Use Case: Requirement and Capability Matching using Capability Properties

Submitted by: Travis Tripp (HP)

TOSCA has the notion of using requirements and capabilities to support matching of types as well as for dynamic substitution.

I would postulate that in a declarative language, that I declare requirements that I need to be fulfilled.  The requirements are based on a shared understanding of known capabilities. The system would then allow different providers to be loaded and used without affecting the supported capabilities. The provider must declare the capabilities that it provides.  In this manner, there is a separation between the notion of API and the implementation of the API.  In this manner, there may be different providers for the same capabilities in different clouds. For example, I may declare that I need a virtual machine.  I may specify the computer requirements of that virtual machine in a non-binding manner (flavor name is binding, whereas the amount of memory or CPUs is not).  It would then be up to the system to load the provider for the cloud and for the provider to translate the requirements into the specific details of that cloud (choose the right flavor based on the normative requirement properties). In order to have such a system, there would need to be a shared dictionary of capability types.  They types in such a dictionary do not need to be normative across the industry, but do need to be understood at least by the container.

The below diagram which Thomas presented at the OpenStack summit highlight this concept:



As a part of matching, it isn’t just a Capability Type name, it is also matching on the properties of that capability type and using the properties of the capability to drive scripting.

Problem: I spent time trying to map this into TOSCA XML and it wasn’t clean or even clear if requirement and capability matching supported specification of requirements like I state below.

Example 1: If I have an application that requires MySQL, I may require a specific version. Additionally, when I script against it, I will need to know values like the port that the MySQL is running on. Therefore, when I have a capability of MySQL, properties have more than one purpose (matching and scripting).

Example 2: If I have an application that requires compute resources, I will require some minimum amount of resources to be provided. I need a way to specify that.

To illustrate the above examples, I am providing some JSON based sytanx below, since it is simpler and more compact. Note, that rather than an external XSD for property definitions that I used inline JSON schema syntax.

Following the examples, I will expand the use case.

Example 1

|  |  |  |
| --- | --- | --- |
| Capability Type Definition for MySQL. | A type that declares the capability. | And finally a declaration of a requirement for the capability: (I need MySQL version 5 to 5.6) |
| { "name":"MySQL", "namespace":"http://sample.com", "description":"My SQL Database", "derived\_from":{ "name":"Database", "namespace":"http://sample.com" }, "properties":{ "version":{ "type":"string", "description":"The version", "minLength":0, "maxLength":25, "pattern":"^(?:(\\d+)\\.)?(?:(\\d+)\\.)?(\\\*|\\d+)$" }, "port":{ "type":"number", "description":"The port", "minimum":1024, "maximum":49141 } }} | {"capabilities": [ { "name": "My SQL Instance 1", "capability\_type\_namespace": "http://www.sample.com/", "capability\_type\_name": "MySQL", "properties": [ { "version" : "5.5" }, { "port" : 3606 } ] } ]} | {"requirements": [ { "name": "MySQL requirement", "capability\_type\_namespace": "http://www.sample.com/", "capability\_type\_name": "MySQL", "property\_criteria": [ { "name": "Version", "operators" : [ {"greater\_than\_or\_equal\_to" : 5.0}, {"less\_than\_or\_equal\_to": 5.6} ] } ] } ]} |

Example 2: A requirement for Compute Resources (not providing example of corresponding Capability Type)

{

"requirements": [

 {

 "name": "minimunHostServerRequirement",

 "capability\_type\_namespace": "http://www.sample.com/",

 "capability\_type\_name": " ComputeResources",

 "property\_criteria": [

 {

 "name": "NumCpus",

 "operators" : [

 {"greater\_than\_or\_equal\_to" : 1} ,

 {"less\_than\_or\_equal\_to" : 24}

 ]},

 {

 "name": "CPUArchitecture",

 "operators" : [

 { "in" : "[32 Bit,64 Bit]"}

 ]},

 {

 "name": "MemoryInMB",

 "operators" : [ {"greater\_than\_or\_equal\_to" : "1"}]

 },

 {

 "name": "DiskInGB",

 "operators" : [{"equal" : "10"}]

 },

 {

 "name": "Foo",

 "operators" : [{"like": "%Bar"}]

 }

 ]

}

]}

Secondarily, this becomes more important when you get into the notion of Abstract Node Type replacement based on Server Template replacement. Reference Section 3.5 of the 1.0 Specification for full text:

*Service Templates can be based on and built on-top of other Service Templates based on the concept of Requirements and Capabilities introduced in the previous section. For example, a Service Template for a business application that is hosted on an application server tier might focus on defining the structure and manageability behavior of the application itself.*

*…. Thus, a substitution with any Service Template that has the same boundary definitions as a certain Node Template in one Service Template becomes possible, allowing for a flexible composition of different Service Templates. This concept also allows for providing substitutable alternatives in the form of Service Templates.*



Figure : Service Template Composition

With the above, according to the spec it is reliant upon boundary definitions which are defined on the service template. If for, example, I am doing application to infrastructure matching and in one case I am looking for infrastructure that provides high availability vs infrastructure that is for testing, the substituted service template will differ based upon the set of requirements needing to be fulfilled. In effect, you may have an application template with 1..N infrastructure templates. To create different deployments, you may have a wrapper deployment template around the app and infra templates to represent a deploy to tiny vs large infrastructure and the engine would have to expand both templates at runtime and appropriately match the deployment specs.

 