCA-based Integration



Node-centric topology

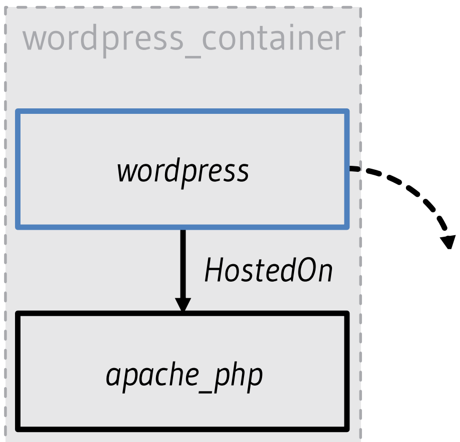
```yaml
wordpress:
  type: cst.nodes.DockerInternal.WordPress
  ...
  artifacts:
    build_spec:
      file: wp/Dockerfile
      type: cst.artifacts.Deployment.ImageSpec.Docker
    ...
  interfaces:
    Standard:
      create:
        implementation:
          primary: build_spec
          dependencies: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

wordpress node template in YAML



Node-centric deployment phase

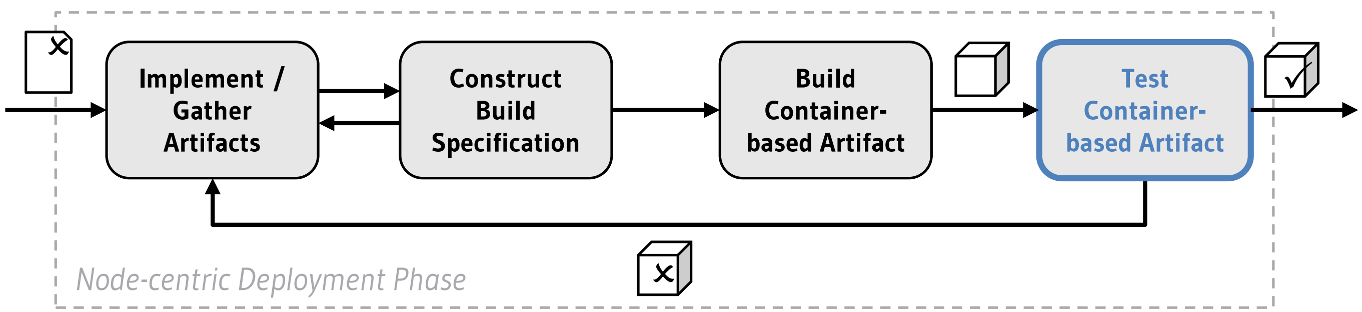
TOSCA-based Integration



Node-centric topology

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

wordpress node template in YAML



Node-centric deployment phase

Agenda

Hochschule Reutlingen
Reutlingen University

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

Conclusion

Hochschule Reutlingen
Reutlingen University

» 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

[4] Kratzke N, Quint PC (2017) Understanding cloudnative applications after 10 years of cloud computing - a systematic mapping study. Journal of Systems and Software 126:1-16

Thank You

Hochschule Reutlingen
Reutlingen University

Thank You

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