TOSCA Simple Profile in YAML Version 1.0

Committee Specification Draft 01

27 March 2014

Specification URIs

This version:

http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/csd01/TOSCA-Simple-Profile-YAML-v1.0-csd01.pdf (Authoritative)
http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/csd01/TOSCA-Simple-Profile-YAML-v1.0-csd01.html
http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/csd01/TOSCA-Simple-Profile-YAML-v1.0-csd01.doc

Previous version:

N/A

Latest version:

http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/TOSCA-Simple-Profile-YAML-v1.0.pdf (Authoritative)
http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/TOSCA-Simple-Profile-YAML-v1.0.html
http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/TOSCA-Simple-Profile-YAML-v1.0.doc

Technical Committee:

OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC

Chairs:

Paul Lipton (paul.lipton@ca.com), CA Technologies
Simon Moser (smoser@de.ibm.com), IBM

Editors:

Derek Palma (dpalma@vnomic.com), Vnomic
Matt Rutkowski (mrutkows@us.ibm.com), IBM
Thomas Spatzier (thomas.spatzier@de.ibm.com), IBM

Related work:

This specification is related to:

- *Topology and Orchestration Specification for Cloud Applications Version 1.0*. Edited by Derek Palma and Thomas Spatzier. 25 November 2013. OASIS Standard. http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html.

Declared XML namespace:

- http://docs.oasis-open.org/tosca/ns/simple/yaml/1.0

Abstract:

This document defines a simplified profile of the TOSCA version 1.0 specification in a YAML rendering which is intended to simplify the authoring of TOSCA service templates. This profile defines a less verbose and more human-readable YAML rendering, reduced level of indirection between different modeling artifacts as well as the assumption of a base type system.

Status:

This document was last revised or approved by the OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at https://www.oasis-open.org/committees/tosca/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (https://www.oasis-open.org/committees/tosca/ipr.php.

Citation format:

When referencing this specification the following citation format should be used:

[TOSCA-Simple-Profile-YAML-v1.0]

TOSCA Simple Profile in YAML Version 1.0. Edited by Derek Palma, Matt Rutkowski, and Thomas Spatzier. 27 March 2014. OASIS Committee Specification Draft 01. http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/csd01/TOSCA-Simple-Profile-YAML-v1.0-csd01.html. Latest version: http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/TOSCA-Simple-Profile-YAML-v1.0.html.

Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see https://www.oasis-open.org/policies-guidelines/trademark for above guidance.

Table of Contents

Table of Figures

1 Objective

The TOSCA Simple Profile in YAML specifies a rendering of TOSCA which aims to provide a more accessible syntax as well as a more concise and incremental expressiveness of the TOSCA DSL in order to minimize the learning curve and speed the adoption of the use of TOSCA to portably describe cloud applications.

This proposal describes a YAML rendering for TOSCA. YAML is a human friendly data serialization standard (http://yaml.org/) with a syntax much easier to read and edit than XML. As there are a number of DSLs encoded in YAML, a YAML encoding of the TOSCA DSL makes TOSCA more accessible by these communities.

This proposal prescribes an isomorphic rendering in YAML of a subset of the TOSCA v1.0 ensuring that TOSCA semantics are preserved and can be transformed from XML to YAML or from YAML to XML. Additionally, in order to streamline the expression of TOSCA semantics, the YAML rendering is sought to be more concise and compact through the use of the YAML syntax.

2 Summary of key TOSCA concepts

The TOSCA metamodel uses the concept of service templates to describe cloud workloads as a graph of node templates modeling the components a workload is made up of and as relationship templates modeling the relations between those components. TOSCA further provides a type system of node types to describe the possible building blocks for constructing a service template, was well as relationship type to describe possible kinds of relations. Both node- and relationship types may define lifecycle operations to implement the behavior an orchestration engine can invoke when instantiating a service template. For example, a node type for some software product might provide a ‘create’ operation to handle the creation of an instance of a component at runtime, or a ‘start’ or ‘stop’ operation to handle a start or stop event triggered by an orchestration engine. Those lifecycle operations are backed by implementation artifacts such as scripts or Chef recipes that implement the actual behavior.

An orchestration engine processing a TOSCA service template uses the mentioned lifecycle operations to instantiate single components at runtime, and it uses the relationship between components to derive the order of component instantiation. For example, during the instantiation of a two-tier application that includes a web application that depends on a database, an orchestration engine would first invoke the ‘create’ operation on the database component to install and configure the database, and it would then invoke the ‘create’ operation of the web application to install and configure the application (which includes configuration of the database connection).

The TOSCA simple profile assumes a number of base types (node types and relationship types) to be supported by each compliant environment such as a ‘Compute’ node type, a ‘Network’ node type or a generic ‘Database’ node type (see Appendix B). Furthermore, it is envisioned that a large number of additional types for use in service templates will be defined by a community over time. Therefore, template authors in many cases will not have to define types themselves but can simply start writing service templates that use existing types. In addition, the simple profile will provide means for easily customizing existing types, for example by providing a customized ‘create’ script for some software.

3 A “hello world” template for TOSCA Simple Profile in YAML

As mentioned before, the TOSCA simple profile assumes the existence of a base set of node types (e.g., a ‘Compute’ node) and other types for creating TOSCA Service Templates. It is envisioned that many additional node types for building service templates will be created by communities. Consequently, a most basic TOSCA template for deploying just a single server would look like the following:

Example 1 - TOSCA Simple "Hello World"

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: Template for deploying a single server with predefined properties.

node_templates:
  my_server:
    type: tosca.nodes.Compute
    properties:
      # compute properties
      disk_size: 10
      num_cpus: 2
      mem_size: 4
      # host image properties
      os_arch: x86_64
       os_type: linux
       os_distribution: rhel
       os_version: 6.5
```

The template above contains the definition of one single ‘Compute’ node template with predefined (hardcoded) values for number of CPUs, memory size, etc. When instantiated in a provider environment, the provider would allocate a physical or virtual server that meets those specifications. The set of properties of any node type, as well as their schema definition, is defined by the respective node type definitions, which a TOSCA orchestration engine can resolve to validate the properties provided in a template.

3.1 Requesting input parameters and providing output

Typically, one would want to allow users to customize deployments by providing input parameters instead of using hardcoded values inside a template. In addition, it is useful to pass output that describes the deployed environment (such as the IP address of the deployed server) to the user. A refined service template with corresponding **`inputs`** and **`outputs`** sections is shown below.

Example 2 - Template with input and output parameter sections

```
tosca_definitions_version: tosca_simple_yaml_1_0
```



```
description: Template for deploying a single server with predefined properties.

inputs:
  cpus:
    type: integer
    description: Number of CPUs for the server.
    constraints:
      - valid_values: [ 1, 2, 4, 8 ]

node_templates:
  my_server:
    type: tosca.nodes.Compute
    properties:
      # Compute properties
      num_cpus: { get_input: cpus }
      mem_size: 4
      disk_size: 10
      # host image properties
      os_arch: x86_32
       os_type: linux
       os_distribution: ubuntu
       os_version: 12.04

outputs:
  server_ip:
    description: The IP address of the provisioned server.
    value: { get_property: [ my_server, ip_address ] }
```

Note that the **inputs** section of a TOSCA template allows for defining optional constraints on each input parameter to restrict possible user input. Further note that TOSCA provides for a set of intrinsic functions like **get_input** or **get_property** to reference elements within the template or to retrieve runtime values.

4 TOSCA template for a simple software installation

Software installations can be modeled in TOSCA as node templates that get related to the node template for a server on which the software shall be installed. With a number of existing software node types (e.g. either created by the TOSCA work group or a community) template authors can just use those node types for writing service templates as shown below.

Example 3 - Simple (MySQL) software installation on a TOSCA Compute node

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: Template for deploying a single server with MySQL software on top.

inputs:
  # omitted here for sake of brevity

node_templates:
  mysql:
    type: tosca.nodes.DBMS.MySQL
    properties:
      dbms_root_password: { get_input: my_mysql_rootpw }
      dbms_port: { get_input: my_mysql_port }
    requirements:
      - host: db_server

  db_server:
    type: tosca.nodes.Compute
    properties:
      # omitted here for sake of brevity
```

The example above makes use of a node type **tosca.nodes.DBMS.MySQL** for the **mysql** node template to install MySQL on a server. This node type allows for setting a property **dbms_root_password** to adapt the password of the MySQL root user at deployment. The set of properties and their schema has been defined in the node type definition. By means of the **get_input** function, a value provided by the user at deployment time is used as value for the **dbms_root_password** property. The same is true for the **dbms_port** property.

The **mysql** node template is related to the **db_server** node template (of type **tosca.nodes.Compute**) via the **requirements** section to indicate where MySQL is to be installed. In the TOSCA metamodel, nodes get related to each other when one node has a requirement against some feature provided by another node. What kinds of requirements exist is defined by the respective node type. In case of MySQL, which is software that needs to be installed or hosted on a compute resource, the node type defines a requirement called **host**, which needs to be fulfilled by pointing to a node template of type **tosca.nodes.Compute**.

Within the **`requirements`** section, all entries contain the name of a requirement as key and the identifier of the fulfilling entity as value, expressing basically a named reference to some other node. In the example above, the **`host`** requirement is fulfilled by referencing the **`db_server`** node template.

5 Overriding behavior of predefined node types

Node types in TOSCA have associated implementations that provide the automation (e.g. in the form of scripts or Chef recipes) for lifecycle operations of a node. For example, the node type implementation for MySQL will provide the scripts to configure, start, or stop MySQL at runtime.

If it is desired to use a custom script for one of the operation defined by a node type in the context of a specific template, the default implementation can be easily overridden by providing a reference to the own automation in the template as shown in the following example:

Example 4 - Node Template overriding its Node Type's "configure" interface

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: Template for deploying a single server with MySQL software on top.

inputs:
  # omitted here for sake of brevity

node_templates:
  mysql:
    type: tosca.nodes.DBMS.MySQL
    properties:
      dbms_root_password: { get_input: my_mysql_rootpw }
      dbms_port: { get_input: my_mysql_port }
    requirements:
      - host: db_server
    interfaces:
      Lifecycle:
        configure: scripts/my_own_configure.sh

  db_server:
    type: tosca.nodes.Compute
    properties:
      # omitted here for sake of brevity
```

In the example above, an own script for the **configure** operation of the MySQL node type's lifecycle interface is provided. The path given in the example above is interpreted relative to the template file, but it would also be possible to provide an absolute URI to the location of the script.

Operations defined by node types can be thought of as hooks into which automation can be injected. Typically, node type implementations provide the automation for those hooks. However, within a template, custom automation can be injected to run in a hook in the context of the one, specific node template (i.e. without changing the node type).

6 TOSCA template for database content deployment

In the example shown in section 4 the deployment of the MySQL middleware only, i.e. without actual database content was shown. The following example shows how such a template can be extended to also contain the definition of custom database content on-top of the MySQL DBMS software.

Example 5 - Template for deploying database content on-top of MySQL DBMS middleware

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: Template for deploying MySQL and database content.

inputs:
  # omitted here for sake of brevity

node_templates:
  my_db:
    type: tosca.nodes.Database.MySQLDatabase
    properties:
      db_name: { get_input: database_name }
      db_user: { get_input: database_user }
      db_password: { get_input: database_password }
      db_port: { get_input: database_port }
    artifacts:
      - db_content: files/my_db_content.txt
        type: tosca.artifacts.File

    requirements:
      - host: mysql

  mysql:
    type: tosca.nodes.DBMS.MySQL
    properties:
      dbms_root_password: { get_input: mysql_rootpw }
      dbms_port: { get_input: mysql_port }
    requirements:
      - host: db_server

  db_server:
    type: tosca.nodes.Compute
    properties:
      # omitted here for sake of brevity
```


In the example above, the **my_db** node template or type **tosca.nodes.Database.MySQL** represents an actual MySQL database instance managed by a MySQL DBMS installation. In its **artifacts** section, the node template points to a text file (i.e., **`my_db_content.txt`**) which can be used to help create the database content during deployment time. The **requirements** section of the **my_db** node template expresses that the database is hosted on a MySQL DBMS represented by the **mysql** node.

Note that while it would be possible to define one node type and corresponding node templates that represent both the DBMS middleware and actual database content as one entity, TOSCA distinguishes between middleware node types and application layer node types. This allows at the one hand to have better re-use of generic middleware node types without binding them to content running on top, and on the other hand this allows for better substitutability of, for example, middleware components during the deployment of TOSCA models.

7 TOSCA template for a two-tier application

The definition of multi-tier applications in TOSCA is quite similar to the example shown in section 4, with the only difference that multiple software node stacks (i.e., node templates for middleware and application layer components), typically hosted on different servers, are defined and related to each other. The example below defines a web application stack hosted on the **web_server** "compute" resource, and a database software stack similar to the one shown earlier in section 6 hosted on the **db_server** compute resource.

Example 6 - Basic two-tier application (web application and database server tiers)

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: Template for deploying a two-tier application servers on two

inputs:
  # Admin user name and password to use with the WordPress application
  wp_admin_username:
     type: string
  wp_admin_password:
     type string
  wp_db_name:
    type: string
  wp_db_user:
    type: string
  wp_db_password:
    type: string
  wp_db_port:
    type: integer
  mysql_root_password:
     type string
  mysql_port:
     type integer

node_templates:
  wordpress:
    type: tosca.nodes.WebApplication.WordPress
    properties:
      admin_user: { get_input: wp_admin_username }
      admin_password: { get_input: wp_admin_password }
      db_host: { get_property: [ db_server, ip_address ] }
    requirements:
      - host: apache
```



```
    - database_endpoint: wordpress_db
  interaces:
    Lifecycle:
      inputs:
        db_host: { get_property: [ db_server, ip_address ] }
        db_port: { get_property: [ wordpress_db, db_port ] }
        db_name: { get_property: [ wordpress_db, db_name ] }
        db_user: { get_property: [ wordpress_db, db_user ] }
        db_password: { get_property: [ wordpress_db, db_password ] }

apache:
  type: tosca.nodes.WebServer.Apache
  properties:
    # omitted here for sake of brevity
  requirements:
    - host: web_server

web_server:
  type: tosca.nodes.Compute
  properties:
    # omitted here for sake of brevity

wordpress_db:
  type: tosca.nodes.Database.MySQL
  properties:
    db_name: { get_input: wp_db_name }
    db_user: { get_input: wp_db_user }
    db_password: { get_input: wp_db_password }
    db_port: { get_input: wp_db_port }
  requirements:
    - host: mysql

mysql:
  type: tosca.nodes.DBMS.MySQL
  properties:
    dbms_root_password: { get_input: mysql_rootpw }
    dbms_port: { get_input: mysql_port }
  requirements:
    - host: db_server

db_server:
```



```
    type: tosca.nodes.Compute
    properties:
      # omitted here for sake of brevity
```

The web application stack consists of the **wordpress**, the **apache** and the **web_server** node templates. The **wordpress** node template represents a custom web application of type **tosca.nodes.WebApplication.WordPress** which is hosted on an Apache web server represented by the **apache** node template. This hosting relationship is expressed via the **host** entry in the **requirements** section of the **wordpress** node template. The **apache** node template, finally, is hosted on the **web_server** compute node.

The database stack consists of the **wordpress_db**, the **mysql** and the **db_server** node templates. The **wordpress_db** node represents a custom database of type **tosca.nodes.Database.MySQL** which is hosted on a MySQL DBMS represented by the **mysql** node template. This node, in turn, is hosted on the **db_server** compute node.

The **wordpress** node requires the **wordpress_db** node, since the WordPress application needs a database to store its data in. This relationship is established through the **database** entry in the **requirements** section of the **wordpress** node template. For configuring the WordPress web application, information about the database to connect to is required as input to the **configure** operation. Therefore, the respective input parameters (as defined for the configure operation of node type **tosca.nodes.WebApplication.WordPress** – see section 6) are mapped to properties of the **wordpress_db** node via the **get_property** function.

Note: besides the configure operation of the wordpress node template, more operations would be listed in a complete TOSCA template. Those other operations have been omitted for the sake of brevity.

8 Using a custom script to establish a relationship in a template

In previous examples, the template author did not have to think about explicit relationship types to be used to link a requirement of a node to another node of a model, nor did the template author have to think about special logic to establish those links. For example, the **host** requirement in previous examples just pointed to another node template and based on metadata in the corresponding node type definition the relationship type to be established is implicitly given.

In some cases it might be necessary to provide special processing logic to be executed when establishing relationships between nodes at runtime. For example, when connecting the WordPress application from previous examples to the MySQL database, it might be desired to apply custom configuration logic in addition to that already implemented in the application node type. In such a case, it is possible for the template author to provide a custom script as implementation for an operation to be executed at runtime as shown in the following example.

Example 7 – Providing a custom script to establish a connection

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: Template for deploying a two-tier application on two servers.

inputs:
  # omitted here for sake of brevity

node_templates:
  wordpress:
    type: tosca.nodes.WebApplication.WordPress
    properties:
      # omitted here for sake of brevity
    requirements:
      - host: apache
      - database: wordpress_db
        interfaces:
          tosca.interfaces.relationships.Configure:
            pre_configure_source: scripts/wp_db_configure.sh

  wordpress_db:
    type: tosca.nodes.Database.MySQL
    properties:
      # omitted here for the sake of brevity
    requirements:
      - host: mysql
```



```
  # other resources not shown for this example ...
```

From metadata in the node type definitions of WordPress and MySQL it is clear that a ConnectsTo relationship will be used to establish the link between the **wordpress** node and the **wordpress_db** node at runtime. The ConnectsTo relationship type (see B.4.4) defines an interface with operations that get executed when establishing the relationship. For one of those operations – **pre_configure_source** – a custom script **wp_db_configure.sh** is provided. In this example, it is assumed that this script is located at a location relative to the referencing service template, perhaps provided in some application packaging format (e.g., the TOSCA Cloud Service Archive (CSAR) format).

This approach allows for conveniently hooking in custom behavior without having to define a completely new derived relationship type.

9 Using custom relationship types in a TOSCA template

In the previous section it was shown how custom behavior can be injected by specifying scripts inline in the requirements section of node templates. When the same custom behavior is required in many templates, it does make sense to define a new relationship type that encapsulates the custom behavior in a re-usable way instead of repeating the same reference to a script (or even references to multiple scripts) in many places.

Such a custom relationship type can then be used in templates as shown in the following example.

Example 8 – A web application Node Template requiring a custom database connection type

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: Template for deploying a two-tier application on two servers.

inputs:
  # omitted here for sake of brevity

node_templates:
  wordpress:
    type: tosca.nodes.WebApplication.WordPress
    properties:
      # omitted here for sake of brevity
    requirements:
      - host: apache
      - database: wordpress_db
        relationship_type: my.types.WordpressDbConnection

  wordpress_db:
    type: tosca.nodes.Database.MySQL
    properties:
      # omitted here for the sake of brevity
    requirements:
      - host: mysql

 # other resources not shown here ...
```

In the example above, a special relationship type **`my.types.WordpressDbConnection`** is specified for establishing the link between the **`wordpress`** node and the **`wordpress_db`** node through the use of the **`relationship_type`** (keyword) attribute in the **`database`** reference. It is assumed, that this special relationship type provides some extra behavior (e.g., an operation with a script) in addition to what a

generic “connects to” relationship would provide. The definition of this custom relationship type is shown in the following section.

9.1 Definition of a custom relationship type

The following YAML snippet shows the definition of the custom relationship type used in the previous section. This type derives from the base “ConnectsTo” and overrides one operation defined by that base relationship type. For the **pre_configure_source** operation defined in the **Configure** interface of the ConnectsTo relationship type, a script implementation is provided. It is again assumed that the custom configure script is located at a location relative to the referencing service template, perhaps provided in some application packaging format (e.g., the TOSCA Cloud Service Archive (CSAR) format).

Example 9 - Defining a custom relationship type

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: Definition of custom WordpressDbConnection relationship type

relationship_types:
  my.types.WordpressDbConnection:
    derived_from: tosca.relations.ConnectsTo
    interfaces:
      Configure:
        pre_configure_source: scripts/wp_db_configure.sh
```

In the above example, the **Configure** interface is the specified alias or shorthand name for the TOSCA interface type with the full name of **tosca.interfaces.relationship.Configure** which is defined in the appendix.

10 Defining generic dependencies between nodes in a template

In some cases it can be necessary to define a generic dependency between two nodes in a template to influence orchestration behavior, i.e. to first have one node processed before another dependent node gets processed. This can be done by using the generic **dependency** requirement which is defined by the TOSCA Root Node Type and thus gets inherited by all other node types in TOSCA (see section B.6.1).

Example 10 - Simple dependency relationship between two nodes

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: Template with a generic dependency between two nodes.

inputs:
  # omitted here for sake of brevity

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

  some_service:
    type: some.type.SomeService
    properties:
      # omitted here for sake of brevity
```

As in previous examples, the relation that one node depends on another node is expressed in the **requirements** section using the **dependency** requirement that exists for all node types in TOSCA. Even if the creator of the **MyApplication** node type did not define a specific requirement for **SomeService** (similar to the **database** requirement in the example in section 8), the template author who knows that there is a timing dependency and can use the generic **dependency** requirement to express that constraint using the very same syntax as used for all other references.

11 Defining requirements on the hosting infrastructure for a software installation

Instead of defining software installations and the hosting infrastructure (the servers) in the same template, it is also possible to define only the software components of an application in a template and just express constrained requirements against the hosting infrastructure. At deployment time, the provider can then do a late binding and dynamically allocate or assign the required hosting infrastructure and place software components on top.

The following example shows how such generic hosting requirements can be expressed in the **requirements** section of node templates.

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: Template with requirements against hosting infrastructure.

inputs:
  # omitted here for sake of brevity

node_templates:
  mysql:
    type: tosca.nodes.DBMS.MySQL
    properties:
      # omitted here for sake of brevity
    requirements:
      - host: tosca.nodes.Compute
        constraints:
          - num_cpus: { in_range: { 1, 4 } }
          - mem_size: { greater_or_equal: 2 }
          - os_arch: x86_64
          - os_type: linux
          - os_distribution: ubuntu
```

In the example above, it is expressed that the **mysql** component requires a host of type **Compute**. In contrast to previous examples, there is no reference to any node template but just a specification of the type of required node. At deployment time, the provider will thus have to allocate or assign a resource of the given type.

In the **constraints** section, the characteristics of the required compute node can be narrowed down by defining boundaries for the memory size, number of CPUs, etc. Those constraints can either be expressed by means of concrete values (e.g. for the **os_arch** attribute) which will require a perfect match, or by means of qualifier functions such as **greater_or_equal**.

12 Defining requirements on a database for an application

In the same way requirements can be defined on the hosting infrastructure for an application, it is possible to express requirements against application or middleware components such as a database that is not defined in the same template. The provider may then allocate a database by any means, e.g. using a database-as-a-service solution.

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: Template with a database requirement.

inputs:
  # omitted here for sake of brevity

node_templates:
  my_app:
    type: my.types.MyApplication
    properties:
      admin_user: { get_input: admin_username }
      admin_password: { get_input: admin_password }
      db_endpoint_url: { get_ref_property: [ database, db_endpoint_url ] }
    requirements:
      - database: tosca.nodes.DBMS.MySQL
        constraints:
          - mysql_version: { greater_or_equal: 5.5 }
```

In the example above, the application **my_app** needs a MySQL database, where the version of MySQL must be 5.5 or higher. The example shows an additional feature of referencing a property of the database to get the database connection endpoint URL at runtime via the **get_ref_property** intrinsic function. In contrast to the **get_property** function used in earlier examples, which assumes that a node template in the same service template is referenced, the **get_ref_property** function allows for getting a property via a reference expressed in the **requirements** section. The first argument is the name of a reference – **database** in the example above – and the second argument is the name of the property of the referenced node, which must be defined by the respective node type **tosca.types.nodes.MySQLDatabase**.

13 Grouping node templates

In designing applications composed of several interdependent software components (or nodes) it is often desirable to manage these components as a named group. This can provide an effective way of associating policies (e.g., scaling, placement, security or other) that orchestration tools can apply to all the components of group during deployment or during other lifecycle stages.

In many realistic scenarios it is desirable to include scaling capabilities into an application to be able to react on load variations at runtime. The example below shows the definition of a scaling web server stack, where a variable number of servers with apache installed on them can exist, depending on the load on the servers.

Example 11 - Grouping Node Templates with same scaling policy

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: Template for a scaling web server.

inputs:
  # omitted here for sake of brevity

node_templates:
  apache:
    type: tosca.types.nodes.ApacheWebserver
    properties:
      http_port: 8080
      https_port: 8443
    requirements:
      - host: server

  server:
    type: tosca.nodes.Compute
    properties:
      # omitted here for sake of brevity

group:
  webserver_group:
    members: [ apache, server ]
    policies:
      - my_scaling_policy:
          # Specific policy definitions are considered domain specific and
          # are not included here
```


The example first of all uses the concept of grouping to express which components (node templates) need to be scaled as a unit – i.e. the compute nodes and the software on-top of each compute node. This is done by defining the **webserver_group** in the **groups** section of the template and by adding both the **apache** node template and the **server** node template as a member to the group.

Furthermore, a scaling policy is defined for the group to express that the group as a whole (i.e. pairs of **server** node and the **apache** component installed on top) should scale up or down under certain conditions.

In cases where no explicit binding between software components and their hosting compute resources is defined in a template, but only requirements are defined as has been shown in section 11, a provider could decide to place software components on the same host if their hosting requirements match, or to place them onto different hosts.

It is often desired, though, to influence placement at deployment time to make sure components get collocation or anti-collocated. This can be expressed via grouping and policies as shown in the example below.

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: Template hosting requirements and placement policy.

inputs:
  # omitted here for sake of brevity

node_templates:
  wordpress:
    type: tosca.types.nodes.Wordpress
    properties:
      # omitted here for sake of brevity
    requirements:
      - host: tosca.nodes.Compute
        constraints:
          mem_size: { greater_or_equal: 2 }
          os_arch: x86_64
          os_type: linux

  mysql:
    type: tosca.types.nodes.MySQL
    properties:
      # omitted here for sake of brevity
    requirements:
      - host: tosca.nodes.Compute
        constraints:
          disk_size: { greater_or_equal: 10 }
```



```
          arch: x86_64
          os_type: linux

groups:
  my_collocation_group:
    members: [ wordpress, mysql ]
    policies:
      - my_anti_collocation_policy:
           # Specific policy definitions are considered domain specific and
           # are not included here
```

In the example above, both software components **wordpress** and **mysql** have identical hosting requirements. Therefore, a provider could decide to put both on the same server. By defining a group of the two components and attaching an anti-collocation policy to the group it can be made sure, though, that both components are put onto different hosts at deployment time.

Appendix A. TOSCA Simple Profile definitions in YAML

This section describes all of the YAML block structure for all keys and mappings that are defined for the TOSCA Version 1.0 Simple Profile specification that are needed to describe a TOSCA Service Template (in YAML).

A.1 TOSCA namespace and alias

The following table defines the namespace alias and (target) namespace values that SHALL be used when referencing the TOSCA Simple Profile version 1.0 specification.

Alias	Target Namespace	Specification Description
tosca_simple_yaml_1_0	http://docs.oasis-open.org/tosca/ns/simple/yaml/1.0	The TOSCA Simple Profile v1.0 (YAML) target namespace and namespace alias.

A.2 Parameter and property types

This clause describes the primitive types that are used for declaring normative properties, parameters and grammar elements throughout this specification.

A.2.1 Referenced YAML Types

Many of the types we use in this profile are built-in types from the YAML 1.2 specification (i.e., tag:yaml.org,2002).

The following table declares the valid YAML type URIs and aliases that SHALL be used when possible when defining parameters or properties within TOSCA Service Templates using this specification:

Valid aliases	Type URI
string	tag:yaml.org,2002:str (default)
integer	tag:yaml.org,2002:int
float	tag:yaml.org,2002:float
boolean	tag:yaml.org,2002:bool
timestamp	tag:yaml.org,2002:timestamp
null	tag:yaml.org,2002:null

A.2.1.1 Notes

- The “string” type is the default type when not specified on a parameter or property declaration.
- While YAML supports further type aliases, such as “str” for “string”, the TOSCA Simple Profile specification promotes the fully expressed alias name for clarity.

A.2.2 TOSCA Types

This specification defines the following types that may be used when defining properties or parameters.

A.2.3 version

TOSCA supports the concept of "reuse" of type definitions, as well as template definitions which could be version and change over time. It is important to provide a reliable, normative means to represent a version string which enables the comparison and management of types and templates over time. Therefore, the TOSCA TC intends to provide a normative version type (string) for this purpose in future Working Drafts of this specification.

A.3 TOSCA Entity and element definitions (meta-model)

This section defines all modelable entities that comprise the TOSCA Version 1.0 Simple Profile specification along with their key names, grammar and requirements.

A.3.1 Description element

This optional element provides a means include single or multiline descriptions within a TOSCA Simple Profile template as a scalar string value.

A.3.1.1 Keyname

The following keyname is used to provide a description within the TOSCA Simple Profile specification:

```
description
```

A.3.1.2 Grammar

The description element is a YAML string.

```
description: <string>
```

A.3.1.3 Examples

Simple descriptions are treated as a single literal that includes the entire contents of the line that immediately follows the **`description`** key:

```
description: This is an example of a single line description (no folding).
```

The YAML "folded" style may also be used for multi-line descriptions which "folds" line breaks as space characters.

```
description: >
  This is an example of a multi-line description using YAML. It permits for line
  breaks for easier readability...

  if needed.  However, (multiple) line breaks are folded into a single space
  character when processed into a single string value.
```

A.3.1.4 Notes

- Use of "folded" style is discouraged for the YAML string type apart from when used with the **`description`** keyname.

A.3.2 Constraint clause

A constraint clause defines an operation along with one or more compatible values that can be used to define a constraint on a property or parameter's allowed values when it is defined in a TOSCA Service Template or one of its entities.

A.3.2.1 Operator keynames

The following is the list of recognized operators (keynames) when defining constraint clauses:

Operator	Type	Value Type	Description
equal	scalar	any	Constrains a property or parameter to a value equal to ('=') the value declared.
greater_than	scalar	comparable	Constrains a property or parameter to a value greater than ('>') the value declared.
greater_or_equal	scalar	comparable	Constrains a property or parameter to a value greater than or equal to ('>=') the value declared.
less_than	scalar	comparable	Constrains a property or parameter to a value less than ('<') the value declared.
less_or_equal	scalar	comparable	Constrains a property or parameter to a value less than or equal to ('<=') the value declared.
in_range	dual scalar	comparable	Constrains a property or parameter to a value in range of (inclusive) the two values declared.
valid_values	list	any	Constrains a property or parameter to a value that is in the list of declared values.
length	scalar	string	Constrains the property or parameter to a value of a given length.
min_length	scalar	string	Constrains the property or parameter to a value to a minimum length.
max_length	scalar	string	Constrains the property or parameter to a value to a maximum length.
pattern	regex	string	Constrains the property or parameter to a value that is allowed by the provided regular expression. **Note**: Future drafts of this specification will detail the use of regular expressions and reference an appropriate standardized grammar.

In the Value Type column above, an entry of "comparable" includes integer, float, timestamp, string and version types, while an entry of "any" refers to any type allowed in the TOSCA simple profile in YAML.

A.3.2.2 Grammar

Constraint clauses take one of the following forms:

```
# Scalar grammar
<operator>: <scalar_value>

# Dual scalar grammar
<operator>: { <scalar_value_1>, <scalar_value_2> }

# List grammar
```



```
<operator> [ <value_1>, <value_2>, ..., <value_n> ]

# Regular expression (regex) grammar
pattern: <regular_expression_value>
```

In the above definition, the pseudo values that appear in angle brackets have the following meaning:

- **operator**: represents a required operator from the specified list shown above (see section A.3.2.1 “Operator keynames”).
- **scalar_value, scalar_value_x**: represents a required scalar (or atomic quantity) that can hold only one value at a time. This will be a value of a primitive type, such as an integer or string that is allowed by this specification.
- **value_x**: represents a required value of the operator that is not limited to scalars.
- **regular_expression_value**: represents a regular expression (string) value.

A.3.2.3 Examples

Constraint clauses used on parameter or property definitions:

```
# equal
equal: 2

# greater_than
greater_than: 1

# greater_or_equal
greater_or_equal: 2

# less_than
less_than: 5

# less_or_equal
less_or_equal: 4

# in_range
in_range: { 1, 4 }

# valid_values
valid_values: [1, 2, 4]

# specific length (in characters)
length: 32

# min_length (in characters)
min_length: 8
```



```
# max_length (in characters)
max_length: 64
```

A.3.2.4 Notes

- Values provided by the operands (i.e., values and scalar values) SHALL be type-compatible with their associated operations.
- Future drafts of this specification will detail the use of regular expressions and reference an appropriate standardized grammar.

A.3.3 Constraints element

The Constraints element specifies a sequenced list of constraints on one or more of the Service Template's properties, parameters or other typed elements of the TOSCA Simple Profile. A constraints element is represented as a YAML block collection that contains a sequenced list of nested constraint clauses.

A.3.3.1 Keyname

The following keyname is used to provide a list of constraints within the TOSCA Simple Profile specification:

```
constraints
```

A.3.3.2 Grammar

The constraints element is described as a YAML block collection that contains a sequence of constraint clauses:

```
<some_typed_property>:
  constraints:
    - <constraint_clause_1>
    - ...
    - <constraint_clause_n>
```

In the above definition, the pseudo values that appear in angle brackets have the following meaning:

- **`some_typed_property`**: represents the name of a typed property definition, as a string, which can be associated to a TOSCA entity.
  - For example, a property (definition) can be declared as part of a Node Type or Node Template definition or it can be used to define an input or output property (parameter) for a Service Template's.
- **`constraint_clause_x`**: represents constraint clauses for the associated property or parameter.

A.3.3.3 Examples

Constraint on an integer-typed parameter definition:

```
# An example input parameter that represents a number of CPUs
```



```
# and constrains its value to a specific range.
inputs:
  num_cpus:
    type: integer
    constraints:
      - in_range: { 2, 4 }
```

Constraints on a string-typed parameter definition:

```
# An example input parameter that represents a user ID and constrains its length.
inputs:
  user_id:
    type: string
    constraints:
      - min_length: 8
      - max_length: 16
```

A.3.3.4 Notes

- Constraints of properties or parameters SHOULD be type-compatible with the type defined for that property or parameter.
- In the TOSCA v1.0 specification constraints are expressed in the XML Schema definitions of Node Type properties referenced in the **PropertiesDefinition** element of **NodeType** definitions.

A.3.4 Operation definition

An operation definition defines a named function or procedure that can be bound to an implementation artifact (e.g., a script).

A.3.4.1 Keynames

The following is the list of recognized keynames recognized for a TOSCA operation definition:

Keyname	Type	Description
description	description	The optional description string for the associated named operation.
implementation	string	The optional implementation artifact name (e.g., a script file name within a TOSCA CSAR file).

A.3.4.2 Grammar

The full grammar for expressing an operation is as follows:

```
<operation_name>:
   description: <operation_description>
   implementation: <implementation_artifact_name>
```

In addition, the following simplified grammar may also be used (where a full definition is not necessary):


```
<operation_name>: <implementation_artifact_name>
```

In the above definitions, the pseudo values that appear in angle brackets have the following meaning:

- **operation_name**: represents the required name of the operation as a string.
- **operation_description**: represents the optional description string for the corresponding **operation_name**.
- **implementation_artifact_name**: represents the name (string) of artifact definition (defined elsewhere), or the direct name of an implementation artifact's relative filename (e.g., a service template-relative, path-inclusive filename or absolute file location using a URL).

A.3.4.3 Notes

- Implementation artifact file names (e.g., script filenames) may include file directory path names that are relative to the TOSCA service template file itself when packaged within a TOSCA Cloud Service ARchive (CSAR) file.

A.3.5 Artifact definition

An artifact definition defines a named, typed file that can be associated with Node Type or Node Template and used by orchestration engine to facilitate deployment and implementation of interface operations.

A.3.5.1 Keynames

The following is the list of recognized keynames recognized for a TOSCA property definition:

Keyname	Type	Description
type	string	The optional data type for the artifact definition.
description	description	The optional description for the artifact definition.
mime_type	string	The optional Mime type for finding the correct artifact definition when it is not clear from the file extension.

A.3.5.2 Grammar

Named artifact definitions have the following grammar:

```
# Simple form
<artifact_name>: <artifact_file_URI>

# Full form
<artifact_name>: <artifact_file_URI>
type: <artifact_type_name>
description: <artifact_description>
mime_type: <artifact_mime_type_name>
```

In the above definition, the pseudo values that appear in angle brackets have the following meaning:

- **artifact_name**: represents the required name of the artifact definition as a string.
- **artifact_file_URI**: represents the required URI string (relative or absolute) which can be used to locate the artifact's file.
- **artifact_type_name**: represents the required artifact type the artifact definition is based upon.
- **artifact_description**: represents the optional description string for the corresponding **artifact_name**.
- **artifact_mime_type_name**: represents the optional, explicit Mime Type (as a string) for the associated artifact definition when it is not clear from the file description.

A.3.5.3 Example

The following represents an artifact definition:

```
my_file_artifact: ../my_apps_files/operation_artifact.txt
```

A.3.6 Artifacts element

The Artifacts element is used to associate one or more typed artifact definitions with a TOSCA Node Type or Node Template.

A.3.6.1 Keynames

The following keyname is used to declare a list of requirements within the TOSCA Simple Profile specification:

```
artifacts
```

A.3.6.2 Grammar

The requirements element is described by a YAML block collection that contains a *sequenced* list of artifact definitions:

```
<some_typed_entity_name>:
  artifacts:
    - <artifact_definition_1>
    - ...
    - <artifact_definition_n>
```

In the above definition, the pseudo values that appear in angle brackets have the following meaning:

- **some_typed_entity_name:** represents the name (string) of a typed TOSCA entity (e.g., a Node Type, Node Template) that has, as part of its definition, a list of artifacts.
- **artifact_definition_x:** represents one or more Artifact definitions for the associated entity.

A.3.6.3 Examples

The following examples show capability definitions in both simple and full forms being associated to Node Types:


```
my_node_type_1:
  # Other keys omitted here for sake of brevity
  capabilities:
    app_container: mytypes.mycapabilities.AppContainer
    app_endpoint:
      type: mytypes.mycapabilities.AppEndpoint
      properties:
        timeout: 300
```

A.3.7 Interface definition

An interface definition defines a named interface that can be associated with a Node or Relationship Type

A.3.7.1 Keynames

The following is the list of recognized keynames recognized for a TOSCA interface definition:

Keyname	Type	Description
None	N/A	N/A

A.3.7.2 Grammar

The following keyname is used to provide a list of properties within the TOSCA Simple Profile specification:

```
<interface_definition_name>:
  <operation_definition_1>
  ...
  <operation_definition_n>
```

In the above definition, the pseudo values that appear in angle brackets have the following meaning:

- **`interface_definition_name:`** represents the required name of the interface definition as a string.
- **`operation_definition_x:`** represents the required name of one or more operation definitions.

A.3.7.3 Examples

```
mycompany.mytypes.myinterfaces.MyConfigure:
  configure_service_A:
    description: My application’s custom configuration interface for service A.
  configure_service_B:
    description: My application’s custom configuration interface for service B.
```


A.3.8 Interfaces element

The Interfaces element describes a list of one or more interface definitions for a modelable entity (e.g., a Node or Relationship Type) as defined within the TOSCA Simple Profile specification. Each interface definition contains one or more interfaces for operations that can be invoked on the associated entity.

A.3.8.1 Keyname

The following keyname is used to declare a list of interfaces definitions within the TOSCA Simple Profile specification:

```
interfaces
```

A.3.8.2 Grammar

```
interfaces: [ <interface_defn_name_1>, ..., <interface_defn_name_n> ]
```

In the above definition, the pseudo values that appear in angle brackets have the following meaning:

- **`interface_defn_name_x`**: represents one or more names of valid TOSCA interface definitions.

A.3.8.3 Example

```
interfaces: [ mytypes.myinterfaces.myLifecycleOperationsDefn ]
```

A.3.9 Property definition

A property definition defines a named, typed value and related data that can be associated with an entity defined in this specification. It is used to associate a transparent property or characteristic of that entity which can either be set on or retrieved from it.

A.3.9.1 Keynames

The following is the list of recognized keynames recognized for a TOSCA property definition:

Keyname	Type	Description
type	string	The required data type for the property.
description	description	The optional description for the property.
required	boolean	An optional key that declares a property as required (true) or not (false). If this key is not declared for property definition, then the property SHALL be considered required by default.
default	Any	An optional key that may provide a value to be used as a default if not provided by another means. This value SHALL be type compatible with the type declared by the property definition's **`type`** keyname.
constraints	constraints	The optional list of sequenced constraints for the property.

A.3.9.2 Grammar

Named property definitions have the following grammar:

```
<property_name>:
  type: <property_type>
  required: <property_required>
  default: <default_value>
  description: <property_description>
  constraints:
    <property_constraints>
```

In the above definition, the pseudo values that appear in angle brackets have the following meaning:

- **property_name**: represents the required name of the property as a string.
- **property_type**: represents the required data type of the property.
- **property_required**: represents an optional boolean value (true or false) indicating whether or not the property is required. If this keyname is not present on a property definition, then the property SHALL be considered required (i.e., true) by default.
- **default_value**: contains a type-compatible value that may be used as a default if not provided by another means.
- **property_description**: represents the optional description of the property
- **property_constraints**: represents the optional sequenced list of one or more constraint clauses (as shown in the constraints element) on the property definition.

A.3.9.3 Example

The following represents a required property definition:

```
num_cpus:
  type: integer
  description: Number of CPUs for a Compute (server) instance.
  default: 1
  constraints:
    - valid_values: [ 1, 2, 4, 8 ]
```

A.3.9.4 Notes

- This element directly maps to the **PropertiesDefinition** element defined as part of the schema for most type and entities defined in the TOSCA v1.0 specification.

A.3.10 Properties element

The Properties element describes one or more typed properties that can be associated with a modelable TOSCA entity (e.g., Node Types, Node Templates, Artifact Types, etc.).

A.3.10.1 Keyname

The following keyname is used to declare a list of properties within the TOSCA Simple Profile specification:


```
properties
```

A.3.10.2 Grammar

The properties element is described as a YAML block collection that contains a list of property definitions:

```
<some_typed_entity_name>:
  properties:
    <property_defn_1>
    ...
    <property_defn_n>
```

In the above definition, the pseudo values that appear in angle brackets have the following meaning:

- **some_typed_entity_name**: represents the name of a typed TOSCA entity (e.g., a Node Type, Node Template, Relationship Type, etc.) that has, as part of its definition, a list of properties.
- **property_defn_x**: represents one or more property definitions for the associated entity.

A.3.10.3 Examples

The following example shows property definitions being associated to a Node Type:

```
my_app_node_type:
  derived_from: tosca.nodes.Root
  properties:
    stylesheet: elegant.css
      type: string
      default: basic.css
    max_connections: 100
      type: integer
      required: no
```

A.3.11 Capability definition

A capability definition defines a named, typed set of data that can be associated with Node Type or Node Template to describe a transparent capability or feature of the software component the node describes.

A.3.11.1 Keynames

The following is the list of recognized keynames recognized for a TOSCA capability definition:

Keyname	Type	Description
type	string	The required name of the Capability Type the capability definition is based upon.
properties	properties	An optional list of property definitions for the capability definition.

A.3.11.2 Grammar

Named capability definitions have one of the following grammars:

```
# Simple definition is as follows:
<capability_defn_name>: <capability_type>

# The full definition is as follows:
<capability_defn_name>:
   type: <capability_type>
   properties:
     <property_definitions>
```

In the above definition, the pseudo values that appear in angle brackets have the following meaning:

- **capability_defn_name:** represents the name of a capability definition as a string.
- **capability_type**: represents the required capability type the capability definition is based upon.
- **property_definitions**: represents the optional list of property definitions for the capability definition.

A.3.11.3 Example

The following examples show capability definitions in both simple and full forms:

```
# Simple form, no properties defined or augmented
app_container: mytypes.mycapabilities.MyAppContainer

# Full form, augmenting properties of the referenced capability type
app_container:
  type: mytypes.mycapabilities.MyAppContainer
  my_containee_types: [ mytypes.mynodes.myAppType ]
```

A.3.11.4 Notes

- This definition directly maps to the **CapabilitiesDefinition** of the Node Type entity as defined in the TOSCA v1.0 specification.

A.3.12 Capabilities element

The Capabilities element is used to associate one or more typed capabilities definitions with a TOSCA Node Type or Node Template.

A.3.12.1 Keyname

The following keyname is used to declare a list of capabilities within the TOSCA Simple Profile specification:

```
capabilities
```

A.3.12.2 Grammar

The capabilities element is described by a YAML block collection that contains a list of capability definitions:

```
<some_typed_entity_name>:
  capabilities:
    <capability_definition_1>
    ...
    <capability_definition_n>
```

In the above definition, the pseudo values that appear in angle brackets have the following meaning:

- **some_typed_entity_name:** represents the name of a typed TOSCA entity (e.g., a Node Type, Node Template) that has, as part of its definition, a list of capabilities.
- **capability_definition_x:** represents one or more Capability definitions for the associated entity.

A.3.12.3 Examples

The following examples show capability definitions in both simple and full forms being associated to Node Types:

```
my_node_type_1:
  # Other keys omitted here for sake of brevity
  capabilities:
    app_container: mytypes.mycapabilities.AppContainer
    app_endpoint:
      type: mytypes.mycapabilities.AppEndpoint
      properties:
        timeout: 300
```

A.3.12.4 Notes

- This element directly maps to the **Capabilities** element defined as part of the schema for the Node Template entity as defined in the TOSCA v1.0 specification.
- The TOSCA Root node type provides a generic named **Feature** capability (i.e., **tosca.capabilities.Feature**) called “feature” that nodes that derive from it may readily extend to export a significant capability the node supplies.

A.3.13 Requirements element

The Requirements element describes one or more typed requirements (dependencies) of a modelable entity (e.g., Node Types, Node Templates, Artifact Types, etc.) defined within the TOSCA Simple Profile specification. A requirements element is represented as a YAML block collection that contains a sequenced list of nested requirement definitions.

A.3.13.1 Keynames

The following keyname is used to declare a list of requirements within the TOSCA Simple Profile specification:

```
requirements
```

The following is the list of recognized keynames recognized for a TOSCA requirement definition:

Keyname	Type	Description
relationship_type	string	The optional reserved keyname used to provide a named relationship to use when fulfilling the associated named requirement.

A.3.13.2 Grammar

The requirements element is described by a YAML block collection that contains a *sequenced* list of requirement definitions:

```
<some_typed_entity_name>:
  requirements:
    - <requirement_definition_1>
    - ...
    - <requirement_definition_n>
```

Where each named requirement definition has one of the following forms:

```
# Requirement for a specific named entity (e.g., a Node Type or Node Template)
- <requirement_name>: <entity_name>

# Requirement clause for a specific named Capability Type
- <requirement_name>: <capability_type_name>

# Requirement for a node type with an optional, explicit Relationship type
- <requirement_name>: <node_name>
  relationship_type: <relationship_name>
```

In the above definition, the pseudo values that appear in angle brackets have the following meaning:

- **`some_typed_entity_name:`** represents the name (a string) of a typed TOSCA entity (e.g., a Node Type, Node Template) that has, as part of its definition, a sequenced list of requirements.
- **`requirement_name:`** represents the name of a requirement definition as a string.
- **`capability_type_name`**: represents the name of a capability type (exported by a Node Type or Template) that the requirement would be fulfilled by.
- **`node_name`**: represents the name of a Node Type or Node Template as a string.
- **`relationship_name`**: represents the name of an explicit, relationship type or definition to be used when relating the node the requirement appears in to another node.

A.3.13.3 Example

A web application requires hosting (with the named relationship of '**host'**) on a web server that is defined elsewhere within the Service Template as a node template with the name '**my_web_server'**. Similarly, the web application requires a connection to a database (using the named relationship '**database**') to another node template named '**my_database**'. However, the connection between the web application and the database further requires a custom relationship designated by the keyword '**relationship_type**' and having the custom relationship type definition name of '**my.types.CustomDbConnection**'.

```
# Example of a requirement that can be fulfilled by any web server node type
my_webapp_node_template:
  requirements:
    - host: tosca.nodes.WebServer

# Example of a requirement that is fulfilled by a feature (exported by a Node Type)
my_webapp_node_template:
  requirements:
    - database: tosca.capabilities.DatabaseEndpoint

# Example of a (database) requirement that is fulfilled by a node template named
# "my_database", but also requires a custom database connection relationship
my_webapp_node_template:
  requirements:
    - database: my_database
      relationship_type: my.types.CustomDbConnection
```

A.3.13.4 Notes

- This element directly maps to the **Requirements** element defined as part of the schema for the Node Templates entity (as part of a Service Template's Topology Template), as well as the matching **RequirementsDefinition** of the Node Type entity as defined in the TOSCA v1.0 specification.

A.3.14 Artifact Type

An Artifact Type is a reusable entity that defines the type of one or more files which Node Types or Node Templates can have dependent relationships and used during operations such as during installation or deployment.

A.3.14.1 Keynames

The following is the list of recognized keynames recognized for a TOSCA Artifact Type definition:

Keyname	Definition/Type	Description
derived_from	string	An optional parent Artifact Type name the Artifact Type derives from.

Keyname	Definition/Type	Description
description	description	An optional description for the Artifact Type.
mime_type	string	The required mime type property for the Artifact Type.
file_ext	string[]	The required file extension property for the Artifact Type.
properties	properties	An optional list of property definitions for the Artifact Type.

A.3.14.2 Grammar

```
<artifact_type_name>:
  derived_from: <parent_artifact_type_name>
  description: <artifact_description>
  mime_type: <mime_type_string>
  file_ext: [ <file_extension_1>, ..., <file_extension_n> ]
  properties:
    <property_definitions>
```

In the above definition, the pseudo values that appear in angle brackets have the following meaning:

- **artifact_type_name**: represents the name of the Artifact Type being declared as a string.
- **parent_artifact_type_name**: represents the name of the Artifact Type this Artifact Type definition derives from (i.e., its "parent" type).
- **artifact_description**: represents the optional description string for the corresponding **artifact_type_name**.
- **mime_type_string**: represents the Multipurpose Internet Mail Extensions (MIME) standard string value that describes the file contents for this type of artifact as a string.
- **file_extension_x**: represents one or more recognized file extensions for this type of artifact as strings.
- **property_definitions**: represents the optional list of property definitions for the artifact type.

A.3.14.3 Examples

```
my_artifact_type:
  description: Java Archive artifact type
  derived_from: tosca.artifact.Root
  mime_type: application/java-archive
  file_ext: [ jar ]
```

A.3.15 Capability Type

A Capability Type is a reusable entity that describes a kind of capability that a Node Type can declare to expose. Requirements (implicit or explicit) that are declared as part of one node can be matched to (i.e., fulfilled by) the Capabilities declared by other node.

577 The following is the list of recognized keynames recognized for a TOSCA Capability Type definition:

Keyname	Definition/Type	Description
derived_from	string	An optional parent capability type name this new capability type derives from.
description	description	An optional description for the capability type.
properties	properties	An optional list of property definitions for the capability type.

578 ### A.3.15.1 Grammar

```
<capability_type_name>:
  derived_from: <parent_capability_type_name>
  description: <capability_description>
  properties:
    <property_definitions>
```

579 In the above definition, the pseudo values that appear in angle brackets have the following meaning:

580 • **capability_type_name**: represents the name of the Capability Type being declared as a string.
581 • **parent_capability_type_name**: represents the name of the Capability Type this Capability
582 Type definition derives from (i.e., its "parent" type).
583 • **capability_description**: represents the optional description string for the corresponding
584 **capability_type_name**.
585 • **property_definitions**: represents an optional list of property definitions that the capability
586 type exports.

587 ### A.3.15.2 Example

```
mycompany.mytypes.myapplication.MyFeature:
  derived_from: tosca.capabilities.Feature
  description: a custom feature of my company's application
  properties:
    my_feature_version:
      type: string
    my_feature_value:
      type: integer
```

588 ## A.3.16 Requirement Type

589 A Requirement Type is a reusable entity that describes a kind of requirement that a Node Type can
590 declare to expose. The TOSCA Simple Profile seeks to simplify the need for declaring specific
591 Requirement Types from nodes and instead rely upon nodes declaring their features sets using TOSCA
592 Capability Types along with a named Feature notation.

593 Currently, there are no use cases in this TOSCA Simple Profile in YAML specification that utilize an
594 independently defined Requirement Type. This is a desired effect as part of the simplification of the
595 TOSCA v1.0 specification.

A.3.17 Relationship Type

A Relationship Type is a reusable entity that defines the type of one or more relationships between Node Types or Node Templates.

A.3.17.1 Keynames

The following is the list of recognized keynames recognized for a TOSCA Relationship Type definition:

Keyname	Definition/Type	Description
derived_from	string	An optional parent Relationship Type name the Relationship Type derives from.
description	description	An optional description for the Relationship Type.
properties	properties	An optional list of property definitions for the Relationship Type.
interfaces	interfaces	An optional list of named interfaces for the Relationship Type.
valid_targets	string[]	A required list of one or more valid target entities or entity types (i.e., a Node Types or Capability Types)

A.3.17.2 Grammar

```
<relationship_type_name>:
  derived_from: <parent_relationship_type_name>
  description: <relationship_description>
  properties:
    <property_definitions>
  interfaces: <interface_definitions>
  valid_targets: [ <entity_name_or_type_1>, ..., <entity_name_or_type_n> ]
```

In the above definition, the pseudo values that appear in angle brackets have the following meaning:

- **relationship_type_name**: represents the name of the Relationship Type being declared as a string.
- **parent_relationship_type_name**: represents the name (string) of the Relationship Type this Relationship Type definition derives from (i.e., its “parent” type).
- **relationship_description**: represents the optional description string for the corresponding **relationship_type_name**.
- **property_definitions**: represents the optional list of property definitions for the Relationship Type.
- **interface_definitions**: represents the optional list of one or more named interface definitions supported by the Relationship Type.
- **entity_name_or_type_x**: represents one or more valid target (types) for the relationship (e.g., Node Types, Capability Types, etc.).

A.3.17.3 Best Practices

- The TOSCA Root relationship type (tosca.relationships.Root) provides a standard configuration interface (tosca.interfaces.relationship.Configure) that SHOULD be used where possible when defining new relationships types.

A.3.17.4 Examples

```
mycompanytypes.myrelationships.AppDependency:
  derived_from: tosca.relationships.DependsOn
  valid_targets: [ mycompanytypes.mycapabilities.SomeAppCapability ]
```

A.3.18 Node Type

A Node Type is a reusable entity that defines the type of one or more Node Templates. As such, a Node Type defines the structure of observable properties via a *Properties Definition, the Requirements and Capabilities of the node as well as its supported interfaces.*

The following is the list of recognized keynames recognized for a TOSCA Node Type definition:

Keyname	Definition/Type	Description
derived_from	string	An optional parent Node Type name this new Node Type derives from.
description	description	An optional description for the Node Type.
properties	properties	An optional list of property definitions for the Node Type.
requirements	requirements	An optional *sequenced* list of requirement definitions for the Node Type.
capabilities	capabilities	An optional list of capability definitions for the Node Type.
interfaces	interfaces	An optional list of named interfaces for the Node Type.
artifacts	artifacts	An optional *sequenced* list of named artifact definitions for the Node Type/

A.3.18.1 Grammar

```
<node_type_name>:
  derived_from: <parent_node_type_name>
  description: <node_type_description>
  properties:
    <property_definitions>
  requirements:
    <requirement_definitions>
  capabilities:
    <capability_definitions>
  interfaces: <interface_definitions>
  artifacts:
    <artifact_definitions>
```


In the above definition, the pseudo values that appear in angle brackets have the following meaning:

- **node_type_name**: represents the name of the Node Type being declared.
- **parent_node_type_name**: represents the name (string) of the Node Type this Node Type definition derives from (i.e., its "parent" type).
- **node_type_description**: represents the optional description string for the corresponding **node_type_name**.
- **property_definitions**: represents the optional list of property definitions for the Node Type.
- **requirement_definitions**: represents the optional *sequenced* list of requirement definitions for the Node Type.
- **capability_definitions**: represents the optional list of capability definitions for the Node Type.
- **interface_definitions**: represents the optional list of one or more named interface definitions supported by the Node Type.
- **artifact_definitions**: represents the optional list of artifact definitions for the Node Template that augment those provided by its declared Node Type.

A.3.18.2 Best Practices

- It is recommended that all Node Types SHOULD derive directly (as a parent) or indirectly (as an ancestor) of the TOSCA "Root" Node Type (i.e., **tosca.nodes.Root**) to promote compatibility and portability. However, it is permitted to author Node Types that do not do so.

A.3.18.3 Example

```
my_company.my_types.my_app_node_type:
  derived_from: tosca.nodes.SoftwareComponent
  description: My company's custom applicaton
  properties:
    my_app_password:
      type: string
      description: application password
      constraints:
        - length: { min: 6, max: 10 }
    my_app_port:
      type: number
      description: application port number
  requirements:
    host: tosca.nodes.Compute
  interfaces: [ Lifecycle ]
```

A.3.19 Node Template

A Node Template specifies the occurrence of a manageable software component as part of an application's topology model which is defined in a TOSCA Service Template. Node template is an

instance of a specified Node Type and can provide customized properties, constraints or operations which override the defaults provided by its Node Type and its implementations.

The following is the list of recognized keynames recognized for a TOSCA Node Template definition:

Keyname	Definition/Type	Description
type	string	The required name of the Node Type the Node Template is based upon.
description	description	An optional description for the Node Template.
properties	properties	An optional list of property definitions for the Node Template.
requirements	requirements	An optional *sequenced* list of requirement definitions for the Node Template.
capabilities	capabilities	An optional list of capability definitions for the Node Template.
interfaces	interfaces	An optional list of named interfaces for the Node Template.
artifacts	artifacts	An optional *sequenced* list of named artifact definitions for the Node Template.

A.3.19.1 Grammar

```
<node_template_name>:
  type: <node_type_name>
  description: <node_template_description>
  properties:
    <property_definitions>
  requirements:
    <requirement_definitions>
  capabilities:
    <capability_definitions>
  interfaces:
    <interface_definitions>
  artifacts:
    <artifact_definitions>
```

In the above definition, the pseudo values that appear in angle brackets have the following meaning:

- **node_template_name**: represents the name of the Node Template being declared.
- **node_type_name**: represents the name of the Node Type this Node Template is based upon.
- **node_template_description**: represents the optional description string for the corresponding **node_template_name**.
- **property_definitons**: represents the optional list of property definitions for the Node Template that augment those provided by its declared Node Type.
- **requirement_definitions**: represents the optional *sequenced* list of requirement definitions for the Node Template that augment those provided by its declared Node Type.
- **capability_definitions**: represents the optional list of capability definitions for the Node Template that augment those provided by its declared Node Type.
- **interface_definitions**: represents the optional list of interface definitions for the Node Template that augment those provided by its declared Node Type.

667 • **artifact_definitions**: represents the optional list of artifact definitions for the Node
668 Template that augment those provided by its declared Node Type.

669 ### A.3.19.2 Example

```
mysql:
  type: tosca.nodes.DBMS.MySQL
  properties:
    dbms_password: { get_input: my_mysql_rootpw }
    dbms_port: { get_input: my_mysql_port }
  requirements:
    - host: db_server
  interfaces:
    Lifecycle:
      configure: scripts/my_own_configure.sh
```

670 ## A.4 Service Template

671 A TOSCA Definitions YAML document contains element definitions of building blocks for cloud
672 application, or complete models of cloud applications.

673 This section describes the top-level structural elements (i.e., YAML keys) which are allowed to appear in
674 a TOSCA Definitions YAML document.

675 ### A.4.1 Keynames

676 A TOSCA Definitions file contains the following element keynames:

Keyname	Required	Description
tosca_definitions_version	yes	Defines the version of the TOSCA Simple Profile specification the template (grammar) complies with.
tosca_default_namespace	no	Defines the namespace of the TOSCA schema to use for validation.
template_name	no	Declares the name of the template.
template_author	no	Declares the author(s) of the template.
template_version	no	Declares the version string for the template.
description	no	Declares a description for this Service Template and its contents.
imports	no	Declares import statements external TOSCA Definitions documents (files).
inputs	no	Defines a set of global input parameters passed to the template when its instantiated. This provides a means for template authors to provide points of variability to users of the template in order to customize each instance within certain constraints.
node_templates	no	Defines a list of Node Templates that model the components of an application or service.
node_types	no	This section contains a set of node type definitions for use in service templates. Such type definitions may be used within the node_templates section of the same file, or a TOSCA Definitions file may also just contain node type definitions for use in other files.

Keyname	Required	Description
relationship_types	no	This section contains a set of relationship type definitions for use in service templates. Such type definitions may be used within the same file, or a TOSCA Definitions file may also just contain relationship type definitions for use in other files.
capability_types	no	This section contains an optional list of capability type definitions for use in service templates. Such type definitions may be used within the same file, or a TOSCA Definitions file may also just contain capability type definitions for use in other files.
artifact_types	no	This section contains an optional list of artifact type definitions for use in service templates. Such type definitions may be used within the same file, or a TOSCA Definitions file may also just contain capability type definitions for use in other files.
outputs	no	This optional section allows for defining a set of output parameters provided to users of the template. For example, this can be used for exposing the URL for logging into a web application that has been set up during the instantiation of a template.
groups	no	This is an optional section that contains grouping definition for node templates.

677 ## A.4.2 Grammar

678 The overall structure of a TOSCA Service Template and its top-level key collations using the TOSCA
679 Simple Profile is shown below:

```
tosca_definitions_version: # Required TOSCA Definitions version string
tosca_default_namespace:   # Optional. default namespace (schema, types version)
template_name:             # Optional name of this service template
template_author:           # Optional author of this service template
template_version:          # Optional version of this service template

description: A short description of the definitions inside the file.

imports:
  # list of import statements for importing other definitions files

inputs:
  # list of global input parameters

node_templates:
  # list of node templates

node_types:
  # list of node type definitions

capability_types:
```



```
  # list of capability type definitions

relationship_types:
  # list of relationship type definitions

artifact_types:
  # list of artifact type definitions

groups:
  # list of groups defined in service template

outputs:
  # list of output parameters
```

A.4.3 Top-level key definitions

A.4.3.1 tosca_definitions_version

This required element provides a means include a reference to the TOSCA Simple Profile specification within the TOSCA Definitions YAML file. It is an indicator for the version of the TOSCA grammar that should be used to parse the remainder of the document.

A.4.3.1.1 *Keyword*

```
tosca_definitions_version
```

A.4.3.1.2 *Grammar*

Single-line form:

```
tosca_definitions_version: <tosca_simple_profile_version>
```

A.4.3.1.3 *Examples:*

TOSCA Simple Profile version 1.0 specification using the defined namespace alias (see Section A.1):

```
tosca_definitions_version: tosca_simple_yaml_1_0
```

TOSCA Simple Profile version 1.0 specification using the fully defined (target) namespace (see Section A.1):

```
tosca_definitions_version: http://docs.oasis-open.org/tosca/simple/1.0
```

A.4.3.2 template_name

This optional element declares the optional name of service template as a single-line string value.

694 A.4.3.2.1 ***Keyword***

```
template_name
```

695 A.4.3.2.2 ***Grammar***

```
template_name: <name string>
```

696 A.4.3.2.3 ***Example***

```
template_name: My service template
```

697 A.4.3.2.4 ***Notes***

698 • Some service templates are designed to be referenced and reused by other service templates.
699 Therefore, in these cases, the **`template_name`** value SHOULD be designed to be used as a unique
700 identifier through the use of namespacing techniques.

701 A.4.3.3 template_author

702 This optional element declares the optional author(s) of the service template as a single-line string value.

703 A.4.3.3.1 ***Keyword***

```
template_author
```

704 A.4.3.3.2 ***Grammar***

```
template_author: <author string>
```

705 A.4.3.3.3 ***Example***

```
template_name: My service template
```

706 A.4.3.4 template_version

707 This element declares the optional version of the service template as a single-line string value.

708 A.4.3.4.1 ***Keyword***

```
template_version
```

709 A.4.3.4.2 ***Grammar***

```
template_version: <version string>
```

710 A.4.3.4.3 ***Example***

```
template_version: v9.17.a
```


A.4.3.4.4 *Notes:*

- Some service templates are designed to be referenced and reused by other service templates and have a lifecycle of their own. Therefore, in these cases, a **`template_version`** value SHOULD be included and used in conjunction with a unique **`template_name`** value to enable lifecycle management of the service template and its contents.

A.4.3.5 Description

This optional element provides a means include single or multiline descriptions within a TOSCA Simple Profile template as a scalar string value.

A.4.3.5.1 *Keyword*

```
description
```

A.4.3.6 imports

This optional element provides a way to import a *block sequence* of one or more TOSCA Definitions documents. TOSCA Definitions documents can contain reusable TOSCA type definitions (e.g., Node Types, Relationship Types, Artifact Types, etc.) defined by other authors. This mechanism provides an effective way for companies and organizations to define normative types and/or describe their software applications for reuse in other TOSCA Service Templates.

A.4.3.6.1 *Keyword*

```
imports
```

A.4.3.6.2 *Grammar*

```
imports:
   - <tosca_definitions_file_1>
   - ...
   - <tosca_definitions_file_n>
```

A.4.3.6.3 *Example*

```
# An example import of definitions files from a location relative to the
# file location of the service template declaring the import.

imports:
  - relative_path/my_defns/my_typesdefs_1.yaml
  - ...
  - relative_path/my_defns/my_typesdefs_n.yaml
```

A.4.3.7 inputs

This optional element provides a means to define parameters, their allowed values via constraints and default values within a TOSCA Simple Profile template.

This section defines template-level input parameter section.

- This would require a change to template schema for v1.1
- Treat input parameters as fixed global variables (not settable within template)
- If not in input take default (nodes use default)

A.4.3.7.1 *Grammar*

```
inputs:
  <property_definition_1>
  ...
  <property_definition_n>
```

A.4.3.7.2 *Examples*

Simple example without any constraints:

```
inputs:
  fooName:
    type: string
    description: Simple string typed property definition with no constraints.
    default: bar
```

Example with constraints:

```
inputs:
  SiteName:
    type: string
    description: string typed property definition with constraints
    default: My Site
    constraints:
      - min_length: 9
```

A.4.3.7.3 *Notes*

- The parameters (properties) that are listed as part of the **`inputs`** block could be mapped to **`PropertyMappings`** provided as part of **`BoundaryDefinitions`** as described by the TOSCA v1.0 specification.

A.4.3.8 node_templates

This element lists the Node Templates that describe the (software) components that are used to compose cloud applications.

A.4.3.8.1 *Keyword*

```
node_templates
```


A.4.3.8.2 *Grammar*

```
node_templates:
  <node_template_defn_1>
  ...
  <node_template_defn_n>
```

A.4.3.8.3 *Example*

```
node_templates:

  my_webapp_node_template:
    type: WebApplication

  my_database_node_template:
    type: Database
```

A.4.3.8.4 *Notes*

- The node templates listed as part of the **node_templates** block can be mapped to the list of **NodeTemplate** definitions provided as part of **TopologyTemplate** of a **ServiceTemplate** as described by the TOSCA v1.0 specification.

A.4.3.9 node_types

This element lists the Node Types that provide the reusable type definitions for software components that Node Templates can be based upon.

A.4.3.9.1 *Keyword*

```
node_types
```

A.4.3.9.2 *Grammar*

```
node_types:
  <node_types_defn_1>
  ...
  <node_type_defn_n>
```

A.4.3.9.3 *Example*

```
node_types:
  my_webapp_node_type:
    derived_from: WebApplication
    properties:
      my_port:
        type: integer
```



```
  my_database_node_type:
    derived_from: Database
    capabilities:
      mytypes.myfeatures.transactSQL
```

A.4.3.9.4 *Notes*

- The node types listed as part of the **node_types** block can be mapped to the list of **NodeType** definitions as described by the TOSCA v1.0 specification.

A.4.3.10 relationship_types

This element lists the Relationship Types that provide the reusable type definitions that can be used to describe dependent relationships between Node Templates or Node Types.

A.4.3.10.1 *Keyword*

```
relationship_types
```

A.4.3.10.2 *Grammar*

```
relationship_types:
  <relationship_types_defn_1>
  ...
  <relationship type_defn_n>
```

A.4.3.10.3 *Example*

```
relationship_types:
  mycompany.mytypes.myCustomClientServerType:
    derived_from: tosca.relationships.HostedOn
    properties:
      # more details ...

  mycompany.mytypes.myCustomConnectionType:
    derived_from: tosca.relationships.ConnectsTo
    properties:
      # more details ...
```

A.4.3.11 capability_types

This element lists the Capability Types that provide the reusable type definitions that can be used to describe features Node Templates or Node Types can declare they support.

A.4.3.11.1 *Keyword*

```
capability_types
```


774 A.4.3.11.2 ***Grammar***

```
capability_types:
  <capability_type_defn_1>
  ...
  <capability type_defn_n>
```

775 A.4.3.11.3 ***Example***

```
capability_types:
  mycompany.mytypes.myCustomEndpoint
    derived_from: tosca.capabilities.Endpoint
    properties:
      # more details ...

  mycompany.mytypes.myCustomFeature
    derived_from: tosca.capabilites.Feature
    properties:
      # more details ...
```

776 ### A.4.3.12 groups

777 The group construct is a composition element used to group one or more node templates within a TOSCA
778 Service Template.

779 A.4.3.12.1 ***Keyword***

```
groups
```

780 A.4.3.12.2 ***Grammar***

```
groups:
  <group_name_A>:
    <node_template_defn_A_1>
    ...
    <node_template_defn_A_n>

  <group_name_B>
    <node_template_defn_B_1>
    ...
    <node_template_defn_B_n>
```

781 A.4.3.12.3 ***Example***

```
node_templates:
  server1:
```



```
    type: tosca.nodes.Compute
    # more details ...

  server2:
    type: tosca.nodes.Compute
    # more details ...

  server3:
    type: tosca.nodes.Compute
    # more details ...

groups:
  server_group_1:
    members: [ server1, server2 ]
    policies:
      - anti_collocation_policy:
          # specific policy declarations omitted, as this is not yet specified
```

A.4.3.13 outputs

This optional element provides a means to define the output parameters that are available from a TOSCA Simple Profile service template.

A.4.3.13.1 *Keyword*

```
outputs
```

A.4.3.13.2 *Grammar*

```
outputs:
  <property_definitions>
```

A.4.3.13.3 *Example*

```
outputs:
  server_ip:
    description: The IP address of the provisioned server.
    value: { get_property: [ my_server, ip_address ] }
```

A.5 Service Template-level functions

This section includes functions that are supported for use within a TOSCA Service Template.

A.5.1 Property functions

A.5.1.1 get_input

- **get_input** is used to retrieve the values of properties declared within the **inputs** section of the a service template.

A.5.1.2 get_property

- **get_property** is used to retrieve property values between entities defined in the same service template.

A.5.1.3 get_ref_property

- **get_ref_property** *is used by an entity defined in one service template to obtain a property value from another entity defined in a second service template. The first entity can reference the name of the other entity (which may be bound at runtime) as declared in its* ***requirements*** *section.*

A.5.2 Navigation functions

- This version of the TOSCA Simple Profile does not define any model navigation functions.

Appendix B. TOSCA normative type definitions

The declarative approach is heavily dependent of the definition of basic types that a declarative container must understand. The definition of these types must be very clear such that the operational semantics can be precisely followed by a declarative container to achieve the effects intended by the modeler of a topology in an interoperable manner.

B.1 Assumptions

- Assumes alignment with/dependence on XML normative types proposal for TOSCA v1.1
- Assumes that the normative types will be versioned and the TOSCA TC will preserve backwards compatibility.
- Assumes that security and access control will be addressed in future revisions or versions of this specification.

B.2 Requirement Types

There are no normative Requirement Types currently defined in this working draft.

B.3 Capabilities Types

B.3.1 tosca.capabilities.Root

This is the default (root) TOSCA Capability Type definition that all other TOSCA Capability Types derive from.

B.3.1.1 Definition

```
tosca.capabilities.Root:
```

B.3.2 tosca.capabilities.Feature

This is the default TOSCA type that should be extended to define any named feature of a node.

Shorthand Name	Feature
Type Qualified Name	tosca:Feature
Type URI	tosca.capabilities.Feature

B.3.2.1 Definition

```
tosca.capabilities.Feature:
  derived_from: tosca.capabilities.Root
```

B.3.3 tosca.capabilities.Container

The Container capability, when included on a Node Type or Template definition, indicates that the node can act as a container for (or a host for) one or more other declared Node Types.

828

Shorthand Name	Container
Type Qualified Name	tosca:Container
Type URI	tosca.capabilities.Container

829 ### B.3.3.1 Keynames

Name	Required	Constraints	Description
containee_types	yes	None	A list of one or more names of Node Types that are supported as containees that declare the Container type as a Capability.

830 ### B.3.3.2 Definition

```
tosca.capabilities.Container:
  derived_from: tosca.capabilities.Feature
  containee_types: [ <node_type_1>,..., <node_type_n> ]
```

831 ## B.3.4 tosca.capabilities.Endpoint

832 This is the default TOSCA type that should be used or extended to define a network endpoint capability.

Shorthand Name	Endpoint
Type Qualified Name	tosca:Endpoint
Type URI	tosca.capabilities.Endpoint

833 ### B.3.4.1 Properties

Name	Required	Type	Constraints	Description
protocol	yes	string	None	The name of the protocol (i.e., the protocol prefix) that the endpoint accepts. Examples: http, https, tcp, udp, etc.
port	yes	integer	greater_or_equal: 1 less_or_equal: 65535	The port of the endpoint.
secure	no	boolean	default = false	Indicates if the endpoint is a secure endpoint.

834 ### B.3.4.2 Definition

```
tosca.capabilities.Endpoint:
  derived_from: tosca.capabilities.Feature
  properties:
    protocol:
      type: string
```



```
      default: http
    port:
      type: integer
      constraints:
        - greater_or_equal: 1
        - less_or_equal: 65535
    secure:
      type: boolean
      default: false
```

B.3.5 tosca.capabilities.DatabaseEndpoint

This is the default TOSCA type that should be used or extended to define a specialized database endpoint capability.

Shorthand Name	DatabaseEndpoint
Type Qualified Name	tosca:DatabaseEndpoint
Type URI	tosca.capabilities.DatabaseEndpoint

B.3.5.1 Properties

Name	Required	Type	Constraints	Description
None	N/A	N/A	N/A	N/A

B.3.5.2 Definition

```
tosca.capabilities.DatabaseEndpoint:
  derived_from: tosca.capabilities.Endpoint
```

B.4 Relationship Types

B.4.1 tosca.relationships.Root

This is the default (root) TOSCA Relationship Type definition that all other TOSCA Relationship Types derive from.

B.4.1.1 Definition

```
tosca.relationships.Root:
  # The TOSCA root relationship type has no property mappings
  interfaces: [ tosca.interfaces.relationship.Configure ]
```

B.4.2 tosca.relationships.DependsOn

This type represents a general dependency relationship between two nodes.

Shorthand Name	DependsOn
Type Qualified Name	tosca:DependsOn
Type URI	tosca.relationships.DependsOn

847 #### B.4.2.1 Definition

```
tosca.relationships.DependsOn:
  derived_from: tosca.relationships.Root
  valid_targets: [ tosca.capabilities.Feature ]
```

848 ### B.4.3 tosca.relationships.HostedOn

849 This type represents a hosting relationship between two nodes.

Shorthand Name	HostedOn
Type Qualified Name	tosca:HostedOn
Type URI	tosca.relationships.HostedOn

850 #### B.4.3.1 Definition

```
  tosca.relationships.HostedOn:
    derived_from: tosca.relationships.DependsOn
    valid_targets: [ tosca.capabilities.Container ]
```

851 ### B.4.4 tosca.relationships.ConnectsTo

852 This type represents a network connection relationship between two nodes.

Shorthand Name	ConnectsTo
Type Qualified Name	tosca:ConnectsTo
Type URI	tosca.relationships.ConnectsTo

853 #### B.4.4.1 Definition

```
tosca.relations.ConnectsTo:
  derived_from: tosca.relationships.DependsOn
  valid_targets: [ tosca.capabilities.Endpoint ]
```

854 ## B.5 Interfaces

855 Interfaces are reusable entities that define a set of operations that that can be included as part of a Node
856 type or Relationship Type definition. Each named operations may have code or scripts associated with
857 them that orchestrators can execute for when transitioning an application to a given state.

B.5.1 Notes

- Designers of Node or Relationship types are not required to actually provide/associate code or scripts with every operation for a given interface it supports. In these cases, orchestrators SHALL consider that a “No Operation” or “no-op”.
- Template designers MAY provide or override code or scripts provided by a type for a specified interface defined for the type (even if the type itself does not provide a script for that operation).

B.5.2 tosca.interfaces.node.Lifecycle

The lifecycle interfaces define the essential, normative operations that TOSCA nodes may support.

Shorthand Name	Lifecycle
Type Qualified Name	tosca:Lifecycle
Type URI	tosca.relationships.node.Lifecycle

B.5.2.1 Definition

```
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.
```

B.5.3 tosca.interfaces.relationship.Configure

The lifecycle interfaces define the essential, normative operations that each TOSCA Relationship Types may support.

Shorthand Name	Configure
Type Qualified Name	tosca:Configure
Type URI	tosca.interfaces.relationship.Configure

B.5.3.1 Definition

```
tosca.interfaces.relationship.Configure:
  pre_configure_source:
```



```
    description: Operation to pre-configure the source endpoint.
  pre_configure_target:
    description: Operation to pre-configure the target endpoint.
  post_configure_source:
    description: Operation to post-configure the source endpoint.
  post_configure_target:
    description: Operation to post-configure the target endpoint.
  add_target:
    description: Operation to add a target node.
  remove_target:
    description: Operation to remove a target node.
```

B.6 Node Types

B.6.1 tosca.nodes.Root

The TOSCA **Root** Node Type is the default type that all other TOSCA base Node Types derive from. This allows for all TOSCA nodes to have a consistent set of features for modeling and management (e.g., consistent definitions for requirements, capabilities and lifecycle interfaces).

B.6.1.1 Properties

Name	Required	Type	Constraints	Description
N/A	N/A	N/A	N/A	The TOSCA Root Node type has no specified properties.

B.6.1.2 Definition

```
tosca.nodes.Root:
  description: The TOSCA Node Type all other TOSCA base Node Types derive from
  requirements:
    - dependency:
      type: tosca.capabilities.Feature
      lower_bound: 0
      upper_bound: unbounded
  capabilities:
    feature: tosca.capabilities.Feature
  interfaces: [ tosca.interfaces.node.Lifecycle ]
```

B.6.1.3 Additional Requirements

- All Node Type definitions that wish to adhere to the TOSCA Simple Profile SHOULD extend from the TOSCA Root Node Type to be assured of compatibility and portability across implementations.

B.6.2 tosca.nodes.Compute

The TOSCA **Compute** node represents one or more real or virtual processors of software applications or services along with other essential local resources. Collectively, the resources the compute node represents can logically be viewed as a (real or virtual) "server".

Shorthand Name	Compute
Type Qualified Name	tosca:Compute
Type URI	tosca.nodes.Compute

B.6.2.1 Properties

Name	Required	Type	Constraints	Description
num_cpus	No	integer	>= 1	Number of (actual or virtual) CPUs associated with the Compute node.
disk_size	No	integer	>=0	Size of the local disk, in Gigabytes (GB), available to applications running on the Compute node.
mem_size	No	integer	>= 0	Size of memory, in Megabytes (MB), available to applications running on the Compute node.
os_arch	Yes	string	None	The host Operating System (OS) architecture. Examples of valid values include: x86_32, x86_64, etc.
os_type	Yes	string	None	The host Operating System (OS) type. Examples of valid values include: linux, aix, mac, windows, etc.
os_distribution	No	string	None	The host Operating System (OS) distribution. Examples of valid values for an "os_type" of "Linux" would include: debian, fedora, rhel and ubuntu.
os_version	No	string	None	The host Operating System version.
ip_address	No	string	None	The primary IP address assigned by the cloud provider that applications may use to access the Compute node. • **Note**: This is used by the platform provider to convey the primary address used to access the compute node. Future working drafts will address implementations that support floating or multiple IP addresses.

B.6.2.2 Definition

```
type: tosca.nodes.Compute
  derived_from: tosca.nodes.Root
  properties:
      # compute properties
```

```
      num_cpus:
         type: integer
         constraints:
           - greater_or_equal: 1
      disk_size:
         type: integer
         constraints:
           - greater_or_equal: 0
      mem_size:
         type: integer
         constraints:
           - greater_or_equal: 0

      # host image properties
      os_arch:
         type: string
      os_type:
         type: string
      os_distribution:
         type: string
      os_version:
         type: string

      # Compute node's primary IP address
      ip_address:
         type: string
    capabilities:
      host:
        type: Container
        containee_types: [tosca.nodes.SoftwareComponent]
```

B.6.2.3 Additional Requirements

- Please note that the string values for the properties "os_arch", "os_type" and "os_distribution" SHALL be normalized to lowercase by processors of the service template for matching purposes. For example, if an "os_type" value is set to either "Linux", "LINUX" or "linux" in a service template, the processor would normalize all three values to "linux" for matching purposes.

B.6.3 tosca.nodes.SoftwareComponent

The TOSCA **`SoftwareComponent`** node represents a generic software component that can be managed and run by a TOSCA **`Compute`** Node Type.

Shorthand Name	SoftwareComponent
Type Qualified Name	tosca:SoftwareComponent
Type URI	tosca.nodes.SoftwareComponent

B.6.3.1 Properties

Name	Required	Type	Constraints	Description
version	no	string	None	The software component's version.

B.6.3.2 Definition

```
tosca.nodes.SoftwareComponent:
  derived_from: tosca.nodes.Root
  properties:
      # software component version
      version:
        type: string
        required: false
  requirements:
    - host: tosca.nodes.Compute
```

B.6.3.3 Additional Requirements

- Nodes that can directly be managed and run by a TOSCA **`Compute`** Node Type SHOULD extend from this type.

B.6.4 tosca.nodes.WebServer

This TOSA **`WebServer`** Node Type represents an abstract software component or service that is capable of hosting and providing management operations for one or more **`WebApplication`** nodes.

Shorthand Name	WebServer
Type Qualified Name	tosca:WebServer
Type URI	tosca.nodes.WebServer

B.6.4.1 Properties

Name	Required	Type	Constraints	Description
None	N/A	N/A	N/A	N/A

B.6.4.2 Definition

```
tosca.nodes.WebServer
  derived_from: tosca.nodes.SoftwareComponent
  capabilities:
    http_endpoint: tosca.capabilites.Endpoint
    https_endpoint: tosca.capabilities.Endpoint
    host:
      type: Container
      containee_types: [ tosca.nodes.WebApplication ]
```

B.6.4.3 Additional Requirements

- None

B.6.5 tosca.nodes.WebApplication

The TOSCA **WebApplication** node represents a software application that can be managed and run by a TOSCA **WebServer** node. Specific types of web applications such as Java, etc. could be derived from this type.

Shorthand Name	WebApplication
Type Qualified Name	tosca: WebApplication
Type URI	tosca.nodes.WebApplication

B.6.5.1 Properties

Name	Required	Type	Constraints	Description
None	N/A	N/A	N/A	N/A

B.6.5.2 Definition

```
tosca.nodes.WebApplication:
  derived_from: tosca.nodes.Root
  requirements:
    - host: tosca.nodes.WebServer
```

B.6.5.3 Additional Requirements

- None

B.6.6 tosca.nodes.DBMS

The TOSCA **DBMS** node represents a typical relational, SQL Database Management System software component or service.

B.6.6.1 Properties

Name	Required	Type	Constraints	Description
dbms_root_password	yes	string	None	The DBMS server's root password.
dbms_port	no	integer	None	The DBMS server's port.

B.6.6.2 Definition

```
tosca.nodes.DBMS
  derived_from: tosca.nodes.SoftwareComponent
  properties:
    dbms_root_password:
      type: string
      description: the root password for the DBMS service
    dbms_port:
      type: integer
      description: the port the DBMS service will listen to for data and requests
  capabilities:
    host:
      type: Container
      containee_types: [ tosca.nodes.Database ]
```

B.6.6.3 Additional Requirements

- None

B.6.7 tosca.nodes.Database

Base type for the schema and content associated with a DBMS.

The TOSCA **Database** node represents a logical database that can be managed and hosted by a TOSCA **DBMS** node.

Shorthand Name	Database
Type Qualified Name	tosca:Database
Type URI	tosca.nodes.Database

B.6.7.1 Properties

Name	Required	Type	Constraints	Description
db_user	yes	string	None	The special user account used for database administration.
db_password	yes	string	None	The password associated with the user account provided in the 'db_user' property.
db_port	yes	integer	None	The port the database service will use to listen for incoming data and requests.

Name	Required	Type	Constraints	Description
db_name	yes	string	None	The logical database Name

930 #### B.6.7.2 Definition

```
tosca.nodes.Database:
  derived_from: tosca.nodes.Root
  properties:
    db_user:
      type: string
      description: user account name for DB administration
    db_password:
      type: string
      description: the password for the DB user account
    db_port:
      type: integer
      description: the port the underlying database service will listen to data
    db_name:
      type: string
      description: the logical name of the database
  requirements:
    - host: tosca.nodes.DBMS
  capabilities:
    - database_endpoint: tosca.capabilities.DatabaseEndpoint
```

931 #### B.6.7.3 Additional Requirements

932 • None

933 ## B.6.8 tosca.nodes.ObjectStorage

934 The TOSCA **`ObjectStorage`** node represents storage that provides the ability to store data as objects (or
935 BLOBs of data) without consideration for the underlying filesystem or devices.

Shorthand Name	ObjectStorage
Type Qualified Name	tosca:ObjectStorage
Type URI	tosca.nodes.ObjectStorage

936 #### B.6.8.1 Properties

Name	Required	Type	Constraints	Description
store_name	yes	string	None	The logical name of the object store (or container).
store_size	no	integer	>=0	The requested initial storage size in Gigabytes.
store_maxsize	no	integer	>=0	The requested maximum storage size in Gigabytes.

937

938 #### B.6.8.2 Definition

```
tosca.nodes.ObjectStorage
  derived_from: tosca.nodes.Root
  properties:
    store_name:
       type: string
    store_size:
       type: integer
       constraints:
         - greater_or_equal: 0
    store_maxsize:
       type: integer
       constraints:
         - greater_or_equal: 0
```

939 #### B.6.8.3 Additional Requirements

940 • None

941 #### B.6.8.4 Notes:

942 • Subclasses of the ObjectStorage node may impose further constraints on properties such as
943 **`store_name`**, such as minimum and maximum lengths or include regular expressions to
944 constrain allowed characters.

945 ## B.6.9 tosca.nodes.BlockStorage

946 The TOSCA **`BlockStorage`** node currently represents a server-local block storage device (i.e., not
947 shared) offering evenly sized blocks of data from which raw storage volumes can be created.

948 **Note**: In this draft of the TOSCA Simple Profile, distributed or Network Attached Storage (NAS) are not
949 yet considered (nor are clustered file systems), but the TC plans to do so in future drafts.

Shorthand Name	BlockStorage
Type Qualified Name	tosca:BlockStorage
Type URI	tosca.nodes.BlockStorage

950 #### B.6.9.1 Properties

Name	Required	Type	Constraints	Description
store_mount_path	yes	string	min_length: 1	The relative directory on the file system, which provides the root directory for the mounted volume.

Name	Required	Type	Constraints	Description
store_fs_type	no	string	None	The type of disk file system. Examples include: ext2, ext3, reiser, etc.

951 ### B.6.9.2 Definition

```
type: tosca.nodes.BlockStorage
  derived_from: tosca.nodes.Root
  properties:
      store_fs_type:
         type: string
      store_mount_path:
         type: string
         constraints:
           - min_length: 1
```

952 ### B.6.9.3 Additional Requirements

953 - None

954 ## B.6.10 tosca.nodes.Network

955 The TOSCA **Network** node represents a simple, logical network service.

956 **Note**: This base Node Type will be further developed in future drafts of this specification.

Shorthand Name	Network
Type Qualified Name	tosca:Network
Type URI	tosca.nodes.Network

957 ### B.6.10.1 Properties

Name	Required	Type	Constraints	Description
TBD	N/A	N/A	N/A	N/A

958 ### B.6.10.2 Definition

```
  tosca.nodes.Network:
    derived_from: tosca.nodes.Root
```

959 ### B.6.10.3 Additional Requirements

960 - TBD

B.7 Artifact Types

TOSCA Artifacts represent the packages and imperative used by the orchestrator when invoking TOSCA Interfaces on Node or Relationship Types. Currently, artifacts are logically divided into three categories:

- **Deployment Types**: includes those artifacts that are used during deployment (e.g., referenced on create and install operations) and include packaging files such as RPMs, ZIPs, or TAR files.
- **Implementation Types**: includes those artifacts that represent imperative logic and are used to implement TOSCA Interface operations. These typically include scripting languages such as Bash (.sh), Chef and Puppet.
- **Runtime Types**: includes those artifacts that are used during runtime by a service or component of the application. This could include a library or language runtime that is needed by an application such as a PHP or Java library.

Note: Normative TOSCA Artifact Types will be developed in future drafts of this specification.

B.7.1 tosca.artifacts.Root

This is the default (root) TOSCA Artifact Type definition that all other TOSCA base Artifact Types derive from.

B.7.1.1 Definition

```
tosca.artifacts.Root:
  description: The TOSCA Artifact Type all other TOSCA Artifact Types derive from
```

B.7.2 tosca.artifacts.File

This artifact type is used when an artifact definition needs to have its associated file simply treated as a file and no special handling/handlers are invoked.

B.7.2.1 Definition

```
tosca.artifacts.File:
  derived_from: tosca.artifacts.Root
```

B.7.3 Implementation Types

B.7.3.1 Script Types

B.7.3.1.1 *tosca.artifacts.impl.Bash*

This artifact type represents a Bash script type that contains Bash commands that can be executed on the Unix Bash shell.

988 **B.7.3.2 Definition**

```
tosca.artifacts.impl.Bash:
  derived_from: tosca.artifacts.Root
  description: Script artifact for the Unix Bash shell
  properties:
    mime_type: application/x-sh
    file_ext: [ sh ]
```


Appendix C. Non-normative type definitions

This section defines non-normative types used in examples or use cases within this specification.

C.1 Capability Types

C.1.1 tosca.capabilities.DatabaseEndpoint.MySQL

This type defines a custom MySQL database endpoint capability.

C.1.1.1 Properties

Name	Required	Type	Constraints	Description
None	N/A	N/A	N/A	N/A

C.1.1.2 Definition

```
tosca.capabilities.DatabaseEndpoint.MySQL:
  derived_from: tosca.capabilities.DatabaseEndpoint
```

C.2 Node Types

C.2.1 tosca.nodes.Database.MySQL

C.2.1.1 Properties

Name	Required	Type	Constraints	Description
None	N/A	N/A	N/A	N/A

C.2.1.2 Definition

```
tosca.nodes.Database.MySQL:
  derived_from: tosca.nodes.Database
  requirements:
    - host: tosca.nodes.DBMS.MySQL
  capabilities:
    database_endpoint: tosca.capabilities.DatabaseEndpoint.MySQL
```

C.2.2 tosca.nodes.DBMS.MySQL

C.2.2.1 Properties

Name	Required	Type	Constraints	Description
None	N/A	N/A	N/A	N/A

1002 #### C.2.2.2 Definition

```
tosca.nodes.Database.MySQL:
  derived_from: tosca.nodes.DBMS
  properties:
    dbms_port:
      description: reflect the default MySQL server port
      default: 3306
  capabilities:
    host:
      type: Container
      containee_types: [ tosca.nodes.Database.MySQL ]
```

1003 ## C.2.3 tosca.nodes.WebServer.Apache

1004 #### C.2.3.1 Properties

Name	Required	Type	Constraints	Description
None	N/A	N/A	N/A	N/A

1005 #### C.2.3.2 Definition

```
tosca.nodes.WebServer.Apache:
  derived_from: tosca.nodes.WebServer
```

1006 ## C.2.4 tosca.nodes.WebApplication.WordPress

1007 #### C.2.4.1 Properties

Name	Required	Type	Constraints	Description
None	N/A	N/A	N/A	N/A

1008 #### C.2.4.2 Definition

```
tosca.nodes.WebApplication.WordPress:
  derived_from: tosca.nodes.WebApplication
  properties:
    admin_user:
      type: string
    admin_password:
      type: string
    db_host:
      type: string
```



```
  requirements:
    - host: tosca.nodes.WebServer
    - database_endpoint: tosca.nodes.Database
  interfaces:
    Lifecycle:
      inputs:
        db_host: string
        db_port: integer
        db_name: string
        db_user: string
        db_password: string
```


Appendix D. Use Cases

D.1 Application Modeling Use Cases:

Short description	Interesting Feature	Description
Virtual Machine (VM), single instance	• Introduces the TOSCA base Node Type for "Compute".	TOSCA simple profile ates how to stand up a single instance of a Virtual Machine (VM) image using a normative TOSCA Compute node.
WordPress + MySQL, single instance	• Introduces the TOSCA base Node Types of: "WebServer", "WebApplication", "DBMS" and "Database" along with their dependent hosting and connection relationships.	TOSCA simple profile service showing the WordPress web application with a MySQL database hosted on a single server (instance).
WordPress + MySQL + Object Storage, single instance	• Introduces the TOSCA base Node Type for "ObjectStorage".	TOSCA simple profile service showing the WordPress web application hosted on a single server (instance) with attached (Object) storage.
WordPress + MySQL + Block Storage, single instance	• Introduces the TOSCA base Node Type for "BlockStorage" (i.e., for Volume-based storage).	TOSCA simple profile service showing the WordPress web application hosted on a single server (instance) with attached (Block) storage.
WordPress + MySQL, each on separate instances	• Instantiates 2 tiers, 1 for WordPress, 1 for DMBS and coordinates both.	Template installs two instances: one running a WordPress deployment and the other using a specific (local) MySQL database to store the data.
WordPress + MySQL + Network, single instance	• Introduces the TOSCA base Node Type for a simple "Network".	TOSCA simple profile service showing the WordPress web application and MySQL database hosted on a single server (instance) along with demonstrating how to define associate the instance to a simple named network.
WordPress + MySQL + Floating IPs, single instance	• Connects to an external (relational) DBMS service	TOSCA simple profile service showing the WordPress web application and MySQL database hosted on a single server (instance) along with demonstrating how to create a network for the application with Floating IP addresses.

D.1.1 Virtual Machine (VM), single instance

D.1.1.1 Description

This use case demonstrates how the TOSCA Simple Profile specification can be used to stand up a single instance of a Virtual Machine (VM) image using a normative TOSCA **`Compute`** node. The TOSCA Compute node is declarative in that the service template describes both the processor and host operating system platform characteristics (i.e., properties) that are desired by the template author. The cloud provider would attempt to fulfill these properties (to the best of its abilities) during orchestration.

D.1.1.2 Features

This use case introduces the following TOSCA Simple Profile features:

- A node template that uses the normative TOSCA **`Compute`** Node Type along with showing an exemplary set of its properties being configured.

- Use of the TOSCA Service Template **inputs** section to declare a configurable value the template user may supply at runtime. In this case, the property named “cpus” (of type integer) is declared.
  - Use of a property constraint to limit the allowed integer values for the “cpus” property to a specific list supplied in the property declaration.
- Use of the TOSCA Service Template **outputs** section to declare a value the template user may request at runtime. In this case, the property named “instance_ip” is declared
  - The “instance_ip” output property is programmatically retrieved from the **Compute** node’s “ip_address” property using the TOSCA Service Template-level **get_property** function.

D.1.1.3 Logical Diagram

TBD

D.1.1.4 Sample YAML

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: >
  TOSCA simple profile that just defines a single compute instance. Note, this
example does not include default values on inputs properties.

inputs:
  cpus:
    type: integer
    description: Number of CPUs for the server.
    constraints:
      - valid_values: [ 1, 2, 4, 8 ]

node_templates:
  my_server:
    type: tosca.nodes.Compute
    properties:
      # compute properties
      disk_size: 10 # in GB
      num_cpus: { get_input: cpus }
      mem_size: 4  # in MB
      # host image properties
      os_arch: x86_64
       os_type: linux
       os_distribution: ubuntu
       os_version: 12.04
```



```
outputs:
  instance_ip:
    description: The IP address of the deployed instance.
    value: { get_property: [my_server, ip_address] }
```

D.1.1.5 Notes

- This use case uses a versioned, Linux Ubuntu distribution on the Compute node.

D.1.2 WordPress + MySQL, single instance

D.1.2.1 Description

TOSCA simple profile service showing the WordPress web application with a MySQL database hosted on a single server (instance).

This use case is built upon the following templates fro, OpenStack Heat's Cloud Formation (CFN) template and from an OpenStack Heat-native template:

- https://github.com/openstack/heat-templates/blob/master/cfn/F17/WordPress_With_RDS.template
- https://github.com/openstack/heat-templates/blob/master/hot/F18/WordPress_Native.yaml

However, where the CFN template simply connects to an existing Relational Database Service (RDS) our template below will also install a MySQL database explicitly and connect to it.

D.1.2.2 Logical Diagram

TBD

D.1.2.3 Sample YAML

```
tosca_definitions_version: tosca_simple_1.0

description: >
  TOSCA simple profile with WordPress, a web server, mMySQL DBMS and mysql database
on the same server. Does not have input defaults or constraints.

inputs:
  cpus:
    type: number
    description: Number of CPUs for the server.
  db_name:
    type: string
    description: The name of the database.
  db_user:
```



```
    type: string
    description: The username of the DB user.
  db_pwd:
    type: string
    description: The WordPress database admin account password.
  db_root_pwd:
    type: string
    description: Root password for MySQL.
  db_port:
    type:integer
    description: Port for the MySQL database

node_templates:
  wordpress:
    type: tosca.nodes.WebApplication.WordPress
    requirements:
      - host: webserver
      - database_endpoint: mysql_database
    interfaces:
      create: wordpress_install.sh
      configure:
        implementation: wordpress_configure.sh
        input:
          wp_db_name: { get_property: [ mysql_database, db_name ] }
          wp_db_user: { get_property: [ mysql_database, db_user ] }
          wp_db_password: { get_property: [ mysql_database, db_password ] }
          # goto requirement, goto capability, goto port property
          wp_db_port: { get_ref_property: [ database_endpoint, database_endpoint,
port ] }

  mysql_database:
    type: tosca.nodes.Database
    properties:
      db_name: { get_input: db_name }
      db_user: { get_input: db_user }
      db_password: { get_input: db_pwd }
    capabilities:
      database_endpoint:
        properties:
          port: { get_input: db_port }
    requirements:
```



```
      - host: mysql_dbms
    interfaces:
      configure: mysql_database_configure.sh

  mysql_dbms:
    type: tosca.nodes.DBMS
    properties:
      dbms_root_password: { get_input: db_root_pwd }
      dbms_port: { get_input: db_root_pwd }
    requirements:
      - host: server
    interfaces:
      create: mysql_dbms_install.sh
      start: mysql_dbms_start.sh
      configure: mysql_dbms_configure
        input:
          db_root_password: { get_property: [ mysql_dbms, dbms_root_password ] }

  webserver:
    type: tosca.nodes.WebServer
    requirements:
      - host: server
    interfaces:
      create: webserver_install.sh
      start: webserver_start.sh

  server:
    type: tosca.nodes.Compute
    properties:
      # compute properties (flavor)
      disk_size: 10
      num_cpus: { get_input: cpus }
      mem_size: 4096
      # host image properties
      os_arch: x86_64
      os_type: Linux
      os_distribution: Fedora
      os_version: 17

outputs:
  website_url:
```

```
    description: URL for Wordpress wiki.
    value: { get_property: [server, ip_address] }
```

D.1.2.4 Sample scripts

Where the referenced implementation scripts in the example above would have the following contents

D.1.2.4.1 *wordpress_install.sh*

```
yum -y install wordpress
```

D.1.2.4.2 *wordpress_configure.sh*

```
sed -i "/Deny from All/d" /etc/httpd/conf.d/wordpress.conf
sed -i "s/Require local/Require all granted/" /etc/httpd/conf.d/wordpress.conf
sed -i s/database_name_here/db_name/ /etc/wordpress/wp-config.php
sed -i s/username_here/db_user/ /etc/wordpress/wp-config.php
sed -i s/password_here/db_password/ /etc/wordpress/wp-config.php
systemctl restart httpd.service
```

D.1.2.4.3 *mysql_database_configure.sh*

```
# Setup MySQL root password and create user
cat << EOF | mysql -u root --password=db_rootpassword
CREATE DATABASE db_name;
GRANT ALL PRIVILEGES ON db_name.* TO "db_user"@"localhost"
IDENTIFIED BY "db_password";
FLUSH PRIVILEGES;
EXIT
EOF
```

D.1.2.4.4 *mysql_dbms_install.sh*

```
yum -y install mysql mysql-server
# Use systemd to start MySQL server at system boot time
systemctl enable mysqld.service
```

D.1.2.4.5 *mysql_dbms_start.sh*

```
# Start the MySQL service (NOTE: may already be started at image boot time)
systemctl start mysqld.service
```


D.1.2.4.6 *mysql_dbms_configure*

```
# Set the MySQL server root password
mysqladmin -u root password db_rootpassword
```

D.1.2.4.7 *webserver_install.sh*

```
yum -y install httpd
systemctl enable httpd.service
```

D.1.2.4.8 *webserver_start.sh*

```
# Start the httpd service (NOTE: may already be started at image boot time)
systemctl start httpd.service
```

D.1.3 WordPress + MySQL + Object Storage, single instance

D.1.3.1 Description

This use case shows a WordPress application that makes use of an Object Storage service to application artifacts.

Note: Future drafts of this specification will detail this use case

D.1.3.2 Logical Diagram

TBD

D.1.3.3 Sample YAML

```
TBD
```

D.1.4 WordPress + MySQL + Block Storage, single instance

D.1.4.1 Description

This use case is based upon OpenStack Heat's Cloud Formation (CFN) template:

- https://github.com/openstack/heat-templates/blob/master/cfn/F17/WordPress_Single_Instance_With_EBS.template

Note: Future drafts of this specification will detail this use case.

D.1.4.2 Logical Diagram

TBD

D.1.4.3 Sample YAML

```
TBD
```


D.1.5 WordPress + MySQL, each on separate instances

D.1.5.1 Description

TOSCA simple profile service showing the WordPress web application hosted on one server (instance) and a MySQL database hosted on another server (instance).

This is based upon OpenStack Heat's Cloud Formation (CFN) template:

- https://github.com/openstack/heat-templates/blob/master/cfn/F17/WordPress_2_Instances.template

Note: Future drafts of this specification will detail this use case.

D.1.5.2 Logical Diagram

TBD

D.1.5.3 Sample YAML

```
TBD
```

D.1.6 WordPress + MySQL + Network, single instance

D.1.6.1 Description

This use case is based upon OpenStack Heat's Cloud Formation (CFN) template:

- https://github.com/openstack/heat-templates/blob/master/cfn/F17/WordPress_Single_Instance_With_Quantum.template

Note: Future drafts of this specification will detail this use case.

D.1.6.2 Logical Diagram

TBD

D.1.6.3 Sample YAML

```
TBD
```

D.1.7 WordPress + MySQL + Floating IPs, single instance

D.1.7.1 Description

This use case is based upon OpenStack Heat's Cloud Formation (CFN) template:

- https://github.com/openstack/heat-templates/blob/master/cfn/F17/WordPress_Single_Instance_With_EIP.template

Note: Future drafts of this specification will detail this use case.

D.1.7.2 Logical Diagram

TBD

D.1.7.3 Sample YAML

```
TBD
```

D.1.7.4 Notes

- The Heat/CFN use case also introduces the concept of “Elastic IP” (EIP) addresses which is the Amazon AWS term for floating IPs.
- The Heat/CFN use case provides a “key_name” as input which we will not attempt to show in this use case as this is a future security/credential topic.
- The Heat/CFN use case assumes that the “image” uses the “yum” installer to install Apache, MySQL and Wordpress and installs, starts and configures them all in one script (i.e., under Compute). In TOSCA we represent each of these software components as their own Nodes each with independent scripts.

Appendix E. Notes and Issues

E.1 Known Extensions to TOSCA v1.0

The following items will need to be reflected in the TOSCA (XML) specification to allow for isomorphic mapping between the XML and YAML service templates.

E.1.1 Model Changes

- The “TOSCA Simple ‘Hello World’” example introduces this concept in Section 3. Specifically, a VM image assumed to accessible by the cloud provider.
- Introduce template Input and Output parameters
- The “Template with input and output parameter” example introduces concept in Section 3.1.
  - “Inputs” could be mapped to BoundaryDefinitions in TOSCA v1.0. Maybe needs some usability enhancement and better description.
  - “outputs” are a new feature.
- Grouping of Node Templates
  - This was part of original TOSCA proposal, but removed early on from v1.0 This allows grouping of node templates that have some type of logically managed together as a group (perhaps to apply a scaling or placement policy).
- Lifecycle Operation definition independent/separate from Node Types or Relationship types (allows reuse). For now we added Lifecycle and Relationship
- Override of Interfaces (operations) in the Node Template.
- Service Template Naming/Versioning
  - Should include TOSCA spec. (or profile) version number (as part of namespace)
- Allow the referencing artifacts using a URL (e.g., as a property value).

E.1.2 Normative Types

- Constraint (addresses TOSCA-117)
- Property / Parameter
  - Includes YAML intrinsic types.
- Node
- Relationship
  - Root, DependsOn, HostedOn, ConnectsTo
- Artifact
  - Deployment: Bash (for WD01)
- Requirements
  - (TBD), Goal is to rely less upon source defined requirements that point to types, and instead reference names of features exported by the target nodes.
- Capabilities
  - Feature, Container, Endpoint
- Lifecycle
  - Lifecycle, Relationship
- Resource
  - In HEAT they have concept of key pairs (an additional resource type in the template).

E.1.3 Functions

- Intrinsic functions for model navigation, referencing etc.
  - get_input
  - get_property
  - get_ref_property

E.2 Issues to resolve in future drafts

Issue #	Target	Title	Notes
TOSCA-132	WD02	Use "set_property" methods to "push" values from template inputs to nodes	None
TOSCA-133	WD02	Add text/examples/grammar for defining a nested template that implements a node type	Proposed draft text exists, needs review/update.
TOSCA-134	WD02	Define TOSCA version type based upon Apache Maven versioning	None
TOSCA-135	WD02	Define/reference a Regex language (or subset) we wish to support for constraints	None
TOSCA-136	WD02	Need rules to assure non-collision (uniqueness) of requirement or capability names	None
TOSCA-137	WD02	Need to address "optional" and "best can" on node requirements (constraints) for matching/resolution	None
TOSCA-138	WD02	Define a Network topology for L2 Networks along with support for Gateways, Subnets, Floating IPs and Routers	Luc Boutier has rough proposal in MS Word format.
TOSCA-142	WD02	WD02 - Define normative Artifact Types (including deployment/packages, impls., and runtime types)	None
TOSCA-143	WD02	WD02 - Define normative tosca.nodes.Network Node Type (for simple networks)	Separate use case as what Luc proposes in TOSCA-138.
TOSCA-146	WD02	WD02 - Define a grammar for each property function and provide examples.	None
TOSCA-147	WD02	WD02 - Define grammar for and examples of using Relationship templates	None
TOSCA-148	WD02	WD02 - Need a means to express cardinality on relationships (e.g., number of connections allowed)	None
TOSCA-149	WD02	WD02 - Create an independent section to describe a single requirement definitions' grammar	Improvement for readability of grammar.
TOSCA-150	WD02	WD02 - Work towards a common syntax for Requirement definitions (currently 3 variants)	Related to TOSCA-149
TOSCA-151	WD02	WD02 - Resolve spec. behavior if name collisions occur on named Requirements	Dale assigned
TOSCA-152	WD02	WD02 - Extend Requirement grammar to support "Optional/Best Can" Capability Type matching	Derek assgined
TOSCA-153	WD02	WD02 - Define grammar and usage of Service Template keyname (schema namespace) "tosca_default_namespace"	Need to define what normative types may be implied to be automatically imported as part of the schema declaration.
TOSCA-154	WD02	WD02 - Decide how security/access control work with Nodes, update grammar, author descriptive text/examples	
TOSCA-155	WD02	WD02 - How do we provide constraints on properties declared as simple YAML lists (sets)	
TOSCA-156	WD02	WD02 - Are there IPv6 considerations (e.g., new properties) for tosca.capabilities.Endpoint	
TOSCA-157	WD02	WD02 - Can/how do we make a property defn. "final" or "read-only"	
TOSCA-158	WD02	WD02 - Provide prose describing how Feature matching is	Dependency on TOSCA-137,

		done by orchestrators	Future item, W03 or beyond.
TOSCA-159	WD02	WD02 - Describe how not all interfaces need to supply scripts (artifacts), it is a no-op behavior	
TOSCA-160	WD02	WD02 - Need examples of using the "tosca.interfaces.relationship.Configure" interface	
TOSCA-161	WD02	WD02 - Need examples of using the built-in feature (Capability) and dependency (Requirement) of tosca.nodes.Root	
TOSCA-162	WD02	WD02 - Provide recognized values for tosca.nodes.compute properties: os_arch	Could be WD03 item
TOSCA-163	WD02	WD02 - Provide recognized values for tosca.nodes.BlockStorage: store_fs_type	Could be WD03 item
TOSCA-164	WD02	WD02 - Do we need a restart lifecycle operation for nodes?	
TOSCA-165	WD02	WD02 - New use case / example: Selection/Replacement of web server type (e.g. Apache, NGinx, Lighttpd, etc.)	Could be WD03 item
TOSCA-166	WD02	WD02 - New use case / example: Web Server with (one or more) runtimes environments (e.g., PHP, Java, etc.)	Could be WD03 item
TOSCA-167	WD03	WD02 - New use case / example: Show abstract substitution of Compute node OS with different Node Type Impls.	Could be WD03 item
TOSCA-168	WD03	WD02 - New use case / example: Show how substitution of IaaS can be accomplished.	Could be WD03 item

1167

Appendix F. References

F.1 Terminology

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in **[TOSCA-1.0]**.

F.2 Normative References

[TOSCA-1.0] Topology and Orchestration Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0, an OASIS Standard, 25 November 2013, http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.pdf

[YAML-1.2] YAML, Version 1.2, 3rd Edition, Patched at 2009-10-01, Oren Ben-Kiki, Clark Evans, Ingy döt Net http://www.yaml.org/spec/1.2/spec.html

[YAML-TS-1.1] Timestamp Language-Independent Type for YAML Version 1.1, Working Draft 2005-01-18, http://yaml.org/type/timestamp.html

F.3 Non-Normative References

[AWS-CFN] Amazon Cloud Formation (CFN), http://aws.amazon.com/cloudformation/

[Chef] Chef, https://wiki.opscode.com/display/chef/Home

[OS-Heat] OpenStack Project Heat, https://wiki.openstack.org/wiki/Heat

[Puppet] Puppet, http://puppetlabs.com/

[WordPress] WordPress, https://wordpress.org/

Appendix G. Acknowledgments

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Contributors:

Derek Palma (dpalma@vnomic.com), Vnomic

Frank Leymann (Frank.Leymann@informatik.uni-stuttgart.de), Univ. of Stuttgart

Gerd Breiter (gbreiter@de.ibm.com), IBM

Jacques Durand (jdurand@us.fujitsu.com), Fujitsu

Juergen Meynert (juergen.meynert@ts.fujitsu.com), Fujitsu

Karsten Beins (karsten.beins@ts.fujitsu.com), Fujitsu

Kevin Wilson (kevin.l.wilson@hp.com), HP

Krishna Raman (kraman@redhat.com) , Red Hat

Luc Boutier (luc.boutier@fastconnect.fr), FastConnect

Matt Rutkowski (mrutkows@us.ibm.com), IBM

Richard Probst (richard.probst@sap.com), SAP AG

Sahdev Zala (spzala@us.ibm.com), IBM

Stephane Maes (stephane.maes@hp.com), HP

Thomas Spatzier (thomas.spatzier@de.ibm.com), IBM

Travis Tripp (travis.tripp@hp.com), HP

Appendix H. Revision History

Revision	Date	Editor	Changes Made
38	2014-03-20	Matt Rutkowski, IBM	Updated to OASIS latest template

