TOSCA Functional Architecture
“Pure” TOSCA Orchestration

Orchestration functionality is based exclusively on features in the TOSCA language

– Without any built-in knowledge about domain-specific types
– Without any built-in knowledge about platform-specific implementations.
– Without any built-in knowledge about external resources
Uebicity Support for TOSCA Profiles

Profiles are collections of domain-specific type definitions
- A TOSCA Version 2.0 feature
- TOSCA Profiles advertise a well-known profile name
- Ubicity supports TOSCA Profiles using the TOSCA v1.3 namespace keyword

TOSCA Profiles can include
- Type definitions
- Artifacts that provide implementations for interface operations for those types
- Service Templates for substitution mapping

Profiles must be onboared into the Ubicity Orchestrator
- Profiles are packaged as CSARs
- Must be onboared into Profile Catalog before they can be used
- By default, only TOSCA Built-In Type “profile” is availabl

Service templates import profiles using well-known profile name

$ ubicity profile list
$ ubicity profile add <csar_name>
$ ubicity profile show <profile_id>
$ ubicity profile delete <profile_id>
**Ubicity Service Model Catalog**

**TOSCA CSAR files are onboarded into a Service Catalog maintained by the orchestrator**
- To allow service ordering and instantiation from the catalog.

**Deploying Services requires resources from inventory**
- Resources are also modeled using TOSCA Service Templates
- Ubicity stores Resource Model Templates in its Service Catalog

```
$ ubicity catalog list
$ ubicity catalog add <csar_file_name>
$ ubicity catalog show <template_id>
$ ubicity catalog delete <template_id>
```
**Ubicity Support for Onboarding External Resources**

**Ubicity Orchestrator** has no built-in knowledge about external resources on which services can be deployed

- Orchestrator is made aware of external resources by onboarding resources in inventory

**External resources are modeled using service templates**

- Node templates that represent external resource components are “information-only nodes”
- No difference in inventory between pre-existing resources and instantiated services

**Allows the use of interface operations and notifications for managing external resources using TOSCA**

```
$ ubicity resource list
$ ubicity resource add <name> <template_id> \
       -i <input_file_name>
$ ubicity resource show <resource_id>
$ ubicity resource delete <resource_id>
```
Deploying Services

Services are deployed using templates stored in the Service Catalog
- Identified using Service ID

Service templates define a number of service instance-specific input parameters (“Service Characteristics”)
- That need to be provided at service instantiation time
- And provide initial values for “templatized” node and relationship properties.

$ ubicity service list
$ ubicity service add <name> <template_id> \\
   -i <input_file_name>
$ ubicity service show <service_id>
$ ubicity service delete <service_id>
Example Service—Deploying Kubernetes Clusters
Abstract Kubernetes Cluster Service Template

Kubernetes Cluster Service Template

- **master node**
  - JoinedTo
  - LinksTo worker node

- **host network**
  - LinksTo worker node

- **worker node 1**
- **worker node N**
Substitution Mapping—Implementing Abstract Nodes

Master Node Service Template

- calico
- kubernetes
  - master
  - master_docker
  - master_host

Master Properties
- cluster_name
- name

Master Capabilities
- master

Master Interfaces
- Standard
  - create
  - configure
  - start

Master Requirements
- Linkable
  - cluster_network

maps

substitutes
Substitution Mapping—Platform-Specific Implementations

- **Master Service Template**
  - k8s
  - master_host
  - region

- **AWS Compute Service Template**
  - compute
  - network interface
  - region

- **Openstack Compute Service Template**
  - compute
  - port
  - region
Ubicity Orchestrator Fulfills Dangling Requirements at Deployment Time

Service template specifies requirements (including node filters)

Entities in Active Inventory expose capabilities

Orchestrator matches capabilities to requirements

Active Inventory:
- AWS Region
- K8s Cluster
- Bare Metal
- Edge device
- Router/Switch
- ROADM

Ubicity Orchestrator
Using TOSCA to Deploy Kubernetes Services—O-RAN
Updating Services—Let’s Move the Cluster First

Services are updating by providing updated input values
– Since “inputs” are the only externally-visible modifiable parameters of the service

Orchestrator calculates differences between the running service instance and the updated service instance
– Schedule “reconfiguration” or “deletion” of existing service components
– Schedule “creation” of new service components
– Re-do substitution mappings where necessary
– Re-do requirement fulfillment where necessary

$ ubiquity service update <service_id> \
    -i <updated_input_file_name> \
    [-n <updated_name>]

Copyright © 2021 Ubicity Corp.
Benefits of Using TOSCA for O-RAN

Single technology for orchestrating and managing all aspects of O-RAN deployments
- (virtualization) infrastructure
  - Including to the Edge
- Management plane
  - SMO
- Control Plane
  - Non-RT-RIC, RIC
- Data plane
  - CU, DU, RU
- xApps and rApps