Policy API

DESCRIPTION

The Policy subsystem of ONAP maintains, distributes, and operates on the set of rules that underlie ONAP's control, orchestration, and management functions. Policy provides a centralized environment for the creation and management of easily-updatable conditional rules. It enables users to validate policies and rules, identify and resolve overlaps and conflicts, and derive additional policies where needed. The following operations are supported by the policy API:

- Create policies on the PAP
- Update policies on the PAP
- Delete policies on the PAP or PDP
- Push policies from the PAP to the PDP
- List policies on the PDP
- Get config data of policies on the PDP
- Create Dictionary Items
- Update Dictionary Items
- Retrieve Dictionary Items
- Import Micro Services Models
- Retrieve Metrics for policy counts from PDP and PAP

POLICY ENGINE SERVICES

PUT	/createConfig	DEPRECATED	Creates a Config Policy based on given Policy Parameters.
PUT	/createDictionaryItem		Creates a Dictionary Item for a specific dictionary based on given Parameters.
PUT	/createFirewallConfig	DEPRECATED	Creates a Config Firewall Policy
PUT	/createPolicy		Creates a Policy based on given Policy Parameters.
DELETE	/deletePolicy		Deletes the specified policy from the PDP Group or PAP.
POST	/getConfig		Gets the configuration from the PolicyDecisionPoint(PDP)
POST	/getConfigByPolicyName	DEPRECATED	Gets the configuration from the PolicyDecisionPoint(PDP) using PolicyName
POST	/getDecision		Gets the Decision using specified decision parameters
POST	/getDictionaryItems		Gets the dictionary items from the PAP
GET	/getMetrics		Gets the policy metrics from the PolicyAccessPoint(PAP)
POST	/getNotification		Registers DMaaP Topic to recieve notification from Policy Engine
POST	/listConfig		Gets the list of configuration policies from the PDP
POST	/policyEngineImport		Imports Policy based on the parameters which represent the service used to create a policy Service.
PUT	/pushPolicy		Pushes the specified policy to the PDP Group.
POST	/sendEvent		Sends the Events specified to the Policy Engine
POST	/sendHeartbeat		Sends Heartbeat to DMaaP Topic Registry to continue recieving notifications from Policy Engine
POST	/stopNotification		De-Registers DMaaP Topic to stop recieving notifications from Policy Engine
PUT	/updateConfig	DEPRECATED	Updates a Config Policy based on given Policy Parameters.
PUT	/updateDictionaryItem		Updates a Dictionary Item for a specific dictionary based on given Parameters.
PUT	/updateFirewallConfig	DEPRECATED	Updates a Config Firewall Policy
PUT	/updatePolicy		Updates a Policy based on given Policy Parameters.

SAMPLE JAVA CLIENT CODE

Get Config Example

```
/*-
 * ============LICENSE_START=======================================================
 * PolicyEngineClient
 * ================================================================================
 * Copyright (C) 2017 AT&T Intellectual Property. All rights reserved.
 * ================================================================================
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 * ============LICENSE_END=========================================================
 */

package org.onap.policyEngine;

import java.util.Collection;

import org.onap.policy.api.ConfigRequestParameters;
import org.onap.policy.api.PolicyConfig;
import org.onap.policy.api.PolicyEngine;

public class GetConfigSample {

    public static void main(String[] args) throws Exception {
        PolicyEngine pe = new PolicyEngine("config.properties");
        ConfigRequestParameters configRequestParams = new ConfigRequestParameters();
        configRequestParams.setPolicyName(".*");
        Collection<PolicyConfig> configs = pe.getConfig(configRequestParams);
        for (PolicyConfig config: configs){
            System.out.println(config.getPolicyConfigMessage());
            System.out.println(config.getPolicyConfigStatus());
        }
    }
}
```

Create Config FIrewall Policy Example

```
/*-
 * ============LICENSE_START=======================================================
 * PolicyEngineClient
 * ================================================================================
 * Copyright (C) 2017 AT&T Intellectual Property. All rights reserved.
 * ================================================================================
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 * ============LICENSE_END=========================================================
 */

package org.onap.policyEngine;
```

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.StringReader;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.UUID;

import javax.json.Json;
import javax.json.JsonObject;
import javax.json.JsonReader;

import org.onap.policy.api.PolicyChangeResponse;
import org.onap.policy.api.PolicyConfigType;
import org.onap.policy.api.PolicyEngine;
import org.onap.policy.api.PolicyParameters;
import org.onap.policy.api.PolicyType;

public class ConfigFirewallPolicyClient {
    static Boolean isEdit = false;
    public static void main(String[] args) {
        try{
            PolicyEngine policyEngine = new PolicyEngine("config.properties");
            PolicyParameters policyParameters = new PolicyParameters();
            // Set Policy Type
            policyParameters.setPolicyConfigType(PolicyConfigType.Firewall); //required
            policyParameters.setPolicyName("MikeAPItesting.testConfigFirewallPolicy1607_1"); //required
            //policyParameters.setPolicyScope("MikeAPItesting");
 //Directory will be created where the Policies are saved... this
displays a a subscope on the GUI
            policyParameters.setRequestID(UUID.randomUUID());

            // Set Safe Policy value for Risk Type
            SimpleDateFormat dateformat3 = new SimpleDateFormat("dd/MM/yyyy");
            Date date = dateformat3.parse("15/10/2016");
            policyParameters.setTtlDate(date);
            // Set Safe Policy value for Guard
            policyParameters.setGuard(true);
            // Set Safe Policy value for Risk Level
            policyParameters.setRiskLevel("5");
            // Set Safe Policy value for Risk Type
            policyParameters.setRiskType("PROD");
            File jsonFile = null;
            String jsonRuleList = null;
            Path file = Paths.get("C:\\policyAPI\\firewallRulesJSON\\Config_FW_Sample.json");
            jsonFile = file.toFile();

            //buildJSON(jsonFile, jsonRuleList);
            policyParameters.setConfigBody(buildJSON(jsonFile, jsonRuleList).toString());
            policyParameters.setConfigBodyType(PolicyType.JSON);
            // API method to create Policy or update policy
            PolicyChangeResponse response = null;
            if (!isEdit) {
                response = policyEngine.createPolicy(policyParameters);
            } else {
                response = policyEngine.updatePolicy(policyParameters);
            }

            if(response.getResponseCode()==200){
                System.out.println(response.getResponseMessage());
                System.out.println("Policy Created Successfully!");
            }else{
                System.out.println("Error! " + response.getResponseMessage());
            }
        } catch (Exception e) {
            System.err.println(e.getMessage());
```

```
        }

}

    private static JsonObject buildJSON(File jsonInput, String jsonString) throws FileNotFoundException {
        JsonObject json = null;
        JsonReader jsonReader = null;
        if (jsonString != null && jsonInput == null) {
            StringReader in = null;
            in = new StringReader(jsonString);
            jsonReader = Json.createReader(in);
            json = jsonReader.readObject();
            in.close();
        } else {
            InputStream in = null;
            in = new FileInputStream(jsonInput);
            jsonReader = Json.createReader(in);
            json = jsonReader.readObject();
            try {
                in.close();
            } catch (IOException e) {
                System.err.println("Exception Occured while closing input stream"+e);
            }
        }
        jsonReader.close();
        return json;
    }

}
```

Sample JSON file - Config_FW_Sample.json


```
{
      "serviceTypeId": "/v0/firewall/pan",
      "configName": "AFTTFwPolicy1Config",
      "deploymentOption": {
            "deployNow": false
      },
      "securityZoneId": "cloudsite:dev1a",
      "serviceGroups": [{
            "name": "SSH",
            "description": "Ssh service entry in service list",
            "type": "SERVICE",
            "transportProtocol": "tcp",
            "appProtocol": null,
            "ports": "22"
      }],
      "addressGroups": [{
            "name": "CiscoVCE",
            "description": "Destination CiscoCVE",
            "members": [{
                  "type": "SUBNET",
                  "value": "12.63.31.61/12"
            }]
      }, {
            "name": "HOHOServers",
            "description": "Source HOHOServers for first testing",
            "members": [{
                  "type": "SUBNET",
                  "value": "12.60.32.11/23"
            }]
      }],
      "firewallRuleList": [{
            "position": "1",
            "ruleName": "FWRuleHOHOServerToCiscoVCE",
            "fromZones": ["UntrustedZoneCiscoCVEName"],
            "toZones": ["TrustedZoneHOHOName"],
            "negateSource": false,
            "negateDestination": false,
            "sourceList": [{
                  "type": "REFERENCE",
                  "name": "HOHOServers"
            }],
            "destinationList": [{
                  "type": "REFERENCE",
                  "name": "CiscoVCE"
            }],
            "sourceServices": [],
            "destServices": [{
                  "type": "REFERENCE",
                  "name": "SSH"
            }],
            "action": "accept",
            "description": "FW rule for HOHO source to CiscoVCE destination",
            "enabled": true,
            "log": true
      }]
}
```

Delete Policy Example

```
/*-
 * ============LICENSE_START=======================================================
 * PolicyEngineClient
 * ================================================================================
 * Copyright (C) 2017 AT&T Intellectual Property. All rights reserved.
 * ================================================================================
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 * ============LICENSE_END=========================================================
 */

package org.onap.policyEngine;

import org.onap.policy.api.DeletePolicyCondition;
import org.onap.policy.api.DeletePolicyParameters;
import org.onap.policy.api.PolicyChangeResponse;
import org.onap.policy.api.PolicyEngine;

public class DeletePolicyClient {

    public static void main(String[] args) {
        try {

            PolicyEngine policyEngine = new PolicyEngine("config.properties");
            DeletePolicyParameters policyParameters = new DeletePolicyParameters();

            //Parameter arguments
            policyParameters.setPolicyName("MikeConsole.Config_testDeleteAPI6.1.xml");
            policyParameters.setPolicyComponent("PDP");
            policyParameters.setPdpGroup("default");
            policyParameters.setDeleteCondition(DeletePolicyCondition.ALL);
            policyParameters.setRequestID(null);

            // API method to Push Policy to PDP
            PolicyChangeResponse response = null;
            response = policyEngine.deletePolicy(policyParameters);

            if(response.getResponseCode()==200){
                System.out.println(response.getResponseMessage());
                System.out.println("Policy Deleted Successfully!");
            }else{
                System.out.println("Error! " + response.getResponseMessage());
            }

        } catch (Exception e) {
            System.err.println(e.getMessage());

        }

    }


}
```

Push Policy Example

```
/*-
 * ============LICENSE_START=======================================================
 * PolicyEngineClient
 * ================================================================================
 * Copyright (C) 2017 AT&T Intellectual Property. All rights reserved.
 * ================================================================================
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 * ============LICENSE_END=========================================================
 */

package org.onap.policyEngine;

import org.onap.policy.api.PolicyChangeResponse;
import org.onap.policy.api.PolicyEngine;
import org.onap.policy.api.PushPolicyParameters;

public class PushPoliciesToPDP {
    public static void main(String[] args) {
        try {

            PolicyEngine policyEngine = new PolicyEngine("config.properties");
            PushPolicyParameters policyParameters = new PushPolicyParameters();

            //Parameter arguments
            policyParameters.setPolicyName("Mike.testCase1");
            policyParameters.setPolicyType("Base");
            //policyParameters.setPolicyScope("MikeAPItesting");
            policyParameters.setPdpGroup("default");
            policyParameters.setRequestID(null);

            // API method to Push Policy to PDP
            PolicyChangeResponse response = null;
            response = policyEngine.pushPolicy(policyParameters);

            if(response.getResponseCode()==204){
                System.out.println(response.getResponseMessage());
                System.out.println("Policy Pushed Successfully!");
            }else{
                System.out.println("Error! " + response.getResponseMessage());
            }

        } catch (Exception e) {
            System.err.println(e.getMessage());

        }

    }

}
```

Decision Policy Example

```
/*-
 * ============LICENSE_START=======================================================
 * PolicyEngineClient
```

```
 * ==============================================================================
 * Copyright (C) 2017 AT&T Intellectual Property. All rights reserved.
 * ==============================================================================
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 * ============LICENSE_END=========================================================
 */

package org.onap.policyEngine;

import java.util.Arrays;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.UUID;

import org.onap.policy.api.AttributeType;
import org.onap.policy.api.PolicyChangeResponse;
import org.onap.policy.api.PolicyClass;
import org.onap.policy.api.PolicyEngine;
import org.onap.policy.api.PolicyParameters;

public class DecisionPolicyClient {
    static Boolean isEdit = true;
    public static void main(String[] args) {
        try {
            PolicyEngine policyEngine = new PolicyEngine("config.properties");
            PolicyParameters policyParameters = new PolicyParameters();
            // Set Policy Type
            policyParameters.setPolicyClass(PolicyClass.Decision); //required
            policyParameters.setPolicyName("MikeAPItests.testDecisionAPI"); //required
            policyParameters.setOnapName("java"); //required
            policyParameters.setPolicyDescription("This is a sample Decision policy UPDATE example with
Settings");  //optional
            //policyParameters.setPolicyScope("MikeAPItests");
 //Directory will be created where the Policies are saved... this
displays a a subscope on the GUI

            //Set the Component Attributes... These are Optional
            Map<String, String> configAttributes = new HashMap<>();
            configAttributes.put("Template", "UpdateTemplate");
            configAttributes.put("controller", "default");
            configAttributes.put("SamPoll", "30");
            configAttributes.put("value", "abcd");

            Map<AttributeType, Map<String,String>> attributes = new HashMap<>();
            attributes.put(AttributeType.MATCHING, configAttributes);

            //Set the settings... These are Optional
            Map<String, String> settingsMap = new HashMap<>();
            settingsMap.put("server", "5");

            attributes.put(AttributeType.SETTINGS, settingsMap);
            policyParameters.setAttributes(attributes);


            List<String> dynamicRuleAlgorithmLabels = new LinkedList<>();
            List<String> dynamicRuleAlgorithmFunctions = new LinkedList<>();
            List<String> dynamicRuleAlgorithmField1 = new LinkedList<>();
            List<String> dynamicRuleAlgorithmField2 = new LinkedList<>();
```

```
            //Example of a complex Rule algorithm using the settings in the Field1
            /* label    field1      function                field2
             * ****************************************************
             * A1       S_server    integer-equal               90
             * A2       cap         string-contains             ca
             * A3       cobal       integer-equal               90
             * A4       A2          and                         A3
             * A5       Config      integer-greater-than        45
             * A6       A4  `       or                          A5
             * A7       A1          and                         A6
             */
            dynamicRuleAlgorithmLabels = Arrays.asList("A1","A2","A3","A4","A5","A6","A7");
            dynamicRuleAlgorithmField1 = Arrays.asList("S_server","cap","cobal","A2","Config","A4","A1");
            dynamicRuleAlgorithmFunctions = Arrays.asList("integer-equal","string-contains","integer-equal","
and","integer-greater-than","or","and");
            dynamicRuleAlgorithmField2 = Arrays.asList("90","ca","90","A3","45","A5","A6");

            policyParameters.setDynamicRuleAlgorithmLabels(dynamicRuleAlgorithmLabels);
            policyParameters.setDynamicRuleAlgorithmField1(dynamicRuleAlgorithmField1);
            policyParameters.setDynamicRuleAlgorithmFunctions(dynamicRuleAlgorithmFunctions);
            policyParameters.setDynamicRuleAlgorithmField2(dynamicRuleAlgorithmField2);

            policyParameters.setRequestID(UUID.randomUUID());

            // API method to create Policy or update policy
            PolicyChangeResponse response = null;
            if (!isEdit) {
                response = policyEngine.createPolicy(policyParameters);
            } else {
                response = policyEngine.updatePolicy(policyParameters);
            }

            if(response.getResponseCode()==200){
                System.out.println(response.getResponseMessage());
                System.out.println("Policy Created Successfully!");
            }else{
                System.out.println("Error! " + response.getResponseMessage());
            }
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }

}
```

List Config Policy Example

```
/*-
 * ============LICENSE_START=======================================================
 * PolicyEngineClient
 * ================================================================================
 * Copyright (C) 2017 AT&T Intellectual Property. All rights reserved.
 * ================================================================================
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 * ============LICENSE_END=========================================================
 */

package org.onap.policyEngine;
```

```java
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;
import java.util.UUID;

import org.onap.policy.api.ConfigRequestParameters;
import org.onap.policy.api.PolicyConfigException;
import org.onap.policy.api.PolicyEngine;
import org.onap.policy.api.PolicyEngineException;
import org.onap.policy.common.logging.flexlogger.FlexLogger;
import org.onap.policy.common.logging.flexlogger.Logger;

public class ListConfigPoliciesClient {

    private static final Logger LOGGER  = FlexLogger.getLogger(ListConfigPoliciesClient.class);

    public static void main(String[] args) {
        PolicyEngine policyEngine;

        // List Config Policies Example
        try {
            policyEngine = new PolicyEngine("config.properties");
            ConfigRequestParameters parameters = new ConfigRequestParameters();

            parameters.setPolicyName(".*");
            parameters.setOnapName(".*");
            parameters.setConfigName(".*");

            Map<String, String> configAttributes = new HashMap<>();
            configAttributes.put("java", "java");
            configAttributes.put("peach", "Tar");
            configAttributes.put("true", "false");
            configAttributes.put("small", "testPass");
            parameters.setConfigAttributes(configAttributes);

            parameters.setRequestID(UUID.randomUUID());

            Collection<String> response = policyEngine.listConfig(parameters);
            if(response!=null && !response.contains("PE300")){
                for(String configList : response){
                    System.out.println(configList.toString()+"\n");
                }
            }else{
                System.out.println("Error! " +response);
            }

        } catch (PolicyConfigException e) {
            LOGGER.error("Exception Occured"+e);
        } catch (PolicyEngineException e) {
            LOGGER.error("Exception Occured"+e);
        }
    }

}
```

JSON EXAMPLES

Create Microservice Policy

```
API: createPolicy
OPERATION: PUT
REQUEST BODY:
{
    "configBody": "{
                        \"service\":\"ControllerServiceSampleSdnlServiceInstance\",
                        \"location\":\"Edge\",
                        \"uuid\":\"TestUUID\",
                        \"policyName\":\"testRestCreateMicroServicesNewParams\",
                        \"description\":\"testing Create\",
                        \"configName\":\"TestName\",
                        \"templateVersion\":\"1604\",
                        \"priority\":\"4\",
                        \"version\":\"0.1.0-SNAPSHOT\",
                        \"policyScope\":\"resource=F5,service=vSCP,type=configuration,
closedLoopControlName=vSCP_F5_Firewall_d925ed73-8231-4d02-9545-db4e101f88f8\",
                        \"content\":{
                                        \"taskOrchestratedConfiguration\":\"test\",
                                        \"taskCustomConfiguration\":\"set\",
                                        \"configuration\":\"test\",
                                        \"cdapUrl\":\"testurl\",
                                        \"taskName\":\"test\",
                                        \"taskNameTEST\":\"TEST\",
                                        \"users\":\"[tuser]\",
                                        \"adminUsers\":\"[lji]\",
                                        \"taskConfigFilePath\":\"test\",
                                        \"managerPortNumber\":\"999\",
                                        \"taskType\":\"test\",
                                        \"taskCommandFilePath\":\"tset\",
                                        \"contailIp\":\"test\",
                                        \"consoleUsers\":\"[odu-e2e]\",
                                        \"taskStatusFilePath\":\"test\"
                        }
        }",
    "policyConfigType": "MicroService",
    "policyName": "MikeAPITesting.testRestCreateMicroServicesNewParams",
    "ecompName": "SDNC"
}
```

Update Microservice Policy

```
API: updatePolicy
OPERATION: PUT
REQUEST BODY:
{
    "configBody": "{
                        \"service\":\"ControllerServiceSampleSdnlServiceInstance\",
                        \"location\":\"Edge\",
                        \"uuid\":\"TestUUID\",
                        \"policyName\":\"testRestCreateMicroServicesNewParams\",
                        \"description\":\"testing Update\",
                        \"configName\":\"TestName\",
                        \"templateVersion\":\"1604\",
                        \"priority\":\"4\",
                        \"version\":\"0.1.0-SNAPSHOT\",
                        \"policyScope\":\"resource=F5,service=vSCP,type=configuration,
closedLoopControlName=vSCP_F5_Firewall_d925ed73-8231-4d02-9545-db4e101f88f8\",
                        \"content\":{
                                        \"taskOrchestratedConfiguration\":\"test\",
                                        \"taskCustomConfiguration\":\"set\",
                                        \"configuration\":\"test\",
                                        \"cdapUrl\":\"testurl\",
                                        \"taskName\":\"test\",
                                        \"taskNameTEST\":\"TEST\",
                                        \"users\":\"[tuser]\",
                                        \"adminUsers\":\"[lji]\",
                                        \"taskConfigFilePath\":\"test\",
                                        \"managerPortNumber\":\"999\",
                                        \"taskType\":\"test\",
                                        \"taskCommandFilePath\":\"tset\",
                                        \"contailIp\":\"test\",
                                        \"consoleUsers\":\"[odu-e2e]\",
                                        \"taskStatusFilePath\":\"test\"
                        }
        }",
    "policyConfigType": "MicroService",
    "policyName": "MikeAPITesting.testRestUpdateMicroServicesNewParams",
    "ecompName": "SDNC"
}
```

CURL EXAMPLES

Push Policy

```
echo "pushPolicy : PUT : com.vLoadBalancer"
echo "pushPolicy : PUT : com.vLoadBalancer"
curl -v --silent -X PUT --header 'Content-Type: application/json' --header 'Accept: text/plain' --header
'ClientAuth: XYZ' --header 'Authorization: Basic XYZ' --header 'Environment: TEST' -d '{
  "pdpGroup": "default",
  "policyName": "com.vLoadBalancer",
  "policyType": "MicroService"
}' 'http://pdp:8081/pdp/api/pushPolicy'
```

Delete Policy

```
echo "deletePolicy : DELETE : com.vFirewall"
curl -v --silent -X DELETE --header 'Content-Type: application/json' --header 'Accept: text/plain' --header
'ClientAuth: XYZ' --header 'Authorization: Basic XYZ' --header 'Environment: TEST' -d '{
  "pdpGroup": "default",
  "policyComponent": "PDP",
  "policyName": "com.vFirewall",
  "policyType": "MicroService"
}' 'http://pdp:8081/pdp/api/deletePolicy'
```

Get Config

```
echo "Get all Config Policy example"
curl -i -v -H 'Content-Type: application/json' -H 'Accept: application/json' -H 'ClientAuth: XYZ' -H
'Authorization: Basic XYZ -H 'Environment: TEST' -X POST -d '{
"policyName": ".*"
}' http://${PDP_IP}:8081/pdp/api/getConfig
```

ADDITIONAL EXAMPLES

Deleting a Policy from PAP

```
// Deleting from PAP will remove the policy from the PolicyEntity & PolicyVersion tables (UI-Editor tab).
// This means that the policy is no longer be available in Policy System.

// PayLoad:
{    "policyName": "com.testpolicy",    //scope.policyName
     "policyType": "Base",              //policy type
     "policyComponent": "PAP",          //component name
     "deleteCondition": "ALL"           //versions (ALL or CURRENT)
}
```

Deleting a Policy from PDP

```
// Deleting from PDP will delete the policy from the PDP Group.  The policy is still available in Policy
System.
// When the policy is needed again, the policy should be pushed to the PDP.

// PayLoad:
{    "policyName": "com.testpolicy",   //scope.policyName
     "policyType": "Base",             //policy type
     "policyComponent": "PDP",         //component name
     "pdpGroup": "Default"             //group name
}
```

POLICY ENGINE API DETAILS

Header parameters apply to each API.

Header Parameter	Required / Optional	Data Type	Description
ClientAuth	Required	string	Encoded client authentication details
X-ECOMP-RequestID	Optional	string	Request ID used to track the requests
Authorization	Required	string	Authentication details in order to be able to connect to the servers
Environment	Required	string	Execution environment

Service	Request Parameter	Data Type	Request Sample	HTTP Response Codes	Response	Response Sample

put /createDictionaryItem	dictionaryParameters	DictionaryParameters {
		dictionary (string, optional),
		dictionaryJson (string, optional),
		dictionaryType (string, optional) = ['Common', 'Action', 'ClosedLoop', 'Firewall', 'Decision', 'BRMS', 'MicroService', 'DescriptiveScope', 'PolicyScope', 'Enforcer', 'SafePolicy', 'Extended'],
		requestID (string, optional)
		}
			{
			"dictionary": "string",
			"dictionaryJson": "string",
			"dictionaryType": "Common",
			"requestID": "string"
			}
				200 Successful
				400 Invalid Request
				401 Unauthorized
				500 Error		

put /createPolicy	policyParameters	PolicyParameters {
		actionAttribute (string, optional),
		actionPerformer (string, optional),
		attributes (object, optional),
		configBody (string, optional),
		configBodyType (string, optional) = ['Properties', 'json', 'xml', 'other'],
		configName (string, optional),
		controllerName (string, optional),
		dependencyNames (Array[string], optional),
		dynamicRuleAlgorithmField1 (Array[string], optional),
		dynamicRuleAlgorithmField2 (Array[string], optional),
		dynamicRuleAlgorithmFunctions (Array[string], optional),
		dynamicRuleAlgorithmLabels (Array[string], optional),
		ecompName (string, optional),
		extendedOption (string, optional),
		guard (boolean, optional),
		policyClass (string, optional) = ['Config', 'Action', 'Decision'],
		policyConfigType (string, optional) = ['Base', 'ClosedLoop_Fault', 'ClosedLoop_PM', 'Firewall', 'BRMS_Raw', 'BRMS_Param', 'MicroService', 'EXTENDED'],
		policyDescription (string, optional),
		policyName (string, optional),
		priority (string, optional),
		requestID (string, optional),
		riskLevel (string, optional),
		riskType (string, optional),
		ruleProvider (string, optional) = ['Custom', 'AAF', 'GUARD_YAML', 'GUARD_BL_YAML'],
		ttlDate (string, optional)
		}
			{
			"attributes": {"MATCHING":{"key":"value"}},
			"configBody": "test body",
			"configBodyType": "OTHER",
			"configName": "testConfig",
			"ecompName": "DCAE",
			"policyClass": "Config",
			"policyConfigType": "Base",
			"policyDescription": "Testing through REST API",
			"policyName": "scope.testPolicyName",
			"requestID": "6be5a05a-fb62-11e5-86aa-5e5517507c66"
			}
				200 Successful
				400 Invalid Request
				401 Unauthorized
				500 Error		

delete /deletePolicy	deletePolicyParameters	DeletePolicyParameters {
		deleteCondition (string, optional) = ['Current Version', 'All Versions'],
		pdpGroup (string, optional),
		policyComponent (string, optional),
		policyName (string, optional),
		policyType (string, optional),
		requestID (string, optional)
		}
			{
			"deleteCondition": "Current Version",
			"pdpGroup": "string",
			"policyComponent": "string",
			"policyName": "string",
			"policyType": "string",
			"requestID": "string"
			}
				200 Successful
				202 Locked Down
				400 Invalid Request
				403 Forbidden
				404 Not found
				500 Error		

post /getConfig	configRequestParameters	ConfigRequestParameters {
		configAttributes (object, optional),
		configName (string, optional),
		ecompName (string, optional),
		policyName (string, optional),
		requestID (string, optional),
		unique (boolean, optional)
		}
			{
			"configAttributes": {"key":"value"},
			"configName": "sample",
			"ecompname": "DCAE",
			"policyName": ".*",
			"requestID": "a7c6b20c-fb5e-11e5-86aa-5e5517507c66",
			"unique": false
			}
				200 Successful
				400 Invalid Request
				401 Unauthorized
				500 Error		

post /getDecision	decisionRequestParameters	DecisionRequestParameters {
		decisionAttributes (object, optional),
		ecompcomponentName (string, optional),
		requestID (string, optional)
		}
			{
			"decisionAttributes": {},
			"ecompcomponentName": "string",
			"requestID": "string"
			}
				200 Successful
				400 Invalid Request
				401 Unauthorized
				500 Error
					DecisionResponse {
					decision (string, optional) = ['permit', 'deny', 'error']
					string
					Enum: "permit", "deny", "error"
					,
					details (string, optional)
					}
						{
						"decision": "permit",
						"details": "string"
						}

post /getDictionaryItems	dictionaryParameters	DictionaryParameters {
		dictionary (string, optional),
		dictionaryJson (string, optional),
		dictionaryType (string, optional) = ['Common', 'Action', 'ClosedLoop', 'Firewall', 'Decision', 'BRMS', 'MicroService', 'DescriptiveScope', 'PolicyScope', 'Enforcer', 'SafePolicy', 'Extended'],
		requestID (string, optional)
		}
			{
			"dictionary": "string",
			"dictionaryJson": "string",
			"dictionaryType": "Common",
			"requestID": "string"
			}
				200 Successful
				400 Invalid Request
				401 Unauthorized
				500 Error
					DictionaryResponse {
					dictionaryData (object, optional),
					dictionaryJson (object, optional),
					responseCode (integer, optional),
					responseMessage (string, optional)
					}
						{
						"dictionaryData": {},
						"dictionaryJson": {},
						"responseCode": 0,
						"responseMessage": "string"
						}

get /getMetrics				200 Successful
				400 Invalid Request
				401 Unauthorized
				500 Error
					MetricsResponse {
					metricsTotal (integer, optional),
					papMetrics (integer, optional),
					pdpMetrics (integer, optional),
					responseCode (integer, optional),
					responseMessage (string, optional)
					}
						{
						"metricsTotal": 0,
						"papMetrics": 0,
						"pdpMetrics": 0,
						"responseCode": 0,
						"responseMessage": "string"
						}

post /getNotification	notificationTopic	string, required		200 Successful 400 Invalid Request 401 Unauthorized 500 Error		

post /listConfig	configRequestParameters	ConfigRequestParameters {
		configAttributes (object, optional),
		configName (string, optional),
		ecompName (string, optional),
		policyName (string, optional),
		requestID (string, optional),
		unique (boolean, optional)
		}
			{
			"configAttributes": {},
			"configName": "string",
			"ecompName": "string",
			"policyName": "string",
			"requestID": "string",
			"unique": true
			}
				200 Successful
				400 Invalid Request
				401 Unauthorized
				500 Error		

post /policyEngineImport	importParametersJson	string, required (parameter Type: query)		200 Successful 400 Invalid Request 401 Unauthorized 500 Error		
	file	file, required (parameter Type: formData)				

put /pushPolicy	pushPolicyParameters	PushPolicyParameters {
		pdpGroup (string, optional),
		policyName (string, optional),
		policyType (string, optional),
		requestID (string, optional)
		}
			{
			"pdpGroup": "default",
			"policyName": "scope.testPolicyName",
			"policyType": "Base"
			}
				200 Successful
				400 Invalid Request
				401 Unauthorized
				500 Error		

		EventRequestParameters {	{	200 Successful		

post /sendEvent	eventRequestParameters	eventAttributes (object, optional), requestID (string, optional) }	"eventAttributes": { "cpu": "97", "memory": "65" }, "requestID": "463247b2-fb5f-11e5-86aa-5e5517507c66" }	400 Invalid Request 401 Unauthorized 500 Error		
post /sendHeartbeat	notificationTopic	string, required		200 Successful 400 Invalid Request 401 Unauthorized 500 Error		
post /stopNotification	notificationTopic	string, required		200 Successful 400 Invalid Request 401 Unauthorized 500 Error		

put /updateDictionaryItem	dictionaryParameters	DictionaryParameters {
		dictionary (string, optional),
		dictionaryJson (string, optional),
		dictionaryType (string, optional) = {'Common', 'Action', 'ClosedLoop', 'Firewall', 'Decision', 'BRMS', 'MicroService', 'DescriptiveScope', 'PolicyScope', 'Enforcer', 'SafePolicy', 'Extended'},
		requestID (string, optional)
		}
			{
			"dictionary": "string",
			"dictionaryJson": "string",
			"dictionaryType": "Common",
			"requestID": "string"
			}
				200 Successful
				400 Invalid Request
				401 Unauthorized
				500 Error		

put /updatePolicy	policyParameters	same as createPolicy	{
			"attributes": {"MATCHING":{"key":"value"}},
			"configBody": "test edit Body",
			"configBodyType": "OTHER",
			"configName": "testConfig",
			"ecompName": "DCAE",
			"policyClass": "Config",
			"policyConfigType": "Base",
			"policyDescription": "Testing through REST API",
			"policyName": "scope.testPolicyName",
			"requestID": "6be5a05a-fb62-11e5-86aa-5e5517507c66"
			
			}
				200 Successful
				400 Invalid Request
				401 Unauthorized
				500 Error		