Policy Framework in ONAP

Agenda

- Policy Introduction
  - Policy Framework in ONAP
  - Policy in close loop
- Architecture
  - Level 0 view
  - Level 1 view
- Policy Framework and External Interaction
  - Policy Framework Exposed API
  - Policy Framework Consumed API
- Policy Framework Internal Interaction
- Policy Engine Introduction
  - Policy APEX PDP Engine
  - Policy Drools PDP Engine
  - Policy XACML PDP Engine
- Policy: Installation, Development, Distribution & Execution w.r.t APEX
  - w. Demonstration
- Comparison among different Engines

Introduction: Basic Understanding of Policy Type, Policy and Policy Impl

A Policy Type describes the properties, targets, and triggers that the policy for a feature can have.

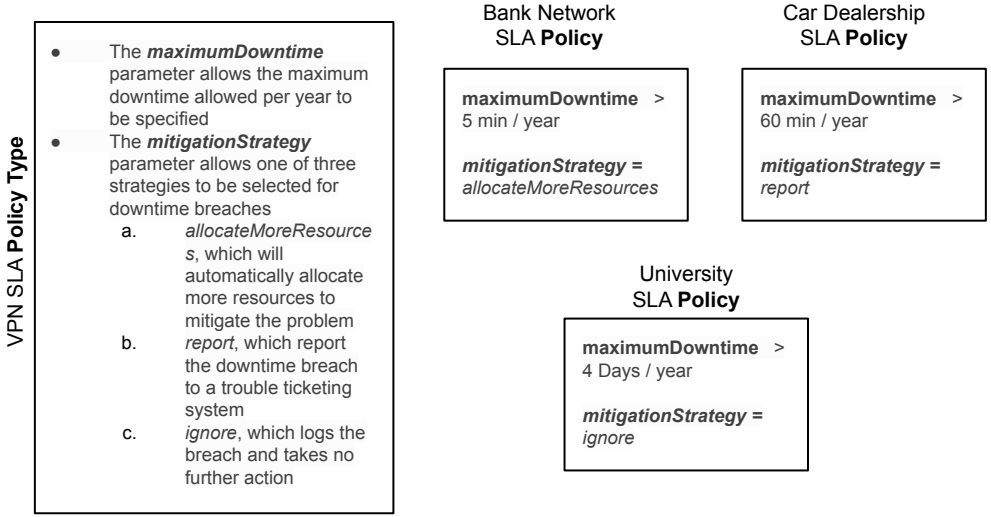
The metadata that specifies:
1. The Properties/configuration data that the policy can take.(May describe default value, optionality etc)
2. The targets such as services, or resources on which a policy of the given type can act.
3. The triggers such as the event type, filtered event, scheduled trigger, or conditions that can activate a policy of the given type.

Policy Types are hierarchical. A Policy Type can inherit from a parent Policy Type, inheriting the properties, targets, and triggers of its parent. A policy type is implementation independent.

A Policy is defined using a Policy Type. The Policy defines:
1. the values for each property of the policy type
2. the specific targets (network elements, functions, services, resources) on which this policy will act
3. the specific triggers that trigger this policy.

A Policy Type Implementation or Raw Policy, is the logic that implements the policy. Policy type implementation is specific to a PDP type.

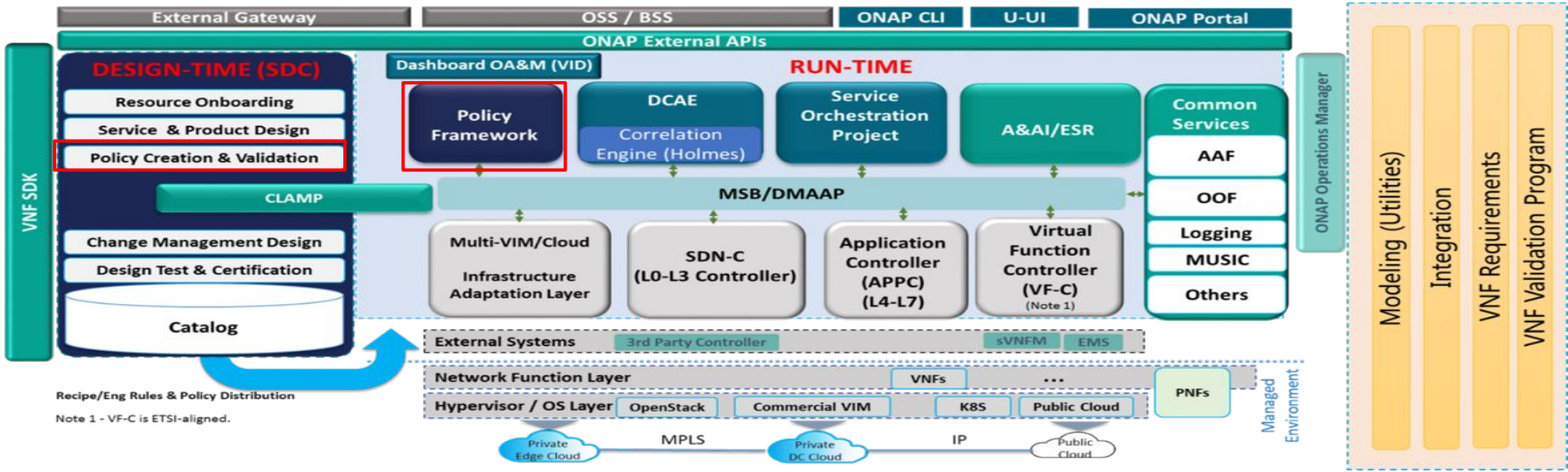
Example: A Policy Type could be written to describe how to manage Service Level Agreements for VPNs. The VPN Policy Type can be used to create VPN policies for a bank network, a car dealership network, or a university with many campuses. The Policy Type has two parameters.



Above diagram show how policy type can create 3 different policy

https://docs.onap.org/projects/onap-policy-parent/en/latest/architecture/architecture.html

Introduction: Policy Framework in ONAP



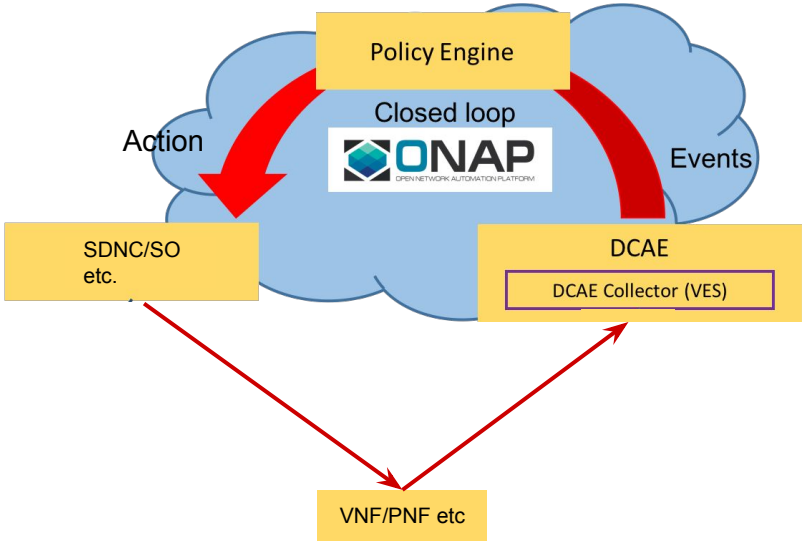
The Policy Framework is the component that is the source of truth for all policy decisions.The ONAP Policy Framework is a comprehensive policy design, deployment, and execution environment.

Policy framework can be applied on closed loop, orchestration, or more traditional open loop use case implementations. Use case implementations such as control loops benefit from ONAP policy framework because they can use the capabilities of the framework to manage and execute their policies rather than embedding the decision making in their applications.

Policy framework comprise of **design** time and **run** time component.

https://docs.onap.org/projects/onap-aai-aai-common/en/istanbul/platform/architecture.html

Policy Framework In Close Loop



The above picture depicts policy in a close loop.

DCAE collects data (Eg. metrics and fault data) through DCAE collectors from the VNFs, PNFs, and computing infrastructure, performing local and global analytics, and generates event. The event has been consumed by Policy engine. Policy engine makes decision based on defined logics/conditions and further trigger action towards actors components such as SDNC, CDS, SO etc.

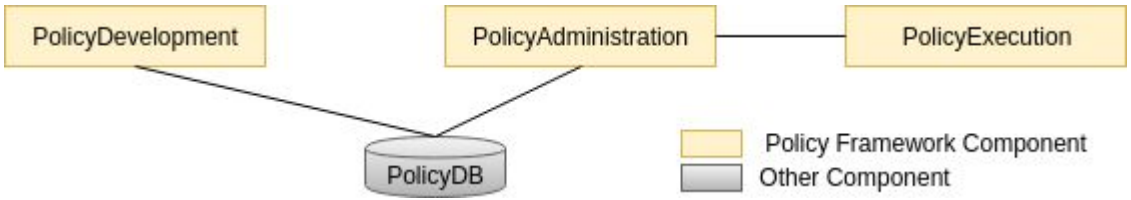
Architecture : Level 0 view

Policy Development component implements the functionality for development of policy types and policies.

Policy Administration
Responsible for

- Deployment life cycle of policies. Example Activate, deactivate, update policy etc.
- Administration of policies at run time; Example Health check, status query etc.

Policy Execution is the execution of a policy in a PDP(Generally a docker container where Policy engine is running)



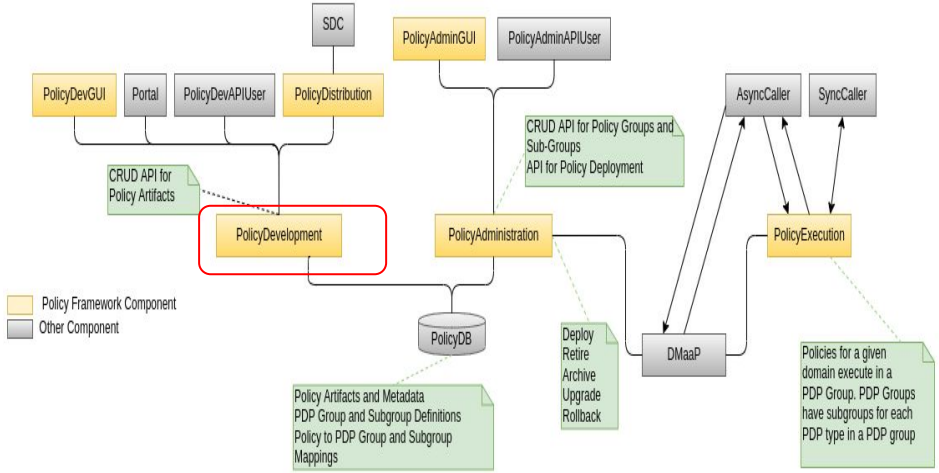
ONAP Policy Framework at its highest level

https://docs.onap.org/projects/onap-policy-parent/en/latest/architecture/architecture.html

Architecture: Policy Development

Policy Development:

- The PolicyDevelopment component implements the functionality for CRUD of policy types and policies.
- The policy types and policy artifacts and their metadata are stored in the PolicyDB.
- The PolicyDevGUI (CLAMP), PolicyDistribution, and other applications can use the PolicyDevelopment API to create, update, delete, and read policy types and policies.



Detailed view of the architecture

https://docs.onap.org/projects/onap-policy-parent/en/latest/architecture/architecture.html

Architecture: Policy Development API

Name	REST API	Description
HealthCheck	GET /policy/api/v1/healthcheck	Returns healthy status of the Policy API component
Statistics	GET /policy/api/v1/statistics	Returns current statistics including the counters of API invocation
Policy Type	GET /policy/api/v1/policytypes	Returns a list of existing policy types stored in Policy Framework
	POST /policy/api/v1/policytypes	Create a new policy type. Client should provide TOSCA body of the new policy type
	GET /policy/api/v1/policytypes/{policyTypeId}	Returns a list of all available versions for the specified policy type
	GET /policy/api/v1/policytypes/{policyTypeId}/versions/{versionId}	Returns a particular version for the specified policy type
	DELETE /policy/api/v1/policytypes/{policyTypeId}/versions/{versionId}	Delete one version of a policy type. It must follow two rules. Rule 1: pre-defined policy types cannot be deleted; Rule 2: policy types that are in use (parameterized by a TOSCA policy) cannot be deleted. The parameterizing TOSCA policies must be deleted first.
	GET /policy/api/v1/policytypes/{policyTypeId}/versions/latest	Returns latest version for the specified policy type
Policy	GET /policy/api/v1/policytypes/{policyTypeId}/versions/{policyTypeVersion}/policies	Returns a list of all versions of specified policy created for the specified policy type version

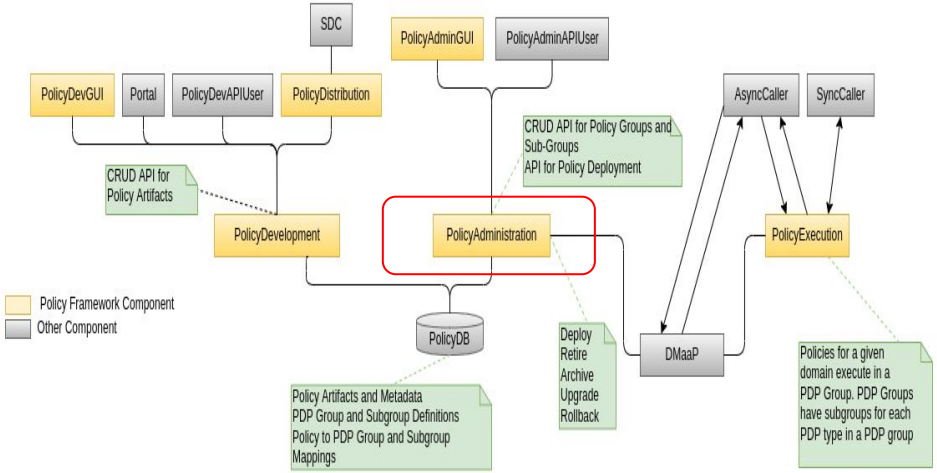
Name	REST API	Description
Policy	POST /policy/api/v1/policytypes/{policyTypeId}/versions/{policyTypeVersion}/policies	Create a new policy for a policy type. Client should provide TOSCA body of the new policy
	GET /policy/api/v1/policytypes/{policyTypeId}/versions/{policyTypeVersion}/policies/{policyId}	Returns a list of all version details of the specified policy
	GET /policy/api/v1/policytypes/{policyTypeId}/versions/{policyTypeVersion}/policies/{policyId}/versions/{policyVersion}	Returns a particular version of specified policy created for the specified policy type version
	DELETE /policy/api/v1/policytypes/{policyTypeId}/versions/{policyTypeVersion}/policies/{policyId}/versions/{policyVersion}	Delete a particular version of a policy. It must follow one rule. Rule: the version that has been deployed in PDP group(s) cannot be deleted
	GET /policy/api/v1/policytypes/{policyTypeId}/versions/{policyTypeVersion}/policies/{policyId}/versions/latest	Returns the latest version of specified policy
	GET /policy/api/v1/policies/{policyId}/versions/{policyVersion}	Returns a particular version of specified policy
	DELETE /policy/api/v1/policies/{policyId}/versions/{policyVersion}	Rule: the version that has been deployed in PDP group(s) cannot be deleted
	GET /policy/api/v1/policies	Returns all version of available policies
	POST /policy/api/v1/policies	Create one or more new policies. Client should provide TOSCA body of the new policies

https://docs.onap.org/projects/onap-policy-parent/en/latest/api/api.html

Architecture: Policy Administration

Policy Administration handles PDPs and policy allocation to PDPs using asynchronous messaging over DMaaP. It provides three type of APIs:

- CRUD APIs for policy groups and subgroups
- APIs to allocate of policies to PDP groups and subgroups.
- API allows policy execution to be managed, showing the status of policy execution on PDP Groups, subgroups, and individual PDPs as well as the life cycle state of PDPs.



Detailed view of the architecture

https://docs.onap.org/projects/onap-policy-parent/en/latest/architecture/architecture.html

Architecture: Policy Administration (PAP) API

REST API (User to PAP)

Operation	Description
Health check	Queries the health of the PAP
Consolidated healthcheck	Queries the health of all policy components
Statistics	Queries various statistics
PDP state change	Changes the state of all PDPs in a PDP Group
PDP Group create/update	Creates/updates PDP Groups
PDP Group delete	Deletes a PDP Group
PDP Group query	Queries all PDP Groups
Deployment update	Deploy/undeploy one or more policies in specified PdpGroups
Deploy policy	Deploys one or more policies to the PDPs
Undeploy policy	Undeploys a policy from the PDPs
Policy Status	Queries the status of all policies
Policy deployment status	Queries the status of all deployed policies
PDP statistics	Queries the statistics of PDPs
Policy Audit	Queries the audit records of policies

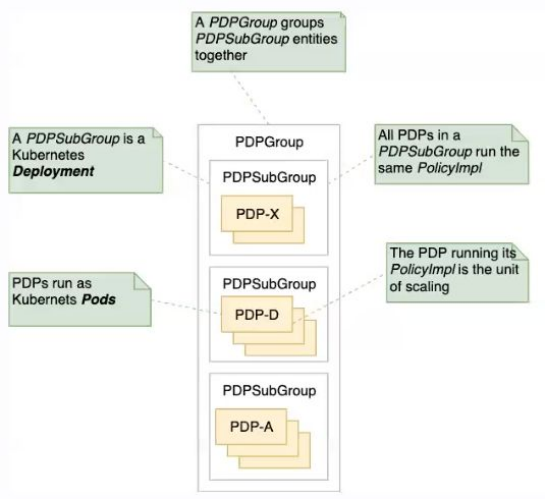
DMaaP API (PAP - PDP internal communication)

Message	Direction	Description
PDP status	Incoming	Registers a PDP with PAP; also sent as a periodic heart beat; also sent in response to requests from the PAP
PDP update	Outgoing	Assigns a PDP to a PDP Group and Subgroup; also deploys or undeploys policies from the PDP
PDP state change	Outgoing	Changes the state of a PDP or all PDPs within a PDP Group or Subgroup

In addition, PAP generates notifications via the DMaaP Message Router when policies are successfully or unsuccessfully deployed (or undeployed) from all relevant PDPs.

https://docs.onap.org/projects/onap-policy-parent/en/istanbul/pap/pap.html#

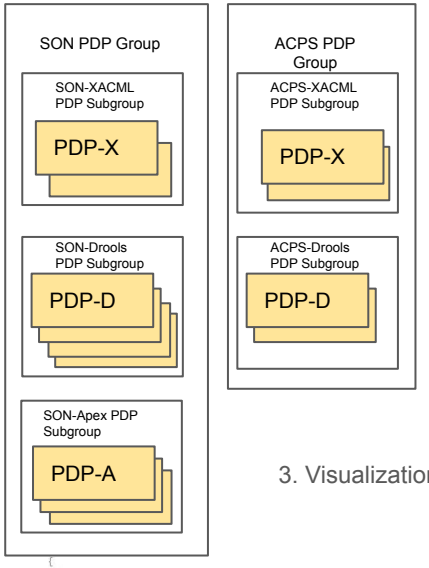
Architecture: Policy Execution



1. Grouping of PDPs

PDP Group	PDP Subgroup	Kubernetes Deployment	Kubernetes Deployment Strategy	PDPs in Pods
SON	SON-XACML	SON-XACML-Dep	Always 2, be geo redundant	2 PDP-X
	SON-Drools	SON-Drools-Dep	At Least 4, scale up on 70% load, scale down on 40% load, be geo-redundant	>= 4 PDP-D
	SON-APEX	SON-APEX-Dep	At Least 3, scale up on 70% load, scale down on 40% load, be geo-redundant	>= 3 PDP-A
ACPS	ACPS-XACML	ACPS-XACML-Dep	Always 2	2 PDP-X
	ACPS-Drools	ACPS-Drools-Dep	At Least 2, scale up on 80% load, scale down on 50% load	>=2 PDP-D

2. Example Requirements Grouping of PDPs



3. Visualization Grouping of PDPs

Note: There are 3 types of PDP engines provided by ONAP. APEX, DROOL and XACML.
User can add their own PDP engines too.


```
{
  "groups": [
    {
      "description": "This group should be used for managing all control loop related policies and pdps of SON use case",
      "name": "SON-PDP-Group",
      "pdpGroupState": "ACTIVE",
      "pdpSubgroups": [
        {
          "currentInstanceCount": 0,
          "desiredInstanceCount": 2,
          "pdpInstances": [],
          "pdpType": "apex",
          "policies": [],
          "properties": {},
          "supportedPolicyTypes": [
            {
              "name": "onap.policies.controlloop.operational.common.Apex",
              "version": "1.0.0"
            }
          ]
        },
        {
          "currentInstanceCount": 0,
          "desiredInstanceCount": 4,
          "pdpInstances": [],
          "pdpType": "drools",
          "policies": [],
          "properties": {},
          "supportedPolicyTypes": [
            {
              "name": "onap.policies.controlloop.operational.common.Drools",
```


4. A sample query to PAP showing PDP groups and sup groups etc

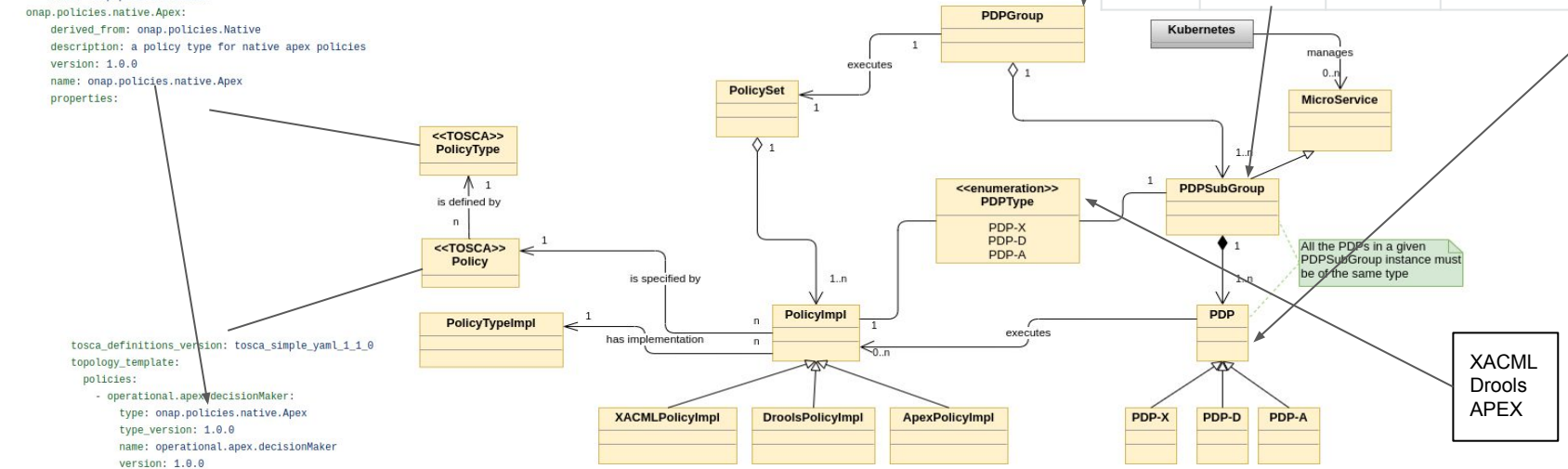
PolicyExecution is the set of running PDPs that are executing policies, logically partitioned into PDP groups and subgroups.
Above picture(s) showing grouping of policy runtime entities.

1. This way of grouping helps PAP to manage life cycles of PDPs for a particular use case easily.
2. Also it helps easy horizontal scaling of PDPs based on load or geo redundancy

Policy Framework Object Model

PDP Group	PDP Subgroup	Kubernetes Deployment	Kubernetes Deployment Strategy	PDPs in Pods
SON	SON-XACML	SON-XACML-Dep	Always 2, be geo redundant	2 PDP-X
	SON-Drools	SON-Drools-Dep	At Least 4, scale up on 70% load, scale down on 40% load, be geo-redundant	>= 4 PDP-D
ACPS	ACPS-XACML	ACPS-XACML-Dep	Always 2	2 PDP-X

```
tosca_definitions_version: tosca_simple_yaml_1_1_0
policy_types:
    onap.policies.Native:
        derived_from: tosca.policies.Root
        description: a base policy type for all native PDP policies
        version: 1.0.0
        name: onap.policies.Native
    onap.policies.native.Apex:
        derived_from: onap.policies.Native
        description: a policy type for native apex policies
        version: 1.0.0
        name: onap.policies.native.Apex
        properties:
```




```
tosca_definitions_version: tosca_simple_yaml_1_1_0
topology_template:
  policies:
    - operational.apex.decisionMaker:
        type: onap.policies.native.Apex
        type_version: 1.0.0
        name: operational.apex.decisionMaker
        version: 1.0.0
        metadata:
          metadataSetName: apexMetadata_decisionMaker
          metadataSetVersion:  1.0.0
```

UML class diagram above shows the relationship between different entities like pdp, pdp group, pdp subgroup, policy set etc.

https://docs.onap.org/projects/onap-policy-parent/en/latest/architecture/architecture.html

https://github.com/onap/policy-models/tree/master/models-examples/src/main/resources

Policy Types Supports by Policy Engine in ONAP

- These Policy Types are supported by XACML PDP
  - Monitoring Policy Type supported by the XACML PDP
    - Tca (threshold crossing), datafile-app-server
    - Used in Control Loop
    - Integrated with CLAMP/DCAE
  - Guard policy Type for protecting regarding Control Loop operations
    - Blacklist, frequency limiters, min/max, coordination
    - Used in control loop (optional)
    - Integrated with Clamp
  - Optimization Policy Type for OOF Services
    - All inherit from service and resource specific policy type
    - Affinity, distance, hardware placement, optimization algorithm, VIM fit, vnf, pci query, subscriber
  - Naming Policy Type for SDNC Naming Services
- These Policy Types are supported by Drools and Apex PDP
  - Operational Policy Types for enforcement of Control Loop operations
    - Used in Control Loops
    - Integrated into CLAMP

https://wiki.onap.org/pages/viewpage.action?pageId=84654718&preview=/84654718/84655264/2020-06-26-policy-tutorial-overview.mp4

Preloaded Policy Types in ONAP

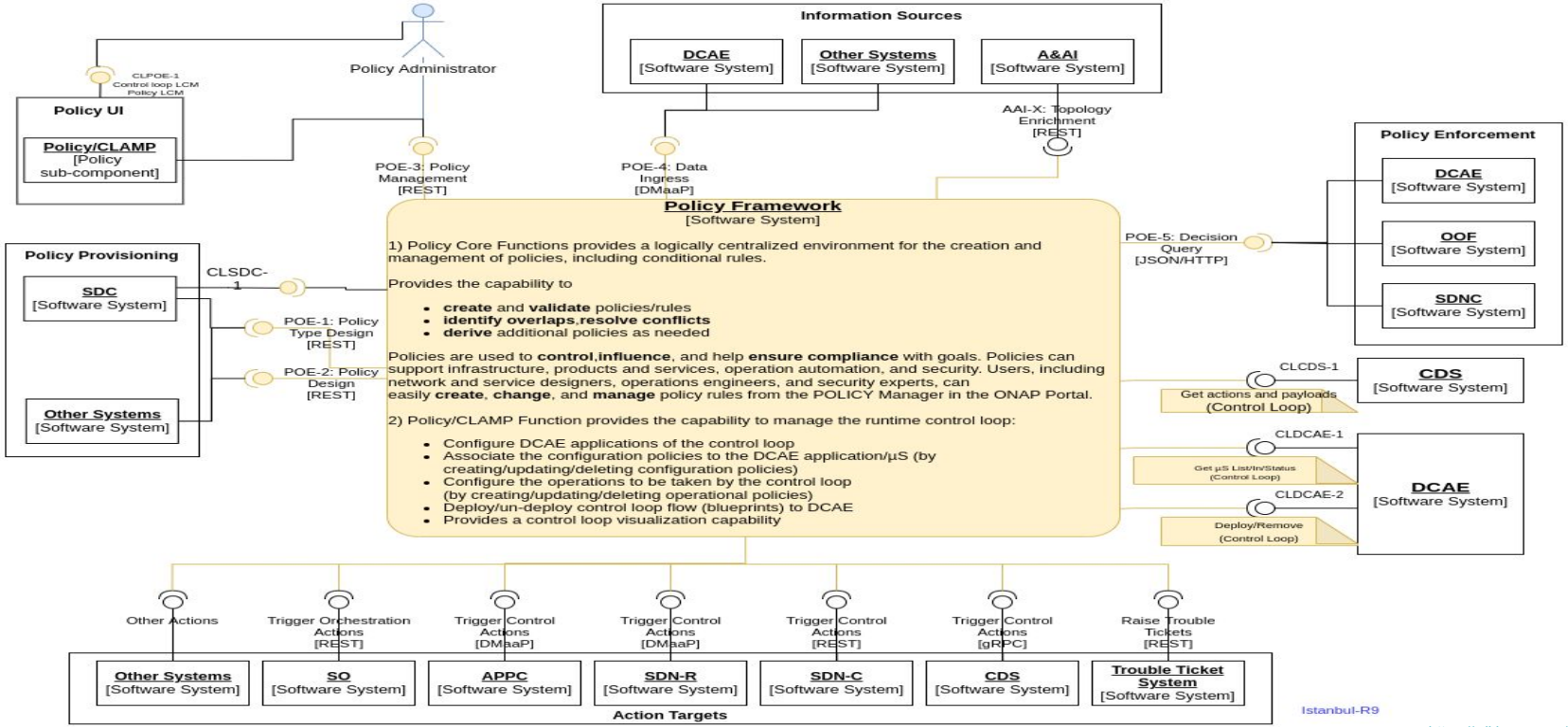
Policy Types	Description
onap.policies.Monitoring	This is a base Policy Type that supports Policy driven DCAE microservice components used in a Control Loops. The implementation of this Policy Type is developed using the XACML PDP to support question/answer Policy Decisions during runtime for the DCAE Policy Handler.
onap.policies.controlloop.Operational	This policy type is used to support actor/action operational policies for control loops. There are two types of implementations for this policy type 1.Drools implementations that supports runtime Control Loop actions taken on components such as SO/APPC/VFC/SDNC/SDNR 2.Implementations using APEX to support Control Loops.
onap.policies.controlloop.Guard	This policy type is the the type definition for Control Loop guard policies for frequency limiting, blacklisting and min/max guards to help protect runtime Control Loop Actions from doing harm to the network.
onap.policies.Optimization	The Optimization Base Policy Type supports the OOF optimization policies.
onap.policies.Naming	Naming policies are used in SDNC to enforce which naming policy should be used during instantiation.
onap.policies.Native	Base Native Policy Type for PDPs to inherit from in order to provide their own native policy type.

Onap Provided a set of preloaded policy types. The list of types can be found here:
https://docs.onap.org/projects/onap-policy-parent/en/istanbul/api/api.html#policy-preload-label

User can define their own policy types. But has to modify the PDP (code level changes needed) in such a way that it can understand how to handle this new type.

https://docs.onap.org/projects/onap-policy-parent/en/latest/design/design.html

Policy Framework API and External Interaction



High Level Component Definition and Architectural Relationships

https://wiki.onap.org/display/DW/ARC+Policy+Framework+Component+Description+-+Istanbul-R9

Policy Framework Exposed API

Interface Name	Definition	Used By	Capabilities
POE-1	Policy Type Design	SDC/ Other system	Allows applications to create, update, delete, and query *PolicyType* entities so that they become available for use in ONAP by applications such as CLAMP.
POE-2	Policy Design	SDC/ Other system	Allows applications (such as CLAMP and Integration) to create, update, delete, and query *Policy* entities.
POE-3	Policy Administration	Policy/ CLAMP	Support CRUD of PDP groups and subgroups and to support the deployment and life cycles of *PolicyImpl* entities (TOSCA *Policy* and *PolicyTypeImpl* entities) on PDP sub groups and PDPs.
POE-4	Data Ingress	DCAE/ Other system/ A&AI	Listen on a DMaaP topic.
POE-5	Decision Query	DCAE/ OOF/ SDNC	Policy decisions are required by ONAP components to support the policy-driven ONAP architecture. Policy Decisions are implemented using the XACML and Apex PDPs. The calling application (which may be another policy – e.g. invocation of a guard policy from PDP-D) must provide attributes in order for the PDP to return a correct decision.
CLPOE-1	Control Loop Lifecycle Management and Policy Lifecycle Management Interface		A user interface (GUI) for: • Selecting the control loop flow • Entering configuration policy parameters • Entering operational policy parameters • Managing life cycle of DCAE control flow blueprint • Selecting a Service to associate to a Control Loop to be instantiated • CRUD operation on Policy (outside of Control Loop)

Policy Framework Consumed API

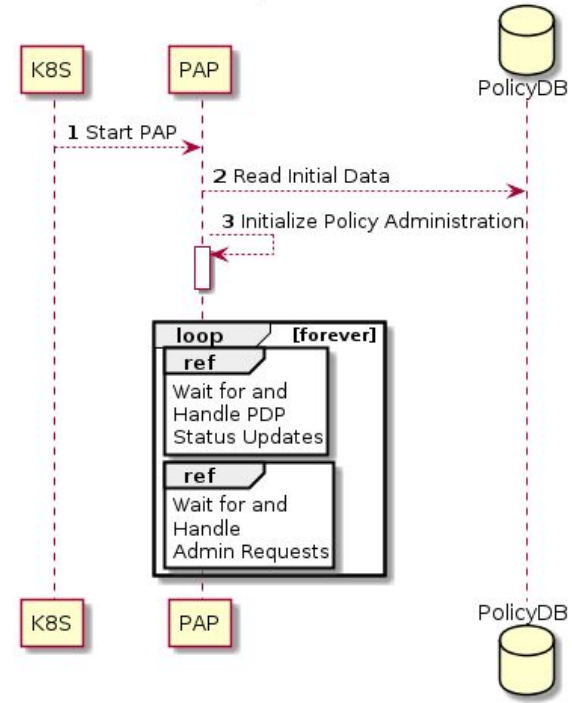
Interface Name	Consumed by	Capabilities
AAF	Policy Framework	Authentication and authorization
DMaaP	Policy Framework Policies	Policy framework uses DMaaP for SDC subscriptions and internal communication. Policies use DMaaP as a transport for contextual information from various sources
CLSDC-1	Policy/CLAMP	• Notification of CSAR; Retrieval of CSAR • To receive the Control Loop Blueprint from SDC
CLDCAE-1	Policy/CLAMP	• Retrieve DCAE application status • Retrieve DCAE µS lists • Retrieve DCAE µS description and blueprints
CLDCAE-2	Decision Query	Deploy/remove DCAE µS.
CLCDS-1	Policy/CLAMP	Get list of operations/actions and corresponding payload for Operational Policy where selected actor is "CDS".
AAI	Policies	Enrich ingress data with topology information
SO	Policies	Trigger orchestration actions (policy driven)
SDNC APPC CDS	Policies	Trigger control actions (policy driven)
Other	Policies	Trigger any interface defined in a policy, for example, trouble ticketing

Policy Framework Components Internal Interaction

Interactions between Policy Framework Components

Interactions between Policy Framework components themselves and with other ONAP components at startup, shutdown and restart.

Figure 1: PAP Startup and Shutdown



PAP startup as mention in Figure 1

PAP shutdown is trivial. On receipt or a shutdown request, the PAP completes or aborts any ongoing operations and shuts down gracefully.

Figure 2: PDP Startup and Shutdown

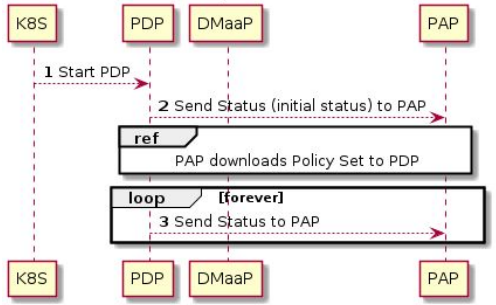
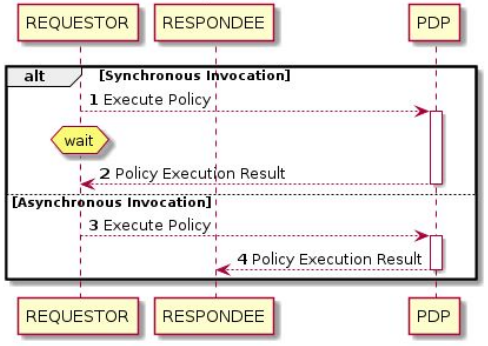


Figure 3: Policy Execution



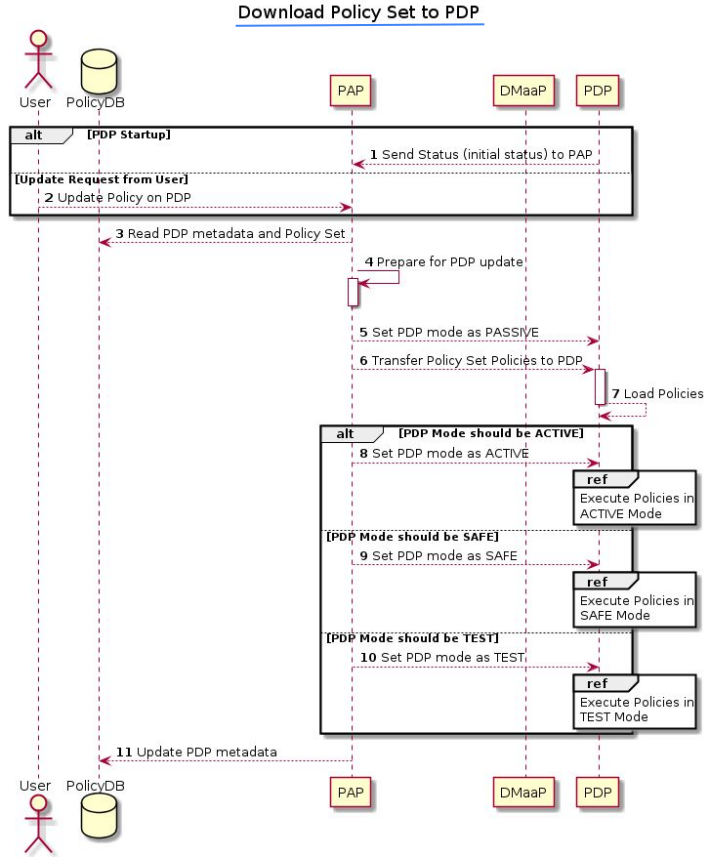
PDP startup as mention in Figure 2

PDP shutdown. On receipt or a shutdown request, the PDP completes or aborts any ongoing policy executions and shuts down gracefully.

NOTE: PDP-X and PDP-A implement synchronous policy execution.
PDP-D and PDP-A implement asynchronous policy execution.

https://docs.onap.org/projects/onap-policy-parent/en/latest/architecture/architecture.html

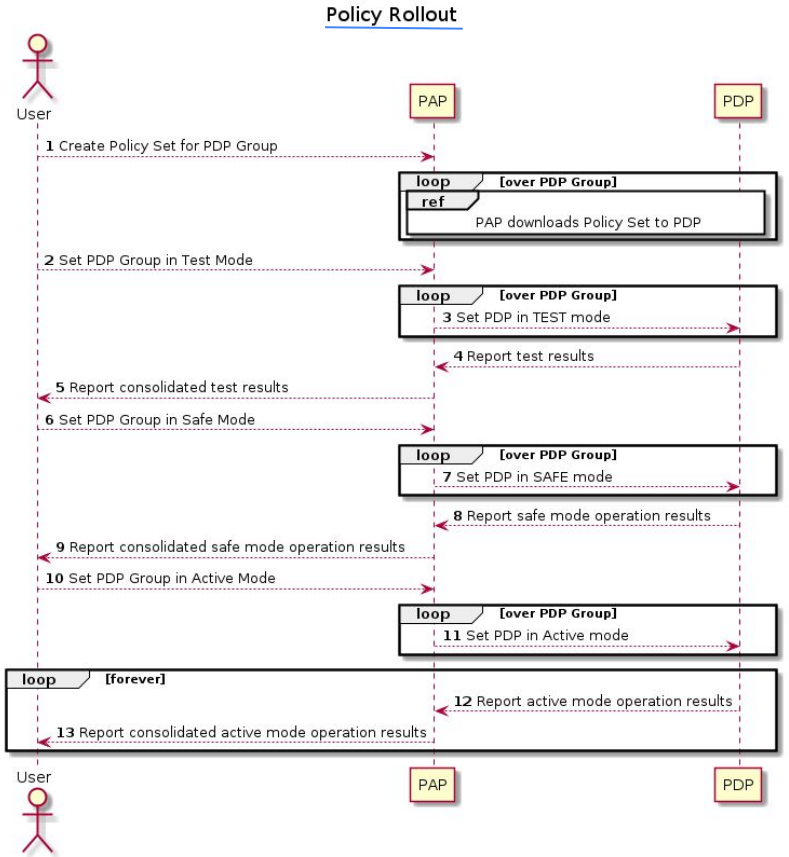
Policy Lifecycle Management



Different lifecycle mode:

1. PASSIVE MODE
2. ACTIVE MODE
3. SAFE MODE*
4. TEST MODE*

* SAFE Mode and TEST Mode will be implemented in future versions of the Policy Framework.



https://docs.onap.org/projects/onap-policy-parent/en/latest/architecture/architecture.html

Policy Engine

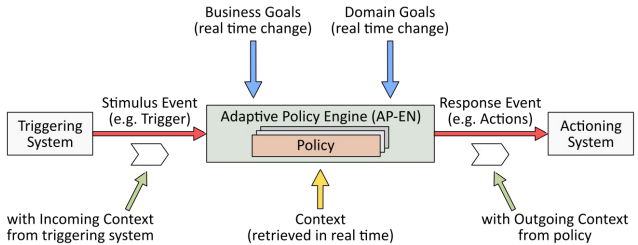
Policy Engine Introduction: APEX

APEX stand for Adaptive Policy EXecution. It is a lightweight engine for execution of policies. APEX allows you to specify logic as a policy, logic that you can adapt on the fly as your system executes. Example you might adapt your of policy flow based on some contextual information which is available during runtime, such as predictive analytics result or time of day etc. This makes APEX different from other policy engine.

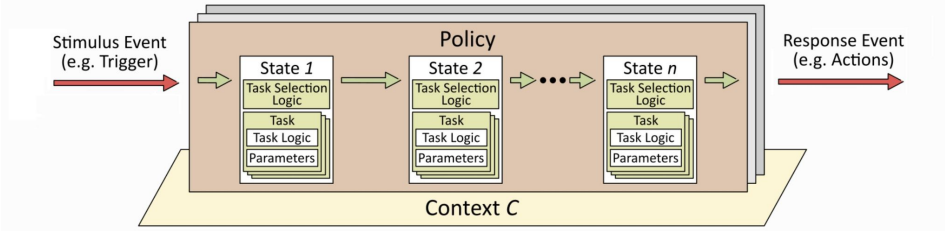
APEX is an open source project, originally initiated by ERICSSON.

Some Key Concepts:

- The APEX Policy Engine in APEX runs your policies. These policies are triggered by **incoming events**. Example a physical link state change event from DCAE.
- The logic (or calls it **Task**) of the policies executes and produces a **Action events**. For example a task can check if link state change is down then it produce a response event such as “notify SDNC”. This event will be act as an **action.**
- The **task** accomplishes the Business/domain goals.
- The Incoming **Context** on the incoming event and the Outgoing Context on the outgoing event are simply the fields and attributes of the event. You design the policies that APEX executes and the trigger and action events that your policies accept and produce. Also context can be build during runtime (Example using some predictive algorithm, env variable, current time etc.).
- The **Task Selection logic** may take help of context and can decide different task and flow of policy in runtime if required.



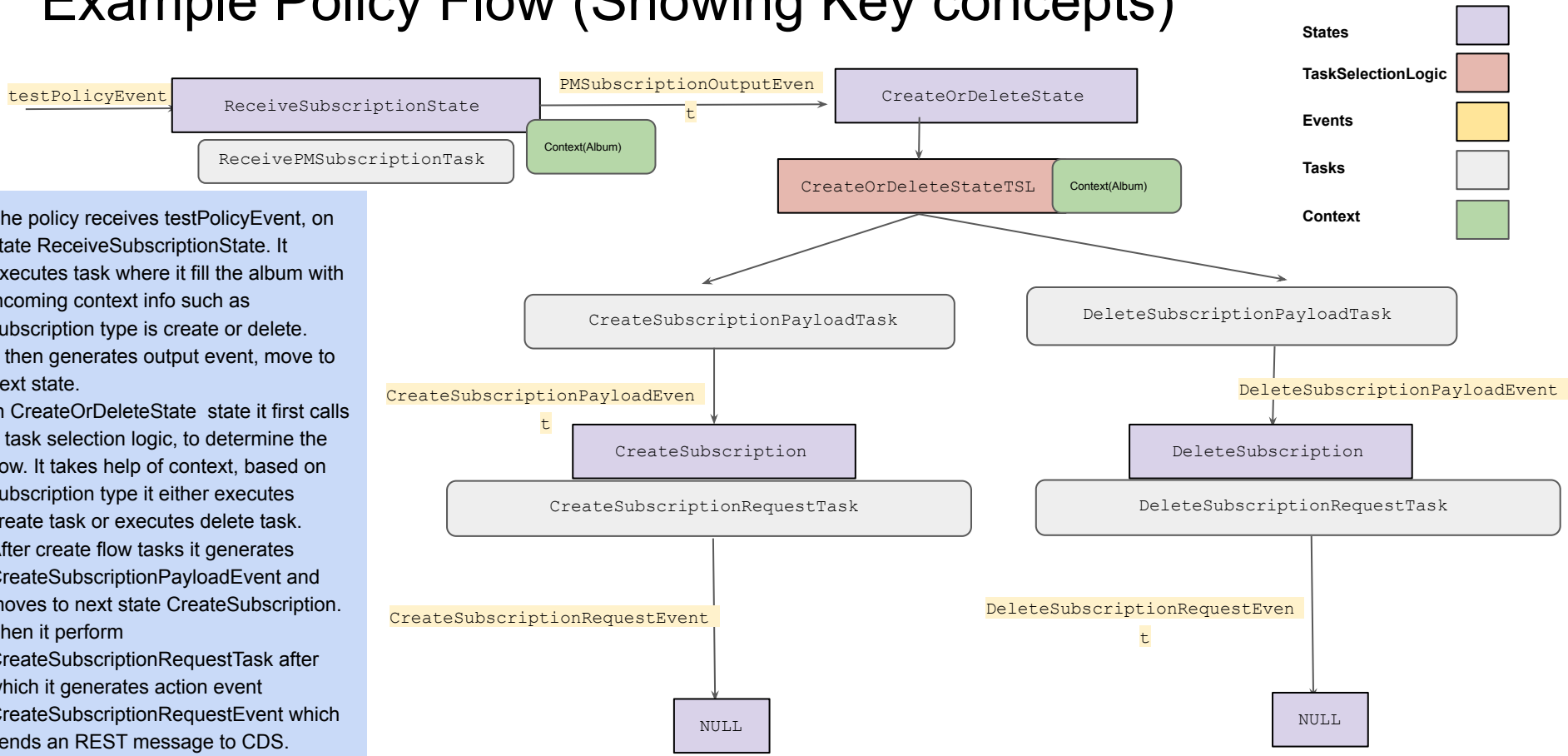
Simple APEX Overview



Policy execution in APEX

https://wiki.onap.org/pages/viewpage.action?pageId=84651833

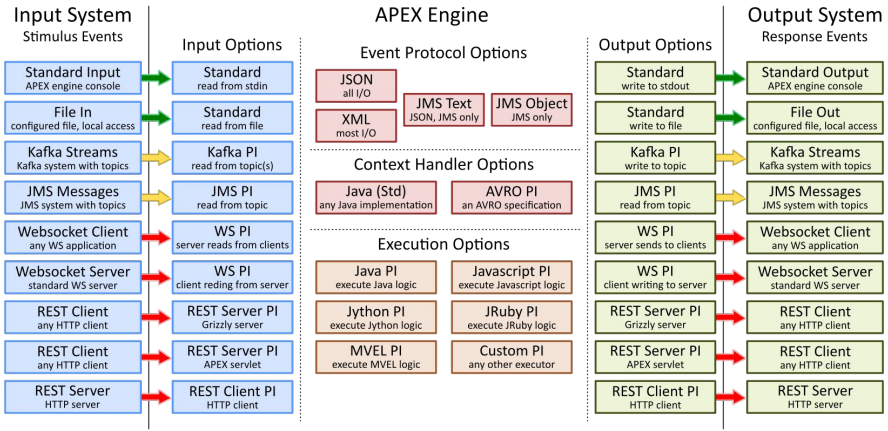
Example Policy Flow (Showing Key concepts)



The policy receives testPolicyEvent, on state ReceiveSubscriptionState. It executes task where it fill the album with incoming context info such as subscription type is create or delete.
It then generates output event, move to next state.
In CreateOrDeleteState state it first calls a task selection logic, to determine the flow. It takes help of context, based on subscription type it either executes create task or executes delete task.
After create flow tasks it generates CreateSubscriptionPayloadEvent and moves to next state CreateSubscription.
Then it perform CreateSubscriptionRequestTask after which it generates action event CreateSubscriptionRequestEvent which sends an REST message to CDS.

APEX PDP Engine - Pluggable Architecture

- System built using a plugin architecture. Each configuration option is realized by a plugin, which can be loaded and configured when the engine is started. New plugins can be added to the system at any time, though to benefit from a new plugin, an engine will need to be restarted.
- An APEX engine can be configured various combinations of event input handlers, event output handlers, event protocols, context handlers, and logic executors.



APEX Configuration Matrix

https://docs.onap.org/projects/onap-policy-parent/en/latest/apex/APEX-Introduction.html

Policy Engine Introduction: Drools

```
rule  <rule_name>
   <attribute> <value>

   when
      <conditions>

   then
      <actions>
end
```

Drool File syntax

- Drools is a Business Rules Management System (BRMS) solution.
- Drools is open source software, released under the Apache License 2.0. It is written in 100% pure Java™, runs on any JVM and is available in the Maven Central repository too.

Key Concepts:

- PDP-D functionality has been partitioned into two functional areas:
    - PDP-D Engine is the infrastructure that policy applications use.
    - PDP-D Applications, ie. a *controller*, contains references to the resources that the application needs.
        - The **Controllers** are defined in <name>-controller.properties files. These file includes references to **communication endpoints, maven artifact coordinates, and *coders*** for message mapping.
        - **Communication Endpoints** describes source (of incoming controller traffic) and sink (of outgoing controller traffic) configuration.
        - Filtering and mapping rules for incoming and outgoing messages known as **coders**.
        - The **maven artifact coordinates** section (rules) points to the controller-usecases kjar artifact. This kJar artifacts consists of drool file, pom dependency files, kmodule.xml.
            - The drools file where we implement the policy.
            - Kmodule.xml where we specify a session, and policy type. A session is a flow, which takes the event and fires a rule.
            - Dependencies describes various dependencies eg. actors framework dependencies.

https://docs.onap.org/projects/onap-policy-parent/en/istanbul/drools/drools.html
https://www.drools.org/
https://docs.onap.org/projects/onap-policy-parent/en/istanbul/drools/pdpdEngine.ht
https://docs.onap.org/projects/onap-policy-parent/en/istanbul/drools/pdpdApps.htm

https://docs.onap.org/projects/onap-policy-parent/en/latest/xacml/xacml.html
https://docs.onap.org/projects/onap-policy-parent/en/latest/xacml/xacml.html#match-policy-type

Policy Engine Introduction: XACML

- The ONAP XACML Policy PDP Engine uses an open source implementation of the OASIS XACML 3.0 Standard to support policy decisions in the ONAP.
- ONAP XACML PDP translates TOSCA Compliant Policies into the XACML policy language, loads the policies into the XACML engine
- XACML applications can be built to support Policy Types that require an ONAP component to query the Decision API
  - Simple question/answer Decisions to support an ONAP component to be policy-driven
    - “What policy(s) should my app enforce given these conditions?”
- There are various standard policy translators available to use.
  - Matchable: gives the user the ability to designate which properties are matched in a decision
  - Guard: These Policy Types are used by Control Loop Drools Engine to support guarding control loop operations and coordination of Control Loops during runtime control loop execution.

```
tosca_definitions_version: tosca_simple_yaml_1_1_0
policy_types:
   onap.policies.match.Test:
      derived_from: onap.policies.Match
      version: 1.0.0
      name: onap.policies.match.Test
      description: Test Matching Policy Type to tes
      properties:
         matchable:
            type: string
            metadata:
               matchable: true
            required: true
         nonmatchable:
            type: string
            required: true
```

Matchable: Policy Type

```
tosca_definitions_version: tosca_simple_yaml_1_1_0
topology_template:
    policies:
    -   test_match_1:
            type: onap.policies.match.Test
            version: 1.0.0
            type_version: 1.0.0
            name: test_match_1
            properties:
               matchable: foo
               nonmatchable: value1
    -   test_match_2:
            type: onap.policies.match.Test
            version: 1.0.0
            type_version: 1.0.0
            name: test_match_2
            properties:
               matchable: bar
               nonmatchable: value2
```

Matchable: Policy

```
{
  "ONAPName": "my-ONAP",
  "ONAPComponent": "my-component",
  "ONAPInstance": "my-instance",
  "requestId": "unique-request-1",
  "action": "match",
  "resource": {
      "matchable": "foo"
  }
}
```

Matchable: Decision API Request

```
{
  "policies": {
    "test_match_1": {
      "type": "onap.policies.match.Test",
      "type_version": "1.0.0",
      "properties": {
        "matchable": "foo",
        "nonmatchable": "value1"
      },
      "name": "test_match_1",
      "version": "1.0.0",
      "metadata": {
        "policy-id": "test_match_1",
        "policy-version": "1.0.0"
      }
    }
  }
}
```

Matchable: Decision API Response

Policy: Installation, Development, Distribution & Execution w. example as APEX

Policy Environment Preparation : APEX

Docker Based (Development)

1. Java and maven should be install in your environment
2. Use the settings.xml found in the oparent repo
3. Save this Maven settings.xml as your ~/.m2/settings.xml
4. To compile and Build code:

   mvn clean install -Pdocker -DskipTests

5. Now run the docker using following command.

   docker run -it --rm onap/policy-apex-pdp:latest

Oom Based (Deployment)

1. Go to oom/kubernetes/onap folder
2. Edit the onap/values.yaml

Below components required to install policy runtime components.

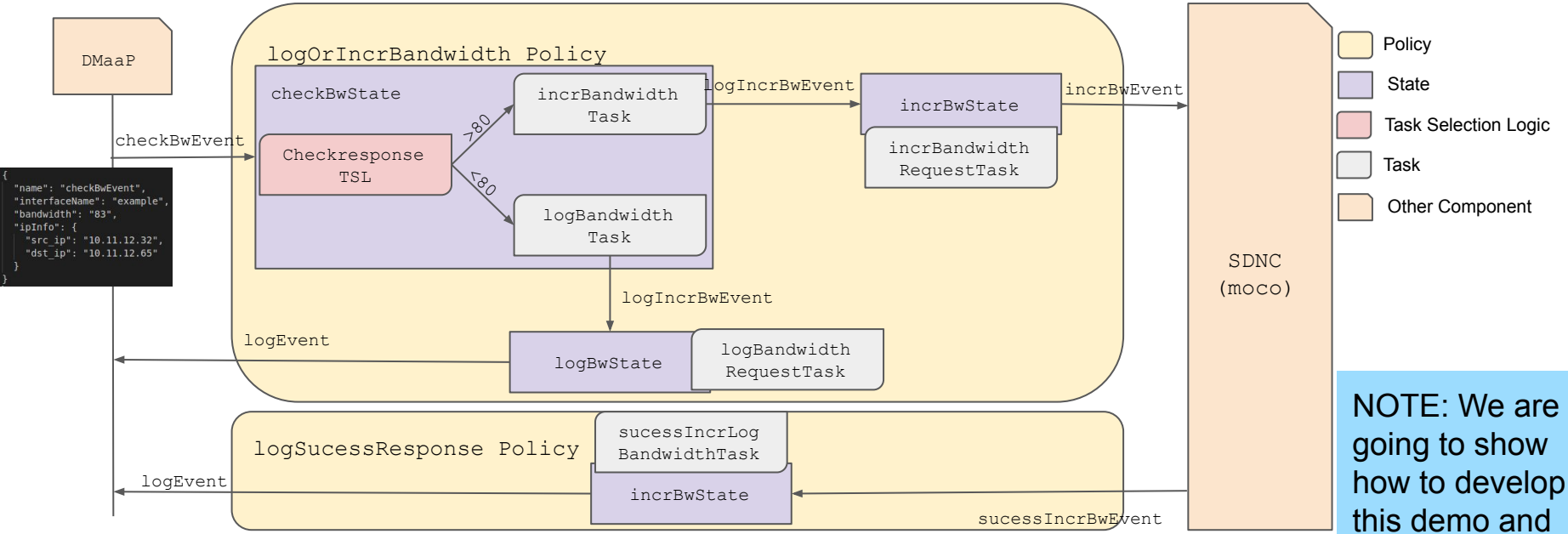
aaf:
  persistence:
    storageClassOverride: "nfs-client"
aaf:
  enabled: true
dmaap:
  enabled: true
policy:
  enabled: true

NOTE: The deployment environment steps are same for all policy engine types

https://wiki.onap.org/display/DW/Setting+Up+Your+Development+Environment

APEX PDP Policy Demo

Use case: Notify SDNC about bandwidth change event if bandwidth utilization is above 80%



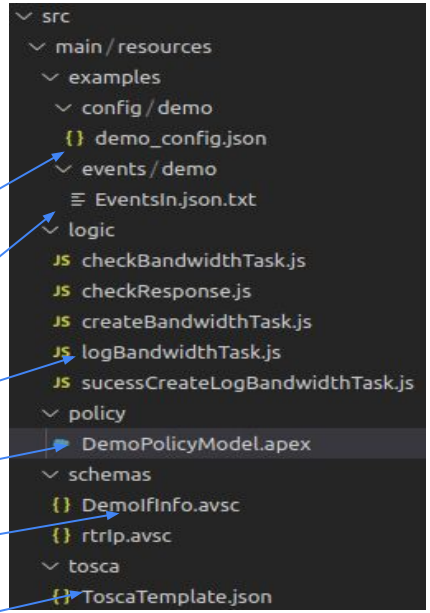
1. DCAE send the bandwidth utilization event to APEX policy via DMaap
2. It check bandwidth utilization.
   a. If utilization is below 80% then it log that event
   b. If utilization is above 80% then it send the event to SDNC to increase bandwidth link. SDNC increase bandwidth link and then it will reply to APEX policy. Then APEX Policy log that information.

NOTE: We are going to show how to develop this demo and then how to deploy it in next slides.

APEX Policy Development guide

Below files are required to develop an APEX policy

File Name	Description
APEX Configuration file (JSON file)	Enables the plugins required to execute the policy. It contains various combinations of event input handlers, event output handlers, event protocols, context handlers, and logic executors.
EventsIn.json	The Sample event file for reference.
JavaScript file	There are multiple files. Each file have some specific task. Some file define the logic for task for input and output event while some file define the logic for selection of tasks, states.
APEX file(.apex)	Defines the Policy. (Defines schemas, events, tasks, states and policy flows)
Avro file	It help to define complex schema. (Optional)
Tosca Template File	Template TOSCA file define the template of policy like policy type, version etc. It's a placeholder. Actual policy is embed in this file after compilation. Will be used to generate final executable Policy file.



Apex follows the above directory Structure for Policy Development

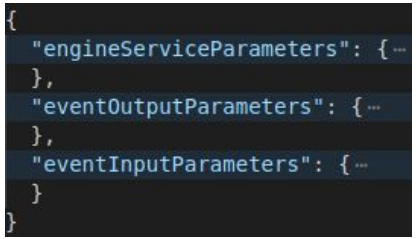
Apex provides below tools for development of policies

Tools	Description
apexCLIToscaEditor.sh	A tools to generate executable policy file. It takes apex policy(.apex), and configuration file, and tosca template as input and produce final tosca policy file in json format.
apexApps.sh	Starts engine, takes tosca policy file as input.

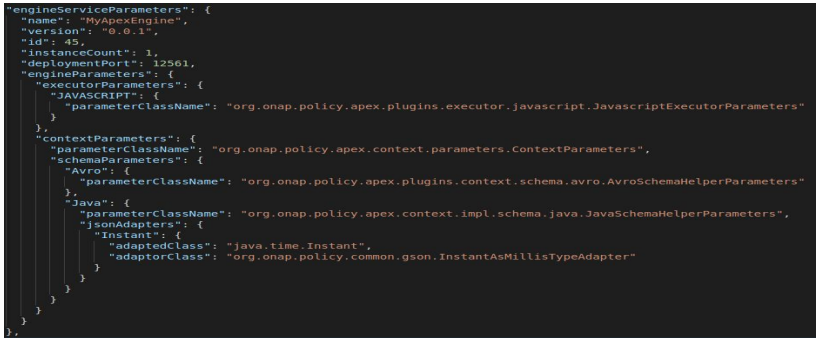
https://docs.onap.org/projects/onap-policy-parent/en/latest/apex/APEX-User-Manual.html

APEX Policy Development guide (Apex Configuration File)

APEX Config (JSON file)



engineServiceParameters

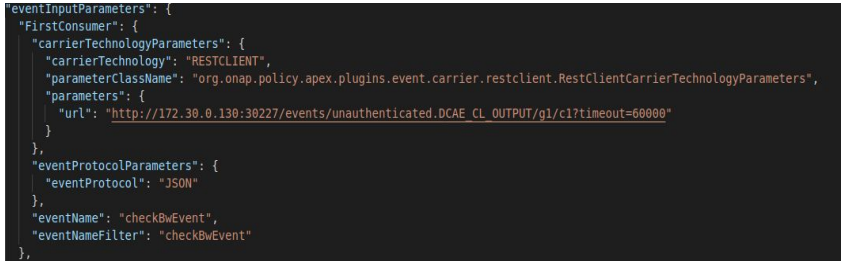


InstanceCount	The number of threads (policy instances executed in parallel) the engine should use, use 1 for single threaded engines
Executor Parameters	Points towards helper parser of task files technologies, ex Javascript
Context parameters	Points towards helper parse for schema tech, ex Avro.

eventOutputParameters

Carrier technology	Describes the plugins types of this output (Action events). Example RESTCLIENT. Also provides protocols example: JSON, XML etc. EventnameFilter to identify map the action events to this output.

eventInputParameters



Carrier technology	Describes the plugins types of this input (Action events). Example RESTCLIENT. Also provides protocols example: JSON, XML etc. EventnameFilter to identify map the received events to this input.

https://docs.onap.org/projects/onap-policy-parent/en/latest/apex/APEX-User-Manual.html#id16

APEX Policy Development guide : Policy APEX File (.apex) & Other Helper files

Steps:

1. Define Policy model file name
2. Define Schemas (use to describe event parameters, etc.)
3. Define context/album (**Optional**)
4. Define events (Inout parameters)
5. Define Tasks (Input and output parameters)
6. Define Policies
   a. Create policy name and init state
   b. Define state machine (Trigger event, state, task to be execute, next state, output event, task selection logic etc.)

```
#=========> create Policy model
model create name=DemoPolicyModel version=1.0.0
```

```
#========> Define Event or other Schema and Data types
schema create name=SimpleStringType         flavour=Java schema=java.lang.String
schema create name=SimpleLongType           flavour=Java schema=java.lang.Long
schema create name=rtrIpType flavour=Avro schema=LS
#MACROFILE:"src/main/resources/schemas/rtrIp.avsc"
LE
```

```
{
    "type": "record",
    "name": "ipInfo",
    "namespace": "org.onap.policy.apex.onap.demo",
    "fields": [
        {
            "name": "src_ip",
            "type": "string"
        },
        {
            "name": "dst_ip",
            "type": "string"
        }
    ]
}
```

```
album create name=newBandwidthLineCreateAlbum scope=policy writable=true schemaName=rtrIpType
```

```
event create name=checkBwEvent version=0.0.1 nameSpace=org.onap.policy.apex.onap.demo source=DCAE target=APEX
event parameter create name=checkBwEvent parName=interfaceName schemaName=SimpleStringType
event parameter create name=checkBwEvent parName=bandwidth schemaName=SimpleLongType
event parameter create name=checkBwEvent parName=ipInfo schemaName=rtrIpType
```

```
{
  "name": "checkBwEvent",
  "interfaceName": "example",
  "bandwidth": "83",
  "ipInfo": {
    "src_ip": "10.11.12.32",
    "dst_ip": "10.11.12.65"
  }
}
```

```
task create name=checkBandwidthTask
task inputfield create name=checkBandwidthTask fieldName=ipInfo schemaName=rtrIpType
task inputfield create name=checkBandwidthTask fieldName=bandwidth schemaName=SimpleLongType
task inputfield create name=checkBandwidthTask fieldName=interfaceName  schemaName=SimpleStringType
task outputfield create name=checkBandwidthTask fieldName=interfaceName schemaName=SimpleStringType
task outputfield create name=checkBandwidthTask fieldName=bandwidth schemaName=SimpleLongType
task outputfield create name=checkBandwidthTask fieldName=status schemaName=SimpleStringType
task outputfield create name=checkBandwidthTask fieldName=ipInfo schemaName=rtrIpType

task logic create name=checkBandwidthTask logicFlavour=JAVASCRIPT logic=LS
#MACROFILE:"src/main/resources/logic/checkBandwidthTask.js"
LE
```

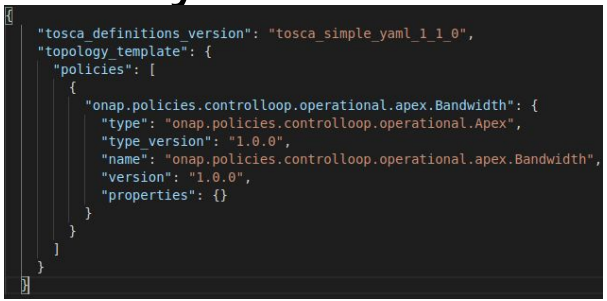
```
#========> Create Policy (2)
# check bandwidth and if it reaches threshold then create bandwidth
policy create name=checkOrCreateBandwidth template=Freestyle firstState=checkBwState


# Define state createBwState (2.3)
policy state create name=checkOrCreateBandwidth stateName=createBwState triggerName=logCreateBwEvent defaultTaskName=createBandwidthTask
policy state output create name=checkOrCreateBandwidth stateName=createBwState outputName=createSDNCOutPut eventName=logCreateBwEvent nextState=NULL
policy state taskref create name=checkOrCreateBandwidth stateName=createBwState taskLocalName=createBandwidthTask taskName=createBandwidthTask outputTyp

# Define state log information (2.2)
policy state create name=checkOrCreateBandwidth stateName=logBwState triggerName=logCreateBwEvent defaultTaskName=logBandwidthTask
policy state output create name=checkOrCreateBandwidth stateName=logBwState outputName=ReceiveIfOutPut eventName=logEvent nextState=NULL
policy state taskref create name=checkOrCreateBandwidth stateName=logBwState taskName=logBandwidthTask taskLocalName=createBandwidthTask outputType=DIRE

#create check bw state
policy state create name=checkOrCreateBandwidth stateName=checkBwState triggerName=checkBwEvent defaultTaskName=checkBandwidthTask
policy state output create name=checkOrCreateBandwidth stateName=checkBwState outputName=bandwidthCreate eventName=logCreateBwEvent nextState=createBwSt
policy state output create name=checkOrCreateBandwidth stateName=checkBwState outputName=bandwidthNotCreate eventName=logCreateBwEvent nextState=logBwSt
```

APEX Policy Development guide : Tosca Template & Final Policy File



```
{
  "tosca_definitions_version": "tosca_simple_yaml_1_1_0",
  "topology_template": {
    "policies": [
      {
        "onap.policies.controlloop.operational.apex.Bandwidth": {
          "type": "onap.policies.controlloop.operational.Apex",
          "type_version": "1.0.0",
          "name": "onap.policies.controlloop.operational.apex.Bandwidth",
          "version": "1.0.0",
          "properties": {}
        }
      }
    ]
  }
}
```

Tosca Template File

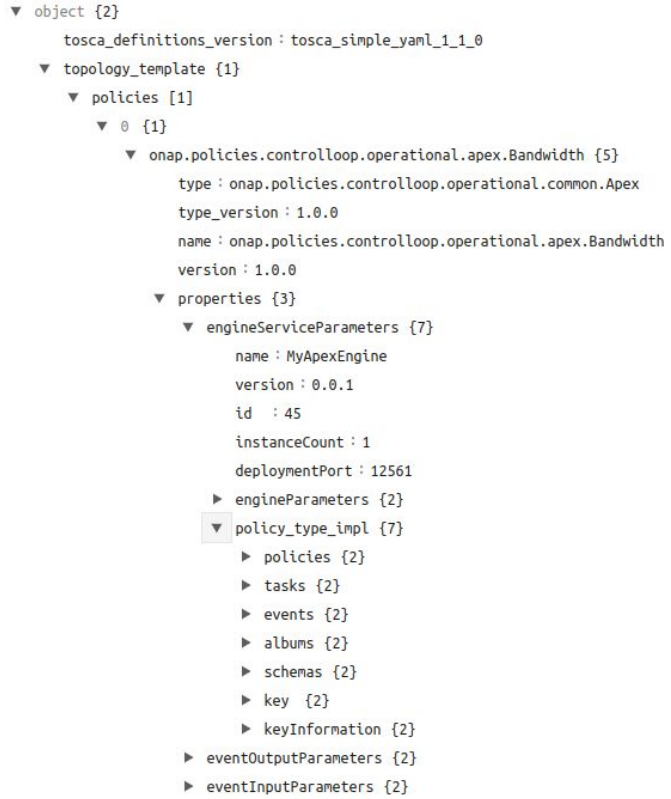
How to generate the executable policy file using CLI Tosca Editor tools:

$> $APEX_HOME/bin/apexCLIToscaEditor.sh -c $APEX_HOME/examples/PolicyModel.apex -ot $APEX_HOME/examples/test.json -l $APEX_HOME/examples/test.log -ac $APEX_HOME/examples/RESTServerStandaloneJsonEvent.json -t $APEX_HOME/examples/ToscaTemplate.json

How to test the executable policy file in local docker env.

$> $APEX_HOME/bin/apexApps.sh engine -c DemoPlicyModel.json

NOTE: If policy receives input from file make sure the input file is in right path.



```
▼ object {2}
    tosca_definitions_version : tosca_simple_yaml_1_1_0
  ▼ topology_template {1}
    ▼ policies [1]
      ▼ 0 {1}
        ▼ onap.policies.controlloop.operational.apex.Bandwidth {5}
            type : onap.policies.controlloop.operational.common.Apex
            type_version : 1.0.0
            name : onap.policies.controlloop.operational.apex.Bandwidth
            version : 1.0.0
          ▼ properties {3}
            ▼ engineServiceParameters {7}
                name : MyApexEngine
                version : 0.0.1
                id  : 45
                instanceCount : 1
                deploymentPort : 12561
              ▶ engineParameters {2}
              ▼ policy_type_impl {7}
                ▶ policies {2}
                ▶ tasks {2}
                ▶ events {2}
                ▶ albums {2}
                ▶ schemas {2}
                ▶ key  {2}
                ▶ keyInformation {2}
            ▶ eventOutputParameters {2}
            ▶ eventInputParameters {2}
```

Final executable Policy file in json viewer

APEX Policy Deployment (ONAP environment)

1. Add a policy type. If policy type exists then this steps is **optional**.

POST {{POLICY-API-URL}}/policy/api/v1/policytypes

2. Create Policy. Send generated executable policy file in payload.

POST {{POLICY-API-URL}}/policy/api/v1/policies

3. Simple deploy Policy

POST {{POLICY-PAP-URL}}/policy/pap/v1/pdps/policies

4. Query details of Policy

GET {{POLICY-API-URL}}/policy/api/v1/policies

Test Demo Policy:

5. Send test event i.e. checkBwEvent to DMaap

POST /events/unauthenticated.DCAE_CL_OUTPUT

Policy Engines Comparison

	APEX	Drools	XACML
Adaptiveness	Adaptive, Policy flow can change based on runtime context information.	Static	Static
Implementation characteristics	State machine based implementation, little complex than drools to implement.	Relatively simple to implement.	Simple to implement.
Event input	Synchronous and asynchronous Policy Execution	Asynchronous	Decision Policy (Support simple question/Answer decision)
Trigger type	Support various input and output plugins.	Support few input/output plugins.	API based.
General Usage	Operational	Operational	Configuration

For control loop if the scenario is complex one can use APEX, else can use drools.

Thank you

Resources

APEX tutorial:

https://wiki.onap.org/download/attachments/84651833/Apex_Tutorial.mkv?version=1&modificationDate=1591376955000&api=v2