TOSCA Processing

Parser/Checker

History

- More than 2 years ago, part of Domain 2.0 effort within AT&T (SDN, network automation, etc), Intelligent Service Composition (ISC) platform
- Multiple aspects:
    - Model driven: resources/services/products
        - Modeling language (TOSCA)
        - Mode catalog
    - Interactive service composition (Composition UI, Composition Server)
    - Deployment (MSO interaction)

Scope

- As generic as it gets
  - Not bound to a particular type schema/profile
  - No particular target orchestrator.
- Generic processing tools and generic composition
  - Composition: the graphical assembly of node templates into a topology
  - Generic model catalog (TOSCA model persistence) and catalog API
  - Limited to TOSCA yaml profile
  - Java environment

Goal

- Ensure that a (set of) TOSCA template(s) is conform to a correctly defined type system
  - Avoid errors at more expensive later processing stages.
  - Pre-requisite to further processing: UI, persistence
- Generate an intermediate form that could facilitate further processing
- Extensibility
  - Accommodate extensions to the standard and checks pertinent to these extensions

Limitations

- We were not building topology templates and most of the time not even topology models: designs
- TOSCA limitations: business constraints that cannot be expressed in TOSCA
  - topology graph connectivity constraints
  - more complex data dependencies

The checker

- Validate yaml document
  - might seem ordinary but watch for streaming documents (multiple documents within one file), yaml anchors, etc
- Syntax check
  - 2 grammars, 1.0 and 1.1, with the possibility of handling a mix of documents version wise
  - Accept the ‘short forms’
    - We declare the short forms within the grammar
    - We build a canonical form from which the shorts forms are eliminated (so further processing steps do not need to handle them)
- Process the entire document tree specified through import statements

Syntax

- Grammar written in Yaml and syntax check performed through a modified version of the kwalify library

```
_requirement_assignment_definition: &requirement_assignment_definition
  type: map
  name: requirement_assignment_definition
  short: node
  mapping:
    capability:
       required: no
        type: str
    node:
        required: no
        type: str
    relationship:
       required: no
       type: map
           short: type
           mapping:
               "type":
                     required: no
                     type: str
                properties:
                     required: no
                     type: map
                     mapping:
                           =:
                              name: property_assignment
                              type: any
                interfaces:
                     required: no
                     type: map
                     mapping:
                          =:
                            *template_interface_definition
           node_filter:
               required: no
               <<: *node_filter_definition
```

Checking

- type hierarchy checks for all constructs
  - valid re-definitions
    - from relatively simple (properties) to rather complicated (interface opearations)
- valid type references: all referenced types are pre-declared (as super-types, as valid source types, etc)
- templates respect their respective type definition
  - example: check type of interface operation inputs
  - other references: capabilities and requirements in substitution mappings
- data checks: assignments match the type specification, function argument compatibility (for built-in functions), constraints matching, ..

Output

- Error reporting
  - Differs depending on the stages
    - Document position indication during parsing
    - Document path and rule during syntax check
    - Document path during checking
- Catalog
  - No explicit representation of TOSCA constructs, offers a query interface with results being exposed as common Java types: maps, lists, ..
    - Experimented with a proxy based approach
  - Apache jxpath based processing (xquery against the in-memory representation of a set of TOSCA templates).

API,CLI and Service

- We offer a simple API that allows access to any stage of the checker
  - One can build an in-memory representation of a TOSCA document and skip the yaml parsing (we use it in our ‘recycler’).
  - Re-usable Catalog (hierarchical Catalogs)
    - One or more documents can be processed and the resulting Catalog be preserved and used for later processing of other documents using the previous catalog as ‘base’ catalog
      - Used in checker service (REST API) where a client can first submit a schema and subsequently check templates against that schema
- Simple CLI for TOSCA yaml documents checking.
- Simple REST service layer on top of checker API

More processing

- Test if a template is ‘complete’, i.e. it constitutes a topology template or self contained (all requirements can be satisfied within the topology model)
- Persistence: we store TOSDA model in a graph database (neo4j) and offer a retrival API
    - Used by the composition engine
- Validation engine
    - Catalog anad topology graph exposed to Java scripts implementing validation rules