This document contains the proposal to push remoteIp configuration from CNI (openshift) and AIM (openstack) in normalized structure to ACI (apic).

**Why optimize remoteIp config pushed to apic?**

1. Significant reduction in hpp config pushed to APIC during pod/instance creation and deletion.
2. Significant reduction in processing all the Mos in APIC.

**Problem definition:**

A network policy inside a namespace (ns1) if refers another namespace (ns2), then all members of ns2 (let's say m) are added as remoteIp under ns1.

If n such namespaces refer ns2, n copies of remoteIp exists.

Total number of remoteIp created : n x m

If a new pod is deleted or created in ns2, total number of Mos sent to apic:

n x 2.  (n for remoteIp and n for tagAnnotation under each remoteIp)

So, any change in pods in ns2, config sent to APIC increase with O(n) where n is the number of namespaces.

Customers are using large namespace in production. Bosch has almost 450 namespaces configured in one of their clusters.

In Bosch, each namespace has 5 network policies as follows:

1. allow communication within namespace
2. allow communication from ingress namespace ---- this was a n-to-1 use case
3. allow communication from monitoring namespace - this is another n-to-1 use case
4. allow communication from router nodes - this is a workaround netwoprk poliy to the current issue
5. five deny all other communication - in general we do not need it when using ACI CNI

So, there are about 450 network policy referencing ingress namespace. This policy defines rules such as allow all tcp traffic from pods in ingress namespace. When a pod is deleted and created in ingress namespace, CNI sent 1800 MOs to apic.  900 Mos to be deleted and 900 to be created.

Due to this 1800 Mos, additional 450 resolved Mos are deleted and 450 are created, all related to security groups. All 450 were hostprotRemoteIp Mo. For each hostprotRemoteIp, a resolved mo hostprotRemoteIpDef is created.

Total Mos processed by APIC becomes: 1800+450+450 = 2700

hence,

total config Mos sent = 1800

total config Mos processed in APIC = 2700

After fixing, config sent and processed for the same operation will become:

total Mos sent = 4

total Mos processed in APIC = 6

Rough avg time taken in a 3 cluster apic to process 1 Mo (doer, replication, persistence) is: 5 ms

Time reduces from 13.5s to 30ms.

**How remoteIP resolution works from opflex-proxy to agents?**

Opflex proxy receives pol resolve request from agent for genie object GbpSubnets.

GbpSubnets contains the dn of the security group rule. From the rule dn, get the Rs relation mo , resolve the remoteIp container, and add the remoteIp

as children of ruleDef in doer thread and pass it along the comm thread as part of polResolve response.

Secgroup-uri:

/PolicyUniverse/PolicySpace/abcdef/GbpSecGroup/88c5e1ef-7745-4800-874d-de8f680612e4

Secgroup hierarchy:

1.GbpSecGroup

2.GbpSecGroupSubject

3.GbpSecGroupRule

GbpeSecGroupRule has relations to:

1.GbpeL24Classifier

2.GbpAllowDenyAction

3. GbpSubnets -----> this contains remoteIp configuration.

With respect to this change, opflex-agent does not need to do any changes and the representation of remoteIps in the messages between proxy and agent won’t change.

Model changes in APIC to be followed by CNI/AIM:

<mo name="RemoteIpContainer"

access="admin"

concrete="yes"

configure="yes"

label="Container"

/>

# RemoteIpContainer is singleton per namespace/hostprotpol

<rn mo="RemoteIpContainer">

<item prefix="remoteip" />

</rn>

<mo name="Namespace"

access="admin"

concrete="yes"

configure="yes"

label="Container"

/>

<mo name=”hostprot:RemoteIp”>

<rel-def name="remoteIpCont"

to="RemoteIpContainer"

cardinality="n-to-1"

owner="management"

enforceable="true"

resolvable="true"

update-type="subtree"

update-modified-only="yes"

configure="yes"

chunk-owner="opflexp"

label="remoteIp container relation"/>

</mo>

<contains parent="Namespace"

child="RemoteIpContainer"

/>

<contains parent="hostprot:Pol"

child="RemoteIpContainer"

/>

<contains parent=" RemoteIpContainer"

child="hostprot:remoteIp"

/>

Diagram

Description automatically generated

Figure 1: Hpp relationship with remoteIp container Mo

**PM changes:**

1. **Distribute remotipcontainer :**PolDef resides in same shard as pol. If multiple polDefs from a tenant needs to be downloaded to multiple switch, switches will download it sequentially. PolDefs are distributed to different shards to allow switches to parallely download polDefs.
2. **Distribute remoteIpContainer:**   
   RemoteIpContainer will be distributed to different shards for same reasoning as polDef, to allow for parallel download to switches.

The relation from polDef to remoteIpContainer will be unresolvable and unenforceable. This saves from unnecessary polUpdates to polDef shard from containerShard. PolDef shard doesn’t need the contents of remoteIpContainer, since conciliation of all remoteIp for a rule is done at leaf. So leaf keeps one copy of each remoteIpContainer as needed.

ODev has a relation to polDef to download polDef. Using ruleDefs under polDefs, ODev will know about the remoteIpContainers it needs. A new relation is created from ODev to remoteIpContainerMo to download remoteIpContainerMo to leaf. Leaf will have polDefs and ruleDefs and corresponding remoteIpContainer. Leaf has sufficient information to collect all remoteIp for a rule and send them to agent.

**PodSelector in Kubernetes:**

Network policy in Kubernetes can also contains podSelector along with namespace selectors. CNI will take care of Namespace selectors. (CNI will create RsHostprotRemoteIpContainer for each namespace from namespace selector.)

There can be two types of podSelectors:

|  |  |
| --- | --- |
| 1. matchExpressions [LabelSelectorRequirement](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.22/#labelselectorrequirement-v1-meta) array | matchExpressions is a list of label selector requirements. The requirements are ANDed. |
| 1. matchLabels object | matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions, whose key field is "key", the operator is "In", and the values array contains only "value". The requirements are ANDed. |

MatchExpression is an array of LabelSelectorRequirement:

|  |  |
| --- | --- |
| key *string* | key is the label key that the selector applies to. |
| operator *string* | operator represents a key's relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist. |
| values *string array* | values is an array of string values. If the operator is **In or NotIn**, the values array must be non-empty. If the operator is **Exists or DoesNotExist**, the values array must be empty. This array is replaced during a strategic merge patch. |

MatchLabels is a specials case of matchExpression, hence can be expressed as a matchExpression of following type:

matchLabels:

component: redis

is equivalent to

matchExpressions:

- {key: component, operator: In, values: [redis]}

.

There is a new mo podFilter to capture these filters:  
<mo name="PodFilter"   
 <property name="key" type="string:Basic" />

<property name="operator" type=" hostprot:Operator " />

<property name="values" type=" string:Basic" />

</mo>

Each remoteIp Mo under RemoteIpContainer Mo needs to contain its labels. Labels are represented using this Mo:

<mo name="EpLabel">

<property name="key" type="string:Basic" />

<property name="value" type="string:Basic" />

</mo>