Policy function as Sidecar

The policy-sync sidecar is a simple python utility that abstracts the ONAP policy interface. It is designed to function well as a Kubernetes sidecar container that is injected into a pod.

Overview

Policy Sync utility is a python based utility that interfaces with the ONAP Policy interface. It is designed to keep a local listing of policies in sync with an environment's policy distribution point (PDP). It functions well as a Kubernetes sidecar container which can pull down the latest policies for consumption by an application container.

The sync utility works with both the new policy interface introduced in Dublin and the legacy interface.

Interface with Policy

The policy interface was redesigned as of the Dublin ONAP release. The policy sync utility can work with both the newer version as well as the older v0 API. By default, it will attempt to speak the newer version of the API.

ONAP Policy Interface (as of Dublin Release).

DMaaP Notifications

The ONAP Policy interface notifies clients of an update over a DMaaP Interface. We will subscribe/poll this interface and trigger an update when an impacted ID changes

response

```
{
            "deployed-policies": [
                {
                    "policy-type": "onap.policies.monitoring.tcagen2",
                    "policy-type-version": "1.0.0",
                    "policy-id": "onap.scaleout.tca",
                    "policy-version": "2.0.0",
                    "success-count": 3,
                    "failure-count": 0
                }
            ],
            "undeployed-policies": [
                {
                    "policy-type": "onap.policies.monitoring.tcagen2",
                    "policy-type-version": "1.0.0",
                    "policy-id": "onap.firewall.tca",
                    "policy-version": "6.0.0",
                    "success-count": 3,
                    "failure-count": 0
                }
            ]
        }
}
```

REST API

We will use the decision API.

response

```
# POST TO v1/decision
{
  "ONAPName": "DCAE",
  "ONAPComponent": "PolicyHandler",
  "ONAPInstance": "622431a4-9dea-4eae-b443-3b2164639c64",
  "action": "configure",
  "resource": {
    "policy-id": [
      "onap.scaleout.tca",
      "onap.restart.tca"
    ]
  }
}
```

response

```
# Gives back a list of policies
{
        "policies": {
            "onap.scaleout.tca": {
                "type": "onap.policies.monitoring.cdap.tca.hi.lo.app",
                "version": "1.0.0",
                "metadata": {"policy-id": "onap.scaleout.tca"},
                "properties": {
                    "tca_policy": {
                        "domain": "measurementsForVfScaling",
```

```
                        "metricsPerEventName": [
                            {
                                "eventName": "vLoadBalancer",
                                "controlLoopSchemaType": "VNF",
                                "policyScope": "type=configuration",
                                "policyName": "onap.scaleout.tca",
                                "policyVersion": "v0.0.1",
                                "thresholds": [
                                    {
                                        "closedLoopControlName": "ControlLoop-vDNS-6f37f56d-a87d-4b85-b6a9-
cc953cf779b3",
                                        "closedLoopEventStatus": "ONSET",
                                        "version": "1.0.2",
                                        "fieldPath": "$.event.measurementsForVfScalingFields.
vNicPerformanceArray[*].receivedBroadcastPacketsAccumulated",
                                        "thresholdValue": 500,
                                        "direction": "LESS_OR_EQUAL",
                                        "severity": "MAJOR",
                                    },
                                    {
                                        "closedLoopControlName": "ControlLoop-vDNS-6f37f56d-a87d-4b85-b6a9-
cc953cf779b3",
                                        "closedLoopEventStatus": "ONSET",
                                        "version": "1.0.2",
                                        "fieldPath": "$.event.measurementsForVfScalingFields.
vNicPerformanceArray[*].receivedBroadcastPacketsAccumulated",
                                        "thresholdValue": 5000,
                                        "direction": "GREATER_OR_EQUAL",
                                        "severity": "CRITICAL",
                                    },
                                ],
                            }
                        ],
                    }
                },
            },
            "onap.restart.tca": {
                "type": "onap.policies.monitoring.cdap.tca.hi.lo.app",
                "version": "1.0.0",
                "metadata": {"policy-id": "onap.restart.tca", "policy-version": 1},
                "properties": {
                    "tca_policy": {
                        "domain": "measurementsForVfScaling",
                        "metricsPerEventName": [
                            {
                                "eventName": "Measurement_vGMUX",
                                "controlLoopSchemaType": "VNF",
                                "policyScope": "DCAE",
                                "policyName": "DCAE.Config_tca-hi-lo",
                                "policyVersion": "v0.0.1",
                                "thresholds": [
                                    {
                                        "closedLoopControlName": "ControlLoop-vCPE-48f0c2c3-a172-4192-9ae3-
052274181b6e",
                                        "version": "1.0.2",
                                        "fieldPath": "$.event.measurementsForVfScalingFields.
additionalMeasurements[*].arrayOfFields[0].value",
                                        "thresholdValue": 0,
                                        "direction": "EQUAL",
                                        "severity": "MAJOR",
                                        "closedLoopEventStatus": "ABATED",
                                    },
                                    {
                                        "closedLoopControlName": "ControlLoop-vCPE-48f0c2c3-a172-4192-9ae3-
052274181b6e",
                                        "version": "1.0.2",
                                        "fieldPath": "$.event.measurementsForVfScalingFields.
additionalMeasurements[*].arrayOfFields[0].value",
                                        "thresholdValue": 0,
                                        "direction": "GREATER",
                                        "severity": "CRITICAL",
```

```
                                    "closedLoopEventStatus": "ONSET",
                                },
                            ],
                        }
                    ],
                }
            },
        },
    }
}
```

Legacy Policy Interface (Prior to Dublin)

ONAP's legacy policy interface is also supported by setting the --use-v0 flag to true at runtime.

Websocket

Policy supports a websocket interface that will send a client notifications about changes to policies. Notifications can be received simply by opening a websocket on /pdp/notifications. This websocket interface allows the sync utility to be aware of policy updates without any need to do frequent polling.

Example websocket notification:

response

```
{
        "removedPolicies": [{
                "policyName": "DCAE.Config_MS_AGING_UVERSE_PROD_Tosca_HP_AGING_Model_cl55973_IT64_testAging.45.
xml",
                "versionNo": "45"
        }],
        "loadedPolicies": [{
                "policyName": "DCAE.Config_MS_AGING_UVERSE_PROD_Tosca_HP_AGING_Model_cl55973_IT64_testAging.46.
xml",
                "versionNo": "46",
                "matches": {
                        "ONAPName": "DCAE",
                        "ConfigName": "DCAE_HighlandPark_AgingConfig",
                        "service": "DCAE_HighlandPark_AgingConfig",
                        "guard": "false",
                        "location": " Edge",
                        "TTLDate": "NA",
                        "uuid": "TestUUID",
                        "RiskLevel": "5",
                        "RiskType": "default"
                },
                "updateType": "UPDATE"
        }],
        "notificationType": "BOTH"
}
```

When the sync container receives a policy notification it will first check if it contains a policy it cares about by matching all policies against its filter. If the notification matches, it will use the policy REST APIs to pull the policies.

REST API

An overview of available rest calls is available via their swagger gui. Of the available REST calls, we need two of the listed rest calls.

listPolicy

For routine checks to see if there is any delta, Praveen from policy said we should be using the listPolicy api first since it is less expensive call and can be used for synchronization purposes.

This returns a json list of policies and either 200 (if all pdps returned the same result) or 206 (pdps returned different results and a sync is in progress):

```
# POST json to /pdp/api/listPolicies
{
                "policyName": "DCAE.Config_MS_AGING_UVERSE_.*",
                "policyType": "Microservice"
}
```

response

```
# Gives 200 OK along with the json body of the policy
[
  "DCAE.Config_MS_AGING_UVERSE_PROD_Tosca_HP_AGING_Model_cl55973_IT64_testAging.70.xml"
]
```

getConfig

Will return a json list containing the full policy configuration. To be run whenever a websocket match (as described above) occurs. This will also be called periodically for resilliency purposes in the face of a missed policy updates.

Request:

```
# POST json to /pdp/api/getConfig
{
                "policyName": "DCAE.Config_MS_AGING_UVERSE_.*",
                "policyType": "Microservice"
}
```

Response:

response

```
# Gives 200 OK along with the json body of the policy
[
  {
    "policyConfigMessage": "Config Retrieved! ",
    "policyConfigStatus": "CONFIG_RETRIEVED",
    "type": "JSON",
    "config": "{\"service\":\"DCAE_HighlandPark_AgingConfig\",\"location\":\" Edge\",\"uuid\":\"TestUUID\",\"
policyName\":\"DCAE.AGING_UVERS_PROD_Tosca_HP_GOC_Model_cl55973_IT64_testAging\",\"configName\":\"
DCAE_HighlandPark_AgingConfig\",\"templateVersion\":\"1607\",\"priority\":\"4\",\"version\":11.0,\"
policyScope\":\"resource=Test1,service=vSCP,type=configuration,
closedLoopControlName=vSCP_F5_Firewall_d925ed73_8231_4d02_9545_db4e101f88f8\",\"riskType\":\"test\",\"
riskLevel\":\"2\",\"guard\":\"False\",\"content\":{\"signature\":{\"filter_clause\":\"event.faultFields.
alarmCondition LIKE('%chrisluckenbaugh%')\"},\"name\":\"testAging\",\"context\":[\"PROD\"],\"priority\":1,\"
prePublishAging\":40,\"preCorrelationAging\":20},\"policyNameWithPrefix\":\"DCAE.
AGING_UVERSE_PSL_Tosca_HP_GOC_Model_cl55973_IT64_testAging\"}",
    "policyName": "DCAE.Config_MS_AGING_UVERSE_PROD_Tosca_HP_AGING_Model_cl55973_IT64_testAging.45.xml",
    "policyType": "MicroService",
    "policyVersion": "45",
    "matchingConditions": {
      "ECOMPName": "DCAE",
      "ONAPName": "DCAE",
      "ConfigName": "DCAE_HighlandPark_AgingConfig",
      "service": "DCAE_HighlandPark_AgingConfig",
      "uuid": "TestUUID",
      "Location": " Edge"
    },
    "responseAttributes": {},
    "property": null
  }
]
```

Integration with a microservice

We will provide the sync utility as a dockerized container that can be run alongside a microservice container as part of a kubernetes POD.

Configuration

Configuration is currently done via either via env variables or by flag. Flags take precendence over env variables, env variables take precedence over default

General configuration

General configuration that is used regardless of which PDP API you are using.

ENV Variable	Flag	Description	Default
POLICY_SYNC_PDP_URL	--pdp-url	PDP URL to query	None (must be set in env or flag)
POLICY_SYNC_FILTER	--filters	yaml list of regex of policies to match	[]
POLICY_SYNC_ID	--ids	yaml list of ids of policies to match	[]
POLICY_SYNC_DURATION	--duration	duration in seconds for periodic checks	2600
POLICY_SYNC_OUTFILE	--outfile	File to output policies to	./policies.json
POLICY_SYNC_PDP_USER	--pdp-user	Set user if you need basic auth for PDP	None
POLICY_SYNC_PDP_PASS	--pdp-password	Set pass if you need basic auth for PDP	None
POLICY_SYNC_HTTP_METRICS	--http-metrics	Whether to expose prometheus metrics	True
POLICY_SYNC_HTTP_BIND	--http-bind	host:port for exporting prometheus metrics	localhost:8000
POLICY_SYNC_LOGGING_CONFIG	--logging-config	Path to a python formatted logging file	None (logs will write to stderr)
POLICY_SYNC_V0_ENABLE	--use-v0	Set to true to enable usage of legacy v0 API	False

V1 Specific Configuration (Used as of the Dublin release)

Configurable variables used for the V1 API used in the ONAP Dublin Release.

Note: Policy filters are not currently supported in the current policy release but will be eventually.

ENV Variable	Flag	Description	Default
POLICY_SYNC_V1_DECISION_ENDPOINT	--v1-decision-endpoint	Endpoint to query for PDP decisions	policy/pdpx/v1/decision
POLICY_SYNC_V1_DMAAP_URL	--v1-dmaap-topic	Dmaap url with topic for notifications	None
POLICY_SYNC_V1_DMAAP_USER	--v1-dmaap-user	User to use for DMaaP notifications	None
POLICY_SYNC_V1_DMAAP_PASS	--v1-dmaap-pass	Password to use for DMaaP notifications	None

V0 Specific Configuration (Legacy Policy API)

Configurable variables used for the legacy V0 API Prior to the ONAP release. Only valid when --use-v0 is set to True

ENV Variable	Flag	Description	Default
POLICY_SYNC_V0_NOTIFIY_ENDPOINT	--v0-notifiy-endpoint	websock endpoint for pdp notifications	pdp/notifications
POLICY_SYNC_V0_DECISION_ENDPOINT	--v0-decision-endpoint	rest endpoint for pdp decisions	pdp/api

Communication with the main application

A shared filesystem will serve as a method of inter-process communication between the policy-sync utility and the main application. The sync utility will update a json file which contains the current output of the /getConfig REST API call whenever a policy is created, updated, or removed from the system. The primary application should watch the json file that the policy-sync utility is creating and trigger app reconfiguration as needed.

To accomplish this applications could use:

1. Shell utilities based on the inotify kernel subsystem that allow you to run commands whenever a file is changed. Busybox's inotifyd can be used for this (included in the alpine based images here)
2. Java's built in watch service
3. Python modules such as watchdog.

Sync Utility Output

In order to make it easier to integrate, formatting and structure are designed to mimic the policies key returned by consul and/or the config binding service. Microservices can then trade calls to the config binding service for reads of this file.

```
{
  "policies": {
    "items": [
      {
        "config": {
          "service": "DCAE_HighlandPark_AgingConfig",
          "location": " Edge",
          "uuid": "TestUUID",
          "policyName": "DCAE.AGING_UVERS_PROD_Tosca_HP_GOC_Model_cl55973_IT64_testAging",
          "configName": "DCAE_HighlandPark_AgingConfig",
          "templateVersion": "1607",
          "priority": "4",
          "version": 11.0,
          "policyScope": "resource=Test1,service=vSCP,type=configuration,
closedLoopControlName=vSCP_F5_Firewall_d925ed73_8231_4d02_9545_db4e101f88f8",
          "riskType": "test",
          "riskLevel": "2",
          "guard": "False",
          "content": {
            "signature": {
              "filter_clause": "event.faultFields.alarmCondition LIKE('%chrisluckenbaugh%')"
            },
            "name": "testAging",
            "context": [
              "PROD"
            ],
            "priority": 1,
            "prePublishAging": 40,
            "preCorrelationAging": 20
          },
          "policyNameWithPrefix": "DCAE.AGING_UVERSE_PSL_Tosca_HP_GOC_Model_cl55973_IT64_testAging"
        },
        "matchingConditions": {
          "ECOMPName": "DCAE",
          "ONAPName": "DCAE",
          "ConfigName": "DCAE_HighlandPark_AgingConfig",
          "service": "DCAE_HighlandPark_AgingConfig",
          "uuid": "TestUUID",
          "Location": " Edge"
        },
        "policyConfigMessage": "Config Retrieved! ",
        "policyConfigStatus": "CONFIG_RETRIEVED",
        "policyName": "DCAE.Config_MS_AGING_UVERSE_PROD_Tosca_HP_AGING_Model_cl55973_IT64_testAging.70.xml",
        "policyType": "MicroService",
        "policyVersion": "70",
        "property": null,
        "responseAttributes": {},
        "type": "JSON"
      }
    ]
  }
}
```


Running the sync utility as a sidecar container (For use with HELM deployments, etc.).

Code will be packaged as a docker container that can be easily injected into a kubernetes mainfest.

Example K8s pod manifest (which would produce something like the diagram above):

```
# policy-config-map
apiVersion: v1
kind: policy-config-map
metadata:
  name: special-config
  namespace: default
data:
  POLICY_SYNC_USER: myusername
  POLICY_SYNC_PASS: mypassword
  POLICY_SYNC_PDP_URL: https://policy-conexus-ist-02.ecomp.cci.att.com:30281
  POLICY_SYNC_FILTER: '["DCAE.Config_MS_AGING_UVERSE_PROD.*"]'


---

apiVersion: v1
kind: Pod
metadata:
  name: Sidecar sample app
spec:
  restartPolicy: Never


  # The shared volume that the two containers use to communicate...empty dir for simplicity
  volumes:
  - name: policy-shared
    emptyDir: {}

  containers:

  # Sample app that uses inotifyd (part of busybox/alpine). For demonstration purposes only...
  - name: main
    image: dockercentral.it.att.com:5100/com.att.ecompcntr.public/baseimages/dcae-alpine-java8:1.1.0
    volumeMounts:
    - name: policy-shared
      mountPath: /etc/policies.json
      subPath: policies.json
    # For details on what this does see: https://wiki.alpinelinux.org/wiki/Inotifyd
    # you can replace '-' arg below with a shell script to do more interesting
    cmd: [ "inotifyd", "-", "/etc/policies.json:c" ]


    # The sidecar app which keeps the policies in sync
  - name: policy-sync
    image: dockercentral.it.att.com:5100/com.att.dcae.hp/policy-sync:0.0.1
    envFrom:
      - configMapRef:
          name: special-config

    volumeMounts:
    - name: policy-shared
      mountPath: /etc/policies
```

Monitoring failures

Most issues should be with the sync container's connectivity to PDP (e.g. connection errors, credential issues, etc..). When these errors arise, the impact may be that a mS no longer receives new policies, receives old policies, receives policies slowly, etc.

For monitoring these kind of errors we can:

- Include a prometheus endpoint that can be automatically discovered and scraped by Prometheus to track numeric metrics (e.g. # of failures, # of collected policies, etc.).
- Log errors and other info to the pod logs. Basic troubleshooting could then be done using kubectl logs. You could then use log forwarding tools to route logs to a logging index (Azure monitor, Elasticsearch, loki, splunk).
- Log to a file in the /opt/logs filesystem on the worker node...then pick it up via splunk for alerting (probably necessary for EOM).

Honolulu Release Deliverables

- Policy Sidecar seed code in gerrit (Will be submitted into dcaegen2/deployment repository as new sub-module - ***dcae-services-policy-sync***)
- Policy Sidecar container in Nexus3
- Example helm chart invoking side-car service
- Documentation (wiki)

Attributions

- Inspired by the open source git-sync utility which essentially does the same thing for git repositories.
- The existing policy handler microservice: https://github.com/onap/dcaegen2-platform-policy-handler/tree/master/policyhandler