**ACP AWS self managed k8s N/W Expansion - POC**

**Background:**

When the k8s clusters were built in the past, it was designed only to accommodate 254 worker nodes in the k8s cluster. Now that certain Aruba cluster has grown big in scale and they want to support 800k devices, we have reached to a point where we could not horizontally scale the cluster neither vertically scale the kubernetes cluster.

**Objectives:**

Unblock the BU to grow beyond 255 nodes quickly.

**Approaches:**

1)Split the flannel network SubnetLen in place such that we could accommodate more than 254 nodes in a given k8s cluster.

2)Move from flannel to some other CNI solutions like Calico/Cilium.

**Decision and Rational:**

We decided to stick with the Approach 1 of splitting the flannel network SubnetLen in place as a fast track approach, as it was time critical.

**Expectation and Alignment from Aruba on changes coming:**

1)This NW split change will give buffer to go up till ~500 nodes. We cannot go beyond this with current plugins and network cidr subnet cidr restrictions.

2)Long term - We should explore other CNI plugins like Calico/Cilium. Or We should consider moving to CCP world where you will get managed EKS solution.

3)This network change needs to be done during a down time. Downtime Window ~4hours

4)We are moving away from aws-vpc as flannel backend to vxlan, something which is in use in azure as well.

a)it would take away the routes from the aws layer and will be handled within the k8s layer. As the k8s route tables has limitation on number of routes that can be supported per route table.

b)It would eliminate frequent black holes created on the aws routes as well.

5)The podCidr IP range that gets allocated will be different now after this network split.

a)The nodes will operate in /25 subnet range now. The node podCidr will look something like below

<node1> 10.2.0.0/25

<node2> 10.2.0.128/25

<node3> 10.2.1.0/25

<node4> 10.2.1.128/25

b)Earlier the nodes used to operate in /24 network range and the node podCidr used to loou jenk something like below

<node1> 10.2.0.0/24

<node2> 10.2.1.0/24

<node3> 10.2.2.0/24

6)We would need Aruba to validate the changes/network performance from the application point of view.

**Consolidated Steps based on the POC:**

**1)** Prep up the cluster with all the required configs tweaks

**2) Action:**

update node-cidr-mask-size in kube-contoller-manager

**Steps:**

#login to each master node:

#add node-cidr-mask-size key in /etc/kubernetes/manifests/kube-controller-manager.yaml file

**example:**

ubuntu@ip-10-155-100-90:~$ sudo grep node-cidr-mask-size -A1 -B1 /etc/kubernetes/manifests/kube-controller-manager.yaml

- --cluster-cidr=10.2.0.0/16

- --node-cidr-mask-size=25

- --cluster-name=kubernetes

**3**)**Action:**

update the flannel config map with

SubnetLen as 25

backend as vxlan

**Steps:**

jenkins@ip-10-155-130-156:/home/ubuntu$ kubectl get cm -A|grep flannelkube-system kube-flannel-cfg 2 8d

#take backup of current flannel config

#edit the config map make the below change and apply

from:

net-conf.json: |

{

"Network": "10.2.0.0/16",

"Backend": {

"Type": "aws-vpc"

}

}

To:

net-conf.json: |

{

"Network": "10.2.0.0/16",

"SubnetLen": 25,

"Backend": {

"Type": "vxlan"

}

}

**Comments:**

Here we are changing the SubnetLen from 24 to 25. which would allow us to scale upto ~500 nodes.

Also, we are changing from aws-vpc to vxlan, to eliminate dependency on cloud specific backend like ‚Äúaws-vpc‚Äù.

also, it would take away the routes from the aws layer and will be handled within the k8s layer. As the k8s route tables has limitation on number of routes that can be supported per route table.

It would eliminate frequent black holes created on the aws routes.

**4)**

**Action:**

update the etcd /coreos.com/network/config key with new SubnetLen and backend

**Steps:**

#get the current value

ubuntu@ip-10-155-100-48:~$ etcdctl --endpoints=http://10.155.100.48:2380 get /coreos.com/network/config

/coreos.com/network/config

{"Network" : "10.2.0.0/16", "Backend" : {"Type" : "aws-vpc"}}

ubuntu@ip-10-155-100-48:~$

#update the key

etcdctl --endpoints=http://10.155.102.115:2379 put /coreos.com/network/config '{"Network" : "10.2.0.0/16", "SubnetLen": 25, "Backend" : {"Type" : "vxlan"}}'

#verify the key

ubuntu@ip-10-155-100-48:~$ etcdctl --endpoints=http://10.155.102.115:2379 get /coreos.com/network/config

/coreos.com/network/config

{"Network" : "10.2.0.0/16", "SubnetLen": 25, "Backend" : {"Type" : "vxlan"}}

**5)**#Following steps will make sure the changes takes effect

**6)**

**Action:**

Scale down all the ASGs

**Steps:**

scale down all the asg from console

**7)**

Step #7 to #9 needs to be done each master node in a rolling fashion

**8)**

**Action:**

delete master node from the cluster in rolling fashion

**Steps:**

kubectl delete node <master node 1>

**9)**

**Action:**

login to each master node

**Steps:**

sudo rm /var/run/flannel/subnet.env

sudo systemctl stop kubelet

sudo systemctl stop docker

sudo systemctl start docker

sudo systemctl start kubelet

**10)**

**Action:**

Add required label and taints

**Steps:**

kubectl taint node <master node ip> node-role.kubernetes.io/master:NoSchedule

kubectl label node <master node ip> node-role.kubernetes.io/control-plane=""

kubectl label node <master node ip> node-role.kubernetes.io/master=""

**11)**

**Action:**

Scale up all the ASGs

**Steps:**

scale up all the asg from console

**Comments:**

This will make sure the pods and nodes gets new network range and are consistent. This will be a downtime activity.

**12)**

**Action:**

Check if each nodes podCidr is changed to /25 range

**Steps:**

#each nodes podCidr should be changed from /24 to /25 range

**example:**

jenkins@ip-10-155-130-156:/home/ubuntu$ kubectl get node -o jsonpath="{range .items[\*]}{.metadata.name} {.spec.podCIDR}{'\n'}{end}" | column -t

ip-10-155-100-90.us-west-2.compute.internal 10.2.0.0/25

ip-10-155-101-114.us-west-2.compute.internal 10.2.3.128/25

ip-10-155-101-91.us-west-2.compute.internal 10.2.0.128/25

ip-10-155-102-77.us-west-2.compute.internal 10.2.1.0/25

ip-10-155-102-94.us-west-2.compute.internal 10.2.3.0/25

**Test Validation:**

Test cluster used: k8snwpoc

**1)**

**TC Summary:**

Check if each nodes podCidr is changed to /25 range

**TC Steps:**

kubectl get node -o jsonpath="{range .items[\*]}{.metadata.name} {.spec.podCIDR}{'\n'}{end}" | column -t

**Expected Result:**

each nodes podCidr should be changed from /24 to /25 range

jenkins@ip-10-155-130-156:/home/ubuntu$ kubectl get node -o jsonpath="{range .items[\*]}{.metadata.name} {.spec.podCIDR}{'\n'}{end}" | column -t

ip-10-155-100-90.us-west-2.compute.internal 10.2.0.0/25

ip-10-155-101-114.us-west-2.compute.internal 10.2.3.128/25

ip-10-155-101-91.us-west-2.compute.internal 10.2.0.128/25

ip-10-155-102-77.us-west-2.compute.internal 10.2.1.0/25

ip-10-155-102-94.us-west-2.compute.internal 10.2.3.0/25

**2)**

**TC Summary:**

Check the N/W connectivity between two pods with in a same node

**TC Steps:**

#get two pods from same node

kubectl get pods -o wide -A|grep sample|grep ip-10-155-101-114.us-west-2.compute.internal|head -2

#try to reach podA from podB and viceversa

kubectl exec -it -n default sample-deployment-7959c588c5-4hhhp -- curl http://10.2.3.157 |grep "Thank you"

**Expected Result:**

jenkins@ip-10-155-130-156:/home/ubuntu$ kubectl get pods -o wide -A|grep sample|grep ip-10-155-101-114.us-west-2.compute.internal|head -2

default sample-deployment-7959c588c5-2t259 1/1 Running 0 14h 10.2.3.157 ip-10-155-101-114.us-west-2.compute.internal <none> <none>

default sample-deployment-7959c588c5-4hhhp 1/1 Running 0 14h 10.2.3.134 ip-10-155-101-114.us-west-2.compute.internal <none> <none>

jenkins@ip-10-155-130-156:/home/ubuntu$ kubectl exec -it -n default sample-deployment-7959c588c5-4hhhp -- curl http://10.2.3.157 |grep "Thank you"

<p><em>Thank you for using nginx.</em></p>

jenkins@ip-10-155-130-156:/home/ubuntu$ kubectl exec -it -n default sample-deployment-7959c588c5-2t259 -- curl http://10.2.3.134 |grep "Thank you"

<p><em>Thank you for using nginx.</em></p>

jenkins@ip-10-155-130-156:/home/ubuntu$

**3)**

**TC Summary:**

Check the N/W connectivity between two pods in two different nodes

**TC Steps:**

#get two pods from two different nodes

kubectl get pods -o wide -A|grep sample|egrep "ip-10-155-102-94.us-west-2.compute.internal|ip-10-155-101-114.us-west-2.compute.internal"|head -2

#try to reach from podA in node1 to podB in node2 and viceversa

kubectl exec -it -n default sample-deployment-7959c588c5-2gs96 -- curl http://10.2.3.157 |grep "Thank you"

**Expected Result**:

jenkins@ip-10-155-130-156:/home/ubuntu$ kubectl get pods -o wide -A|grep sample|egrep "ip-10-155-102-94.us-west-2.compute.internal|ip-10-155-101-114.us-west-2.compute.internal"|head -2

default sample-deployment-7959c588c5-2gs96 1/1 Running 0 14h 10.2.3.13 ip-10-155-102-94.us-west-2.compute.internal <none> <none>

default sample-deployment-7959c588c5-2t259 1/1 Running 0 14h 10.2.3.157 ip-10-155-101-114.us-west-2.compute.internal <none> <none>

jenkins@ip-10-155-130-156:/home/ubuntu$ kubectl exec -it -n default sample-deployment-7959c588c5-2gs96 -- curl http://10.2.3.157 |grep "Thank you"

<p><em>Thank you for using nginx.</em></p>

jenkins@ip-10-155-130-156:/home/ubuntu$ kubectl exec -it -n default sample-deployment-7959c588c5-2t259 -- curl http://10.2.3.13 |grep "Thank you"

<p><em>Thank you for using nginx.</em></p>

jenkins@ip-10-155-130-156:/home/ubuntu$

**4)**

**TC Summary:**

Check the N/W connectivity between a node and a pod

**Tc Steps:**

#login to the node and try reaching any pod IP

ubuntu@ip-10-155-102-94:~$ ping 10.2.3.157

**Expected Result:**

ubuntu@ip-10-155-102-94:~$ ping 10.2.3.157

PING 10.2.3.157 (10.2.3.157): 56 data bytes

64 bytes from 10.2.3.157: icmp\_seq=0 ttl=63 time=1.105 ms

64 bytes from 10.2.3.157: icmp\_seq=1 ttl=63 time=1.052 ms

**5)**

**TC Summary:**

Expand one node in the asg and validate it joins the cluster and get the podCidr in /25 range

**TC Steps:**

kubectl describe nodes ip-10-155-100-140.us-west-2.compute.internal|grep -i podcidr:

**Expected Result:**

jenkins@ip-10-155-130-156:/home/ubuntu$ kubectl describe nodes ip-10-155-100-140.us-west-2.compute.internal|grep -i podcidr:

PodCIDR: 10.2.1.128/25

**6)**

**TC Summary:**

validate no routes are getting added in the route table in aws as we moved from aws-vpc to vxlan backend

**TC Steps:**

go to aws console ‚Üí pick any one worker node ‚Üí check its subnet route table

**Expected Result**:

Need to refer PIC in the page

**7)**

**TC Summary:**

Check if can scale beyond 256 nodes

**TC Steps:**

Increase the asg max node count beyond 256 and check if all the nodes are joining the cluster.

kubectl get nodes |wc -l

**Expected Result:**

jenkins@ip-10-155-130-156:/home/ubuntu$ kubectl get nodes|grep -v NAME|wc -l

510

Capturing PodCidr range of all nodes for reference

Get Nodes stats

**8)**

**TC Summary:**

network performance between aws-vpc and vxlan

**TC Steps:**

#setup iperf server and iperf client on two different pods

#create a server pod

kubectl run iperf-server --restart='Never' --image networkstatic/iperf3 --namespace default --command -- sleep infinity

kubectl exec -it iperf-server bash

root@iperf-server:/# iperf3 -s

-----------------------------------------------------------

Server listening on 5201

-----------------------------------------------------------

#initiate a connection from client

kubectl run iperf-client --restart='Never' --image networkstatic/iperf3 --namespace default --command -- sleep infinity

#get the pod ip of the iperf server created in the last step

root@iperf-client:/# iperf3 -c 10.2.31.19

**Expected Result:**

on **nirtest256** - with aws-vpc backend

root@iperf-client:/# iperf3 -c 10.2.23.58

Connecting to host 10.2.23.58, port 5201

[ 5] local 10.2.18.20 port 48212 connected to 10.2.23.58 port 5201

[ ID] Interval Transfer Bitrate Retr Cwnd

[ 5] 0.00-1.00 sec 601 MBytes 5.04 Gbits/sec 14 3.90 MBytes

[ 5] 1.00-2.00 sec 592 MBytes 4.97 Gbits/sec 7 3.56 MBytes

[ 5] 2.00-3.00 sec 591 MBytes 4.96 Gbits/sec 7 3.29 MBytes

[ 5] 3.00-4.00 sec 592 MBytes 4.97 Gbits/sec 7 3.02 MBytes

[ 5] 4.00-5.00 sec 592 MBytes 4.97 Gbits/sec 7 2.71 MBytes

[ 5] 5.00-6.00 sec 592 MBytes 4.97 Gbits/sec 0 3.58 MBytes

[ 5] 6.00-7.00 sec 592 MBytes 4.97 Gbits/sec 7 3.29 MBytes

[ 5] 7.00-8.00 sec 591 MBytes 4.96 Gbits/sec 7 3.05 MBytes

[ 5] 8.00-9.00 sec 592 MBytes 4.97 Gbits/sec 7 2.74 MBytes

[ 5] 9.00-10.00 sec 592 MBytes 4.97 Gbits/sec 0 3.58 MBytes

- - - - - - - - - - - - - - - - - - - - - - - - -

[ ID] Interval Transfer Bitrate Retr

[ 5] 0.00-10.00 sec 5.79 GBytes 4.98 Gbits/sec 63 sender

[ 5] 0.00-10.00 sec 5.78 GBytes 4.96 Gbits/sec receiver

iperf Done.

**on k8snwpoc** - with vxlan backend

root@iperf-client:/# iperf3 -c 10.2.31.19

Connecting to host 10.2.31.19, port 5201

[ 5] local 10.2.31.146 port 34940 connected to 10.2.31.19 port 5201

[ ID] Interval Transfer Bitrate Retr Cwnd

[ 5] 0.00-1.00 sec 596 MBytes 5.00 Gbits/sec 14 3.87 MBytes

[ 5] 1.00-2.00 sec 589 MBytes 4.94 Gbits/sec 7 3.53 MBytes

[ 5] 2.00-3.00 sec 589 MBytes 4.94 Gbits/sec 7 3.24 MBytes

[ 5] 3.00-4.00 sec 590 MBytes 4.95 Gbits/sec 7 2.89 MBytes

[ 5] 4.00-5.00 sec 589 MBytes 4.94 Gbits/sec 0 3.70 MBytes

[ 5] 5.00-6.00 sec 589 MBytes 4.94 Gbits/sec 7 3.41 MBytes

[ 5] 6.00-7.00 sec 589 MBytes 4.94 Gbits/sec 7 3.07 MBytes

[ 5] 7.00-8.00 sec 589 MBytes 4.94 Gbits/sec 7 2.70 MBytes

[ 5] 8.00-9.00 sec 589 MBytes 4.94 Gbits/sec 0 3.55 MBytes

[ 5] 9.00-10.00 sec 589 MBytes 4.94 Gbits/sec 7 3.23 MBytes

- - - - - - - - - - - - - - - - - - - - - - - - -

[ ID] Interval Transfer Bitrate Retr

[ 5] 0.00-10.00 sec 5.76 GBytes 4.95 Gbits/sec 63 sender

[ 5] 0.00-10.00 sec 5.74 GBytes 4.93 Gbits/sec receiver

iperf Done.

**iperf3 test using the svc ip**

root@iperf-client:/# iperf3 -c 10.3.7.154

Connecting to host 10.3.7.154, port 5201

[ 5] local 10.2.31.146 port 55206 connected to 10.3.7.154 port 5201

[ ID] Interval Transfer Bitrate Retr Cwnd

[ 5] 0.00-1.00 sec 596 MBytes 5.00 Gbits/sec 14 3.87 MBytes

[ 5] 1.00-2.00 sec 589 MBytes 4.94 Gbits/sec 7 3.51 MBytes

[ 5] 2.00-3.00 sec 589 MBytes 4.94 Gbits/sec 7 3.17 MBytes

[ 5] 3.00-4.00 sec 590 MBytes 4.95 Gbits/sec 7 2.86 MBytes

[ 5] 4.00-5.00 sec 589 MBytes 4.94 Gbits/sec 0 3.67 MBytes

[ 5] 5.00-6.00 sec 589 MBytes 4.94 Gbits/sec 7 3.36 MBytes

[ 5] 6.00-7.00 sec 589 MBytes 4.94 Gbits/sec 7 3.01 MBytes

[ 5] 7.00-8.00 sec 589 MBytes 4.94 Gbits/sec 0 3.79 MBytes

[ 5] 8.00-9.00 sec 589 MBytes 4.94 Gbits/sec 7 3.48 MBytes

[ 5] 9.00-10.00 sec 589 MBytes 4.94 Gbits/sec 7 3.17 MBytes

- - - - - - - - - - - - - - - - - - - - - - - - -

[ ID] Interval Transfer Bitrate Retr

[ 5] 0.00-10.00 sec 5.76 GBytes 4.95 Gbits/sec 63 sender

[ 5] 0.00-10.00 sec 5.74 GBytes 4.93 Gbits/sec receiver

iperf Done.