ONAP

OPEN NETWORK AUTOMATION PLATFORM

Controller Design Studio (CDS)

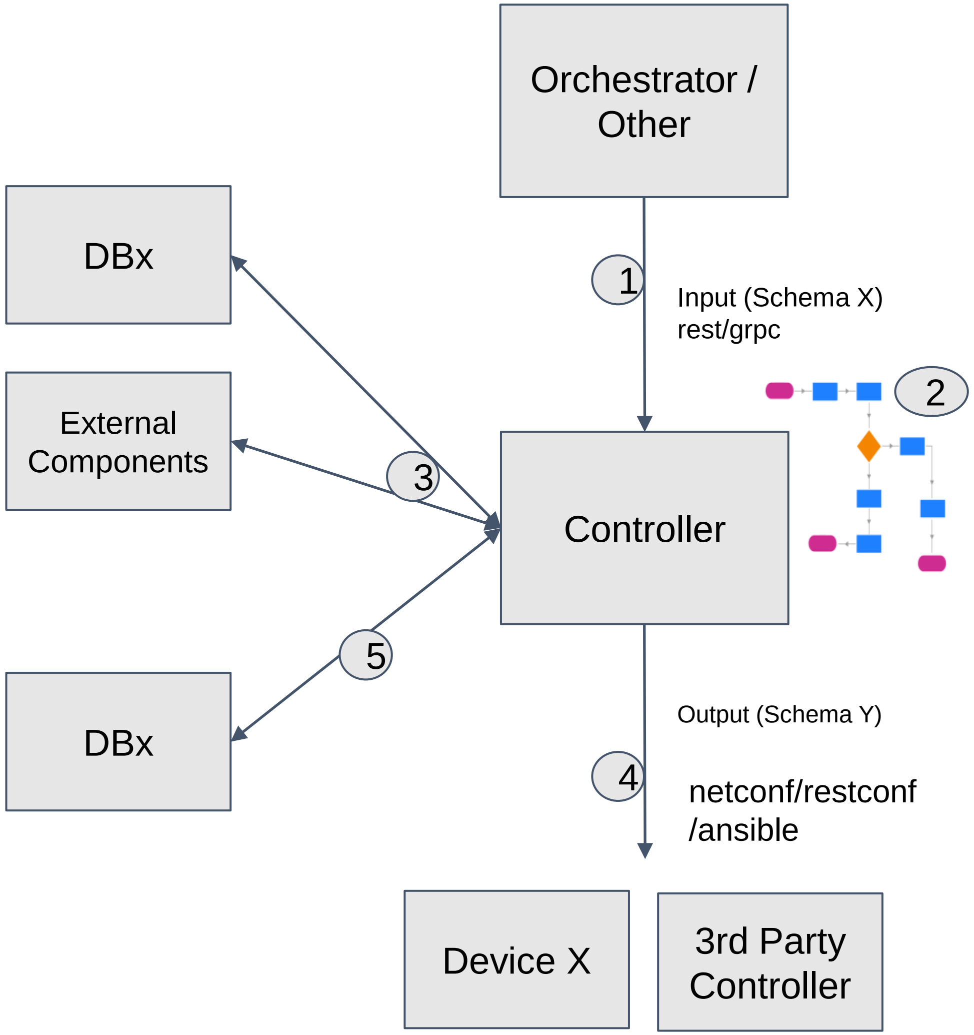
Agenda

- ➢ CDS Overview
- ➢ Architecture
- ➢ Data Flow
- ➢ API + External Interactions
- ➢ Modeling Concepts + CBA Format
- ➢ Installation, Design & Distribution
- ➢ Use-cases Involvement
- ➢ Code Summary
- ➢ SDNC vs CDS

CDS Overview: Model driven, self-service approach

https://www.aarnanetworks.com/post/2017/11/02/the-magic-of-model-driven-design-in-onap

What Typically a Controller Does



Controller Objective: Manage device/network etc.

Typical Controller Flow

1. Receives an Input request from orchestration/others and validates the same.
2. Start executing a workflow to process this request.
3. As part of this workflow queries DB using input parameters or calls other external components to get additional data - All this data is needed either for further workflow logic or to form the Output to be sent to Device/ 3rd party controller.
4. Configure device/3rd party controller.
5. Based on response, store it in DB, respond to orchestrator

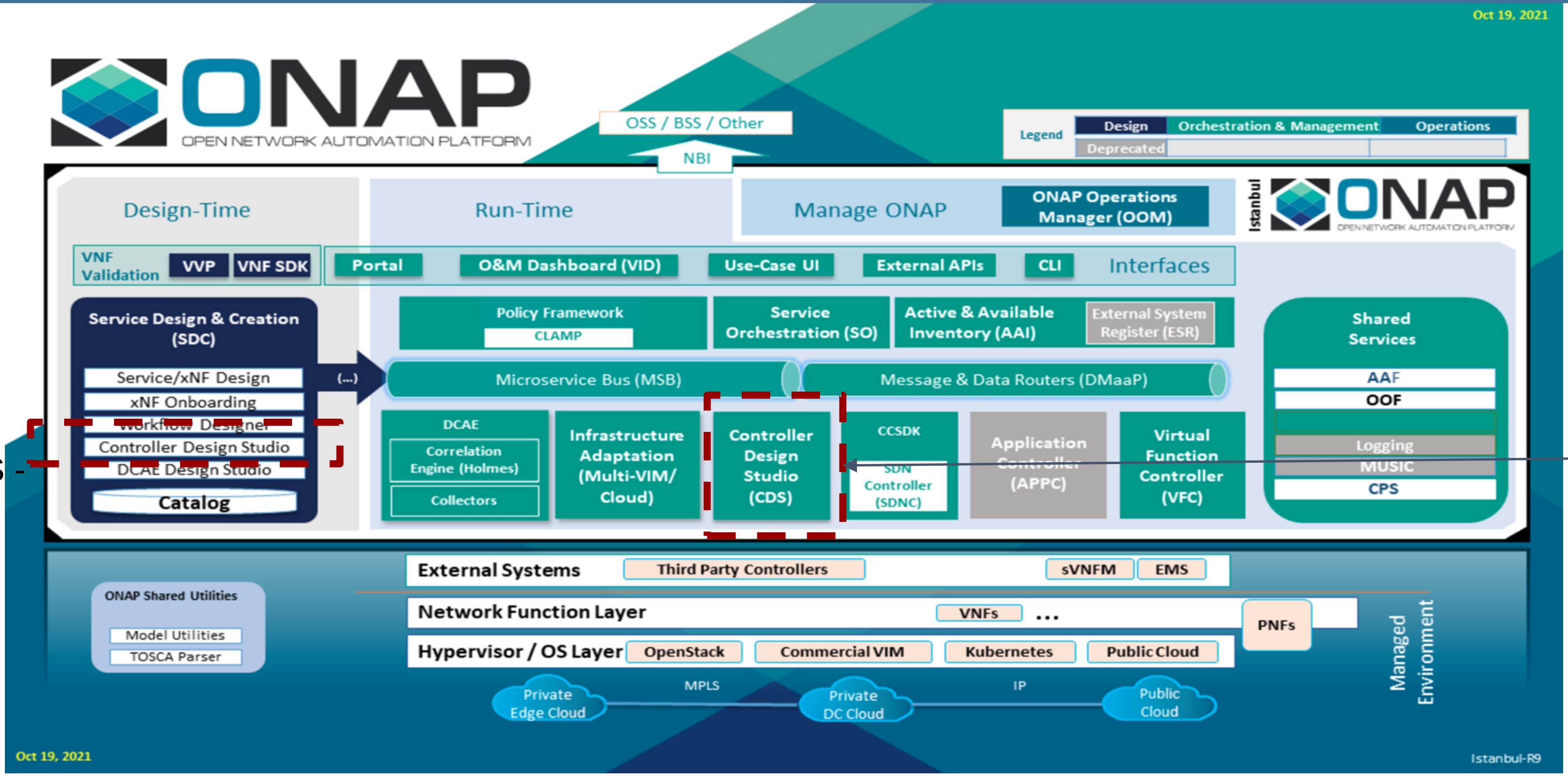
Point to Note:

1. Orchestrator usually has abstract implementation independent of usecase/service. It is controller where usecase/service specifics are pushed. So all the controller steps are specific to a service/usecase.
2. **Above point demands controller architecture to be flexible for supporting new emerging services with ease at Runtime.**

CDS provides model-driven self-service approach for controllers for above mentioned requirement:

- **Self-Service: Users**, not just programmers, can **reconfigure** the **software system** as needed to meet customer/usecase requirements.
- **Model Driven**: To accomplish this goal, the system is built around **models** that **dictates how** the system operates. Users merely need to change a model to change how a service operates.
- Model is nothing but combination of multiple smaller level **re-usable LEGO blocks** (functions) along with their input and output.

CDS Overview



Design Time component of CDS - **CDS UI**

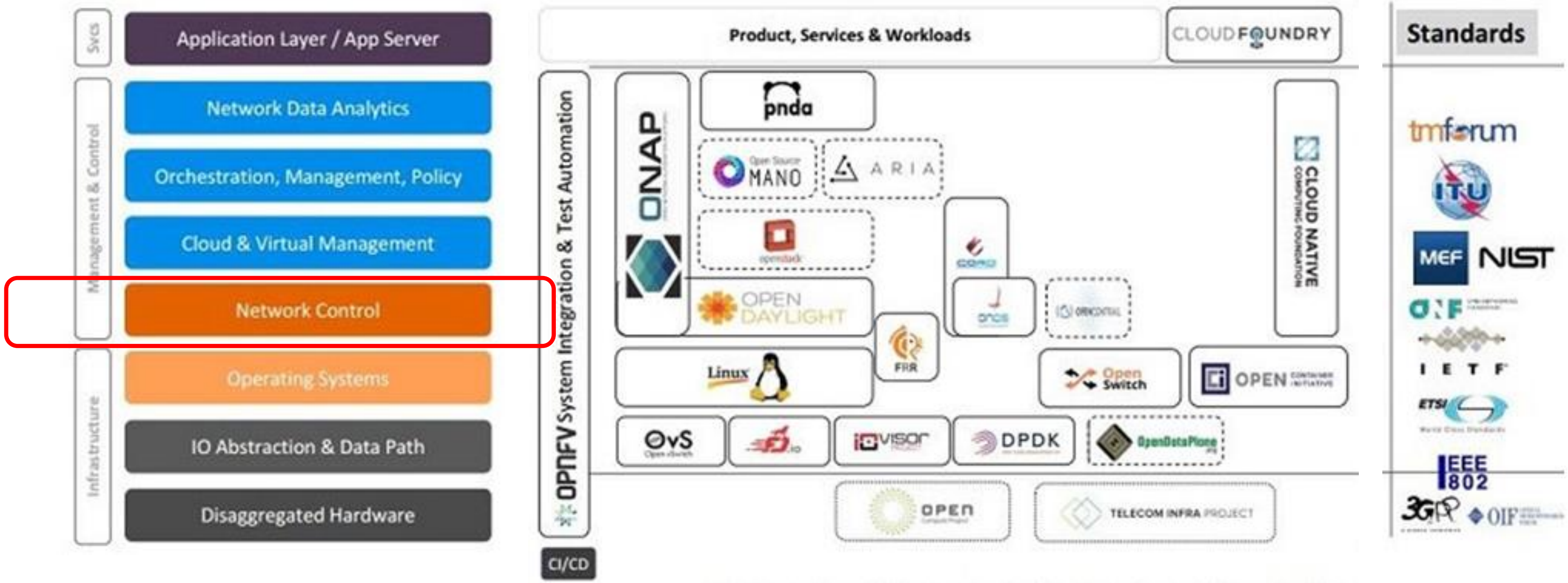
Runtime component of CDS - **CDS Blueprint Processor**

- ★ CDS is **one of the controller type** in ONAP. It can work independently, even it can work together with SDNC.

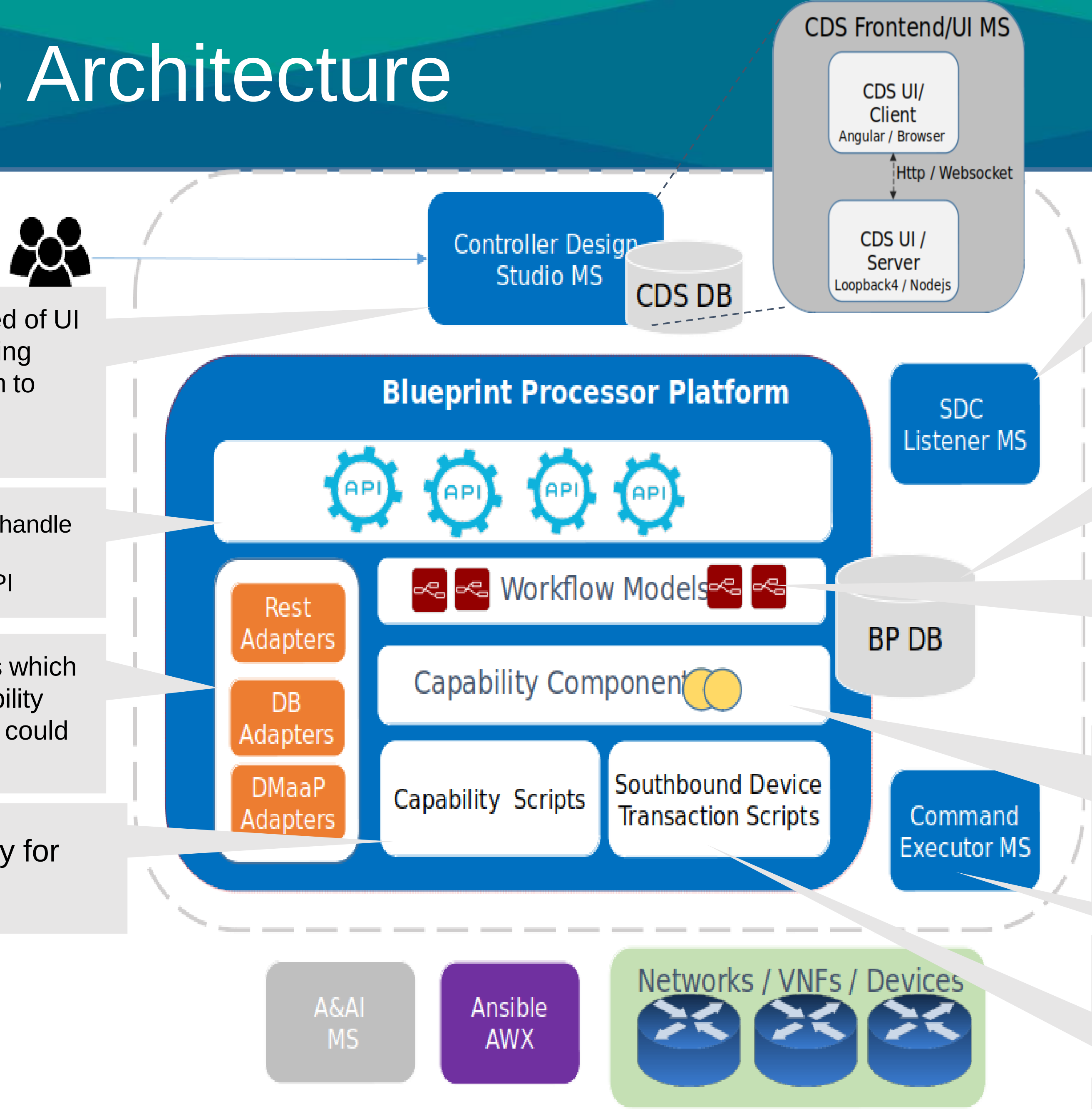
- ★ CDS is designed to provide **model-driven, self-serve approach**, which means that users, not just programmers, can reconfigure the software system as needed to meet customer/usecase requirements. To accomplish this goal, the system is built around models that dictates how the system operates. Users merely need to change a model to change how a service operates.

- ★ CDS has a both **design time** and **run time** activities; during design time, **Designer** can **define** what **actions** are required for a given service, along with anything comprising the action. The design produce a CBA Package . Its **content** is driven from a **catalog** of **reusable data dictionary** and **component**, delivering a reusable and simplified **self service** experience.

Inter-Project Collaboration



CDS Architecture



Designer comprised of UI & backend. Providing graphical approach to design & distribute blueprint.

NB API Handlers to handle Self Service API BP/Model related API

Common Adapters which are used by Capability Components. Also could be used by scripts

Re-usable library for usage by script

Listens to CBA published by SDC and give it to Blueprint processor Platform for storage

MariaDB used to store Blueprints, resources, models etc.

Workflow Service, Only DG is supported currently

Reusable Functions like

- script-executor
- resource resolution
- remote-ansible-executor
- netconf-executor

Python script runtime execution

Re-usable libraries for usage by scripts for device transaction

Software:

- Spring Boot Netty / GRPC (CB MS, BP MS)
- Loopback (CDS UI MS)
- Maria DB (CB MS, SDNC, CDS DB)

Frameworks:

- Spring Boot Webflux / GRPC
- Kotlin Coroutines, Scripting
- Loopback, Node, Angular

Technologies:

- Directed Graph (Micro Flows)
- Kotlin (Capability Components)
- Jython, Kotlin (Capability Scripts)
- Typescript, nodejs, Loopback (UI Server)

Modeling :

- JSON, Kotlin DSL (Blueprints)
- Open API (MS APIs)
- Velocity, Jinja(Config files)
- Proto (API)
- SQL (DB)

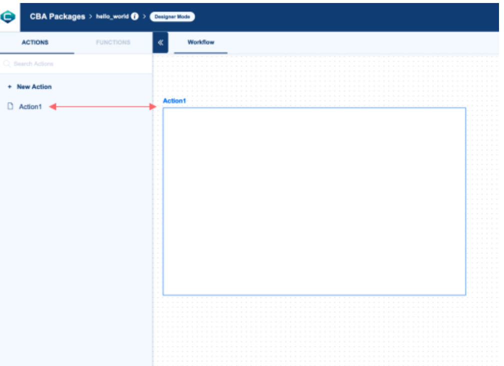
Adaptors:

- Netconf
- Restconf
- SSH
- GRPC
- Database

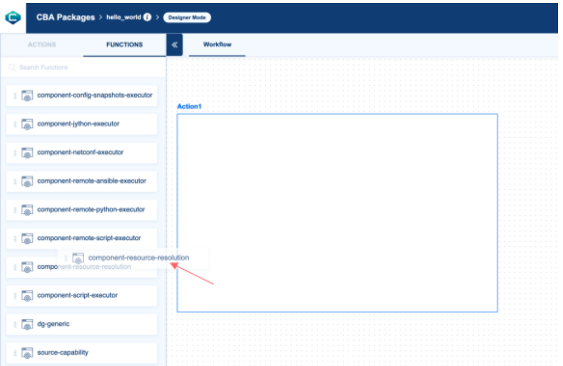
CDS Overview - Design/Runtime

Design Time - CDS UI

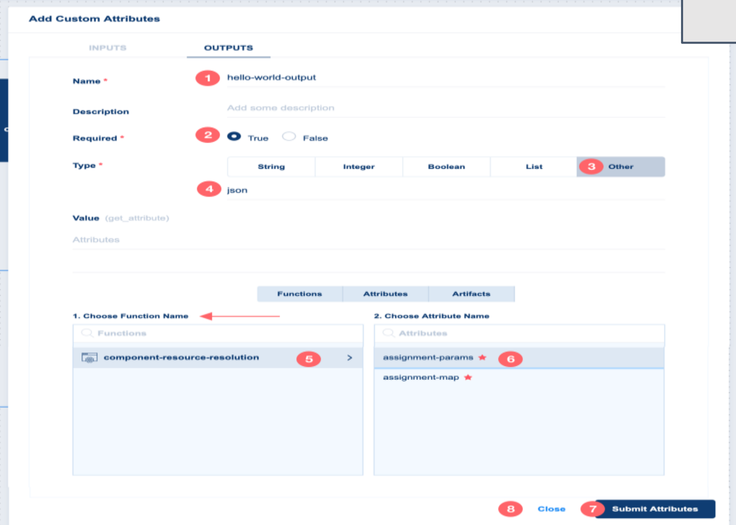
Create Controller Workflow



Drag Drop Existing Functions to workflow


CBA Packages > hello_world > Designer Mode
ACTIONS
FUNCTIONS
Workflow
Search Functions
component-config-snapshots-executor
component-jython-executor
component-netconf-executor
component-remote-ansible-executor
component-remote-python-executor
component-remote-script-executor
component-resource-resolution
component-resource-resolution
component-script-executor
dg-generic
source-capability
Action1

Define/Model Input/Output of workflow & also of every function



Add Custom Attributes

INPUTS | OUTPUTS

Name * (1) hello-world-output

Description: Add some description

Required * (2) ● True ○ False

Type * String | Integer | Boolean | List | (3) Other

(4) json

Value (get_attribute)

Attributes

Functions | Attributes | Artifacts

1. Choose Function Name — Functions: component-resource-resolution (5) >

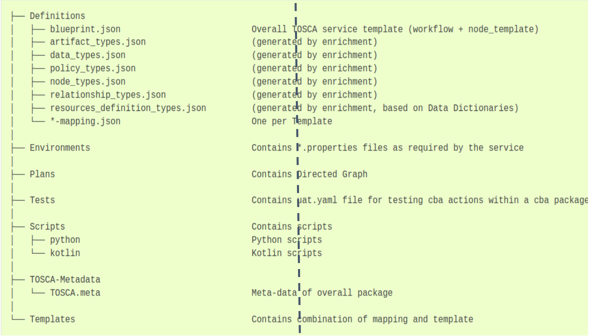
2. Choose Attribute Name — Attributes: assignment-params * (6); assignment-map *

(8) Close (7) Submit Attributes

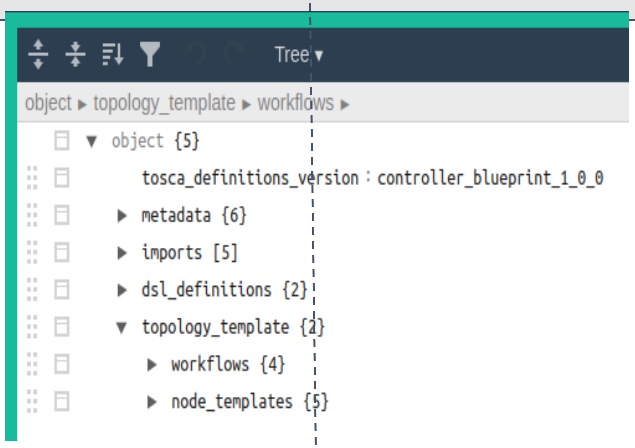
CDS UI

BP (CBA)

CBA Content



```
├── Definitions
│   ├── blueprint.json                          Overall TOSCA service template (workflow + node_template)
│   ├── artifact_types.json                     (generated by enrichment)
│   ├── data_types.json                         (generated by enrichment)
│   ├── policy_types.json                       (generated by enrichment)
│   ├── node_types.json                         (generated by enrichment)
│   ├── relationship_types.json                 (generated by enrichment)
│   ├── resources_definition_types.json         (generated by enrichment, based on Data Dictionaries)
│   └── *-mapping.json                          One per Template
│
├── Environments                                Contains *.properties files as required by the service
│
├── Plans                                       Contains Directed Graph
│
├── Tests                                       Contains uat.yaml file for testing cba actions within a cba package
│
├── Scripts                                     Contains scripts
│   ├── python                                  Python scripts
│   └── kotlin                                  Kotlin scripts
│
├── TOSCA-Metadata
│   └── TOSCA.meta                              Meta-data of overall package
│
└── Templates                                   Contains combination of mapping and template
```



```
object ▸ topology_template ▸ workflows ▸
▼ object {5}
    tosca_definitions_version : controller_blueprint_1_0_0
  ▶ metadata {6}
  ▶ imports [5]
  ▶ dsl_definitions {2}
  ▼ topology_template {2}
    ▶ workflows {4}
    ▶ node_templates {5}
```

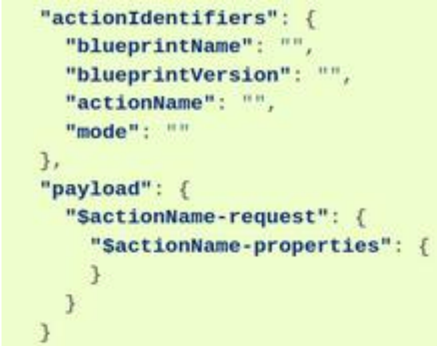
Runtime - CDS Blueprint Processor

SDNC

SO

Policy

Request



```
"actionIdentifiers": {
  "blueprintName": "",
  "blueprintVersion": "",
  "actionName": "",
  "mode": ""
},
"payload": {
  "$actionName-request": {
    "$actionName-properties": {
    }
  }
}
```

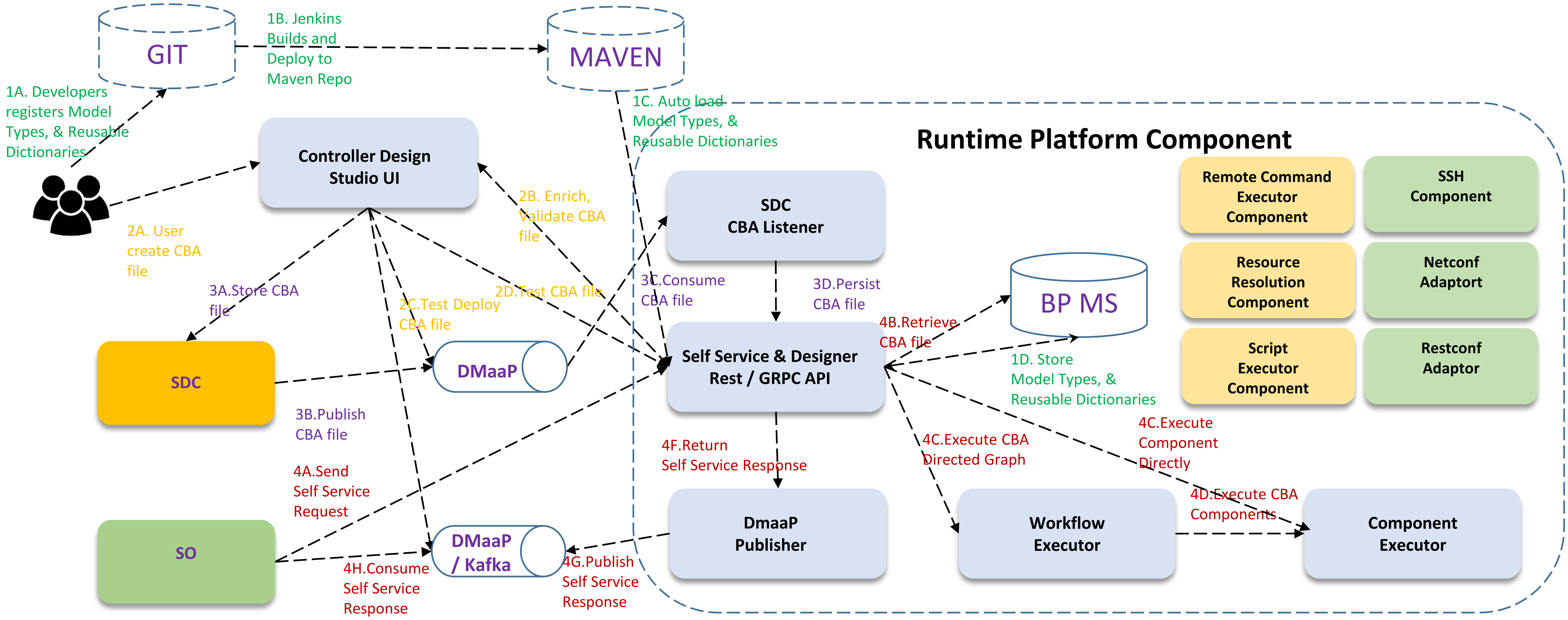
BP Processor

BP DB

Device

3rd Party Controller

CDS Data Flow



GIT

1B. Jenkins Builds and Deploy to Maven Repo

MAVEN

1A. Developers registers Model Types, & Reusable Dictionaries

1C. Auto load Model Types, & Reusable Dictionaries

Controller Design Studio UI

Runtime Platform Component

2A. User create CBA file

2B. Enrich, Validate CBA file

SDC CBA Listener

Remote Command Executor Component

SSH Component

Resource Resolution Component

Netconf Adaptort

Script Executor Component

Restconf Adaptor

3A.Store CBA file

2C.Test Deploy CBA file

2D.Test CBA file

3C.Consume CBA file

3D.Persist CBA file

BP MS

4B.Retrieve CBA file

1D. Store Model Types, & Reusable Dictionaries

SDC

DMaaP

Self Service & Designer Rest / GRPC API

3B.Publish CBA file

4F.Return Self Service Response

4C.Execute CBA Directed Graph

4C.Execute Component Directly

4A.Send Self Service Request

4D.Execute CBA Components

SO

DMaaP / Kafka

DmaaP Publisher

Workflow Executor

Component Executor

4H.Consume Self Service Response

4G.Publish Self Service Response

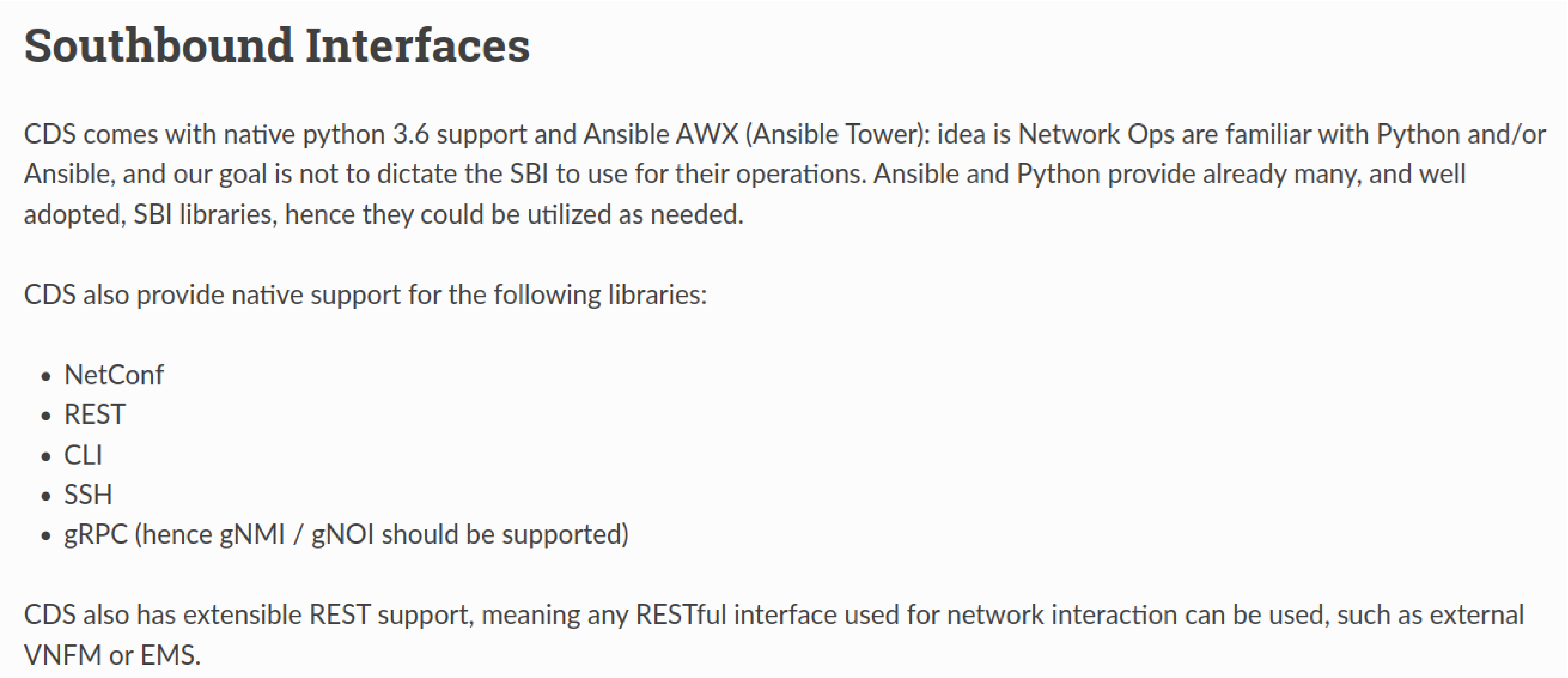
1 Starter/Bootstrapping Packs are loaded to DB during startup

2 User design CBA, Enrich & validate it

3 Validated & Enriched CBA distribution

4 Runtime Execution of CBA

CDS - SB Interfaces



Southbound Interfaces

CDS comes with native python 3.6 support and Ansible AWX (Ansible Tower): idea is Network Ops are familiar with Python and/or Ansible, and our goal is not to dictate the SBI to use for their operations. Ansible and Python provide already many, and well adopted, SBI libraries, hence they could be utilized as needed.

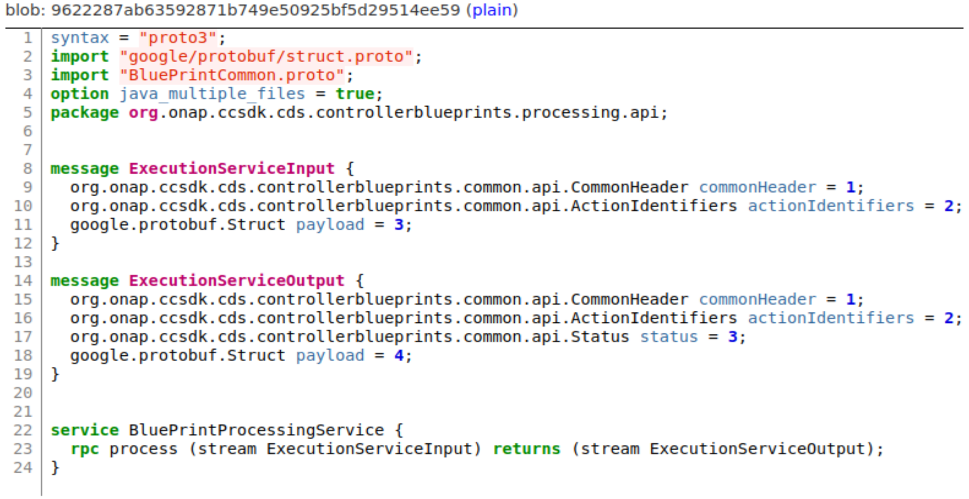
CDS also provide native support for the following libraries:

- NetConf
- REST
- CLI
- SSH
- gRPC (hence gNMI / gNOI should be supported)

CDS also has extensible REST support, meaning any RESTful interface used for network interaction can be used, such as external VNFM or EMS.

CDS API - Execution API

Execution Service API – Process a BP



blob: 9622287ab63592871b749e50925bf5d29514ee59 (plain)

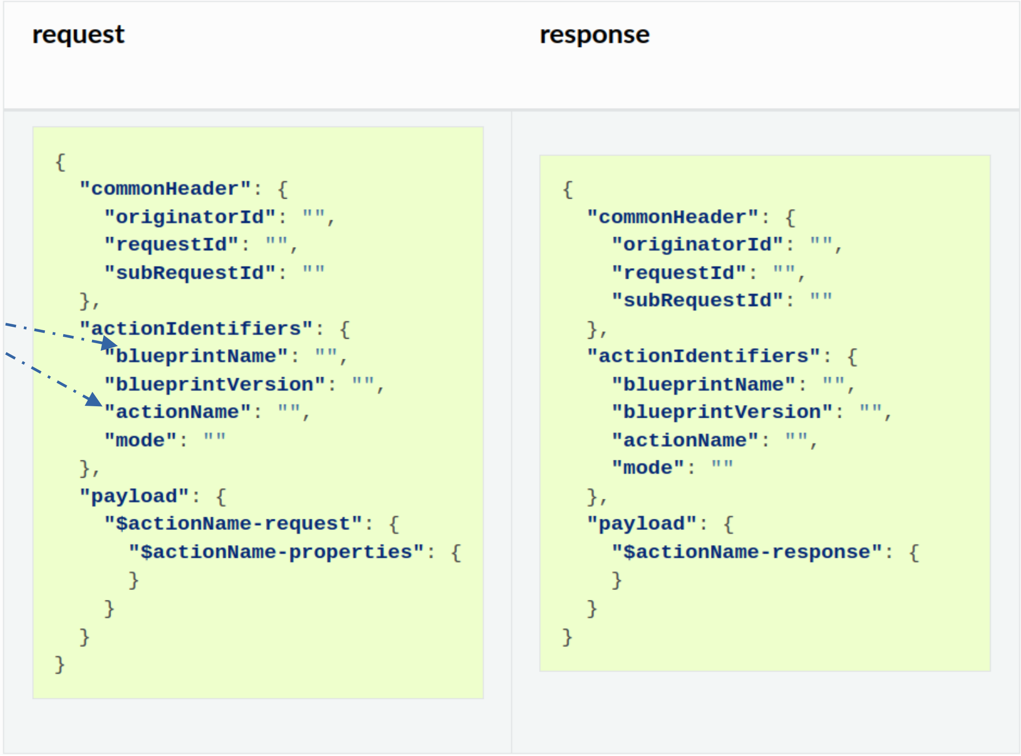
```
syntax = "proto3";
import "google/protobuf/struct.proto";
import "BluePrintCommon.proto";
option java_multiple_files = true;
package org.onap.ccsdk.cds.controllerblueprints.processing.api;


message ExecutionServiceInput {
  org.onap.ccsdk.cds.controllerblueprints.common.api.CommonHeader commonHeader = 1;
  org.onap.ccsdk.cds.controllerblueprints.common.api.ActionIdentifiers actionIdentifiers = 2;
  google.protobuf.Struct payload = 3;
}

message ExecutionServiceOutput {
  org.onap.ccsdk.cds.controllerblueprints.common.api.CommonHeader commonHeader = 1;
  org.onap.ccsdk.cds.controllerblueprints.common.api.ActionIdentifiers actionIdentifiers = 2;
  org.onap.ccsdk.cds.controllerblueprints.common.api.Status status = 3;
  google.protobuf.Struct payload = 4;
}


service BluePrintProcessingService {
  rpc process (stream ExecutionServiceInput) returns (stream ExecutionServiceOutput);
}
```

Execution API will execute a specific BP’s Action/Workflow



request

```
{
  "commonHeader": {
    "originatorId": "",
    "requestId": "",
    "subRequestId": ""
  },
  "actionIdentifiers": {
    "blueprintName": "",
    "blueprintVersion": "",
    "actionName": "",
    "mode": ""
  },
  "payload": {
    "$actionName-request": {
      "$actionName-properties": {
      }
    }
  }
}
```

response

```
{
  "commonHeader": {
    "originatorId": "",
    "requestId": "",
    "subRequestId": ""
  },
  "actionIdentifiers": {
    "blueprintName": "",
    "blueprintVersion": "",
    "actionName": "",
    "mode": ""
  },
  "payload": {
    "$actionName-response": {
    }
  }
}
```

Sample Request

curl --location --request POST '10.12.7.33:30699/api/v1/execution-service/process' \--header 'Content-Type: application/json' \--header 'Authorization: Basic Y2NzZGthcHBzOmNjc2RrYXBwcw==' \--header 'Content-Type: text/plain' \--data-raw 'data’

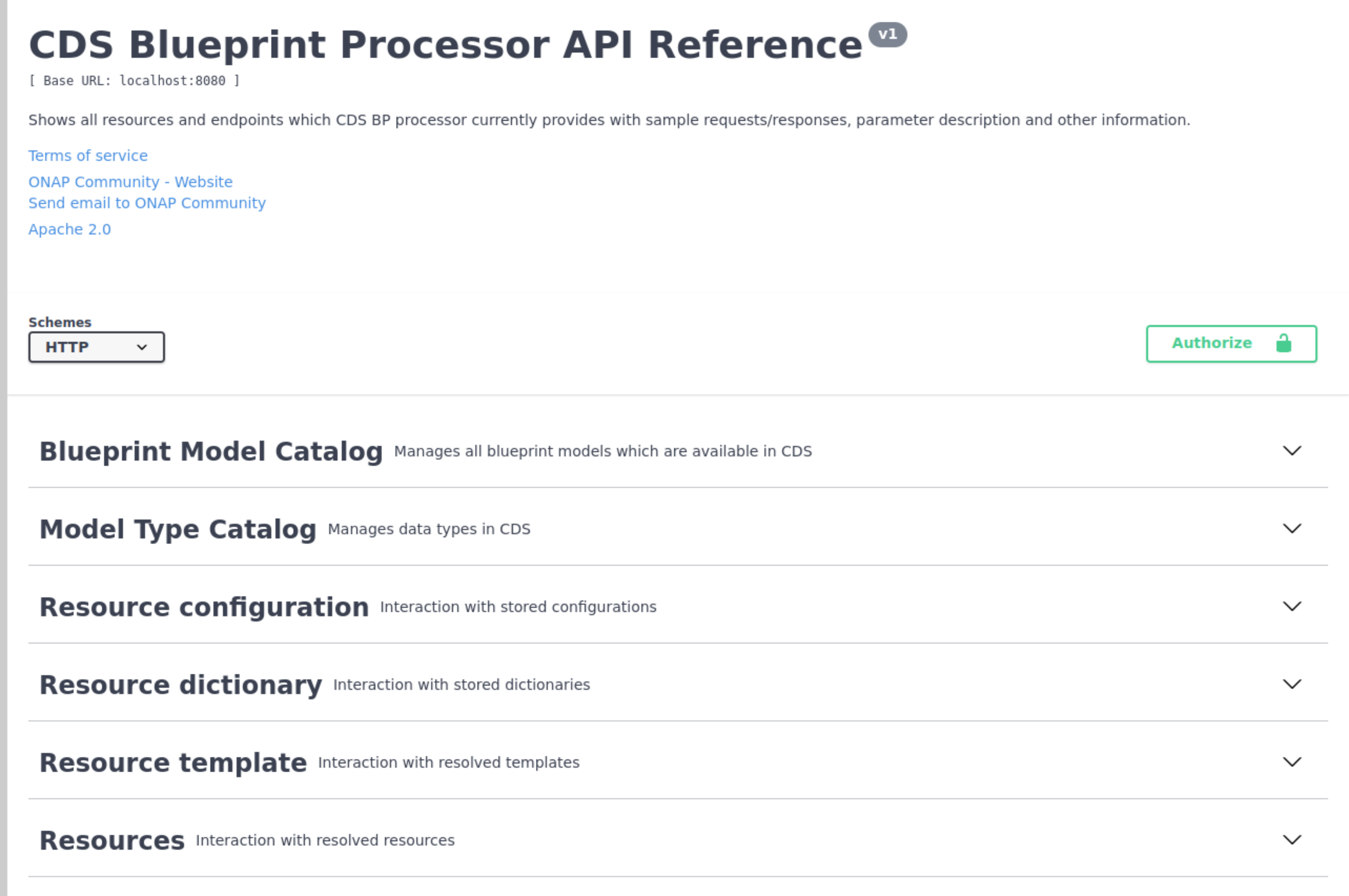
```
{
  "actionIdentifiers": {
    "mode": "sync",
    "blueprintName": "vLB_CDS",
    "blueprintVersion": "1.0.0",
    "actionName": "resource-assignment"
  },
  "payload": {
    "resource-assignment-request": {
      "template-prefix": [
        "vnf"
      ],
      "resource-assignment-properties": {
        "image_name": "ubuntu-16-04-cloud-amd64",
        "vpg_0_int_pktgen_private_port_0_mac": "fa:16:3e:00:00:20",
        "repo_url_artifacts": "https://nexus.onap.org/content/groups/staging",
        "flavor_name": "m1.medium",
        "dcae_collector_ip": "10.12.5.214",
        "onap_private_subnet_id": "oam_network_N0qx",
        "key_name": "olc-key"
      }
    }
  },
  "commonHeader": {
    "subRequestId": "3f259ee6-cd7e-4a83-8b2b-5e6da3a05ce1",
    "requestId": "073e64ed-734e-4937-abb7-0b4c634b52e1",
    "originatorId": "SDNC_DG"
  }
```

1. The actionName, under the actionIdentifiers refers to the name of a Workflow (see Workflow).
2. The content of the payload is what is fully dynamic / model driven.
3. The first top level element will always be either $actionName-request for a request or $actionName-response for a response.
4. Then the content within this element is fully based on the workflow inputs and outputs

Note: During the Enrichment CDS will aggregate all the resources defined to be resolved as input

CDS - Other API

APIs for Blueprint, model, resource, template, data dictionary etc. management



CDS Blueprint Processor API Reference v1

[Base URL: localhost:8080]

Shows all resources and endpoints which CDS BP processor currently provides with sample requests/responses, parameter description and other information.

Terms of service

ONAP Community - Website
Send email to ONAP Community

Apache 2.0

Blueprint Model Catalog Manages all blueprint models which are available in CDS

Model Type Catalog Manages data types in CDS

Resource configuration Interaction with stored configurations

Resource dictionary Interaction with stored dictionaries

Resource template Interaction with resolved templates

Resources Interaction with resolved resources

Modelling Concepts

Controller Blueprints Archive(CBA) Format

Controller Blueprints definitions file, Resource Definition, Others
Formats : .json

Definition

Blueprint environment properties or application properties file.
Formats: .json

Environments

Flow Definitions files, such as directed graph, dataflow dsl, etc.
Formats: .json, .xml

Plans

Executions scripts used during flows.
Formats: .py, .js, .kotlin

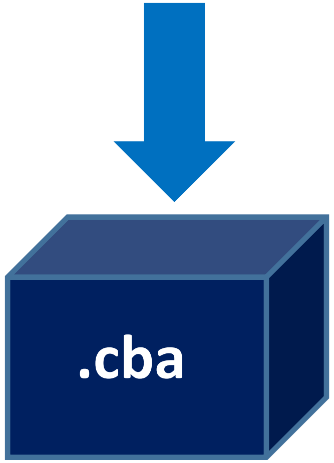
Scripts

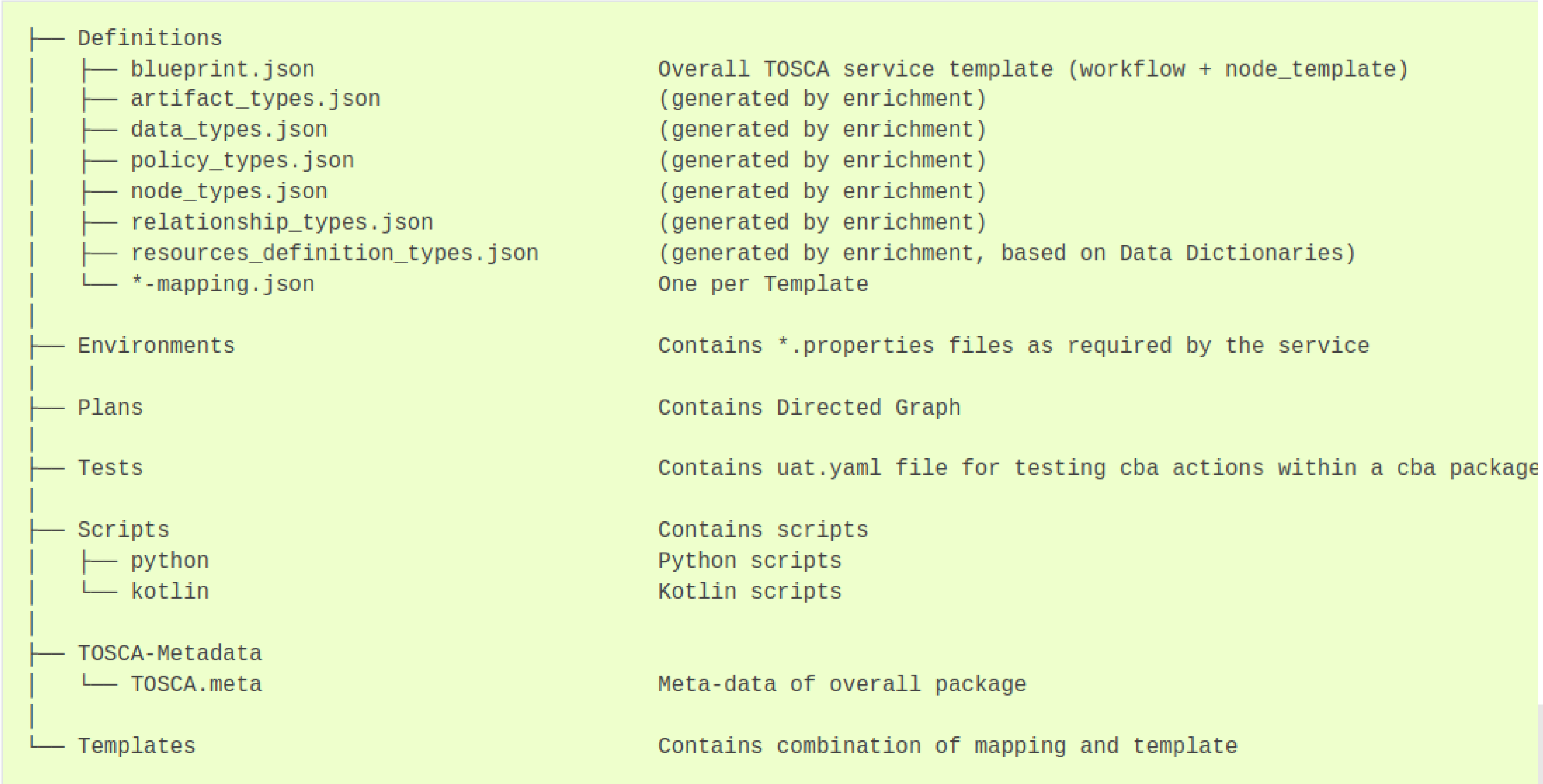
Templates used duting processing.
Format: .vtl

Templates

Meta-data of overall package
Format: .meta

TOSCA-Metadata



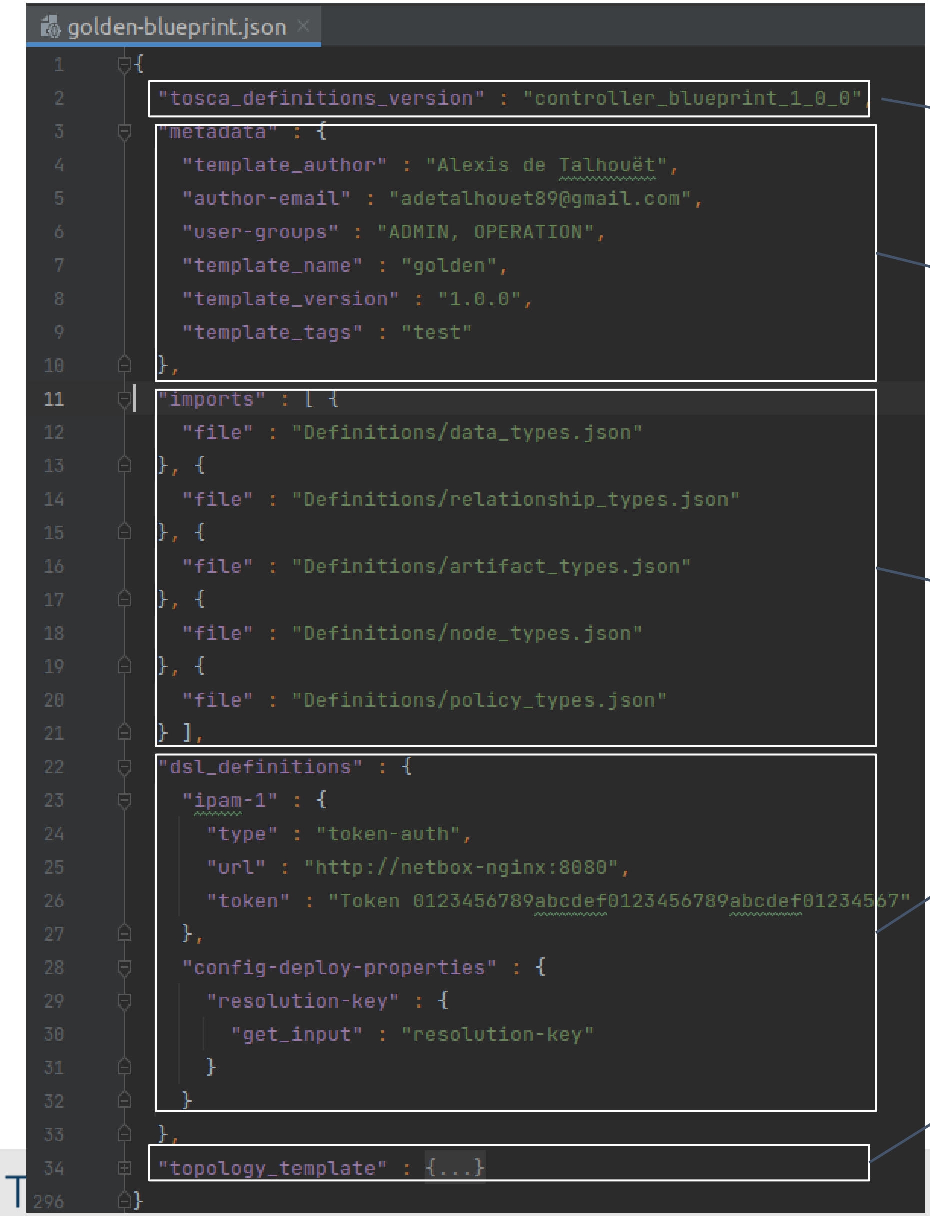


```
├── Definitions
│   ├── blueprint.json                      Overall TOSCA service template (workflow + node_template)
│   ├── artifact_types.json                 (generated by enrichment)
│   ├── data_types.json                     (generated by enrichment)
│   ├── policy_types.json                   (generated by enrichment)
│   ├── node_types.json                     (generated by enrichment)
│   ├── relationship_types.json             (generated by enrichment)
│   ├── resources_definition_types.json     (generated by enrichment, based on Data Dictionaries)
│   └── *-mapping.json                      One per Template
│
├── Environments                            Contains *.properties files as required by the service
│
├── Plans                                   Contains Directed Graph
│
├── Tests                                   Contains uat.yaml file for testing cba actions within a cba package
│
├── Scripts                                 Contains scripts
│   ├── python                              Python scripts
│   └── kotlin                              Kotlin scripts
│
├── TOSCA-Metadata
│   └── TOSCA.meta                          Meta-data of overall package
│
└── Templates                               Contains combination of mapping and template
```

Note: CDS BP modelling is mainly based on TOSCA standard, using JSON as reprensentation format.

Example BP: Golden Blueprint

https://github.com/onap/ccsdk-cds/tree/master/components/model-catalog/blueprint-model/test-blueprint/golden



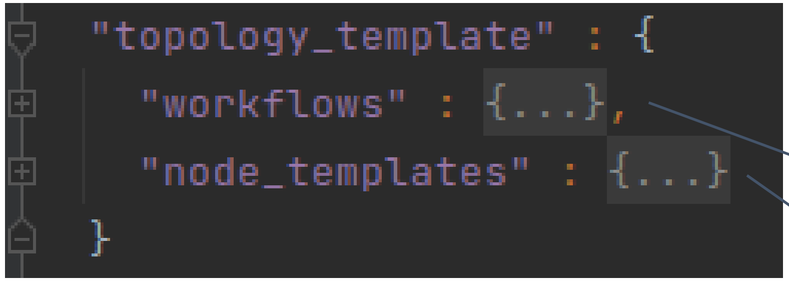
golden-blueprint.json

```
{
  "tosca_definitions_version" : "controller_blueprint_1_0_0",
  "metadata" : {
    "template_author" : "Alexis de Talhouët",
    "author-email" : "adetalhouet89@gmail.com",
    "user-groups" : "ADMIN, OPERATION",
    "template_name" : "golden",
    "template_version" : "1.0.0",
    "template_tags" : "test"
  },
  "imports" : [ {
    "file" : "Definitions/data_types.json"
  }, {
    "file" : "Definitions/relationship_types.json"
  }, {
    "file" : "Definitions/artifact_types.json"
  }, {
    "file" : "Definitions/node_types.json"
  }, {
    "file" : "Definitions/policy_types.json"
  } ],
  "dsl_definitions" : {
    "ipam-1" : {
      "type" : "token-auth",
      "url" : "http://netbox-nginx:8080",
      "token" : "Token 0123456789abcdef0123456789abcdef01234567"
    },
    "config-deploy-properties" : {
      "resolution-key" : {
        "get_input" : "resolution-key"
      }
    }
  },
  "topology_template" : {...}
}
```

Type	Description
string	Defines the version of the Controller Blueprints(CB) Simple Profile specification the template (grammar) complies with.
map of string	Defines a section used to declare additional metadata information. Domain-specific TOSCA profile specifications may define keynames that are required for their implementations.
list of Import Definitions	Declares import statements external CB Definitions documents, may be file location or URIs relative to the service template file within the same CBA file.
dsl definition	Interaction with external systems is made dynamic and plug-able removing development cycle to support new endpoint. In order to share the external system information, TOSCA provides a way to create macros using dsl_definitions
Topology Template definition	Defines the topology template of an application or service, consisting of node templates that represent the application's or service's components, as well as relationship templates representing relations between the components.

Example BP: Golden Blueprint

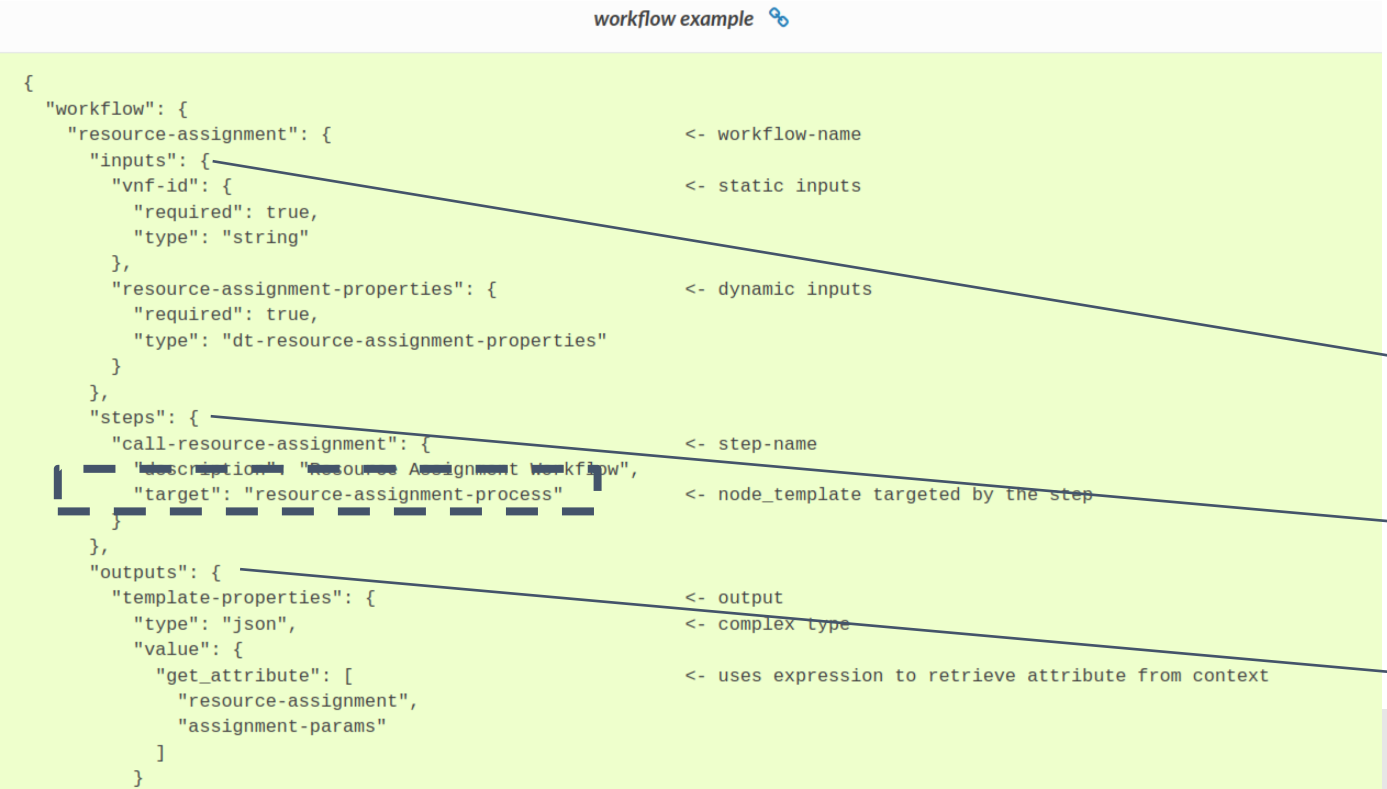
https://github.com/onap/ccsdk-cds/tree/master/components/model-catalog/blueprint-model/test-blueprint/golden



```
"topology_template" : {
  "workflows" : {...},
  "node_templates" : {...}
}
```

Type	Description
list of parameter definitions	An optional list of input parameters (i.e., as parameter definitions) for the Topology Template.
list of imperative workflow definitions	An optional map of imperative workflow definition for the Topology Template.
list of node templates	An optional list of node template definitions for the Topology Template.

Note: Workflow executes node either of type component or DG. Refer Workflow section for details



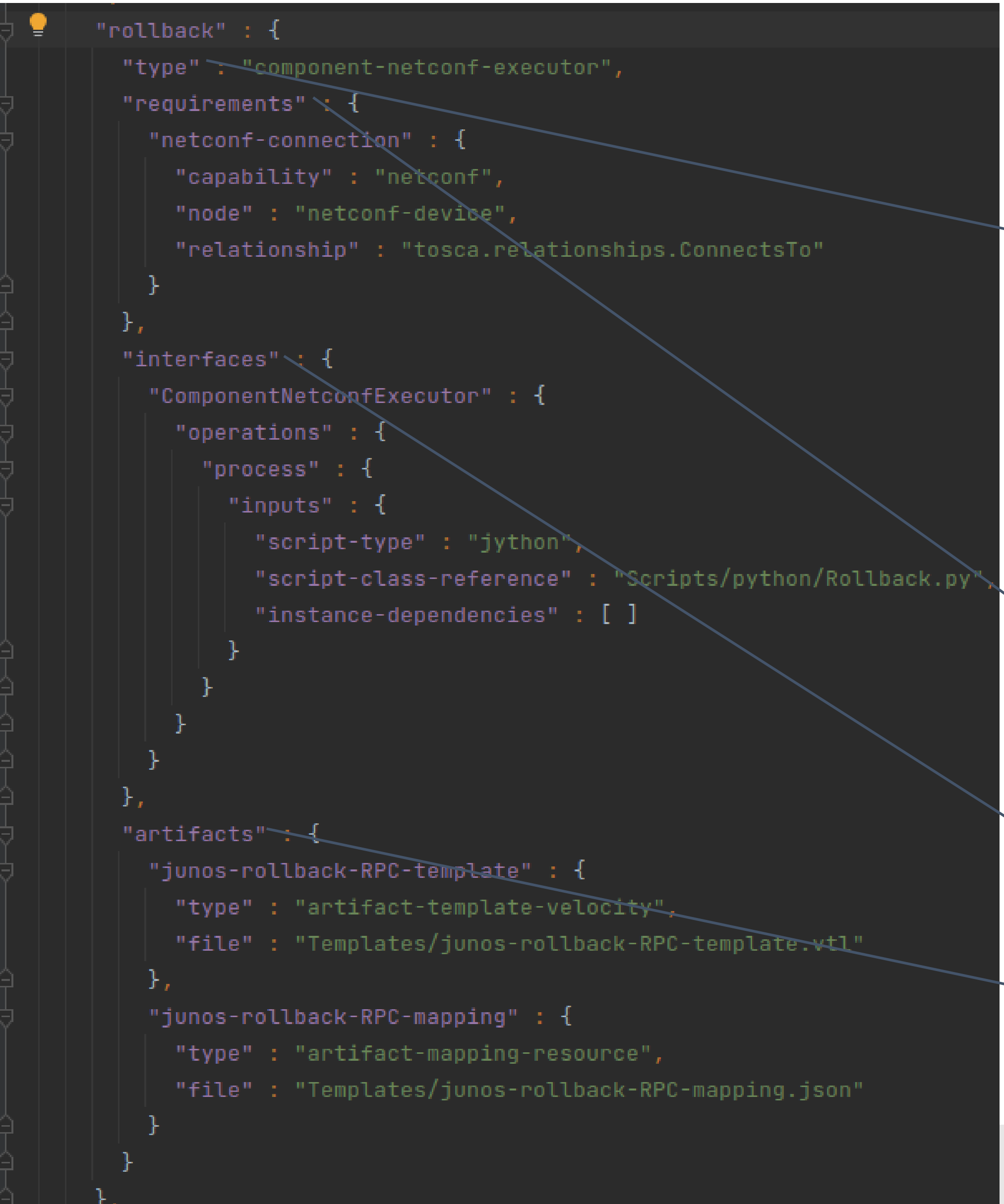
workflow example

```
{
  "workflow": {
    "resource-assignment": {                                <- workflow-name
      "inputs": {
        "vnf-id": {                                         <- static inputs
          "required": true,
          "type": "string"
        },
        "resource-assignment-properties": {                 <- dynamic inputs
          "required": true,
          "type": "dt-resource-assignment-properties"
        }
      },
      "steps": {
        "call-resource-assignment": {                       <- step-name
          "description": "Resource Assignment Workflow",
          "target": "resource-assignment-process"           <- node_template targeted by the step
        }
      },
      "outputs": {
        "template-properties": {                            <- output
          "type": "json",                                   <- complex type
          "value": {
            "get_attribute": [                              <- uses expression to retrieve attribute from context
              "resource-assignment",
              "assignment-params"
            ]
          }
```

Type	Description
list of property definitions	The optional list of input parameter definitions.
list of step definitions	An optional list of valid Node Templates or Groups the Policy can be applied to.
list of property definitions	The optional list of input parameter definitions along with values or expressions.

Example BP: Golden Blueprint

https://github.com/onap/ccsdk-cds/tree/master/components/model-catalog/blueprint-model/test-blueprint/golden

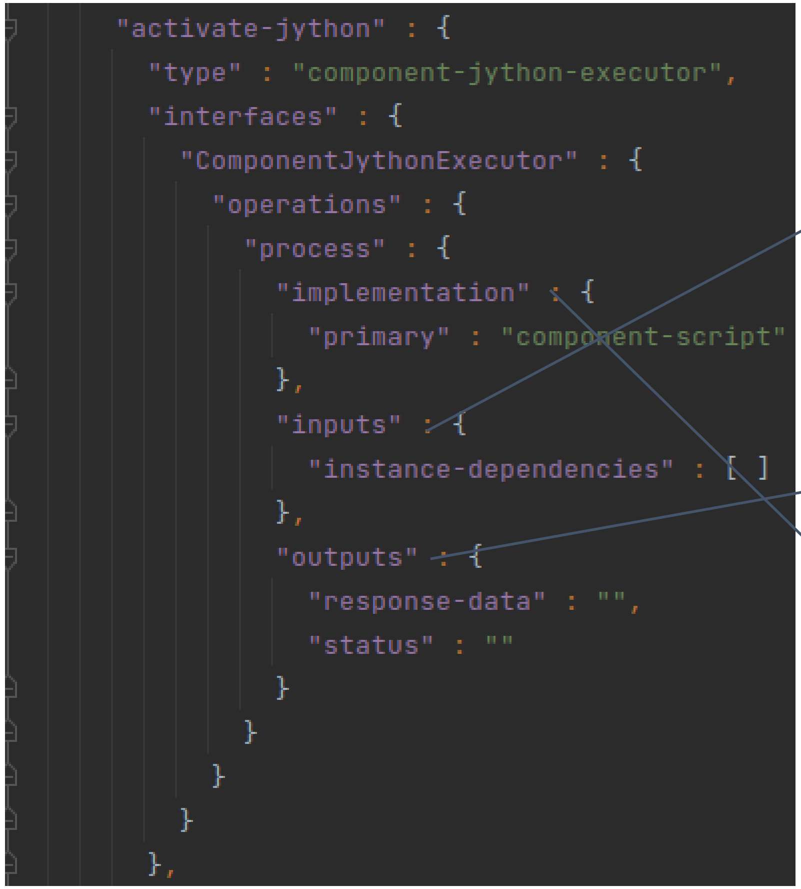


```
"rollback" : {
  "type" : "component-netconf-executor",
  "requirements" : {
    "netconf-connection" : {
      "capability" : "netconf",
      "node" : "netconf-device",
      "relationship" : "tosca.relationships.ConnectsTo"
    }
  },
  "interfaces" : {
    "ComponentNetconfExecutor" : {
      "operations" : {
        "process" : {
          "inputs" : {
            "script-type" : "jython",
            "script-class-reference" : "Scripts/python/Rollback.py",
            "instance-dependencies" : [ ]
          }
        }
      }
    }
  },
  "artifacts" : {
    "junos-rollback-RPC-template" : {
      "type" : "artifact-template-velocity",
      "file" : "Templates/junos-rollback-RPC-template.vtl"
    },
    "junos-rollback-RPC-mapping" : {
      "type" : "artifact-mapping-resource",
      "file" : "Templates/junos-rollback-RPC-mapping.json"
    }
  }
}
```

Node Template Keys	Required	Type	Description
type	yes	string	The required name of the Node Type the Node Template is based upon.
description	no	description	An optional description for the Node Template.
properties	no	list of property assignments	An optional list of property value assignments for the Node Template.
attributes	no	list of attribute assignments	An optional list of attribute value assignments for the Node Template.
requirements	no	list of requirement assignments	An optional list of requirement assignments for the Node Template.
capabilities	no	list of capability assignments	An optional list of capability assignments for the Node Template.
interfaces	no	list of interface definitions	An optional list of named interface definitions for the Node Template.
artifacts	no	list of artifact definitions	An optional list of named artifact definitions for the Node Template.

Example BP: Golden Blueprint

Activation Blueprint



```
"activate-jython" : {
  "type" : "component-jython-executor",
  "interfaces" : {
    "ComponentJythonExecutor" : {
      "operations" : {
        "process" : {
          "implementation" : {
            "primary" : "component-script"
          },
          "inputs" : {
            "instance-dependencies" : [ ]
          },
          "outputs" : {
            "response-data" : "",
            "status" : ""
          }
        }
      }
    }
  },
```

Operation Definition Keys	Required	Type	Description
inputs	no	list of property definitions	The optional list of input property definitions available to all defined operations for interface definitions that are within Node or Relationship Type definitions.
	no	list of property assignments	The optional list of input property assignments (i.e., parameters assignments) for interface definitions that are within Node or Relationship Template definitions.
outputs	no	list of property definitions	The optional list of output property definitions available to all defined operations for interface definitions that are within Node or Relationship Type definitions
	no	list of property assignments	The optional list of output property assignments (i.e., parameters assignments) for interface definitions that are within Node or Relationship Template definitions.
implementation	no	Operation implementation definition	The optional definition for operation implementations.
policies	no	String[]	An optional list of Policy definition name for the Operation Definition.

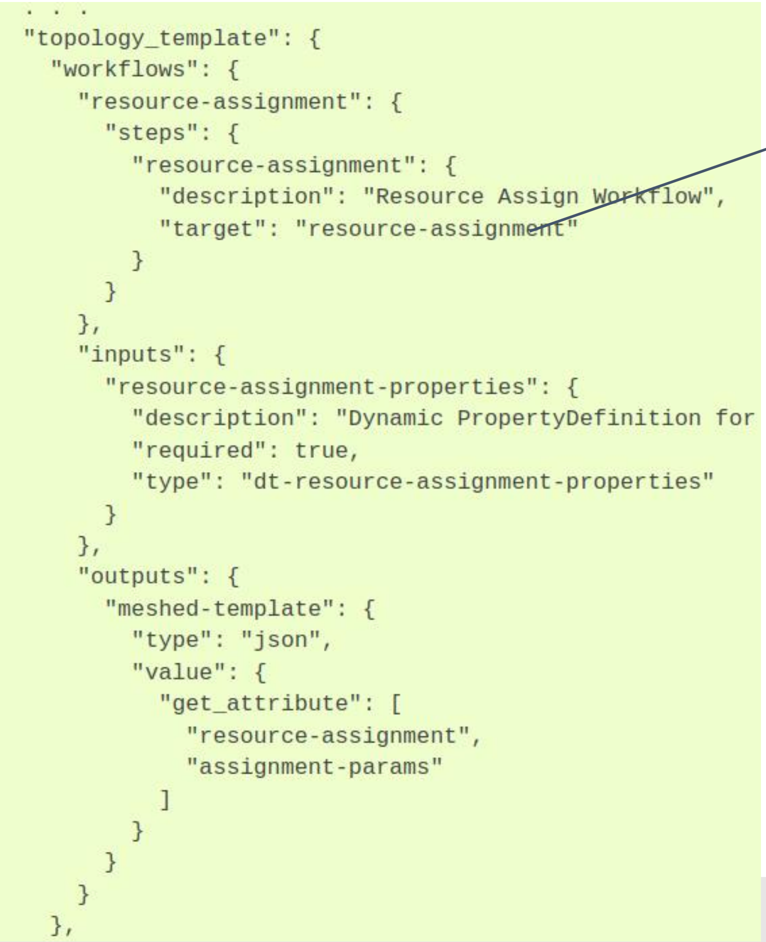
Workflow

/plans

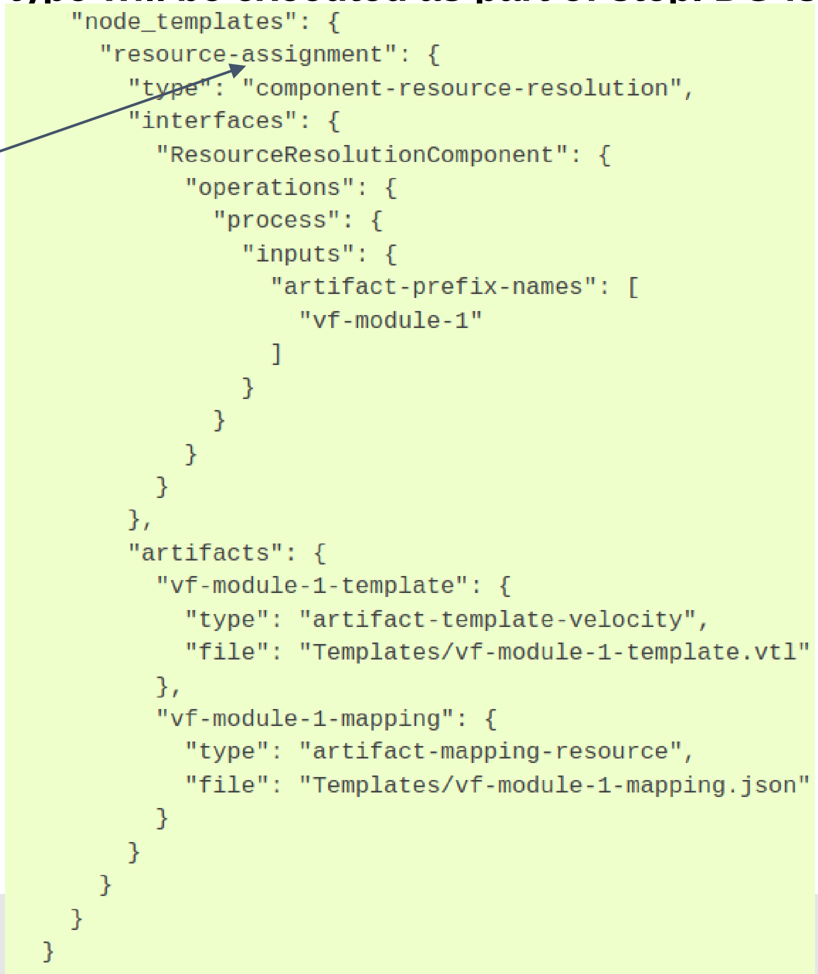
https://docs.onap.org/projects/onap-ccsdk-cds/en/latest/modelingconcepts/workflow.html

A workflow defines an overall action to be taken on the service, hence is an entry-point for the run-time execution of the CBA Package. A workflow also defines **inputs** and **outputs** that will defined the **payload contract** of the **request** and **response.** A workflow can be **composed** of one or multiple **sub-actions** to execute. A CBA package can have as **many workflows** as needed.

Single action: Directly a component node type will be executed as part of step. DG is not required for this



```
. . .
"topology_template": {
  "workflows": {
    "resource-assignment": {
      "steps": {
        "resource-assignment": {
          "description": "Resource Assign Workflow",
          "target": "resource-assignment"
        }
      }
    },
    "inputs": {
      "resource-assignment-properties": {
        "description": "Dynamic PropertyDefinition for
        "required": true,
        "type": "dt-resource-assignment-properties"
      }
    },
    "outputs": {
      "meshed-template": {
        "type": "json",
        "value": {
          "get_attribute": [
            "resource-assignment",
            "assignment-params"
          ]
        }
      }
    }
  },
```

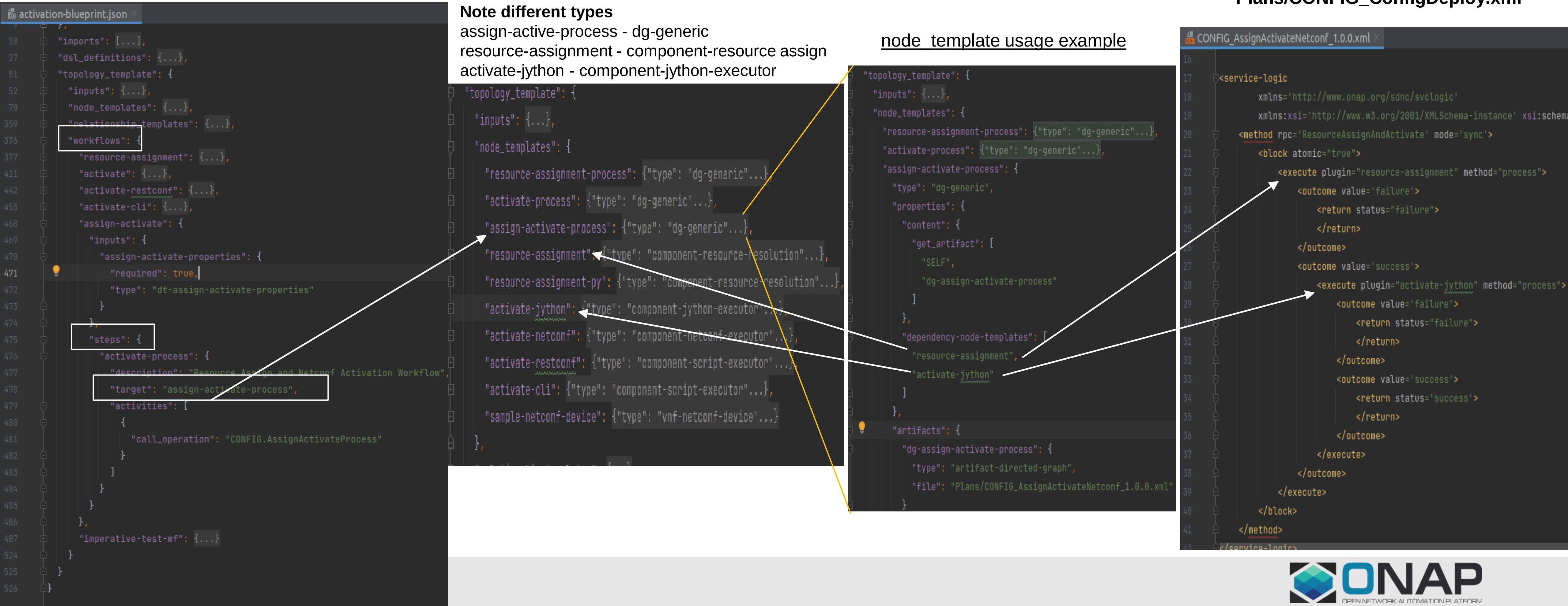


```
  "node_templates": {
    "resource-assignment": {
      "type": "component-resource-resolution",
      "interfaces": {
        "ResourceResolutionComponent": {
          "operations": {
            "process": {
              "inputs": {
                "artifact-prefix-names": [
                  "vf-module-1"
                ]
              }
            }
          }
        }
      },
      "artifacts": {
        "vf-module-1-template": {
          "type": "artifact-template-velocity",
          "file": "Templates/vf-module-1-template.vtl"
        },
        "vf-module-1-mapping": {
          "type": "artifact-mapping-resource",
          "file": "Templates/vf-module-1-mapping.json"
        }
      }
    }
  }
}
```

Workflow cont..

Multiple sub-actions: When multiple functions has to be executed for a given workflow. As part of step a DG node is executed. DG than execute series of nodes. A DG used as workflow for CDS is composed of multiple execute nodes; each individual execute node refers to an modelled Component.

Workflow - Detail in later slides



activation-blueprint.json

```
"imports": [...],
"dsl_definitions": {...},
"topology_template": {
  "inputs": {...},
  "node_templates": {...},
  "relationship_templates": {...},
  "workflows": {
    "resource-assignment": {...},
    "activate": {...},
    "activate-restconf": {...},
    "activate-cli": {...},
    "assign-activate": {
      "inputs": {
        "assign-activate-properties": {
          "required": true,
          "type": "dt-assign-activate-properties"
        }
      },
      "steps": {
        "activate-process": {
          "description": "Resource Assign and Netconf Activation Workflow",
          "target": "assign-activate-process",
          "activities": [
            {
              "call_operation": "CONFIG.AssignActivateProcess"
            }
          ]
        }
      }
    },
    "imperative-test-wf": {...}
  }
}
```

Note different types
assign-active-process - dg-generic
resource-assignment - component-resource assign
activate-jython - component-jython-executor

```
"topology_template": {
  "inputs": {...},
  "node_templates": {
    "resource-assignment-process": {"type": "dg-generic"...},
    "activate-process": {"type": "dg-generic"...},
    "assign-activate-process": {"type": "dg-generic"...},
    "resource-assignment": {"type": "component-resource-resolution"...},
    "resource-assignment-py": {"type": "component-resource-resolution"...},
    "activate-jython": {"type": "component-jython-executor"...},
    "activate-netconf": {"type": "component-netconf-executor"...},
    "activate-restconf": {"type": "component-script-executor"...},
    "activate-cli": {"type": "component-script-executor"...},
    "sample-netconf-device": {"type": "vnf-netconf-device"...}
  },
```

node_template usage example

```
"topology_template": {
  "inputs": {...},
  "node_templates": {
    "resource-assignment-process": {"type": "dg-generic"...},
    "activate-process": {"type": "dg-generic"...},
    "assign-activate-process": {
      "type": "dg-generic",
      "properties": {
        "content": {
          "get_artifact": [
            "SELF",
            "dg-assign-activate-process"
          ]
        },
        "dependency-node-templates": [
          "resource-assignment",
          "activate-jython"
        ]
      },
      "artifacts": {
        "dg-assign-activate-process": {
          "type": "artifact-directed-graph",
          "file": "Plans/CONFIG_AssignActivateNetconf_1.0.0.xml"
        }
```

Plans/CONFIG_ConfigDeploy.xml

CONFIG_AssignActivateNetconf_1.0.0.xml

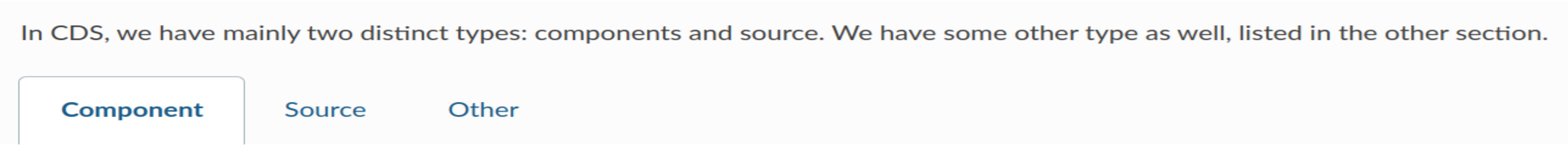
```

<service-logic
        xmlns='http://www.onap.org/sdnc/svclogic'
        xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' xsi:schema
    <method rpc='ResourceAssignAndActivate' mode='sync'>
        <block atomic="true">
            <execute plugin="resource-assignment" method="process">
                <outcome value='failure'>
                    <return status="failure">
                    </return>
                </outcome>
                <outcome value='success'>
                    <execute plugin="activate-jython" method="process">
                        <outcome value='failure'>
                            <return status="failure">
                            </return>
                        </outcome>
                        <outcome value='success'>
                            <return status='success'>
                            </return>
                        </outcome>
                    </execute>
                </outcome>
            </execute>
        </block>
    </method>
</service-logic>
```

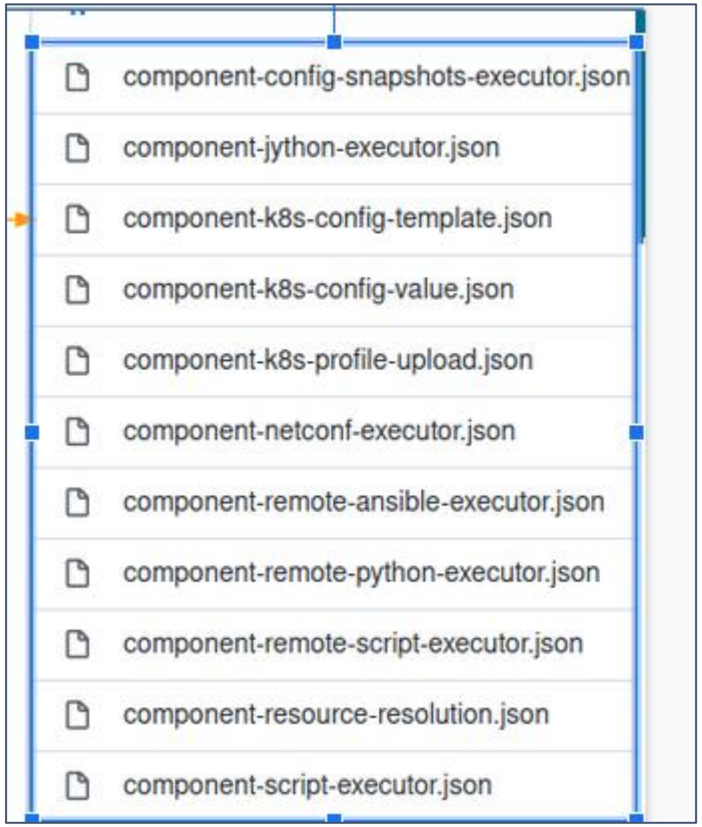
Node Type

/Definitions/node_types.json

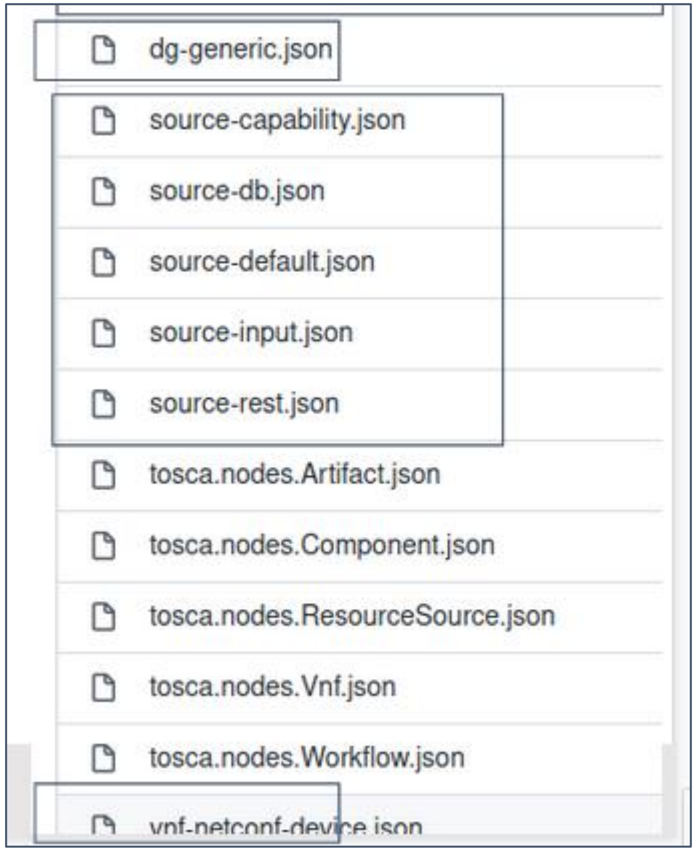
https://docs.onap.org/projects/onap-ccsdk-cds/en/latest/modelingconcepts/node-type.html#
https://github.com/onap/ccsdk-cds/tree/master/components/model-catalog/definition-type/starter-type/node_type



In CDS, we have mainly two distinct types: components and source. We have some other type as well, listed in the other section.



- component-config-snapshots-executor.json
- component-jython-executor.json
- component-k8s-config-template.json
- component-k8s-config-value.json
- component-k8s-profile-upload.json
- component-netconf-executor.json
- component-remote-ansible-executor.json
- component-remote-python-executor.json
- component-remote-script-executor.json
- component-resource-resolution.json
- component-script-executor.json

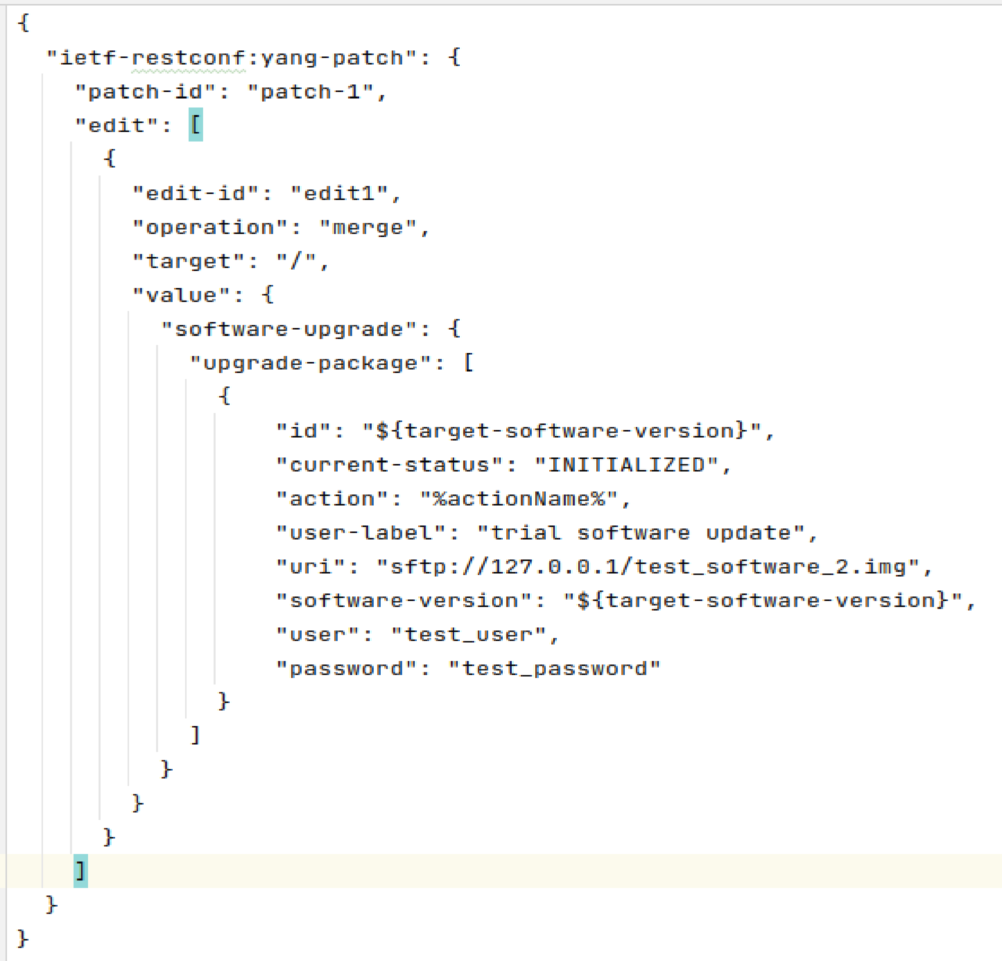


- dg-generic.json
- source-capability.json
- source-db.json
- source-default.json
- source-input.json
- source-rest.json
- tosca.nodes.Artifact.json
- tosca.nodes.Component.json
- tosca.nodes.ResourceSource.json
- tosca.nodes.Vnf.json
- tosca.nodes.Workflow.json
- vnf-netconf-device.json

https://docs.onap.org/projects/onap-ccsdk-cds/en/latest/modelingconcepts/node-type.html#

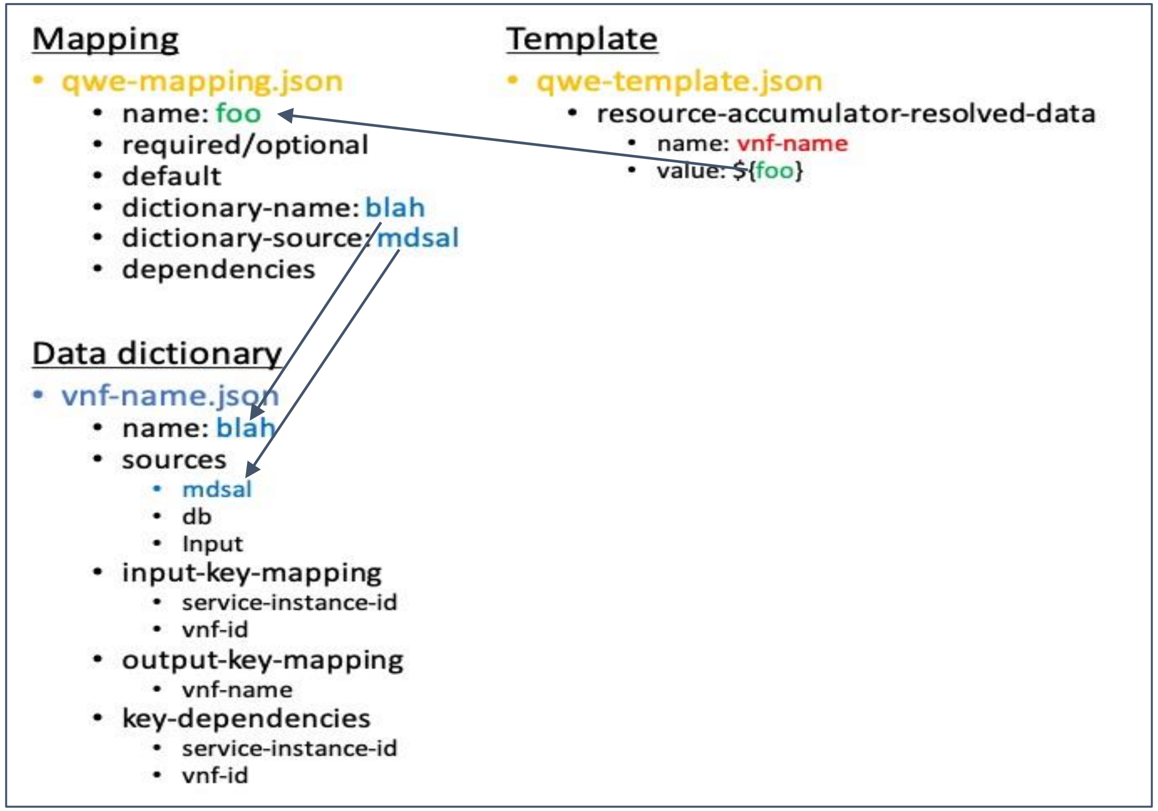
Node Type - Component - Resource Resolution

Payload while invoking 3rd party API, which has many parameters which needs to be resolved - Resource resolution component is meant for this



```
{
  "ietf-restconf:yang-patch": {
    "patch-id": "patch-1",
    "edit": [
      {
        "edit-id": "edit1",
        "operation": "merge",
        "target": "/",
        "value": {
          "software-upgrade": {
            "upgrade-package": [
              {
                  "id": "${target-software-version}",
                  "current-status": "INITIALIZED",
                  "action": "%actionName%",
                  "user-label": "trial software update",
                  "uri": "sftp://127.0.0.1/test_software_2.img",
                  "software-version": "${target-software-version}",
                  "user": "test_user",
                  "password": "test_password"
              }
            ]
          }
        }
      }
    ]
  }
}
```

Example showing link between all Artifacts used for resource resolution



Mapping

- qwe-mapping.json
  - name: foo
  - required/optional
  - default
  - dictionary-name: blah
  - dictionary-source: mdsal
  - dependencies

Template

- qwe-template.json
  - resource-accumulator-resolved-data
    - name: vnf-name
    - value: ${foo}

Data dictionary

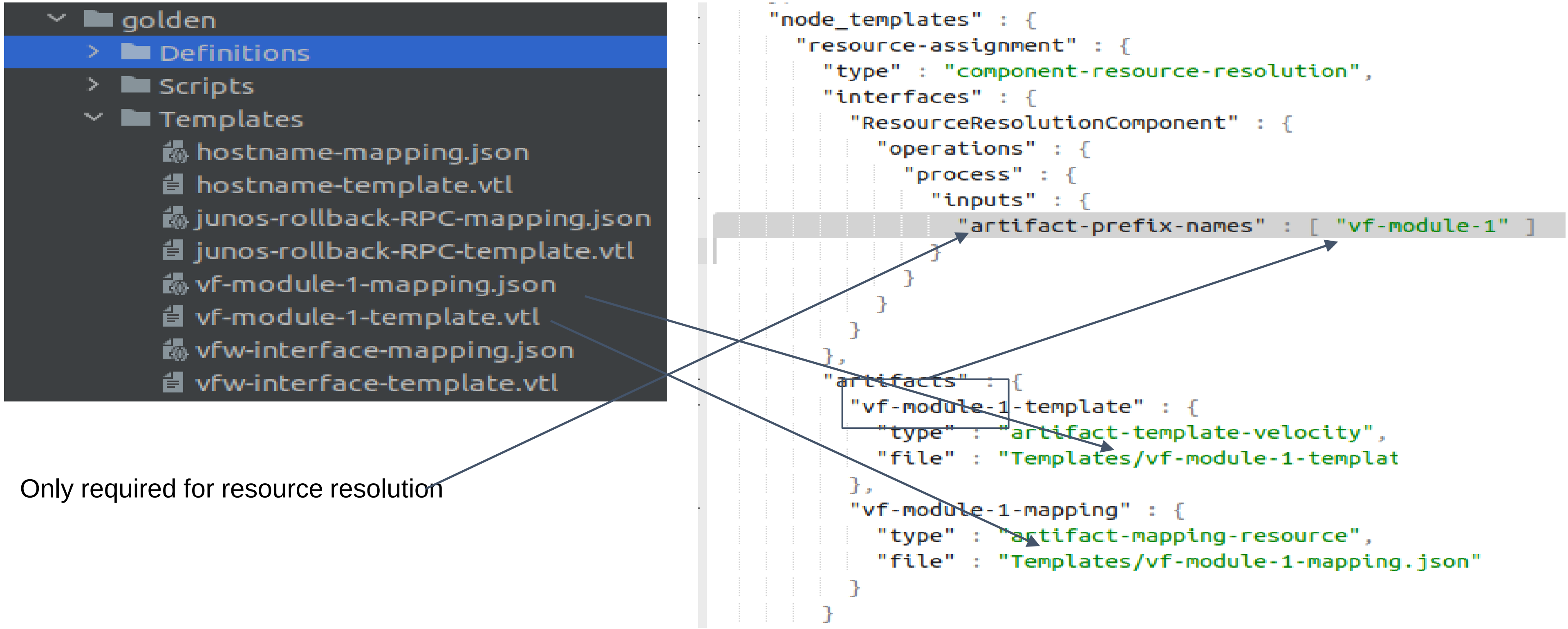
- vnf-name.json
  - name: blah
  - sources
    - mdsal
    - db
    - Input
  - input-key-mapping
    - service-instance-id
    - vnf-id
  - output-key-mapping
    - vnf-name
  - key-dependencies
    - service-instance-id
    - vnf-id

https://docs.onap.org/projects/onap-ccsdk-cds/en/latest/modelingconcepts/node-type.html#

Node Type - Component - Resource Resolution

Resource Resolution Component Usage Example

Node template referring to Template & Mapping for resource resolution



```
golden
  Definitions
  Scripts
  Templates
    hostname-mapping.json
    hostname-template.vtl
    junos-rollback-RPC-mapping.json
    junos-rollback-RPC-template.vtl
    vf-module-1-mapping.json
    vf-module-1-template.vtl
    vfw-interface-mapping.json
    vfw-interface-template.vtl
```

```
"node_templates" : {
  "resource-assignment" : {
    "type" : "component-resource-resolution",
    "interfaces" : {
      "ResourceResolutionComponent" : {
        "operations" : {
          "process" : {
            "inputs" : {
              "artifact-prefix-names" : [ "vf-module-1" ]
            }
          }
        }
      }
    },
    "artifacts" : {
      "vf-module-1-template" : {
        "type" : "artifact-template-velocity",
        "file" : "Templates/vf-module-1-templat
      },
      "vf-module-1-mapping" : {
        "type" : "artifact-mapping-resource",
        "file" : "Templates/vf-module-1-mapping.json"
      }
    }
```

Only required for resource resolution

Resource Resolution: Data Dictionary

/Definitions/resources_definition_types.json

https://docs.onap.org/projects/onap-ccsdk-cds/en/latest/modelingconcepts/data-dictionary.html
https://wiki.onap.org/display/DW/User+Guide#DD-10035809
Example: https://gerrit.onap.org/r/gitweb?p=ccsdk/cds.git;a=blob;f=components/model-catalog/blueprint-model/test-blueprint/golden/Definitions/resources_definition_types.json;h=475d8641a87b781ffa65bca7e45191cc83455908;hb=refs/heads/istanbul

What: A data dictionary models how a specific resource can be resolved. It is used for resource resolution by resource resolution component

Why: The main goal of data dictionary is to define **re-usable entity** that could be shared.

Note:

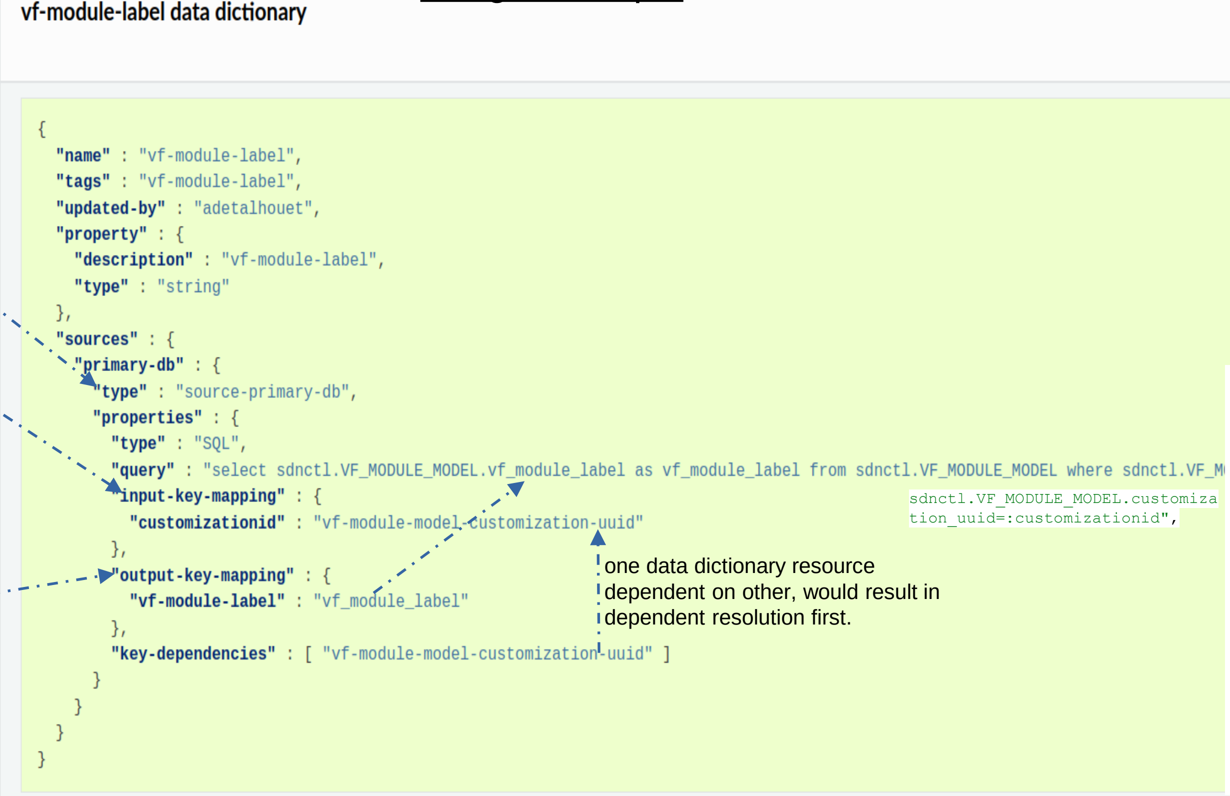
1. After Enrichment required Data Dictionary added to definitions/ resources_definition_types.json
2. Creation of data dictionaries is a standalone activity, separated from the blueprint design. Some starter pack of dictionary comes with CDS. Also CDS provide API to add/delete new/existing. **Also can be added via CDS UI**

Usage Example

Data Dictionary to resolve **vf-module-label** by obtaining value from DB

Input-key-mapping to **rename** input variable
Also input variable obtained by resource resolution defined in dictionary for Vf-module-model-customization-uuid

Output-key-mapping to **rename** output variable



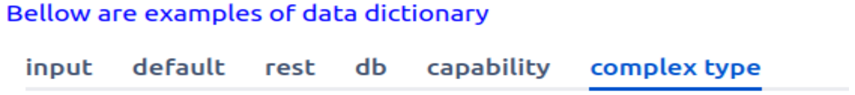
vf-module-label data dictionary

```
{
  "name" : "vf-module-label",
  "tags" : "vf-module-label",
  "updated-by" : "adetalhouet",
  "property" : {
    "description" : "vf-module-label",
    "type" : "string"
  },
  "sources" : {
    "primary-db" : {
      "type" : "source-primary-db",
      "properties" : {
        "type" : "SQL",
        "query" : "select sdnctl.VF_MODULE_MODEL.vf_module_label as vf_module_label from sdnctl.VF_MODULE_MODEL where sdnctl.VF_M
        "input-key-mapping" : {
          "customizationid" : "vf-module-model-customization-uuid"
        },
        "output-key-mapping" : {
          "vf-module-label" : "vf_module_label"
        },
        "key-dependencies" : [ "vf-module-model-customization-uuid" ]
      }
    }
  }
}
```

```
sdnctl.VF_MODULE_MODEL.customiza
tion_uuid=:customizationid",
```

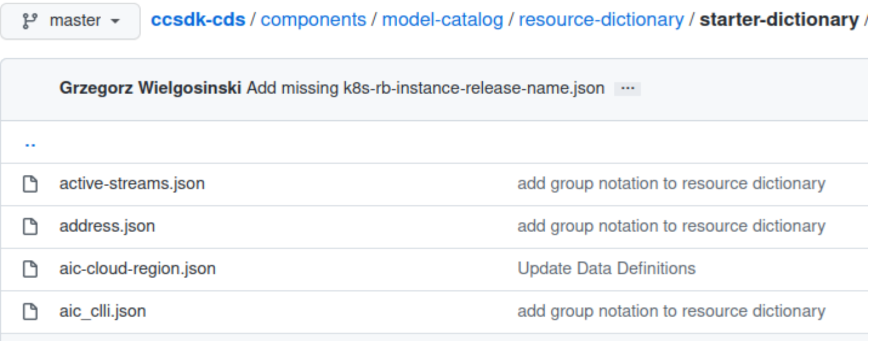
one data dictionary resource dependent on other, would result in dependent resolution first.

Supported Data Dictionary Sources



Bellow are examples of data dictionary

Available/Starter Data Dictionary



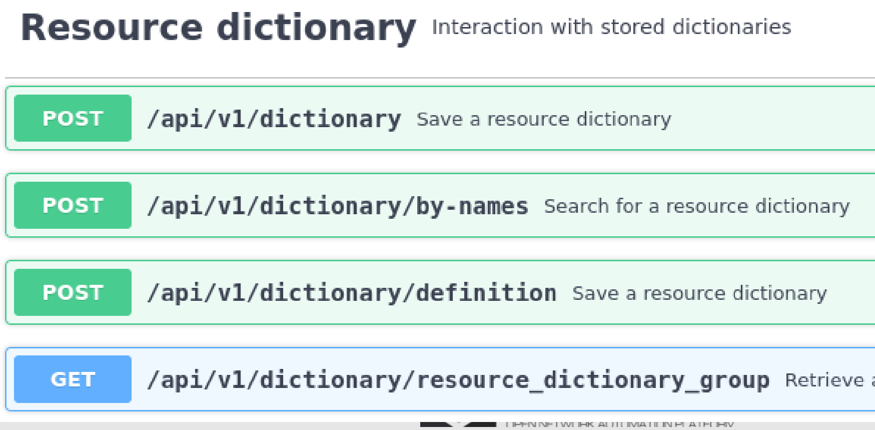
ccsdk-cds / components / model-catalog / resource-dictionary / **starter-dictionary** /

Grzegorz Wielgosinski Add missing k8s-rb-instance-release-name.json ···

..

active-streams.json	add group notation to resource dictionary
address.json	add group notation to resource dictionary
aic-cloud-region.json	Update Data Definitions
aic_clli.json	add group notation to resource dictionary

API to manage Data Dictionary



Resource dictionary Interaction with stored dictionaries

POST	/api/v1/dictionary	Save a resource dictionary
POST	/api/v1/dictionary/by-names	Search for a resource dictionary
POST	/api/v1/dictionary/definition	Save a resource dictionary
GET	/api/v1/dictionary/resource_dictionary_group	Retrieve a

https://docs.onap.org/projects/onap-ccsdk-cds/en/latest/modelingconcepts/node-type.html#

Node Type - Component - Script Executor

component-script-executor:

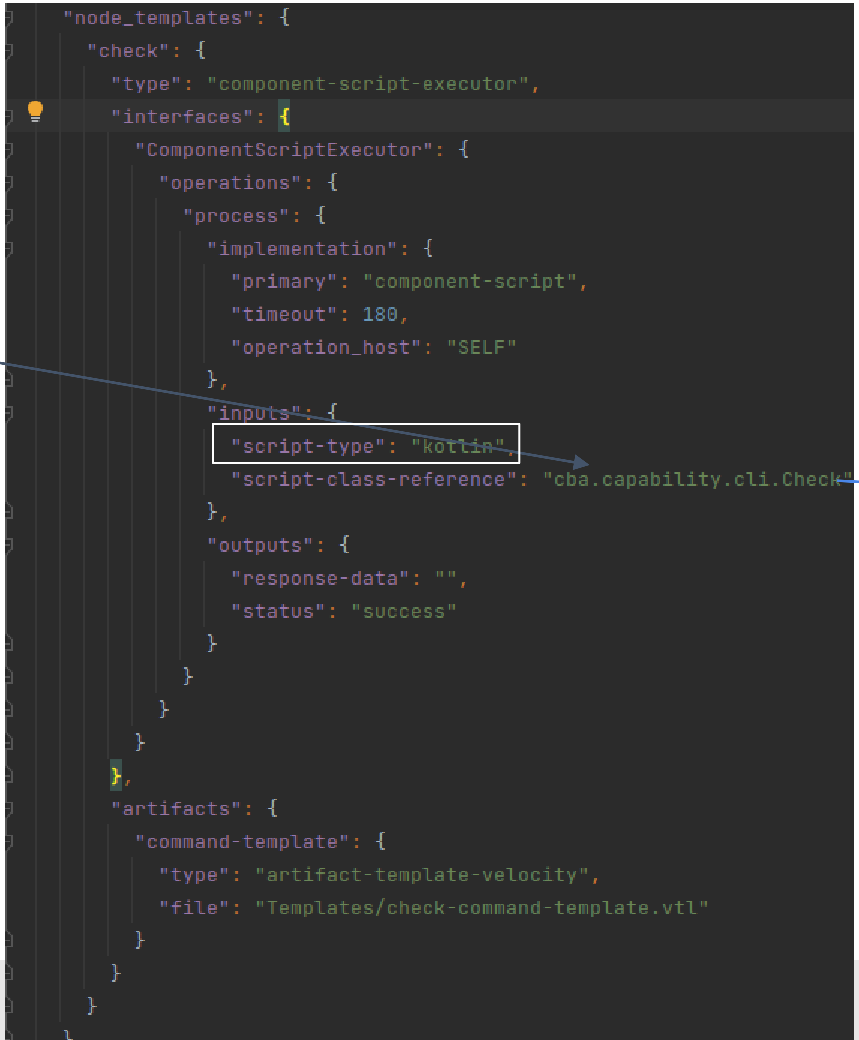
Used to **execute** a script to perform **NETCONF, RESTCONF, SSH commands** etc. from within the runtime container of CDS

Two type of scripts are supported:

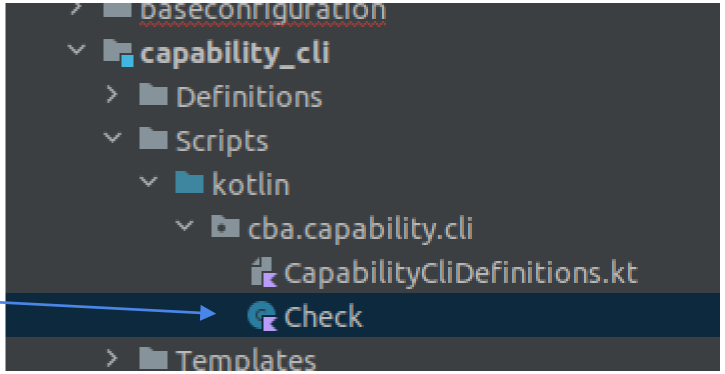
- **Kotlin**: offer a way more integrated scripting framework.
- **Python**: uses Jython for python script execution

The script-class-reference field need to reference

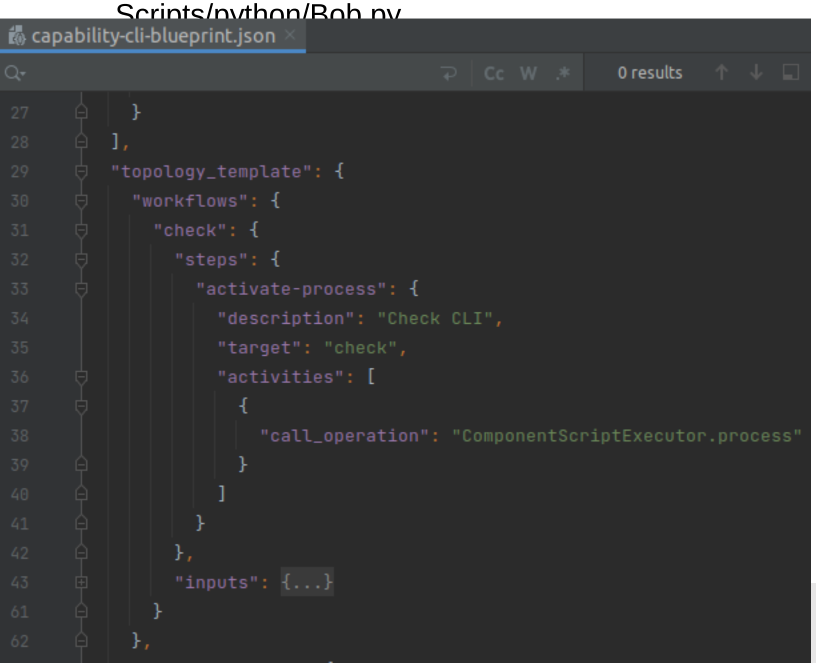
- for kotlin: the package name up to the class. e.g. com.example.Bob
- for python: it has to be the path from the Scripts folder, e.g. Scripts/python/Bob.py



```
"node_templates": {
  "check": {
    "type": "component-script-executor",
    "interfaces": {
      "ComponentScriptExecutor": {
        "operations": {
          "process": {
            "implementation": {
              "primary": "component-script",
              "timeout": 180,
              "operation_host": "SELF"
            },
            "inputs": {
              "script-type": "kotlin",
              "script-class-reference": "cba.capability.cli.Check"
            },
            "outputs": {
              "response-data": "",
              "status": "success"
            }
          }
        }
      }
    },
    "artifacts": {
      "command-template": {
        "type": "artifact-template-velocity",
        "file": "Templates/check-command-template.vtl"
      }
    }
  }
}
```



```
> baseconfiguration
v capability_cli
  > Definitions
  v Scripts
    v kotlin
      v cba.capability.cli
          CapabilityCliDefinitions.kt
          Check
  > Templates
```



```
    }
  ],
  "topology_template": {
    "workflows": {
      "check": {
        "steps": {
          "activate-process": {
            "description": "Check CLI",
            "target": "check",
            "activities": [
              {
                "call_operation": "ComponentScriptExecutor.process"
              }
            ]
          }
        },
        "inputs": {...}
      }
    },
```

Script Executor - Script

```
   open class HelloWorld : AbstractScriptComponentFunction() {
   private val log = LoggerFactory.getLogger(HelloWorld::class.java)!!

   override fun getName(): String {
       return "HelloWorld"
   }

   override suspend fun processNB(executionRequest: ExecutionServiceInput) {
       log.info("executing hello world script ")
       val url = getDynamicProperties("url").asText()
       log.info("url : $url")
       val RestInfo: String = "{\n" +
               " \"type\" : \"basic-auth\",\n" +
               " \"url\" : \"" + url + "\",\n" +
               " \"username\" : \"" + "username" + "\",\n" +
               " \"password\" : \"" + "password" + "\"\n" +
               "}"
       val mapper = ObjectMapper()
       val jsonRestInfo: JsonNode = mapper.readTree(RestInfo)
       val web_client_service = BluePrintDependencyService.restClientService(jsonRestInfo)
       val headers = mutableMapOf<String, String>()
       headers["Content-Type"] = "application/json"
       val mountPayload = storedContentFromResolvedArtifactNB("testkey", "Test")
       log.info("mountPayload : $mountPayload")
       val response = web_client_service.exchangeResource("POST", "/test", mountPayload, headers)
       setAttribute("response-data", "Success".asJsonPrimitive())
   }

   override suspend fun recoverNB(runtimeException: RuntimeException, executionRequest: ExecutionServiceInput) {
       log.info("Executing Recovery")
       addError("${runtimeException.message}")
   }
}
```

Script needs to

1. Extend `AbstractScriptComponentFunction`
2. Implement `processNB & recoverNB`
3. Use CDS provided libraries to get input, do connections, resolve resource, get resolved resources, error handling etc.

Steps: Installation, Blueprint Design to Distribution

Prerequisite: Installation - Local setup

https://wiki.onap.org/display/DW/Running+Blueprints+Processor+Microservice+Locally

1. Checkout the code from gerrit : https://gerrit.onap.org/r/#/admin/projects/ccsdk/cds
2. Build the checked out cds repository by running cmd: mvn clean install -Pq
   Requirement: https://wiki.onap.org/display/DW/Setting+Up+Your+Development+Environment

Start Backend

1. To Build docker images of backend:
   a. From the CDS home directory (where the code was checked out), navigate to the module: “cd ms/blueprintsprocessor/application/”
   b. Build docker image using the Maven profile called Docker:
      “mvn clean install -Pdocker -Ddocker.skip.push=true”
1. To Start backend docker containers using docker compose
   a. Navigate to the docker compose file in the application module:
      “cd ~/cds/ms/blueprintsprocessor/application/src/main/dc/”
   a. Edit docker-compose file(comment command-executor and py-executor-default services) and start containers using “docker-compose up -d” command

Prerequisite: Installation - Local setup

Front end:

UI

1. Navigate to cds-ui folder: “cd ~/cds/cds-ui/designer-client”
2. download all the dependencies using cmd: “npm install”
3. Run cds-ui using cmd: “npm start”
4. If you encounter an error regarding node-sass module not found, then download sass module using cmd: “npm install sass” and run UI again.

Server:

1. Navigate to cds-ui folder: “cd ~/cds/cds-ui/server”.
2. download all the dependencies using cmd: “npm install”
3. Run using cmd: “npm start”

Note: For local installation, if you encounter connection refused between UI-Server and Blueprint Processor, you would need to find and change port from 8080 to 8000 in: /server/src/config/app-config.ts & /server/dist/src/config/app-config.js

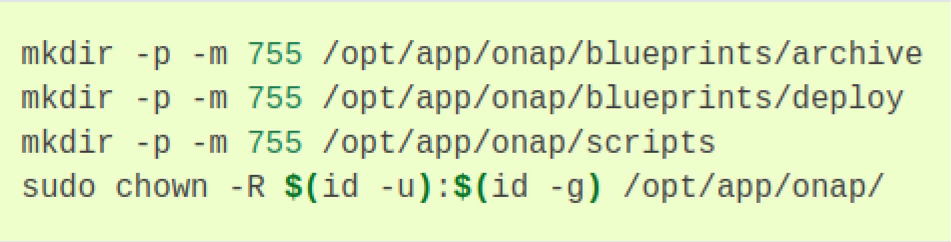
Prerequisite: Installation - Other Options

Start Backend with IDE : https://docs.onap.org/projects/onap-ccsdk-cds/en/latest/userguides/developer-guide/running-bp-processor-in-ide.html

CDS UI via docker-compose
https://docs.onap.org/projects/onap-ccsdk-cds/en/latest/userguides/developer-guide/running-cds-ui-locally.html

Directory to be corrected and should match docker-compose

OOM based Installation
https://docs.onap.org/projects/onap-ccsdk-cds/en/latest/userguides/installation.html



```
mkdir -p -m 755 /opt/app/onap/blueprints/archive
mkdir -p -m 755 /opt/app/onap/blueprints/deploy
mkdir -p -m 755 /opt/app/onap/scripts
sudo chown -R $(id -u):$(id -g) /opt/app/onap/
```

Command Executor Running Locally
https://wiki.onap.org/display/DW/Running+Command+Executor+Locally

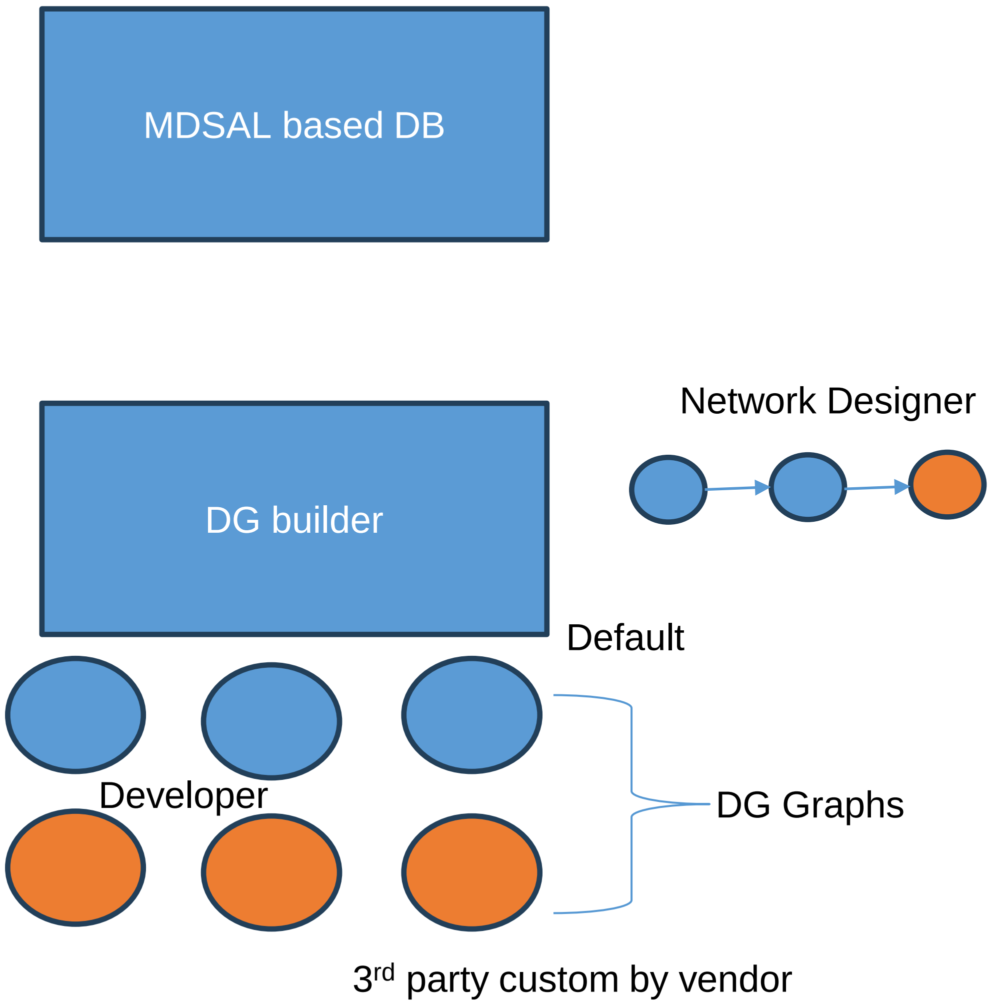
Troubelshooting

Swagger: https://docs.onap.org/projects/onap-ccsdk-cds/en/latest/_downloads/444e8f98bca7be17408b03b51550cba9/cds-bp-processor-api-swagger.json
Postman: https://docs.onap.org/projects/onap-ccsdk-cds/en/latest/_downloads/2566f0533cfd2df155d7a513590db4d5/bp-processor.postman_collection.json

Check CDS Blueprint Processor Logs:

```
kubectl -n onap get pods | grep blueprints-processor | grep Running | cut -f1 -d" " | xargs -i kubectl -n onap logs
kubectl -n onap get pods | grep blueprints-processor | grep Running | cut -f1 -d" " | xargs -i kubectl -n onap logs {} -f
```

SDNC



MDSAL based DB

DG builder

Network Designer

Default

Developer

DG Graphs

3rd party custom by vendor

SDNC vs CDS

Comparison Point	SDNC	CDS
Controller Implementation	via DGs (written using Graphical tool: DGBuilder) **Codeless Development Approach**	via Models (created using CDS UI (except scripts)) **Model Driven Self Service Approach**
Implementation includes	1. DG 2. Template	1. DG 2. Template 3. Script 4. Data Dictionary
Purpose of DG	complete controller/service logic	For controller workflow orchestration
Resource Resolution	as part of DG	via **re-usable** Data Dictionary
Service specific Custom Logic (command execution, Restconf/Netconf based configuration, Ansible configuration etc.)	to be written in DGs	to be written in script (python/kotlin)
Input/Output Modelling	Input: Flexible key:value pair which are not modelled Output: Velocity/Jinja Templates	Input: Modelled Input, Not limited to key(string):value(string)
Re-Usable building blocks for usecases	1. Re-usable Plugins & adapters for DG	1. Re-usable Data dictionary 2. Re-usable capability components/functions 3. Re-usable libraries for scripts

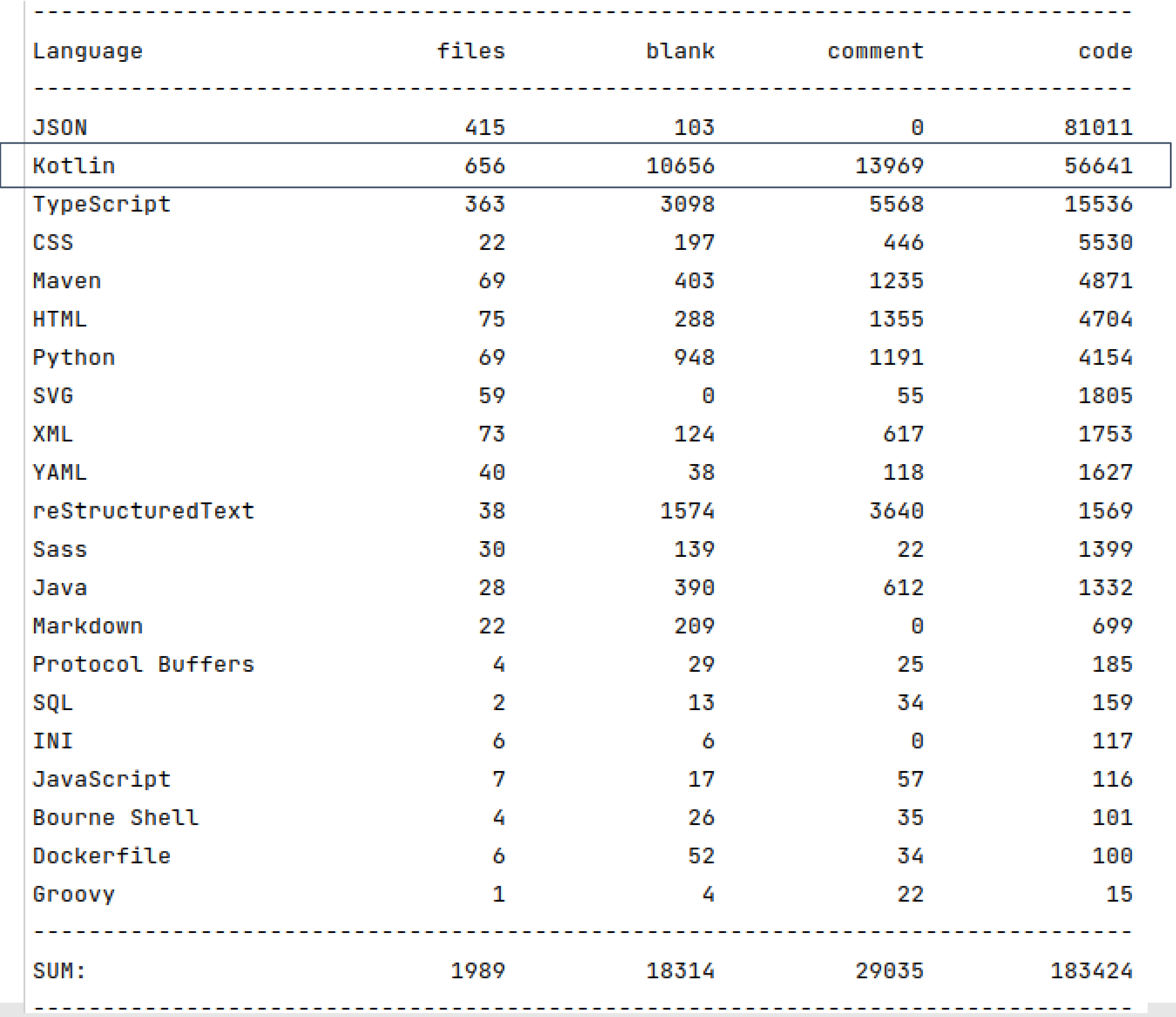
SDNC vs CDS

Comparison Point	SDNC	CDS
Controller Programmability New Service addition at runtime	Possible by adding/modifying DGs (Limitation: Provided resources defined in generic-resource-api is sufficient for new use case)	Possible by creating Blueprint (Even new resource type will have no impact)
Dependency on ODL	SDNC is based on ODL. it is required to upgrade the ODL version for every release.	CDS is not dependent on ODL
Complex logic implementation	Difficult in DG as DG functionalities are limited (Example: Multi Threading)	In Scripts it's possible
Initial learning curve	Medium (only understanding of DG is needed)	High 1. User needs to learn usage of all re-usable entities. (CDS system is little complex and involved much more logic, though CDS UI tries to hide many technical details from users.)
Debugging and maintenance	Only log based troubleshooting is possible, DGs can be run in debug mode Once size of DG grows, its difficult to maintain	Scripts can be debugged and maintained.

Note: For simple logics it's easier to use DG, but if use case demands for complex logics CDS seems to be a better choice. Also the skill set of user is important to decide which controller to use, usually OAM engineers are much aware and comfortable with scripting, compared to DG.

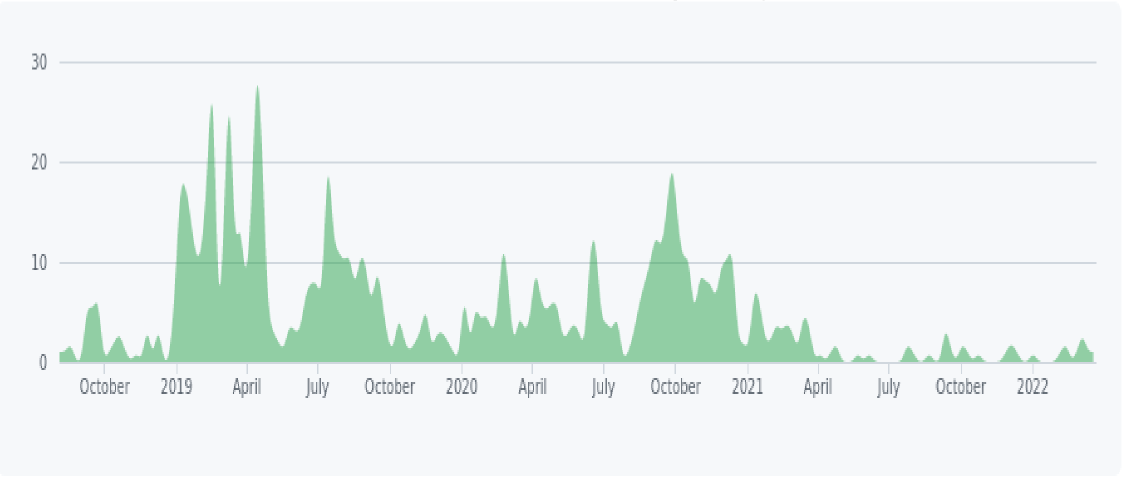
Code/Contribution Overview

https://github.com/onap/ccsdk-cds



```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
JSON                           415            103              0          81011
Kotlin                         656          10656          13969          56641
TypeScript                     363           3098           5568          15536
CSS                             22            197            446           5530
Maven                           69            403           1235           4871
HTML                            75            288           1355           4704
Python                          69            948           1191           4154
SVG                             59              0             55           1805
XML                             73            124            617           1753
YAML                            40             38            118           1627
reStructuredText                38           1574           3640           1569
Sass                            30            139             22           1399
Java                            28            390            612           1332
Markdown                        22            209              0            699
Protocol Buffers                 4             29             25            185
SQL                              2             13             34            159
INI                              6              6              0            117
JavaScript                       7             17             57            116
Bourne Shell                     4             26             35            101
Dockerfile                       6             52             34            100
Groovy                           1              4             22             15
-------------------------------------------------------------------------------
SUM:                          1989          18314          29035         183424
-------------------------------------------------------------------------------
```

Code Contribution Frequency



CDS backend is written in KOTLIN & UI is in Angular

Contributing Organization
CDS was started by AT&T, Bell Canada, Ericsson, IBM had an active participation.

Currently T-Mobile (Marek Szwałkiewicz) is holding meetings, contributions are limited to maintenance.

Thank You