ONAP DCAE Controller and policy – take 6.5.1

- no more merging of policies into application_config by plugin
- using config-binding service for retrieval of the whole collection of policies along the application config
- no CDAP in ONAP anymore

contributors: Alex Shatov & Jack Lucas & Chris Rath & Tommy Carpenter & Mike Hwang & Shu Shi & Andrew Gauld & Dominic Lunanuova & Avi Zahavi & Terry Schmalzried & Lusheng Ji & Shrikant Acharya & Patricia Heffner & Dan Musgrove

2018-0226

License

Policies and DCAE controller – definitions and assumptions

Definitions, assumptions, relationships, and restrictions

- **Policy** is a volatile subset of configuration that can be changed after the component is deployed-installed-started
- Changing the policy on the component (= node instance) is expected not to affect the topology of deployment-installation that is described in the blueprint
- **Many-to-many relationship between the policies and components** (= node instances) **across multiple deployments/blueprints**
  - There can be **many policies per each component**
  - Each policy can also be assigned to **many components**
- **cloudify** does expose the direct REST API (/node-instances) to node-instances=components with run-time-properties across all the deployments.
  - this data is going to be used by the deployment-handler to find all the components the policy applies to
- Policy change on one component is expected not to have any dependence on other components = Policy can be randomly updated on any component with no need to change another component
- Both the collection and the fields of policies on the components are expected to change (CUD) after the deployment of the component
- The size of a single policy_body is expected not to exceed 512kB when encoded into base64 – this is the value length limitation in consul

Policy storage and using config-binding service to bring the collection of policies along config

- Policy related logic in DCAE-Controller will not change any properties in application config
  - **no more merging** of policy config into application config by the DCAE-Controller.
  - **it is responsibility of the component to figure out how to merge the collection of policies into application config**
- Config-binding service (CBS) used to set the config properties like resl, dmaap, etc will be enhanced to retrieve the records from **<service_component_name>:policies/** folder from consul kv and return along the component config
- Policy related plugins of cloudify will store and update the full collection of policies (keyed by policy_id) on the component into folder **<service_component_name>:policies/items/** on consul
- The policies collection is an unsorted list of policy_body objects
- The policy object (policy_body) is the single json structure associated with a single policy received from policy-engine on /getConfig call

Option #1 - Single policy per each **dcae.nodes.policy** node, multiple **dcae.nodes.policy** nodes per component

Structure and Lifecycle for the single policy per policy node – multiple policy nodes per blueprint

- Predefined=fixed number of policies per component that is specified in the blueprint + inputs on deployment of the blueprint
- Each policy is identified by **policy-id** (versionless policyName)**,** whereas the content of the policy content is referred to as **policy_body["config"]**
- Ways to provide the policy_id and policy_body values to DCAE controller
  a. multiple nodes of type **dcae.nodes.policy** each with the property of **policy_id** are expected to be provided in the **blueprint**. The property policy_id will either have the default or assigned value that contains the policy-id value. The component nodes will have a depend_on kind of relationship towards one or more policy nodes
  b. multiple policy_id values can also be provided in the **inputs to the deployment** create step.
  c. DCAE controller (respective node level plugin inside cloudify) will retrieve the latest **policy_body** for each **policy_id** from the policy-engine through the call to policy-handler (DCAE-C microservice).
  d. event 'policy-updated' from the **policy-engine** contains only policyName & policyVersion - the policy_id can be extracted from policyName.
  e. policy-handler filters the 'policy-updated' events by scope prefixes ["DCAE.Config_"] – specified in policy-handler config on consul-kv
  f. DCAE controller (respective node level plugin inside cloudify) will populate the policy related record in consul-kv with the updated collection of policies on the component
  g. CBS will expose a new API to bring the policies collection from consul kv

Option #2 - varying collection of policies per policy-filter in **dcae.nodes.policies** node, multiple dcae.nodes.policies nodes per component

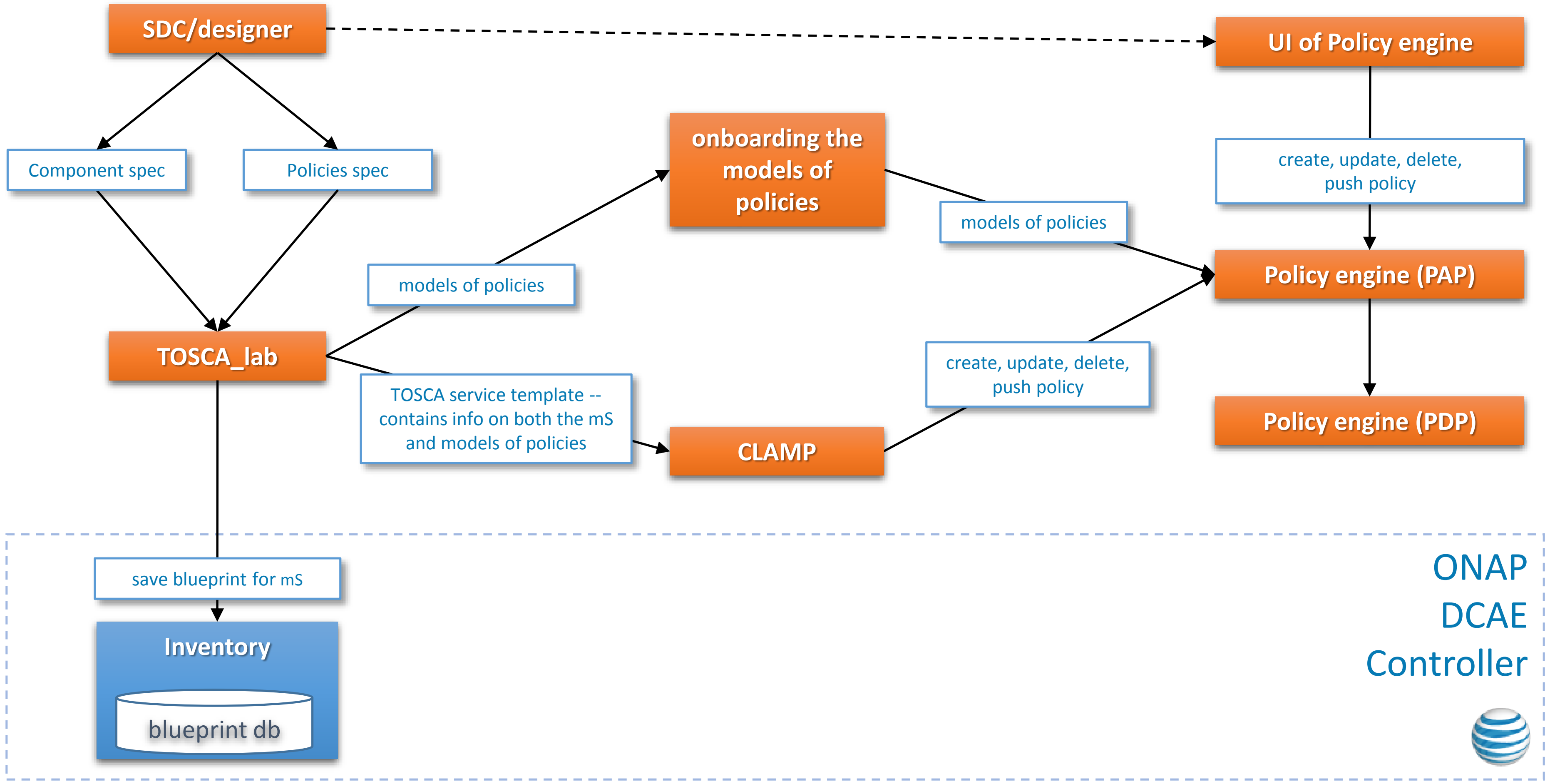
Structure and Lifecycle for the policies with policy-filter

- Collection of Policies can be defined by the **dcae.nodes.policies** node in the blueprint
- The policy-filter = properties of the **dcae.nodes.policies** node will mimic the parameters of PDP /getConfig API except the requestID (uuid) that is unique per each request. Here is the sample.
    a) "**configAttributes**": {"key1":"value1"},
    b) "**configName**": "alex_config_name",
    c) "**ecompName**": "DCAE",
    d) "**policyName**": "DCAE.Config_multi.*",
    e) "**unique**": false
- The size of the collection of policies that matches the policy-filter may increase or decrease over time.
- Policy handler in DCAE-Controller will bring multiple latest policies from PDP that match the policy-filter
- The policy related plugins (and decorators) will store the multiple policies in runtime_properties along with the collection of the policy-filters
- Respective node level plugin inside cloudify will populate the policy related record in consul-kv with the updated collection of policies on the component
- CBS will expose a new API to bring the policies collection from consul kv

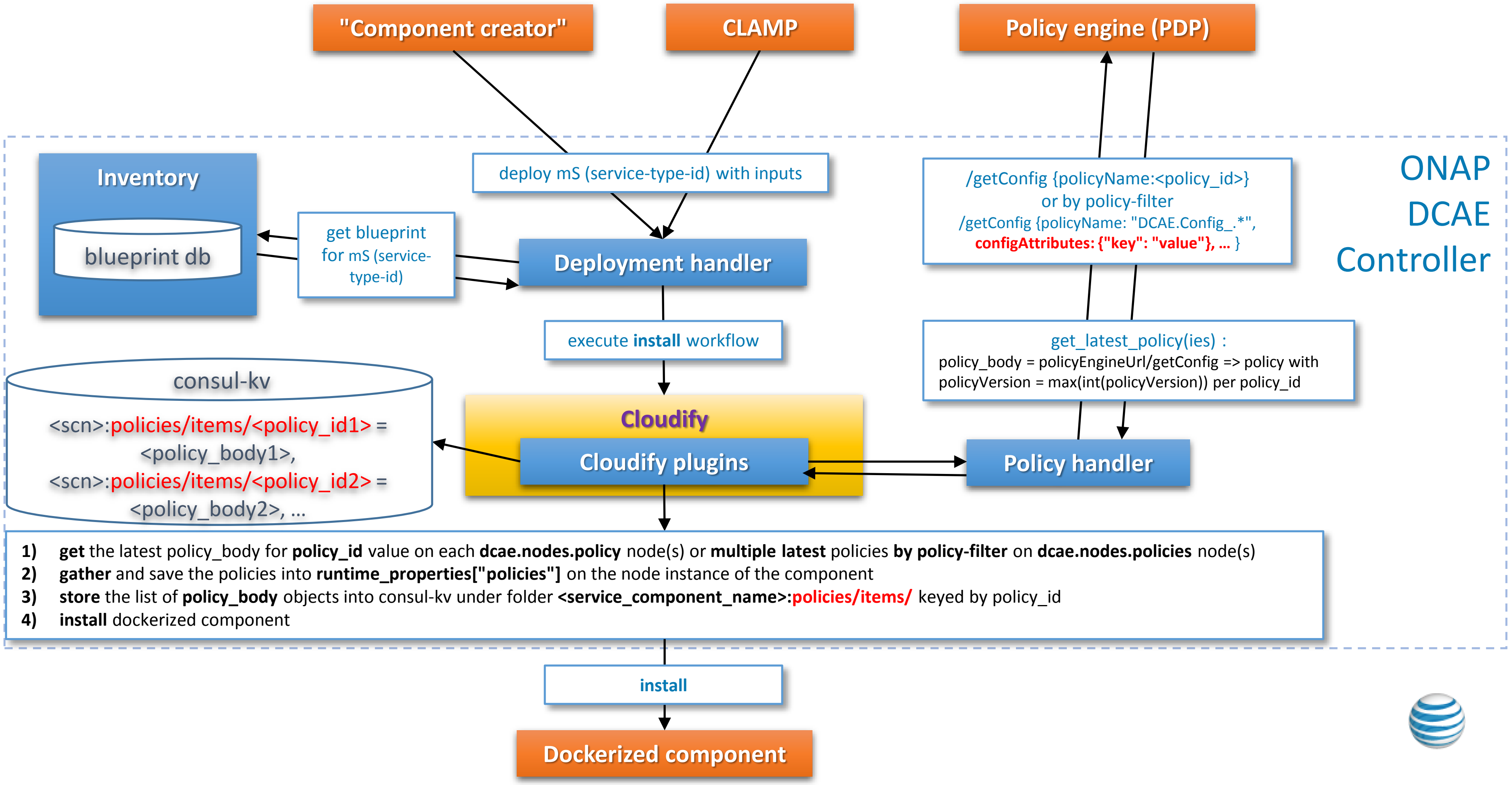
projects affected by move from take 5.x to 6.y

1. **component spec**
   a. no need to specify the properties that are part of policies – DCAE-Controller will not merge the policies into application_config
   b. do not add or remove the spec for policy apply_order if it was created for R2 already
2. **TOSCA_lab**
   a. optionally separate the generation of component blueprint and policy models from the component spec or from separate component spec and policy spec
   b. support for multiple dcae.nodes.policy nodes in the blueprint
   c. support for multiple dcae.nodes.policies nodes with policy_filter in the blueprint as planned for R2
   d. related changes to TOSCA service template consumed by CLAMP
3. **DCAE_CLI** tool – keep up with TOSCA Lab
4. **onap-dcae-policy-lib**
   a. store/delete the <scn>:**policies/ folder** of records in consul kv
   b. remove the shallow merge logic and remove the policy sorting logic
   c. pass the whole collection of policy_body objects instead of policy config to plugins
5. **dockerplugin**
   a. remove the policy merge call – we are no longer merging the policies into application config
   b. change the policy-update notification message to pass policies, updated_policies, removed_policies (lists of policy_body objects) with the new message type of "policies" instead of "policy"
   c. on delete the component node – we need to delete-tree in consul-kv or use the new decorator @Policies.cleanup_policies_on_node() to delete the policies record in consul-kv
6. ~~**cdapplugin, CDAP broker**~~ – obsolete
7. **config-binding service** (CBS)
   a. new http API GET <CBS>/service_component_**all**/<service_component_name> to return the <scn>:**policies/** folder of record from consul kv and along the config and other data
   b. add a new API into the client lib
8. **components under DCAE-C**
   a. get the **full collection of policies** (policy_body objects) from <CBS>/service_component_**all**/<service_component_name>
   b. all the **merging of the policies into application config** is the responsibility of the component itself – DCAE-Controller is no longer doing any policy config merging into application config
   c. policy-update notification will no longer contain the application_config – the component is responsible to retrieve the full config from CBS
   d. policy-update notification data format changed – it will contain the lists of policy_body objects for the new field of **policies** on component, as well as for **updated_policies**, and **removed_policies**

mS blueprint and policies defined North of DCAE controller

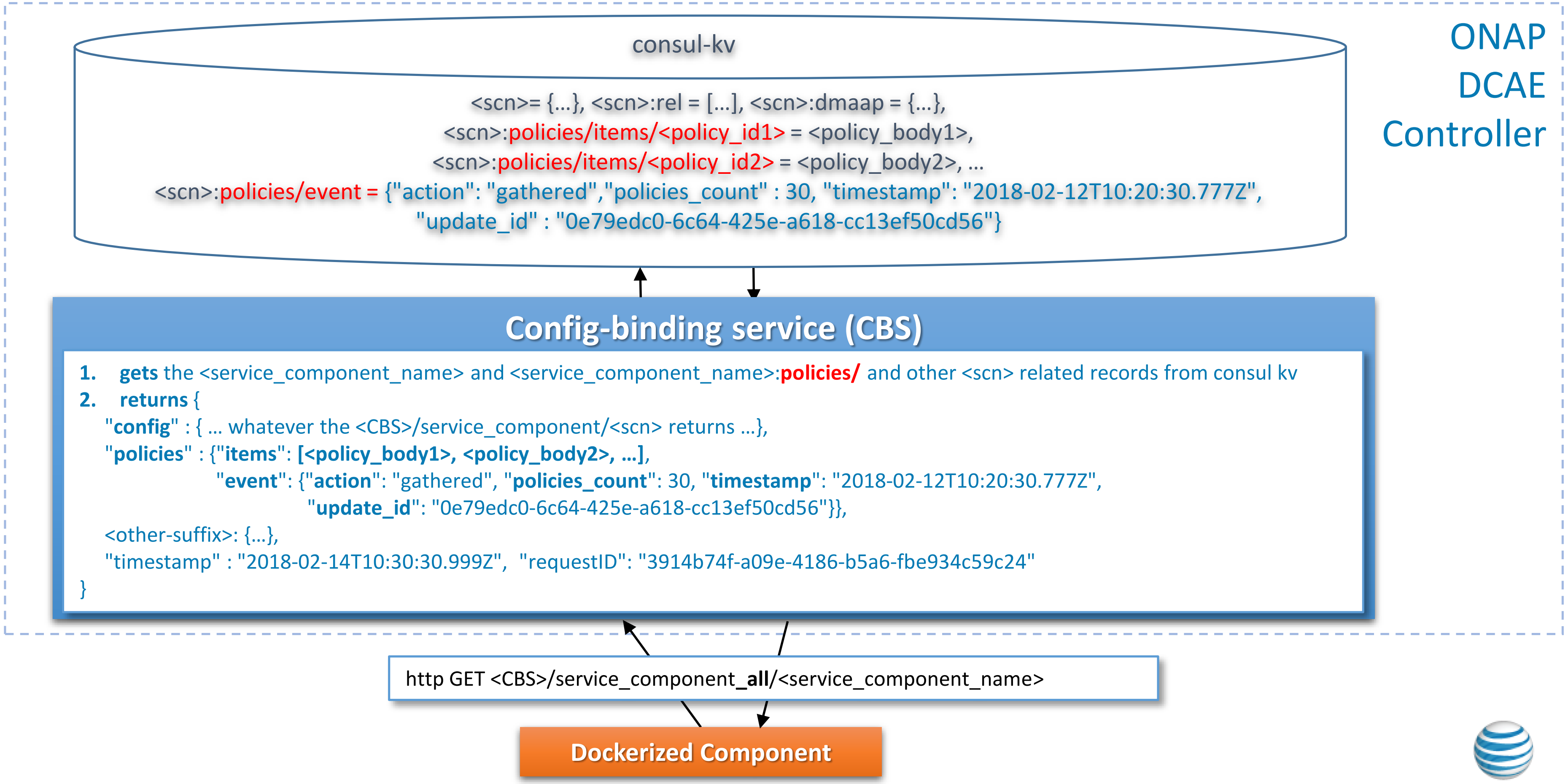


policies during the install workflow in DCAE controller


"Component creator"
CLAMP
Policy engine (PDP)
ONAP DCAE Controller
Inventory
blueprint db
deploy mS (service-type-id) with inputs
get blueprint for mS (service-type-id)
Deployment handler
/getConfig {policyName:<policy_id>}
or by policy-filter
/getConfig {policyName: "DCAE.Config_.*",
configAttributes: {"key": "value"}, ... }
execute install workflow
get_latest_policy(ies) :
policy_body = policyEngineUrl/getConfig => policy with policyVersion = max(int(policyVersion)) per policy_id
consul-kv
<scn>:policies/items/<policy_id1> = <policy_body1>,
<scn>:policies/items/<policy_id2> = <policy_body2>, ...
Cloudify
Cloudify plugins
Policy handler
install
Dockerized component

1) **get** the latest policy_body for **policy_id** value on each **dcae.nodes.policy** node(s) or **multiple latest** policies **by policy-filter** on **dcae.nodes.policies** node(s)
2) **gather** and save the policies into **runtime_properties["policies"]** on the node instance of the component
3) **store** the list of **policy_body** objects into consul-kv under folder **<service_component_name>:**policies/items/ keyed by policy_id
4) **install** dockerized component

component gets policies from config-binding service (CBS) on install and any time later



consul-kv:

```
<scn>= {…}, <scn>:rel = […], <scn>:dmaap = {…},
<scn>:policies/items/<policy_id1> = <policy_body1>,
<scn>:policies/items/<policy_id2> = <policy_body2>, …
<scn>:policies/event = {"action": "gathered","policies_count" : 30, "timestamp": "2018-02-12T10:20:30.777Z",
"update_id" : "0e79edc0-6c64-425e-a618-cc13ef50cd56"}
```

Config-binding service (CBS):

1. **gets** the <service_component_name> and <service_component_name>:**policies/** and other <scn> related records from consul kv
2. **returns** {

```
  "config" : { … whatever the <CBS>/service_component/<scn> returns …},
  "policies" : {"items": [<policy_body1>, <policy_body2>, …],
                "event": {"action": "gathered", "policies_count": 30, "timestamp": "2018-02-12T10:20:30.777Z",
                          "update_id": "0e79edc0-6c64-425e-a618-cc13ef50cd56"}},
  <other-suffix>: {…},
  "timestamp" : "2018-02-14T10:30:30.999Z",  "requestID": "3914b74f-a09e-4186-b5a6-fbe934c59c24"
}
```

http GET <CBS>/service_component_**all**/<service_component_name>

Policies and DCAE controller flows – high level view

Policy flow – on install

1. **Cloudify plugin - policy_id** or **policy-filter** from the **blueprint + inputs**
   a) **gets** the latest policy_body(ies) from Policy Handler on dcae.nodes.policy and dcae.nodes.policies nodes
   b) **gathers** and saves all policies in runtime_properties["policies"] as dict by policy_id on component node instance
   c) **store** the list of **policy_body** objects into consul-kv under folder **<service_component_name>:policies/items** keyed by policy_id
   d) **install** the component

2. Policy Handler to **get the latest policy by policy_id** (http get /policy_latest/<policy_id>) or by policy-filter (http post /policies_latest policy-filter: {…})
   a) passes policy_id as policyName or passes the policy-filter to policy-engine
   b) gets the collection of policies from policy engine (/getConfig)
   c) picks the latest policy_body by max(int(policyVersion)) on each returned policy_id
   d) returns the collection of policy object(s) to the policy_get plugin

3. Component after install
   a. get the **full collection of policies** (policy_body objects) from <CBS>/service_component_**all**/<service_component_name>
   b. all the **merging of the policies into application config** is the responsibility of the component itself – DCAE-Controller is no longer doing any policy config merging into application config
   c. policy-update notification will no longer contain the application_config – the component is responsible to retrieve the full config from CBS
   d. policy-update notification data format changed – it will contain the lists of policy_body objects for the new field of **policies** on component, as well as for **updated_policies**, and **removed_policies**

Cloudify blueprint for policy in DCAE controller – Shu’s approach + Mike's docker_config

set policy-update interface

```
node_types:
 dcae.nodes.vm
    derived_from: cloudify.nodes.Root
        properties:
          docker_config:
               default: {}

interfaces:
          cloudify.interfaces.lifecycle:
              create: node_plugin.vm.vm_create
          dcae.interfaces.policy:
              policy_update: your_plugin.yourplugin.your_policy_update
```

interface name is hardcoded in deployment-handler for policy-update execution

provide policy_id on policy node(s)

```
imports:
- https://nexus01.research.att.com:8443/repository/solutioning01-mte2-raw/type_files/dcaepolicy/1.0.0/node-type.yaml

node_templates:
 storage_policy:
   type: dcae.nodes.policy
   properties:
       policy_id: DCAE.Config_VM_STORAGE

 memory_policy:
   type: dcae.nodes.policy
   properties:
       policy_id: DCAE.Config_VM_MEMORY
```

in config of policy-handler "scope_prefixes" : ["DCAE.Config_"]

property names are hardcoded in policy-related plugins for policy-update notification

assign policies to components thru relationships, specify the field for policies in application_config and set policy script-path

```
node_templates:
 vm_1:
   type: dcae.nodes.vm
   properties:
     docker_config:
       policy:
         trigger_type: "docker"
         script_path: "/opt/app/reconfigure.sh"

   relationships:
     - target: storage_policy
       type: cloudify.relationships.depends_on
     - target: memory_policy
       type: cloudify.relationships.depends_on
...
```

Alternative entry of policy_id through the inputs for the blueprint

set policy_id property to **get_input** on policy node(s) in the blueprint

```
imports:
- https://nexus01.research.att.com:8443/repository/solutioning01-mte2-raw/type_files/dcaepolicy/1.0.0/node-type.yaml

node_templates:
 storage_policy:
   type: dcae.nodes.policy
   properties:
      policy_id: { get_input: input_key_for_storage }

 memory_policy:
   type: dcae.nodes.policy
   properties:
      policy_id: { get_input: input_key_for_memory }
```

provide **inputs** values during deployment (yaml)

```
inputs:
  input_key_for_storage: DCAE.Config_VM_STORAGE
  input_key_for_memory: DCAE.Config_VM_MEMORY
```

the same **inputs** as **json**

```
inputs: {
  "input_key_for_storage": "DCAE.Config_VM_STORAGE",
  "input_key_for_memory": "DCAE.Config_VM_MEMORY"
}
```

in config of policy-handler
"scope_prefixes" :
["DCAE.Config_"]

Alternative - **varying** collection of policies by policy-filter – dcae.nodes.policies

dcae.nodes.policies in

https://nexus01.research.att.com:8443/repository/solutioning01-mte2-raw/type_files/**dcaepolicy/2.0.0/node-type.yaml**

```
data_types:
  dcae.data.policy_filter:
    properties:
      policyName :
        type : string
        default: "DCAE.Config_.*"
      configName :
        type : string
        default: ""
      onapName :
        type : string
        default: "DCAE"
      configAttributes :
        default: {}
      unique :
        type : boolean
        default: false

node_types:
  dcae.nodes.policies
    derived_from: cloudify.nodes.Root
      properties:
        policy_filter:
          type: dcae.data.policy_filter
          default: {}

      interfaces:
        cloudify.interfaces.lifecycle:
          create: dcaepolicy.dcaepolicyplugin.policy_get
```

params for PDP/getConfig (identical to PDP API) = policy-filter

find the link to PDP /getConfig wiki on last page

PDP /getConfig wiki: The filter works as a **combined "AND"** operation. To retrieve all policies using "sample" as configName, The request needs to have policyName = ".*" and configName = "sample"

provide policy-filter on dcae.nodes.policies node(s)

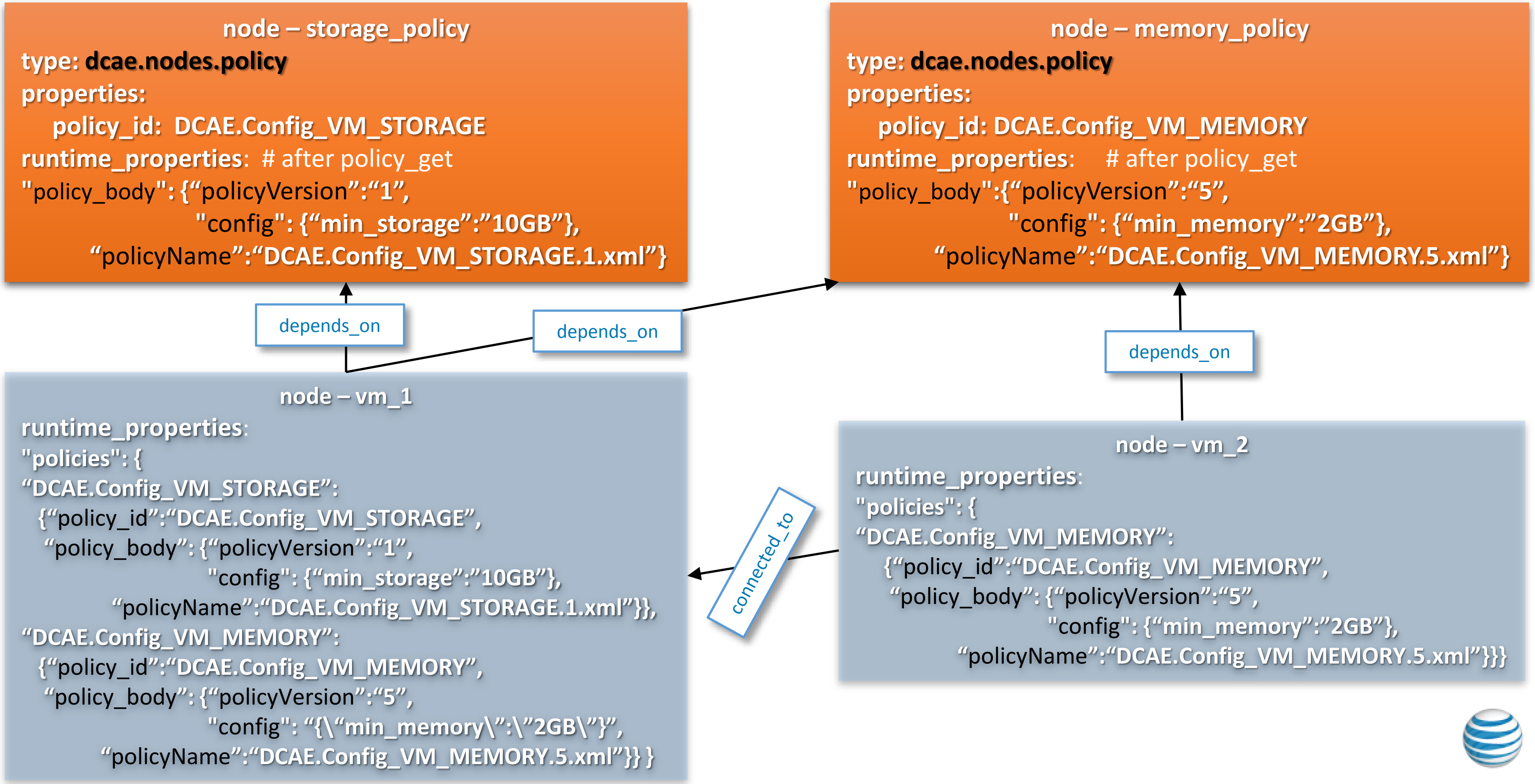
```
imports:
- https://nexus01.research.att.com:8443/repository/solutioning01-mte2-
raw/type_files/dcaepolicy/2.0.0/node-type.yaml

node_templates:
 storage_policies:
   type: dcae.nodes.policies
   properties:
     policy_filter:
       policyName : "DCAE.Config_MS_DKAT.*"
       configAttributes : {"key1":"value1"}

 memory_policy:
   type: dcae.nodes.policy
   properties:
     policy_id: DCAE.Config_VM_MEMORY
```

in config of policy-handler "scope_prefixes" : ["DCAE.Config_"]

assign policies to components thru relationships, specify the field for policies in application_config and set script_path for policy-update notification

```
node_templates:
 vm_1:
   type: dcae.nodes.vm
   properties:
     docker_config:
       policy:
         trigger_type: "docker"
         script_path: "/opt/app/reconfigure.sh"

   relationships:
     - target: storage_policies
       type: cloudify.relationships.depends_on
     - target: memory_policy
       type: cloudify.relationships.depends_on
…
```

SAMPLE topology - policy nodes in composition– Shu’s approach



```
node – storage_policy
type: dcae.nodes.policy
properties:
    policy_id:  DCAE.Config_VM_STORAGE
runtime_properties:  # after policy_get
"policy_body": {“policyVersion”:“1”,
                         "config": {“min_storage”:”10GB”},
         “policyName”:“DCAE.Config_VM_STORAGE.1.xml”}
```

```
node – memory_policy
type: dcae.nodes.policy
properties:
    policy_id: DCAE.Config_VM_MEMORY
runtime_properties:     # after policy_get
"policy_body":{“policyVersion”:“5”,
                         "config": {“min_memory”:”2GB”},
         “policyName”:“DCAE.Config_VM_MEMORY.5.xml”}
```

```
node – vm_1
runtime_properties:
"policies": {
“DCAE.Config_VM_STORAGE”:
  {“policy_id”:“DCAE.Config_VM_STORAGE”,
   “policy_body”: {“policyVersion”:“1”,
                         "config": {“min_storage”:”10GB”},
           “policyName”:“DCAE.Config_VM_STORAGE.1.xml”}},
“DCAE.Config_VM_MEMORY”:
  {“policy_id”:“DCAE.Config_VM_MEMORY”,
   “policy_body”: {“policyVersion”:“5”,
                         "config": “{\“min_memory\”:\”2GB\”}”,
          “policyName”:“DCAE.Config_VM_MEMORY.5.xml”}} }
```

```
node – vm_2
runtime_properties:
"policies": {
“DCAE.Config_VM_MEMORY”:
   {“policy_id”:“DCAE.Config_VM_MEMORY”,
    “policy_body”: {“policyVersion”:“5”,
                         "config": {“min_memory”:”2GB”},
          “policyName”:“DCAE.Config_VM_MEMORY.5.xml”}}}
```

Terminology in Policy engine and mapping to DCAE controller

Blueprint – yaml	DCAE-C	Policy Handler to use the Policy engine (PDP) API
• **dcae.nodes.policy** – node type for single policy identified by policy_id • **dcae.nodes.policies** – node type for varying collection of policies by policy-filter (the same as in PDP /getConfig)	• identify policy node(s) by the type **dcae.nodes.policy** or **dcae.nodes.policies** • find policies on component node by following the depends.on relationship(s) up to the policy node(s)	**policyName -- ** String - PK to policy object in PDP – formatted as a ***file name***. *example: "**DCAE.Config**_dcae_policy_name.2.xml"* *The delimiter is "."* DCAE *is the scope of the policy, there could be several* subscopes *each delimited by the dot "."* Config_ *is the **policyClass="Config"** followed by the delimiter "_" – PAP does not have that in the policyName – only PDP has it* *dcae_policy_name is the actual name (code) of the policy* *"2" is the stringified integer value of the policy version - owned by PAP* *"xml" is the extension that does not correlate with the format of the policy body*
policy_id – string property in blueprint + input startsWith **"DCAE.Config_"**	**policy_id --** *is the prefix (scope + ".Config_" + policy_name) of the policy file name* **policy_id** *is the versionless left part of the policyName*	**policyName = policy_id + "." + <version> + ".xml"**
blueprint does not know about policy version	**policy_body["policyVersion"]** – stringified int – use to detect the update	**policyVersion** – *stringified integer that is owned and autoincremented by PAP on each policy create and update*
runtime policy config is not expected to be used inside blueprint	**policy_body["config"]** – safely parses to JSON	*on creating/updating the policy*: **configBody** – stringified JSON *on retrieving the policy*: http /getConfig: **config** – stringified JSON

Sample messages to REST API of Policy-engine – *maze of policyNames*

/createPolicy or /updatePolicy -- in PAP

```
// policyName = <scope>.<policy-name>  -- yes 'recursive' definition ☹
{
  "policyName": "DCAE.host_capacity_policy_id_value",
  "policyClass": "Config",
  "configBody": "{\"hello\":\"world\"}",
  "configBodyType": "JSON",
  "configName": "alex_config_name",
  "ecompName": "DCAE",
  "policyConfigType": "Base",
  "policyDescription": "sample policy" ,
  "ttlDate": "2017-08-14T16:54:31.696Z"
}
```

find the link to PE wiki on last page

// PAP creates-updates the policy with name and auto-increments the version
// PAP policyName = <scope>.**<class>_**<policy-name>.<version>.xml
PAP response: "content-type": "**text/plain**;charset=ISO-8859-1"
Transaction ID: a8b1b3fe-60b3-4b34-89e2-8fabd430282c --Policy with the name **DCAE.Config_host_capacity_policy_id_value**.1.xml was successfully created.

/pushPolicy – from PAP to PDP into 'default' group

```
// policyName = <scope>.<policy-name>
{
  "policyName": "DCAE.host_capacity_policy_id_value",
  "policyType": "Base"
}
```

policy_id in blueprint (yaml)

```
// policy_id = <scope>.<class>_<policy-name>
policy_id: DCAE.Config_host_capacity_policy_id_value
```

DCAE policy-handler: config in consul

```
"scope_prefixes" : ["DCAE.Config_"]
```

DCAE policy-handler: /getConfig – from PDP

```
// all policies (all versions) in scope.class "DCAE.Config_"  -- append ".*"
{
  "policyName": "DCAE.Config_.*"
}
// specific policy by policy_id -- PDP returns all versions
{
  "policyName": "DCAE.Config_host_capacity_policy_id_value"
}
// get the single version of the policy only if you know the version and the
// full-policy-name = <scope>.<class>_<policy-name>.<version>.xml
{
  "policyName": "DCAE.Config_host_capacity_policy_id_value.5.xml"
}
```

Sample policy_body structure in policy-handler retrieved from PDP for policy_id

/createPolicy or /updatePolicy -- in PAP

```
{
  "policyName": "DCAE.host_capacity_policy_id_value",
  "policyClass": "Config",
  "configBody": "{\"hello\":\"world\"}",
  "configBodyType": "JSON",
  "attributes": {"MATCHING":{"priority":"1"}},
  "configName": "alex_config_name",
  "ecompName": "DCAE",…
}
```

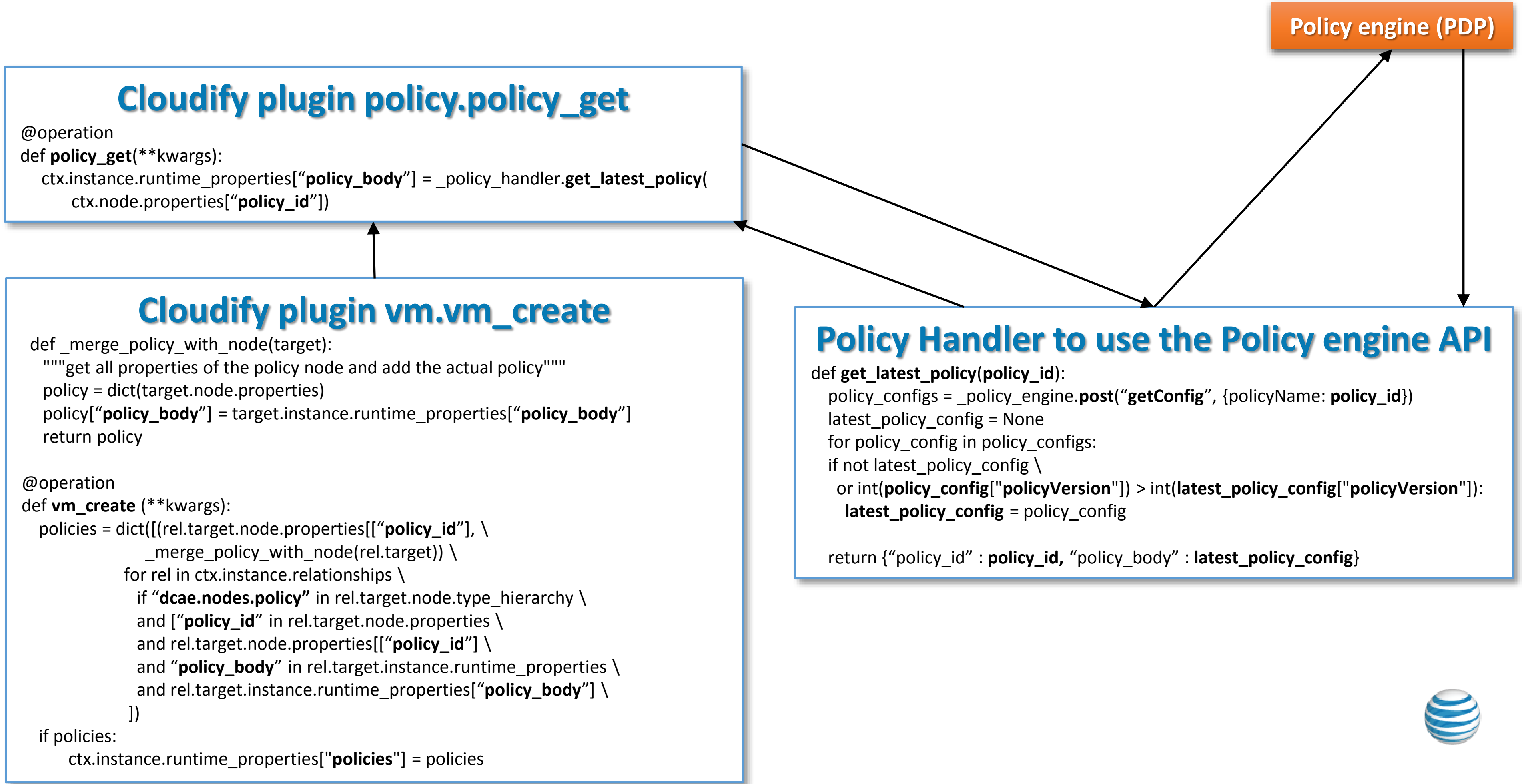
DCAE policy-handler retrieves all versions for policy_id: /getConfig – from PDP

```
// specific policy by policy_id -- PDP returns all versions
// policy_id = "DCAE.Config_host_capacity_policy_id_value"
{
  "policyName": "DCAE.Config_host_capacity_policy_id_value"
}
```

DCAE policy-handler picks the **max int(policyVersion)** and parses the config as json

```
"policy_body": {
                    "policyName": "DCAE.Config_host_capacity_policy_id_value.69.xml",
                    "policyConfigMessage": "Config Retrieved! ",
                    "responseAttributes": {},
                    "policyConfigStatus": "CONFIG_RETRIEVED",
                    "type": "JSON",
                    "matchingConditions": { "ECOMPName": "DCAE", "ConfigName": "alex_config_name", "priority":"1"},
                    "property": null,
                    "config": {
                                  "hello": "world"
                    },
                    "policyVersion": "69"
}
```

Cloudify plugins getting the policy values from Policy engine through Policy Handler



Cloudify plugin policy.policy_get

```
@operation
def policy_get(**kwargs):
   ctx.instance.runtime_properties["policy_body"] = _policy_handler.get_latest_policy(
        ctx.node.properties["policy_id"])
```

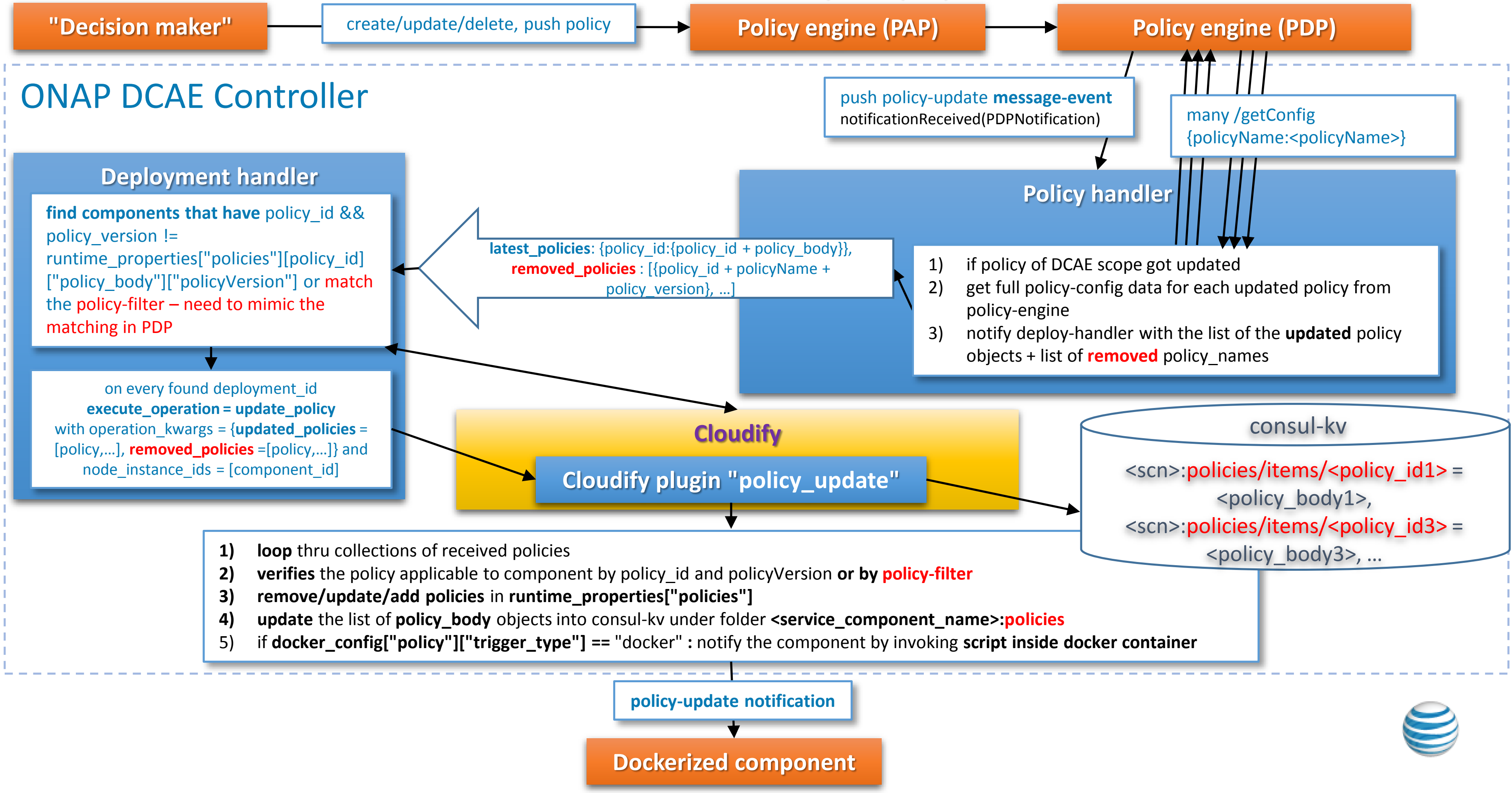
Cloudify plugin vm.vm_create

```
def _merge_policy_with_node(target):
  """get all properties of the policy node and add the actual policy"""
  policy = dict(target.node.properties)
  policy["policy_body"] = target.instance.runtime_properties["policy_body"]
  return policy

@operation
def vm_create (**kwargs):
   policies = dict([(rel.target.node.properties[["policy_id"], \
                     _merge_policy_with_node(rel.target)) \
                for rel in ctx.instance.relationships \
                  if "dcae.nodes.policy" in rel.target.node.type_hierarchy \
                  and ["policy_id" in rel.target.node.properties \
                  and rel.target.node.properties[["policy_id"] \
                  and "policy_body" in rel.target.instance.runtime_properties \
                  and rel.target.instance.runtime_properties["policy_body"] \
                 ])
   if policies:
       ctx.instance.runtime_properties["policies"] = policies
```

Policy Handler to use the Policy engine API

```
def get_latest_policy(policy_id):
   policy_configs = _policy_engine.post("getConfig", {policyName: policy_id})
   latest_policy_config = None
   for policy_config in policy_configs:
   if not latest_policy_config \
    or int(policy_config["policyVersion"]) > int(latest_policy_config["policyVersion"]):
     latest_policy_config = policy_config

   return {"policy_id" : policy_id, "policy_body" : latest_policy_config}
```

DCAE controller – how the policy updated


"Decision maker"
create/update/delete, push policy
Policy engine (PAP)
Policy engine (PDP)
ONAP DCAE Controller
push policy-update message-event
notificationReceived(PDPNotification)
many /getConfig
{policyName:<policyName>}
Deployment handler
Policy handler
latest_policies: {policy_id:{policy_id + policy_body}},
removed_policies : [{policy_id + policyName + policy_version}, …]
Cloudify
Cloudify plugin "policy_update"
consul-kv
<scn>:policies/items/<policy_id1> = <policy_body1>,
<scn>:policies/items/<policy_id3> = <policy_body3>, …
policy-update notification
Dockerized component

find components that have policy_id && policy_version != runtime_properties["policies"][policy_id]["policy_body"]["policyVersion"] or match the policy-filter – need to mimic the matching in PDP

on every found deployment_id
execute_operation = update_policy
with operation_kwargs = {**updated_policies** = [policy,…], **removed_policies** =[policy,…]} and node_instance_ids = [component_id]

1) if policy of DCAE scope got updated
2) get full policy-config data for each updated policy from policy-engine
3) notify deploy-handler with the list of the **updated** policy objects + list of **removed** policy_names

1) **loop** thru collections of received policies
2) **verifies** the policy applicable to component by policy_id and policyVersion **or by policy-filter**
3) **remove/update/add policies** in **runtime_properties["policies"]**
4) **update** the list of **policy_body** objects into consul-kv under folder **<service_component_name>:policies**
5) if **docker_config["policy"]["trigger_type"]** == "docker" : notify the component by invoking **script inside docker container**

Policy update flow inside DCAE controller – high level view

1. Policy Handler North API gets **policy-update event** from the policy engine
    a) receives the collection of policy names and policy_version, no policy_body
    b) event contains many irrelevant policies
    c) filters by policyName.startsWith(“**DCAE.Config_**”)
    d) retrieves full policy-object from policy-engine per each updated policyName
    e) extracts policy_id from each policyName – trims off the extension and the version
    f) tries to get from PDP the latest policy for each policy_id extracted from the each policyName of removed-policy list
        a) if another version of the removed-policy found in PDP – the policy-handler will use that as an update-policy rather than remove policy
        b) if policy not found in PDP, then policy-handler will pass this policyName as removed policy downstream to deployment handler
    g) sends policy-updated list with full policy objects along with the list of removed {policy_id + policyVersion} to Deploy-handler
2. **Deploy-handler** {policy_id: {policy_id, policy_body : {<full object from PDP>}}} + removed_policies [policy_id+policyVersion+policyName]
    a) calls cloudify and finds **all the components** under cloudify = node instances
    b) finds the node-instances that have the policy_id and not equal policy_version
    c) finds the node-instances that match the policy-filter -- mimics the matching done in PDP
    d) finds the node-instances that have the removed policies and collects the removed policies
    e) calls cloudify to **execute_operation=update_policy** on each found deployment with the list of node-instances and passes updated_policies: [{policy_id, policy_body}], added_policies: {policy_filter_id: {policy_filter_id, policies: {…}}], and removed_policies: [policy_id, …]
3. **Plugin in cloudify**
    a) **verifies** the received policies are applicable to component by policy_id and policy_version or matches by policy-filter
    b) **remove/update/add policies** in **runtime_properties["policies"]** of the node instance for the component
    c) **store** the list of **policy_body** objects into consul-kv under key of **<service_component_name>:policies**
    d) **notify** the component about the policies change by invoking script **docker_config**["**policy**"]["**script_path**"] inside the docker container plugins for Dockerized components in case **docker_config**["**policy**"]["**trigger_type**"] == "docker"

Jack L. approach on execute_operation policy_update – high-level

policy_update in blueprint

```
node_types:
  dcae.node.vm
    derived_from: cloudify.nodes.Root
    interfaces:
      dcae.interfaces.policy:
        policy_update: your_plugin.yourplugin.your_policy_update
```

yourplugin.your_policy_update in Cloudify

```
@operation
def your_policy_update(updated_policies, removed_policies,**kwargs):
   policies = ctx.instance.runtime_properties[“policies”]
   for policy in updated_policies:
      if policy[“policy_id”] in policies:
         policies[policy[“policy_id”]]["policy_body"] = policy["policy_body"]
   ctx.instance.runtime_properties[“policies”] = policies
```

Run execute_operation policy_update on specific component_id = vm_2_9d982

```
cfy executions start -d alex-game_depl -w execute_operation -p
"{'operation':'dcae.interfaces.policy.policy_update','operation_kwargs':{'updated_policies':[{'policy_id':' DCAE.Config_VM_MEMORY', 'policy_body':
{"config": '{\“min_memory\”:\”8GB\”}','policyVersion':’222’,'policyName':'DCAE.Config_VM_MEMORY.222.xml'}}], 'removed_policies':[]},
'node_instance_ids':[‘vm_2_9d982']}"
```

node – vm_2 (instance vm_2_9d982) before:

```
policies: {
“DCAE.Config_VM_MEMORY”:
{“policy_id”:“DCAE.Config_VM_MEMORY”,
 “policy_body”: {“policyVersion”:“5”, "config":
“{\“min_memory\”:\”2GB\”}”,
“policyName”:“DCAE.Config_VM_MEMORY.5.xml”}}
```

node – vm_2 (instance vm_2_9d982) after:

```
policies: {
“DCAE.Config_VM_MEMORY”:
{“policy_id”:“DCAE.Config_VM_MEMORY”,
 “policy_body”: {“policyVersion”:“222”, "config":
“{\“min_memory\”:\”8GB\”}”,
“policyName”:“DCAE.Config_VM_MEMORY.222.xml”}}
```

On **policy-update:** pass policies delta and state to the component

dockerized component

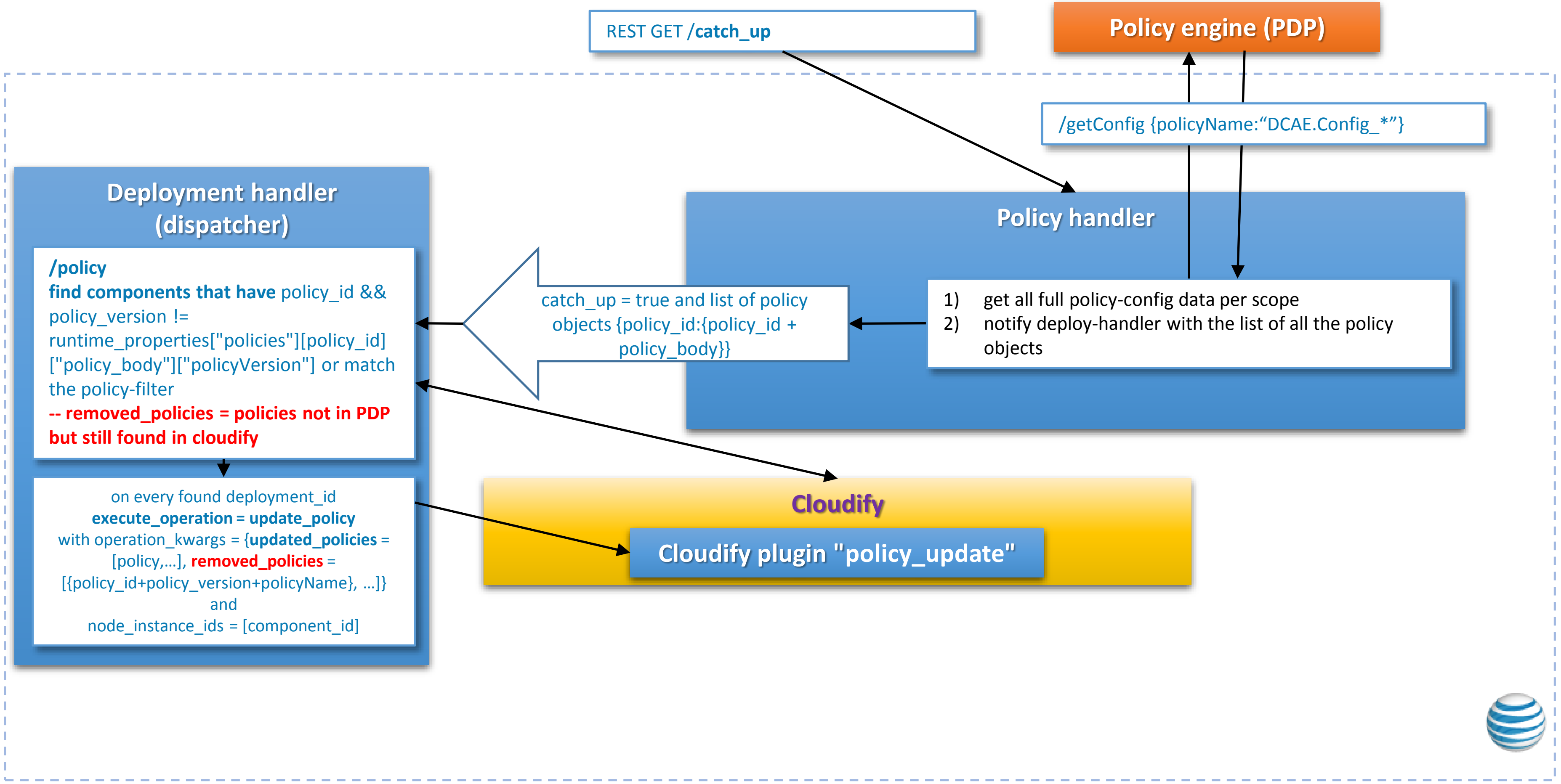
1. **Trigger**: **docker_config**["**policy**"]["**trigger_type**"] == "**docker**" in the blueprint of the component
2. **Path** to **script** in the blueprint of the component at **docker_config**["**policy**"]["**script_path**"]
3. In this case, the component needs to have the reconfigure-policy-update script inside Docker container at the path specified in the blueprint under **docker_config**["**policy**"]["**script_path**"].
4. Script args
   a) $1=<reconfigure-type> ("**policies**") – new type of "policies" instead of older "policy" to indicate the change of API
   b) $2=<updated_policies_message> - json
5. Cloudify **plugin** on policy_updated event on each component (node instance) will
   a) policy-lib will store the list of policy_body objects into consul-kv under key of <service_component_name>:policies
   b) invoke **/bin/sh <script_path> "policies" "{\"updated_policies\":[{…}], \"removed_policies\":[{…}], \"policies\":[{…}]}"** inside the docker container to notify the component about the policy-update
   c) the plugin will **no longer** send the application_config to the script – instead it will send the full collection of **policies**.
   d) Each of collections **updated_policies, removed_policies**, and **policies** is now a **list of policy_body objects**, rather than the list policy_body["config"] objects
6. It is now **the responsibility of the component to retrieve the latest application config** from the config-binding service (**CBS**)
7. It is **up to script what to do with the event** – either raise (kill) the signal to the app, or call IPC on the app or call REST API on the app or anything else the app and script developer can come up with.

This way the DCAE-Controller platform does not need to know anything about the application logic to apply the policy change and each application will have the full control and easy testing of how to react to reconfigure (policy-update) event.

Sample /opt/app/reconfigure.sh inside the Docker container to process the policy-update notification

```
/opt/app/reconfigure.sh
--------------------------------------------------------

#!/bin/bash

## /opt/app/reconfigure.sh “policies” "{\“updated_policies\”:[{…}], \"removed_policies\":[{…}], \"policies\":[…]}"

MSG_TYPE=$1
MSG=$2

DEMO_DIR=/opt/app/demo_dir
RUNSCRIPT=/opt/app/reconfigure.sh
DATAFILE=$(date +%Y_%m%d-%H%M%S)_${MSG_TYPE}.json

cd ${DEMO_DIR}
LOG_FILE=demo.log

echo "==================" ${LOG_FILE} | tee -a ${LOG_FILE}
(date && whoami && hostname -f && pwd) | tee -a ${LOG_FILE}

echo "running script" ${RUNSCRIPT} "with params($@)" | tee -a ${LOG_FILE}

# save the message into datafile named as <datetime>_<msg_type>.json
printf "${MSG}\n" >> ${DATAFILE}

ls -la | tee -a ${LOG_FILE}
echo ${DATAFILE} | tee -a ${LOG_FILE}
cat ${DATAFILE} | tee -a ${LOG_FILE}
```

DCAE controller – recovery after downtime – send REST GET /catch_up to policy-handler or restart the policy-handler



REST GET **/catch_up**
Policy engine (PDP)
/getConfig {policyName:“DCAE.Config_*”}
Deployment handler (dispatcher)
Policy handler
Cloudify
Cloudify plugin "policy_update"

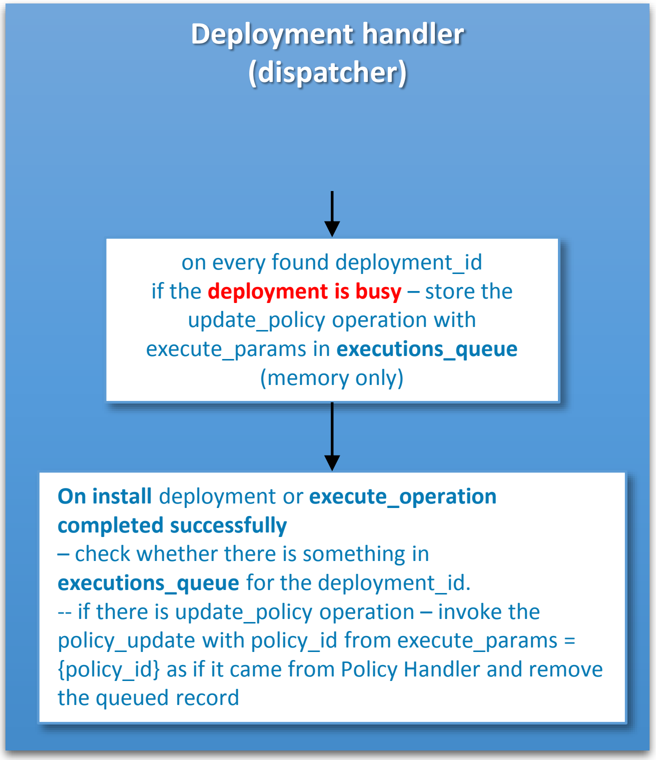
/policy
find components that have policy_id && policy_version != runtime_properties["policies"][policy_id]["policy_body"]["policyVersion"] or match the policy-filter
-- removed_policies = policies not in PDP but still found in cloudify

on every found deployment_id
execute_operation = update_policy
with operation_kwargs = {**updated_policies** = [policy,…], **removed_policies** = [{policy_id+policy_version+policyName}, …]}
and
node_instance_ids = [component_id]

catch_up = true and list of policy objects {policy_id:{policy_id + policy_body}}

1) get all full policy-config data per scope
2) notify deploy-handler with the list of all the policy objects

DCAE controller – how the policy update **queued**



Deployment handler (dispatcher)

on every found deployment_id
if the **deployment is busy** – store the update_policy operation with execute_params in **executions_queue** (memory only)

On install deployment or **execute_operation completed successfully**
– check whether there is something in **executions_queue** for the deployment_id.
-- if there is update_policy operation – invoke the policy_update with policy_id from execute_params = {policy_id} as if it came from Policy Handler and remove the queued record

Links related to policy in DCAE controller

- **policy-handler:**
  - ✓ https://gerrit.onap.org/r/#/admin/projects/dcaegen2/platform/policy-handler
  - ✓ https://gerrit.onap.org/r/gitweb?p=dcaegen2/platform/policy-handler.git;a=summary
- **policy-handler** installation **blueprint:**
  - ✓ https://gerrit.onap.org/r/#/admin/projects/dcaegen2/platform/blueprints
  - ✓ https://gerrit.onap.org/r/gitweb?p=dcaegen2/platform/blueprints.git;a=tree;f=blueprints;hb=HEAD
- **deployment-handler** with policy-update api:
  - ✓ https://gerrit.onap.org/r/#/admin/projects/dcaegen2/platform/deployment-handler
  - ✓ https://gerrit.onap.org/r/gitweb?p=dcaegen2/platform/deployment-handler.git;a=summary
  - ✓ swagger UI http://<deployment-handler>/swagger-ui/#
- **dcaepolicyplugin plugin** and **dcae.nodes.policy (*.policies)** node types
  - ✓ https://gerrit.onap.org/r/#/admin/projects/dcaegen2/platform/plugins
  - ✓ https://gerrit.onap.org/r/gitweb?p=dcaegen2/platform/plugins.git;a=tree;f=dcae-policy;hb=HEAD
- **onap-dcae-dcaepolicy-lib** -- consumed by dockerplugin in cloudify of DCAE-C:
  - ✓ https://gerrit.onap.org/r/#/admin/projects/dcae/utils
  - ✓ https://gerrit.onap.org/r/gitweb?p=dcaegen2/utils.git;a=tree;f=python-dcae-policy;hb=HEAD
  - ❖ https://pypi.python.org/pypi/onap-dcae-dcaepolicy-lib -- **obsolete**
- **policy-engine** wiki and API
  - ✓ https://wiki.onap.org/display/DW/Policy
  - ✓ https://wiki.onap.org/display/DW/Policy+API
- ONAP sonar reports
  - ✓ https://sonar.onap.org/projects?search=dcaegen2&sort=-analysis_date