Modeling API

3 TOSCA Parsers

There are three TOSCA parsers submited into modeling project, their API are listed below:

1. NFV Tosca Parser API: (Python Lib + CLI + REST API)

Implementation of nfv-toscaparser derived from openstack tosca parser is based on the following OASIS specification:

TOSCA Simple Profile YAML 1.2 Referecne http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.2/TOSCA-Simple-Profile-YAML-v1.2.html

TOSCA Simple Profile YAML NFV 1.0 Referecne http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/tosca-nfv-v1.0.html

- **1) CLI**
  - Installation nfv-toscaparser

Firstly, uninstall the previous verson of toscaparser

```
pip uninstall -y tosca-parser
```

Then install nfv-toscaparser :

```
pip install --pre -U nfv-toscaparser
```

  - Use cli, which is used to validate tosca simple based service template. It can be used as:

```
tosca-parser --template-file=<path to the YAML template>  [--nrpv]  [--debug]

tosca-parser --template-file=<path to the CSAR zip file> [--nrpv]  [--debug]

tosca-parser --template-file=<URL to the template or CSAR>  [--nrpv]  [--debug]
```

options:

```
--nrpv Ignore input parameter validation when parse template.

--debug debug mode for print more details other than raise exceptions when errors happen
```

  - The parse output will display in STDOUT, including nodes name, inputs and outputs

- **2) Library(Python)**
  - Installation nfv-toscaparser, same as CLI;
  - Use api, which is used to parse and get the result of service template. it can be used as

```
tosca=ToscaTemplate(path=None, parsed_params=None, a_file=True, yaml_dict_tpl=None,
                    sub_mapped_node_template=None,
                    no_required_paras_valid=False, debug=False )
```

  - Access members(including method or data) in tosca.

- **3) REST API**
  - Installation nfv-toscaparser microservice, and run it;
  - Use Restful API
    - List template versions
      - **PATH:** /v1/template_versions
      - **METHOD:** GET
      - **Decription**: Lists all supported tosca template versions.
      - **Response Codes**

Success

Code	Reason
`200 - OK`	Request was successful.

Error

Code	Reason
`400 - Bad Request`	Some content in the request was invalid.
`404 - Not Found`	The requested resource could not be found.
`500 - Internal Server Error`	Something went wrong inside the service. This should not happen usually. If it does happen, it means the server has experienced some serious problems.

Request Parameters

Name	Type	Description
No		

Response Parameters

Name	Type	Description
template_versions	array	A list of tosca template version object each describes the type name and version information for a template version.

    - **Validates a service template**
      - **PATH:** /v1/validate
      - **METHOD:** POST
      - Decription: Validate a service template and return the result
      - **Response Codes**

Success

Code	Reason
`200 - OK`	Request was successful.

Error

Code	Reason
`400 - Bad Request`	Some content in the request was invalid.
`500 - Internal Server Error`	Something went wrong inside the service. This should not happen usually. If it does happen, it means the server has experienced some serious problems.

Request Parameters

Name	Type	Description

environment (Optional)	object	A JSON environment for the template service.
environment_files (Optional)	object	An ordered list of names for environment files found in the `files` dict.

files (Optional)	object	Supplies the contents of files referenced in the template or the environment.
		
		The value is a JSON object, where each key is a relative or absolute URI which serves as the name of a file, and the associated value provides the contents of the file. The following code shows the general structure of this parameter.
		
		`{ ...`
		`"files": {`
		`"fileA.yaml": "Contents of the file",`
		`"file:///usr/fileB.template": "Contents of the file",`
		`"http://example.com/fileC.template": "Contents of the file"`
		`}`
		`...`
		`}`

ignore_errors (Optional)	string	List of comma-separated error codes to ignore.
show_nested (Optional)	boolean	Set `true` to include nested template service in the list.

template (Optional)	object	The service template on which to perform the operation.
		
		This parameter is always provided as a `string` in the JSON request body. The content of the string is a JSON- or YAML-formatted service template. For example:
		
		`"template": {`
		`"tosca_definitions_version": "tosca_simple_yaml_1_0",`
		
		`...`
		`}`
		
		This parameter is required only when you omit the `template_url` parameter. If you specify both parameters, this value overrides the`template_url` parameter value.

template_url (Optional)	string	A URI to the location containing the service template on which to perform the operation. See the description of the `template` parameter for information about the expected template content located at the URI. This parameter is only required when you omit the `template` parameter. If you specify both parameters, this parameter is ignored.

Request Example

```
{
    "template_url": "/PATH_TO_TOSCA_TEMPLATES/HelloWord_Instance.csar"
}
```

Response Parameters

Name	Type	Description
Description	string	The description specified in the template.
Error Information (Optional)	string	Error information

- **Parse a service template**
  - **PATH:** /v1/parse
  - **METHOD:** POST
  - Decription: Parse a service template and return the parsed info
  - Response Code: same as "**Validates a service template**"
  - Request Parameters: same as "**Validates a service template**"
  - **Response Parameters**

Name	Type	Description
Description	string	The description specified in the template.
Input parameters	object	Input parameter list.
Service Template	object	Service template body
Output parameters	object	Input parameter list.
Error Information (Optional)	string	Error information

2. Apache ARIA-TOSCA(Python + CLI + REST API):

Complete reference: http://ariatosca.incubator.apache.org/docs/html/index.html#

TOSCA Simple Profile 1.0 Referecne http://ariatosca.incubator.apache.org/docs/html/aria_extension_tosca.simple_v1_0.html

TOSCA Simple Profile 1.0 NFV Referecne http://ariatosca.incubator.apache.org/docs/html/aria_extension_tosca.simple_nfv_v1_0.html

Install ARIA from pypi

```
[root@localhost] # pip install apache-ariatosca
```

```
[root@488GTB ariatosca]$ aria --help
Usage: aria [OPTIONS] COMMAND [ARGS]...

ARIA's Command Line Interface.

To activate bash-completion run::

eval "$(_ARIA_COMPLETE=source aria)"

ARIA's working directory resides by default in "~/.aria". To change it,
set the environment variable ARIA_WORKDIR to something else (e.g.
"/tmp/").

Options:
-v, --verbose Show verbose output; you can supply this up to three times
(i.e. -vvv)
--version Display the version and exit
-h, --help Show this message and exit.

Commands:
executions Manage executions
logs Manage logs of workflow executions
node-templates Manages stored service templates' node...
nodes Manage services' nodes
plugins Manage plugins
reset Reset ARIA working directory
service-templates Manage service templates
services Manage services
workflows Manage service workflows
```

3. Java based Tosca Parser API: (Java)

a. Java API

The Java API is described in detail in the javadoc. The main classes are part of the package *org.onap.tosca.checker.* The entry point is the *Checker* class, the TOSCA service templates are represented by the *Target* class an the results of the processing are represented by *Report* (contains errors associated with a Target) and *Catalog* (if successful, i.e. empty report, contains an index of TOSCA constructs).

Common usage:

```
try {
  Catalog cat = Checker.check(uri_or_path_of_service_template);

  for (Target t: cat.targets()) {
    System.err.println(t.getLocation() + "\n" + t.getReport());
  }

}

catch (Exception x) {
  /* error handling */
}
```

The API offers options for custom TOSCA dialects (extended grammars), pluggable validation and consistency checking, different post-processing approaches.

b. CLI tool

The Maven based build assembles a 'all-in-one' jar (in the checker sub-project) with a convenient entry point for cli based validation:

```
java -jar Checker-0.0.1-SNAPSHOT-jar-with-dependencies.jar path_to_service_template_file
```

c. REST service

The same functionality is offered as a REST-based service. The service sub-project assembles a spring framework based REST interface with a built in default configuration. The package is found in the service sub-project. To start the service:

```
java -jar Service-0.0.1-SNAPSHOT.jar
```

Sample client usage (windows power shell):

```
PS C:\Users\serban> Invoke-WebRequest -Uri http://localhost:8080/check_template/ -Method Post -ContentType application/json -InFile C:\src\asc-tosca\DCAE\v3\Database\Postgres-generic\schema.yaml
```

or with stateful 'namespaces':

```
PS C:\Users\serban> Invoke-WebRequest -Uri http://localhost:8080/check_template/database/schema.yaml -Method Post -ContentType application/json -InFile C:\src\asc-tosca\DCAE\v3\Database\Postgres-generic\schema.yaml

PS C:\Users\serban> Invoke-WebRequest -Uri http://localhost:8080/check_template/database -Method Post -ContentType application/json -InFile C:\src\asc-tosca\DCAE\v3\Database\Postgres-generic\template_postgres.yaml
```

The first call creates the 'database' namespace and registers the submitted template as 'schema.yaml'. The second call submits a template as part of the same namespace containing an import of schema.yaml (from the first call); this mechanism allows for validation of multiple service templates with the same schema/type system.

For more details on the different REST API usages see the service sub-project documentation.