Aarna Networks Multi-Cluster Orchestration Platform (AMCOP)

User Guide

Product Version: 3.1.09232022

TABLE OF CONTENTS

INTRODUCTION

This document explains Aarna Networks' Multicluster Orchestration and Automation Platform (AMCOP) user operations to orchestrate and manage Cloud-native functions and applications (CNFs or CNAs). It does not cover administration of AMCOP, such as deployment and upgrades, which are documented in the AMCOP Quickstart Guide.

AMCOP deployment can be done on a single server (all in one), a single VM, on a cloud (GKE, AKS etc.) or multiple servers/VMs. This Quickstart guide covers installation on all the supported configurations.

The configuration in case of a single server (all-in-one) installation looks as follows:

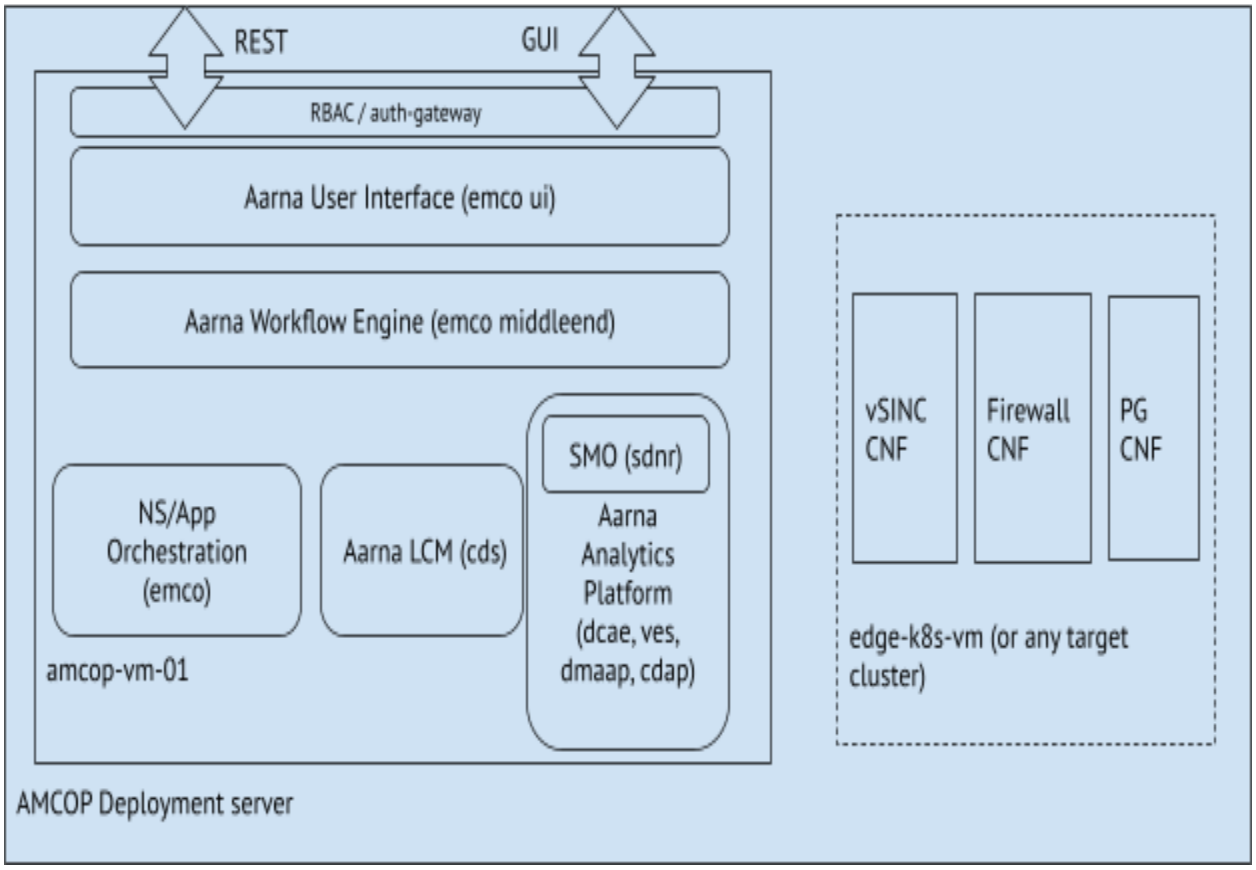


Figure 1: Single Server deployment with KuD cluster

This document uses the following color coding for the commands to be executed by the user.

Jump host refers to the server from which the installation of an AMCOP cluster is done. The VM amcop-vm-01 is the virtual machine where AMCOP is deployed.

Commands in blue font are for AMCOP Deployment server where installation is done, or Install Jump host from where installation is initiated. .
Commands in green courier font are for any AMCOP VMs (amcop-vm-XX)
Commands in orange courier font are for your laptop

Configuring Desktop/Laptop to access AMCOP portal

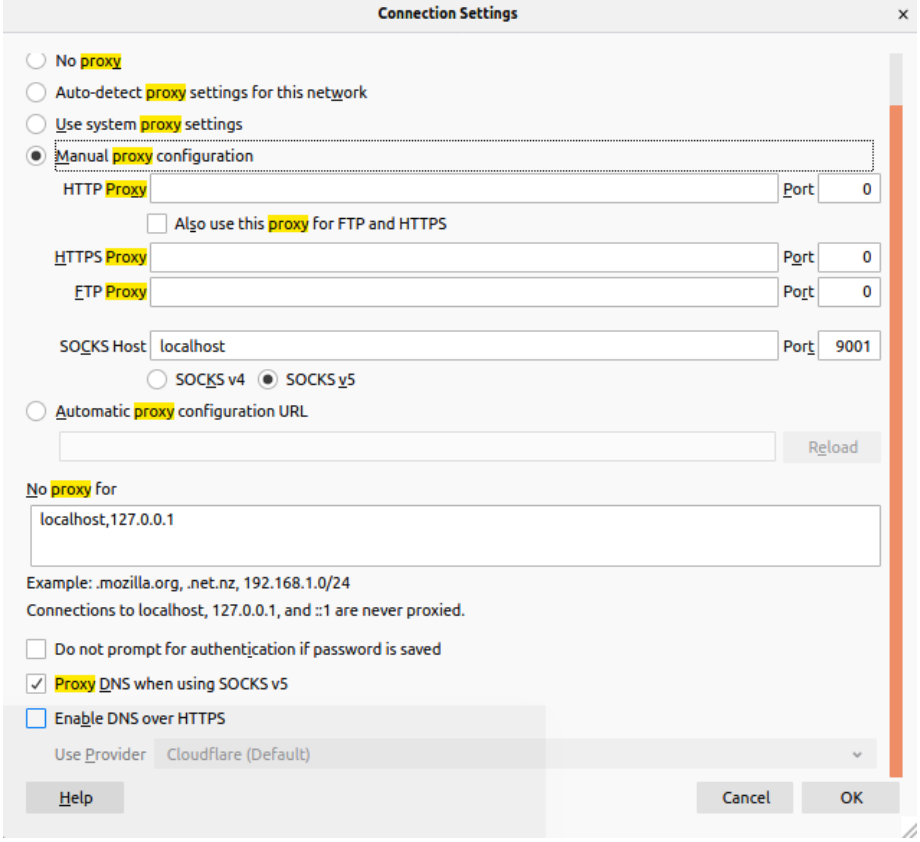
Following are the steps to access the AMCOP portal from your laptop/desktop.

Browser Settings

- Install the Firefox browser to access the AMCOP portal.
- Change your Firefox browser setting by typing about:config in the URL bar (accept the risk to proceed and search for security.mixed_content.block_active_content attribute).

  Allow mixed contents (http & https) security.mixed_content.block_active_content = false (Double click to change the value)

- Enable FireFox SSH socks proxy settings and enable DNS lookup through socks tunnel port 5000. You can skip this step, if you have direct access to amcop-master VM (or the server where AMCOP is installed). Please refer to putty or SSH commands to setup socks tunnel.



Set up SOCKS tunnel

You need to set up a socks tunnel to access the endpoints of AMCOP GUI over the Firefox browser.

The following commands will depend on how your laptop is connected to the server where AMCOP is installed. If it is connected over multiple hops (e.g., a VPN server, followed by a Jump server), you can use the SSH command to connect to the Jump server.

If the AMCOP Jump host (where AMCOP is installed) is accessible from another VPN server and the VPN server is accessible from a localhost or laptop, the following command can be used:

Setup 1: Laptop or Localhost → <VPN server> → <AMCOP Jump host>

ssh -i <ssh_private_key_of_the_Jump_host> -L 5000:localhost:5000 <username_of_the_Jump_host>@<ip_address_of_the_Jump_host> ssh -o CheckHostIP=no -o StrictHostKeyChecking=no -D 5000 <username_of_AMCOP_host>@<IP_address_of_AMCOP_host>

If the AMCOP Jump host (where AMCOP is installed) is directly accessible from Localhost/Server, the following command can be used:

Setup 2: Localhost/Server → <AMCOP Jump host>

```
ssh -i <ssh_private_key_of_the_local_host> -D localhost:5000
<username_of_AMCOP_host>@<IP_address_of_AMCOP_host>
```

Steps to access AMCOP Orchestration Portal

You can follow these steps to access AMCOP Orchestration portal, which can be used to design and deploy composite applications, and perform Day-0/Day-N configuration.

- Find the IP address of AMCOP VM using the following command.

  ```
  # log in to the AMCOP Jump host and execute the following
  sudo virsh domifaddr amcop-vm-01

  # The output will look similar to the below output

       Name       MAC address          Protocol     Address
  -------------------------------------------------------------------------------
       vnet0      52:54:00:18:be:d2    ipv4         192.168.122.74/24
  ```

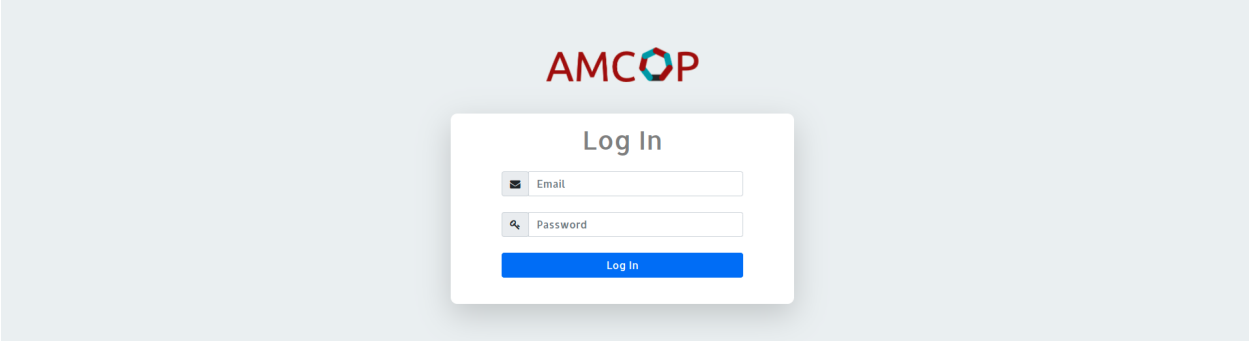
- Login to AMCOP VM (amcop-vm-01) with IP obtained from the above output and execute the below commands to verify if ONAP k8s services are fully functional.

  ```
  ssh ubuntu@<amcop-vm-01 IP address>

  # Example command:
  # ssh ubuntu@192.168.122.74

  kubectl get svc -n amcop-system -o wide

  # The output of this will look like the below.
  NAME   TYPE     CLUSTER-IP     EXTERNAL-IP  PORT(S)    AGE   SELECTOR
  sbackend                  NodePort    10.102.140.127   <none>       5000:30661/TCP
  21h   app=sbackend
  ```

- Open portal URL from Firefox browser (on your laptop or local server) and type the AMCOP deployment VM IP and port number on which the service is exposed (e.g., 30661). Make sure the Firefox browser settings are done, and a tunnel is created and is running.

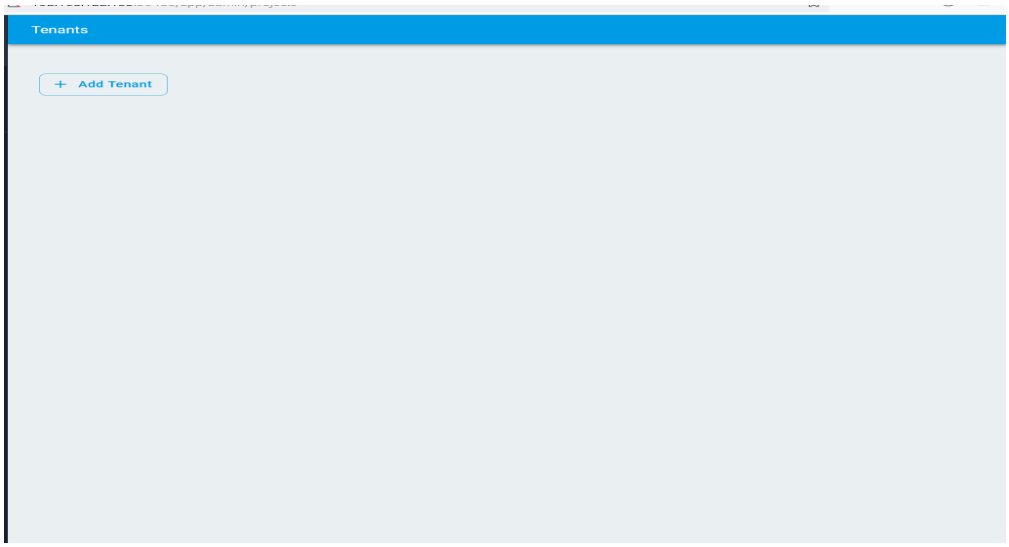
  For example, the URL will look like: 192.168.122.74:30661

The AMCOP UI will appear as below.



Login Credentials: admin@aarnanetworks.com/test

Note: If AMCOP is getting installed more than once, make sure the cache is cleared in the browser



Steps to access AMCOP SMO Portal

You can follow these steps to access AMCOP SMO portal directly, which can be used to perform FCAPS operations on O-RAN components.

> Note:
>
> You can perform the same set of operations using AMCOP SMO UI either through direct access as shown below or accessing it via AMCOP sign in process.

- Find the IP address of AMCOP VM using the following command.

```
# log in to the AMCOP Jump host and execute the following
sudo virsh domifaddr amcop-vm-01

# The output will look similar to the below output

 Name       MAC address          Protocol     Address
-------------------------------------------------------------------------------
 vnet0      52:54:00:18:be:d2    ipv4         192.168.122.74/24
```

- Login to AMCOP VM (amcop-vm-01) with IP obtained from the above output and execute the below commands to verify if ONAP k8s services are fully functional.

```
ssh ubuntu@<amcop-vm-01 IP address>

# Example command:
# ssh ubuntu@192.168.122.74

kubectl get svc -n amcop-system -o wide

# The output of this will look like the below.
NAME                                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                 AGE   SELECTOR
camunda                             NodePort    10.110.39.204   <none>        8443:32569/TCP          9h
app=camunda
cds-blueprints-processor-cluster    ClusterIP   10.108.201.207  <none>        5701/TCP                9h
app=cds-blueprints-processor
cds-blueprints-processor-grpc       ClusterIP   10.96.226.107   <none>        9111/TCP                9h
app=cds-blueprints-processor
cds-blueprints-processor-http       ClusterIP   10.99.252.162   <none>        8080/TCP                9h
app=cds-blueprints-processor
```

```
cds-db                    ClusterIP   None          <none>      3306/TCP                  9h    app=cds-db
cds-py-executor           ClusterIP   10.106.247.228   <none>      50052/TCP,50053/TCP       9h
app.kubernetes.io/name=cds-py-executor
cds-sdc-listener          ClusterIP   10.111.23.108    <none>      8080/TCP                  9h
app=cds-sdc-listener
cds-ui                    NodePort    10.110.27.127    <none>      3000:30497/TCP            9h
app=cds-ui
clm                       NodePort    10.98.65.231     <none>      9061:30461/TCP            9h    app=clm
configsvc                 NodePort    10.100.12.180    <none>      9082:30482/TCP            9h
app=configsvc
datafile-collector        NodePort    10.105.48.53     <none>      8443:31666/TCP,8100:30831/TCP   9h
app=datafile-collector
dcm                       NodePort    10.105.158.90    <none>      9078:30478/TCP,9077:30477/TCP   9h
app=dcm
dmaap                     NodePort    10.107.60.139    <none>      3904:32392/TCP,3905:30768/TCP   9h
io.kompose.service=dmaap
dtc                       NodePort    10.106.209.161   <none>      9048:30483/TCP,9018:30492/TCP   9h
app=dtc
emcoui                    ClusterIP   10.98.249.98     <none>      9080/TCP                  9h    app=emcoui
etcd                      ClusterIP   10.98.118.143    <none>      2379/TCP,2380/TCP         9h
app.kubernetes.io/instance=emco,app.kubernetes.io/name=etcd
etcd-headless             ClusterIP   None          <none>      2379/TCP,2380/TCP         9h
app.kubernetes.io/instance=emco,app.kubernetes.io/name=etcd
gac                       NodePort    10.107.108.117   <none>      9033:30493/TCP,9020:30491/TCP   9h
app=gac
kafka1                    ClusterIP   10.103.84.66     <none>      9092/TCP                  9h
io.kompose.service=kafka1
mariadb-galera            ClusterIP   None          <none>      3306/TCP                  9h
app=mariadb-galera
middleend                 ClusterIP   10.108.160.82    <none>      9051/TCP                  9h
app=middleend
mongo                     ClusterIP   None          <none>      27017/TCP                 9h    app=mongo
mongo-read                ClusterIP   10.105.31.250    <none>      27017/TCP                 9h
app=mongo
ncm                       NodePort    10.104.208.22    <none>      9082:30489/TCP,9081:30431/TCP   9h
app=ncm
orchestrator              NodePort    10.101.5.165     <none>      9016:30416/TCP,9015:30415/TCP   9h
app=orchestrator
ovnaction                 NodePort    10.97.37.86      <none>      9053:30473/TCP,9051:30471/TCP   9h
app=ovnaction
rsync                     NodePort    10.102.146.68    <none>      9031:30441/TCP            9h
app=rsync
sbackend                  NodePort    10.108.109.185   <none>      5000:30661/TCP            9h
app=sbackend
sdnr                      NodePort    10.107.239.69    <none>      8101:30101/TCP,8181:30181/TCP   9h
app=sdnr
sdnrdb                    ClusterIP   10.103.114.26    <none>      9200/TCP,9300/TCP         9h
app=sdnrdb
```

```
vescollector          NodePort    10.106.110.28   <none>        8080:31080/TCP          9h
app=vescollector
zookeeper             ClusterIP   10.100.131.197  <none>        2181/TCP                9h
io.kompose.service=zookeeper
```

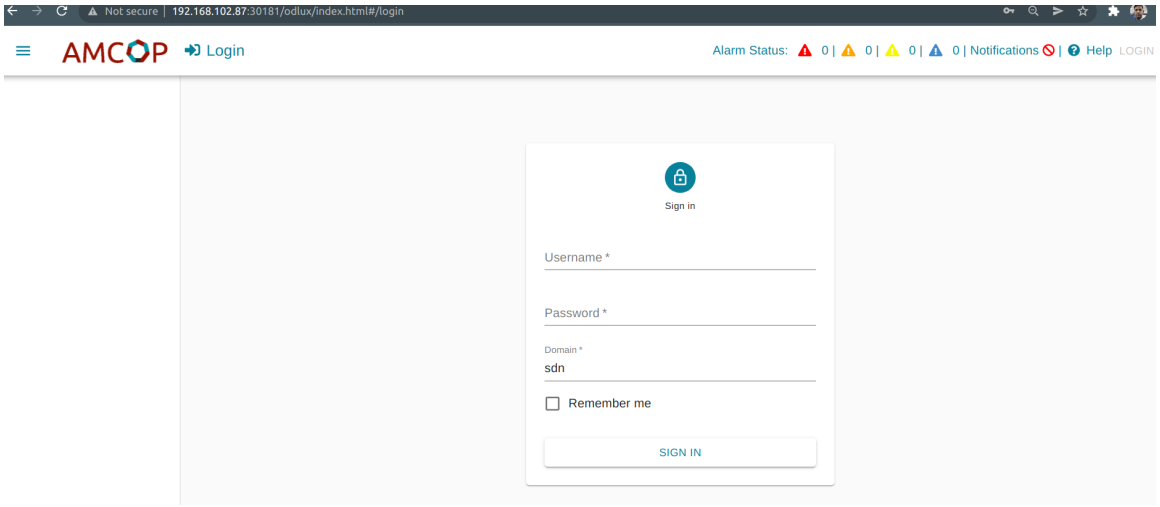
- Open portal URL from Firefox browser (on your laptop or local server) and type the AMCOP deployment VM IP and port number on which the service is exposed. Make sure the Firefox browser settings are updated, and a tunnel is created and is running.

  For example, the URL will look like: 192.168.122.74:30181/odlux/index.html

- GUI is accessible at http://<amcop-vm-ip>:30181/odlux/index.html
  - *username/password: admin/Kp8bJ4SXszM0WXlhak3eHlcse2gAw84vaoGGmJvUy2U*

- GUI is a visualization tool for the devices through which users can add the devices, check the connectivity status, verify the notifications/alerts, performance data and configuration management of the device.

  The AMCOP SMO UI will appear as below.



Note:

If AMCOP is getting installed more than once, make sure the cache is cleared in the browser.

CNF Orchestration

This section shows how CNFs can be orchestrated once AMCOP is deployed.

AMCOP uses ONAP project EMCO (Edge Multicluster Orchestrator) as the building block for CNF Orchestration functionality.

Create Target Kubernetes cluster on bare metal server

This is an optional step, in case you do not have a target k8s cluster to orchestrate CNF/CNAs.

- Create a target KuD cluster (if you do not have one) for instantiating CNFs. This can be done on a single Ubuntu server or a VM, or any existing k8s cluster can be used. If you do not have a k8s cluster, you can follow the instructions to create a single node KuD cluster. The minimum configuration requirement for creating the KuD based k8s cluster is as follows:

CPUs	8
Memory	32GB
Storage	150GB

```
# If you want to try AMCOP features using a simple k8s cluster
# and onboard sample CNFs (vFW), you can create the VM on the
# same server where you are running AMCOP.

cd /home/<user>/amcop_deploy/aarna-stream/util-scripts

# Below command will create Ubuntu 18.04 VM with 16 vCPUs, 32GB RAM
# and 50GB storage. You can change these parameters depending on the
# resources available on your system.

sudo ./create_qem_vm.sh 2 edge_k8s 50 16 32 ubuntu18.04 $HOME/.ssh/id_rsa.pub ubuntu

# Execute the below command to list the created VM.
sudo virsh list --all

# Execute the below command to list the IP address of the created VM
```

```
sudo virsh domifaddr edge_k8s

# log in to the server where k8s is set up and run the following
# commands.

# In case of Ubuntu 18.04 VM, <user> is "ubuntu"
ssh <user>@<ubuntu-server-ip>

# Execute below commands in the VM
sudo apt-get update -y
sudo apt-get upgrade -y
sudo apt-get install -y python-pip
Note: Make sure Python 3.x is the default python version on the server.

git clone https://gitlab.com/aarna-networks/k8s.git

Note: Reach out to Aarna support team to get access to the above repo

# Run script to setup KUD

nohup k8s/kud/hosting_providers/baremetal/aio.sh &

# You can monitor the progress by looking at nohup.out file

tail -f nohup.out

#Sample successful output looks like below.
PLAY RECAP *********************************************************************
localhost                  : ok=27   changed=17   unreachable=0    failed=0    skipped=0    rescued=0
ignored=0


Wednesday 18 November 2020  06:35:03 +0000 (0:00:00.072)       0:02:21.962 ****
===============================================================================
build CMK image ---------------------------------------------------- 75.57s
wait for all cmk daemonset pods to be running ---------------------- 48.21s
install cmk required packges --------------------------------------- 6.87s
clone CMK repository ----------------------------------------------- 1.96s
Run the script and re-evaluate the variable. ----------------------- 1.62s
install cmk required packges --------------------------------------- 1.13s
create a script to check CMK setup --------------------------------- 1.05s
prepare CMK CPU cores per config file ------------------------------ 0.62s
install CMK components --------------------------------------------- 0.57s
```

```
clean CMK directory ------------------------------------------------- 0.55s
customize CMK install yaml file per runtime env ------------------------ 0.53s
untaint nodes ------------------------------------------------------- 0.43s
tag CMK image ------------------------------------------------------- 0.42s
create CMK directory ------------------------------------------------ 0.30s
read current CMK version -------------------------------------------- 0.28s
generate CMK install yaml file -------------------------------------- 0.24s
prepare cmk check file ---------------------------------------------- 0.24s
Changing perm of "sh", adding "+x" ---------------------------------- 0.24s
Clean the script and folder. ---------------------------------------- 0.22s
build list of CMK hosts --------------------------------------------- 0.13s
Run the test cases if testing_enabled is set to true.
Add-ons deployment complete...
```

- Refer to the following link for details:

  https://wiki.onap.org/display/DW/Kubernetes+Baremetal+deployment+setup+instructions

Create Target Kubernetes cluster on Google Cloud

In case you are deploying AMCOP on GKE, you have to create a cluster for orchestration of CNF on Google cloud. To do so, you can allocate a VM and build a k8s cluster on it. Following set of commands will do the VM allocation and cluster creation.

```
cd <dir>/aarna-stream/util-scripts

./create_gke_kud.sh amcop-kud /tmp
```

Note: The above command may take 5 to 10 minutes (sometimes more) to create and initialize the cluster.

Once the cluster is created, you need to copy the kubeconfig file for the cluster.

```
gcloud compute scp amcop-kud:~/.kube/config /tmp/kud_cluster_conf_file
```

Note: The above configuration file is required for the orchestration of CNF using the GUI or the rest interface.

Create Target Kubernetes cluster on Microsoft Azure

When deploying AMCOP on the AKS cluster, you have to create the target cluster on Azure. You can use the following commands to create the cluster.

```
cd <dir>/aarna-stream/util-scripts

./create_aks_kud.sh
```

Once the cluster is created and initialized, you can copy the kubeconfig file from the cluster using the following command.

```
az vm list --show-details -o=table  | grep amcop-kud | awk '{ print $5 }'

scp aarna@<IP ADDR>:~/.kube/config /tmp/kud_cluster_conf_file
```

Create Target Kubernetes cluster on Amazon EKS

When deploying AMCOP on the Amazon EKS cluster, you have to create the target cluster on EKS. You can use the following commands to create the cluster.

```
cd <dir>/aarna-stream/util-scripts

./create_amazon_edge_cluster.sh
```

Note: Running the above script will overwrite the kubeconfig (if one is present) so it is advisable to run it from a different Linux user id.

Note: The above will create another cluster on the same subnet as AMCOP cluster. The new cluster will be used for CNF deployment.

RBAC

AMCOP supports RBAC (Rule based Authentication controls) which is explained in this section.

This section assumes that the AMCOP is deployed with RBAC enabled (which is the default option).

AMCOP RBAC defines two roles: Admin and Tenant.

The admin credentials are *admin@aarnanetworks.com* and the default password is "*test*". These are created by default when the AMCOP is deployed. As an Admin you should change the password after logging in for the first time.

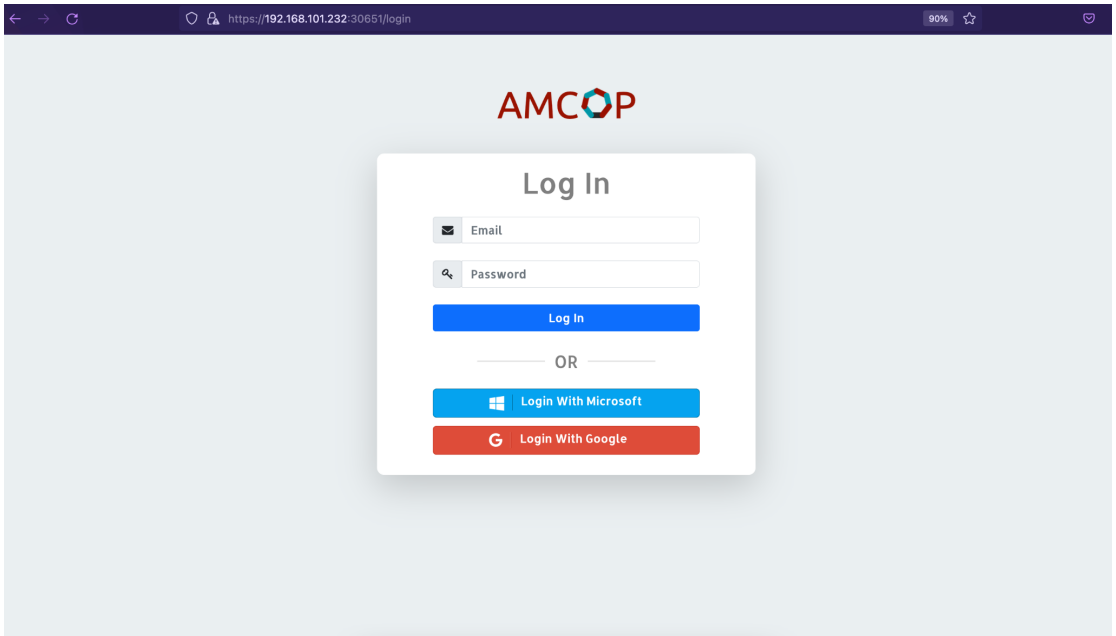
Login as Admin

The admin can perform the following actions,

1. Create Tenants
2. Add k8s controllers
3. Onboard clusters
4. Create users

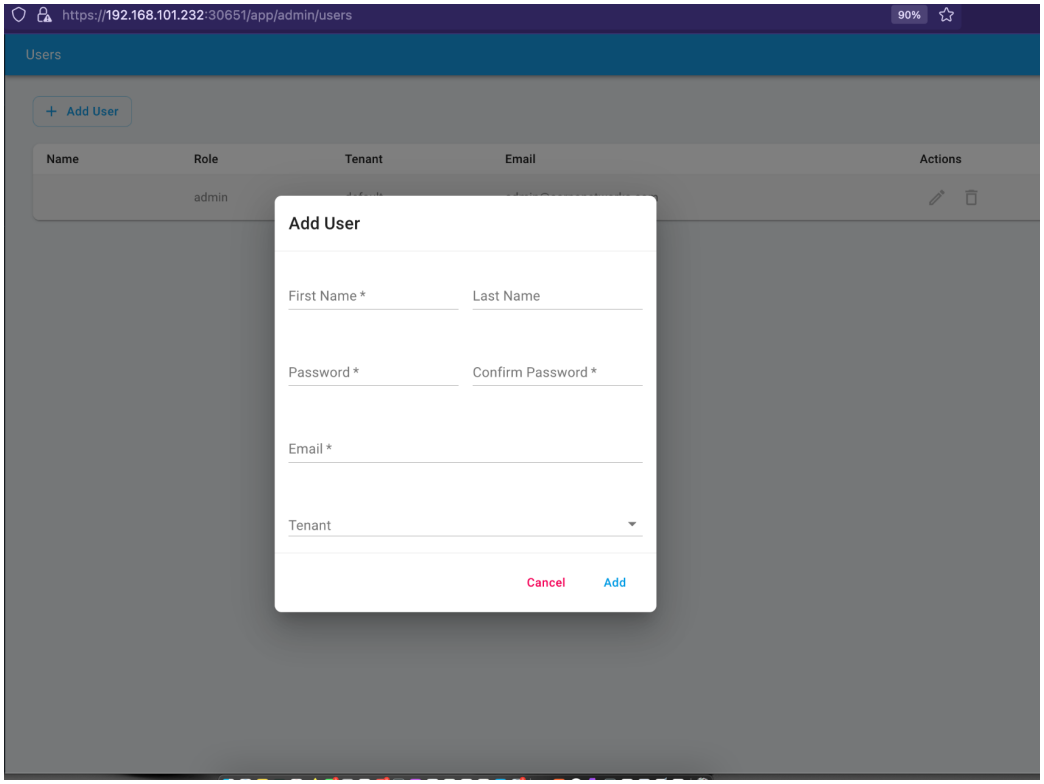
To add User, do the following:

1. On the browser access the AMCOP app at https://<server ip>:30661/app
2. Login to the AMCOP:
    a. Username: admin@aarnanetworks.com
    b. Password: test

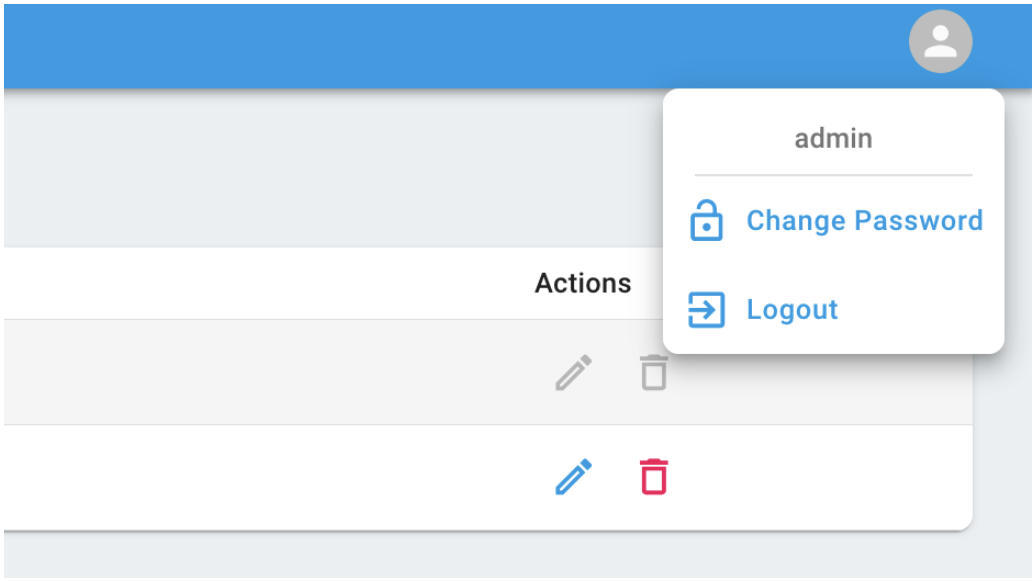


3. Click on the Tenants tab in the left panel and create a tenant.

4. Click on the Users tab in the left panel, and then click on Add User button, and this will pop up a form on the GUI.



5. After adding the user, logout

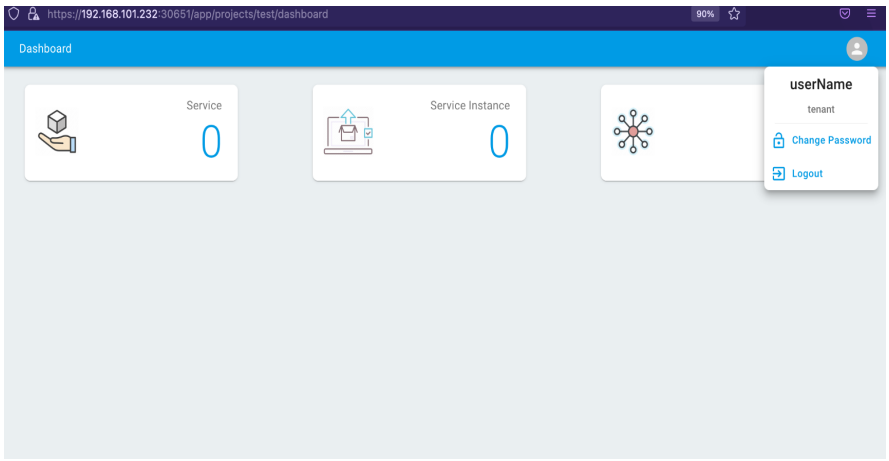


day2

Login as User with Tenant Role

1. User with Tenant Role can perform the following operations,
    a. Create services

   b. Create logical clouds
   c. Create and instantiate service instances.
2. Login with the user name and password provided by the Admin, and your landing page will look like the following,



Orchestration of vFirewall using AMCOP GUI

This section shows how to register a k8s cluster with AMCOP, design a network service (using vFirewall as an example) and orchestrate them using AMCOP GUI.

After setting up the SOCKS tunnel to the AMCOP Jump host (as described above), and setting up a proxy in Firefox browser, the AMCOP GUI can be accessed at:

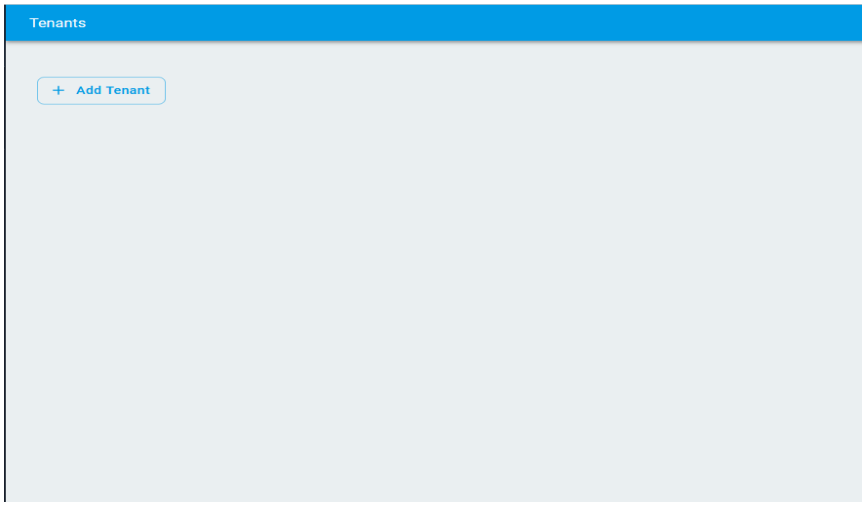
http://<amcop-master-vm-ip>:30661

If AMCOP is deployed on a GKE or AKS cluster then the IP address and port number are different.

The user interface of AMCOP GUI is divided into two parts. One is for admin related functionalities like adding projects, onboarding clusters, adding controllers etc and the other for the service designer related functionalities like Creating service, instantiating service etc.

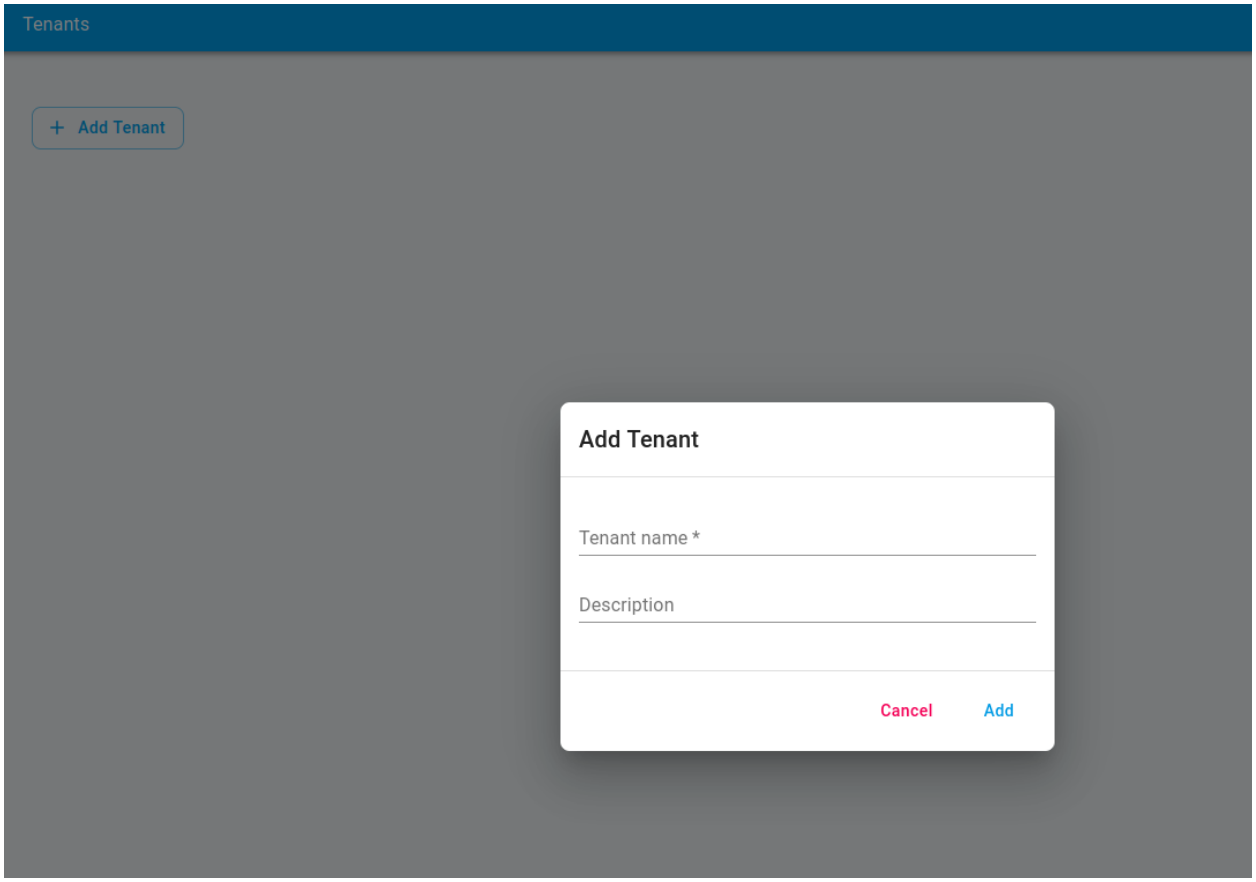
Admin User

Once the GUI is launched, the tenants page will be displayed.

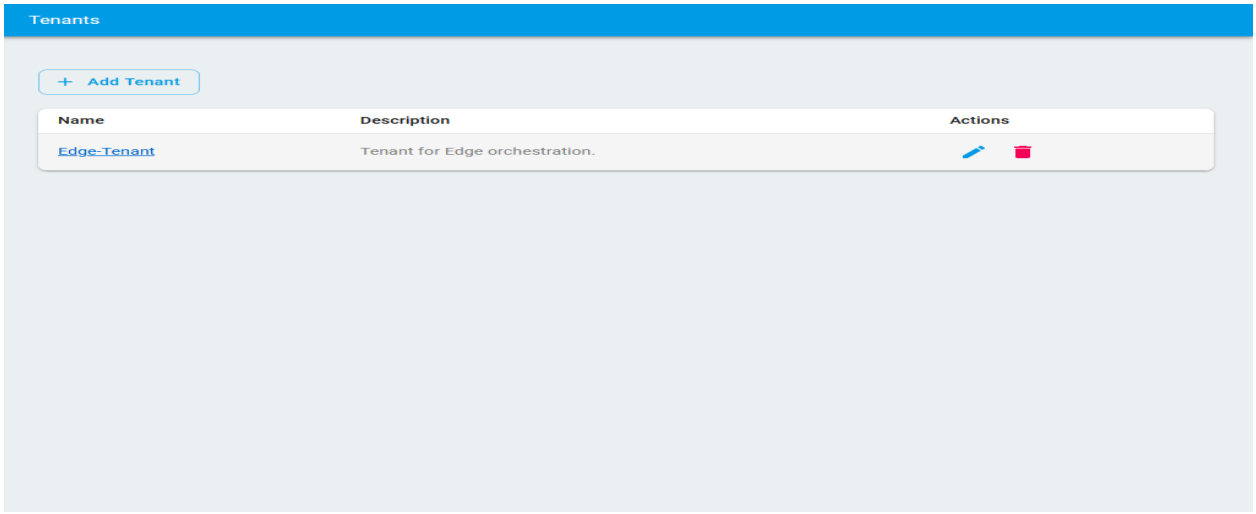


1. Add a Tenant
    a. To start with, add a tenant by clicking on the Add Tenant button and filing the required fields and click Add.

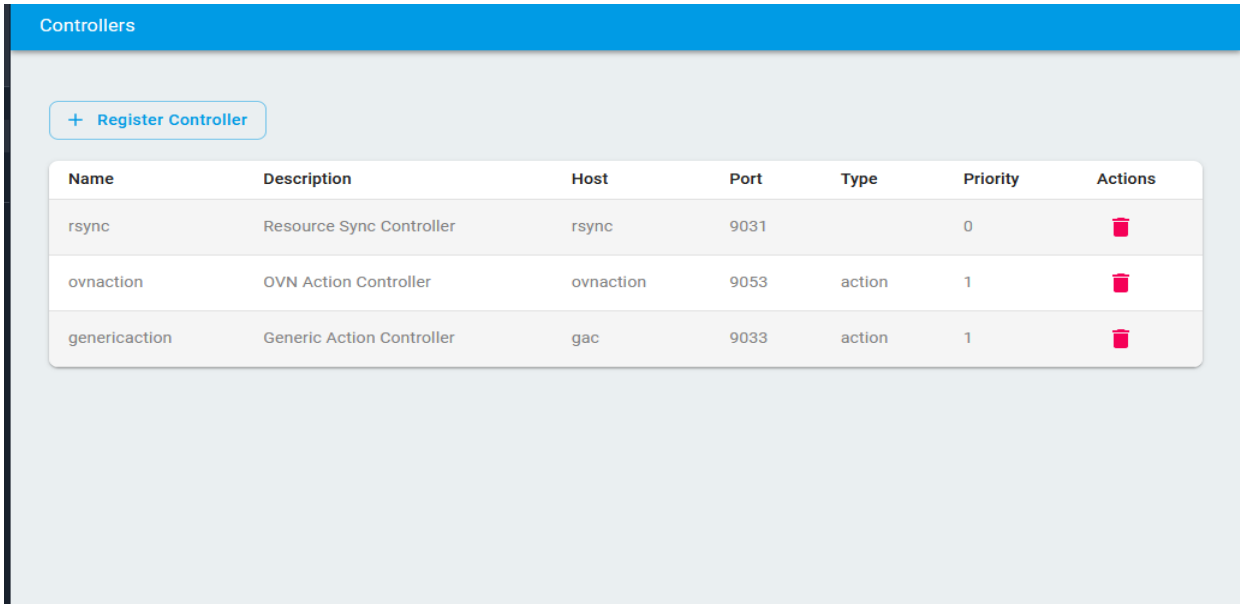
Note: Tenant name should be less than 20 characters



b. Once the tenant is added it will appear in the tenants list. You can edit or delete the tenant from the action buttons.
NOTE: Only the tenant description can be updated and a tenant can only be deleted if there are no resources inside it.



2. Register K8s Controllers

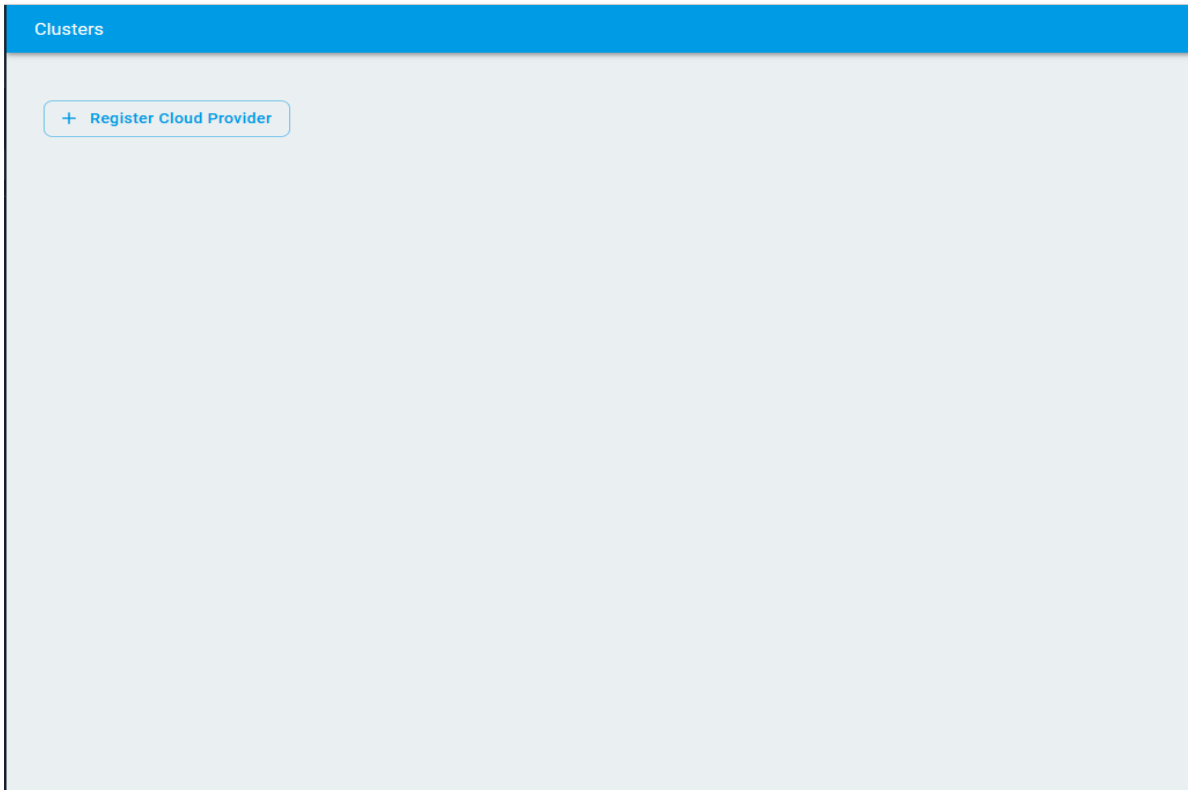


a. You can register an external controller by going to the K8s Controllers page on the left-hand side navigation bar. Currently, two types of controllers “Placement” and “Action” are supported only.

b. A controller can be deleted by clicking the delete button' 🗑 ' in the actions column.

As part of AMCOP deployment, three controllers, "rsync", "ovnaction" and "genericaction" are bundled by default. They do not need to be registered explicitly. These controllers are sufficient for sample applications such as vFW. But if you are onboarding other applications that require different controllers, they need to be registered separately.

3. Onboard Clusters
   a. To onboard a cluster, first, you need to register a cloud provider. To register a cloud provider, go to the *Clusters* tab on the left hand navigation bar and click on *Register CloudProvider*. Fill in the basic details and click Create.



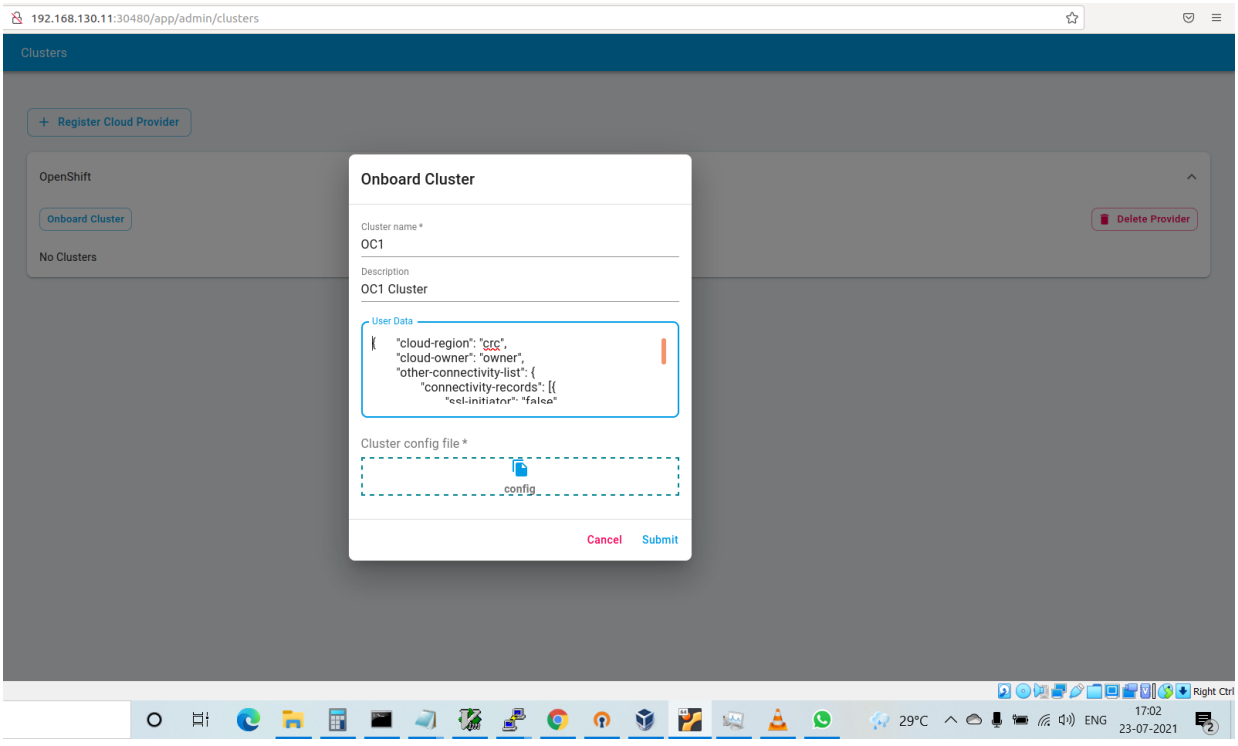
   b. Download the target cluster's k8s config file to your local workstation. It will be required in the next step.
   Once a cloud provider is registered it will appear as an expandable row as shown below. Click on the row to expand it. Once the row is expanded click on the *OnboardCluster* button to onboard a cluster. Fill in the basic details

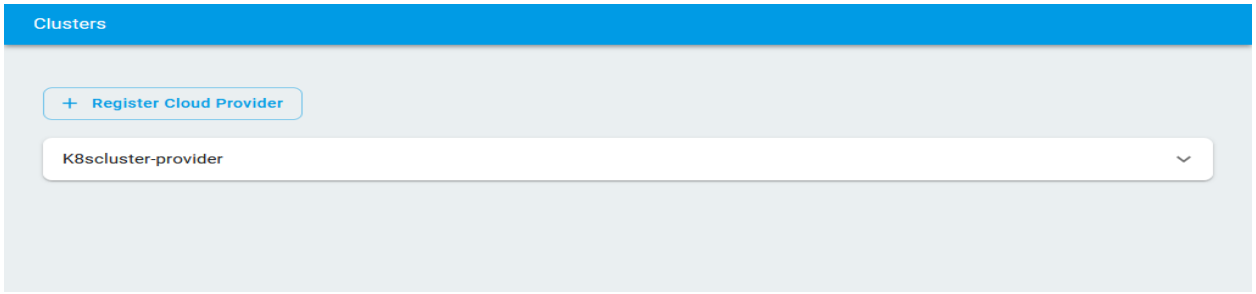
and upload the target cluster's kube config file. If your cluster needs some additional information such as username, password etc, add them to the user data field.

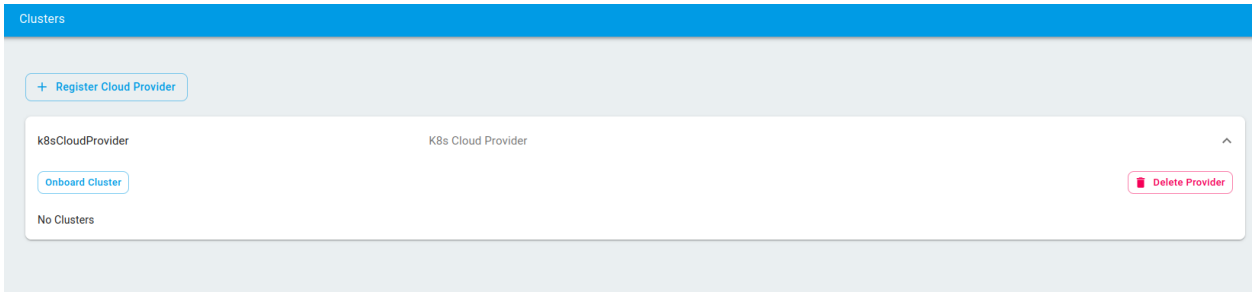
Note:

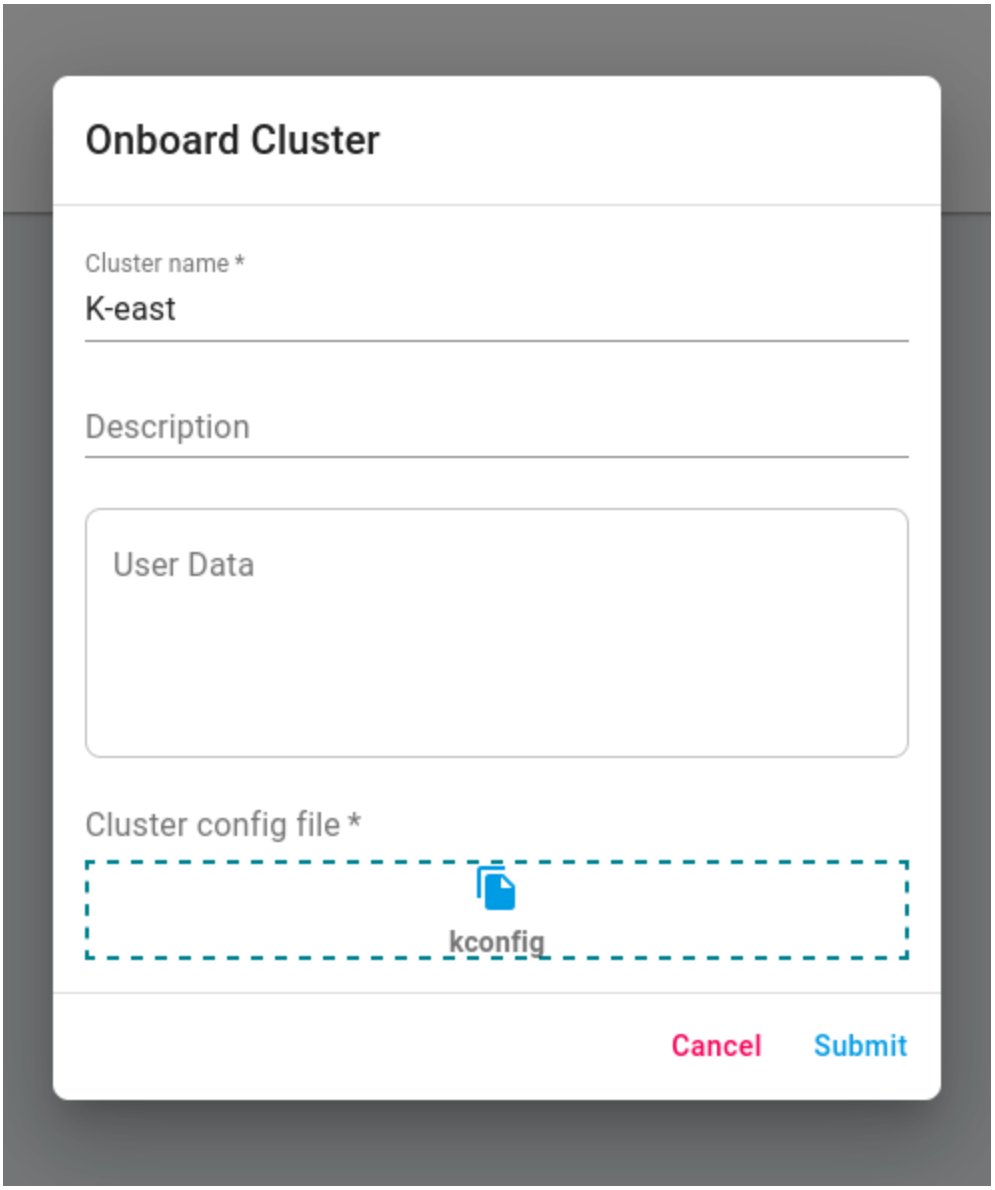
For other platforms such as Openshift, more parameters need to be input, which are mentioned below.

```
{
"cloud-region": "crc",
"cloud-owner": "owner",
"other-connectivity-list":
  {
        "connectivity-records": [{
        "ssl-initiator": "false",
        "user-name": "kubeadmin",
        "password": "f4swJ-AaAA3-c5x8D-rsrbA"
        }]
  }
}
```

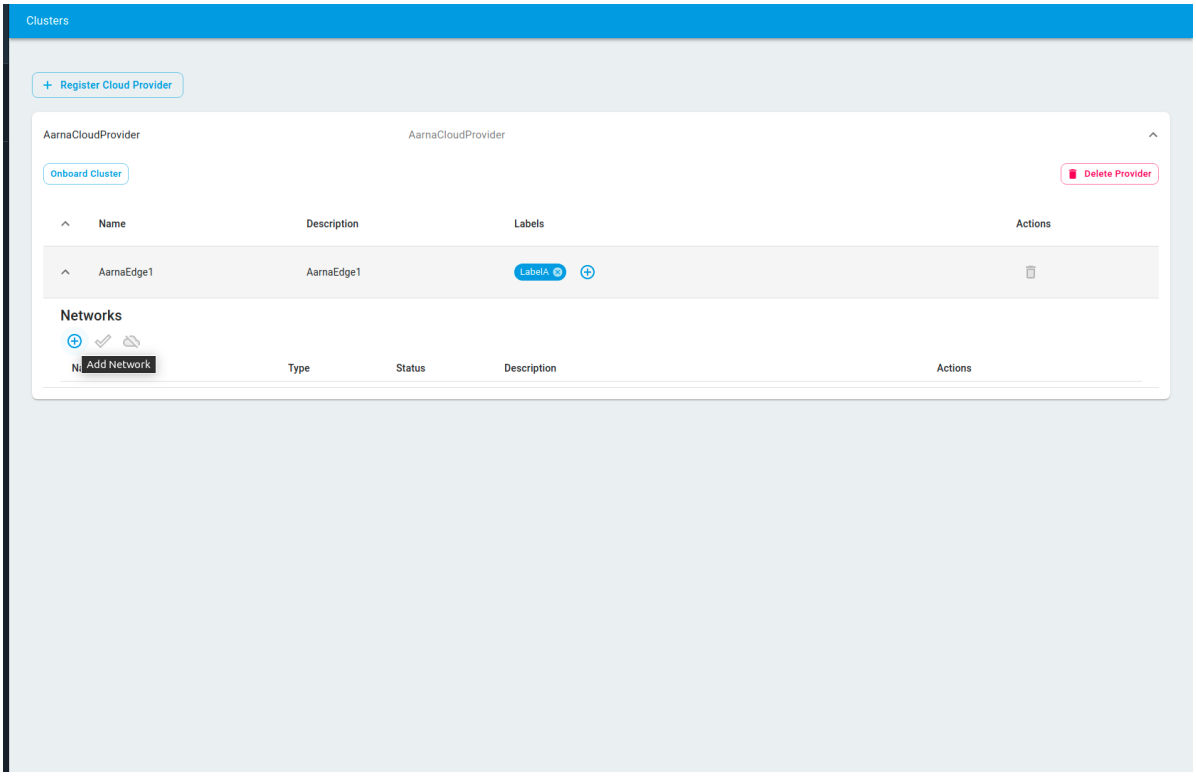



Clusters
Register Cloud Provider
K8scluster-provider


Clusters
Register Cloud Provider
k8sCloudProvider
K8s Cloud Provider
Onboard Cluster
Delete Provider
No Clusters



c. Once a cluster is onboarded, you can add labels to it by clicking on the plus '⊕' icon in the labels column. To delete a label, click on the cross icon on the label.
If required, you can add networks to the cluster, click on the "+ *network*" button in the actions column. For vFW service orchestration, you don't have to add networks. You can skip the below step "d" and continue with the subsequent section.



d. Fill in the required fields and click create. Once a network is added it needs to be applied. To do that click on the check '✓' icon in the Actions column.

In case you need multiple networks, the required K8s controllers (mentioned subsequently) need to be added. If not, the following step of adding multiple networks fails.

For the vFW service following are the networks which are to be created, along with the corresponding network Spec.

1. On the Type drop down select 'provider-networks'
   Network name: emco-private-net

   Copy and paste the following blob to Network Specs*

```
{
        "cniType": "ovn4nfv",
        "ipv4Subnets": [
                {
                        "subnet": "10.10.20.0/24",
                        "name": "subnet1",
                        "gateway":  "10.10.20.1/24"
                }
        ],
```

```
                "providerNetType": "VLAN",
                "vlan": {
                        "vlanId": "102",
                        "providerInterfaceName": "eth1",
                        "logicalInterfaceName": "eth1.102",
                        "vlanNodeSelector": "specific",
                        "nodeLabelList": [
                                "kubernetes.io/hostname=localhost"
                        ]
                }
}
```

Select the “create” button.

2. Type: provider-networks
   Name: unprotected-private-net

   Copy and paste the following blob to Network Specs.

```
        {
                "cniType": "ovn4nfv",
                "ipv4Subnets": [
                        {
                                "subnet": "192.168.10.0/24",
                                "name": "subnet1",
                                "gateway":  "192.168.10.1/24"
                        }
                ],
                "providerNetType": "VLAN",
                "vlan": {
                        "vlanId": "100",
                        "providerInterfaceName": "eth1",
                        "logicalInterfaceName": "eth1.100",
                        "vlanNodeSelector": "specific",
                        "nodeLabelList": [
                                "kubernetes.io/hostname=localhost"
                        ]
                }
        }
```

3. Type: networks
   Name: protected-private-net

   Copy and paste the following blob as Network specs*,

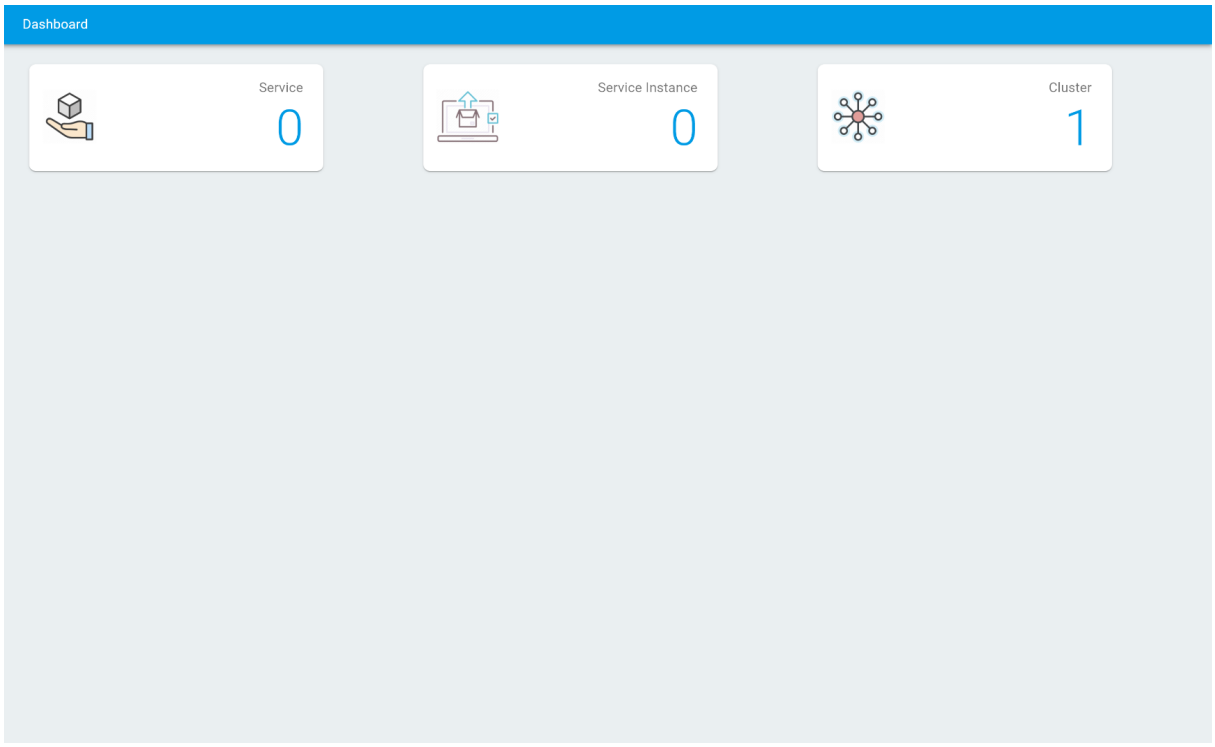
```
{
                "cniType": "ovn4nfv",
```

```
                "ipv4Subnets": [
                    {
                        "subnet": "192.168.20.0/24",
                        "name": "subnet1",
                        "gateway":  "192.168.20.100/32"
                    }
                ]
            }
```

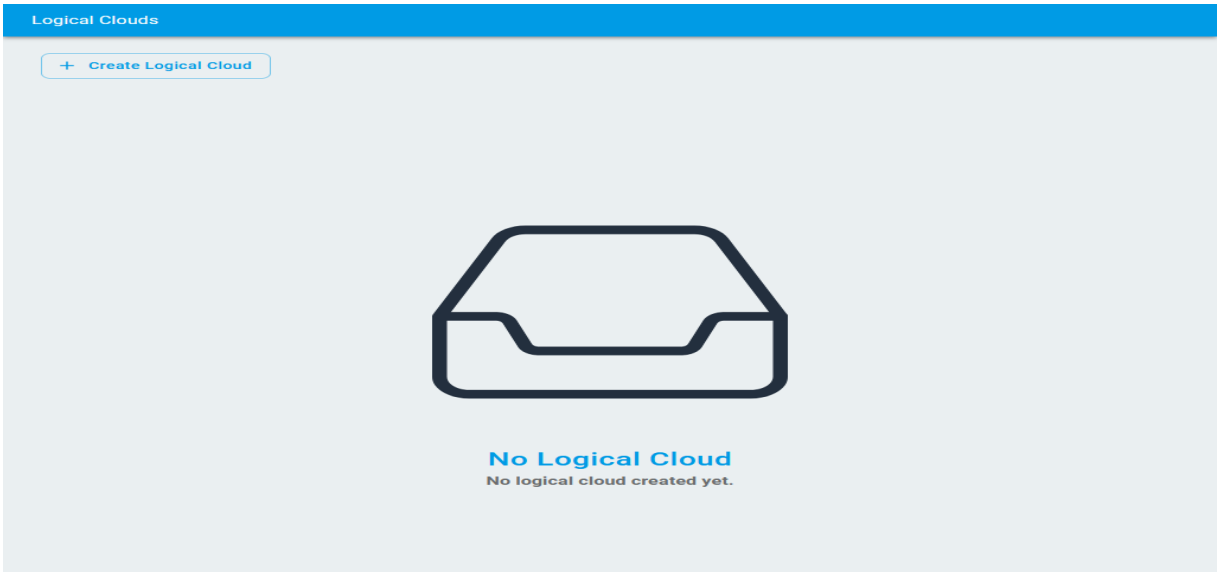
Service Designer User

To go to the service designer page, click on the *Tenants* tab and then click on the name of the tenant from the tenants table.

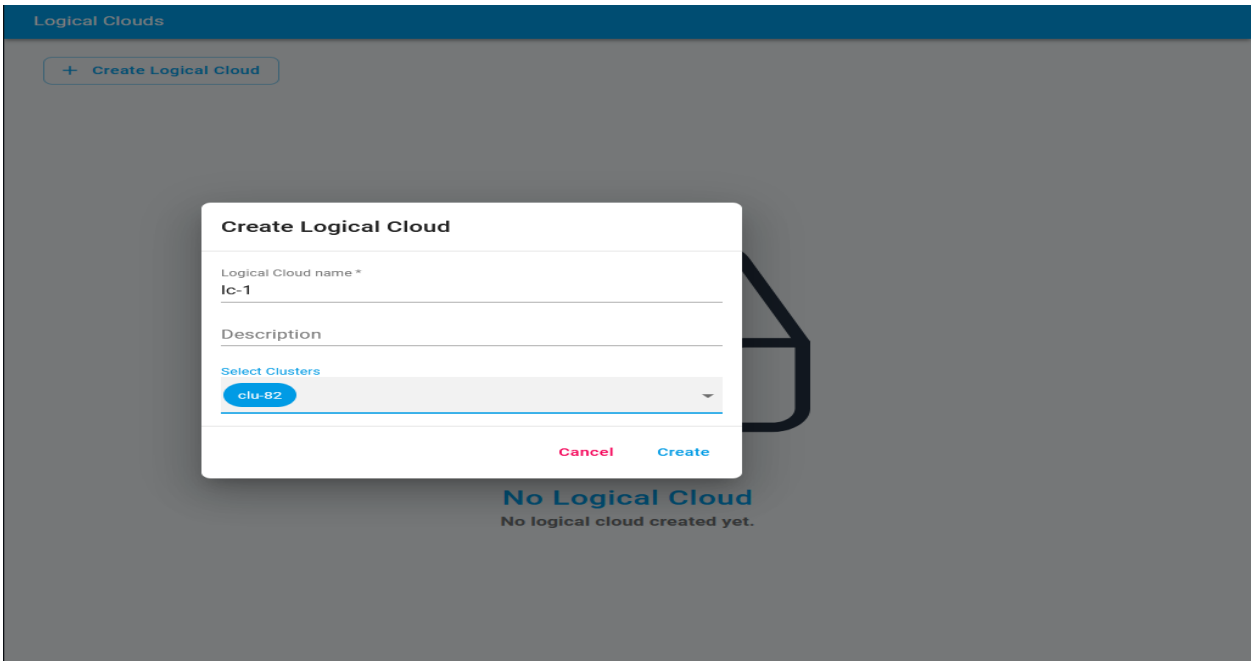
Once a tenant name is clicked, the tenant dashboard is displayed.
The dashboard gives an overview of the tenant resources like total service count, service instance count and available cluster count.



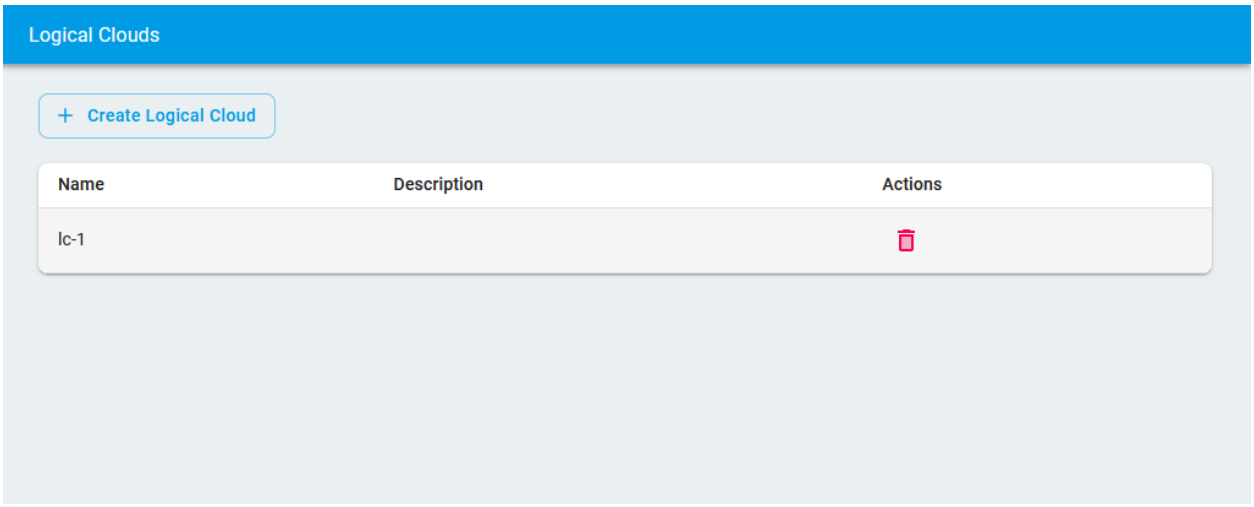
1. Create a logical cloud



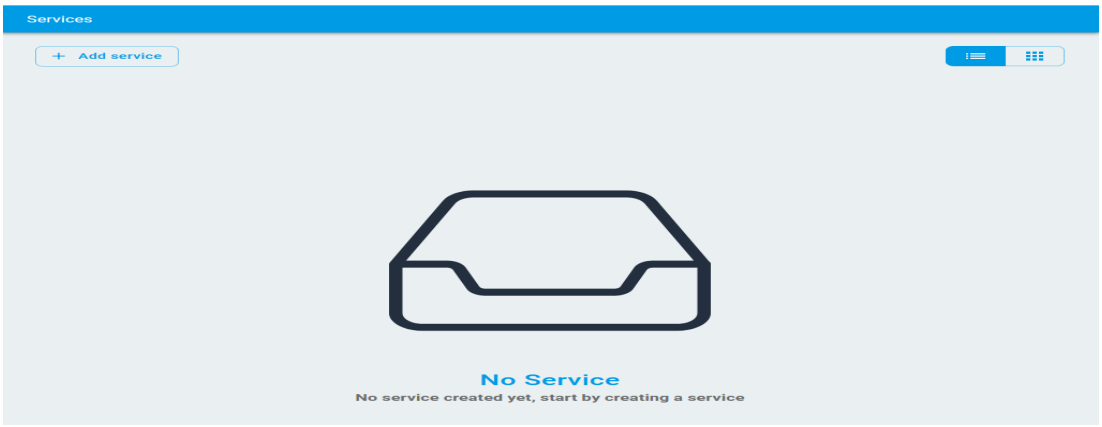
Add cluster to the logical cloud:



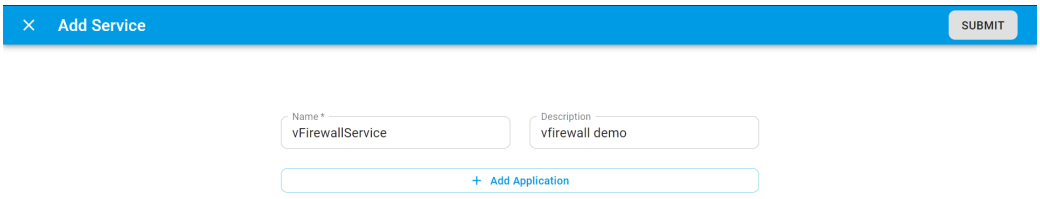
Click on *Create*:



2. Add a Service
    a. To add a service first click on the Services tab from the left-hand side menu.



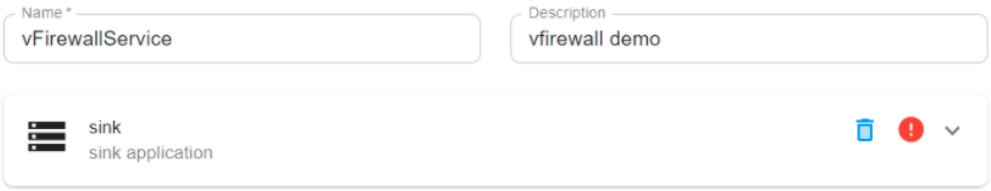
    b. Now click on the *Add Service* button to add a new service.
    c. Fill in the basic information like Service name and add description and then click on *Add Application* to add an application in the service.



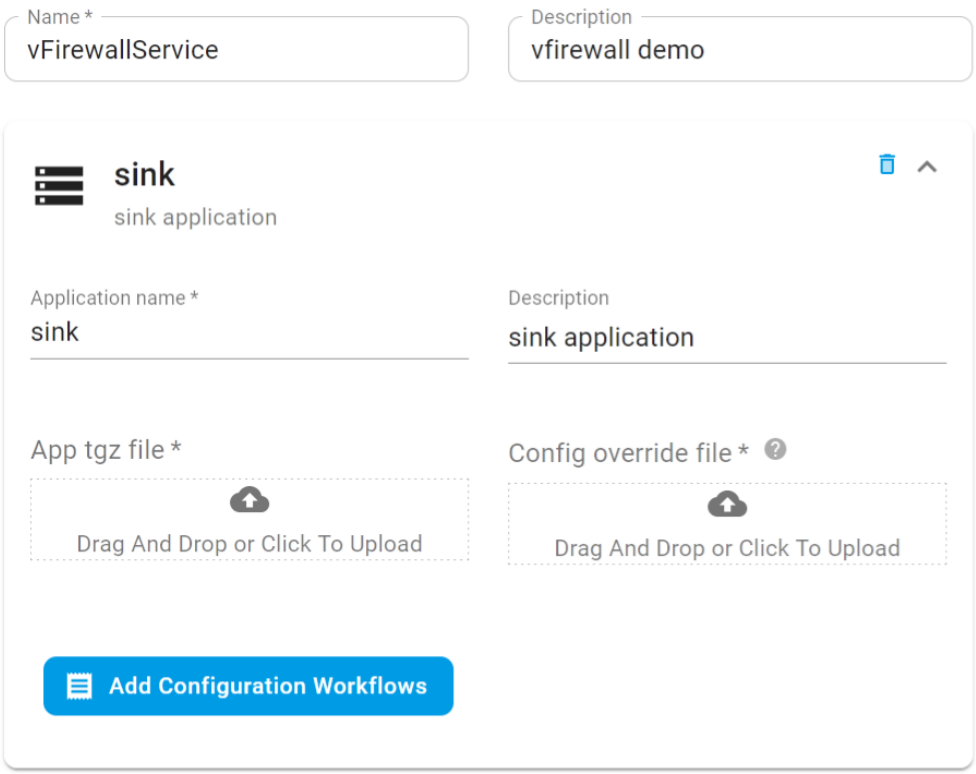
    d. In the form to add the application, fill in the name and description of the application and click *Create*.

NOTE: The application name should match the helm chart name in the package For example in case of vFW the app names should be sink, packetgen and firewall as the helm charts are sink.tgz, packetgen.tgz and firewall.tgz.

e. Once create is clicked, an application row will be added in the service form as shown below. At this point, the app form is not complete yet, so there will be a red exclamation mark. Until the form is valid, the submit action will not succeed.

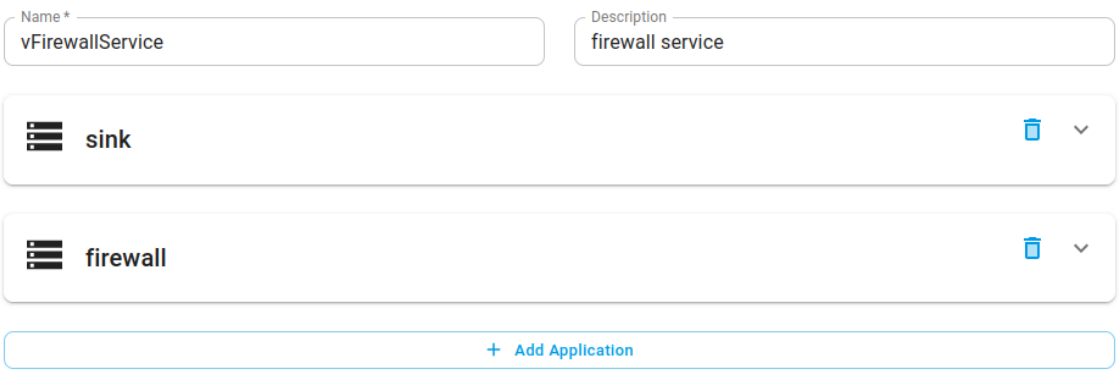


f. Click on the app row to expand it. Now upload the app .tgz package file and profile.tar.gz file which contains the override files by clicking on the upload button or dragging and dropping in the upload area. App name and description can also be changed here.
Note: The helm charts for the vFW are present in /home/<user>/aarna-stream/cnf/vfw_helm_withnw/ directory.
The profile bundle is present in /home/<user>/aarna-stream/cnf/payload/.
The name of the bundle is profile.tar.gz

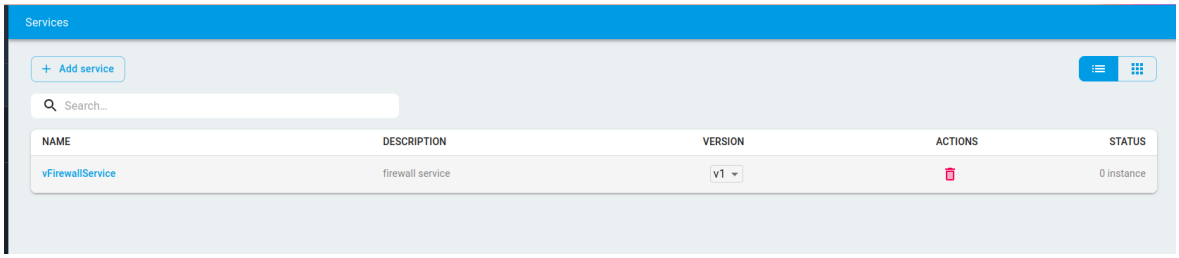


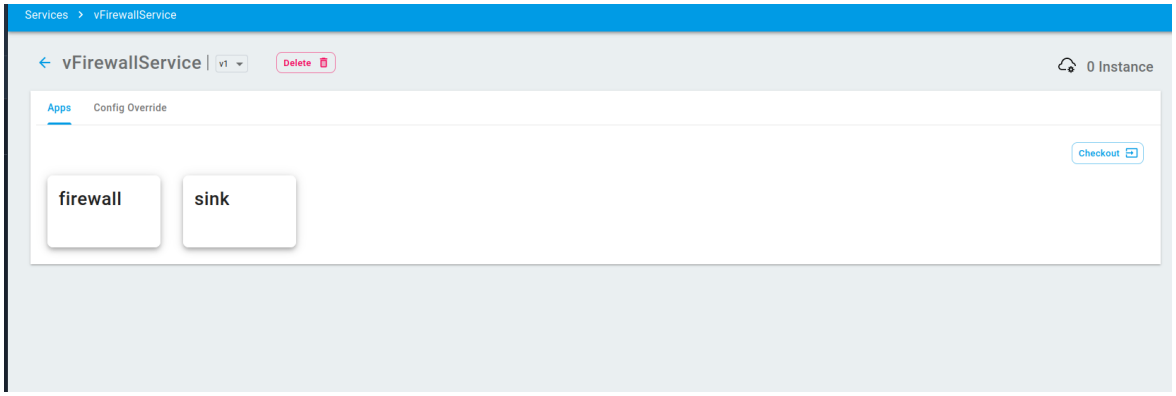
g. For vfirewall, you need to add a total of three applications, sink, packetgen and firewall. So repeat the above step to add the other 2 applications.

h. Once all the applications have been added, click on the *submit* button in the top right corner of the screen to submit the service design as shown below.



i. Now once the service is created, it will appear on the services page. You can look at the details of the service by clicking on the name of the service in the services table.



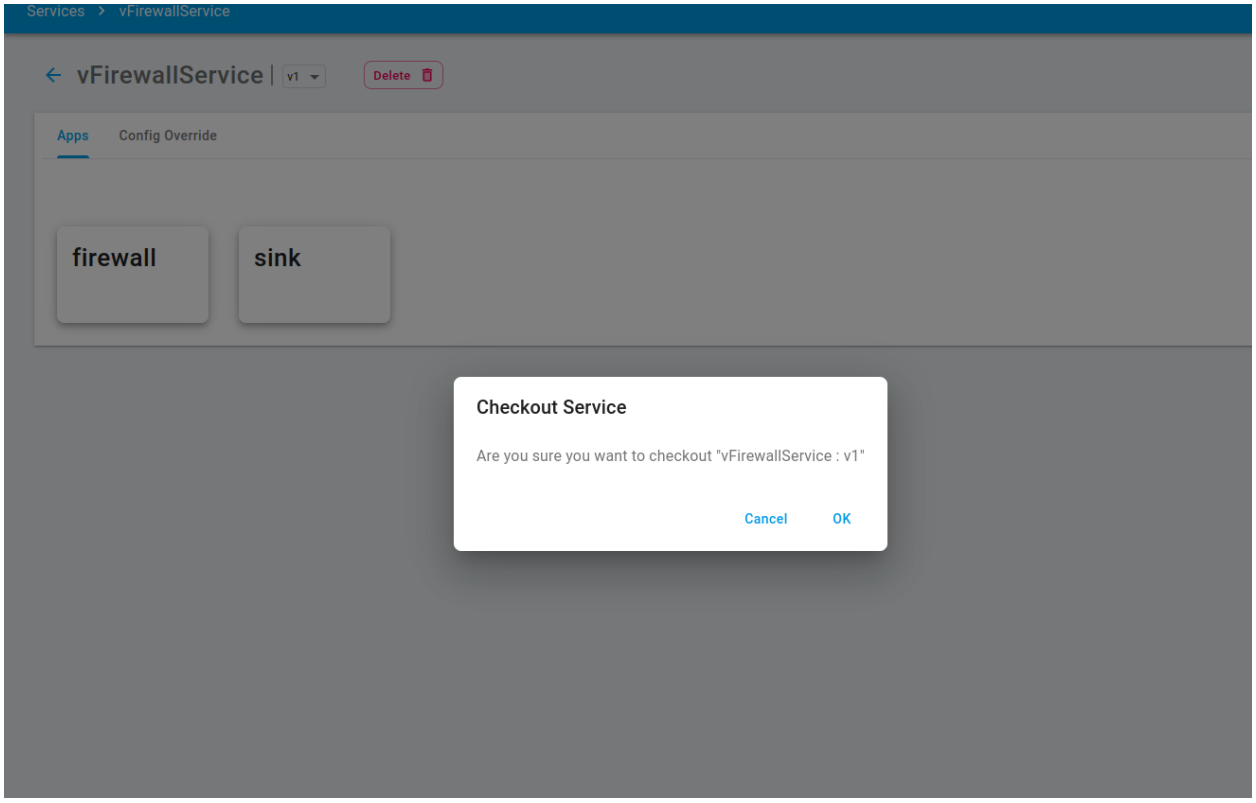


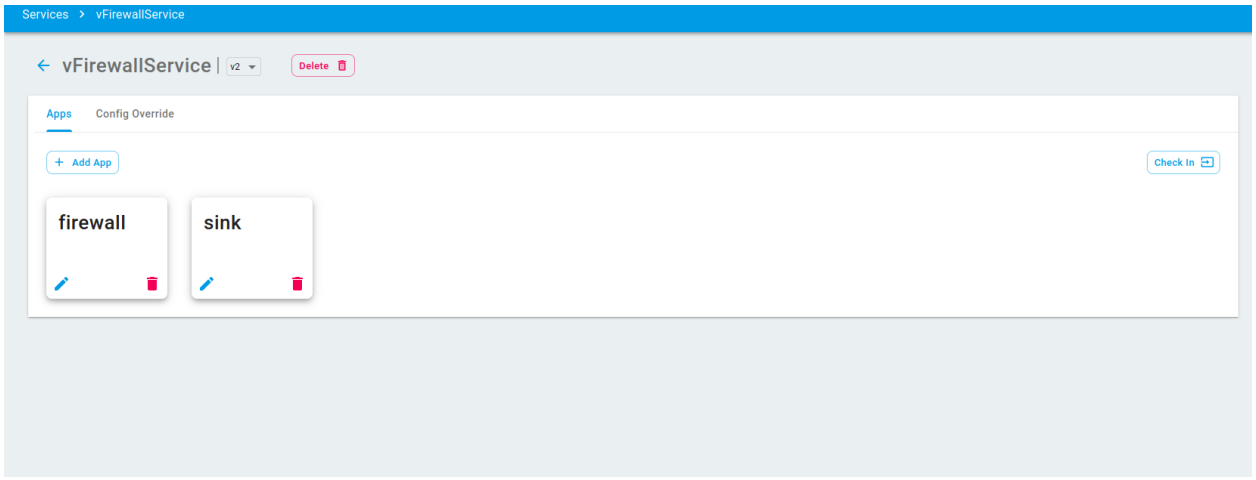
3. Checkout and Edit a service

You can perform the following operations on the created service.

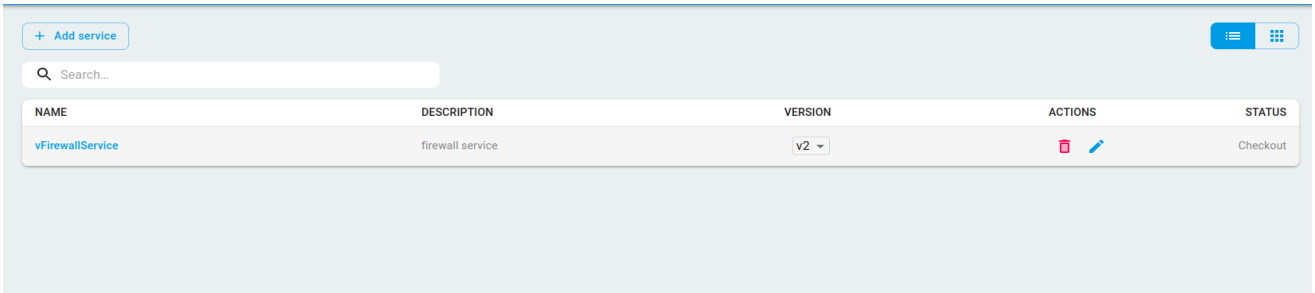
a. Upload new helm charts for existing applications.
b. Delete an application in the service.
c. Add new applications to the service.

1. Checkout service: Select the version of the service that is to be edited and click on the checkout button. This will create a canvas for modifying the service.



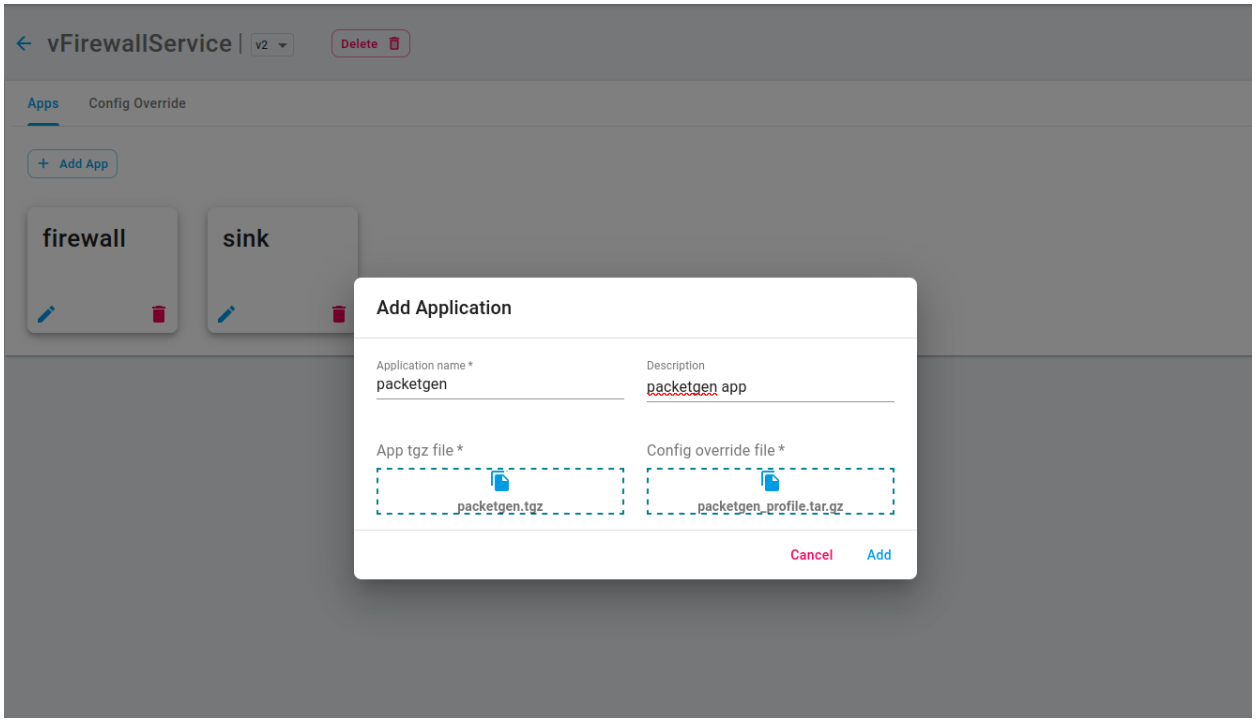


2. If needed, you can return to the main page and take up the modification work later. The status of the service remains in the checkout state.

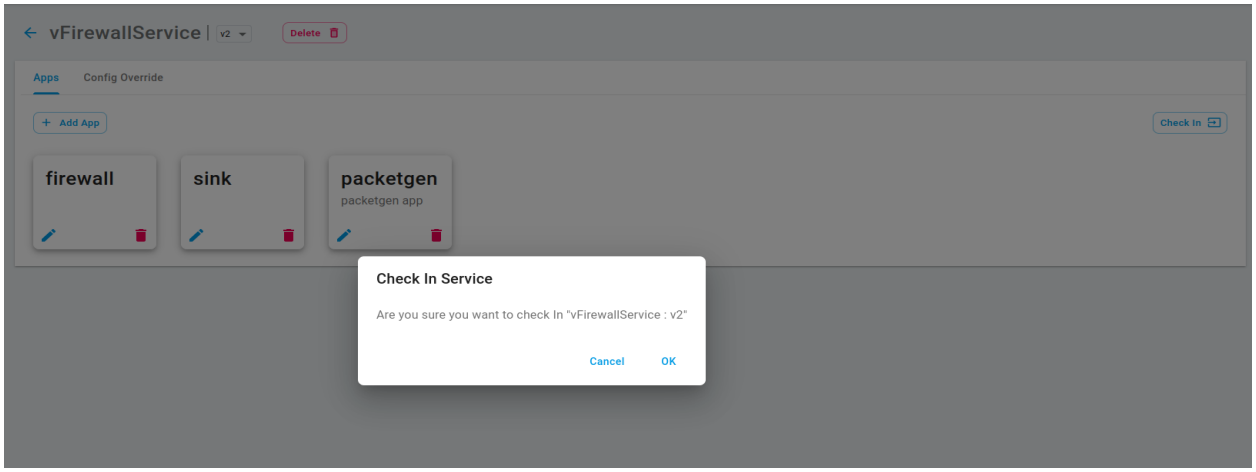


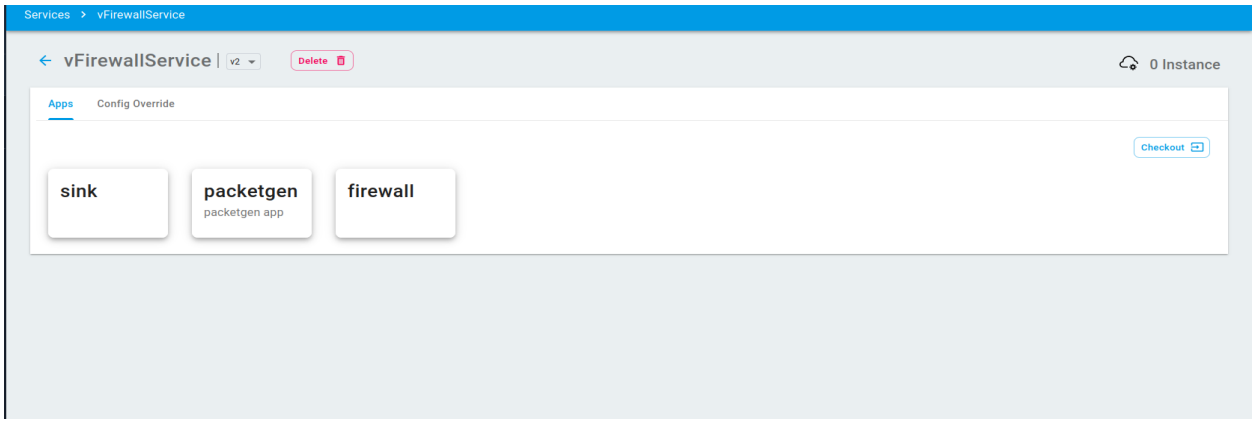
3. Add a new application to the checked-out service.

   NOTE: The application name should match the helm chart name in the package For example in case of vFW the app names should be sink, packetgen and firewall as the helm charts are sink.tgz, packetgen.tgz and firewall.tgz.



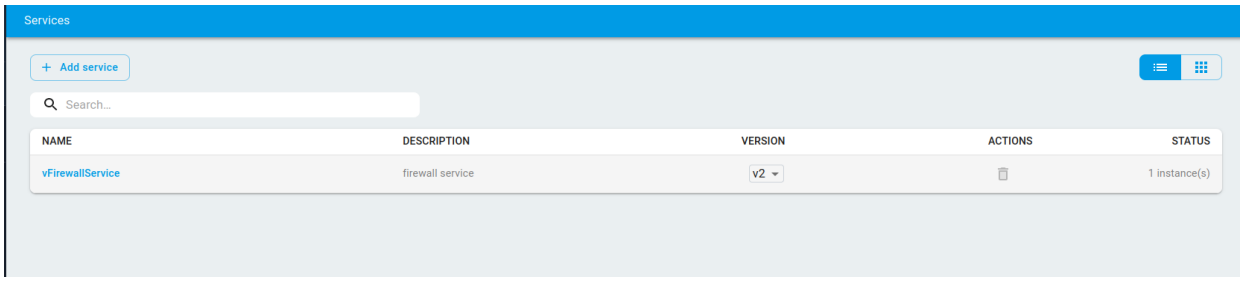
4. Check in.





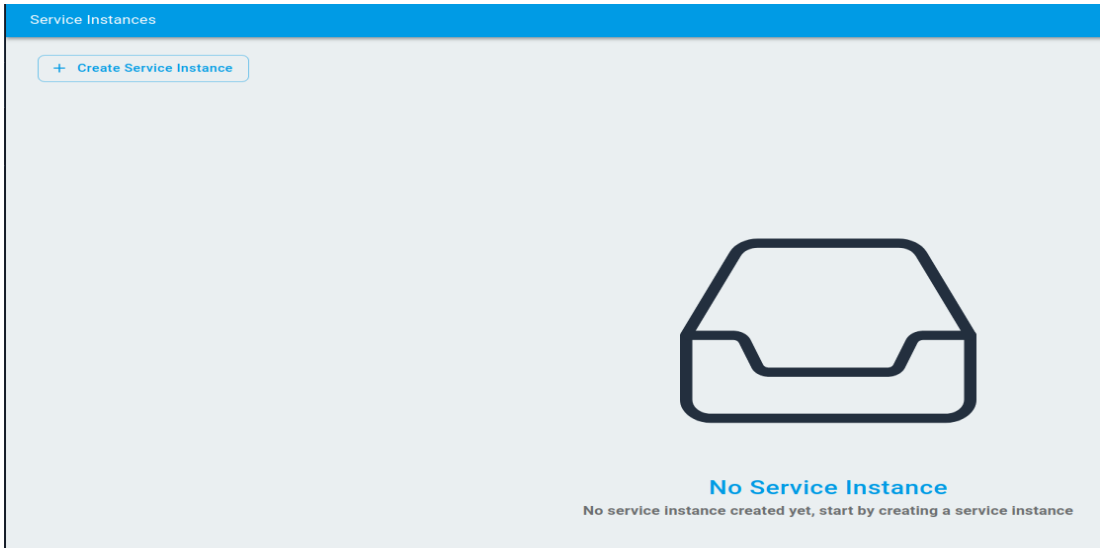
On the service page, the status of the service means the following,

a. Checkout: Service is in checkout state.
b. numberInstances : Number of service instances that are created for a specific version.

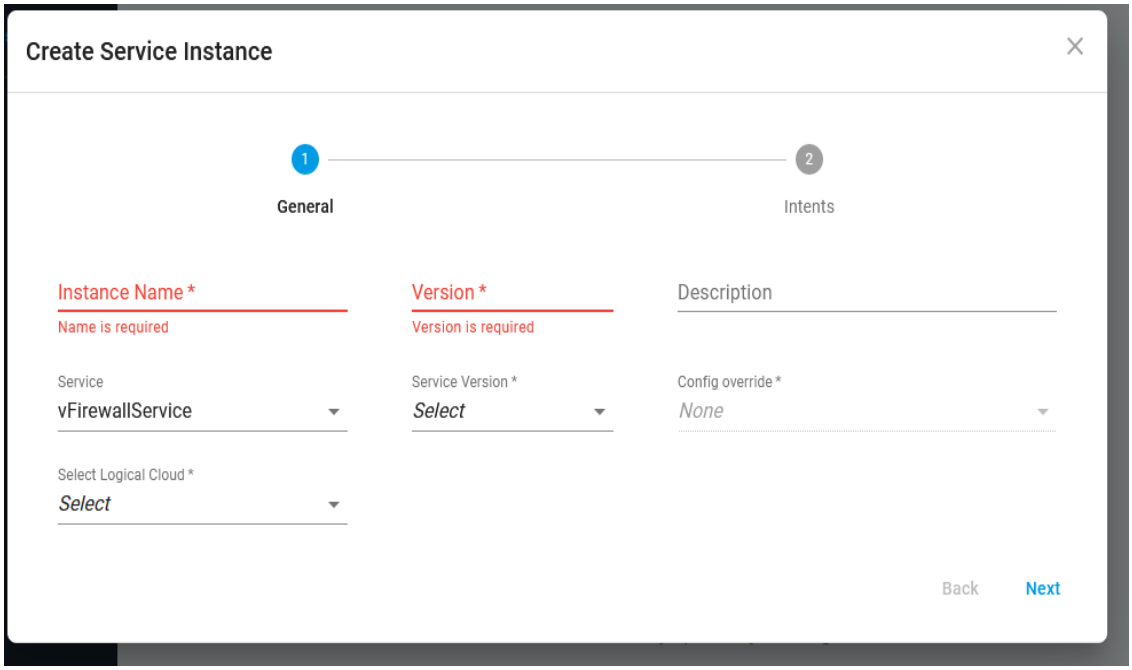


4. Create a service instance.

   a. To create a service instance, go to the service instances screen from the left-hand side navigation and then click on create *service instances* button, this will open the service instance form.

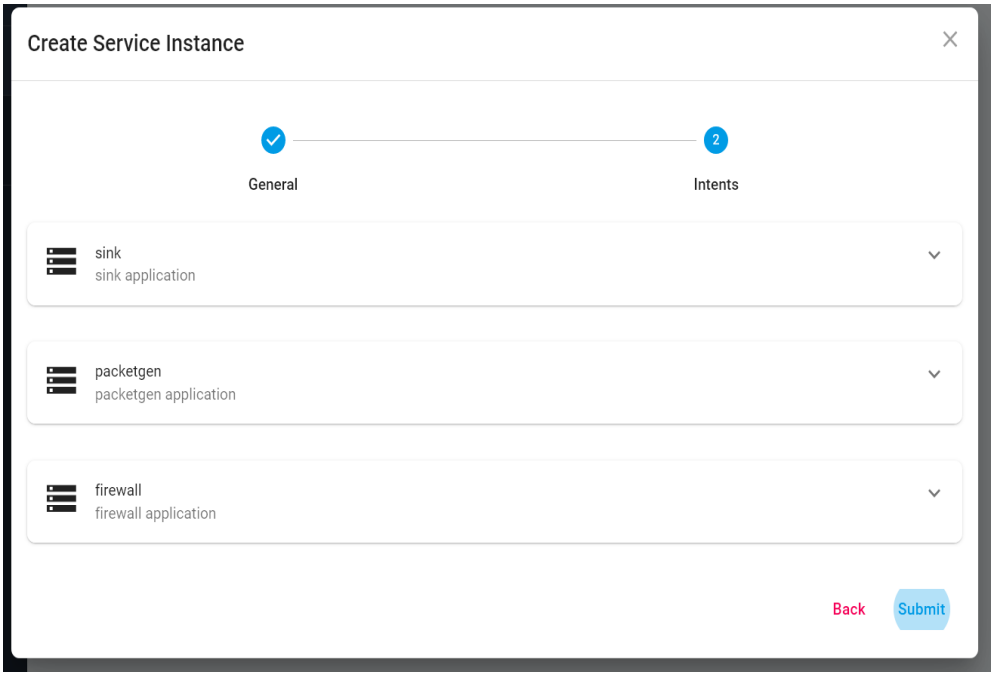


b. In the service instance form, fill in the details like service instance name, version, description, logical cloud, etc. Also, select the service from the dropdown for which the instance needs to be created. In this case select vfirewall Service.

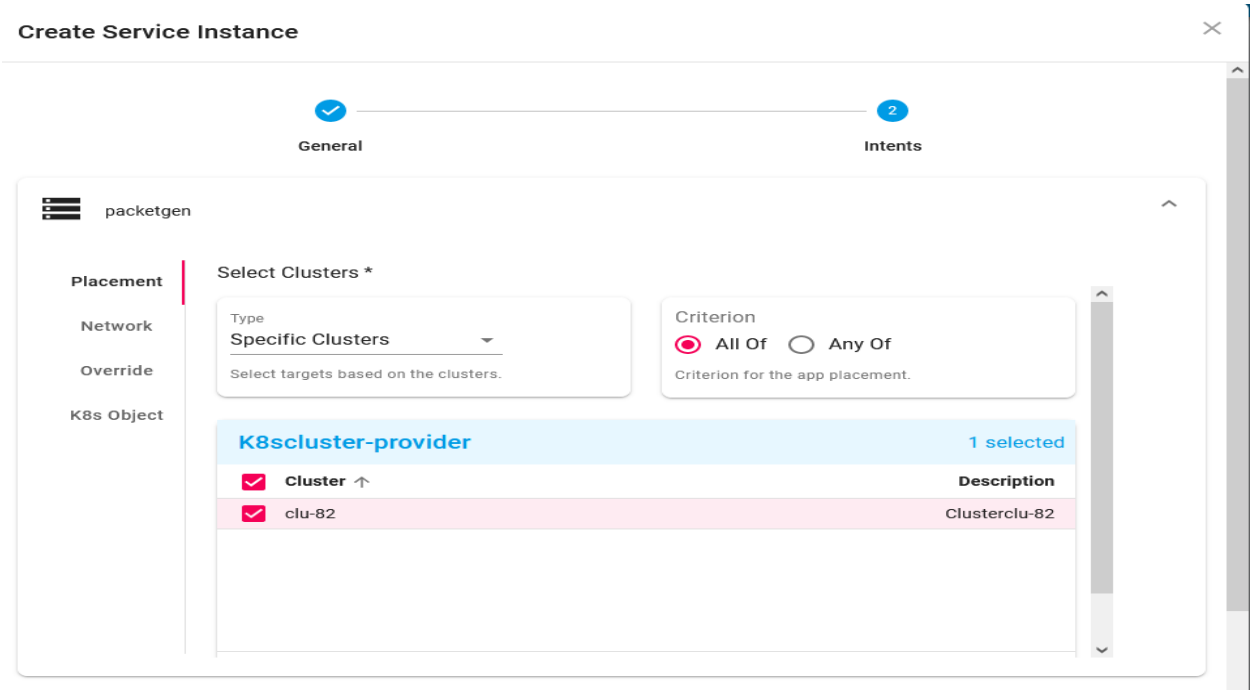


c. When a service is selected, it's corresponding override file is automatically selected.

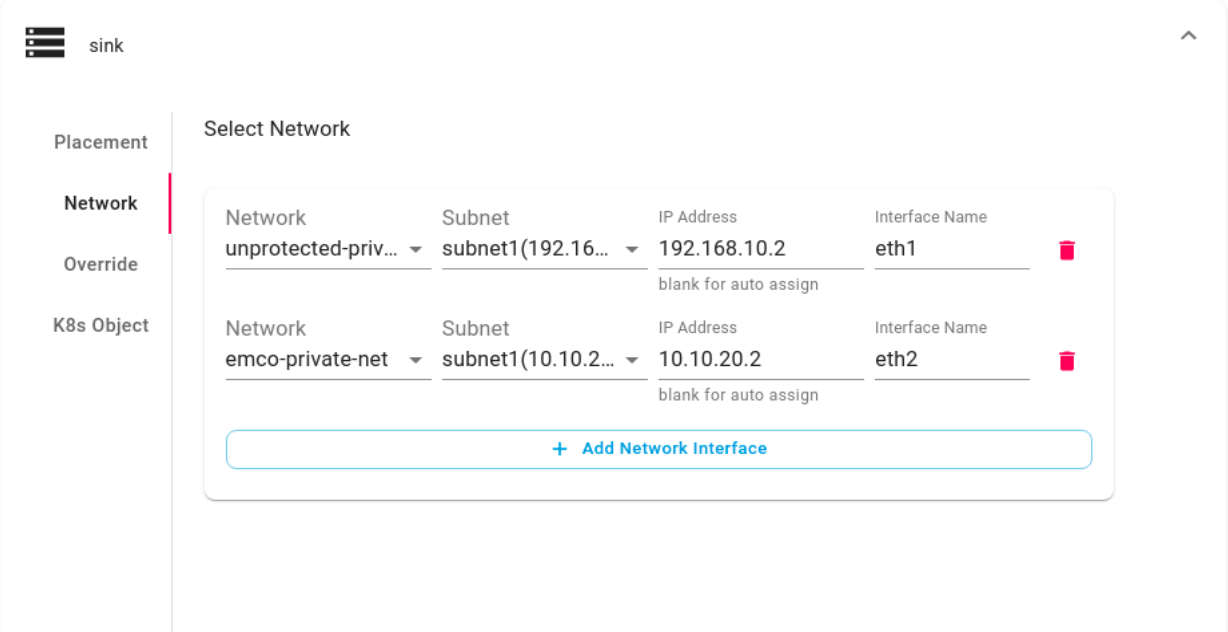
d. You can also provide override values if you need to override any value in the service instance at runtime. For vfirewall leave it blank.
e. Once all the required fields are filled, click on next.
f. Now you will see all the apps which are there in the selected service in the previous step. You can click on each app and expand it.



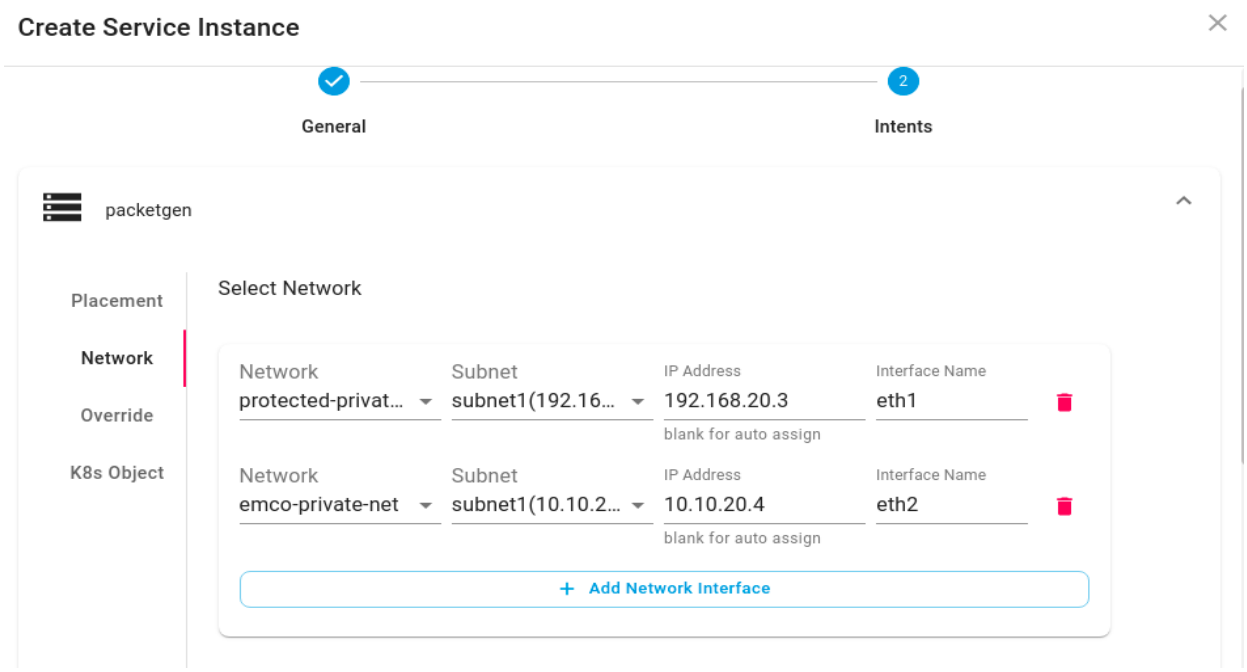
g. Once the application row is expanded, you can see tabs: *placement, network, Override and K8s Object*. These tabs are for placement intents, network interfaces, Override values and adding K8s objects . In the placement tab select the clusters in which you want your app to be deployed.



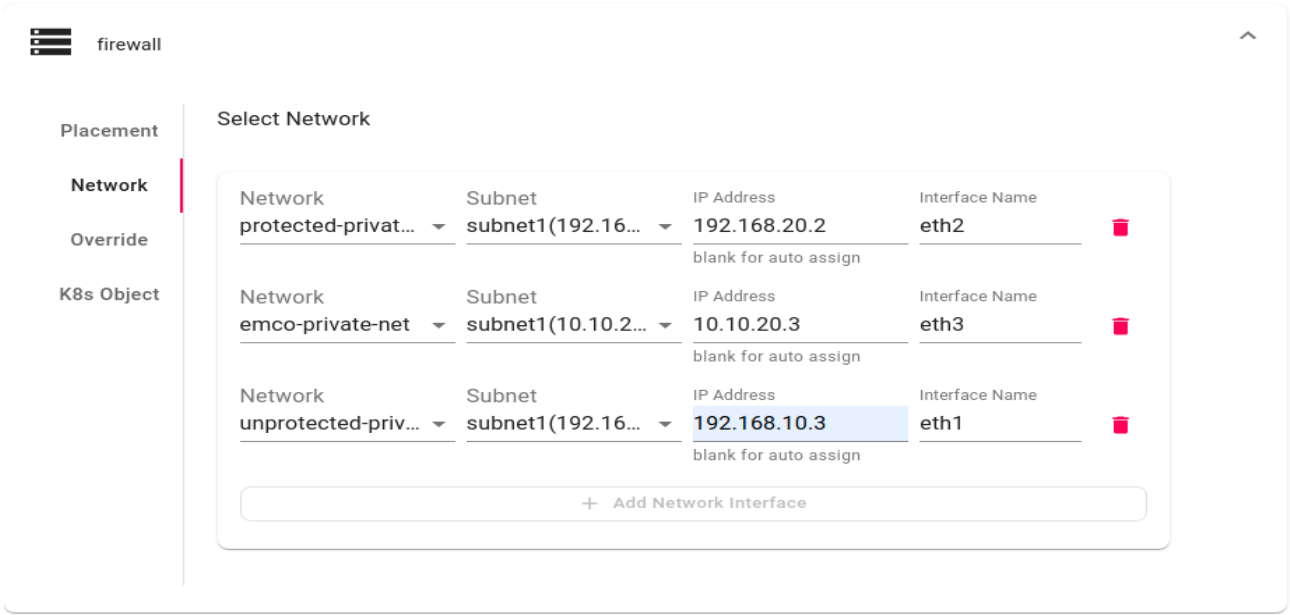
h. Note: This step is not required for vFW service. If you need to add networks for any of the applications, then you can refer to this step of providing network intents. Now click on the *network* tab and then click on *Add Network Interface*. Then select the network from the drop down menu and then the subnet. Leave the IP field blank to auto assign the IP address or fill in the IP address if required. Repeat this step to add more network interfaces.
For the vfw following network interfaces are to be added for each application,
    1. sink:
       protected-private-net, ip: 192.168.20.3
       Interface Name: eth1
       emco-private-net, ip: 10.10.20.4
       Interface Name: eth2



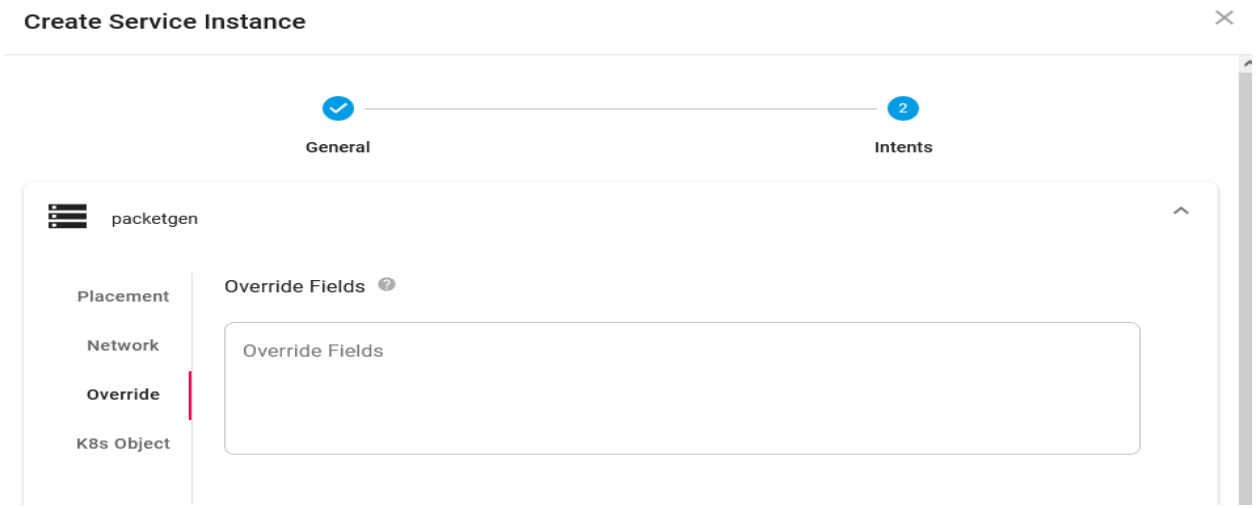
2. packetgen:
   unprotected-private-net, ip: 192.168.10.2
   Interface Name: eth1
   emco-private-net, ip: 10.10.20.2
   Interface Name: eth2



3. firewall:
   protected-private-net, ip: 192.168.20.2
   Interface Name: eth2
   emco-private-net, ip: 10.10.20.3
   Interface Name: eth3
   unprotected-private-net, ip: 192.168.10.3
   Interface Name: eth1

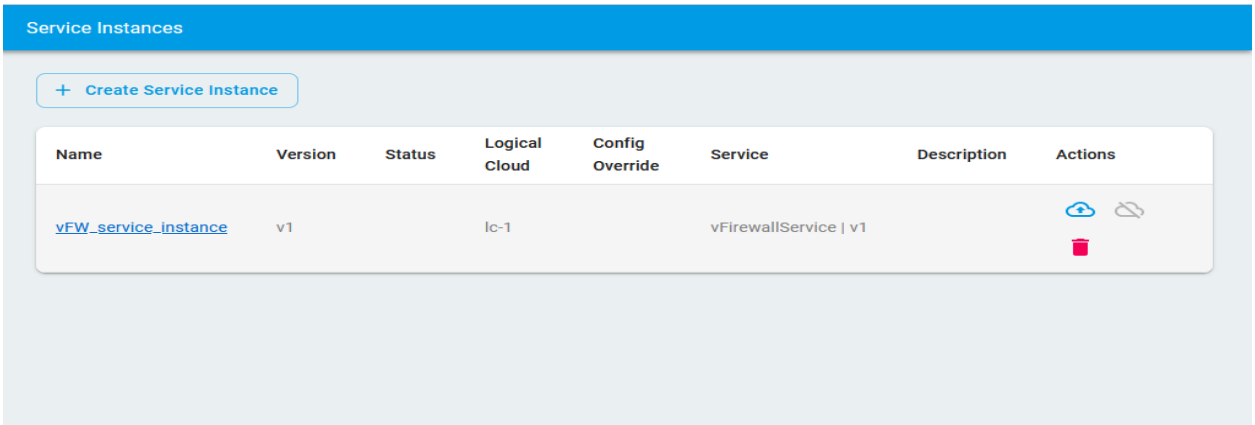


i. If there are any Day 0 override values for an application, they can be added in the *Override* tab. It is optional to add the override values. For documentation purpose following is the screenshot of how the override values can be added,

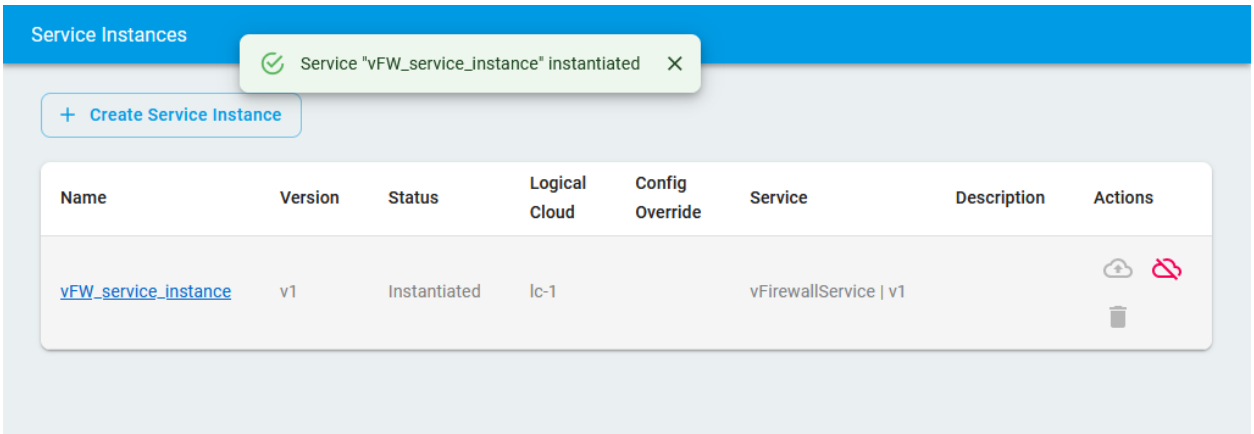


.

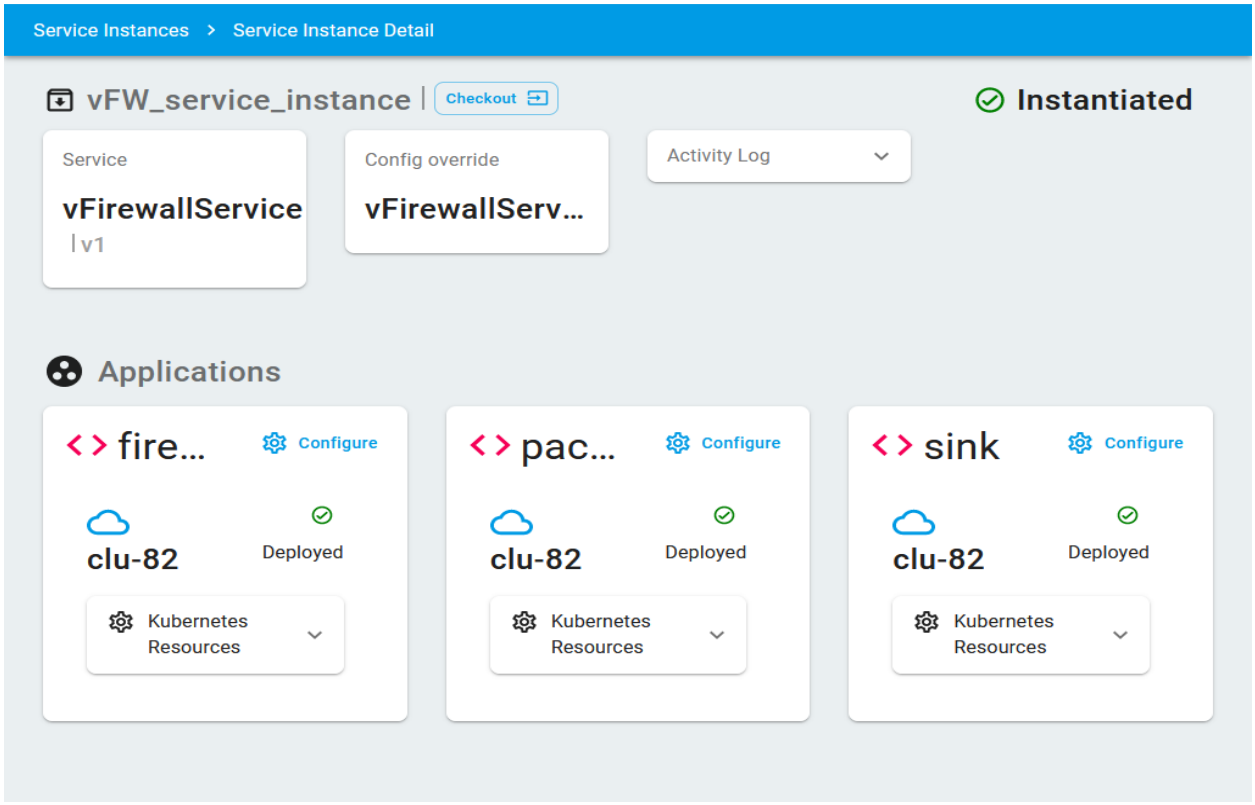
j. Now click on the submit button.
k. Now you can see the service instance. To instantiate the service instance, click on the instantiate button' ' in the actions column.



l. On successful instantiation, there will be a success notification at the top center of the screen.



m. You can click on the service instance name to look at the instance details and status. You can click on the activity log to see the activities on the service instance. Here you can see all the applications in the service instance and their deployment status per cloud. You can also see the Kubernetes resources of each application by clicking at the Kubernetes Resources tab under application widget.

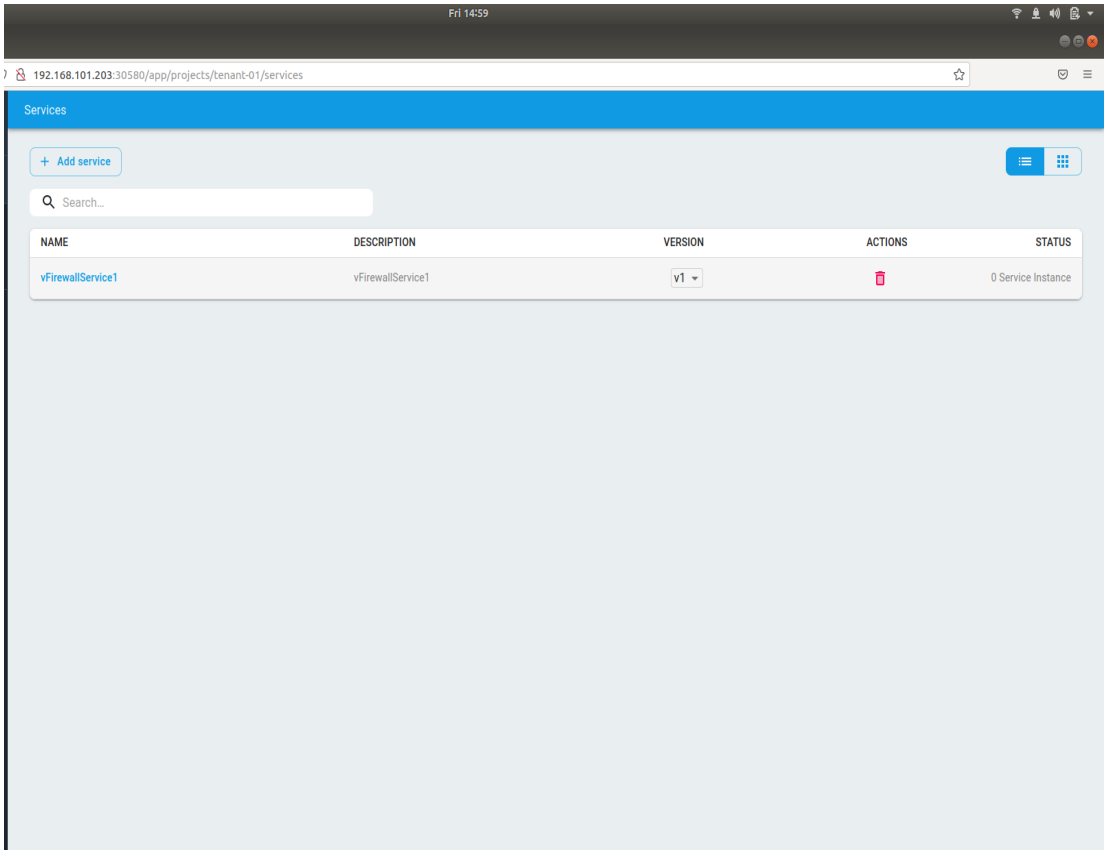


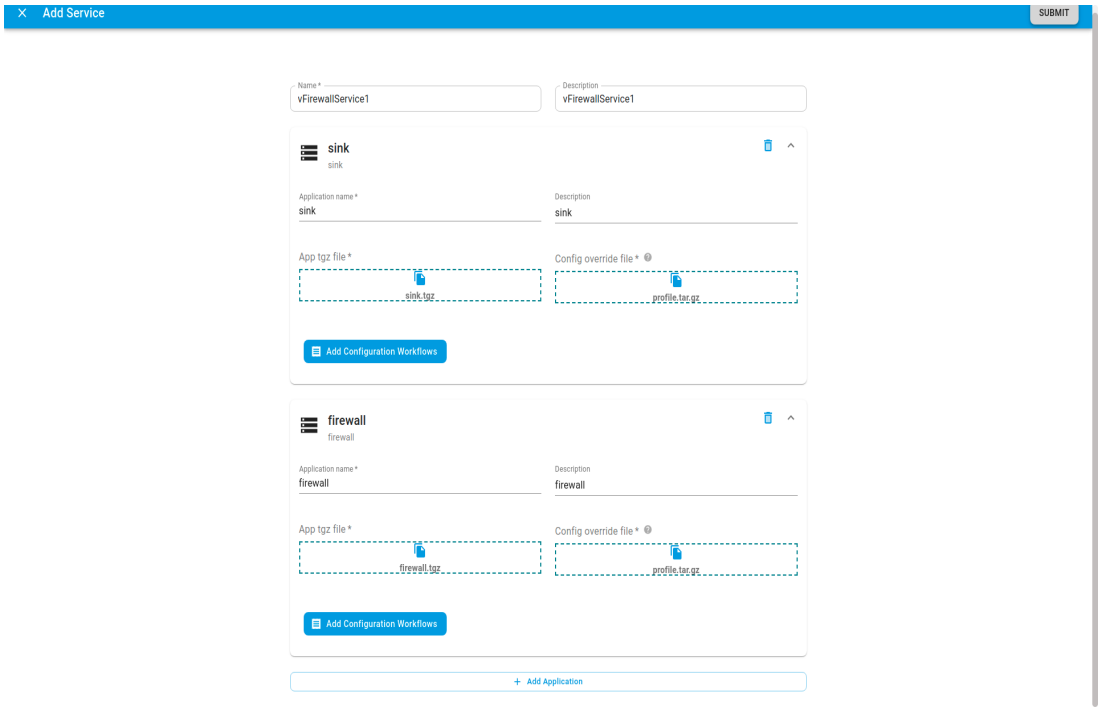
Service Instance Update

Service instance update feature helps you to update the placement intents or network intents in the running service instance.

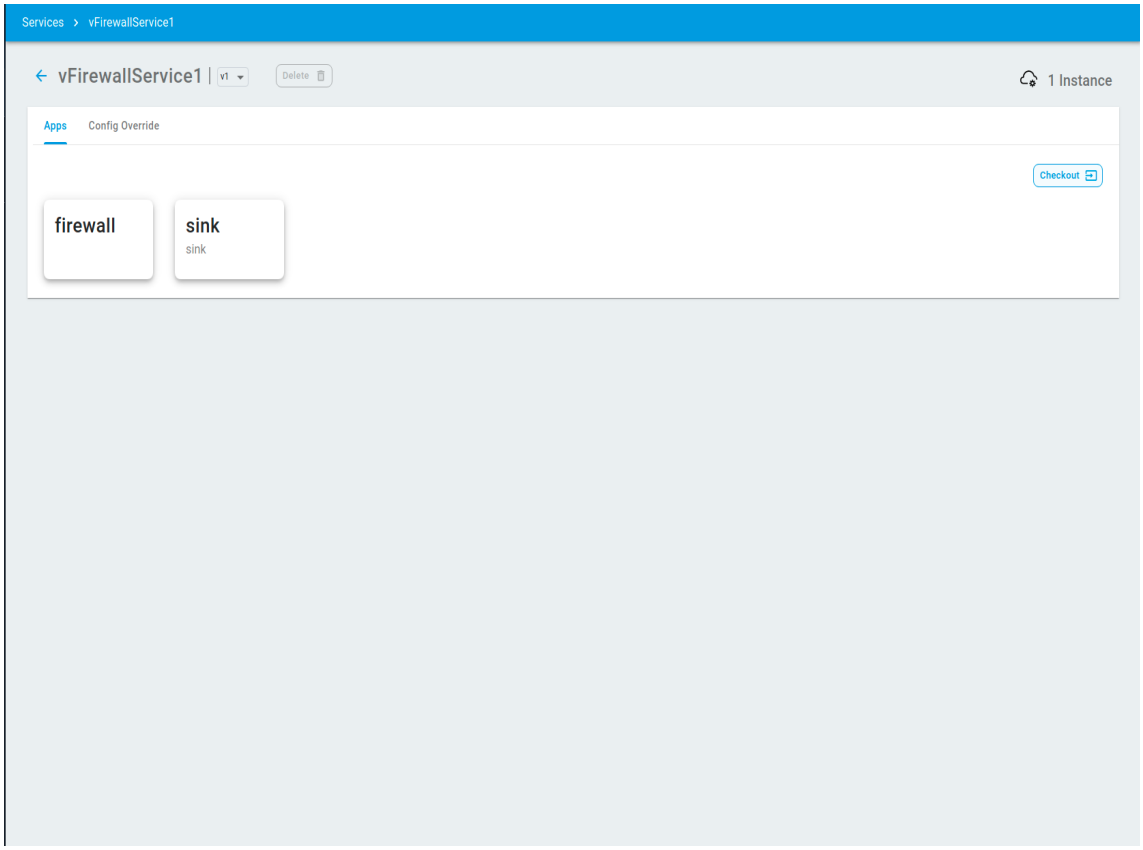
AMCOP supports updating an existing Service instance. Follow the steps given below to perform Service instance update of vFirewall application (shown as an example).

1. Create a vFW service with the name *vFirewallService1* with 2 applications (*sink* and *firewall* as shown below) as a version (v1) of the service.

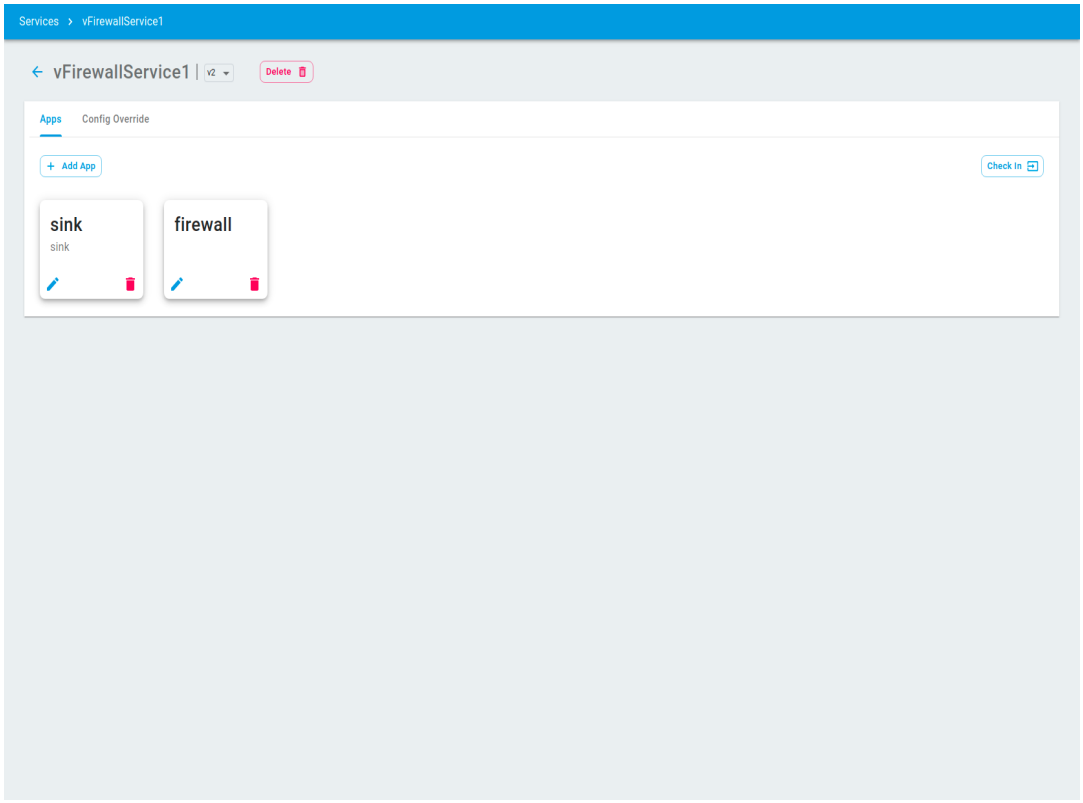

Fri 14:59
192.168.101.203:30580/app/projects/tenant-01/services
Services
Add service
Search...
NAME
DESCRIPTION
VERSION
ACTIONS
STATUS
vFirewallService1
vFirewallService1
v1
0 Service Instance

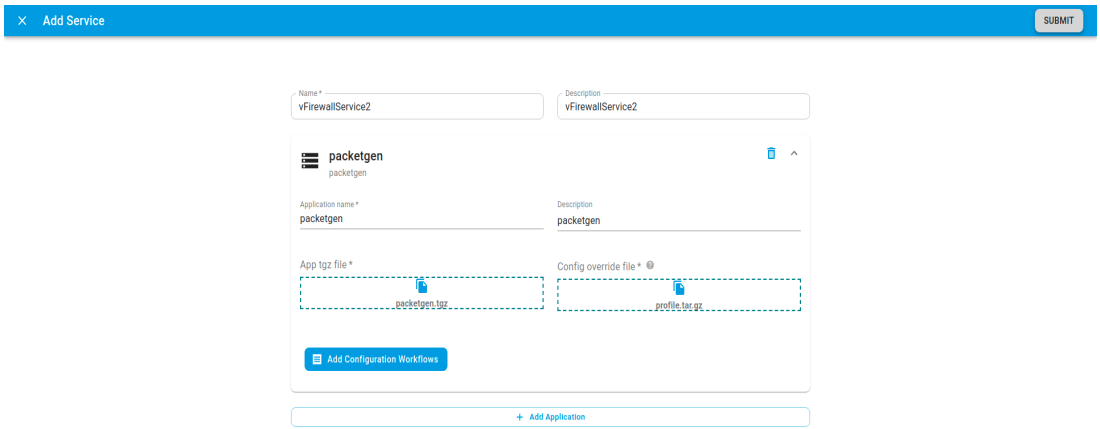

Add Service
SUBMIT
Name *
vFirewallService1
Description
vFirewallService1
sink
sink
Application name *
sink
Description
sink
App tgz file *
sink.tgz
Config override file *
profile.tar.gz
Add Configuration Workflows
firewall
firewall
Application name *
firewall
Description
firewall
App tgz file *
firewall.tgz
Config override file *
profile.tar.gz
Add Configuration Workflows
Add Application

2. As an update to the v1 service add one more app (say *packetgen*, as shown below) by selecting the *Checkout* option in the below screenshot.

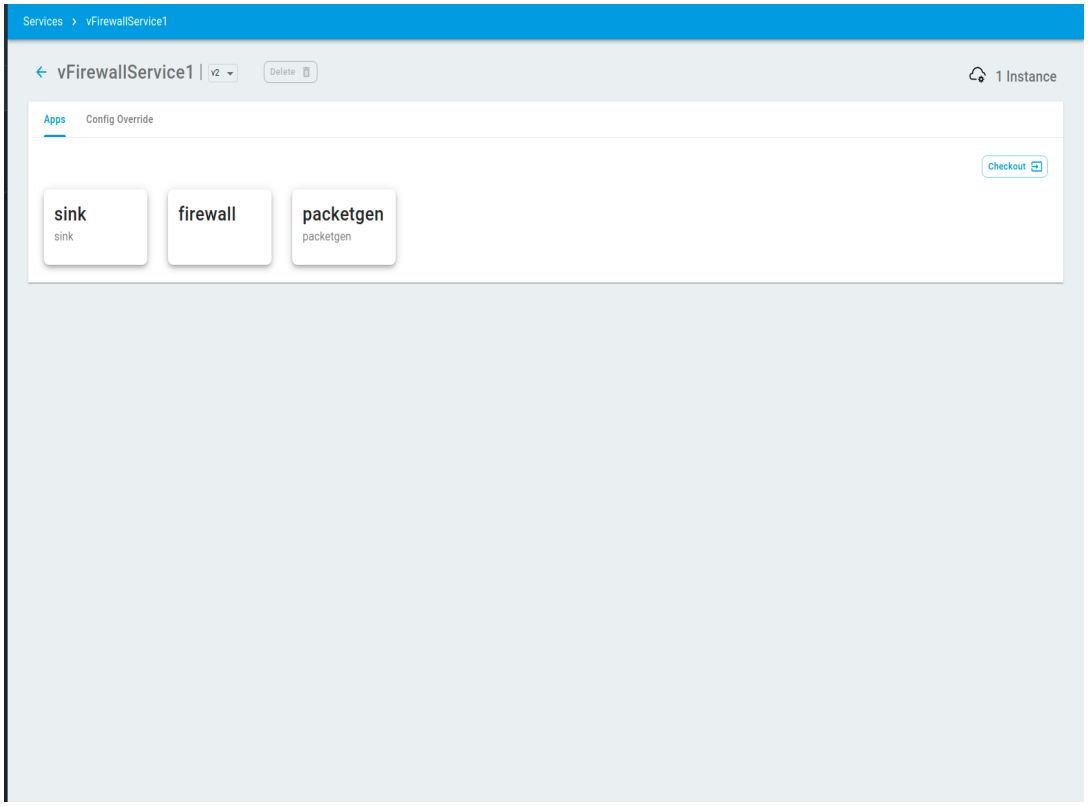


3. Select *Add App* to add the *packetgen* app and then select the Check-In option to update the service.

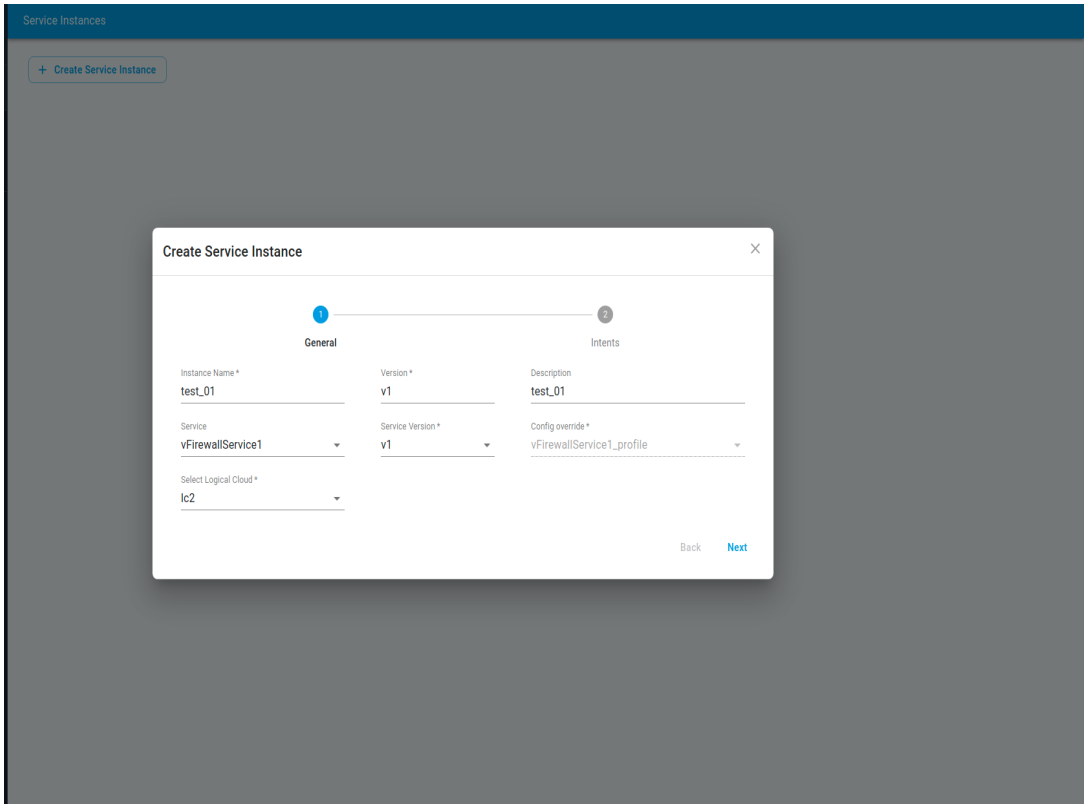




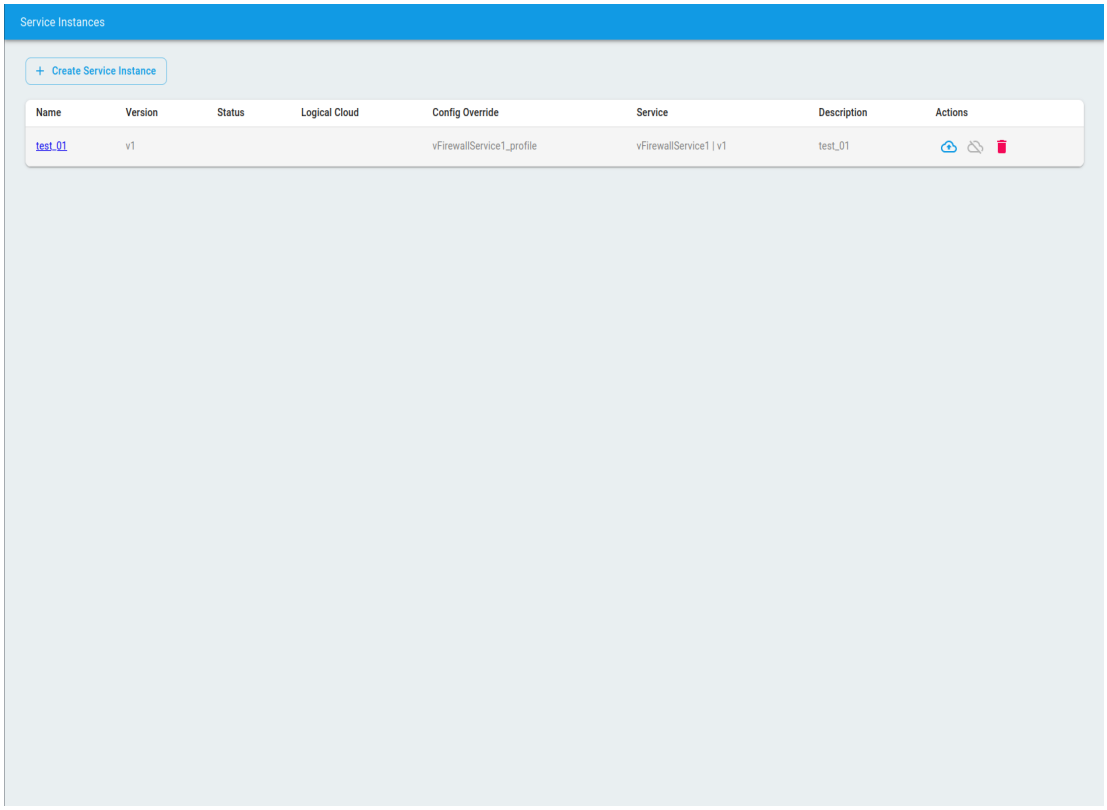
Then click on the submit button on the top right corner to update the changes as version v2 of the service.



4. From the left sidebar menu, go to the *service instances* and create a new service instance using the v1 service version as shown below.

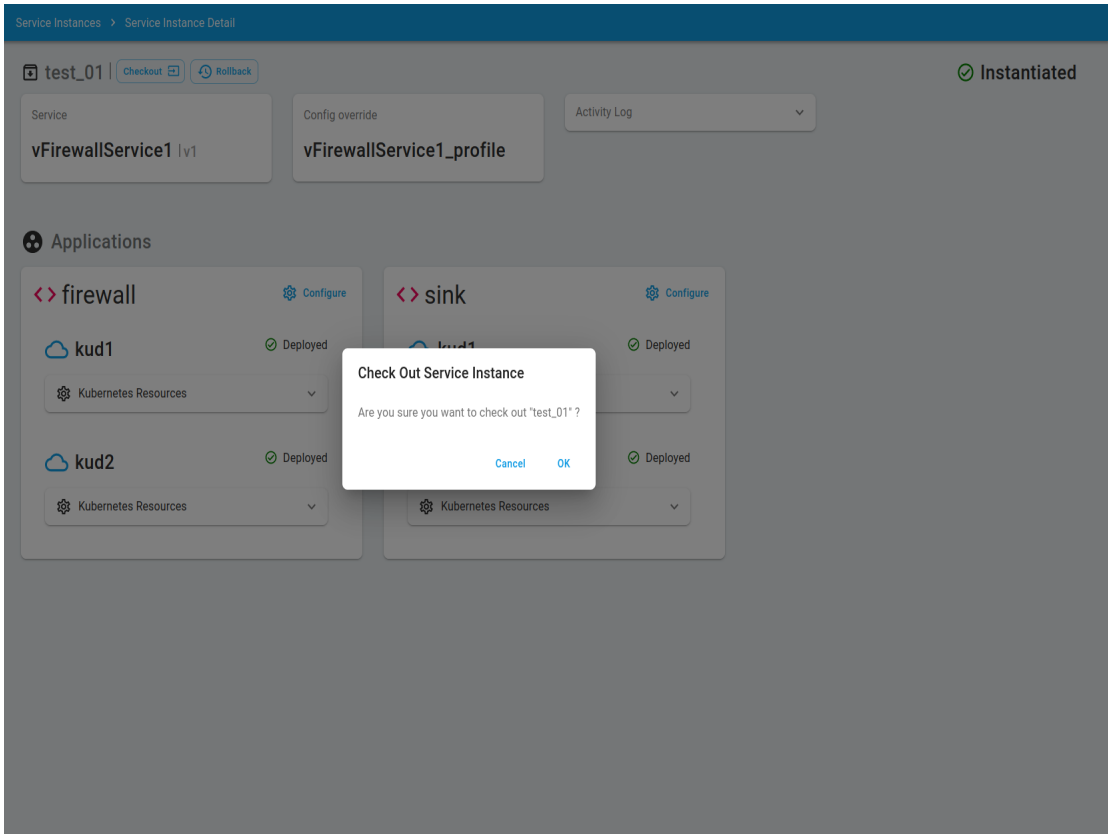


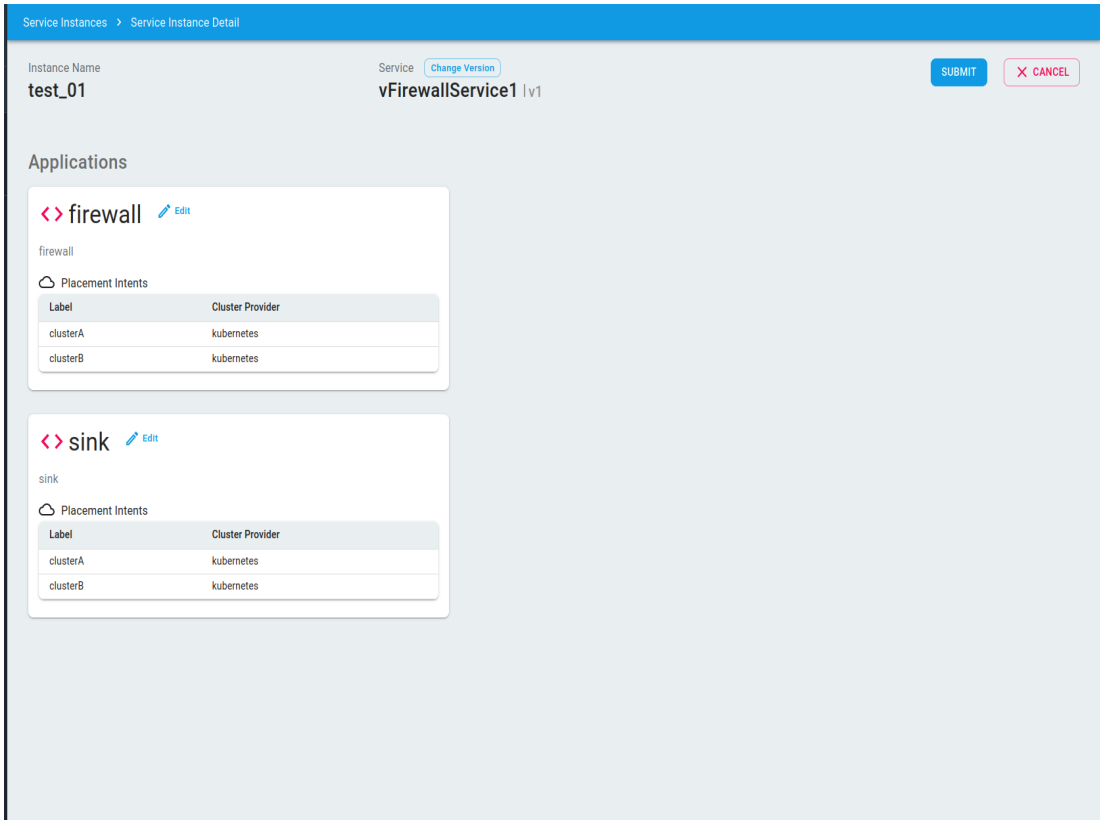
5. Complete the instantiation process as shown below.



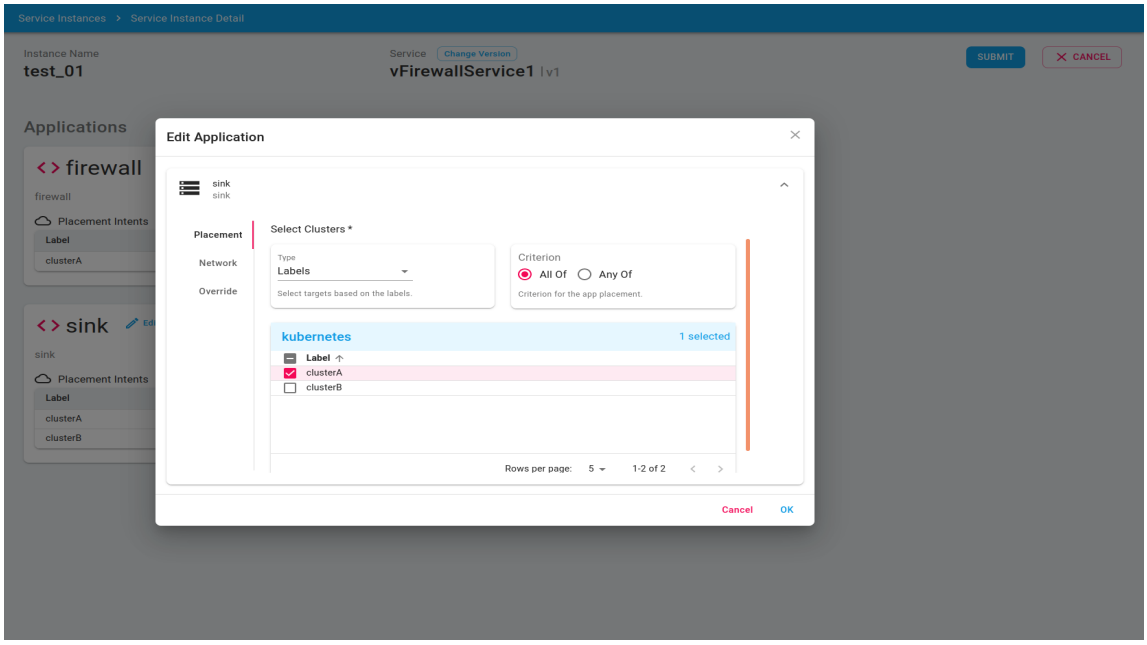
Service Instance "*test_01*" is now created using service "*vFirewallService1*" version v1.

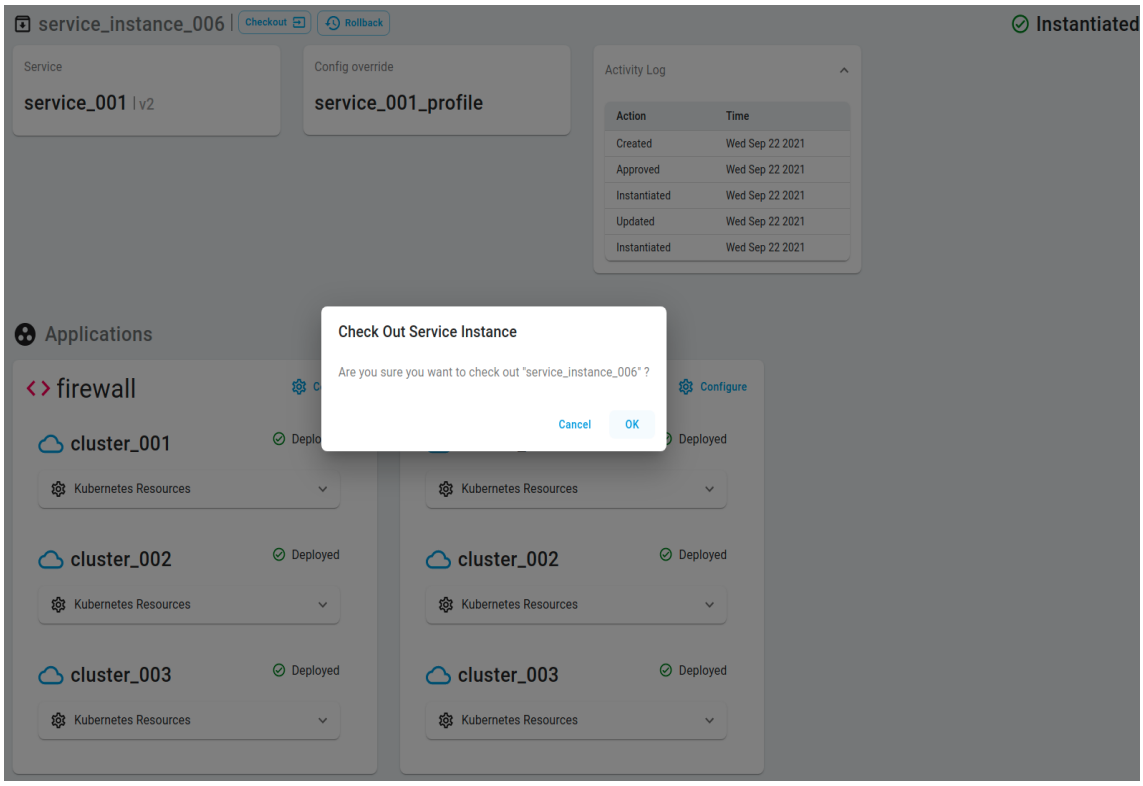
5. Select the *Service Instances* option from the left sidebar, Check out the service instance which was created in the above steps by clicking on the checkout button on the top left right *Checkout* button at the top, next to the service instance name *test_01* and select *ok*.


Service Instances > Service Instance Detail
test_01
Checkout
Rollback
Instantiated
Service
vFirewallService1 | v1
Config override
vFirewallService1_profile
Activity Log
Applications
firewall
Configure
kud1
Deployed
Kubernetes Resources
kud2
Deployed
Kubernetes Resources
sink
Configure
Deployed
Kubernetes Resources
Check Out Service Instance
Are you sure you want to check out "test_01" ?
Cancel
OK

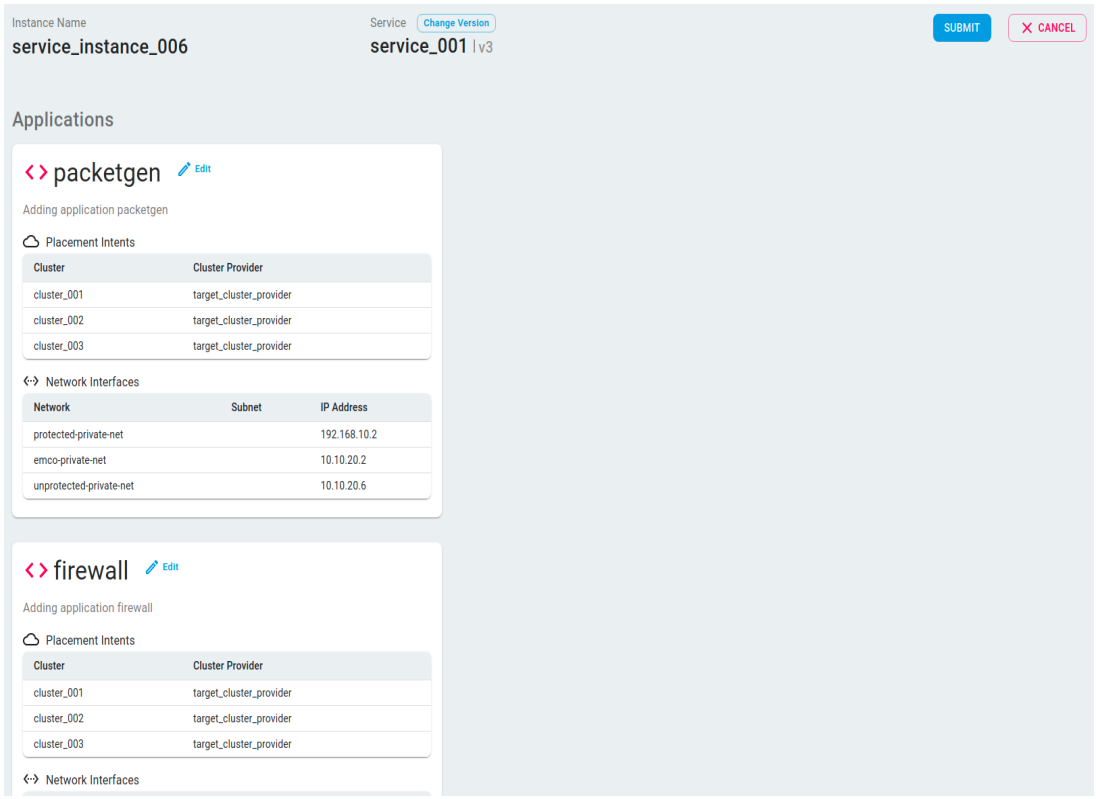

Service Instances > Service Instance Detail
Instance Name
test_01
Service
Change Version
vFirewallService1 | v1
SUBMIT
CANCEL
Applications
firewall
Edit
firewall
Placement Intents
Label
Cluster Provider
clusterA
kubernetes
clusterB
kubernetes
sink
Edit
sink
Placement Intents
Label
Cluster Provider
clusterA
kubernetes
clusterB
kubernetes

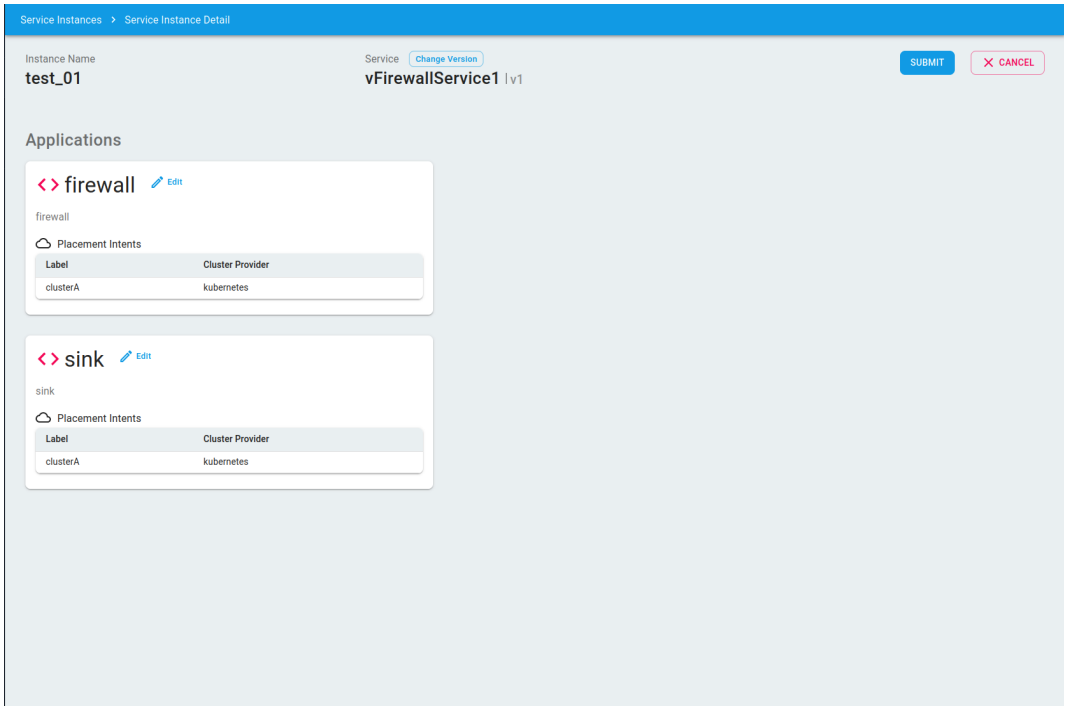
6. As part of the service instance update, Update placement intents of *firewall* and *sink* applications.

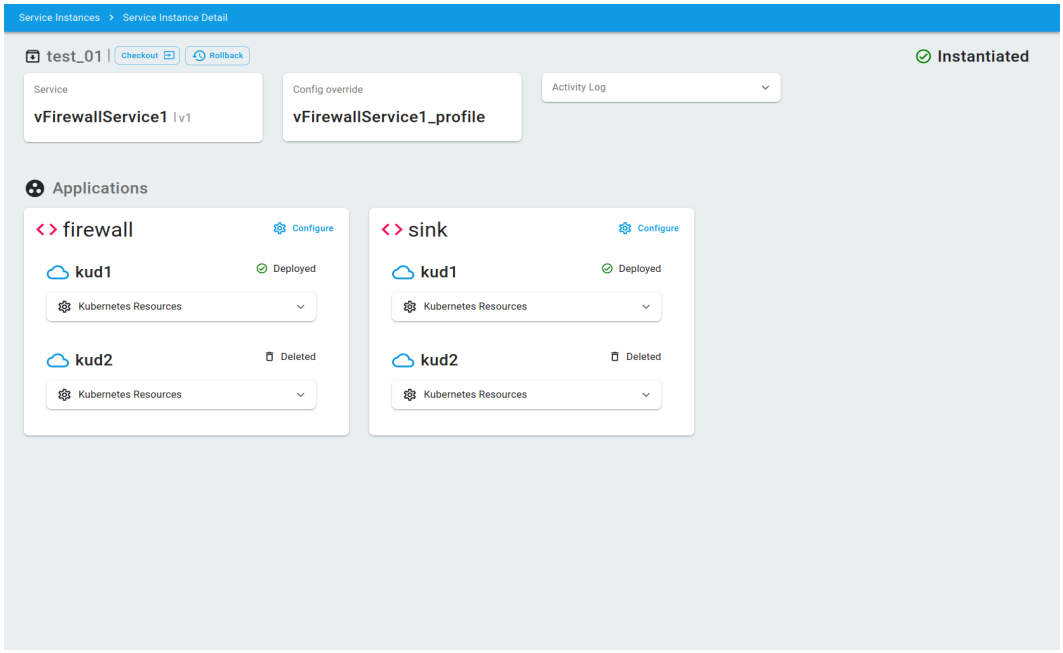




7. Click on the *Submit* button to ensure new placement intents are reflected on target clusters.

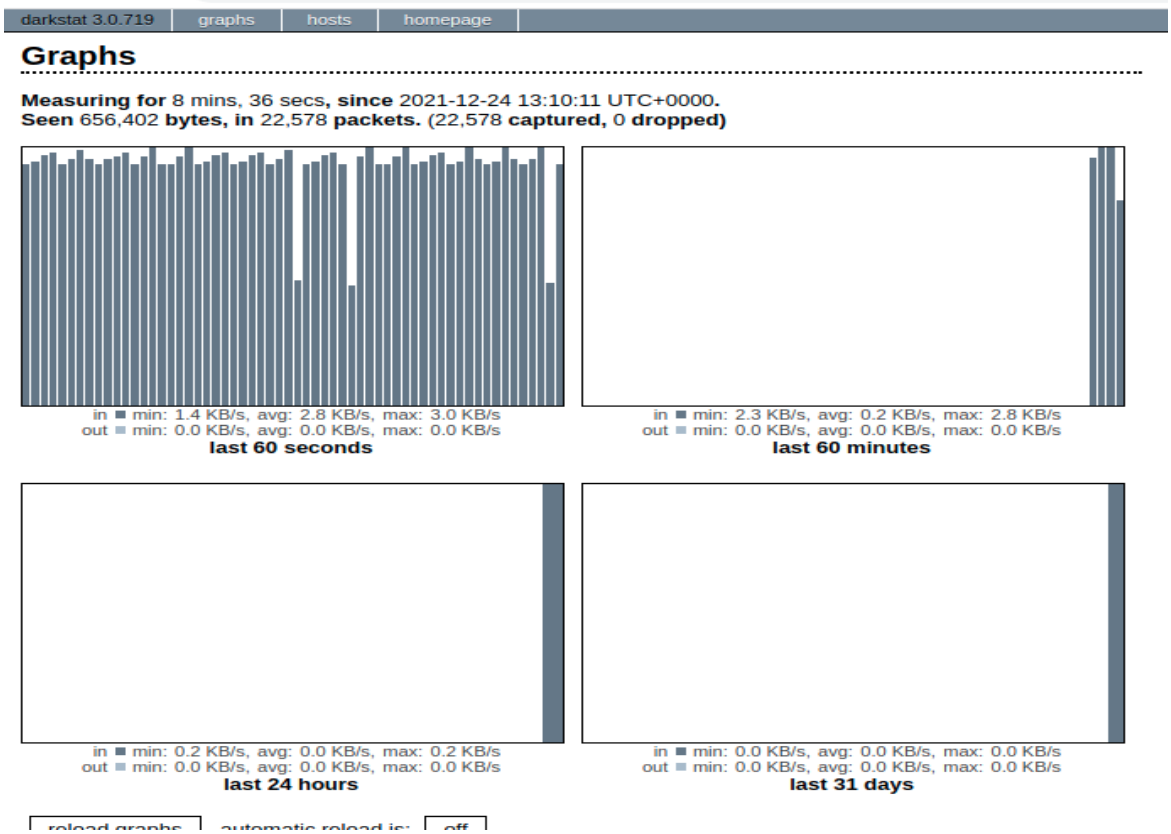






Note: Users shouldn't delete the network services created automatically under amcop-system tenant for any reason. Doing so might lead to AMCOP deployment unusable.

Vfw Service Traffic Output:

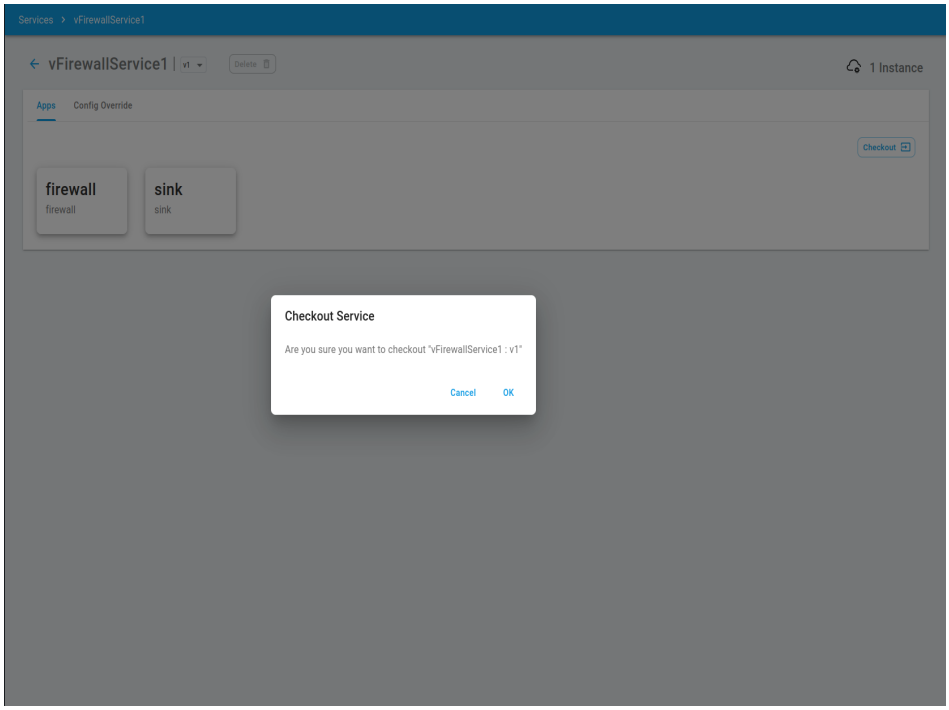


Service Instance Migration

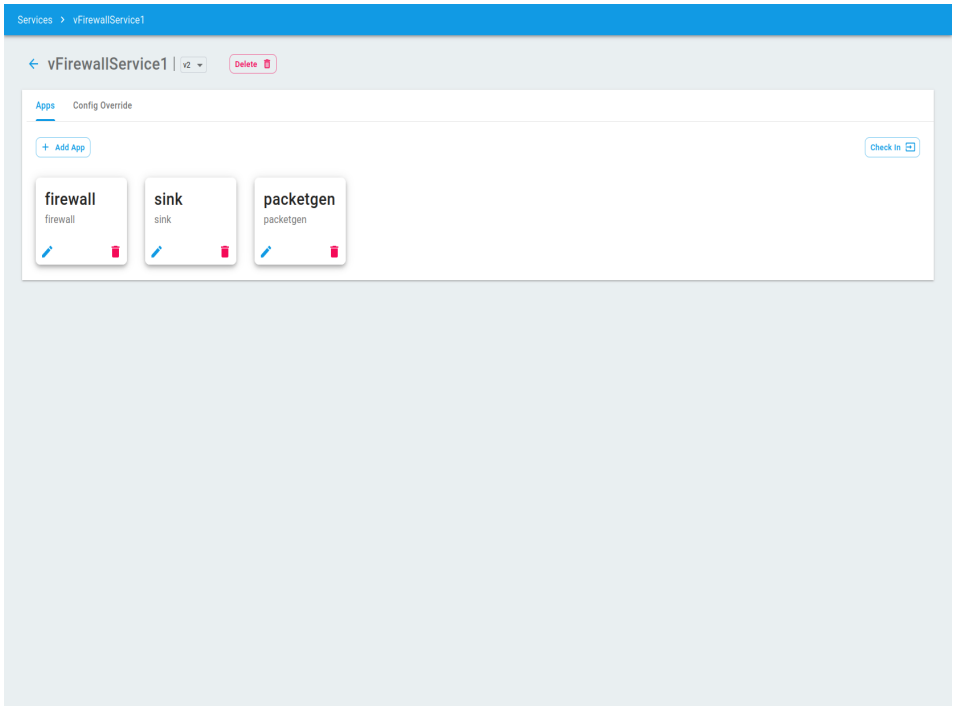
Service instance migration feature helps you to migrate to new versions (such as updated versions of the application helm charts, etc.,) or updated versions (such as adding a new application to the existing composite application) of the same service instance.

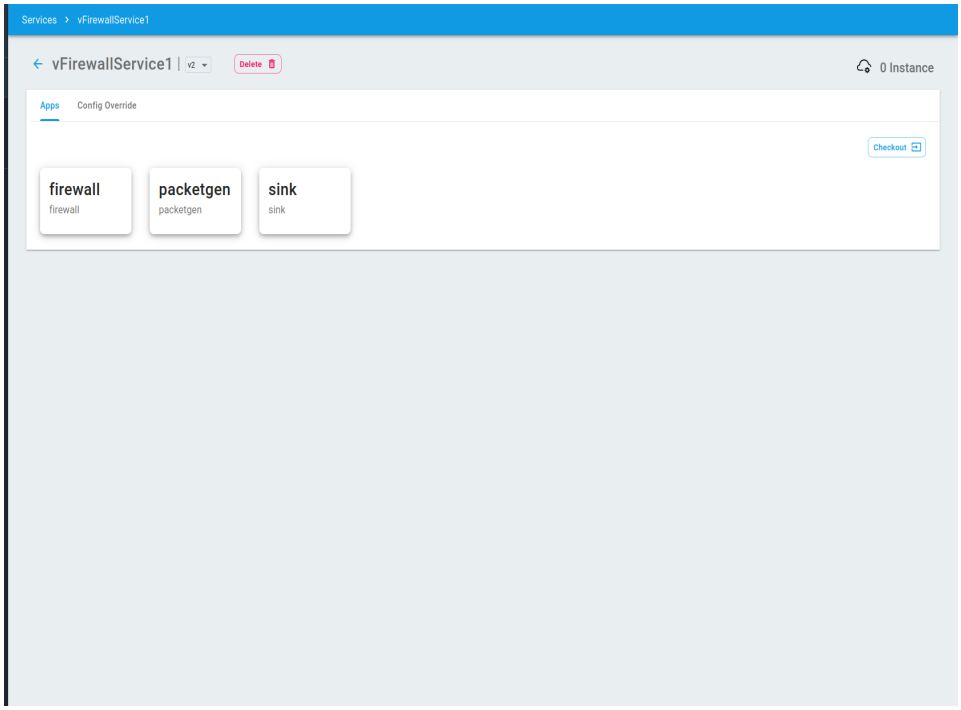
AMCOP supports service instance migration. For service instance migration, you need to first have a service (say vFW) with multiple versions as mentioned in previous sections of this document. You can follow the below steps to create multiple versions of vFW service.

1. Check out version v1 of *vFirewallService1* service with *firewall* and *sink* applications.

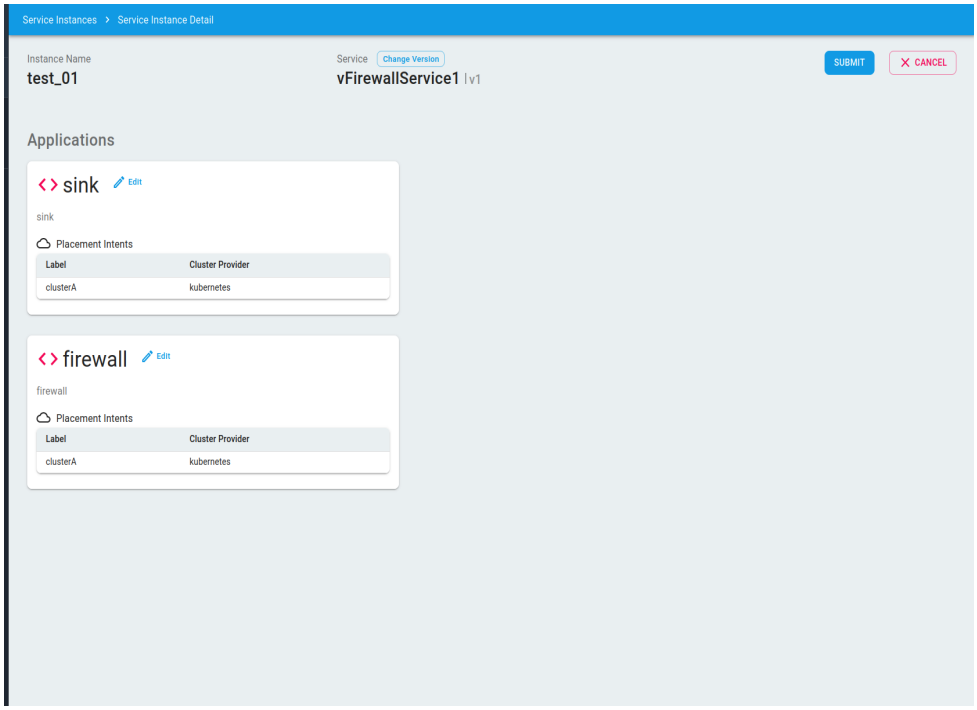


2. Create version v2 of *vFirewallService1* by adding a *packetgen* application to the existing *firewall* and *sink* applications.

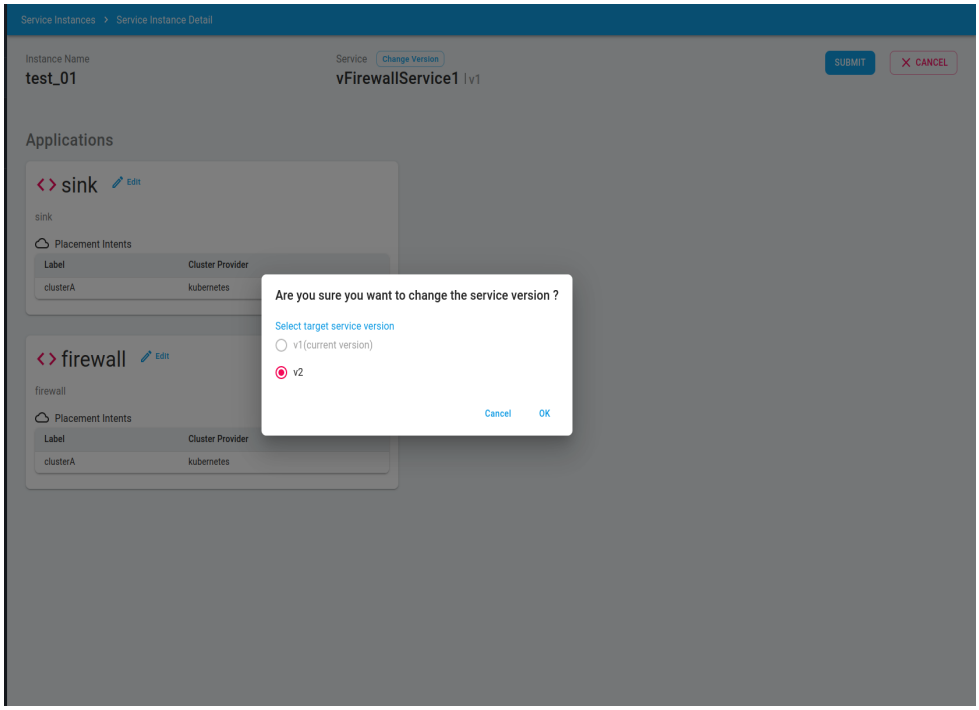




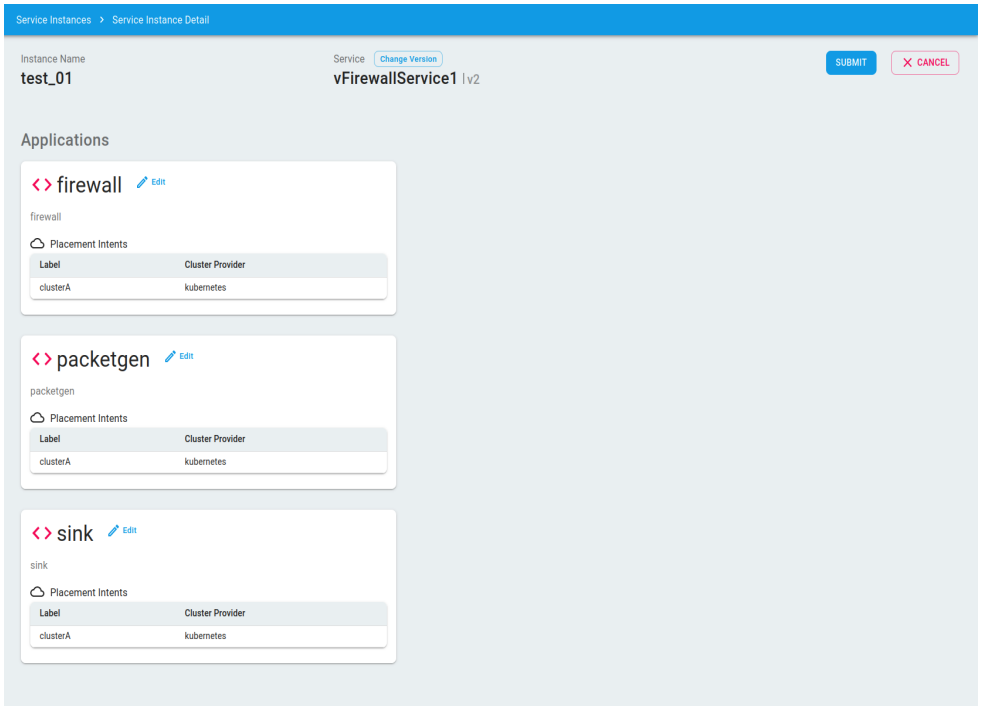
3. Select *test_01* service instance and perform a checkout.



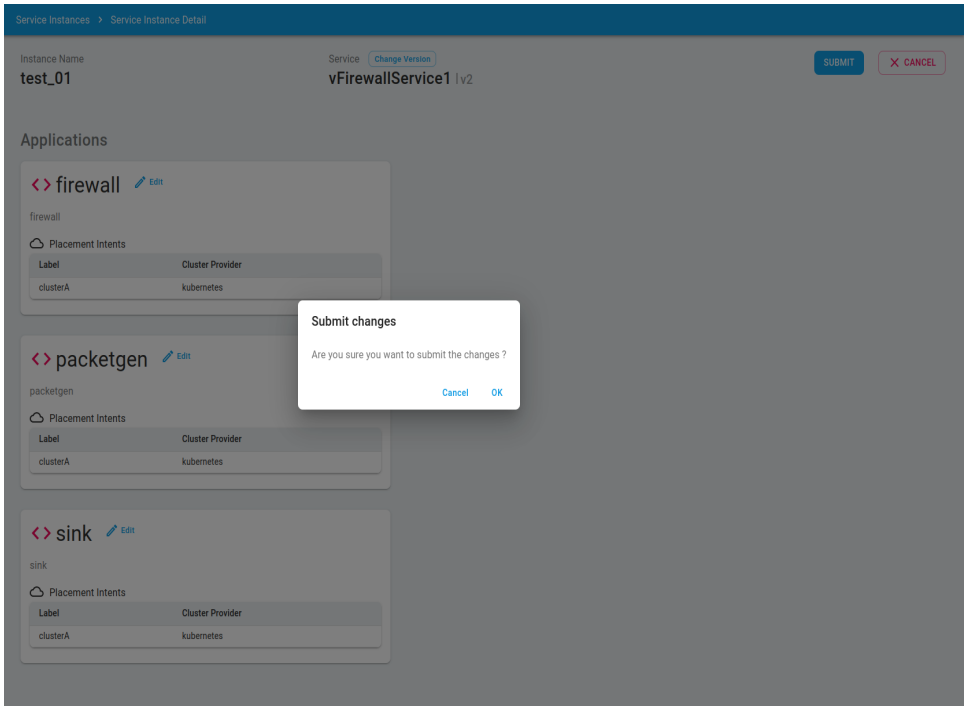
4, Click on the *Change Version* button for selecting version v2 of *vFirewallService1* to perform service instance migration. Then create a service instance, by selecting a version (say v2).

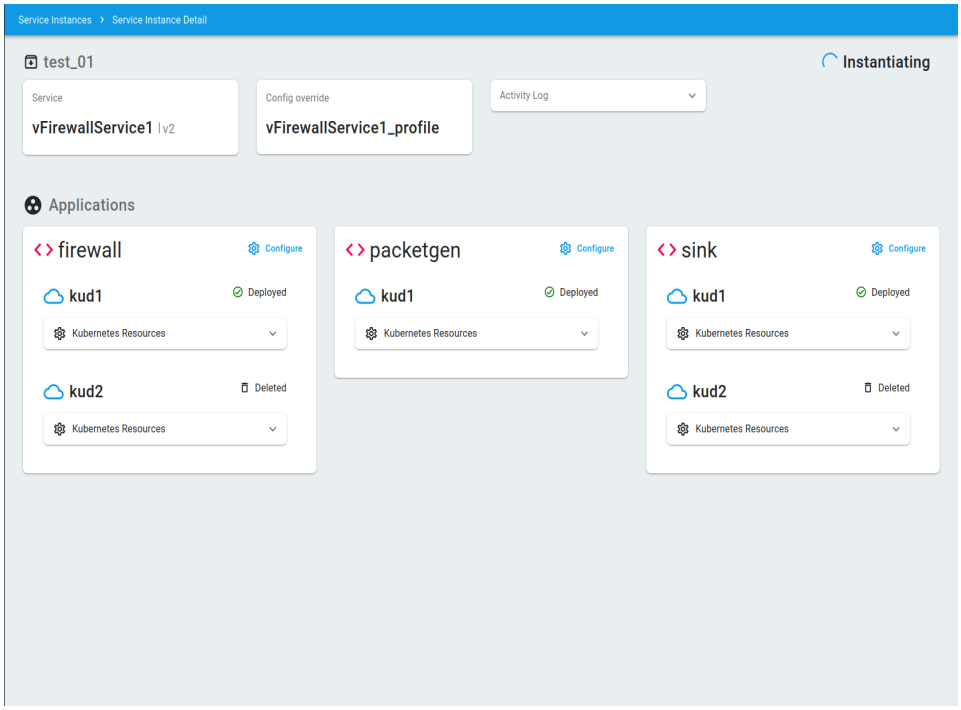


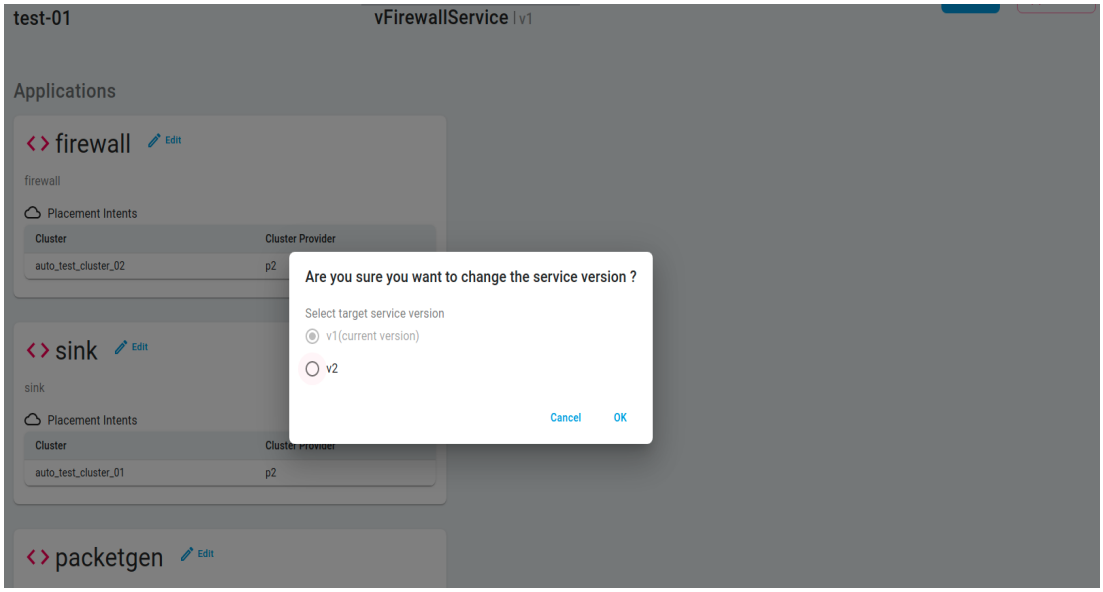
5. Select placement intent for *packetgen* application part of version v2 of *vFirewallService1,* Check out the service instance using a different version (say v2) for the service instance migration.



6. Click on submit button to complete the service instance migration.


Service Instances › Service Instance Detail
Instance Name
test_01
Service
Change Version
vFirewallService1 | v2
SUBMIT
CANCEL
Applications
firewall
Edit
firewall
Placement Intents
Label
Cluster Provider
clusterA
kubernetes
Submit changes
Are you sure you want to submit the changes ?
Cancel
OK
packetgen
Edit
packetgen
Placement Intents
Label
Cluster Provider
clusterA
kubernetes
sink
Edit
sink
Placement Intents
Label
Cluster Provider
clusterA
kubernetes


Service Instances › Service Instance Detail
test_01
Instantiating
Service
vFirewallService1 | v2
Config override
vFirewallService1_profile
Activity Log
Applications
firewall
Configure
kud1
Deployed
Kubernetes Resources
kud2
Deleted
Kubernetes Resources
packetgen
Configure
kud1
Deployed
Kubernetes Resources
sink
Configure
kud1
Deployed
Kubernetes Resources
kud2
Deleted
Kubernetes Resources



7. For service instance migration, you can switch from one version to the other whereas the update feature is used for making changes to the same version of the instance.

Logical Clouds

Logical Clouds are introduced to group and partition clusters in a multi-tenant way and across boundaries, improving flexibility and scalability.

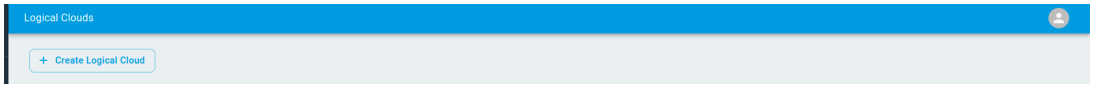
1. Admin Logical Cloud
2. Privileged Logical Cloud
3. User Logical Cloud

Note: User Logical cloud will be supported in the upcoming release.

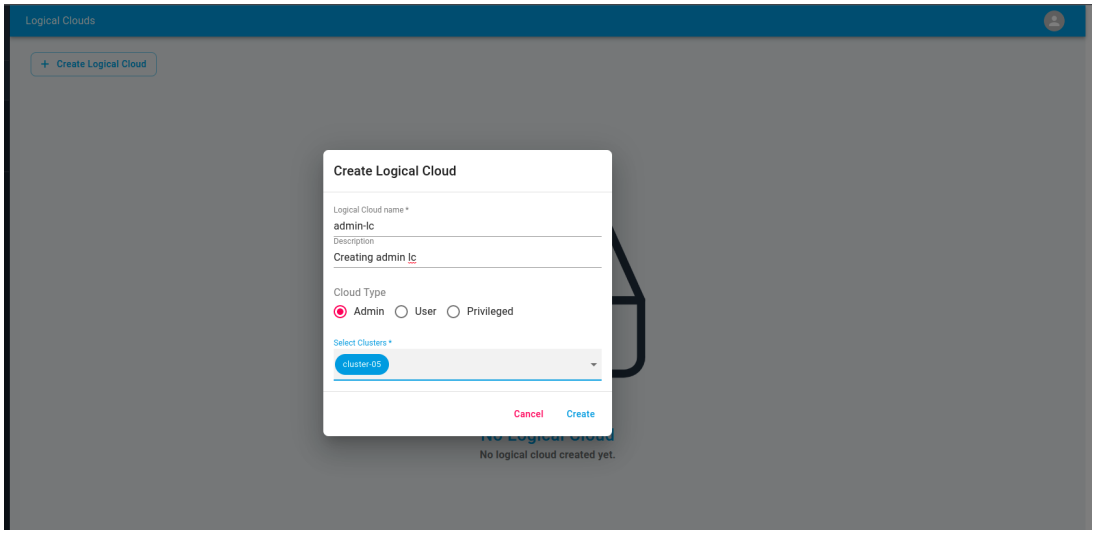
Admin Logical Cloud

In some use cases, and in the administrative domains where it makes sense, a project may want to access raw, unmodified, administrator-level clusters. For such cases, no namespaces need to be created and no new users need to be created or authenticated in the API. To solve this, the Distributed Cloud Manager introduces Admin Logical Clouds, which offer the same consistent interface as Standard Logical Clouds to the Distributed Application Scheduler. Being of type Admin means this is a Logical Cloud at the Administrator level. As such, no changes will be made to the clusters themselves. Instead, the only operation that takes place is the reuse of credentials already provided via the Cluster Registration API for the clusters assigned to the Logical Cloud (instead of generating new credentials, namespace/resources and kubeconfig files.)

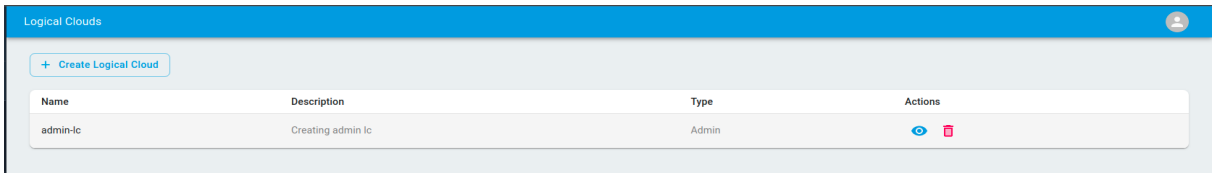
1. Click On Logical Clouds sidebar under tenant and click on Create Logical Cloud.



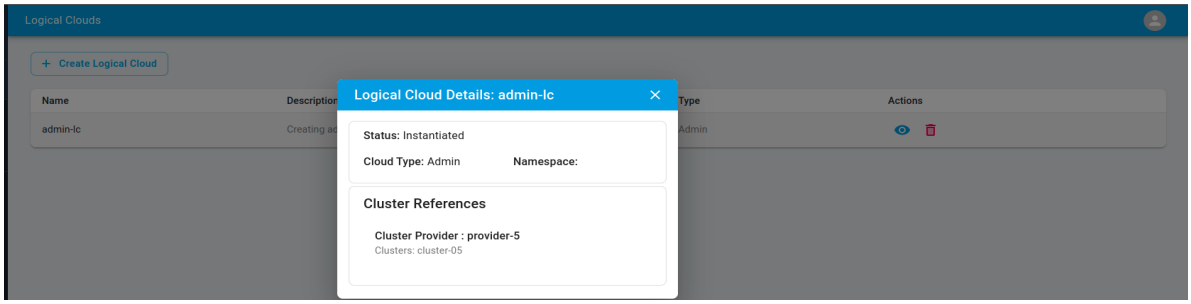
2. Enter all the required details for admin logical cloud, Select the provider and cluster.
   Note: Logical cloud name should not exceed more than 20 char and special Characters are not allowed Example names are(admin-lc, admin-lc-01 etc)



3. Make sure the logical cloud gets created successfully.



4. Click on the info icon to validate information.



Privileged Logical Cloud

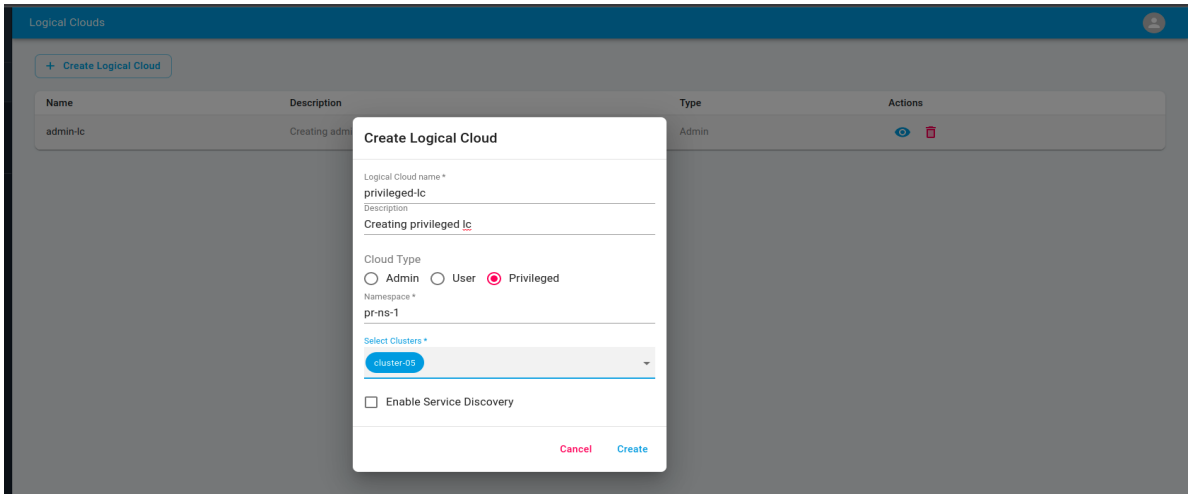
This type of Logical Cloud provides most of the capabilities that an Admin Logical Cloud provides but at the user-level like a Standard Logical Cloud. New namespaces are created, with new user and kubeconfig files. However, EMCO module can now request an enhanced set of permissions/privileges, including targeting cluster-wide Kubernetes resources.

1. Click On Logical Clouds sidebar under tenant, Click on Create Logical Cloud button and choose Privileged Option and enter the required details.

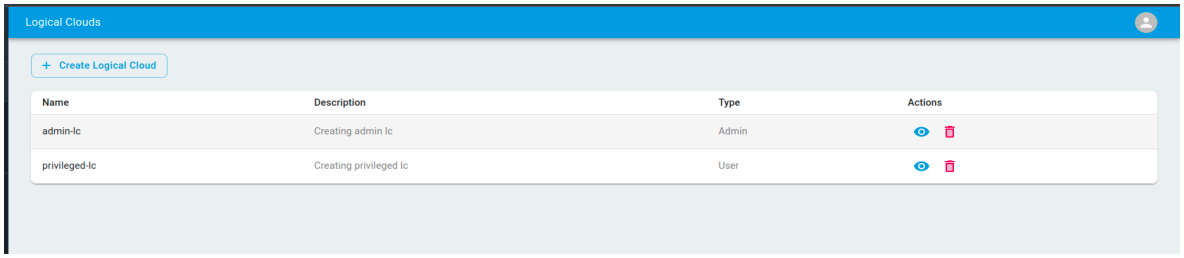
   Note: Logical cloud name should not exceed more than 20 char and special Characters are not allowed Example names are(admin-lc, admin-lc-01 etc).

   Namespace should not exceed more than 20 char and special characters are not

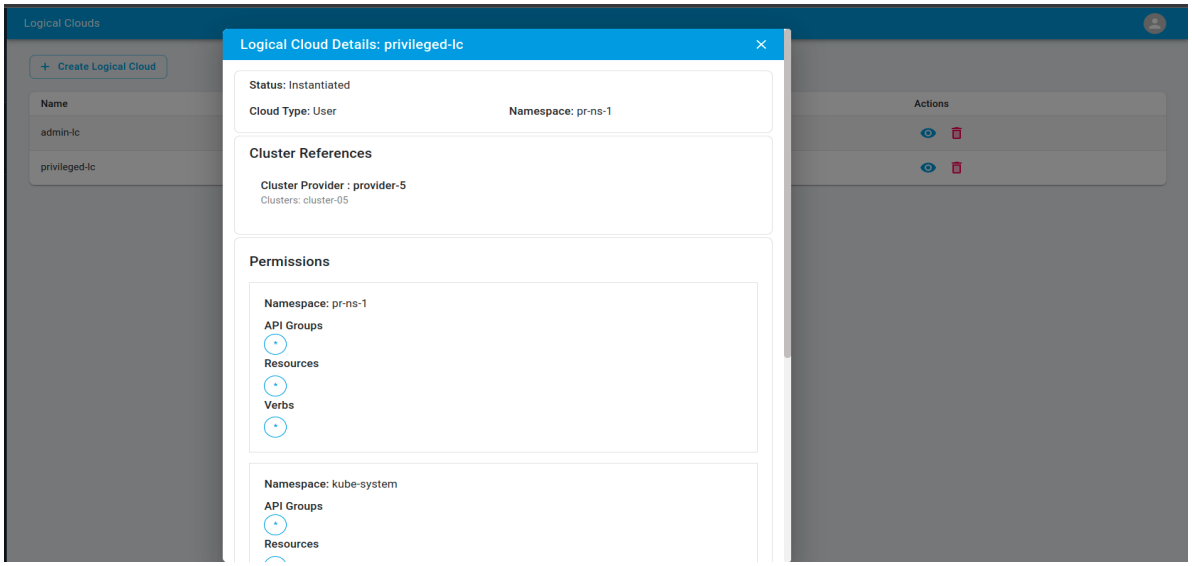
   allowed Example names are(privi-ns, test-ns, test-ns-01 etc ..)



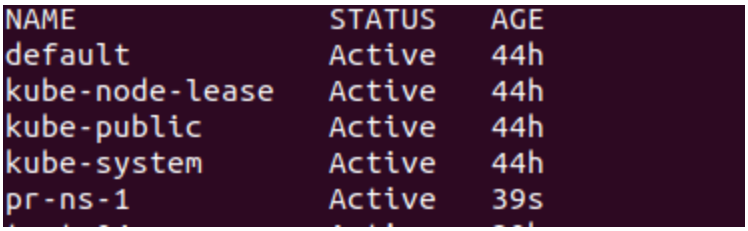
2. Make sure the Privileged logical cloud gets created successfully.



3. Click on the info icon to validate the details and make sure Permissions for API Groups, Resources, Verbs of Namespace (user and kube-system) and Cluster wide should be (*).



4. ssh to target cluster and check the ns gets created successfully by running cmd: kubectl get ns



Orchestration of Free5GC using AMCOP GUI

This section shows how to register a k8s cluster with AMCOP, design a network service (Free5GC) and orchestrate them using AMCOP GUI.

After setting up the SOCKS tunnel to the AMCOP Jump host (as described above), and setting up a proxy in Firefox browser, the AMCOP GUI can be accessed at:

http://<amcop-master-vm-ip>:30661

If AMCOP is deployed on a GKE or AKS cluster then the IP address and port numbear are different.

The user interface of AMCOP GUI is divided into two parts. One is for admin related functionalities like adding tenants, onboarding clusters, adding k8s controllers and the other for the service designer related functionalities like Creating service, instantiating service.

In this section we are going to deploy free5gc using AMCOP GUI.

Before service design and instantiation of Free5GC, you need to setup the target environment as mentioned in section Setting up Free5GC and UERANSIM simulator environment

Admin User

Once the GUI is launched, the tenants page will be displayed.

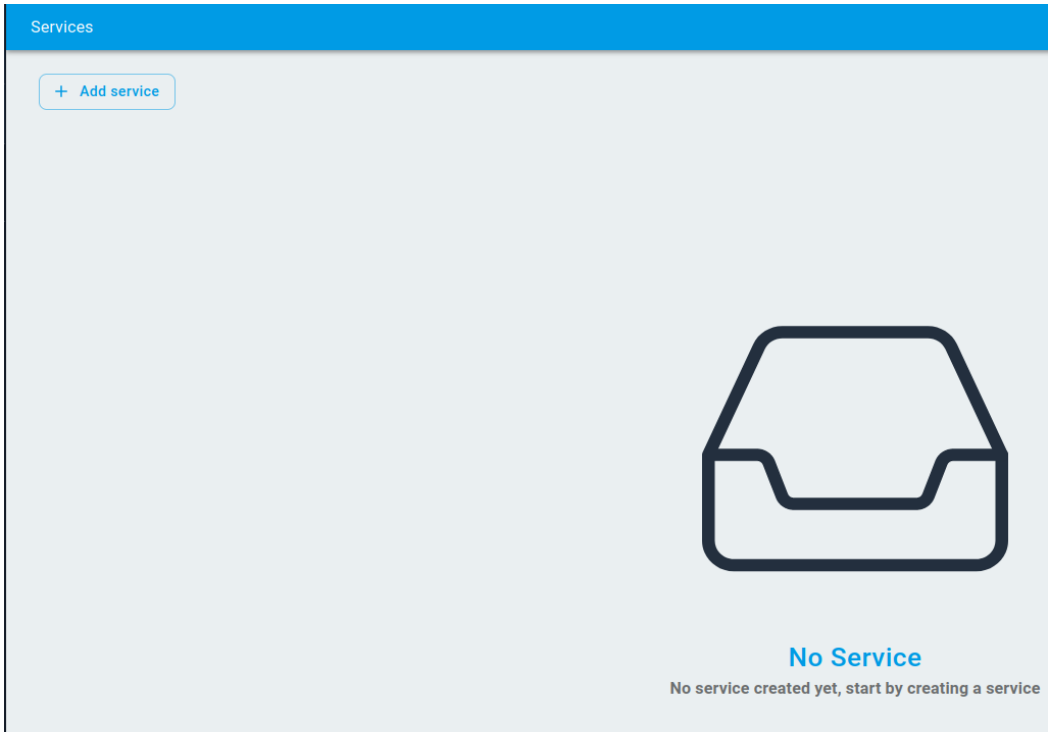
Refer to the steps mentioned in the section Orchestration of vFirewall using AMCOP GUI, to create Tenants, register controllers and onboard a cluster.

If these are already performed, there is no need to repeat these steps.

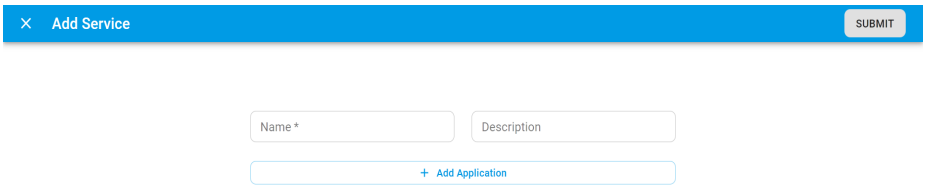
Note: Please make sure the rsync controller is registered, since this is necessary for Free5gc.

Service Designer User

To go to the service designer page, click on the Tenants tab and then click on the name of the tenant from the tenants table.



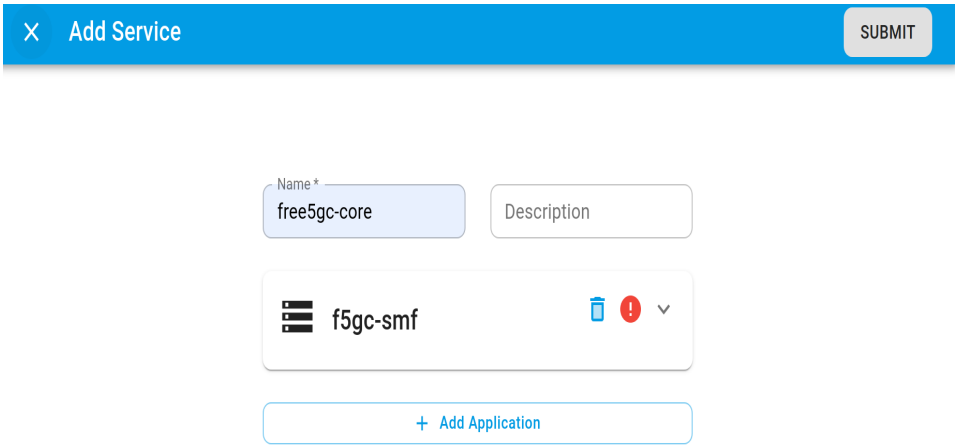
1. Add a Service
    a. Once the service designer view is opened, click on the *Add Service* button to add a new service.
    b. Fill in the basic information like name and description and then click on *Add Application* to add an application to the service.



    c. In the form to add the application, fill in the name and description of the application and click *Create*.

NOTE : Application name should match the helm chart name in the package. For example, in the case of free5gc the app name should be f5gc-smfas the helm chart is f5gc-smf-0.1.0.tgz.

d. Once create is clicked an application row will be added in the service form as shown below. At this point, the app form is not complete yet so there will be a red exclamation mark. Until the form is valid, the submit button will be disabled.



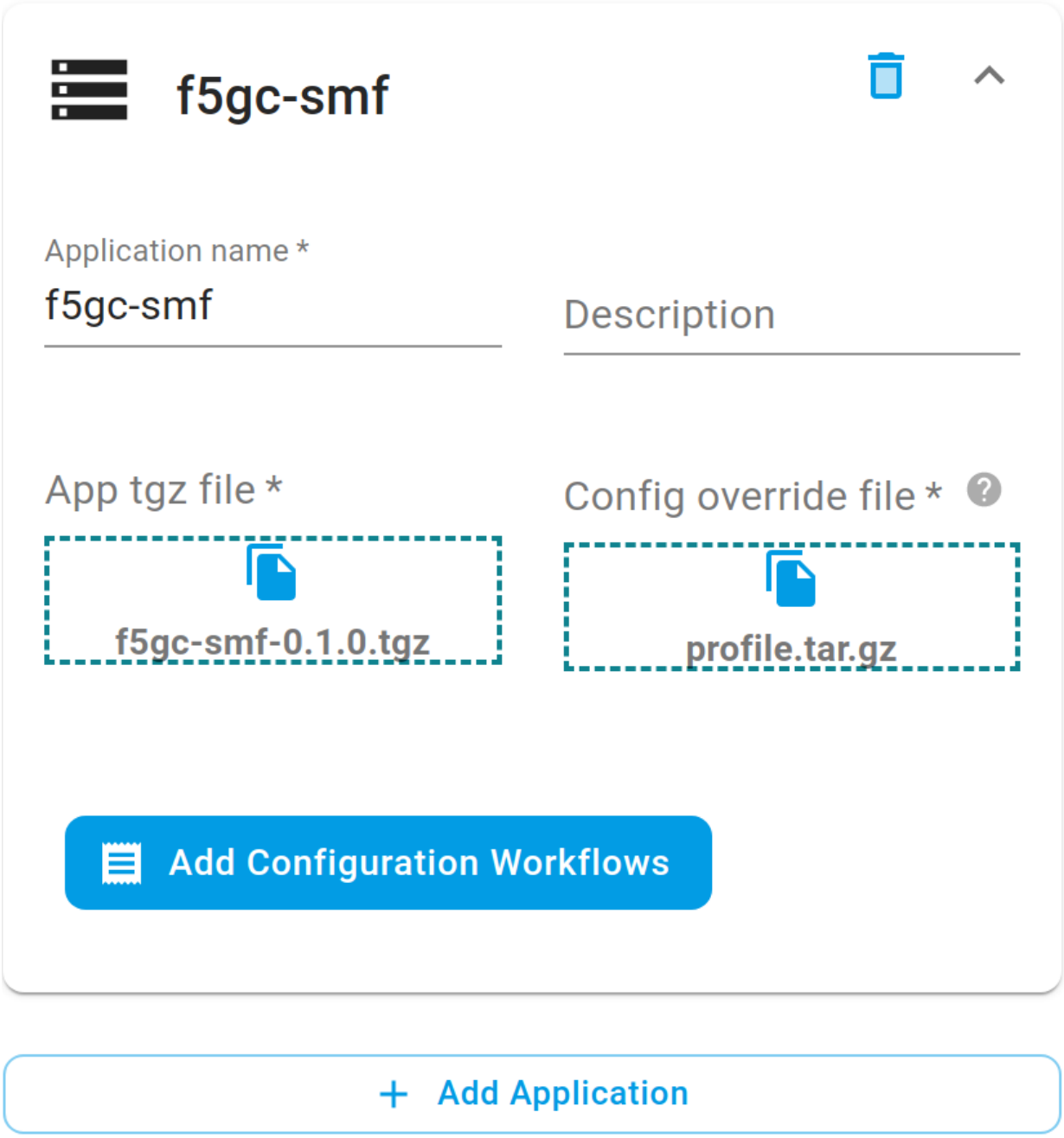
e. Click on the service row to expand it. Now upload the app .tgz package file and profile .gz file which contains the override files by clicking on the upload button or dragging and dropping in the upload area. App name and description can also be changed here.

Note: The helm charts for the free5g are present in /home/<user>/free5c_deploy directory. The name of the bundle is f5gc-smf-0.1.0.tgz

Similarly we need to perform above step for other core services

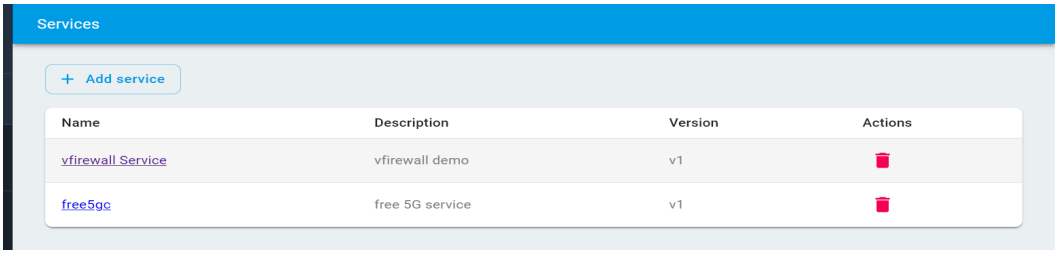
Note: Before service design and instantiation of Free5GC, you need to set up the target environment as mentioned in the section Setting up Free5GC and UERANSIM simulator environment

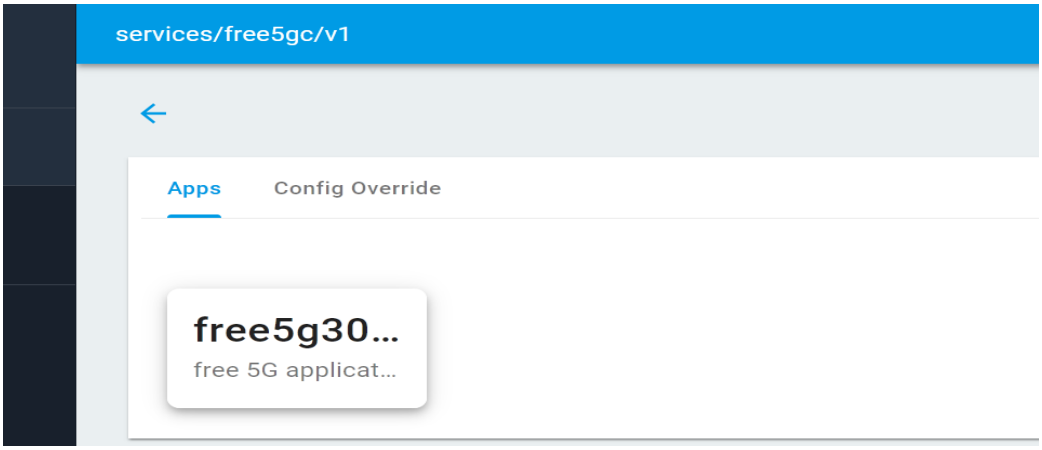
f. The profile bundle is present in /home/<user>/free5c_deploy. The name of the bundle is profile.tar.gz



g. Once the application has been added, click on the submit button in the top right corner of the screen to submit the service design as shown below.

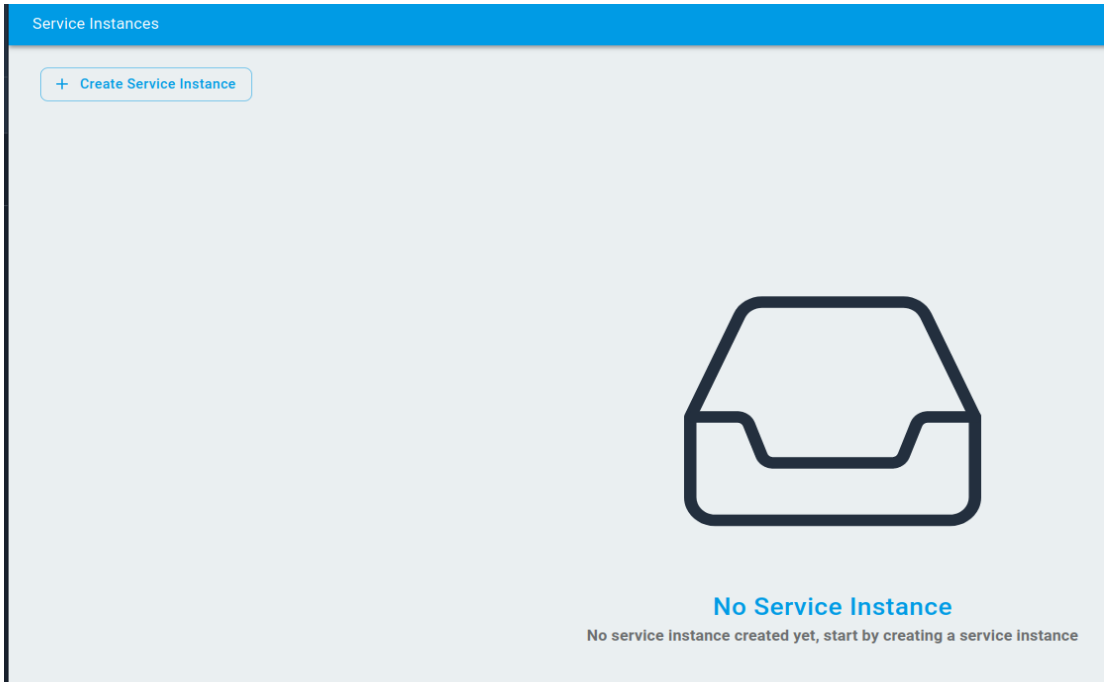
h. Now once the service is created, it will appear on the services page. We can look at the details of the service by clicking on the name of the service in the services table.



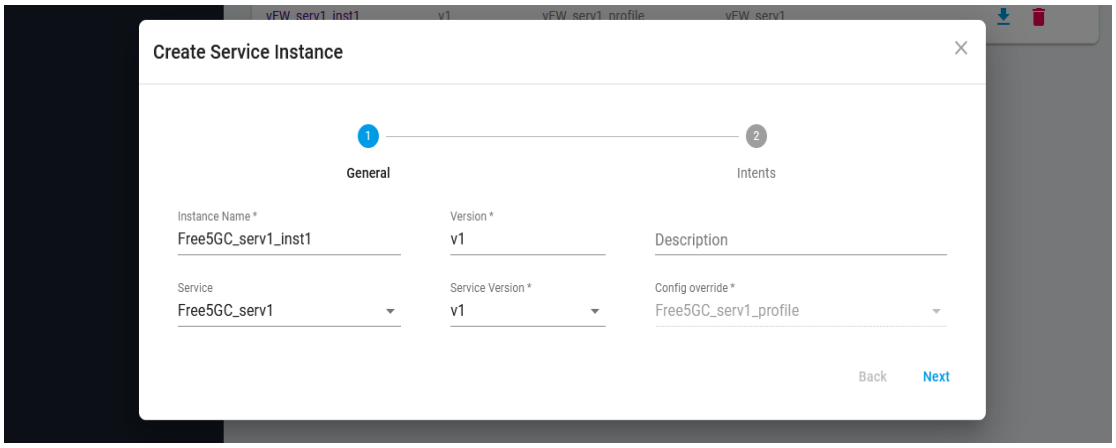


2. Create a service instance.

    a. To create a service instance, go to the service instances screen from the left hand side navigation and then click on *create service instance* button, this will open the service instance form.

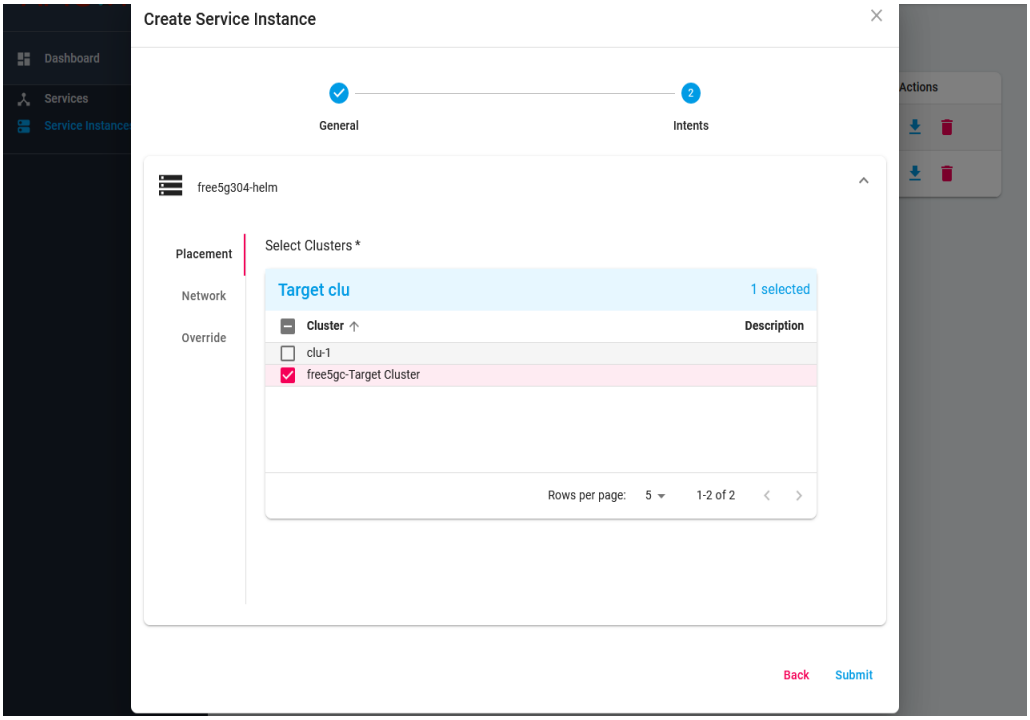


b. In the service instance form, fill in the details like service instance name, version, description etc. Also select the service from the dropdown for which the instance needs to be created. In this case select free5gc Service



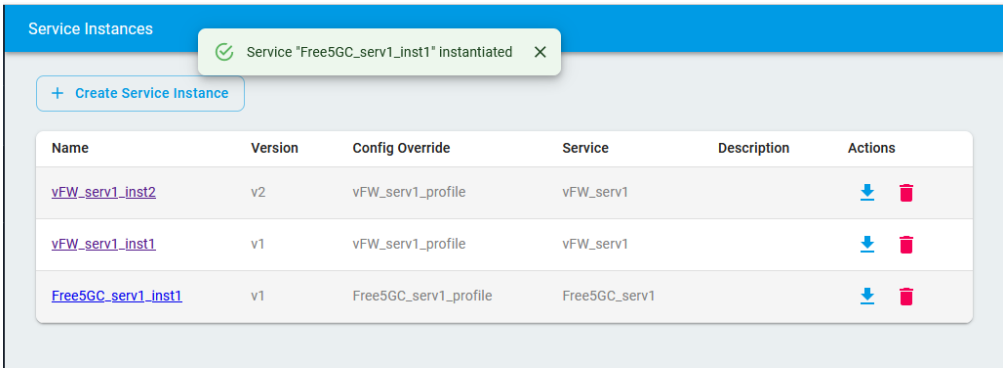
c. When a service is selected, its corresponding override file is automatically selected.
d. We can also provide override values if we need to override any value in the service instance at runtime. For free5gc, leave it blank.
e. Once all the required fields are filled, click on next.
f. Now you will see all the apps which are there in the selected service in the previous step. We can click on each app (in the case of multiple apps) and expand it.

g. Once the application row is expanded, you can see two tabs. One is for placement and the other is for adding the network interfaces. In the case of free5g, you don't need to add network interfaces. In the placement tab select the clusters in which you want your app to be deployed.



h. Now click on the submit button.

i. Now you can see the service instance. To instantiate the service instance, click on the instantiate button' ' in the actions column.



j. On successful instantiation, there will be a success notification at the top center of the screen.
k. You can click on the service instance name to look at the instance details and status. You can click on the activity log to see the activities on the service instance. Here you can see all the applications in the service instance and their deployment status per cloud. You can also see the kubernetes resources of each application by clicking on the Kubernetes Resources tab under the application widget.

3. Verify the deployment of Free5gc

kubectl get pods

```
ubuntu@ubuntu:~$ kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
f5gc-amf-7fb5b8fcb6-5lmwf         2/2     Running   0          53s
f5gc-ausf-857f7cfbf7-jfx7h        2/2     Running   0          53s
f5gc-mongodb-0                    1/1     Running   0          53s
f5gc-nrf-647f4f6576-jvg9f         2/2     Running   0          53s
f5gc-nssf-849b646bb5-5w5wl        2/2     Running   0          53s
f5gc-pcf-6bc4bc57cb-t4zt9         2/2     Running   0          53s
f5gc-smf-5bbcd5b86f-nzpg4         2/2     Running   0          53s
f5gc-udm-86f8c7df8-wrtnn          2/2     Running   0          53s
f5gc-udr-786b5f4c55-tg98j         2/2     Running   0          53s
f5gc-upf-85dff8f64-zvm5g          2/2     Running   0          53s
f5gc-webui-7c788b78d9-2hkmh       2/2     Running   0          52s
```

Free5GC Validation using UERANSIM Simulator

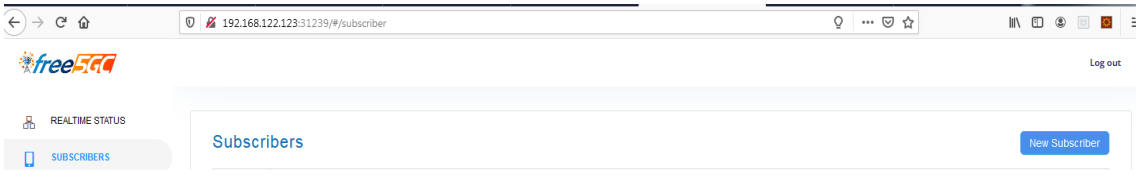
Make sure all free5g components are deployed and in running state, and a SOCKS tunnel is set up to access the webui. Follow the below steps:

1. Login to the webui. Run the below command to identify the port number. The highlighted port is the webui port

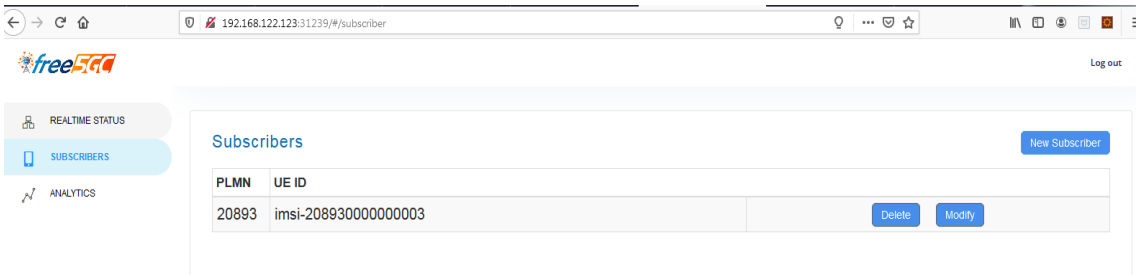
   kubectl get svc -n default

   f5gc-webui NodePort 10.97.75.71 <none> 5000:31239/TCP 95m

2. Open a browser and type http://<VM_IP>:<Port>/#/subscriber. Username and password: admin/free5gc
3. Once login is successful, create a subscriber by clicking the “New Subscriber” button on the right-hand side. Use all default values.



4. Click submit to create a new subscriber

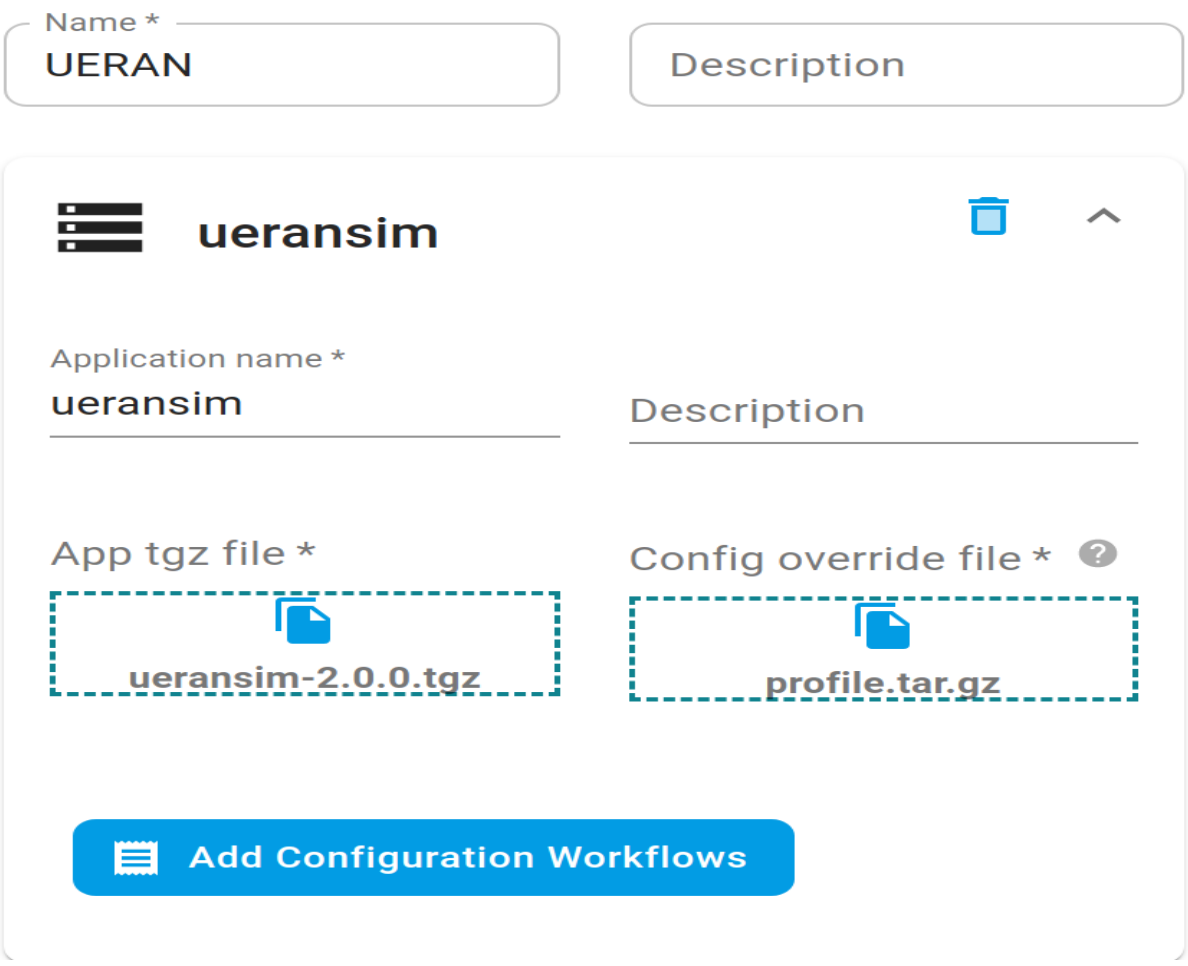


5. Once the Subscriber is added in the webui of free5gc core, create the UERANSIM Service and service instance using from AMCOP UI and instantiate it.(for reference refer below screenshot)

a. Service Orchestration of UERANSIM

The simulator does the following:

1. NGSetup: Perform the handshake between the ueransim gnb and the AMF.
2. UE registration :
    a. Perform UE authentication.
    b. Setup the security encryption.
3. Initial context setup: Setup the communication channel for the UE.
4. PDU session: gtp tunnel setup.
5. To Validate the gtp tunnel creation/setup between UERANSIM and UPF follow below steps:
    a. First Validate the gtp tunnel interface is created or not using below command



i. kubectl logs v1-ueransim-ue-857c46bdbd-fpzf9

1. [2022-03-16 07:54:07.143] [app] [info] Connection setup for PDU session[1] is successful, TUN interface[uesimtun0, 10.1.0.2] is up.

b. If GTP Tunnel is created, Exec to UE Kubernetes Pod using below command
   i. kubectl exec -it v1-ueransim-ue-857c46bdbd-fpzf9 bash
   ii. Try to ping external server using above tunnel interface for example:
      1. ping -I uesimtun0 142.250.183.110
         a. PING 142.250.183.110 (142.250.183.110) from 10.1.0.2 uesimtun0: 56(84) bytes of data.
         b. 64 bytes from 142.250.183.110: icmp_seq=1 ttl=105 time=362 ms

c. Note: If tunnel interface fails to come refer below troubleshooting steps.

Generic Action Controller (GAC)

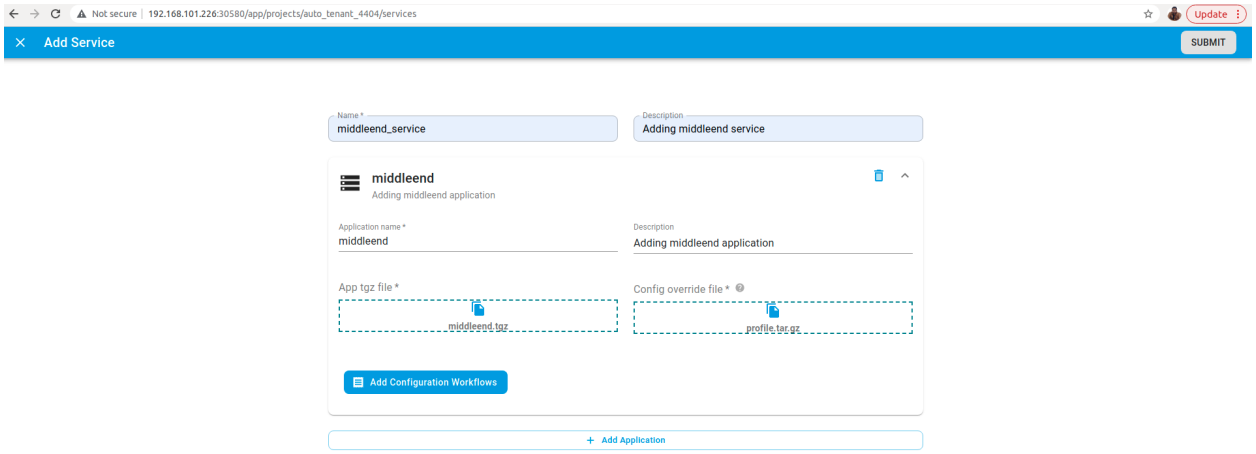
The generic action controller (or GAC) microservice is an action controller which is registered with the central orchestrator.

AMCOP supports the following use cases using GAC.

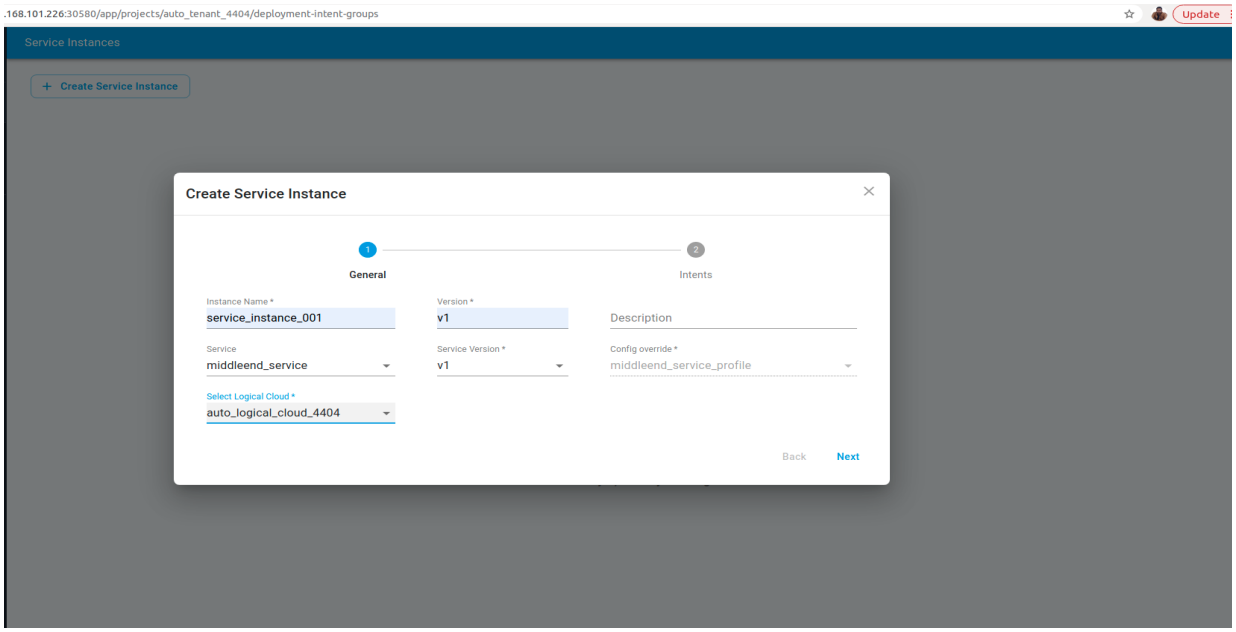
1. Create a new Kubernetes object and deploy it along with a specific application which is part of the composite application
   - Default: Apply the new object to every instance of the app in every cluster where the app is deployed.
   - Cluster-Specific: Apply the new object only where the app is deployed to a specific cluster, denoted by a cluster-name or a list of clusters denoted by a cluster-label
2. Modify an existing Kubernetes object which may have been deployed using the helm chart for an app. Modification may correspond to specific fields in the YAML definition of the object.
   - Resource - Specifies the newly defined object or an existing object.
   - Customization - Specifies the modifications(using YAML Patching) to be applied on the objects.

Add a new Kubernetes resource as configMap

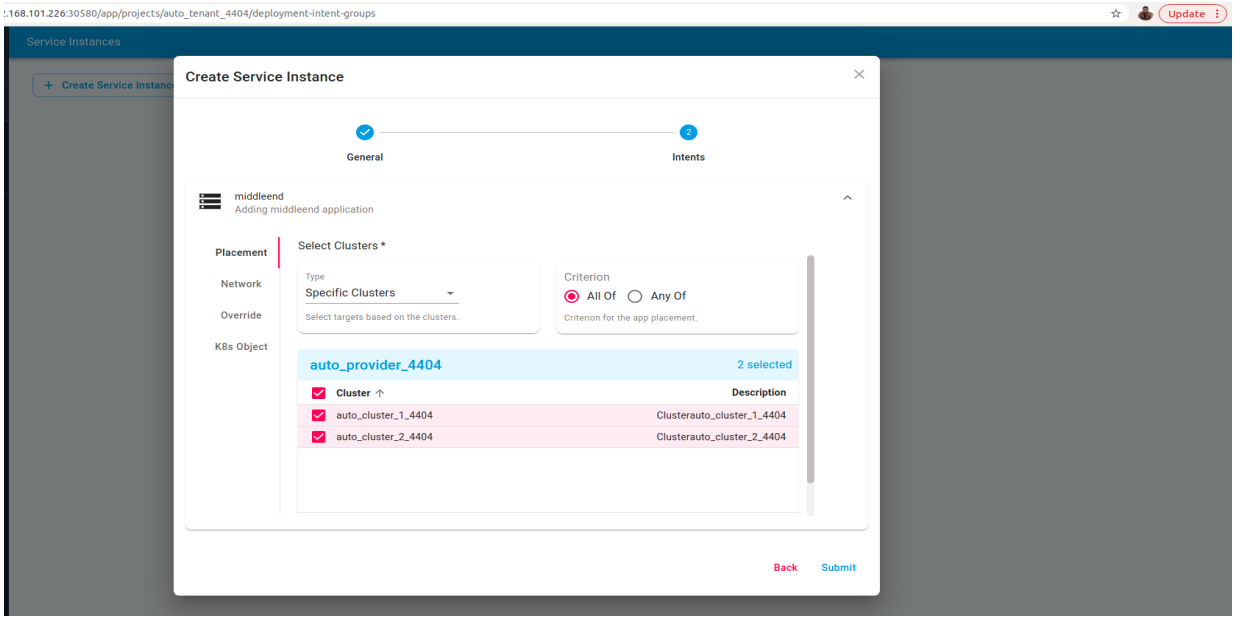
1. Create a Service with the name middleend_service with application middleend:



2. Click on Service Instance Sidebar, and click on Create Service instance Button to fill the details and click Next

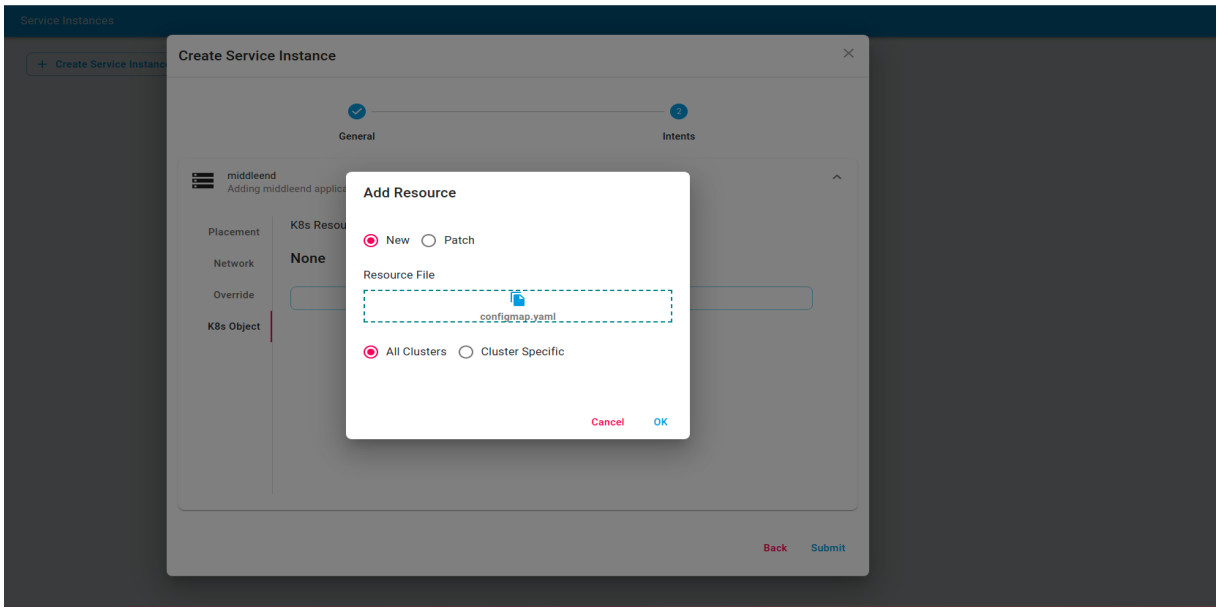


3. Placement details as specific clusters with All of Criterion

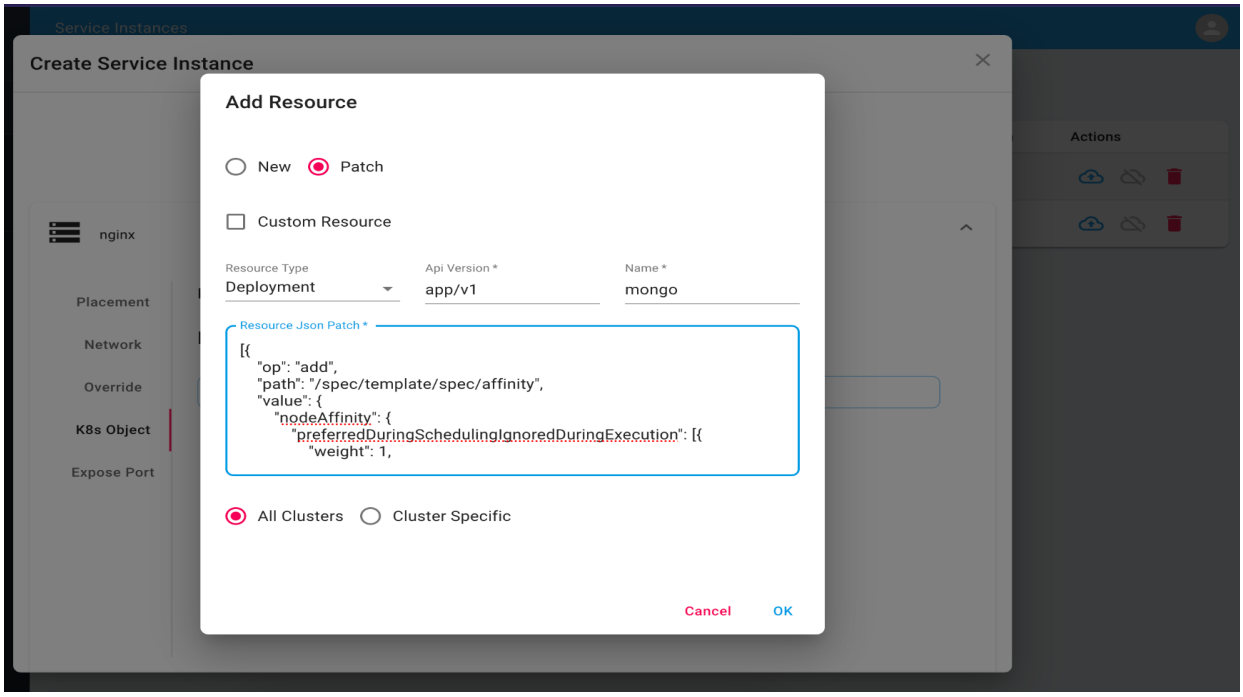


Note: Labels and cluster Specific are supported with *All of* and *Any of* Criterion

4. Click on k8s Object and click on Add button and update the configMap.yaml, click Ok and submit.

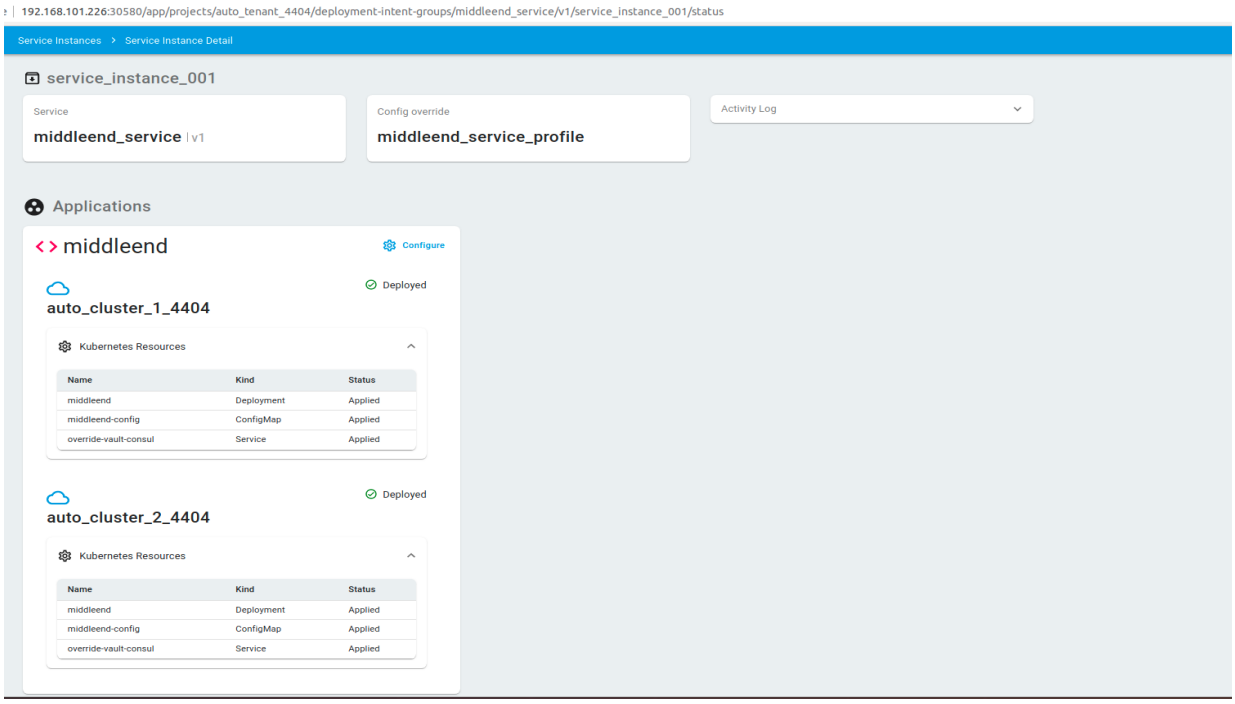


OR
If the requirement is to patch the json then the Patch can be selected,



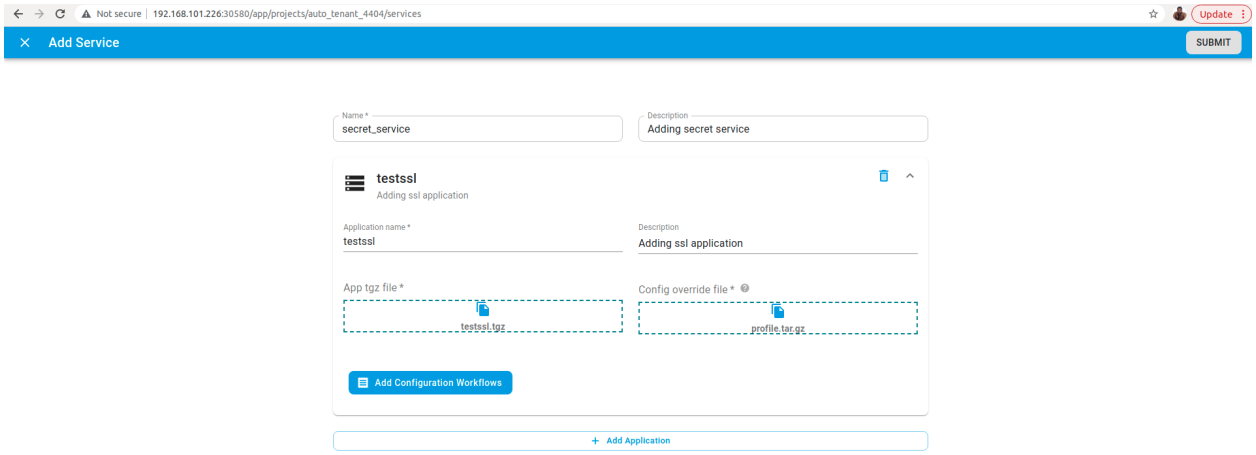
Note: Cluster specific is also supported to choose a specific cluster

5. Instantiate the service and click on the service instance and validate the newly added ConfigMap resource.

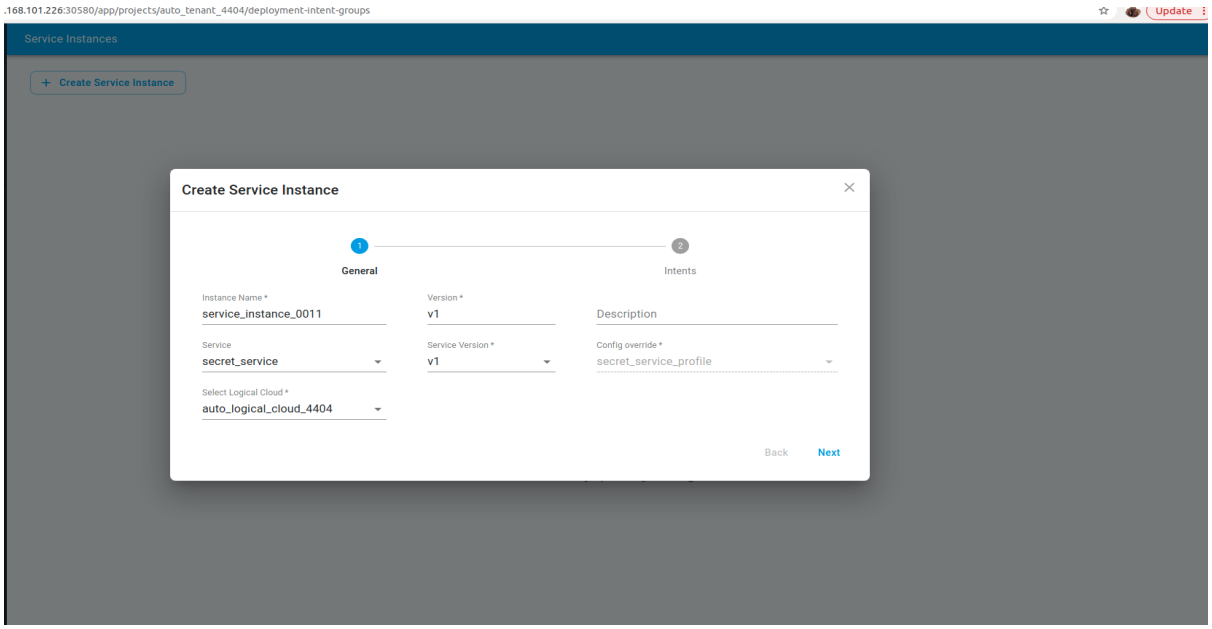


Add a new Kubernetes resource as secret

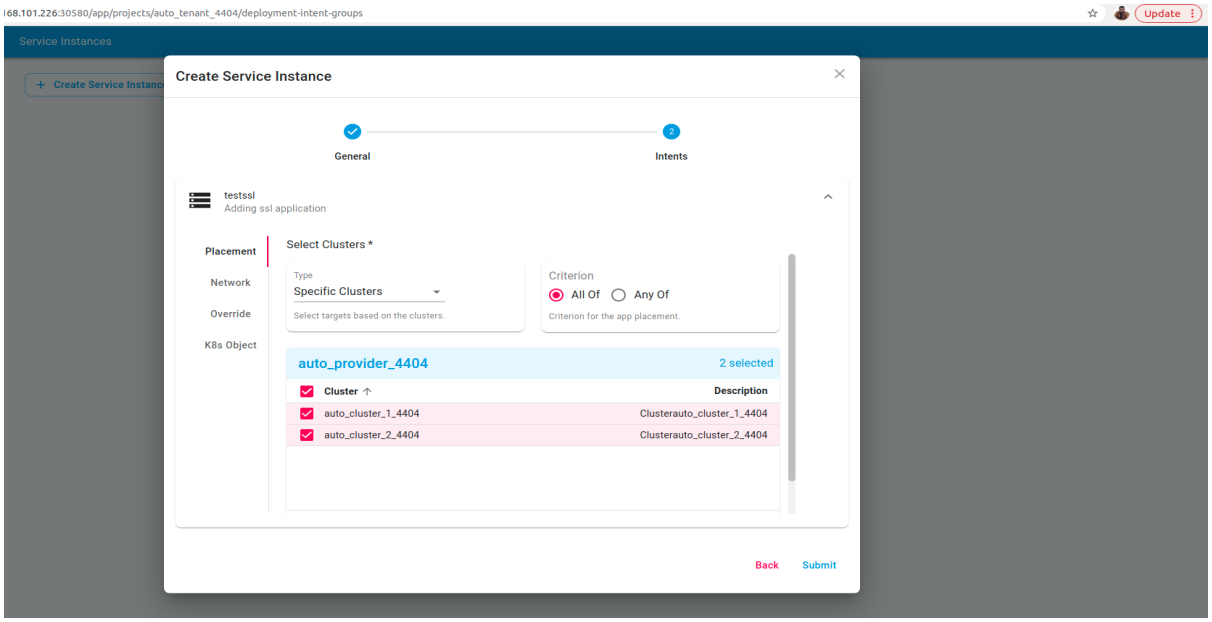
1. Create a Service with the name secret_service with application secret



2. Click on Service Instance Sidebar click on Create Service instance Button fill the details and click Next

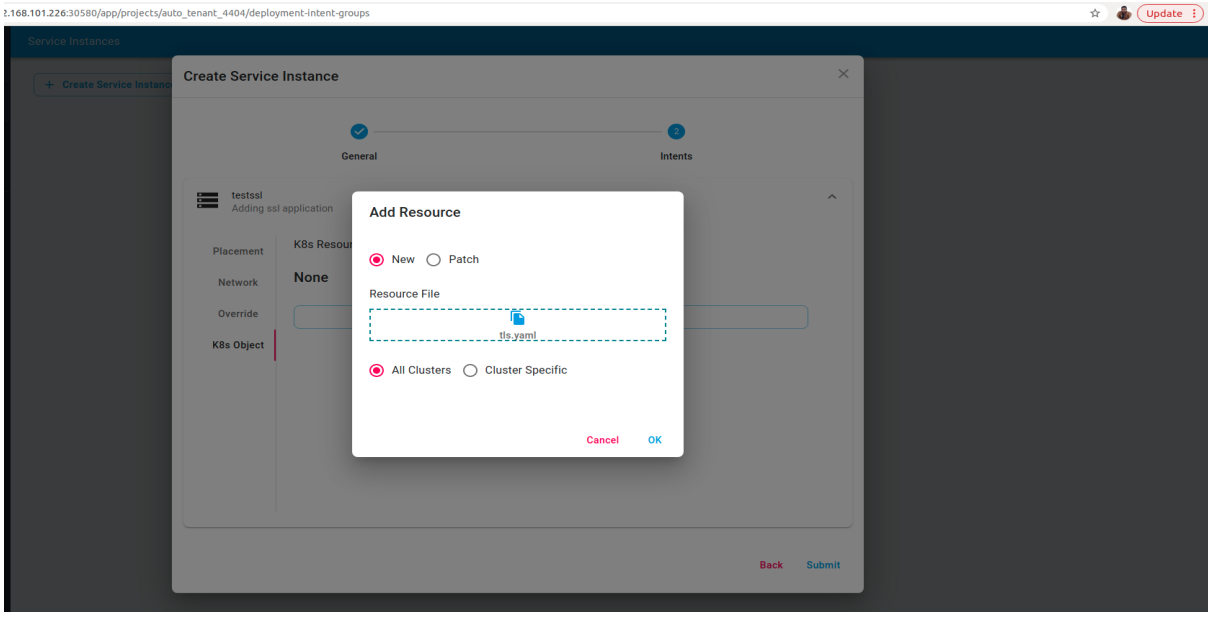


3. Placement details as Specific Clusters with All of Criterion



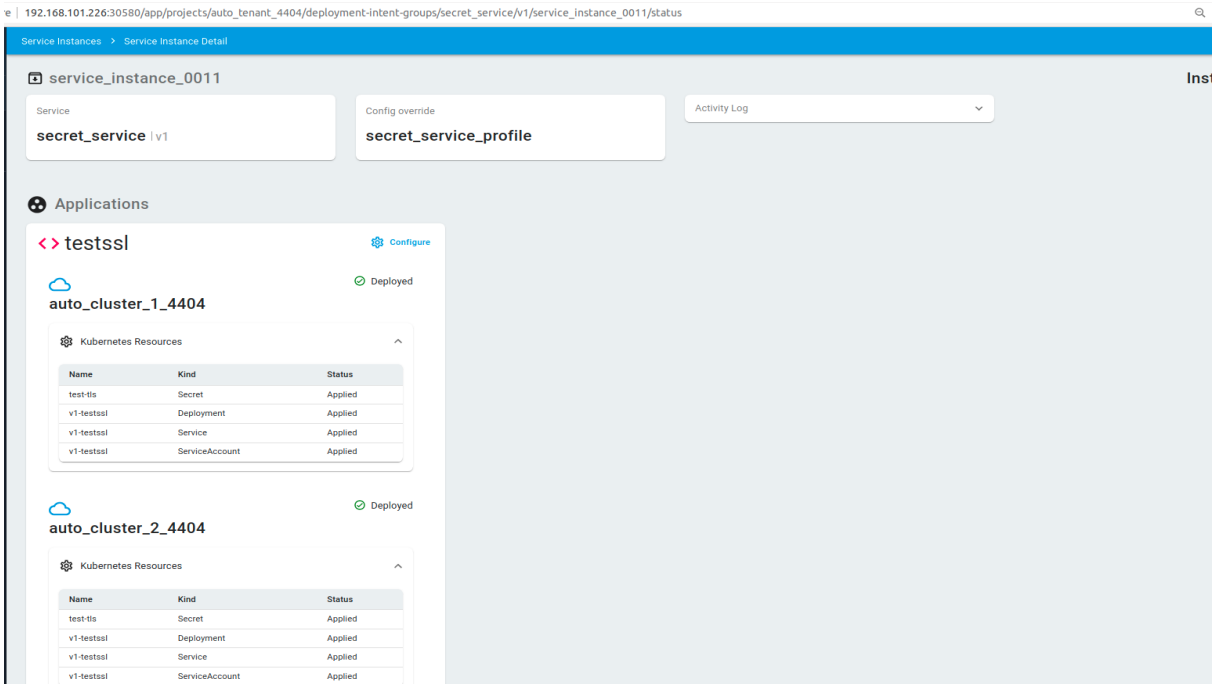
Note: Labels and cluster Specific are supported with All of and Any of Criterion

4. Click on k8s Object and click on Add button and update the tls.yaml, click ok and click submit



Note: Cluster specific is also supported to choose a specific cluster

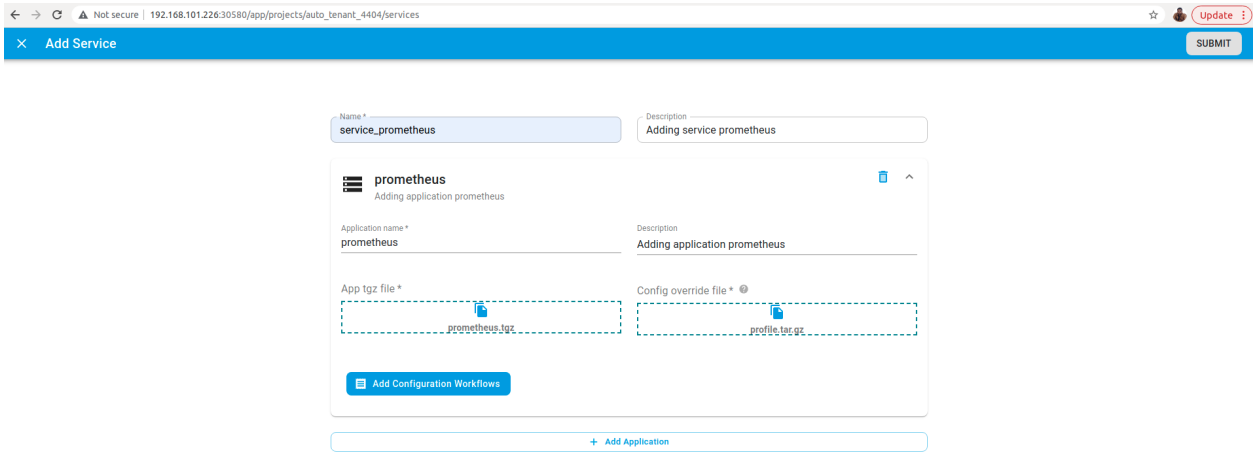
5. Instantiate the service instance and click on service instance and validate newly added secret resource



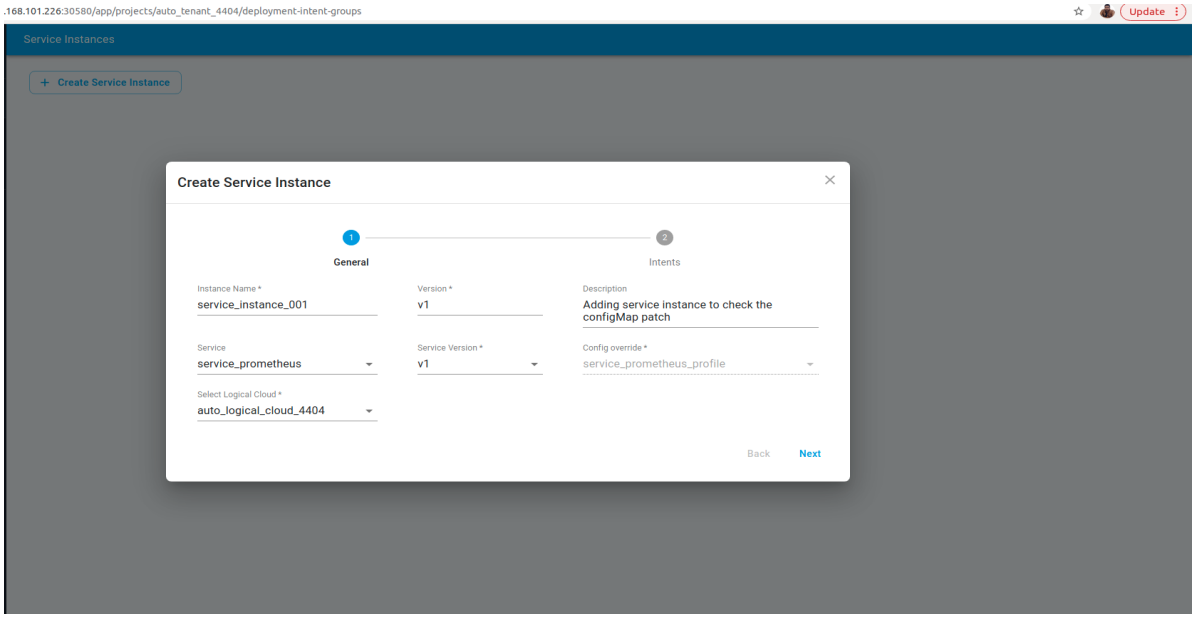
Modifying an existing resource

Patch configMap example:

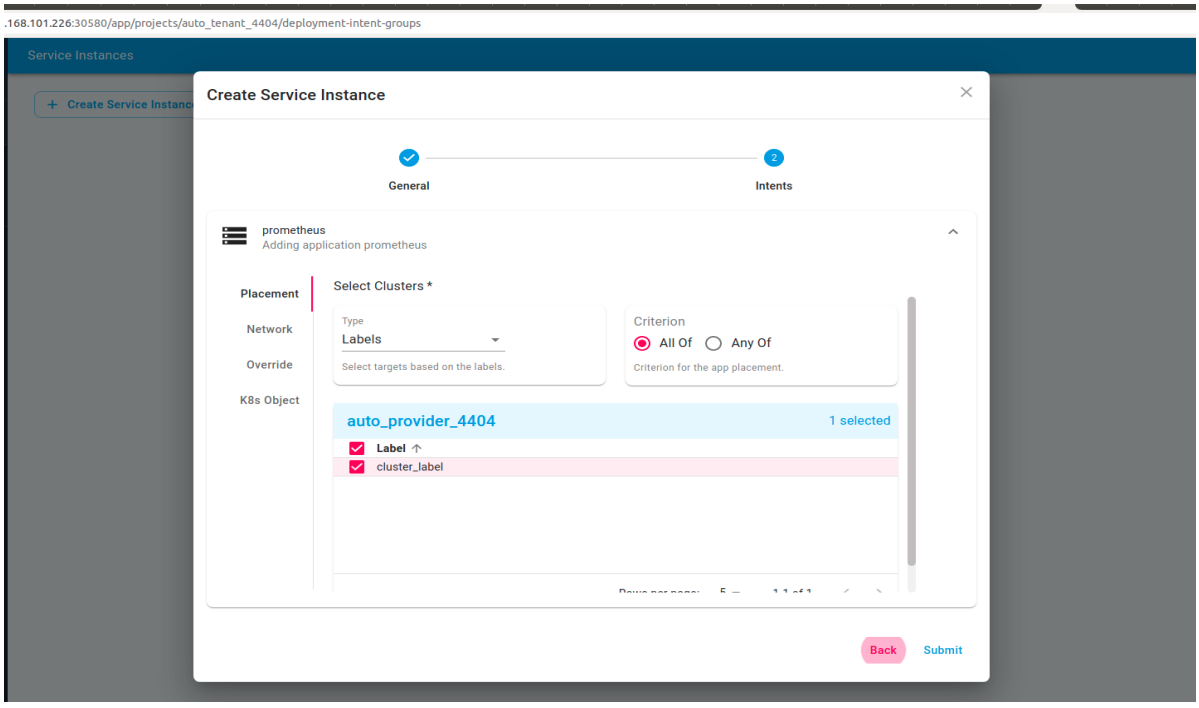
1. Create a Service with the name prometheus_service with application prometheus



2. Click on Service Instance Sidebar click on Create Service instance Button fill the details and click Next

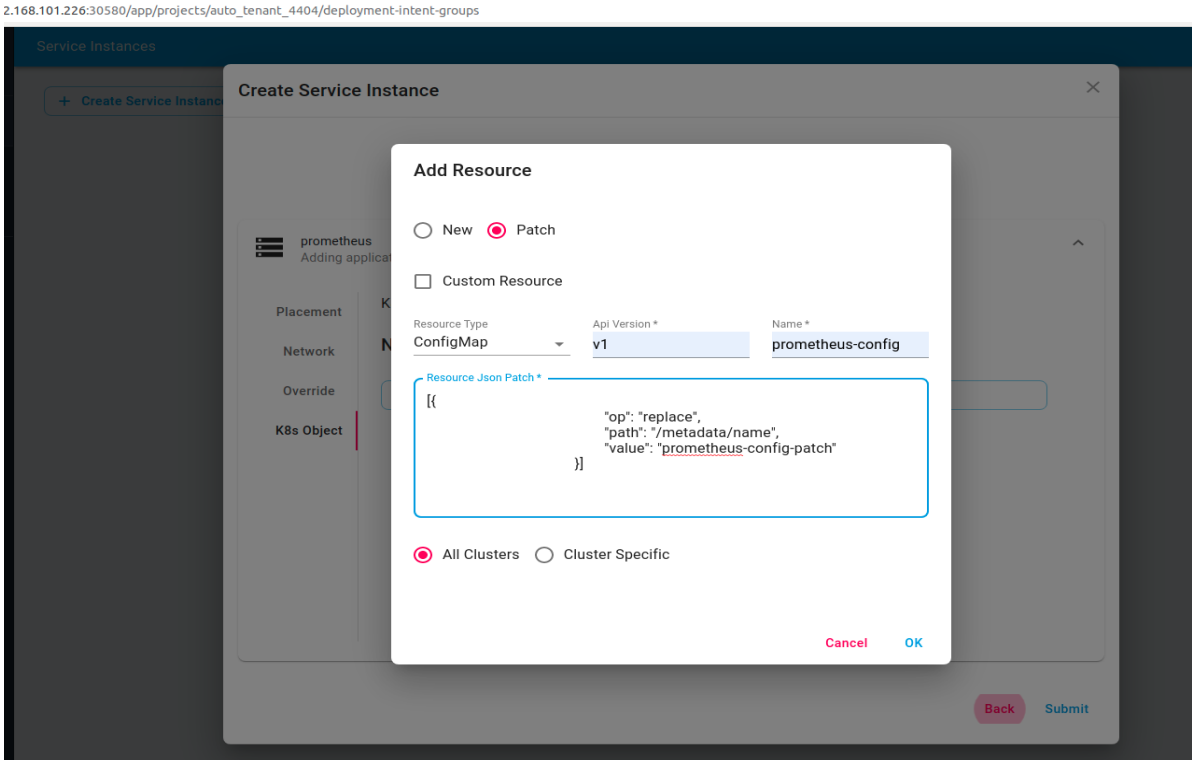


3. Placement details type as Labels withAll of criterion



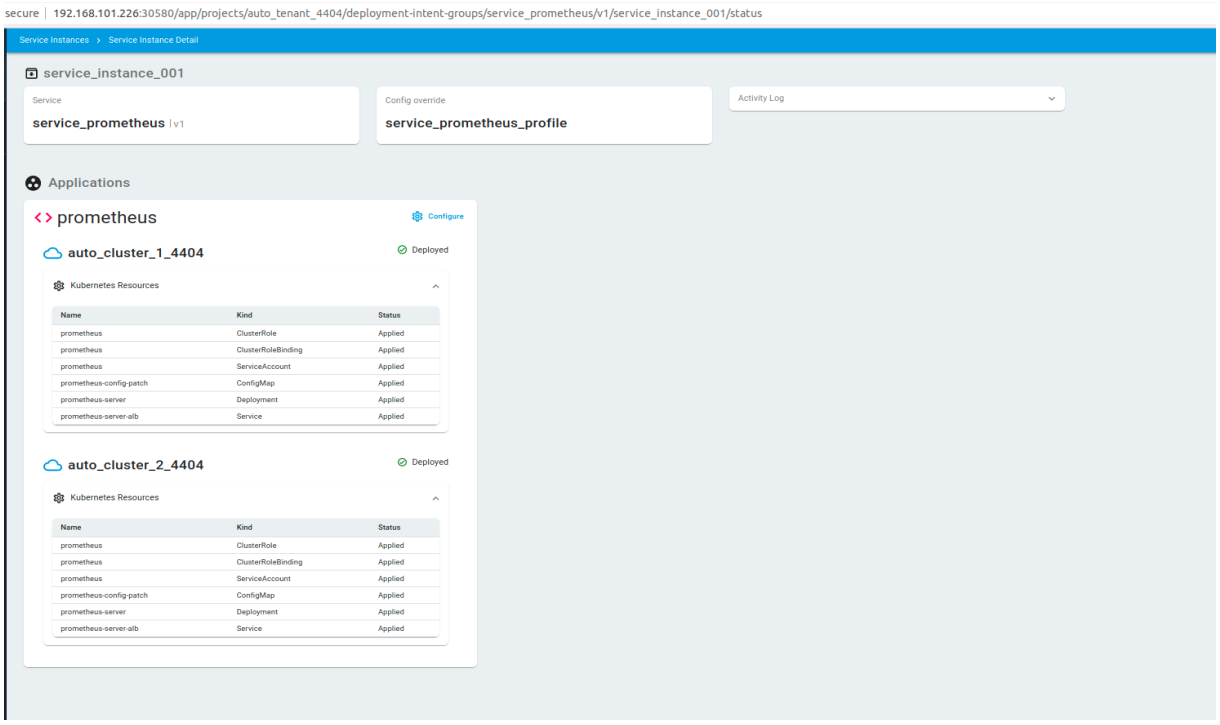
Note: Labels and cluster Specific are supported with *All of* and *Any of* Criterion

4. Click on k8s Object and click on Add button, choose Patch radio button and fill the details as shown in the screenshot for configMap patch (Resource type, version, name and the patch) and click ok and submit



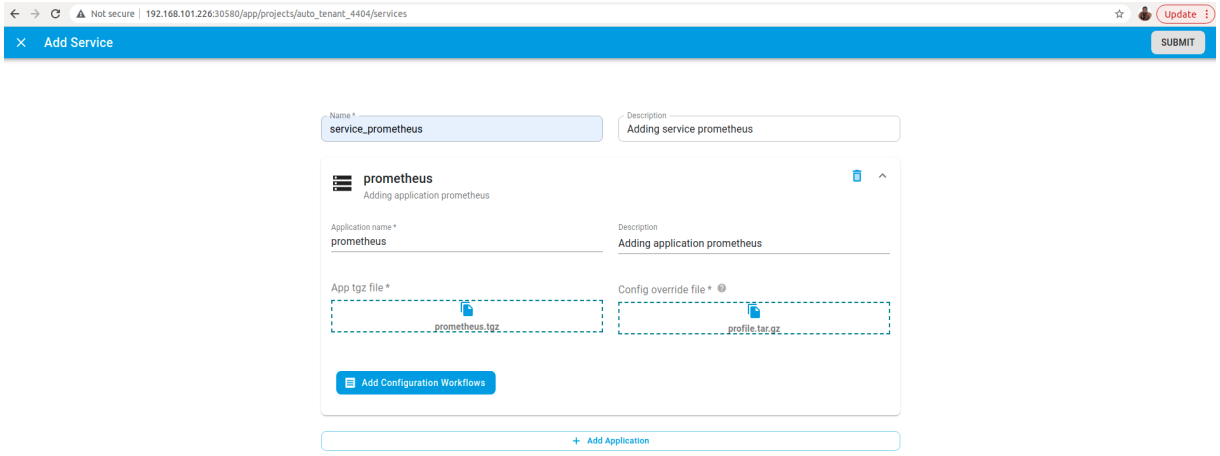
Note: Cluster specific is also supported to choose a specific cluster

5. Instantiate the service instance and click on service instance and validate the patched name for the application configMap

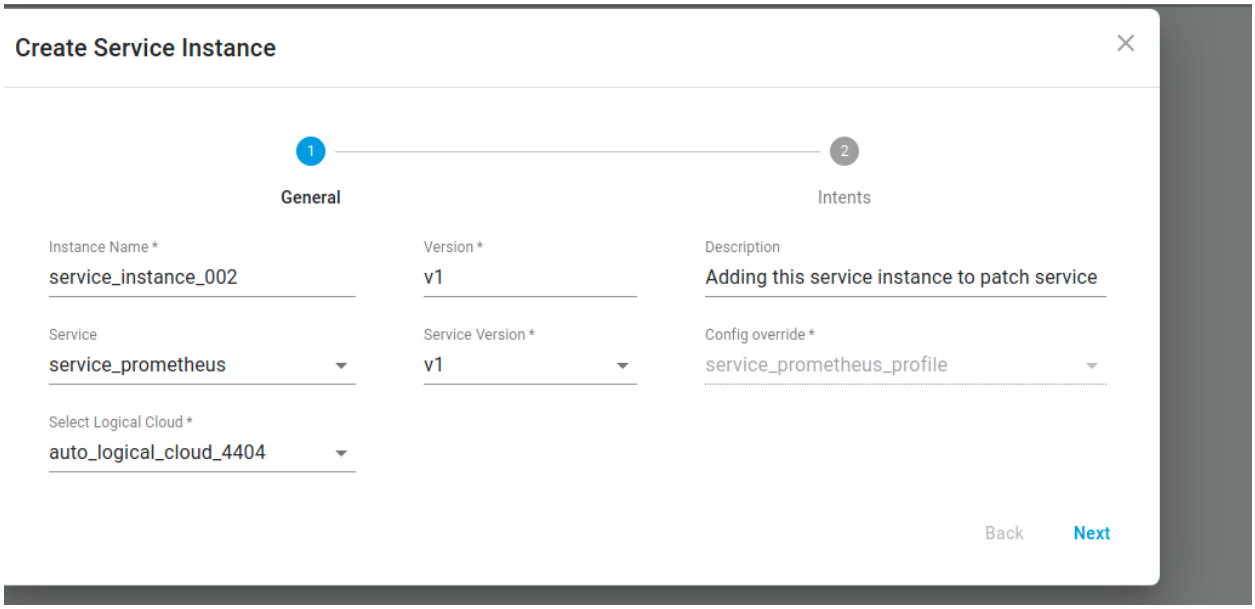


Custom patch service with example:

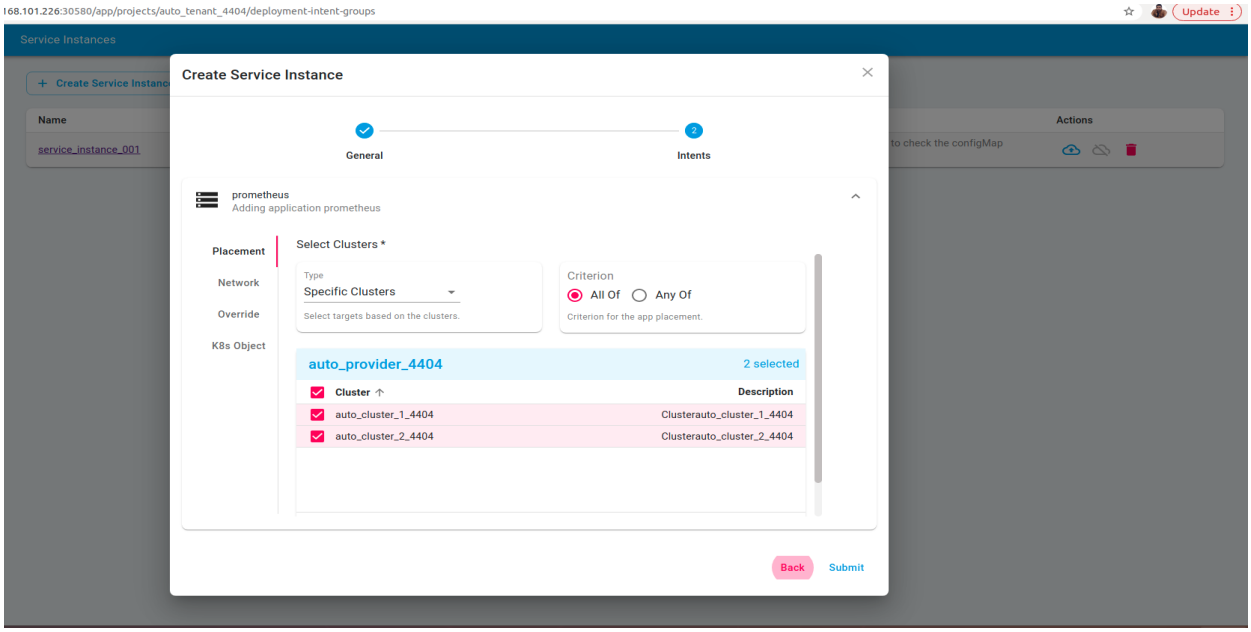
1.Create a Service with the name prometheus_service with application prometheus



2. Click on Service Instance Sidebar click on Create Service instance Button fill the details and click Next

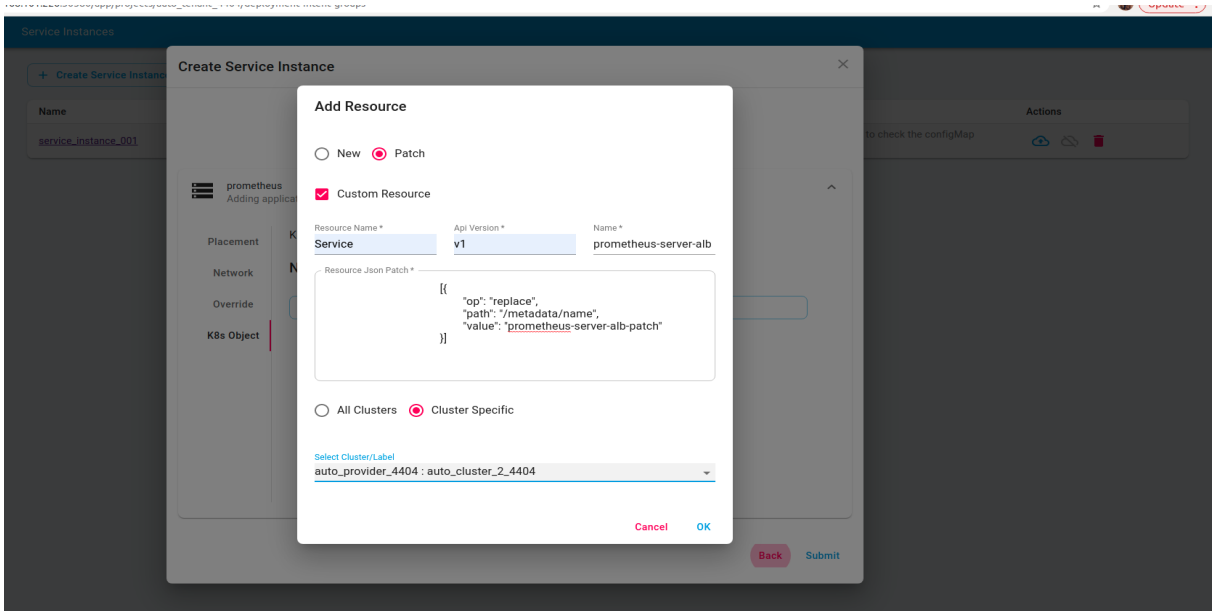


3. Placement details type as cluster specific with All of criterion



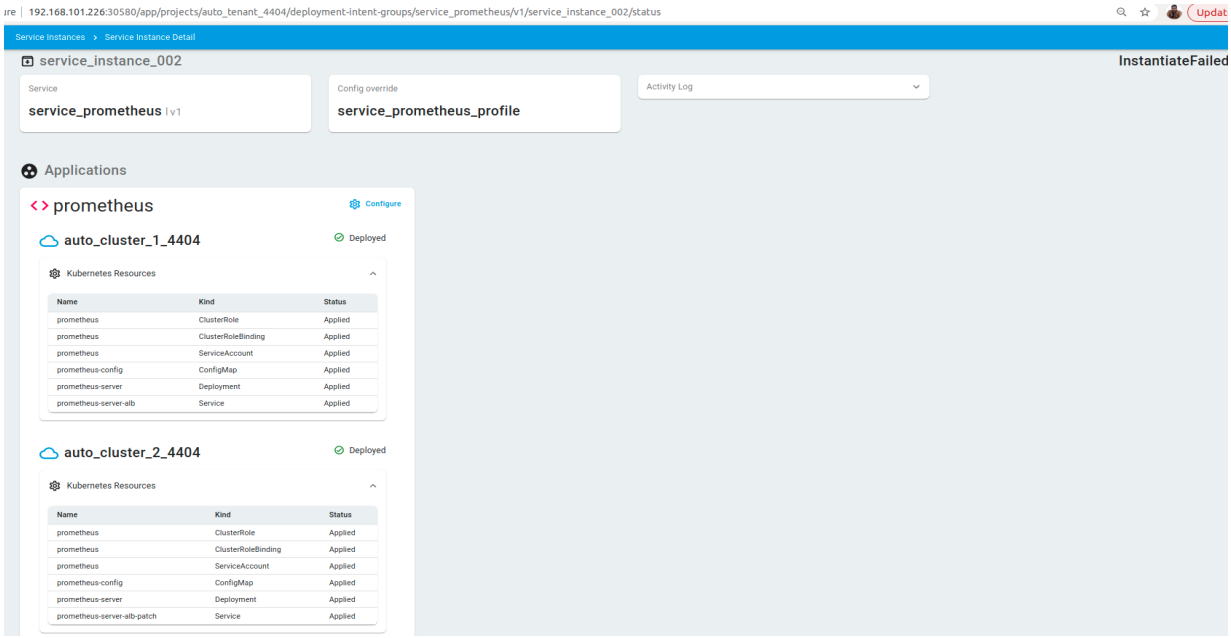
Note: Both Labels and cluster Specific are supported with All of and Any of Criterion

4. Click on k8s Object and click on Add button, choose Patch radio button , check custom patch checkbox and fill the details as shown in the screenshot for service custom patch (Resource type, version, name and the patch) and click ok and submit



Note: Cluster specific is also supported to choose a specific cluster

5. Instantiate the service instance and click on service instance and validate the patched name for the application service



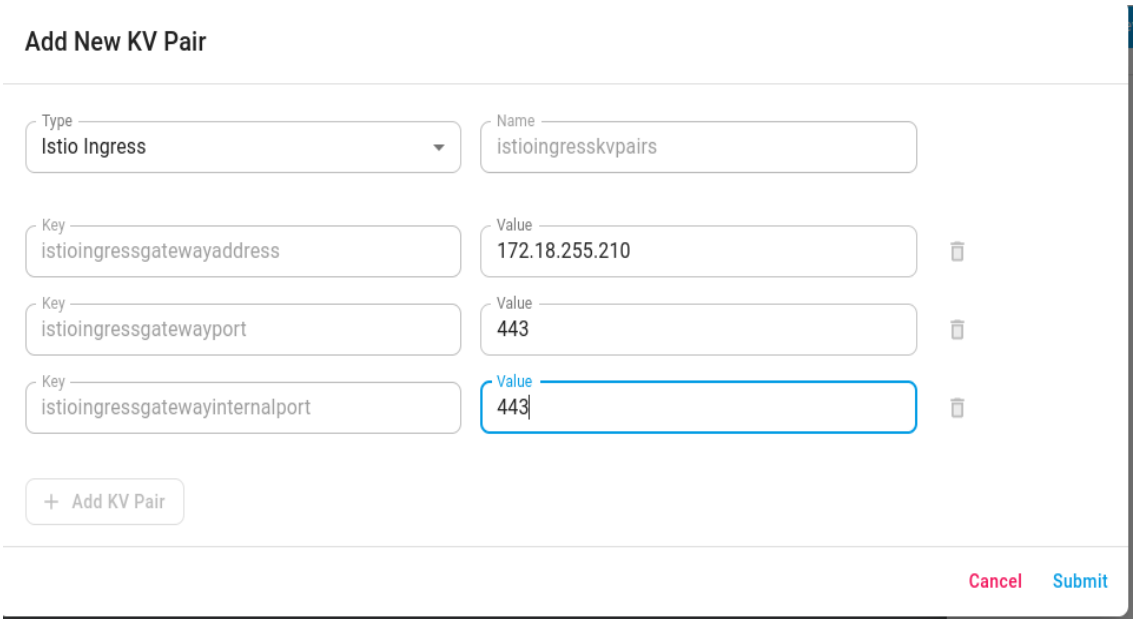
DTC (Distributed Traffic Controller)

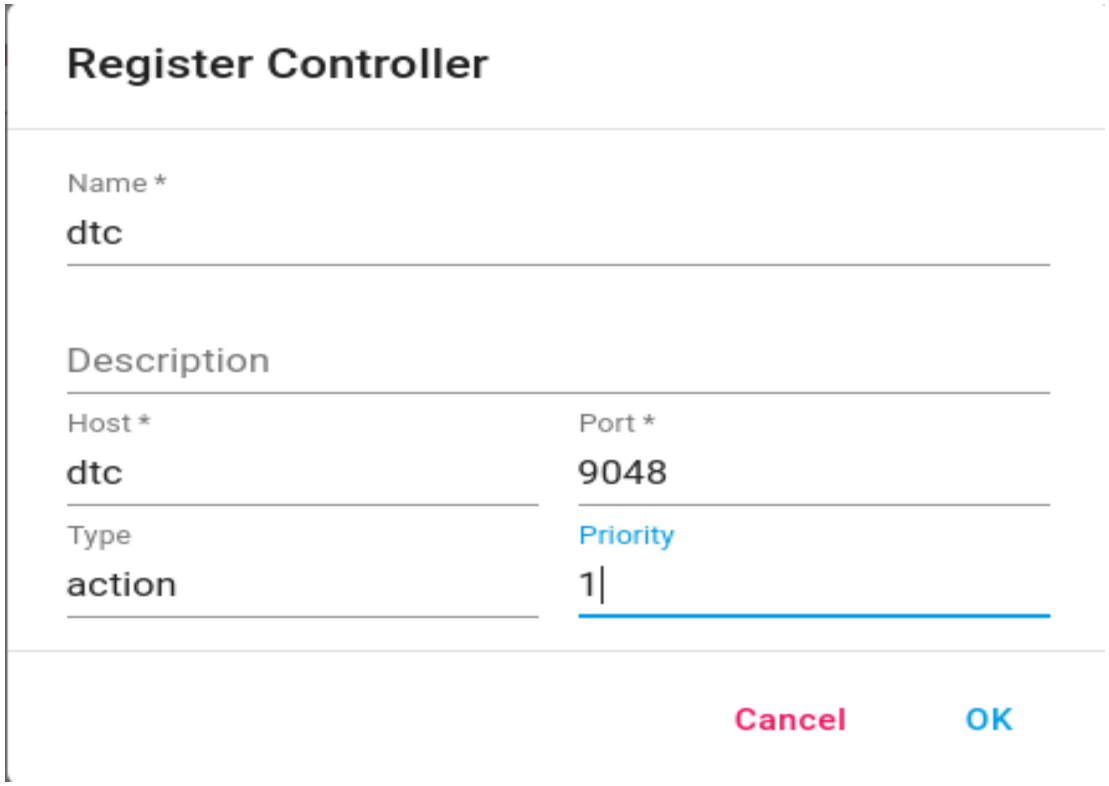
When applications are deployed to multiple clusters by AMCOP, it is important to enable microservices within those applications to discover, resolve their names to IP addresses, and communicate with one another, across applications and clusters. The DTC functionality in AMCOP enables this.

Following are the prerequisites for DTC.

- Prerequisites
  - Istio (refer to https://istio.io/ for details) needs to be installed on target cluster
  - Istio Ingress-gateway service (load balancer type) should have external IP (to acquire external IP, we need to support Load balancer lib on target cluster)
  - Istio sidecar need to be enabled for a given namespace
  - Istio DNS Proxy needs to be enabled for domain name resolution

- DTC AMCOP Requirements
  - K8s DTC and Istio controller needs to be onboarded, as follows:

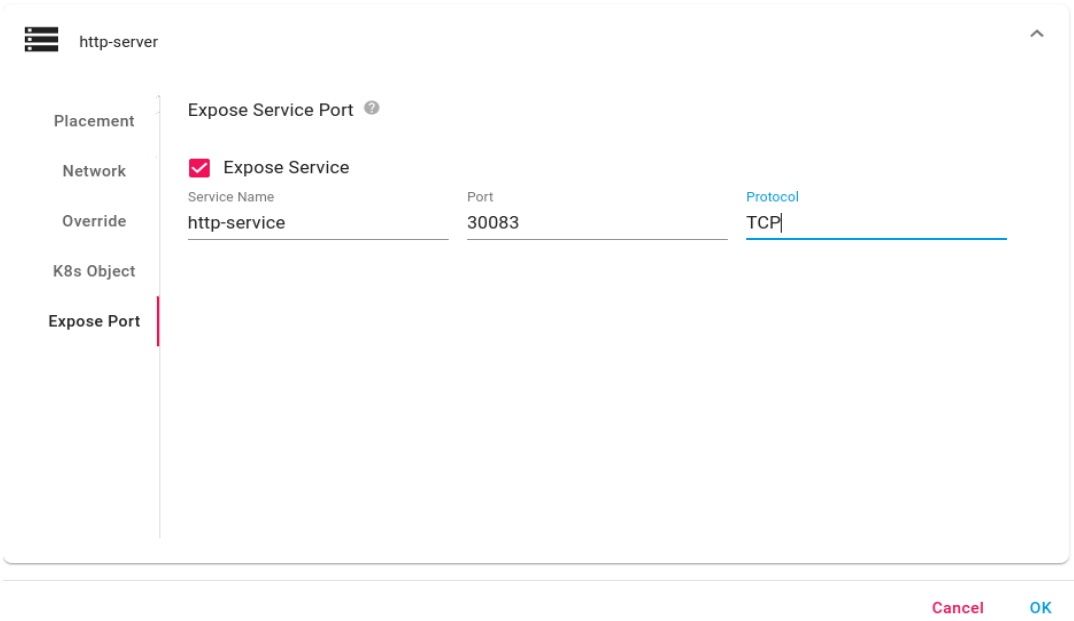




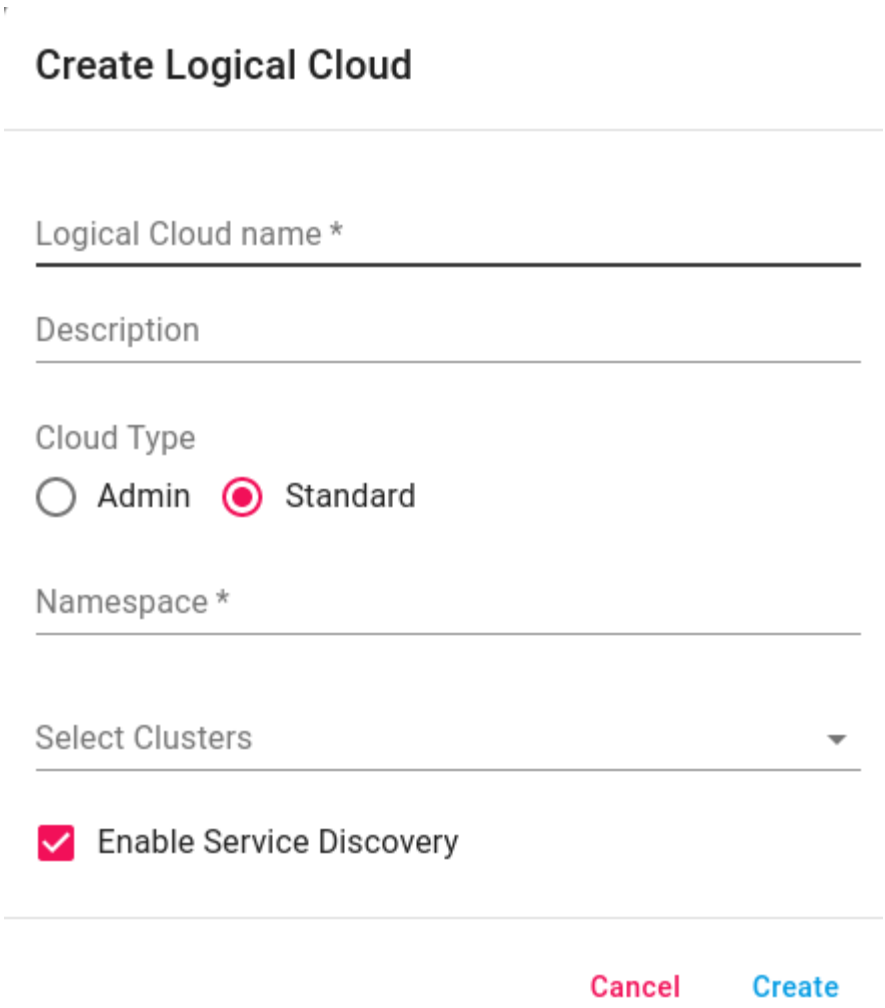
- KV Pair Needed for a target cluster(from where we need to discover server service) with the following parameter:

Note: KV PAIR can be added per cluster

- At the Service instance level, you need to expose the service which you want to expose across multiple clusters in a given logical cloud (L0/L1) with the below parameter.
  - Service Name
  - Port
  - Protocol
- In case L1 Logical cloud we need to enable service discovery parameter



- parameter for dtc intent to create a kubernetes resource in istio-namespace.



- DTC Verification
    - Below Kubernetes resource should be created in target cluster
        - Service entry
        - Destination Rule
        - Gateway

Orchestration across multiple clusters

This section describes how to create a service mesh across two clusters by installing an *istio* controller on both clusters and making each a primary cluster. It can be extrapolated to multiple clusters.

This section assumes that the two cluster names are named *cluster1* and *cluster2*, and that a composite kubeconfig file is created which has two contexts *cluster1* and *cluster2*.

Download Istio

Use the following command to download and install ISTIO software.

```
curl -L https://istio.io/downloadIstio | sh -
```

This will download the latest version of the istio, and export the istioctl present in the bin/ directory.

Plug in CA Certificates

All the commands are to be executed from the istio directory.

Create a certs directory and create the root certificate,

```
mkdir -p certs
pushd certs
make -f ../tools/certs/Makefile.selfsigned.mk root-ca
```

Execute the following commands for each cluster (you can create a script by copying the following instructions into a file)

```
kubectl config use-context cluster1
#cleanup istio
kubectl delete -f samples/addons
istioctl manifest generate --set profile=demo | kubectl delete --ignore-not-found=true -f -
kubectl delete namespace istio-system
kubectl  delete ns foo
#create root and intermediate ca certs
pushd certs
make -f ../tools/certs/Makefile.selfsigned.mk cluster1-cacerts
kubectl create namespace istio-system
kubectl create secret generic cacerts -n istio-system \
    --from-file=cluster1/ca-cert.pem \
    --from-file=cluster1/ca-key.pem \
    --from-file=cluster1/root-cert.pem \
    --from-file=cluster1/cert-chain.pem

popd
#install istio
istioctl install --set profile=demo
#create dummy app
```

```
kubectl create ns foo
kubectl apply -f <(istioctl kube-inject -f samples/httpbin/httpbin.yaml) -n foo
kubectl apply -f <(istioctl kube-inject -f samples/sleep/sleep.yaml) -n foo
#apply tls policy
kubectl apply -n foo -f - <<EOF
apiVersion: "security.istio.io/v1beta1"
kind: "PeerAuthentication"
metadata:
  name: "default"
spec:
  mtls:
    mode: STRICT
EOF

#cleanup
rm httpbin-proxy-cert.txt
rm certs.pem
rm proxy-cert-*

#validate certificate
sleep 20; kubectl exec "$(kubectl get pod -l app=sleep -n foo -o jsonpath={.items..metadata.name})"
-c istio-proxy -n foo -- openssl s_client -showcerts -connect httpbin.foo:8000 > httpbin-proxy-cert.txt
sed -n '/-----BEGIN CERTIFICATE-----/{:start /-----END CERTIFICATE-----/!{N;b start};/.*/p}'
httpbin-proxy-cert.txt > certs.pem
awk 'BEGIN {counter=0;} /BEGIN CERT/{counter++} { print > "proxy-cert-" counter ".pem"}' <
certs.pem
openssl x509 -in certs/cluster1/root-cert.pem -text -noout > /tmp/root-cert.crt.txt
openssl x509 -in ./proxy-cert-3.pem -text -noout > /tmp/pod-root-cert.crt.txt
diff -s /tmp/root-cert.crt.txt /tmp/pod-root-cert.crt.txt

openssl x509 -in certs/cluster1/ca-cert.pem -text -noout > /tmp/ca-cert.crt.txt
openssl x509 -in ./proxy-cert-2.pem -text -noout > /tmp/pod-ca-cert.crt.txt
diff -s /tmp/ca-cert.crt.txt /tmp/pod-ca-cert.crt.txt

openssl verify -CAfile <(cat certs/cluster1/ca-cert.pem certs/kind-cluster1/root-cert.pem)
./proxy-cert-1.pem
```

Install Multi-Primary

After creating the certificate CA in both clusters using the same root certificate, we can now enable the endpoint discovery in each cluster by installing the remote secrets of each of the clusters,

Export the env variable for each of the cluster contexts,

```
export CTX_CLUSTER1=cluster1
export CTX_CLUSTER2=cluster2
```

Make Cluster1 a primary

```
cat <<EOF > cluster1.yaml
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  values:
    global:
      meshID: mesh1
      multiCluster:
        clusterName: cluster1
      network: network1
EOF

istioctl install --context="${CTX_CLUSTER1}" -f cluster1.yaml
```

Make Cluster2 as primary

```
cat <<EOF > cluster2.yaml
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  values:
    global:
      meshID: mesh1
      multiCluster:
        clusterName: cluster2
      network: network1
EOF
istioctl install --context="${CTX_CLUSTER2}" -f cluster2.yaml
```

Enable endpoint Discovery in Cluster 2

```
istioctl x create-remote-secret \
  --context="${CTX_CLUSTER1}" \
  --name=cluster1 | \
  kubectl apply -f - --context="${CTX_CLUSTER2}"
```

Enable endpoint Discovery in Cluster 1

```
istioctl x create-remote-secret \
  --context="${CTX_CLUSTER2}" \
  --name=cluster2 | \
  kubectl apply -f - --context="${CTX_CLUSTER1}"
```

At this point the istio control plane will be able to discover the service endpoints in the remote clusters. Deploy free5g in a namespace which has the istio sidecar injection enabled,

```
kubectl label --context="${CTX_CLUSTER1}" namespace f5gc  istio-injection=enabled
```

Note: Make sure that service endpoints are created in both clusters, for example the service.yaml of the AMF has to be executed in both clusters. The pod of AMF however will reside only in one of the clusters.

Orchestration using REST Interface

This section shows how to register a k8s cluster with AMCOP, design composite applications (CNFs or CNAs) and orchestrate them using AMCOP REST interface. It uses vFirewall composite application as an example.

- If AMCOP is installed on a bare-metal server, log into the AMCOP master node.

  ```
  ssh <user>@<amcop-master-ip>
  ```

- If AMCOP is installed on GKE, we will run the commands from the jump host from where ansible scripts were executed.

- If AMCOP is installed on AKS, we need to log into the k8s cluster VM. Use the following commands to get the IP address.

  ```
  az vm list --show-details -o=table | grep amcop-kud | awk '{ print $5 }'

  # Use the IP address returned by the previous command for ssh

  ssh aarna@<IP_Address>
  ```

  Next, you need to copy the kubeconfig file of the AMCOP cluster to the above VM.

```
scp ~/.kube/config aarna@<IP ADDRESS>:/tmp/amcop_config
```

- Execute the below commands on the master node (amcop VM) to start the composite application. The script *vfw_orchestrate_python.py* uses REST API to perform all the operations on AMCOP.

```
cd ~/aarna-stream/amcop_deploy/gitlab-ci/
cp sink.tgz ~
cp firewall.tgz ~
cp packetgen.tgz ~
cp profile.tar.gz ~

# Copy the k8s config file from the target kud cluster to the VM

scp ubuntu@<target kud cluster IP>:~/.kube/config ~/k8_config



# Note: Install python requests library:
“sudo apt-get install -y python-requests”

python vfw_orchestrate_python.py <amcop_vm_ip> <middle_end_port> <orch_port>
<clm_port> <dcm_port>

# For example:
python vfw_orchestrate_python.py 192.168.122.110 30661
```

- After this, you will be able to see the vFW CNFs started on the target k8s cluster (KuD), with the required networking setup.

    - For bare metal server deployment, you can log in to edge_k8s VM, to verify that vFW CNFs are running. They may take a few minutes to go into the Running state.

```
ssh <user>@<edge_k8s-ip>

kubectl get pods -n default

NAME                        READY   STATUS    RESTARTS   AGE
v1-firewall-cdbf6bc85-79vlc     1/1     Running   0          27h
v1-packetgen-6cd8564898-fpxq2   1/1     Running   0          27h
v1-sink-864867d7b5-rghgl        2/2     Running   0          27h
```

    - For GKE deployment you can log into the target KUD cluster.

```
gcloud compute ssh <CLUSTER NAME>
```

For example:

```
gcloud compute ssh amcop-kud
```

Once logged into the cluster VM, you can run the kubectl command to list all the pods.

```
kubectl get pods -n default

NAME                      READY   STATUS    RESTARTS   AGE
fw0-firewall-cdbf6bc85-79vlc    1/1     Running   0        27h
fw0-packetgen-6cd8564898-fpxq2   1/1     Running   0        27h
fw0-sink-864867d7b5-rghgl       2/2     Running   0        27h
```

- For AKS deployments we log into the VM where the KUD cluster is running and check if the CNF is deployed.

```
az vm list --show-details -o=table | grep amcop-kud | awk '{ print $5 }'

# Use the IP address returned by above command

ssh aarna@<IP_Address>
```

Next, you can run the kubectl command to list all the pods.

```
kubectl get pods -n default

NAME                      READY   STATUS    RESTARTS   AGE
fw0-firewall-cdbf6bc85-79vlc    1/1     Running   0        27h
fw0-packetgen-6cd8564898-fpxq2   1/1     Running   0        27h
fw0-sink-864867d7b5-rghgl       2/2     Running   0        27h
```

GitOps Support

GitOps works by using Git as a single source of truth for declarative infrastructure and applications. With GitOps, the use of software agents can alert on any divergence between Git with what's running in a cluster, and if there's a difference, Kubernetes reconcilers automatically update or rollback the cluster depending on the case.

GitOps support in AMCOP enables administrators to manage applications deployed in target clusters which is enabled by using *Flux* as the agent, through the git repository. Currently only Flux agent is supported for gitOps at the target cluster.

Applications deployed in target clusters through AMCOP which are Flux enabled, are synced to the git repository which is provided as an input while installing the Flux agent.

Prerequisites

1. Run below command on target cluster for enabling flux:

```
flux bootstrap github --owner=$GITUSERNAME --repository=$GITREPONAME --branch=main
--path=./clusters/provider1flux+cluster1  --personal
```

2. Run below command on target cluster for enabling monitor. Ensure monitor package is downloaded and untared before running below command:

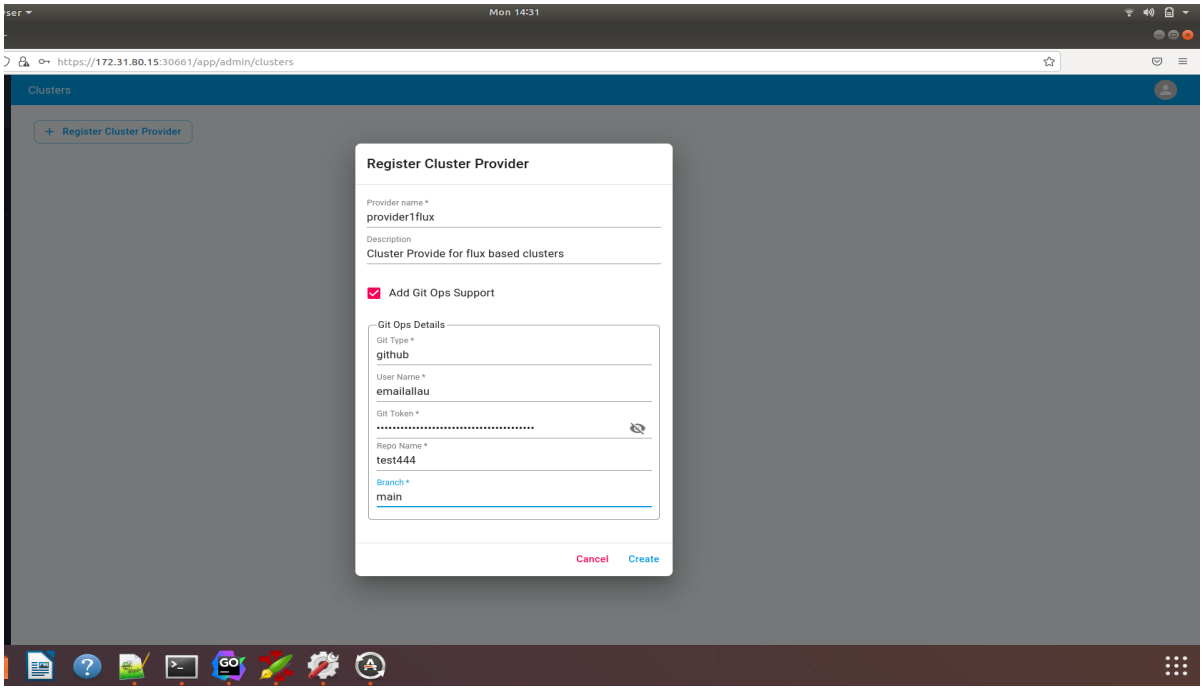
```
helm install --kubeconfig admin.conf --set git.token=$GITTOKEN --set
git.repo=$GITREPONAME --set git.us
```

Cluster provider and Cluster onboarding

In order to enable gitops, extra information needs to be provided while creating a cluster provider on AMCOP GUI. The information is as the following,

The details of parameters below is as follows:

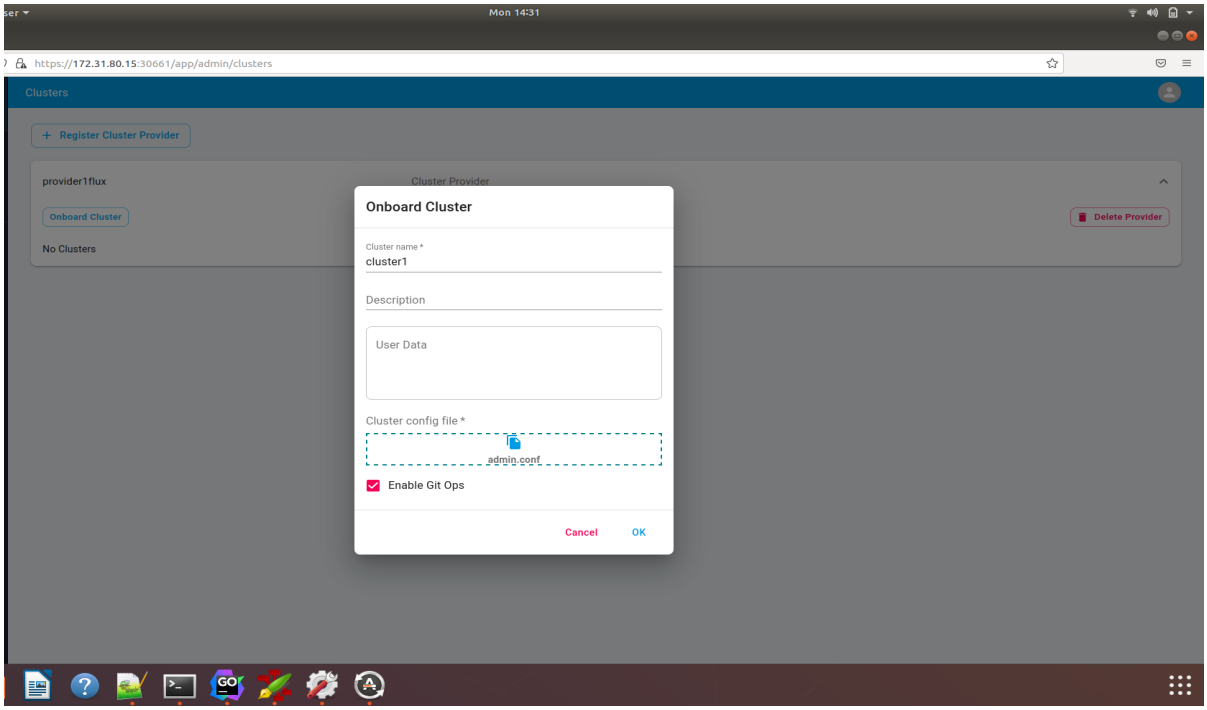
a) Git Type: Supported options are github and gitlab
b) User Name: The name of the user who has access to github/gitlab repository
c) Git Token: PAT (Personal Access Token), that can be generated by navigating to developer settings of a given user.
d) Repo Name: Name of the git repository
e) Branch: Branch of git repository, where the user intends to store the cluster resources.



While onboarding cluster under the cluster provider, check the 'Enable Git Ops' button on the cluster onboarding form,

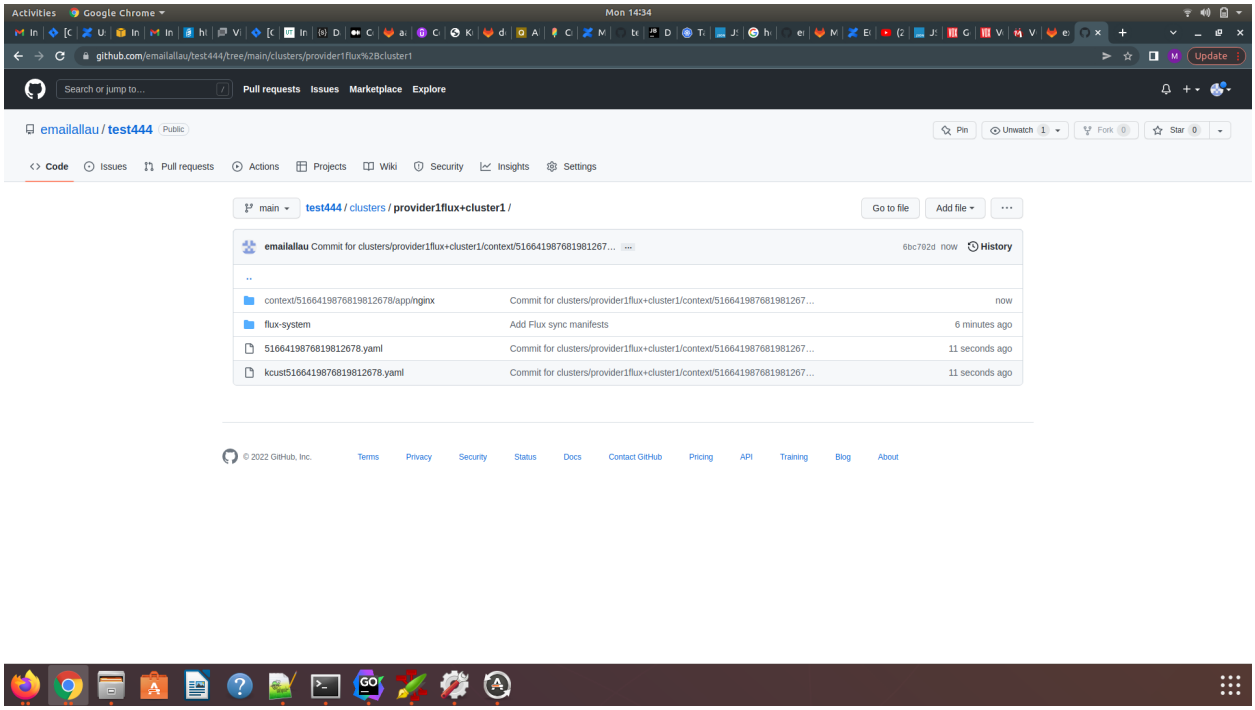
The details of parameters below is as follows:

a) Cluster Name: Name of cluster
b) Description: Description of cluster
c) Cluster Config File: kube config file of target cluster
d) Enable Git Ops: Enable this checkbox, only when the target cluster has flux installed, and administrator intends to have its resources managed through git repository.



The GitRepo and resource sync

On instantiating a service with gitops enabled, the application context of the service will be uploaded to the provided git location. The resources are then synced by Flux in target clusters resulting in the kubernetes resource creation. Following is a screenshot of the gitlab:


Activities
Google Chrome
Mon 14:34
github.com/emaillallau/test444/tree/main/clusters/provider1flux%2Bcluster1
Update
Search or jump to...
Pull requests
Issues
Marketplace
Explore
emaillallau / test444
Public
Pin
Unwatch 1
Fork 0
Star 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
main
test444 / clusters / provider1flux+cluster1 /
Go to file
Add file
emaillallau Commit for clusters/provider1flux+cluster1/context/516641987681981267…
6bc702d now
History
..
context/516641987681981267 8/app/nginx
Commit for clusters/provider1flux+cluster1/context/516641987681981267…
now
flux-system
Add Flux sync manifests
6 minutes ago
516641987681981267 8.yaml
Commit for clusters/provider1flux+cluster1/context/516641987681981267…
11 seconds ago
kcust516641987681981267 8.yaml
Commit for clusters/provider1flux+cluster1/context/516641987681981267…
11 seconds ago
© 2022 GitHub, Inc.
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About

CNF Lifecycle Management

This section shows how CNFs can be configured and managed from AMCOP, once they are deployed on the target k8s cluster(s).

AMCOP uses a combination of the underlying k8s cluster, EMCO and CDS (Controller Design Studio) for supporting CNF Lifecycle management functionality.

Following are the requirements for LCM (Life cycle management) features of AMCOP, and how they are supported.

Feature	CDS	EMCO/k8s
Day-0 Configuration		
Override defaults		✓ (Profile / Override / k8s CfgMap)
Day-N Configuration		
Design Data Dictionary	✓	
Run time resolution	✓	
UI for generating day-0 config (service profile/overrides)	Future (CDS design time GUI)	
UI for modeling BluePrints for day-N config	Future (CDS design time GUI)	
PNF Configuration (eg., RAN DU)	✓	
North-bound interface for LCM	✓	
South-bound interface for LCM		

Netconf/Yang	✓	
RESTconf	✓	
Ansible	✓	
Chef	✓	
Scripting (Python/Kotlin)	✓	
Complex workflows (DGs)	✓	
Container image management/versioning		✓
Upgrade/Downgrade		✓
Scale-up		✓
Scale-out		✓
Healing		✓

CDS Design time GUI (which will be supported in a future release) will generate a profile/overrides file, which can be used as input to day-0 config. This will make it seamless for the user, where they use CDS Design time for generating all the configuration related artifacts.

The user interface of AMCOP GUI is divided into two parts. One is for admin related functionalities like adding projects, onboarding clusters, adding controllers etc and the other for the service designer related functionalities like Creating service, instantiating service etc. The configuration of CNFs also follows a similar flow, using the Admin and Service User Personas.

The configuration is also divided into 2 parts:

1. Day-0 Configuration, which involves overriding certain configurations of the CNFs before orchestrating them on k8s cloud(s).
2. Day-N Configuration, which involves the ongoing configuration at run-time, and all the associated lifecycle management (Scaling up/down etc.).

Day-0 configuration

CDS (run time) does not play any role in Day-0 Configuration. The profiles and override (key-val pairs) provided are generated by CDS design tools (which currently, need to be generated offline using text editors), and these are input to the GUI/REST interface.

EMCO merges the original helm charts, profile and override values, and creates the initial (day-0) state (CfgMaps) for the CNFs.

Profiles and overrides provide a way to override values that are specified in the chart values.yaml. The profiles have to be created for every application.

Structure of Profile and Value Overrides

The format of the profile bundle is tar.gz, the following is the structure of a profile.

```
profile
├── manifest.yaml
├── override_values.yaml
```

manifest.yaml : The manifest.yaml defines the override values files. Override value files are the files that contain the key: value pair of the values that can be overridden. Following is an example of the manifest file,

```
cat manifest.yaml
```

```
---
version: v1
type:
  values: "override_values.yaml"
```

In this example, you can see that the manifest is pointing to a file called override_values.yaml, and the key is values. This means that the file override_values.yaml contains the key:value pairs of override values.

```
cat override_values.yaml
```

```
replicaCount: 4
```

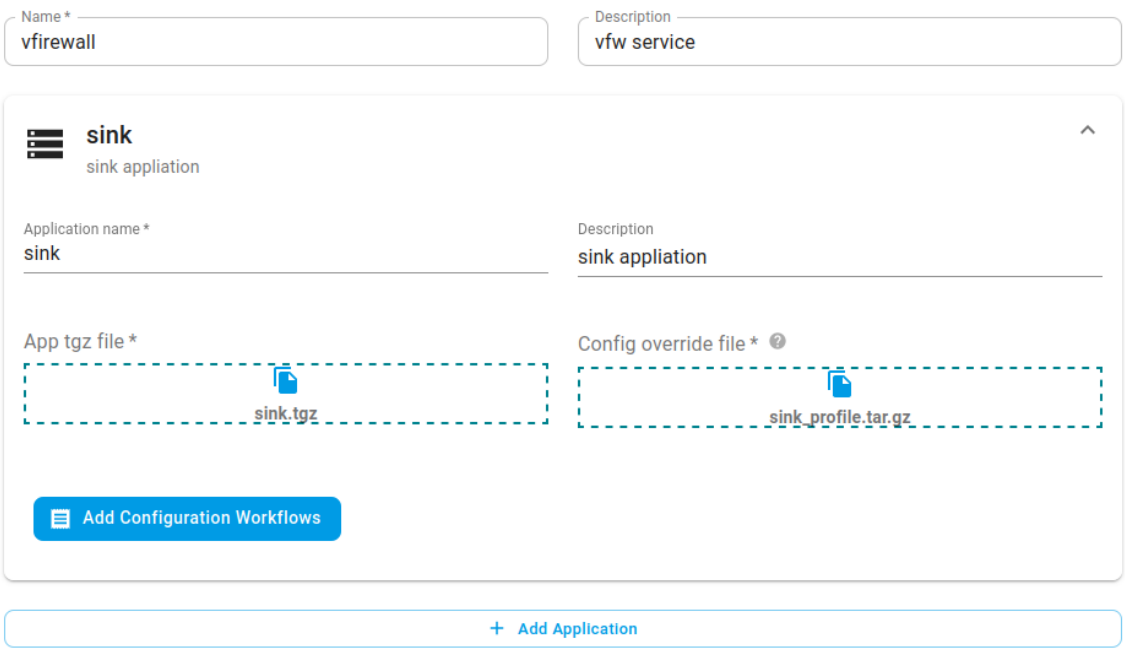
With this profile the value replicaCount in the chart Values.yaml will be overridden with 4.

You can copy the values.yaml of a helm chart to this override_values.yaml, and create a profile. With this mechanism, you can override any of the values in the values.yaml of a helm chart.

The following link contains the sample profiles for vFW CNFs that are used in the Orchestration section, as a reference.

Service Designer User

During the service design phase, you need to upload the profile bundles along with the application helm package, as shown in the screenshot below.



During the service instance creation phase, you can input the override JSON optionally if there is a need to override the default values in the helm chart.

Override Values at Service Instantiation Time

With these profiles uploaded for vfw applications, you can override the values defined in the *override_values.yaml* during the service instantiation time.

The override values during the instantiation time are specified as an array of jsons. The array members map to applications. For example in this case the override payload will be as follows.

```
{
 "app-name": "sink",
 "values": {
  "replicaCount": "2"
 }
}
```

After instantiation of the service, this will create a replica set of 2 for sink application.

Day-N configuration

Day-N configuration of CNF/CNA can be performed using AMCOP GUI or REST APIs. This section explains the steps using AMCOP GUI after the CNF/CNAs are orchestrated and instantiated on a k8s cluster (as explained in previous sections).

The configuration is a 2-step process.

1. Design of CBA (Configuration Blueprint Archive)
2. Run-time of configuration, which involves configuring the required parameters on a running instance

The following are some of the concepts of CDS which are needed to understand the process of configuration in AMCOP.

- Data Dictionary
  - A list of parameters that need to be “resolved” during runtime.
- Resolution
  - Providing a value to a configuration parameter during runtime. The source of this value can be varied, e.g. input, default, REST, SQL, and more.
- Configuration Provisioning
  - For more complex interactions with southbound APIs, CDS allows for workflows (in ONAP Directed Graph format) and scripts (Kotlin, Python)
- Modelling
  - Defining the data dictionary, resolution mechanism, workflows, and scripts
  - CDS uses TOSCA and JSON for modelling
  - Models can be designed to be reusable and xNF/ independent
  - Models are stored as CDS Blueprint Archive (CBA)

CBA Design

Note: The Design of CBA is done outside of AMCOP, using ONAP’s CDS project design tools. The following reference explains this process. This design workflow and necessary tools will be integrated into AMCOP in future releases.

Note:

The CBA designer should create a Mapping/Subordinate workflow to expose the NB API payload for the corresponding workflow.

The mapping/subordinate workflow name should be WORKFLOWNAME-schema. You may contact the Aarna support team for any further information.

The CBA design is explained in the reference material below.

- Follows TOSCA standards
- Should use CDS TOSCA JSON Models
    - https://github.com/onap/ccsdk-cds/tree/master/components/model-catalog/definition-type/starter-type
    - Artifact Type
    - Data Type
    - Node Type
    - Relationship Type
- Easy to extend the JSON TOSCA Model
- Runtime supports backward compatibility
    - CDS Models can be ported to a higher version of the CDS runtime
- Reference blueprints
    - https://github.com/onap/ccsdk-cds/tree/master/components/model-catalog/blueprint-model
- Data dictionaries for how to resolve resources
- Allows embedding other artifacts
    - Python and Kotlin scripts
    - Directed Graph, Ansible scripts
    - SO BPMN workflow
- Easy to create JSON model manually
    - One needs to know the CDS JSON model and Schema definitions
- Simple workflow
    - Resource Assignment (Pre instantiation use cases)
    - Configuration Assign (Post instantiation use cases)
        - Generate Day-0, Day-1 & Day-N configurations
    - Configuration Deployment
        - Runtime configuration changes on the target VNFs, PNFs & CNFs
- Controller Blueprint Archive
    - ZIP file of folders and files

```
# Load the data dictionary to the Database
bash -x ./dd-microk8s.sh ~/aarna-stream/cds-blueprints/vfw_netconf/Scripts/dd.json
```

The CBA format is summarized below:

```
├── Definitions
│   ├── blueprint.json                      Overall TOSCA service template (worfklow + node_template)
│   ├── artifact_types.json                 (generated by enrichment)
│   ├── data_types.json                     (generated by enrichment)
│   ├── node_types.json                     (generated by enrichment)
│   ├── relationship_types.json             (generated by enrichment)
│   └── resources_definition_types.json     (generated by enrichment)
├── Environments                            Contains *.properties files as required by the service
├── Plans                                   Contains Directed Graph
├── Tests                                   Contains uat.yaml file for testing the cba actions within
├── Scripts                                 Contains scripts
│   ├── python                              Python scripts
│   └── kotlin                              Kotlin scripts
├── TOSCA-Metadata
│   └── TOSCA.meta                          Meta-data of overall package
└── Templates                               Contains combination of mapping and template
```

Onboard CBA

Once the CBA is designed, it needs to be onboard onto AMCOP platform, using the following steps. This is done by the Admin user, and it is a one-time process for each service that is created. You can create as many CBAs as needed for each service, and onboard them to AMCOP using the following steps.

The below script will fetch the blueprint models, load the data dictionary into the database, zip the CBA for the Enrichment and save the Enriched CBA.

For example, in case of vfw_netconf CBA:

```
# Download the CBA from the following location and copy it in the
# user home folder ($HOME)in AMCOP VM

vfw_netconf.zip

# Extract the zip file
mkdir -p ~/vfw_netconf
mv vfw_netconf.zip ~/vfw_netconf
cd ~/vfw_netconf
unzip vfw_netconf.zip
```

```
# Replace “localhost” with “<Target Cluster IP>”
    # where vPG will be deployed.

    cd ~/vfw_netconf/Templates
    vi stream-count-config-edit-schema-template.vtl
    vi stream-count-config-get-schema-template.vtl

    cd ~/vfw_netconf/Scripts
    vi stream-count-config-edit-payload.json
    vi stream-count-config-get-payload.json


    cd ~/vfw_netconf
    Zip -r vfw_netconf.zip *

    cd ~/aarna-stream/cds-blueprints/k8s-utility-scripts/
    bash -x ./bootstrap-cds.sh

    bash -x ./dd-microk8s.sh ~/vfw_netconf/Scripts/dd.json

# Enrich the vfw_netconf .zip
    bash -x ./enrich-and-download-cds-blueprint.sh ~/vfw_netconf/vfw_netconf.zip

    # Save the Enrichment
    bash -x ./save-enriched-blueprint.sh /tmp/CBA/ENRICHED-CBA.zip
    bash -x ./get-cds-blueprint-models.sh
```

Note: For any other CBAs, users need to copy the CBA folder under: “~/aarna-stream/cds-blueprints” after logging into AMCOP VM.

Note: If the AMCOP Deployment is done through AMCOP Operator, user need to copy the aarna-stream folder to the home directory of AMCOP VM.

Once the CBAs are onboarded, the run-time configuration is integrated with AMCOP, and it can be performed using either AMCOP GUI or REST APIs.

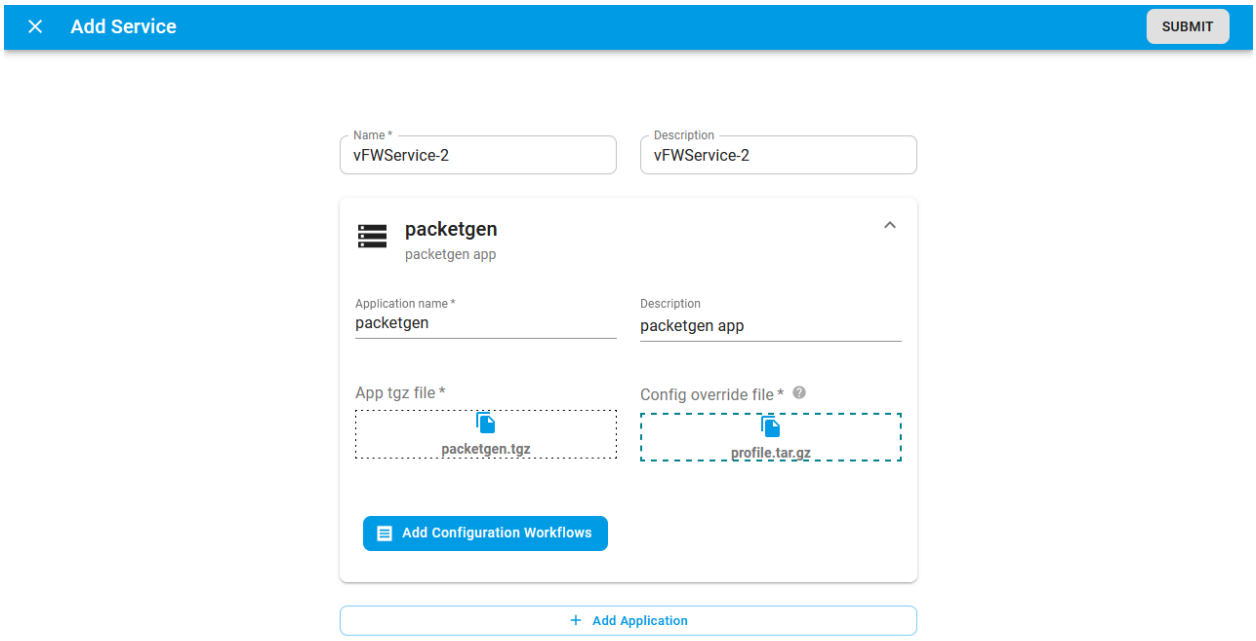
Configuration using AMCOP GUI

Design time

In design time, workflows are associated with applications. Also, the workflows are tagged according to their type, e,g GET, EDIT etc.

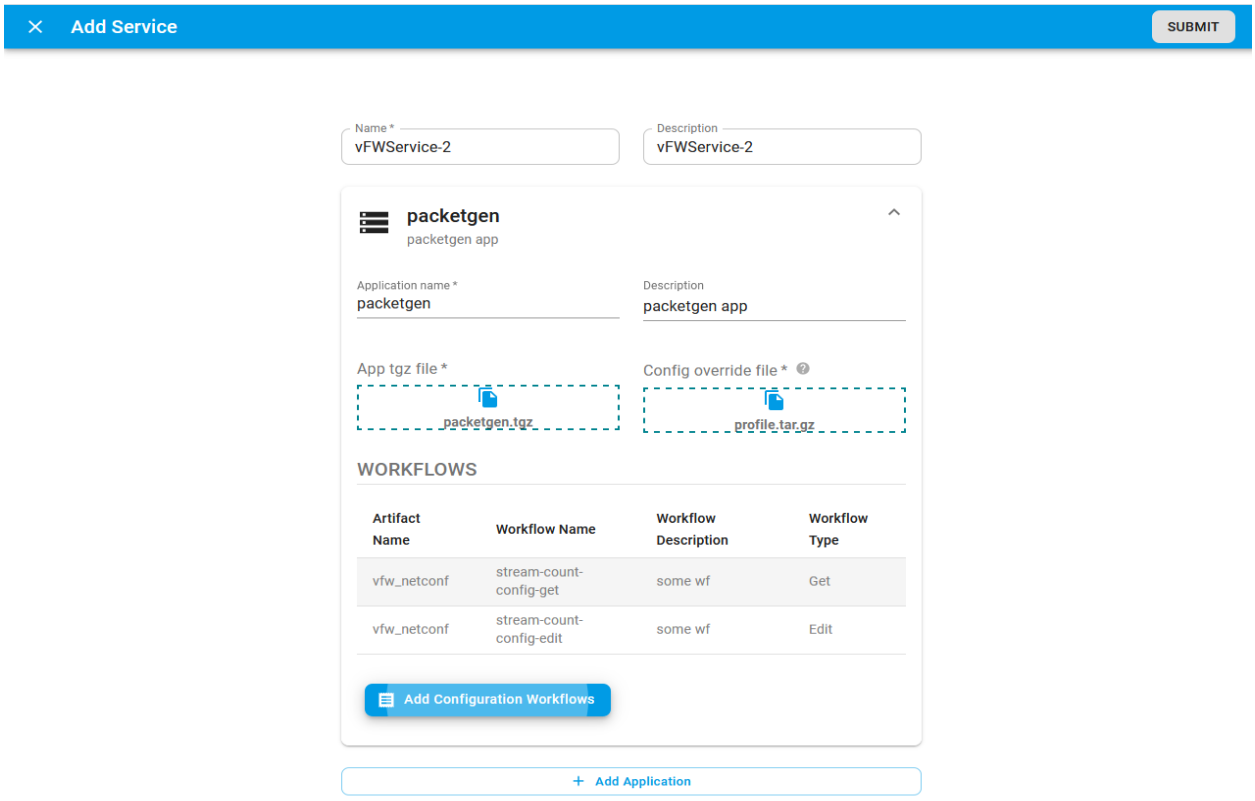
Click on *Add Configuration Workflows* in the app form. This shows all the workflows that have been onboarded to AMCOP through the corresponding CBAs. Then select the workflows which need to be tagged with the application. Select the *packetgen* app under the deployed vFW service and then select "*Add Configuration Workflows*".

> Note:
>
> Associating the CBA/Workflows with the application is a manual process at the moment. This will be made more seamless in future releases so that the association is done automatically.



After selecting "*Add Configuration Workflows*", you will be provided with the below screenshot, which is listing all the available CBA's. Now you can select "*vfw_netconf*" CBA and then further select the available workflows for the "*vfw_netconf*" CBA.

Note: users need to make sure, "Edit" is selected against *stream-count-config-edit* workflow. Now select *Ok.*



Now select *submit* at the top right corner.
Now users need to proceed with Service instance creation for the *packetgen* app similar to prior steps by adding the target cluster and networks to the *packetgen* app.

Now users can see the service instance. To instantiate the service instance, click on the instantiate button' ' in the actions column.

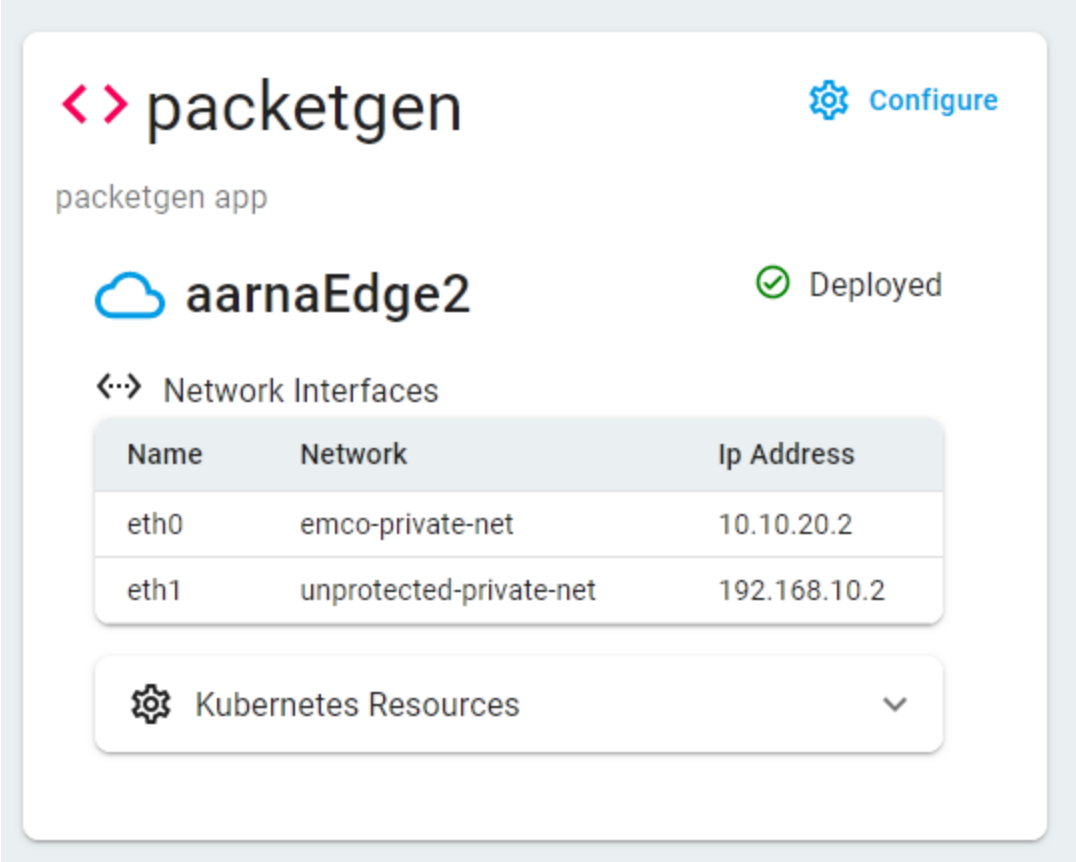
Note: Users need to wait for ~15 mins before the packetgen app cloud init to complete. Then the user can perform the Config GET workflow.

Run time

In runtime, the application can be configured by running the workflows which were tagged to the application during design time.

To configure an application, go to service details by clicking on the service instance name.

Then click on the *configure* button and select the cloud.



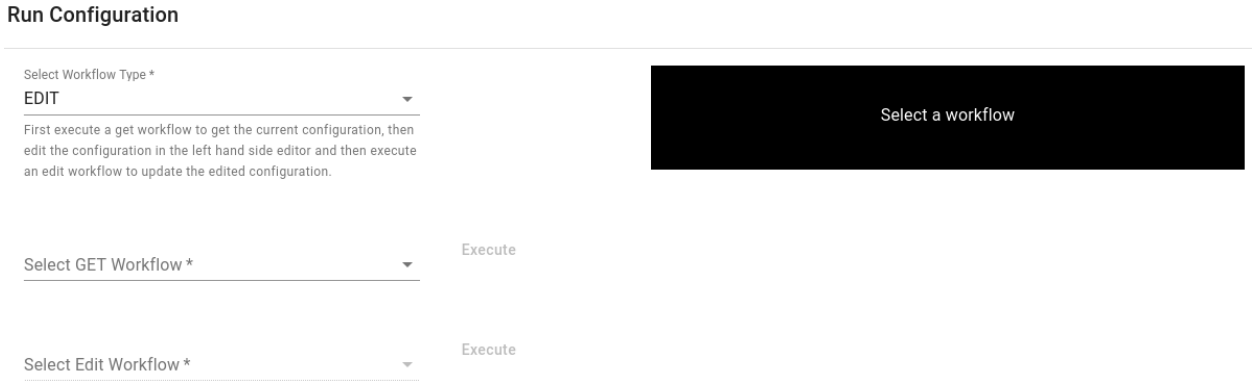
Config GET Workflow

Once the configure window comes up, first select the type of workflow which needs to be run, then select the workflow and click Execute.

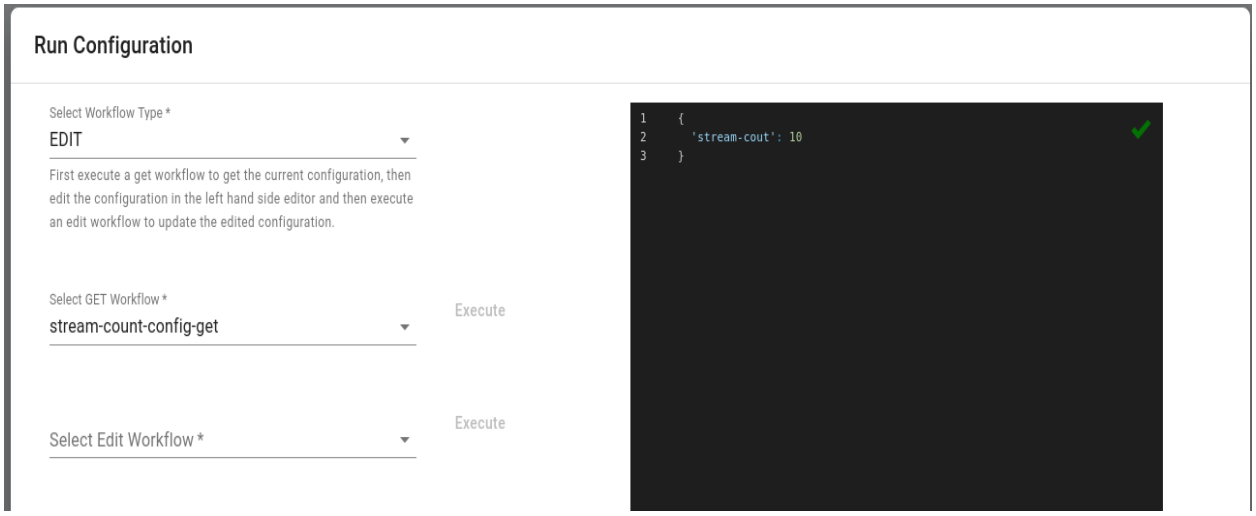
Clicking execute will trigger the workflow. In this example, we are triggering the CONFIG GET workflow, so as expected we will see the running configuration of the packetgen. We are fetching the pg streams config of the packet generator.

Config EDIT Workflow

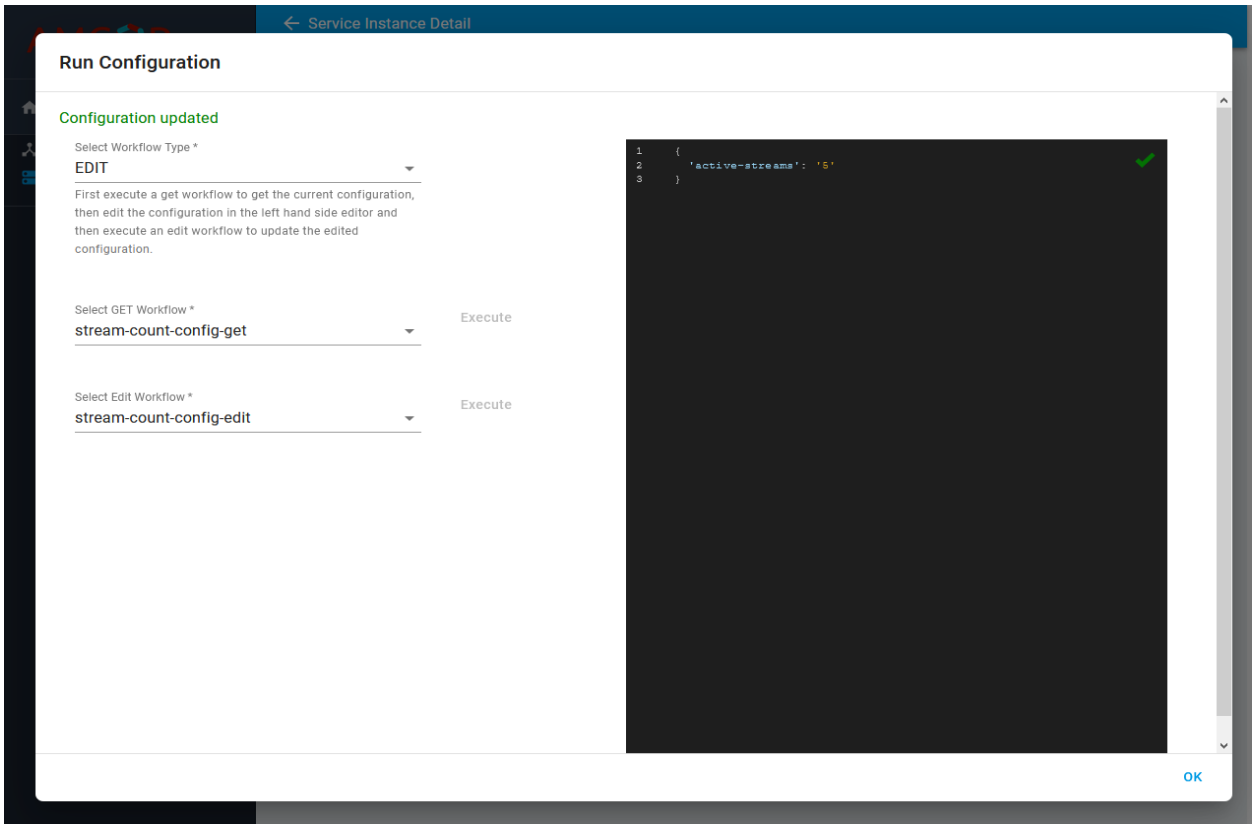
In order to edit the configuration, we will have to first get the current configuration, modify it and then execute the Edit workflow, following images show the flow,
Select EDIT, it will show the options as shown in the image,



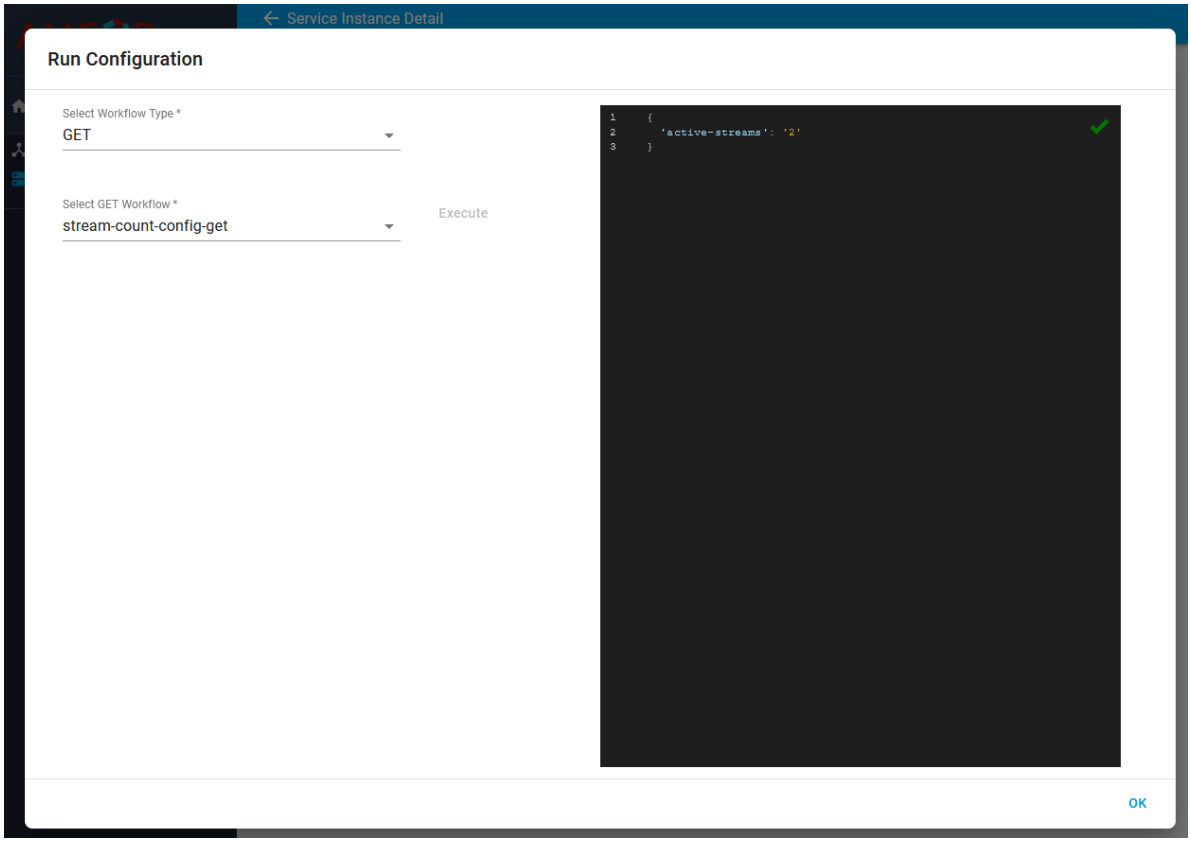
Select the GET, and execute to get the current config, “stream-count”: 10 in this example



Select the Edit workflow to execute, modify the JSON in the editor and execute the workflow. For example, in this example, the value of pg-streams is set to 5.



After getting the message "*Configuration updated*" as shown in the above screenshot, now the user can execute GET workflow to validate the updated configuration as shown below.



Configuration using REST API

The following steps can be used to perform the configuration using the REST API.

Executing the CBA
cd ~/aarna-stream/cds-blueprints/vfw_netconf/Scripts

# Set the BP service IP address
CDS_BP_SVC_IP=$(kubectl get svc -n amcop-system | grep 'cds-blueprints-processor-http' | awk '{print $3}')

# Payload for the Get Configuration
temp_get_file="stream-count-config-get-payload.json"

Now update the Target cluster IP address (highlighted below) where vPG is running.
vi stream-count-config-get-payload.json

```
{
      "actionIdentifiers": {
            "mode": "sync",
            "blueprintName": "vfw_netconf",
            "blueprintVersion": "1.0.0",
            "actionName": "stream-count-config-get"
      },
      "payload": {
            "stream-count-config-get-request": {
                  "stream-count-config-get-properties": {
                        "pnf-id": "vfw PG",
                        "pnf-ipv4-address": "<Target cluster IP address>",
                        "netconf-password": "admin",
                        "netconf-username": "admin",
                        "netconf-server-port": "30831"
                        }
            }
      },
      "commonHeader": {
            "subRequestId": "143748f9-3cd5-4910-81c9-a4601ff2ea58",
            "requestId": "e5eb1f1e-3386-435d-b290-d49d8af8db4c",
            "originatorId": "SDNC_DG"
      }
}
```

```
# Execute curl command for the config-deploy action
curl -v --location --request POST http://${CDS_BP_SVC_IP}:8080/api/v1/execution-service/process \
--header 'Content-Type: application/json;charset=UTF-8' \
--header 'Accept: application/json;charset=UTF-8,application/json' \
--header 'Authorization: Basic Y2NzZGthcHBzOmNjc2RrYXBwcw==' \
--header 'Host: cds-blueprints-processor-http:8080' \
--header 'Content-Type: text/json' \
--data  "@$temp_get_file" | python3 -m json.tool
```

Sample output:

```
            ...
            {
   "correlationUUID": null,
   "commonHeader": {
      "timestamp": "2021-02-08T11:44:02.926Z",
      "originatorId": "SDNC_DG",
      "requestId": "e5eb1f1e-3386-435d-b290-d49d8af8db4c",
      "subRequestId": "143748f9-3cd5-4910-81c9-a4601ff2ea58",
```

```
    "flags": null
  },
  "actionIdentifiers": {
    "blueprintName": "vfw_netconf",
    "blueprintVersion": "1.0.0",
    "actionName": "stream-count-config-get",
    "mode": "sync"
  },
  "status": {
    "code": 200,
    "eventType": "EVENT_COMPONENT_EXECUTED",
    "timestamp": "2021-02-08T11:44:15.494Z",
    "errorMessage": null,
    "message": "success"
  },
  "payload": {
    "stream-count-config-get-response": {
      "resolved-payload": {
        "status": "success",
        "httpStatusCode": "200",
        "httpResponse": {
          "active-streams": "1"
        }
      }
    }
  }
}
```


```
# Payload for the Edit configuration
temp_edit_file="stream-count-config-edit-payload.json"


# Now update the Target cluster IP address (highlighted below) where vPG is running.
vi stream-count-config-edit-payload.json
# Also, you can edit the parameter “stream-count” as needed, to change it
```


```
{
    "actionIdentifiers": {
        "mode": "sync",
        "blueprintName": "vfw_netconf",
        "blueprintVersion": "1.0.0",
        "actionName": "stream-count-config-edit"
    },
```

```
    "payload": {
        "stream-count-config-edit-request": {
            "stream-count-config-edit-properties": {
                "pnf-id": "Packet Generator",
                "pnf-ipv4-address": "<Target cluster IP address>",
                "netconf-password": "admin",
                "netconf-username": "admin",
                "netconf-server-port": "30831",
                "stream-count": "2"
            }
        }
    },
    "commonHeader": {
        "subRequestId": "143748f9-3cd5-4910-81c9-a4601ff2ea58",
        "requestId": "e5eb1f1e-3386-435d-b290-d49d8af8db4c",
        "originatorId": "SDNC_DG"
    }
}


# Execute  curl command for config-deploy action
curl -v --location --request POST http://${CDS_BP_SVC_IP}:8080/api/v1/execution-service/process \
--header 'Content-Type: application/json;charset=UTF-8' \
--header 'Accept: application/json;charset=UTF-8,application/json' \
--header 'Authorization: Basic Y2NzZGthcHBzOmNjc2RrYXBwcw==' \
--header 'Host: cds-blueprints-processor-http:8080' \
--header 'Content-Type: text/json' \
--data  "@$temp_edit_file" | python3 -m json.tool
```

Sample output:

```
...
{
   "correlationUUID": null,
   "commonHeader": {
      "timestamp": "2021-02-08T11:49:35.331Z",
      "originatorId": "SDNC_DG",
      "requestId": "e5eb1f1e-3386-435d-b290-d49d8af8db4c",
      "subRequestId": "143748f9-3cd5-4910-81c9-a4601ff2ea58",
      "flags": null
   },
   "actionIdentifiers": {
      "blueprintName": "vfw_netconf",
      "blueprintVersion": "1.0.0",
      "actionName": "stream-count-config-edit",
      "mode": "sync"
```

```
  },
  "status": {
    "code": 200,
    "eventType": "EVENT_COMPONENT_EXECUTED",
    "timestamp": "2021-02-08T11:49:36.787Z",
    "errorMessage": null,
    "message": "success"
  },
  "payload": {
    "stream-count-config-edit-response": {
      "resolved-payload": {
        "status": "success",
        "httpStatusCode": "200",
        "httpResponse": {
          "active-streams": "2"
        }
      }
    }
  }
}
```


Closed-Loop Automation and Analytics Platform

The closed-loop automation and analytics platform enables you to monitor events, and take actions based on analysed data. The platform allows you to onboard big data applications for analyzing the events/alerts/telemetry data received from the applications which are orchestrated through AMCOP. The analytics application can also have the logic to detect anomalies, respond to alerts etc. and take auto corrective measures in order to avoid any disruptions to the services deployed in the target clouds.

The following components of the big data platform are used in AMCOP:

1. CDAP: CDAP is an application platform for building and managing data applications in hybrid and multi-cloud environments. It enables developers with data and application abstractions to accelerate the development of data applications, addressing a broader range of real-time and batch use cases.

   Note: The CDAP mechanism will be deprecated in future releases of AMCOP. Please use the method described in subsequent sections instead of building CDAP based applications.

2. DMAAP: DMAAP is a data bus based on Kafka. The event reporters, applications running on CDAP, Policy agents publish/subscribe to topics on the Kafka bus.
3. VES Collector: The VES Event Listener is capable of receiving any event sent in the VES Common Event Format.
4. VES Agent: This is the agent whose endpoint is used when subscribing to events from the application functions. For example, when subscribing to the NEF for 5G core, the endpoint of this service will be used. This service converts any message that it receives into VES format and pushes the message to the VES collector over the RESTful interface.

Generate Events/Alarms

A closed-loop process is created with xNFs (or the infrastructure) generating events/alarms, which are analyzed by the analytics application, and taking the appropriate action. There are multiple ways to receive the events.

AMCOP supports multiple ways and formats to receive the events:

- VES (VNF Event Streaming)
- HV-VES (High Volume VES)
- Prometheus
- Future: SNMP
- Future: Proprietary (which requires developing the collectors and onboarding them on AMCOP)

This requires the xNFS (or the physical/virtual infrastructure on which xNFs are running) to generate the events in the necessary format. The entity that generates these events is called the Agent (eg., VES agent, in case of VES events).

VES/HV-VES Events

The xNFs are required to generate the metrics in the VES format, and the remote endpoint will be the VES collector running in the AMCOP platform.

AMCOP VES collector is VES 7.2 compliant.

The VES format spec can be found at the following location:
https://docs.onap.org/projects/onap-vnfrqts-requirements/en/latest/Chapter8/ves_7_2/ves_event_listener_7_2.html

Prometheus

The events can also be collected by configuring Prometheus. The Prometheus service section of this document describes how to deploy the Prometheus and Kafka adapter in order to route the metrics to the DMAAP bus in AMCOP. The CDAP application can consume these metrics and device analytics based on the use case.

Deploy Closed Loop

The closed-loop process is deployed after the analytics application is ready to receive events, and the agent is ready to send events. The loop comes into force as soon as the VES alerts from the xNFs start to flow into the VES collector.

Please note that the action depends on specific use cases, and hence these are not documented. The action can be triggered either of the following ways:

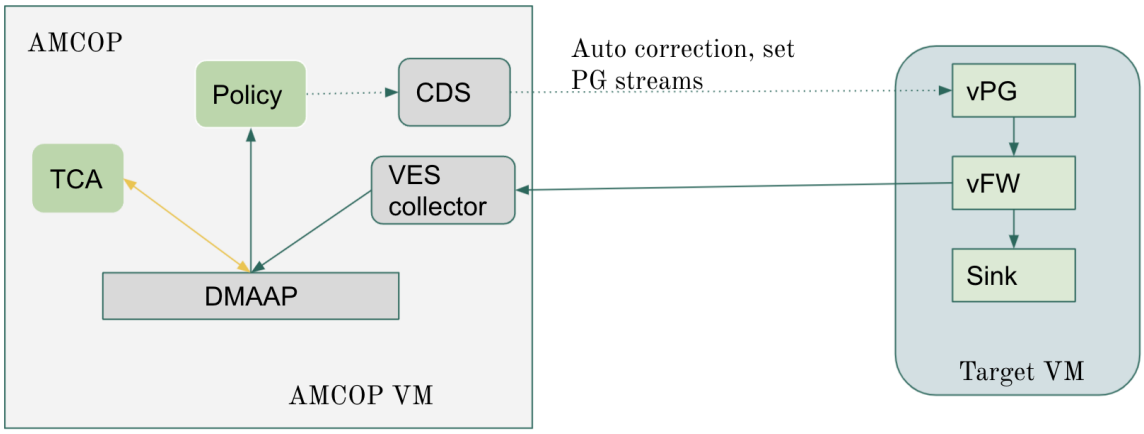
1. Invoke the config-modify API of CDS, which acts as the actor. This requires that the necessary CBAs (Controller Blueprint Archives) need to be developed and onboarded on AMCOP. This process is not documented here, and you can refer to the CDS documentation of ONAP.
2. Invoke the API end-point of the target application directly (eg., if it exposes the REST end-point).

Verify Closed Loop actions

The closed-loop process runs in an automated manner, and it can be verified if it is doing what it is intended to do, by looking at the actions (which will be triggered based on the events that are generated).

Closed Loop with TCA Gen 2

This section describes how the closed loop can be designed and deployed in the AMCOP platform with TCA Gen 2 as the analytics application. It uses vFW as an example.



Note: Assuming that the user has followed the vFW CBA onboarding steps already, this section describes only the design and deployment of the closed loop service.

vFirewall VES Reporting Application Configuration

Make sure that the VES reporting application in the vFW pod is pointing to the correct VES collector.

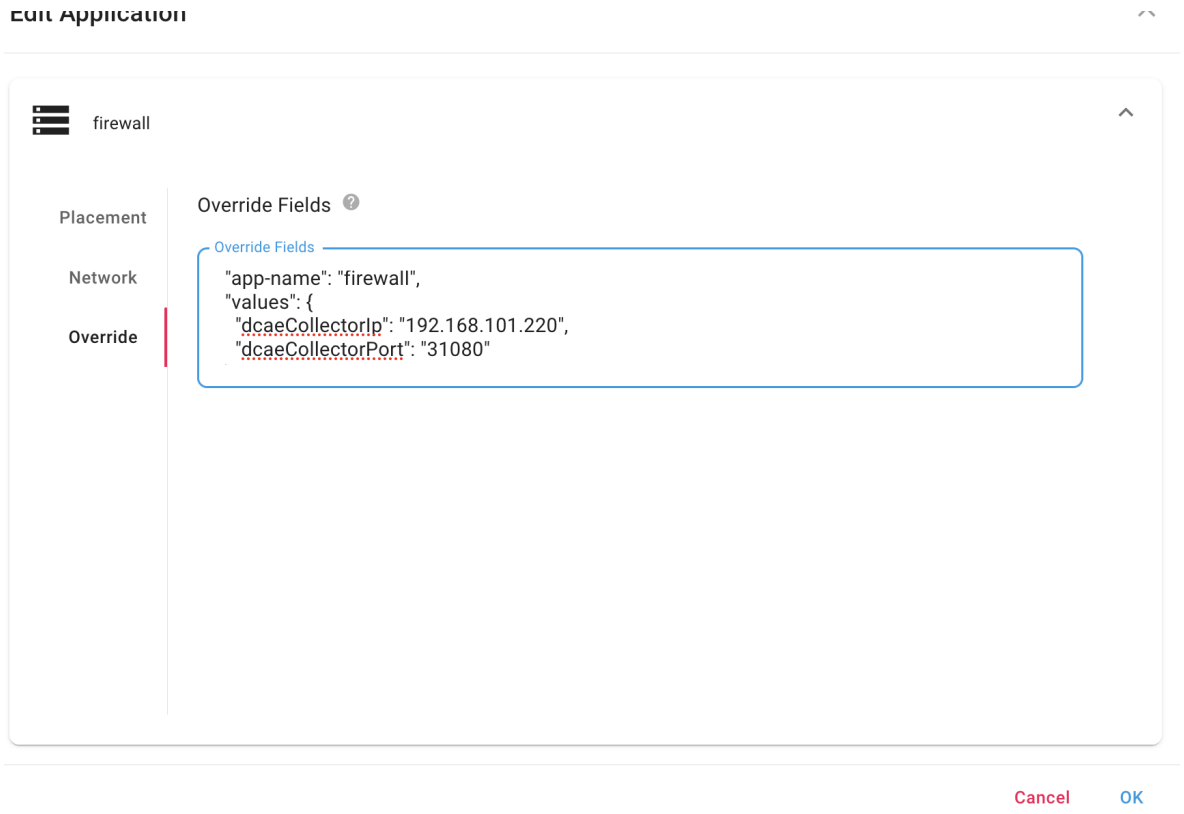
In the values.yaml of the firewall helm chart, make sure the following are setup correctly:

dcaeCollectorIp: 192.168.101.220
dcaeCollectorPort: 31080

In this example 192.168.101.220 is the external IP of the AMCOP node, and 31080 is the NodePort of the Ves Collector service.

These values can be overridden during the creation of service instances. The override json for the vfirewall application would be:

```
{
  "app-name": "firewall",
  "values": {
    "dcaeCollectorIp": "192.168.101.220",
    "dcaeCollectorPort": "31080"
  }
}
```



Closed Loop Service design and deployment

The closed loop service will consist of two applications,

1. TCA gen 2
2. Policy microservice

TCA helm charts can be dowloaed from
https://drive.google.com/file/d/1nuKP8_O8jG3j9C7cWL6k_cPZOtgZyhSw/view?usp=sharing

Policy microservice helm charts can be downloaded from
https://drive.google.com/file/d/1kdmuhLZJHC7y7b-4N5hV2KwZBJvDdO5D/view?usp=sharing

Profile helm chart can be download from
https://drive.google.com/file/d/1lLBYal4c4DbXv7qcUnNaJro6mWZiIM6d/view?usp=sharing

Follow the service design steps as documented in the above sections for the vFW application.

Service Instance Creation and Day 0 config

1. Placement Intent : For both applications, select the AMCOP cluster for deployment.
2. Day 0 Config and configmap.
    a. TCA Gen 2 : For the TCA application, the configurable parameters are the subscriber kafka topic, publisher kafka topic and the thresholds for firewall traffic. These can be configured in the configmap of the TCA application helm chart.
    Please Note that until the GAC (generic action controller) is integrated with AMCOP this has to be configured manually in the helm chart, because these are nested values.

```
 "streams_publishes": {
          "tca_handle_out": {
            "dmaap_info": {
              "client_role": "publisher",
              "client_id": "tca-pub-0",
              "location": "ecomp",
              "topic_url":
"http://192.168.101.220:32392/events/unauthenticated.DCAE_CL_OUTPUT"
            }
          }
        },
        "streams_subscribes": {
          "tca_handle_in": {
            "type": "message_router",
            "dmaap_info": {
              "topic_url":
"http://192.168.101.220:32392/events/unauthenticated.SEC_MEASUREMENT_
OUTPUT"
            },
            "consumer_group": "CG1",
            "consumer_ids": [
              "C0",
              "C1"
            ],
```

For the thresholds configure the tca.policy in the configmap (you are not required to change the below for the default use case)

"policy":
"{\"domain\":\"measurementsForVfScaling\",\"metricsPerEventName\":[{\"eventName\":\"vFirewallBroadcastPackets\",\"controlLoopSchemaType\":\"VM\",\"policyScope\":\"DCAE\",\"policyName\":\"DCAE.Config_tca-hi-lo\",\"policyVersion\":\"v0.0.1\",\"thresholds\":[{\"closedLoopControlName\":\"ControlLoop-vFirewall-d0a1dfc6-94f5-4fd4-a5b5-4630b438850a\",\"version\":\"1.0.2\",\"fieldPath\":\"$.event.measurementsForVfScalingFields.vNicPerformanceArray[*].receivedTotalPacketsDelta\",\"thresholdValue\":300,\"direction\":\"LESS_OR_EQUAL\",\"severity\":\"MAJOR\",\"closedLoopEventStatus\":\"ONSET\"},{\"closedLoopControlName\":\"ControlLoop-vFirewall-d0a1dfc6-94f5-4fd4-a5b5-4630b438850a\",\"version\":\"1.0.2\",\"fieldPath\":\"$.event.measurementsForVfScalingFields.vNicPerformanceArray[*].receivedTotalPacketsDelta\",\"thresholdValue\":700,\"direction\":\"GREATER_OR_EQUAL\",\"severity\":\"CRITICAL\",\"closedLoopEventStatus\":\"ONSET\"}]},{\"eventName\":\"vLoadBalancer\",\"controlLoopSchemaType\":\"VM\",\"policyScope\":\"DCAE\",\"policyName\":\"DCAE.Config_tca-hi-lo\",\"policyVersion\":\"v0.0.1\",\"thresholds\":[{\"closedLoopControlName\":\"ControlLoop-vDNS-6f37f56d-a87d-4b85-b6a9-cc953cf779b3\",\"version\":\"1.0.2\",\"fieldPath\":\"$.event.measurementsForVfScalingFields.vNicPerformanceArray[*].receivedTotalPacketsDelta\",\"thresholdValue\":300,\"direction\":\"GREATER_OR_EQUAL\",\"severity\":\"CRITICAL\",\"closedLoopEventStatus\":\"ONSET\"}]},{\"eventName\":\"Measurement_vGMUX\",\"controlLoopSchemaType\":\"VNF\",\"policyScope\":\"DCAE\",\"policyName\":\"DCAE.Config_tca-hi-lo\",\"policyVersion\":\"v0.0.1\",\"thresholds\":[{\"closedLoopControlName\":\"ControlLoop-vCPE-48f0c2c3-a172-4192-9ae3-052274181b6e\",\"version\":\"1.0.2\",\"fieldPath\":\"$.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\"}]}]}"

b. Policy microservice : The policy microservice needs to know the Kafka endpoint, the target CNF (vPG) IP and port. These values can be provided as day 0 config during the service instance creation,

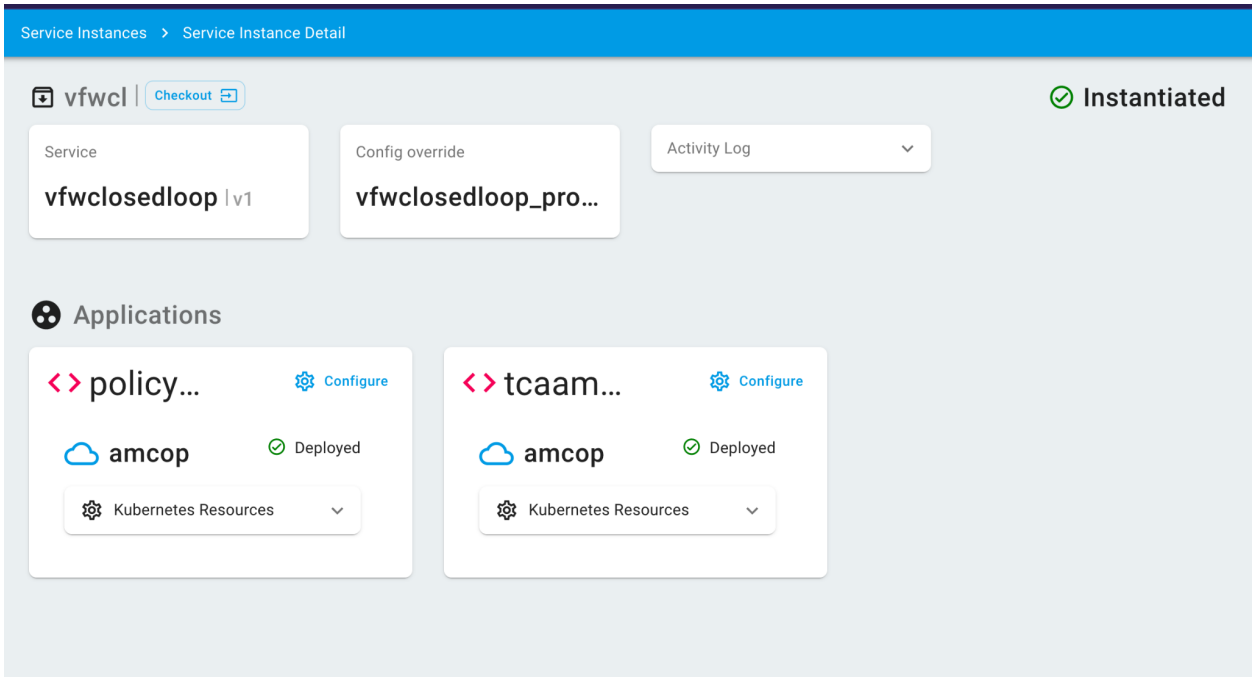
```
{
        "app-name":"policyms",
        "values": {
                "cnfIp": "192.168.102.81",
                "kafkaTopic": "unauthenticated.DCAE_CL_OUTPUT",
```

```
                "cnfPort": "30831"
            }
        }
```

The CDS endpoint etc is configured in the configmap of the policy microservice helm chart. The namespace values can also be overridden in day 0 configuration,

```
 "cds": "cds-blueprints-processor-http.{{ .Values.cdsNamespace }}.svc.cluster.local:8080",
```

Instantiate the service and the closed loop will come into action as soon as the tca and policy comes up.



Closed loop Validation

We can execute the CDS REST API to get the pg stream information, and we should see the value set as 5 most of the time,

```
curl  POST http:/192.168.101.220:30169/api/v1/execution-service/process --header 'Content-Type: application/json;charset=UTF-8' --header 'Accept: application/json;charset=UTF-8,application/json' --header 'Authorization: Basic Y2NzZGthcHBzOmNjc2RrYXBwcw==' --header 'Host: cds-blueprints-processor-http:8080' --header 'Content-Type: text/json' --data  @./pg_payload.json | jq
```

```
{
 "commonHeader": {
  "timestamp": "2021-10-31T08:05:56.691Z",
  "originatorId": "SDNC_DG",
  "requestId": "e5eb1f1e-3386-435d-b290-d49d8af8db4c",
  "subRequestId": "143748f9-3cd5-4910-81c9-a4601ff2ea58",
  "flags": null
 },
 "actionIdentifiers": {
  "blueprintName": "vfw_netconf",
  "blueprintVersion": "1.0.0",
  "actionName": "stream-count-config-get",
  "mode": "sync"
 },
 "correlationUUID": null,
 "status": {
  "code": 200,
  "eventType": "EVENT_COMPONENT_EXECUTED",
  "timestamp": "2021-10-31T08:05:59.270Z",
  "errorMessage": null,
  "message": "success"
 },
 "payload": {
  "stream-count-config-get-response": {
   "resolved-payload": {
    "status": "success",
    "httpStatusCode": "200",
    "httpResponse": {
     "active-streams": "5"
    }
   }
  }
 }
}
```

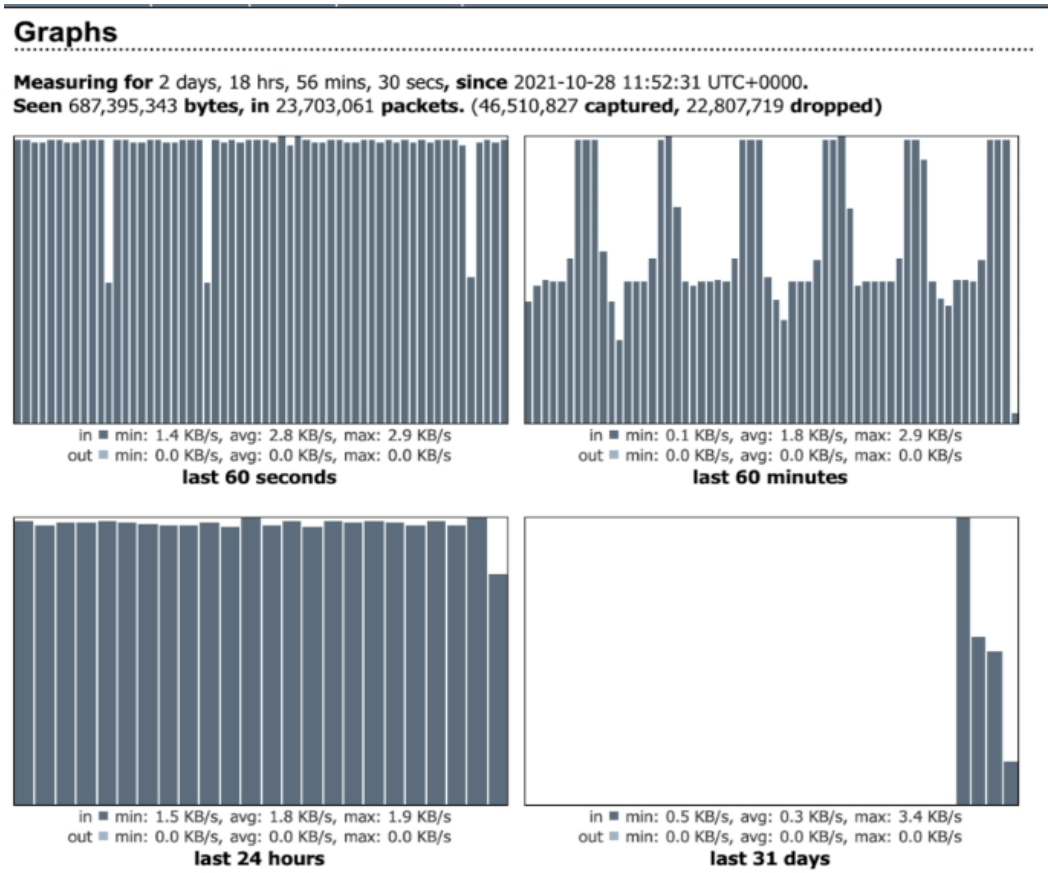

pg_payload.json


```
{
      "actionIdentifiers": {
            "mode": "sync",
            "blueprintName": "vfw_netconf",
            "blueprintVersion": "1.0.0",
            "actionName": "stream-count-config-get"
      },
      "payload": {
```

```
            "stream-count-config-get-request": {
                "stream-count-config-get-properties": {
                    "pnf-id": "vfw pg",
                    "pnf-ipv4-address": "192.168.102.81",
                    "netconf-password": "admin",
                    "netconf-username": "admin",
                    "netconf-server-port": "30831"
                    }
            }
        },
        "commonHeader": {
            "subRequestId": "143748f9-3cd5-4910-81c9-a4601ff2ea58",
            "requestId": "e5eb1f1e-3386-435d-b290-d49d8af8db4c",
            "originatorId": "SDNC_DG"
        }
    }
```


Note : vPG IP and netconf port

The sink graphs also become more stable with a lesser number of spikes, as shown below.



Policy Engine (Early Access Support)

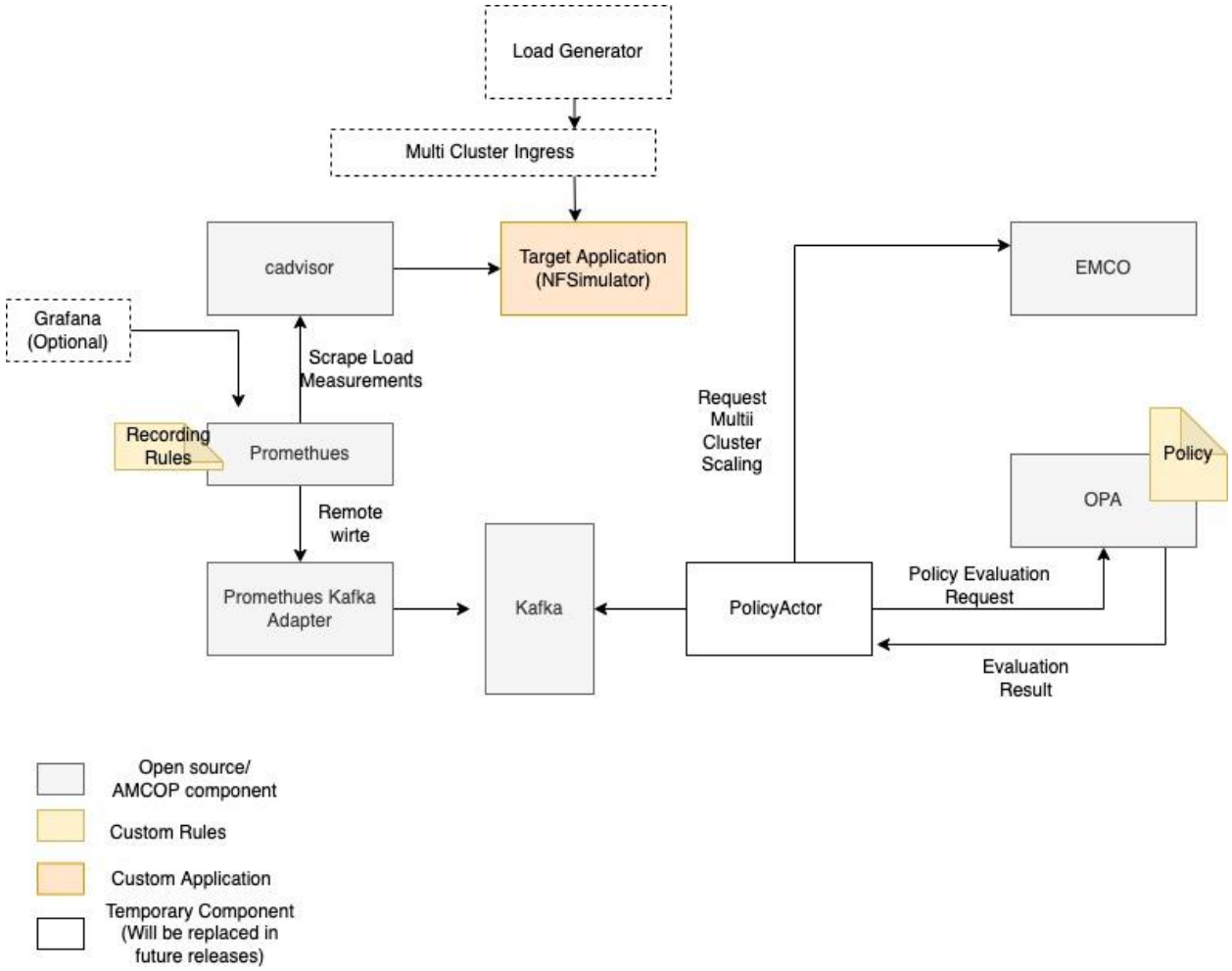
AMCOP is using Open Policy Agent (OPA) as the policy engine. OPA is a lightweight, general-purpose policy engine.

In AMCOP deployment, OPA engine is distributed across the target clusters. OPA engine will be deployed as a microservice running in the target cluster, similar to Prometheus.

OPA chart is available at:
https://drive.google.com/drive/folders/1q2GoZiz2EiVAe2AGXjsuYw3KI98bOhN7?usp=sharing

This chart uses policy as a config map. If you need to use a custom policy, update the config map, cpu-policy. The distribution of policy will be managed centrally by AMCOP in future releases.
AMCOP uses Rego as policy language. Use the following guide to develop Rego based policy.
https://www.openpolicyagent.org/docs/latest/policy-language/

Closed-loop Using OPA engine



This example explains a closed-loop using the OPA engine.

Packages for this closed-loop are available at
https://drive.google.com/drive/folders/1q2GoZiz2EiVAe2AGXjsuYw3KI98bOhN7?usp=sharing

Deploy following packages in target cluster using AMCOP:

- opa.tgz
- prometheus.tgz
- prometheus-kafka-adapter.tgz
- nfsimulator.tgz
- policyactor.tgz
- grafana.tgz (optional)

Generate load to nfsimulator by the following command:

while :; do curl http://<node-ip>:30012/loadcpu; done

loadcpu endpoint of *nfsimulator* will create CPU load on the container.

The remaining steps (below) are done as part of closed loop operation, without any manual intervention.

1. Prometheus scapes container CPU usage (of all containers in the cluster)
2. Prometheus is configured to use Kafka Adapter as remote write. Hence these measurements are sent to the Adapter.
3. Adapter sends these to Kafka endpoint in the required format
4. PolicyActor subscribes to Kafka topic. It receives measurements in JSON format.
5. PolicyActor forwards measurements to OPA
6. Based on the policy, OPA evaluate the measurements and respond to PolicyActor with the result
7. Policy Actor takes the action (Send scale-up request to EMCO) if the threshold limit set in policy is crossed

Writing Policy

AMCOP uses Rego as policy language. Rego is a very simple declarative language, with a syntax similar to Go. Rego takes two input files for evaluation, which is referenced as "Input" and "data" in the policy language.

Input - mandatory input json.
Data - optional data in json format

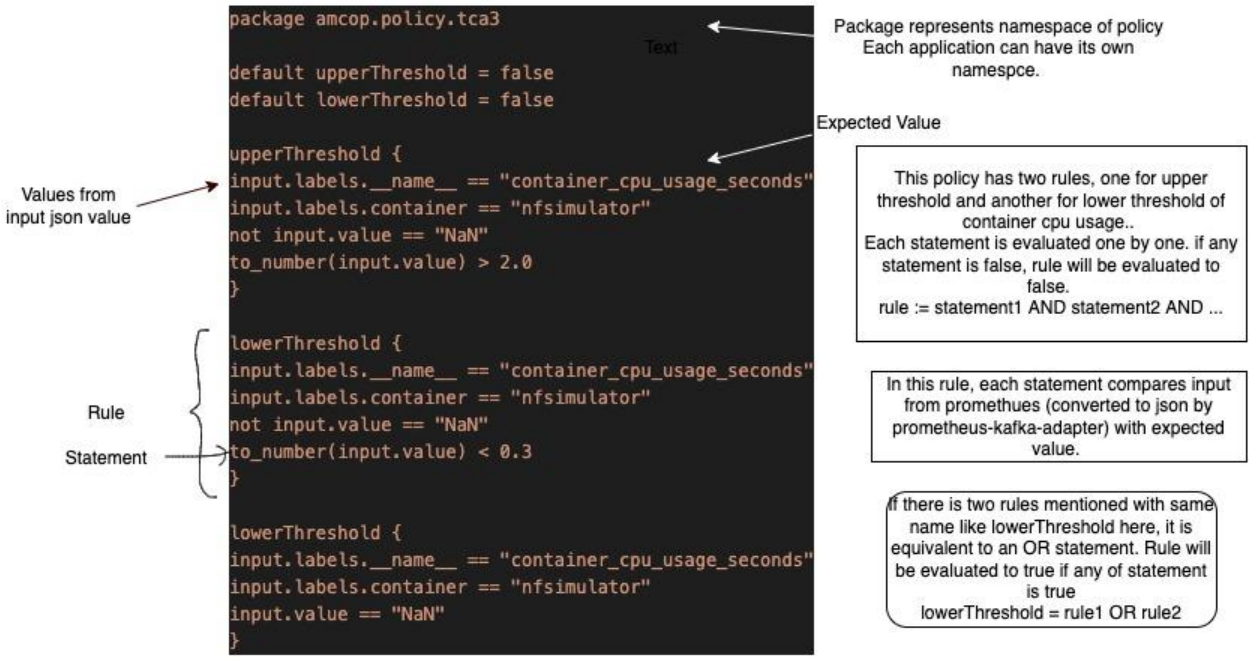
A simple rule in Rego is of following format

```
rule_name := value {
        expression-1.
        expression-2
        expression-3
         ....
}
```

A rule can refer to other rules. Result of rule evaluation can be simple Boolean or a complex JSON structure.

For testing and trying out your policy, use Rego playground (https://play.openpolicyagent.org/).
Detailed language description is available at the following location:
https://www.openpolicyagent.org/docs/latest/policy-language/

The policy for the example closed loop mentioned above is explained below (this is defined in opa.tgz bundle mentioned above).



An example of a policy that produces a json as output is explained below.

Defines custom message

```
package amcop.policy.vfw

alert = [msg] {
   input.domain = "measurementsForVfScaling"
   some i,j,k
   input.metricsPerEventName[i].eventName = "vFirewallBroadcastPackets"
   input.metricsPerEventName[i].thresholds[j].direction = "GREATER"
   input.metricsPerEventName[i].thresholds[j].vNicPerformanceArray[k].receivedTotalPacketsDelta >
70
   alertStatus := input.metricsPerEventName[i].thresholds[j].closedLoopEventStatus
   msg := sprintf("Event %v is %v", [input.metricsPerEventName[i].eventName, alertStatus])


}


actor = mk{
   input.policyType = "monitoring"
   mk := sprintf("%v", [data.actor])
}


endPoint = {"ip" : ip, "port" : port} {
   data.actor = "kafka"
   ip := sprintf("%v", [data.kafkaIp])
   port := sprintf("%v", [data.kafkaPort])
}
```

Actor IP and Port

```
endPoint = {"ip" : ip, "port" : port} {
   data.actor = "cds"
   ip := sprintf("%v", [data.cdsIp])
   port := sprintf("%v", [data.cdsPort])
}
```

This will be provide an evaluation of following format (Depending on input and data **)**

```
{
    alert : “Event vFirewallBroadcastPackets is ONSET”
    actor: “Kafka”
    endPoint : {
        ip : “192.1.2.3”
        port: “8032”
    }
}
```

CDS as closed loop actor

In the bundle provided k8s and emco is used as actor. The ‘policy actor’ (policyactor.tgz) uses k8s/emco APIs for executing the actions, like scaleout of application. If you want to use CDS as actor, the policyactor need to be replaced with an application that call the endpoint of CDS with necessary parameter. Other steps for AMF scaleout use case with CDS is as follows:

The post-operation-cba-amf-scaleout.zip CBA and aarna-stream files can be downloaded from the following link:

post-operation-cba-amf-scaleout.zip
aarna-stream.tgz

Extract aarna-stream.tgz in the HOME directory.

Create scaleout-cba folder
mkdir scaleout-cba

move post-operation-cba-amf-scaleout.zip to scaleout-cba folder
mv post-operation-cba-amf-scaleout.zip ~/scaleout-cba/

Unzip the ZIP file
unzip post-operation-cba-amf-scaleout.zip

Follow these instructions in order to onboard the CBA,

Steps to load and execute the CBA

cd ~/aarna-stream/cds-blueprints/k8s-utility-scripts/
bash -x ./bootstrap-cds.sh

Load the data dictionary to the Database
bash -x ./dd-microk8s.sh ~/scaleout-cba/Scripts/dd.json

Enrich the vfw_netconf .zip
bash -x ./enrich-and-download-cds-blueprint.sh
~/scaleout-cba/post-operation-cba-amf-scaleout.zip

Save the Enrichment
bash -x ./save-enriched-blueprint.sh /tmp/CBA/ENRICHED-CBA.zip
bash -x ./get-cds-blueprint-models.sh

Please note that in the current release, integration of policy and CDS is experimental, and requires few manual steps. Future releases of AMCOP will have a full integration of Policy with EMCO and CDS.

Prometheus and Grafana Orchestration

This section describes how to enable Telemetry service for the target cluster monitoring, using Prometheus and Grafana.

It assumes you have already created clusters and the tenant. Refer to the Section on Orchestration of vFirewall using AMCOP GUI.

Target Kubernetes Cluster and Host Server resource monitoring is done by Prometheus federate(scrape approach). Below are the configurations required for the Prometheus Federate(scrape approach)

Target Cluster (AMCOP installed Cluster)

Prometheus Install on AMCOP cluster

Prometheus deployment is helm based deployment, follow the below steps for Prometheus installation on the AMCOP cluster.

Custom modified Prometheus Helm charts can be downloaded from the below link

amcop-prometheus

Note: This charts are customized from the Prometheus Git Repo : GitHub - prometheus-community/helm-charts: Prometheus community Helm charts

1. Copy the prometheus-target-amcop-cluster.tgz file to the AMCOP server, once copy untar the file.
2. Prometheus installation
   helm install prometheus prometheus-target-amcop-cluster.tgz
3. Prometheus pods and service verification

Verify all Prometheus pods are running

```
kubectl get pods
NAME                                          READY   STATUS    RESTARTS   AGE
prometheus-alertmanager-6764b6b758-jpcn9      2/2     Running   0          96s
prometheus-kube-state-metrics-7c6ffc7686-mkr8r   1/1     Running   0          96s
prometheus-node-exporter-qpb2z                1/1     Running   0          96s
prometheus-pushgateway-6bdd5f56cb-v8jrq       1/1     Running   0          96s
prometheus-server-77f6df8859-2kf6w            2/2     Running   0          96s
```

Verify all Prometheus services are started and Prometheus-Server service is exposed as NodePort on Port 30090

```
kubectl get svc
NAME                          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes                    ClusterIP   10.96.0.1        <none>        443/TCP        13d
prometheus-alertmanager       ClusterIP   10.103.224.138   <none>        80/TCP
99s
prometheus-kube-state-metrics   ClusterIP   10.100.32.164    <none>        8080/TCP
99s
prometheus-node-exporter      ClusterIP   10.104.226.83    <none>        9100/TCP
99s
prometheus-pushgateway        ClusterIP   10.108.6.231     <none>        9091/TCP
99s
prometheus-server             NodePort    10.110.223.204   <none>        80:30090/TCP
99s
```

4. Launch Prometheus server and verify targets are appearing.

open the Prometheus Portal on your browser suffixing the port to machine hostname/ip address as below

http://<ip-address>:30090/

http://<hostname>:30090/

Once Prometheus is launched click on Status-->Targets, and verify all Target state is UP
Note:: Make sure Prometheus Service NodePort is opened in your security groups inbound rule

Target Cluster Host machine monitoring using Node Exporter

1. Download the latest node exporter package. You should check the Prometheus downloads section for the latest version and update this command to get that package.
   cd /tmp
   curl -LO
   https://github.com/prometheus/node_exporter/releases/download/v0.18.1/node_exporter-0.18.1.linux-amd64.tar.gz

2. Unpack the tarball
   tar -xvf node_exporter-0.18.1.linux-amd64.tar.gz

3. Move the node export binary to /usr/local/bin
   sudo mv node_exporter-0.18.1.linux-amd64/node_exporter /usr/local/bin/

4. Create a Custom Node Exporter Service
   Create a node_exporter user to run the node exporter service.
   sudo useradd -rs /bin/false node_exporter

   Create a node_exporter service file under systemd.
   sudo vi /etc/systemd/system/node_exporter.service

   Add the following service file content to the service file and save it.
   [Unit]
   Description=Node Exporter
   After=network.target

   [Service]
   User=node_exporter
   Group=node_exporter
   Type=simple
   ExecStart=/usr/local/bin/node_exporter --web.listen-address=:9200

   [Install]
   WantedBy=multi-user.target

5. Reload the system daemon and start the node exporter service.
   sudo systemctl daemon-reload
   sudo systemctl start node_exporter

6. check the node exporter status to make sure it is running in the active state.
   sudo systemctl status node_exporter

7. Enable the node exporter service to the system startup.
   sudo systemctl enable node_exporter

   Now, node exporter would be exporting metrics on port 9200.

Verify metrics

Run curl command to verify metrics

1curl localhost:9200/metrics

Prometheus Stack installation

It is recommended that Prometheus Stack be run on a separate Kuberenetes cluster from where AMCOP is running, although it can be combined with ELK cluster

Below are the steps for Prometheus-stack deployment on a ubuntu18.04/ubuntu20.04

Prerequisite for Prometheus Stack installation:

- Docker
- Kuberenetes(Kubeadm, kubectl,kubelet)
- Helm

Prometheus stack deployment is helm based deployment. Please follow the below steps for Prometheus stack deployment.

Prometheus Repo: Custom modified Prometheus Helm charts can be downloaded from the below link

amcop-prometheus

Note: This charts are customized from the Prometheus Git Repo:

helm-charts/charts/kube-prometheus-stack at main

prometheus-community/helm-charts

1. Copy the kube-prometheus-stack.tgz file to the server on which Prometheus stack needs to be installed, once copy untar the file and update the server configurations as below.

Note: Please note down the AMCOP server IP address to update the prometheus stack configurations.

```
tar -xvf kube-prometheus-stack.tgz
##go to the kube-prometheus-stack directory
cd kube-prometheus-stack/
vi values.yaml
##update the additionalScrapeConfigs with the correct server IP
## on line 2688 change and update from 192.168.1.200 to correct AMCOP server IP
```

on line 2699 change and update from 192.168.1.200 to correct AMCOP server IP
if needs to add additional cluster/host monitoring update the IP config similar to amcop & amcop host monitoring
once IP details are updated save the file.

2. Prometheus installation
helm install prometheus .
##run from the kube-prometheus-stack directory

4. Prometheus Pod verification
Run the below command to verify all the Promethus-stack pods have started and running

kubectl get pods
NAME READY STATUS RESTARTS AGE
alertmanager-prometheus-kube-prometheus-alertmanager-0 2/2 Running 0 5d19h
prometheus-grafana-6948f99bb5-bkxzf 3/3 Running 0 5d19h
prometheus-kube-prometheus-operator-78dbdc7bb8-k7pzn 1/1 Running 0 5d19h
prometheus-kube-state-metrics-54c585df74-j2r97 1/1 Running 0 5d19h
prometheus-prometheus-kube-prometheus-prometheus-0 2/2 Running 0 5d19h
prometheus-prometheus-node-exporter-fmlpd 1/1 Running 0 5d19h

5. Prometheus Service verification
Run the below command to verify all the Promethus-stack services have started

kubectl get services
NAMETYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
prometheus-grafana NodePort 10.103.4.151 <none> 80:30080/TCP 28h
prometheus-kube-prometheus-alertmanager ClusterIP 10.101.59.15 <none> 9093/TCP 28h
prometheus-kube-prometheus-operator ClusterIP 10.100.195.99 <none> 443/TCP 28h
prometheus-kube-prometheus-prometheus NodePort 10.111.250.63 <none> 9090:30090/TCP 28h
prometheus-kube-state-metrics ClusterIP 10.103.158.138 <none> 8080/TCP 28h
prometheus-operated ClusterIP None <none> 9090/TCP 28h
prometheus-prometheus-node-exporter ClusterIP 10.103.5.208 <none> 9100/TCP 28h
10

Verify Grafana, and Prometheus services are nodePort.

5. Accessing Prometheus Grafana
   Access the Prometheus Grafana on port 30080
   open the Grafana Portal on your browser suffixing the port to machine hostname/ip address as below
   http://<ip-address>:30080/
   http://<hostname>:30080/
   Credentials to access Grafana portal
   username: admin   Password: prom-operator
   Note: Make sure you ports are opened in your security groups inbound rule

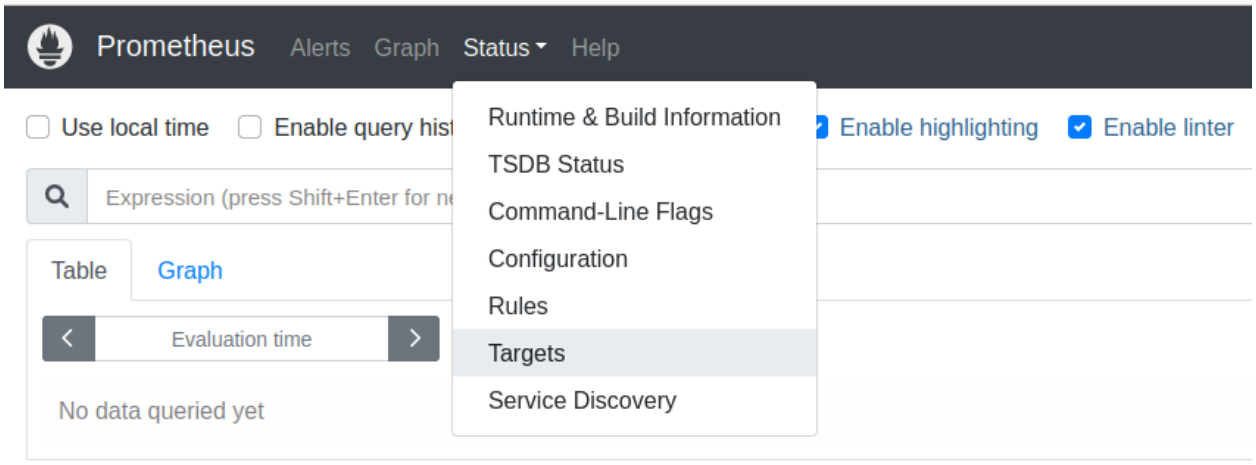
Note:

Kube Proemtheus Stack update for Server IP address changes or adding additional servers for monitoring
To add additional servers for monitoring or update the server IP details follow the below steps on the kube-prometheus stack server
On Prometheus Master Cluster(kube-prometheus-stack installed Cluster)
cd kube-prometheus-stack/
change directory to the kube-prometheus-stack directory
vi values.yaml
##update the additionalScrapeConfigs with the updated server IP
on line 2688 change and update to correct AMCOP server IP
on line 2699 change and update to correct AMCOP server IP
if needs to add additional cluster/host monitoring update the IP config similar to amcop & amcop host monitoring
8## once IP details are updated save the file.

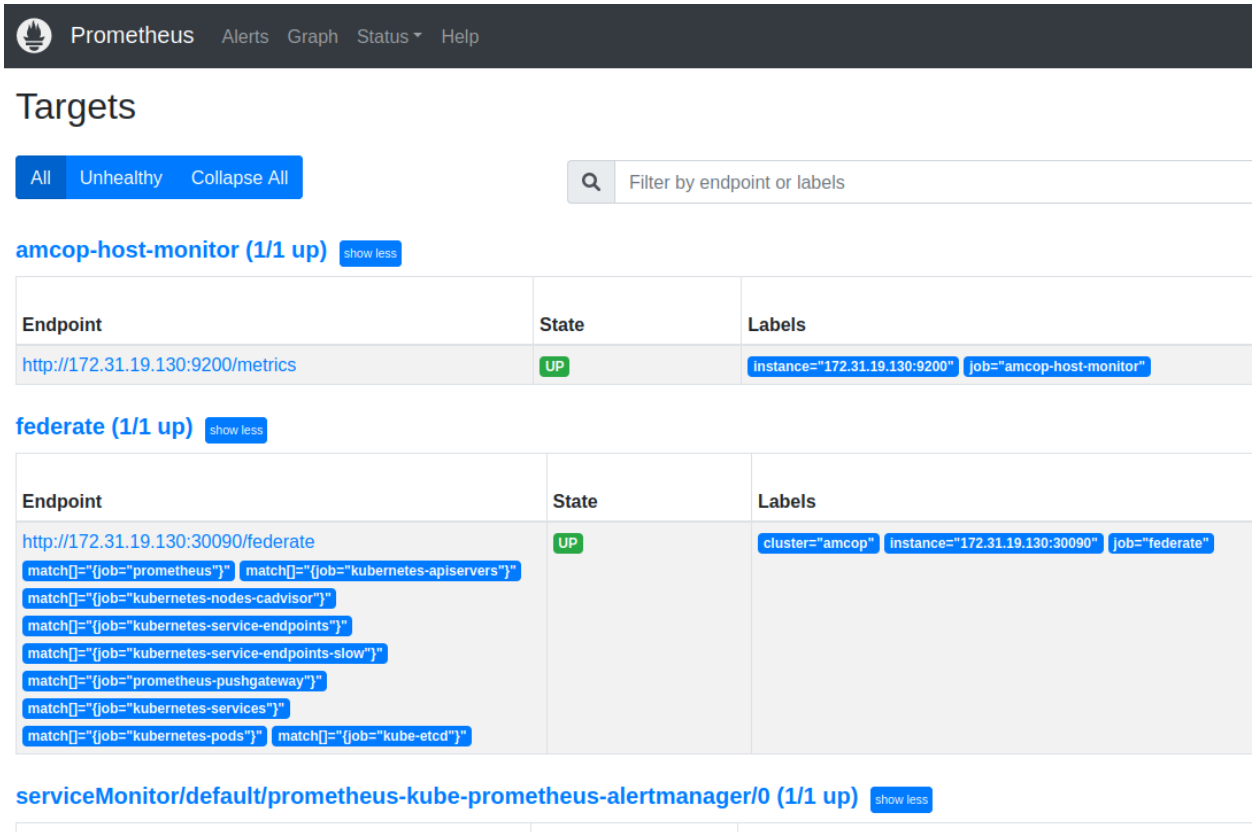
Helm upgrade for updating scrape config
helm upgrade prometheus kube-prometheus-stack/.

AMCOP Cluster and Host Monitoring on Grafana & Prometheus.
Prometheus Server
Once the Kube Promethues Stack PODS and Services are started, Launch Promethues Page and verify Targets are showing as online
http://<ip-address>:30090/
To Launch Prometheus Page

Navigate to Status-->Targets page to check the Target Status

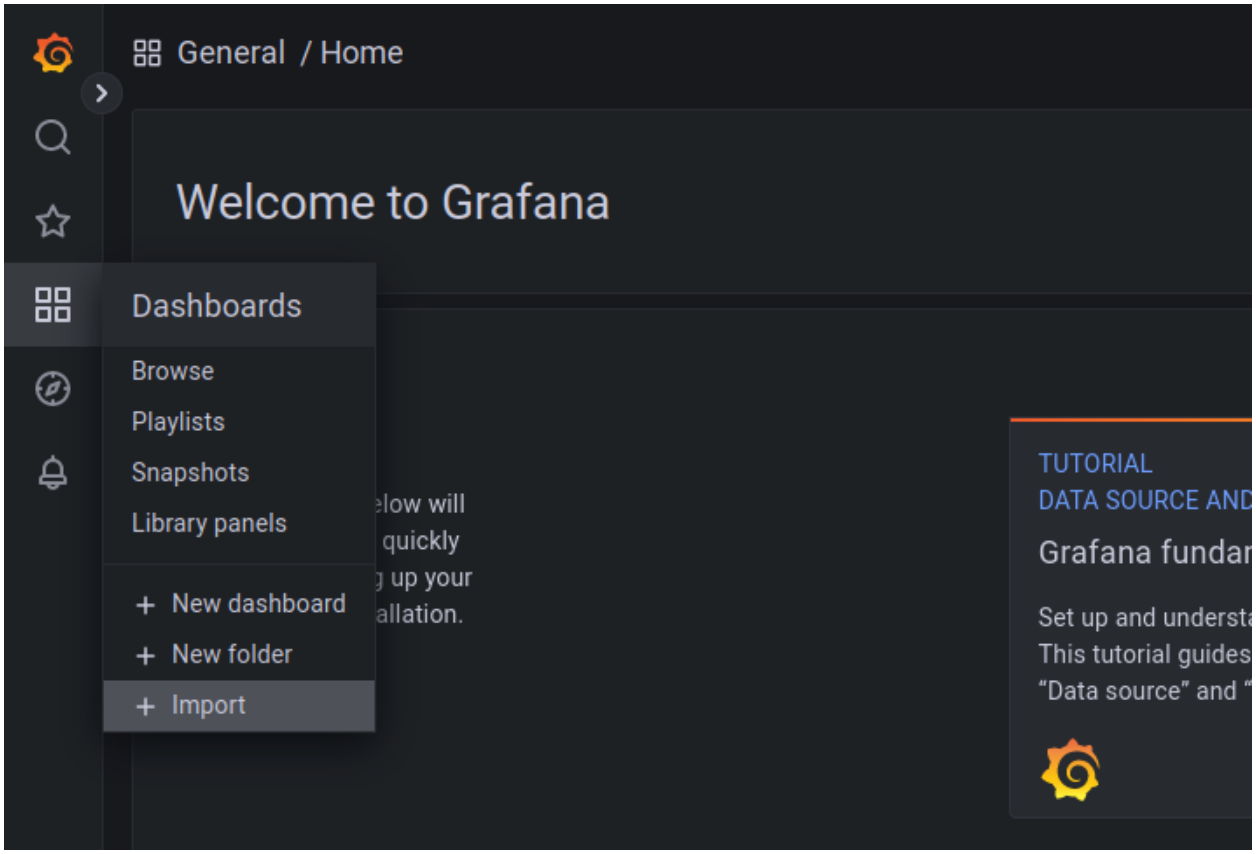


Verify amcop-host-monitor & Federate Targte status is up

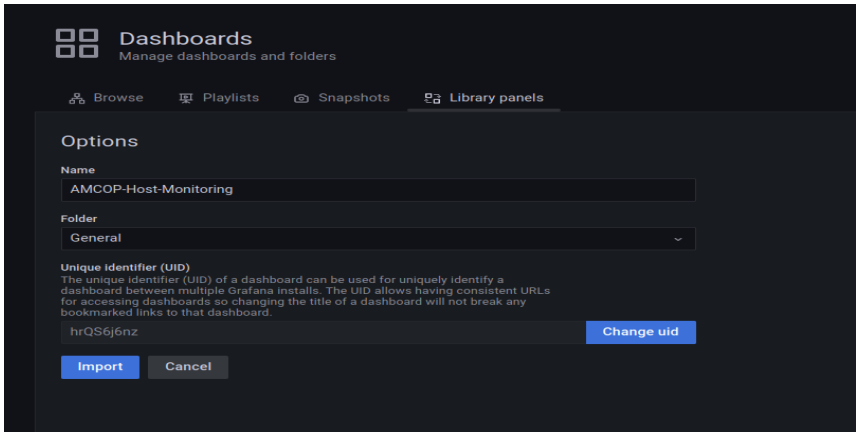


If the Targets are showing up amcop metrics data is available on the Promethues-Stack Server. Can view the metrics dashboard in Grafana.

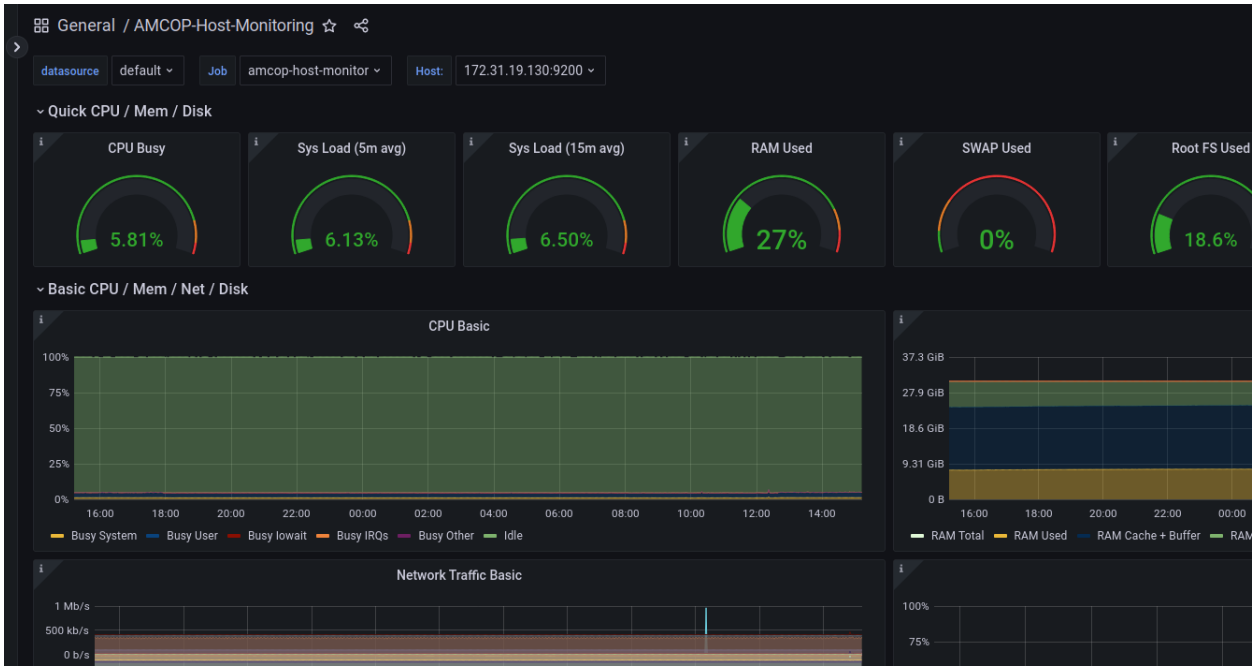
Grafana Dashboard:
To Launch Grafana use the below URL updated with IP address
http://<ip-address>:30080/
Credentials to access Grafana portal
username: admin Password: prom-operator
Navigate to Dashboards and click on Import to import customised dashboards for AMCOP



Upload the the JSON file for host monitoring and amcop-namespace monitor provided in the attachment



Dashboards will be available and will be displayed as below,



Similarly AMCOP Name space monitor dashboard(amcop-ns-monitor.json) can be imported to view the AMCOP namespace dashboard.
Note:

- Grafana custom dashboards can be created and modified for the required metrics/Panel
- Grafana alerting can be created for any graph dashboards and can be integrated with Slack/Email.

Service Management & Orchestrator (SMO)

SMO is one of the AMCOP components that acts as an O1 controller and manages various RAN elements (RU/DU/CU) through the O1 interface and supports FCAPS operations. Please refer to the **AMCOP SMO User Guide** for complete details.

Configure ELK for debugging

You can deploy the ELK stack in the separate k8s clusters and configure it to collect/filter/collate logs from the pods that are deployed in the target cluster. In this section you are going to deploy and set up ELK on a separate k8s cluster and set up a sample index_pattern and log filter on the Kibana dashboard. Also this document will describe how to create alerts in case of any errors in the logs and integrate it with available messaging options like slack, Email, MS Teams etc.

The ELK Stack is a collection of three open-source products — Elasticsearch, Logstash, and Kibana. ELK stack provides centralized logging in order to identify problems with servers or applications. It allows you to search all the logs in a single place. It also helps to find issues in multiple servers by connecting logs during a specific time frame.

- E stands for ElasticSearch: used for storing logs
- L stands for LogStash : used for both shipping as well as processing and storing logs
- K stands for Kibana: is a visualization tool (a web interface) which is hosted through Nginx or Apache
- Filebeat runs on a target cluster (Daemonset, that ships the logs to logstash)

Deploying an Elasticsearch Cluster with Helm

Prerequisite

You need to have a separate K8s cluster up and running (It can either be a single node AIO cluster or multi node) to deploy the ELK stack.

Below are the details of the software versions used:

Below are details of the OS and software used:

OS used: Ubuntu 20.04
Docker: 20.10.17
Kubernetes: 1.23.7
Helm: 3.9.0
ELK Stack: 7.17.3

Step To Deploy ELK Cluster

You can use the helm repository to install the ELK stack.

Add the helm repo:

```
helm repo add elastic https://Helm.elastic.co
"elastic" has been added to your repositories
```

Create local-storage class
Run the below command to create a local-storage class:

```
kubectl apply -f
https://raw.githubusercontent.com/rancher/local-path-provisioner/v0.0.22/deploy/local-path-storage.yaml
```

Reference: GitHub - rancher/local-path-provisioner: Dynamically provisioning persistent local storage with Kubernetes

Create a directory called elasticsearch and inside it create a values.yaml file. This file will be used to overwrite the required helm chart values.

```
mkdir elasticsearch
touch elasticsearch/values.yaml
```

Open the values.yaml file using any editor of your choice. (i.e vi/vim etc) and paste the following contents and save the file.

```
---
# Permit co-located instances for solitary minikube virtual machines.
antiAffinity: "soft"

# Shrink default JVM heap.
esJavaOpts: "-Xmx1g -Xms1g"

# Allocate smaller chunks of memory per pod.
resources:
```

```
  requests:
    cpu: "1000m"
    memory: "2Gi"
  limits:
    cpu: "1000m"
    memory: "2Gi"

# Request smaller persistent volumes.
volumeClaimTemplate:
  accessModes: [ "ReadWriteOnce" ]
  storageClassName: "local-path"
  resources:
    requests:
      storage: 10Gi

# Updating service type from clusterIP to NodePort
service:
       type: NodePort

# This is required to enable alerting module in ES
#esConfig:
# elasticsearch.yml: |
#   xpack.security.enabled: true
#   xpack.security.transport.ssl.enabled: true
#   xpack.security.http.ssl.enabled: true
```

Now, Install the elasticsearch using this configuration i.e values.yaml

```
helm install elasticsearch elastic/elasticsearch -f elasticsearch/values.yaml
```

Once you run the above command, you will see the following output:

```
NAME: elasticsearch
LAST DEPLOYED: Mon Jun 13 18:06:00 2022
```

```
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Watch all cluster members come up.
  $ kubectl get pods --namespace=default -l app=elasticsearch-master -w2. Test cluster
health using Helm test.
  $ helm --namespace=default test elasticsearch
```

Run the below command to see the elasticsearch k8s pods deployed.

```
kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
elasticsearch-master-0   1/1     Running   0          111s
elasticsearch-master-1   1/1     Running   0          111s
elasticsearch-master-2   1/1     Running   0          111s
```

Deploy kibana with Helm

Create a directory called kibana and inside it create a values.yaml file. This file will be used to overwrite the required helm chart values.

```
mkdir kibana
touch kibana/values.yaml
```

Open the values.yaml file using any editor of your choice. (i.e vi/vim etc) and paste the following contents and save the file.

```
mkdir kibana
touch kibana/values.yaml
```

Now, Install the kibana using this configuration i.e values.yaml

```
helm install kibana elastic/kibana -f kibana/values.yaml
```

Once you run the above command, you will see the following output:

```
NAME: kibana
LAST DEPLOYED: Mon Jun 13 18:12:41 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Run the below command to see the kibana k8s pods deployed.

```
kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
elasticsearch-master-0    1/1     Running   0          6m50s
elasticsearch-master-1    1/1     Running   0          6m50s
elasticsearch-master-2    1/1     Running   0          6m50s
kibana-kibana-864bccc7f5-2z8pz   1/1     Running   0          9s
```

Login to Kibana Pod to generate the encryption key which is required to enable the create alert feature in Kibana UI.

```
kubectl exec -it kibana-kibana-864bccc7f5-2z8pz -- /bin/bash
```

Once logged in the pod then run the below command to generate the encryption key

```
bin/kibana-encryption-keys generate
```

From the output of the above command you need to get the following value.

```
xpack.encryptedSavedObjects.encryptionKey: 08bd4823dc4f08096e3e33ecef35b166
```

Now, update the kibana/values.yaml with this encryption key as follows.

```
---
# create a service of type NodePort
service:
  type: NodePort
kibanaConfig:
   kibana.yml: |
     xpack.encryptedSavedObjects.encryptionKey: 08bd4823dc4f08096e3e33ecef35b166
#This is required to enable Alerts and Rules in Kibana. Update the key with actualy key.
```

Once you update the values.yaml, then you need to upgrade the kibana helm deployment using the below command.

```
helm upgrade kibana elastic/kibana -f kibana/values.yaml
```

After this, Kibana will be successfully upgraded with new changes.

Access Kibana UI

Kibana UI can be accessed using the ELK node Primary interface IP followed by the Kibana NodePort number.

Run the following command to get the NodePort details.

```
kubectl get svc
NAME                    TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
elasticsearch-master          NodePort    10.104.190.45   <none>
9200:31217/TCP,9300:30791/TCP   6d17h
elasticsearch-master-headless   ClusterIP   None          <none>
9200/TCP,9300/TCP            6d17h
kibana-kibana                 NodePort    10.109.70.144   <none>       5601:31119/TCP
6d16h
kubernetes                    ClusterIP   10.96.0.1       <none>       443/TCP
10d
```

Here, you can see the NodePort corresponding to kibana service is 31119, so you can access the kibana UI as follows:
Note:: Make sure Prometheus Service NodePort is opened in your security groups inbound rule

Reference: AWS-Documentation for Security group modification

```
http://<primary network interface IP>:NodePort number
e.g
http://192.168.56.4:31119
```

Deploy Logstash with Helm

Create values.yaml to deploy logstash.

```
mkdir logstash
touch logstash/values.yaml
```

Open the values.yaml file using any editor of your choice. (i.e vi/vim etc) and paste the following contents and save the file.

```
---
resources:
  limits:
    cpu: 1000m
    memory: 2Gi
  requests:
    cpu: 500m
    memory: 1Gi
service:
  ports:
    -
      name: beats
      port: 5044
      protocol: TCP
      targetPort: 5044
    -
      name: http
      port: 8080
      protocol: TCP
```

```
    targetPort: 8080
  type: NodePort
logstashConfig:
  logstash.yml: |
    http.host: 0.0.0.0
    xpack.monitoring.elasticsearch.hosts: [ "http://elasticsearch-master:9200" ]
logstashPipeline:
  logstash.conf: |
    input {
      beats {
        port => 5044
      }
    }
    filter {
    }
    output {
      if "console-logs" in [tags] {
      elasticsearch {
        index => "logstash-console-logs-%{+YYYY.MM.dd}"
        hosts => [ "elasticsearch-master:9200" ]
      }
      }
      else if "syslog" in [tags]{
      elasticsearch {
        index => "logstash-syslog-%{+YYYY.MM.dd}"
        hosts => [ "elasticsearch-master:9200" ]
      }
      }
      else {
       elasticsearch {
        index => "logstash-app-logs-%{+YYYY.MM.dd}"
        hosts => [ "elasticsearch-master:9200" ]
      }
      }
      }
```

Once you edited the values.yaml file, now it is the time to run the below command to deploy logstash.

```
helm install logstash elastic/logstash -f logstash/values.yaml
```

Run the below command to see the k8s pods deployed.

```
kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
elasticsearch-master-0   1/1     Running   0          17m
elasticsearch-master-1   1/1     Running   0          17m
elasticsearch-master-2   1/1     Running   0          17m
kibana-kibana-864bccc7f5-2z8pz   1/1     Running   0          11m
logstash-logstash-0      1/1     Running   0          2m39s
```

List all the ELK charts deployed

```
helm list
NAME          NAMESPACE REVISION      UPDATED                                   STATUS
        CHART                   APP VERSION
elasticsearch         default         1       2022-06-13 18:06:00.422354843 +0000
UTC    deployed      elasticsearch-7.17.3          7.17.3
kibana          default         1       2022-06-13 18:12:41.065278477 +0000 UTC
deployed      kibana-7.17.3         7.17.3
logstash        default         1       2022-06-13 18:21:03.444973484 +0000 UTC
deployed      logstash-7.17.3       7.17.3
```

AMCOP ELK Integration

In order to collect all the AMCOP pod's logs from AMCOP target cluster, a log collector needs to be installed on AMCOP cluster. Here you are going to use a log collector called filebeat to collect all the logs.

Running Filebeat as a Daemonset

You will deploy a filebeat logging agent as a k8s daemonset so that it deploy a filebeat pod on all the nodes in AMCOP cluster. All the k8s pod's STDOUT and STDERR logs available at /var/log/containers/* and system logs available at /var/log/syslog etc can be collected by this logging agent.

Deploy Filebeat as Daemonset

Let's deploy filebeat as a daemonset to collect all the k8s pod's stdout, stderr and syslogs and send it to the logstash deployed on another node(ELK node)
Steps To deploy Filebeat In AMCOP Cluster

SSH to the AMCOP Deployment cluster or Target cluster

Make sure you have helm installed
Add the helm repo:

```
helm repo add elastic https://Helm.elastic.co
"elastic" has been added to your repositories
```

Create a directory called filebeat and inside it create a values.yaml file. This file will be used to overwrite the required helm chart values.

```
mkdir filebeat
touch filebeat/values.yaml
```

Open the values.yaml file using any editor of your choice. (i.e vi/vim etc) and paste the following contents and save the file.

```
---
filebeatConfig:
  filebeat.yml: |
    filebeat.inputs:
    - type: container
      tags: ["console-logs"]
      paths:
        - /var/log/containers/*.log
      exclude_files:
        - '^/var/log/containers/filebeat.*'
```

```
    - '^/var/log/containers/*filebeat*.log'
  - type: log
    paths:
      - /var/log/syslog
    tags: ["syslog"]
  output.logstash:
    hosts: ["192.168.56.4:30606"]
```

Note:
hosts is the NodePort - primary network interface IP of elasticsearch cluster and "30606" is the logstash external port.
You can get the IP details and external port number from elasticsearch cluster.
Use the ifconfig command to get the primary network interface IP and the kubectl get svc command to get the external port number for elasticsearch.

Below is the example for the same.

```
kubectl get svc
NAME                     TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
elasticsearch-master          NodePort    10.108.103.189   <none>
9200:32025/TCP,9300:31405/TCP   6d12h
elasticsearch-master-headless   ClusterIP   None          <none>
9200/TCP,9300/TCP            6d12h
kibana-kibana                 NodePort    10.109.70.144    <none>       5601:31119/TCP
38d
kubernetes                    ClusterIP   10.96.0.1        <none>       443/TCP
42d
logstash-logstash             NodePort    10.105.107.87    <none>
5044:30606/TCP,8080:30882/TCP   15d
logstash-logstash-headless    ClusterIP   None          <none>       9600/TCP
15d
```

Once you edited the values.yaml file, now it is the time to run the below command to deploy filebeat.

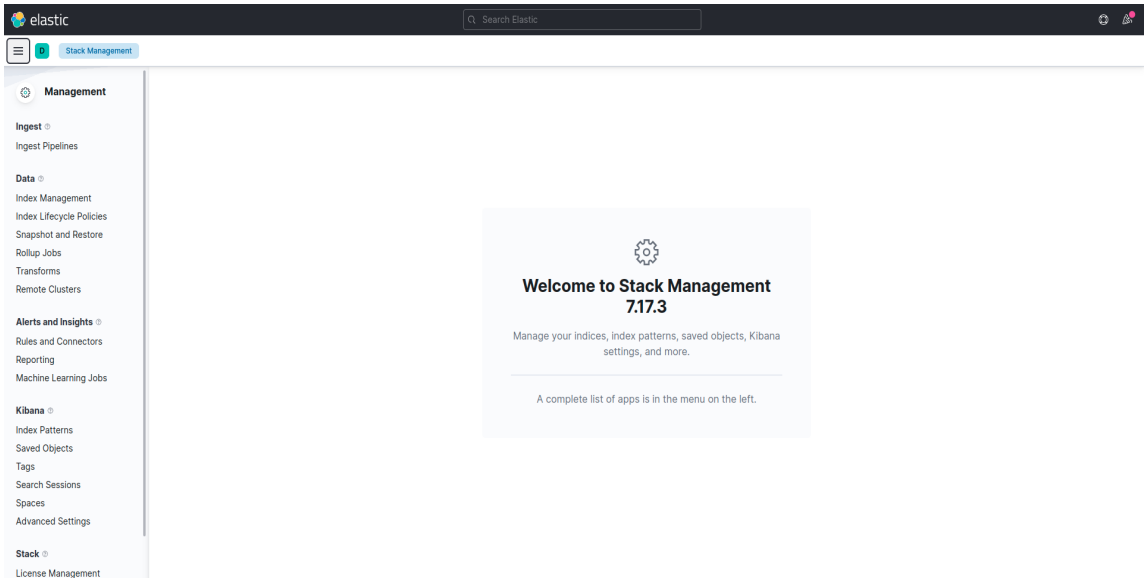
```
helm install filebeat elastic/filebeat -f filebeat/values.yaml
```

It will deploy the filebeat on the target AMCOP cluster successfully.

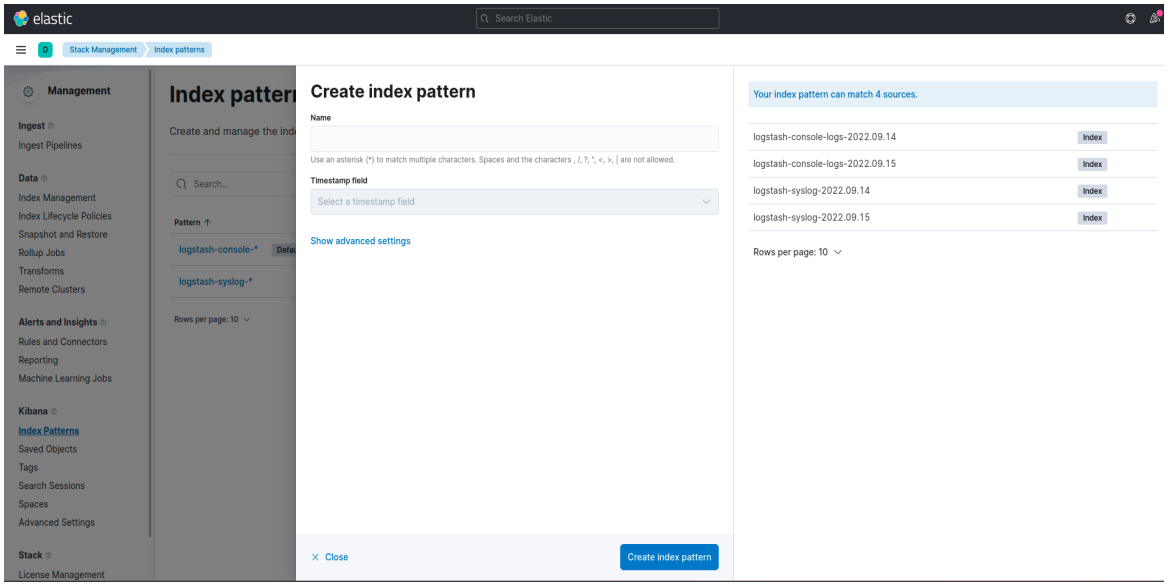
Steps to create log index pattern, Analyse logs and create Alerts in Kibana

In the kibana UI follow the below steps to add the index and to create the rule

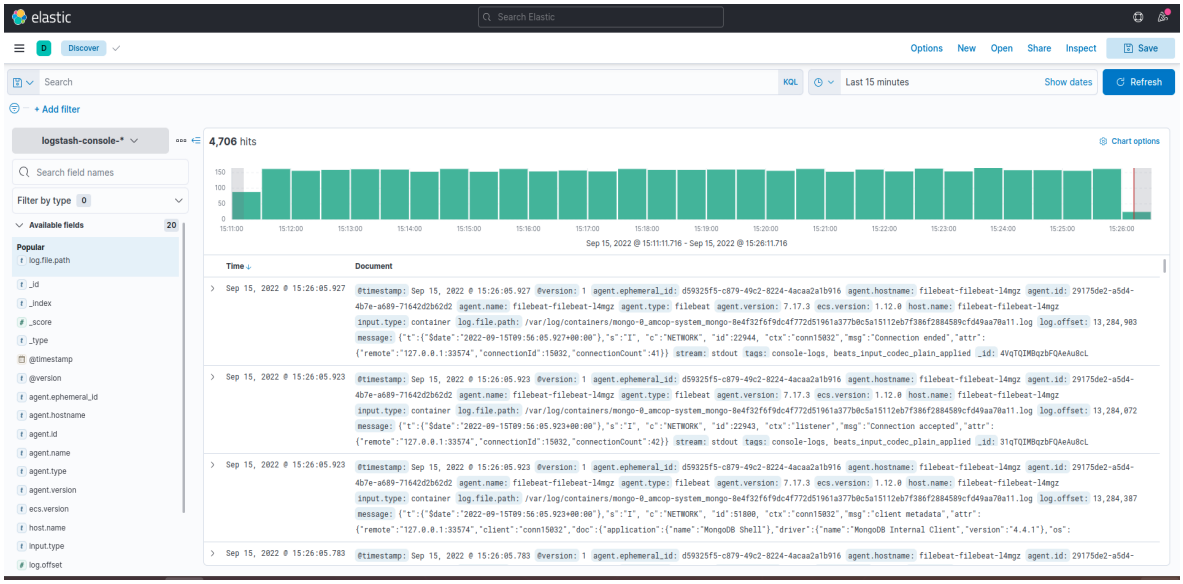
1. Go to Stack management under management



2. Click on index pattern under kibana and click on create index
   Add below indexes with timestamp
   logstash-console-*
   logstash-syslog-*

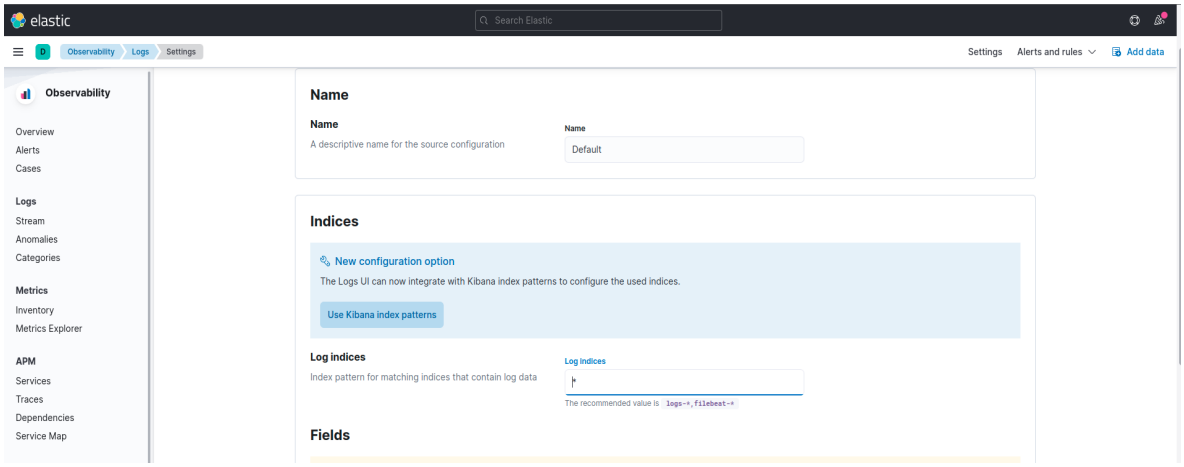


3. Click on discovery you should be able to see all the logs



Create Alerts

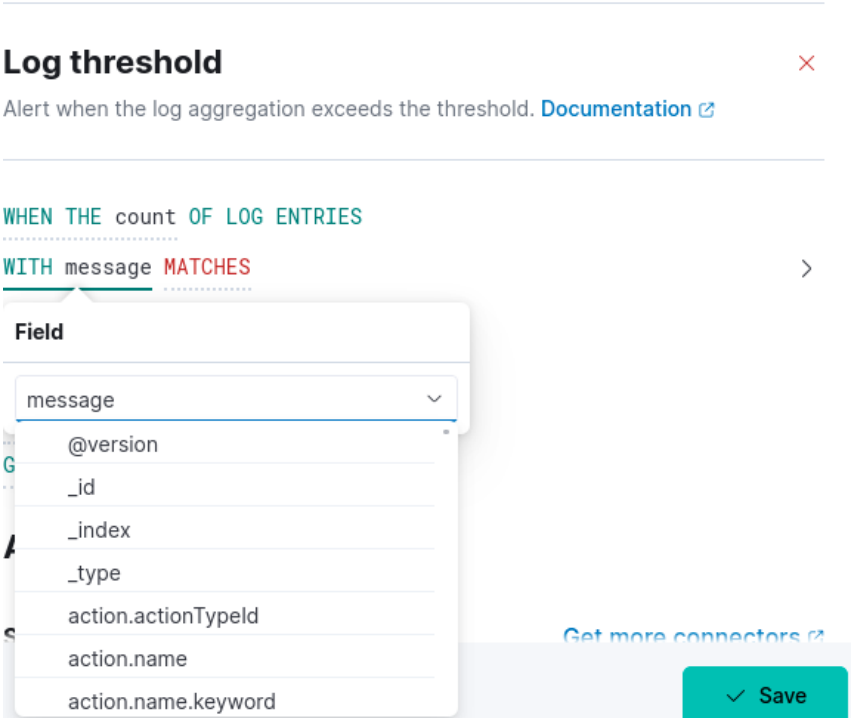
Go to Logs under Observability
Click on Settings and update the log Indices value to * and Apply



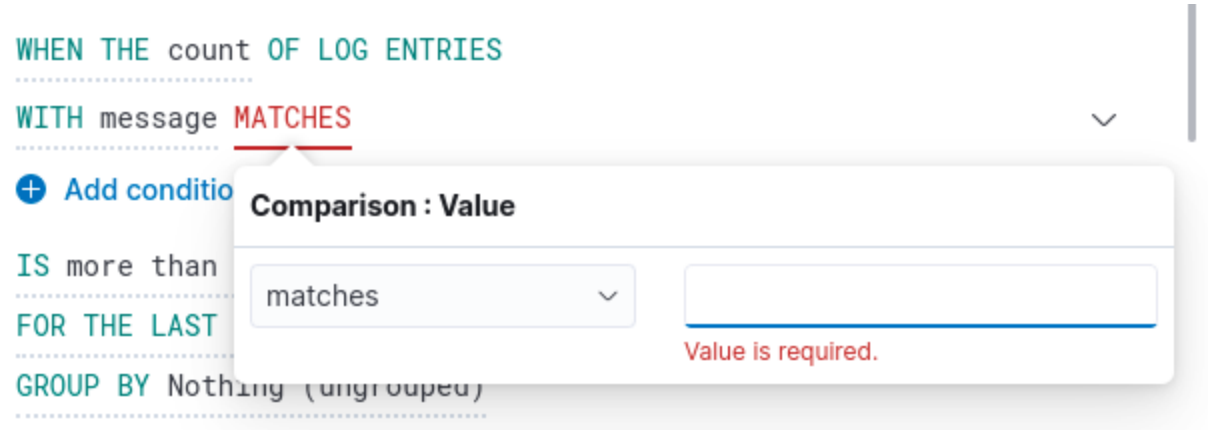
Go to Stack management under management

Click on Rules and Connectors inside Alert and insights

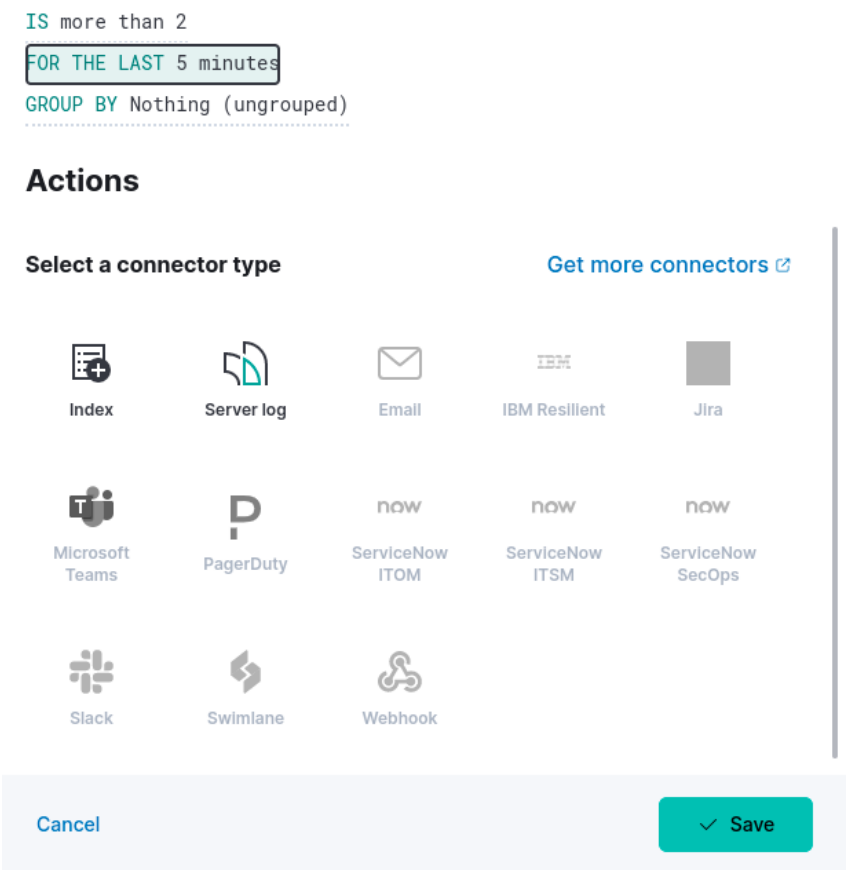
Enter required fields, choose Log threshold and select the required field from the drop down below



For the MATCHES Provide the required value



Choose and configure the required connector to get the notification from the available
options and save Example: Slack, Email etc .



AMCOP Workflow Engine (AWE)

AMCOP supports a workflow engine that can be used to implement custom workflows. The workflow engine is based on Camunda, and it can execute BPMN workflows. AMCOP platform already supports several BPMN workflows that can be readily used for various use cases. But it also supports onboarding your custom workflows.

The steps to develop workflows are beyond the scope of this document. You can refer to any publicly available documentation on BPMN/Camunda workflow, such as: https://docs.camunda.org/manual/7.15/

This following subsection shows how the workflows can be onboarded on AMCOP. The workflow logic is packaged as a WAR file, which is used in the following subsection.

Onboard BPMN workflow to Camunda Engine

1. Execute below command to get camunda pod details.

```
kubectl get pods --all-namespaces | grep camunda

    NAMESPACE           NAME                                READY   STATUS    RESTARTS
    AGE
    emco          amcop-camunda-59cb57565f-nxb9s
    2/2    Running   0        66d
```

2. Copy the war file (which contains the BPMN workflow logic) to Camunda pod.

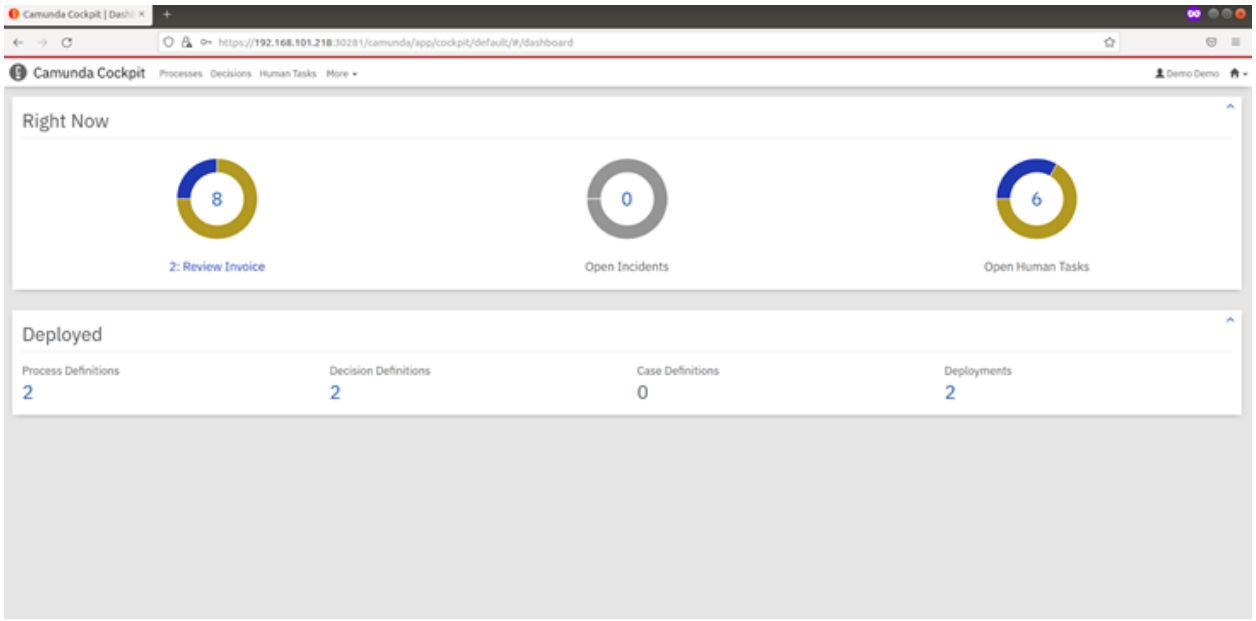
```
kubectl -n emco cp <WAR_FILE_PATH>
amcop-camunda-59cb57565f-nxb9s:/camunda/bpmnapps/.
```

3. Verify the deployment with Camunda cockpit.

```
# Run the below command to get the port name on which camunda service
# is exposed.

kubectl get svc -n emco -o wide

    NAME                TYPE      CLUSTER-IP     EXTERNAL-IP   PORT(S)
    AGE   SELECTOR
```

```
camunda                 NodePort    10.244.11.150   <none>        8443:30281/TCP
66d   app=camunda,release=amcop
```

Open portal URL from FireFox browser (on your laptop or local server) and type the AMCOP deployment VM IP and port number on which the service is exposed (eg., 30281). Make sure the FireFox browser settings are done and a tunnel is created and is running.



For example, the URL will look like: 192.168.101.218:30281

Enter credentials : username/password → souser/mypassword

Camunda dashboard : username/password -> demo/demo

Execute the Workflow

```
# Run the below command to get the port name on which camunda service
# is exposed.

kubectl get svc -n emco | grep camunda

        NAME      TYPE      CLUSTER-IP    EXTERNAL-IP    PORT(S)        AGE
        camunda   NodePort  10.244.2.237  <none>         8443:31082/TCP  68d
```

Set the following environment variables:

- CAMUNDA_IP= 10.244.2.237
- CAMUNDA_PORT= 8443
- WORKFLOW_NAME= PROCESS_BPMN_EXAMPLE
- export CAMUNDA_PAYLOAD_JSON_TEMPALTE_FILE="camunda-sample-payload.json"
  cat <<EOF >$CAMUNDA_PAYLOAD_JSON_TEMPALTE_FILE
  {
  "variables": {
  "payload": {
  "type": "String",
  "value": "{\n\t"name": "sample test"\n}\n\n",
  "valueInfo": {},
  "withVariablesInReturn": true
  }
  }
  }
  EOF

Note: WORKFLOW_NAME - workflow name which is deployed in the previous step as part of war deployment.

1. Run the following command to execute the workflow.

```
# Execute cURL command to start the workflow with request payload

curl -v -X POST
http://${CAMUNDA_IP}:${CAMUNDA-PORT}/engine-rest/process-definition/key/${WORKFLOW_NAME
}/start --header 'Content-Type: application/json'  -d @${CAMUNDA_PAYLOAD_JSON_TEMPALTE_FILE}


# Execute cURL command to start the workflow without request payload

curl -v -X POST
http://${CAMUNDA_IP}:${CAMUNDA-PORT}/engine-rest/process-definition/key/${WORKFLOW_NAME
}/start --header 'Content-Type: application/json'
```

Appendix A - Free5GC & UERANSIM Installation

This section explains the steps and commands required to deploy Free5gc and run UERANSIM to test some of the use cases. Below is a brief description of AMCOP, Free5gc and UERANSIM simulator

Free5GC: The free5GC is an open-source project for 5th generation (5G) mobile core networks. This project implements the 5G core network (5GC) defined in 3GPP Release 15 (R15) and beyond. It is being used to showcase Free5GC test cases using any 3GPP compliant simulator.

UERANMSIM: UERANSIMis a 5G SA gNB/UE (Release 16) simulator for testing the 5G System. The project is aimed to understand 5GC more efficiently than just reading 3GPP standard documents.

Setting up Free5GC and UERANSIM simulator environment

This section provides steps to configure the environment to orchestrate Free5GC using AMCOP. The following should be done on the deployment host.

- Run create_qem_vm.sh from the deployment host as follows. Please make sure the correct version of Ubuntu is selected as per the below instructions.

  ```
  ./create_qem_vm.sh
  ```

  Choose option 3 to create ubuntu 20.04
  Provide required inputs such as name, disk space, vCPU and Memory as follows:

CPUs	16
Memory	32GB
Storage	50GB

- Find the VM's IP address:

  ```
  sudo virsh list --all
  sudo virsh domifaddr <VM_NAME>
  ```

- Make sure you are able to log in to the VM via ssh. The default user name is ubuntu. You should be able to login to the VM as follows

  ```
  ssh  ubuntu@<VM_IP>
  ```

- Log in to the VM, and perform the following.
  - Check the kernel version. It should be >= 5.4. This is important for UERANSIM simulator to run successfully.

```
uname -r
5.4.0-86-generic
```

  - Download the package https://drive.google.com/file/d/1VPi6g2iDhhWTV_d1OAi7lReZDL9yYVgO/view?usp=sharing
  - The contents of this package are,

```
ls free5gc-306-helm
free5gc.tgz                multiclusterkind.tgz      ueransim.tgz
```

  - Install gtp kernel driver,

```
git clone -b v0.1.0 https://github.com/PrinzOwO/gtp5g.git
cd gtp5g
make
sudo make install
```

  - Clone mutlus in home directory,
    - git clone https://github.com/intel/multus-cni.git
  - Add your user to the docker group
    - sudo usermod -aG docker $USER
    - Logout and login
  - Create target clusters for the free5gc deployment,
    - Untar the multiclusterkind.tgz
    - Go to multiclusterkind directory and
    - Execute the script,

```
./kind_create_cluster.sh
Enter Host IP:
192.168.101.216
host ip is provided
Enter AMCOP IP:

AMCOP ip is not provided, skipping cluster register with amcop
Do you wish to install CNI for Free5GC? y
Do you wish to install ISTIO_COMPLETE_INSTALL? n
Do you wish to install ISTIO_CERT_ONLY? n
Do you wish to install ISTIO_ONLY? n
Do you wish to install ISTIO_INJECT_SIDECAR? n
Do you wish to install METALLB_FOR_ISTIO? n
```

    - The kubeconfig files of the kind clusters are present in ~/.kube directory. Use cluster1 config file for onboarding cluster to AMCOP.
    - For the orchestration use the two helm charts free5gc.tgz and ueransim.tgz

Free5gc Orchestration

Create two services, one for the free5gc and another for the ueransim. Use the default profile.tgz. Instantiate the services in the target kind cluster.

Test PDU session

- kubectl exec -it < ue container > – bash

```
# Run this inside the container
ip address
...
5: uesimtun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500
qdisc fq_codel state UNKNOWN group default qlen 500
   link/none
   inet 10.1.0.1/32 scope global uesimtun0
     valid_lft forever preferred_lft forever

ping -I uesimtun0 www.google.com
traceroute -i uesimtun0 www.google.com
curl --interface uesimtun0 www.google.com
```

Troubleshooting tips

This section describes the possible root cause and resolution for some of the errors/exceptions seen while running the UERANSIM simulator.

1. Issue: Connection timeout issue. Logs will be something similar given below.
   a. [ueransim]2021/01/23 13:59:43.617257 example.go:65: failed to dial: connection timed out

   Possible root cause: The Free5GC services may be down or not installed.
   Resolution: Check if AMF is running and make sure all the Free5GC services are up and running.

2. Issue: Invalid memory address. Logs will be something similar given below.
   a. panic: runtime error: invalid memory address or nil pointer dereference
   b. In the AUSF logs, if you see the below error message

i. 2021-01-23T14:27:10Z [INFO][AUSF][UeAuthPost] 403 Forbidden (/go/src/free5gc/src/ausf/producer/ue_authentication.go:123 free5gc/src/ausf/producer.UeAuthPostRequestProcedure) in the AUSF logs

Possible root cause: example.json file is not updated with the correct imeisv, Msin and GTPuAddr.
Resolution: Log in to the webui and check if UE is registered. If UE is registered, update the example.json file with the correct imeisv, Msin and GTPuAddr values.

3. Issue: Invalid memory address. Logs will be something similar given below.
   a. panic: runtime error: invalid memory address or nil pointer dereference

Possible root cause: example.json file is not updated with the correct imeisv, Msin and GTPuAddr.

Resolution: Log in to the webui and copy imeisv value. Follow the AMCOP User Guide to populate the corresponding fields in example.json.

Appendix B- Akraino PCEI Blueprint support

Akraino, a Linux Foundation project initiated by AT&T and Intel, intends to develop a fully integrated edge infrastructure solution, and the project is completely focused towards Edge Computing. This open source software stack provides critical infrastructure to enable high performance, reduce latency, improve availability, lower operational overhead, provide scalability, address security needs, and improve fault management. The Akraino community will address multiple edge use cases and industry, not just Telco Industry. The Akraino community intends to develop solutions and support for carrier, provider, and IoT networks.

One of the blueprints of Akraino is PCEI (Public Cloud Edge Interface), which is fully supported using AMCOP.

Refer to the following Wiki page for the complete documentation of PCEI (Release 4 and Release 5) blueprint:

https://wiki.akraino.org/display/AK/PCEI+Documentation
https://wiki.akraino.org/display/AK/PCEI+Release+5+Documentation
https://wiki.akraino.org/display/AK/PCEI+R5+End-to-end+Validation+Guide

The PCEI blueprint uses EMCO component as the Multi-domain orchestrator, to orchestrate applications across different domains, and CDS component to configure all the components, and program the interconnect (eg., using Equinix APIs).

This functionality can be achieved by using AMCOP, which includes all the necessary components required for implementation of PCEI blueprint (EMCO, CDS). Once AMCOP is installed, you can skip the steps related to EMCO and CDS installation, and directly bring up other PCEI components. You can contact the Aarna support team (support@aarnanetworks.com) for any additional support.

Appendix C - Closed Loop using NWDAF

This appendix shows a specific use case of orchestrating Free5gc using AMCOP, and creating a closed loop, using a pre-standards implementation of NWDAF. This illustrates how a closed loop with 5GC cloud native functions can be built using AMCOP, using Free5gc as an example. .

Installing Target Kubernetes Cluster for orchestrating Free5gc and NWDAF/AF:

Please make sure all the pods including the multus pod are in Running state in the kube-system namespace.

Note: If Multus pods is not Running, please execute the following command from the home directory.

cat ./multus-cni/deployments/multus-daemonset.yml | kubectl apply -f -

```
ubuntu@free5gc-vm:~$ kubectl get pods -n kube-system
NAME                                 READY   STATUS    RESTARTS   AGE
coredns-74ff55c5b-7mpmq              1/1     Running   0          23h
coredns-74ff55c5b-mnblr              1/1     Running   0          23h
etcd-free5gc-vm                      1/1     Running   0          23h
kube-apiserver-free5gc-vm            1/1     Running   0          23h
kube-controller-manager-free5gc-vm   1/1     Running   0          23h
kube-multus-ds-wxhlt                 1/1     Running   0          22h
kube-proxy-frlkv                     1/1     Running   0          23h
kube-scheduler-free5gc-vm            1/1     Running   0          23h
tiller-deploy-7b56c8dfb7-v5lrf       1/1     Running   0          23h
weave-net-tddsf                      2/2     Running   1          23h
```

Orchestrating Free5gc from AMCOP GUI:

Please refer to the section “Orchestration of Free5GC using AMCOP GUI” for orchestrating Free5gc.

Orchestrating NWDAF/AF from AMCOP GUI:

1. Download NWDAF and AF Helm package from Google Drive link here: NWDAF-AF
   Follow the same procedure from AMCOP GUI to create Service and instantiate NWDAF and AF.

   After instantiation, all the pods (Free5gc, NWDAF and AF) will be deployed in the default namespace.

```
ubuntu@free5gc-vm:~$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
analytics-af-6657f98875-xkb7h       1/1     Running   0          19h
cpuprediction-74ff67758b-l7dn7      1/1     Running   0          19h
f5gc-amf-798785fbd-mmmhg            2/2     Running   0          21h
f5gc-ausf-6c8788884c-kzwbh          2/2     Running   0          21h
f5gc-mongodb-0                      1/1     Running   0          21h
f5gc-nrf-5cd4c5bf8b-kw5xq           2/2     Running   0          20h
f5gc-nssf-84c8955b4c-v9wc5          2/2     Running   0          21h
f5gc-pcf-f764bc8c8-dzwkw            2/2     Running   0          21h
f5gc-smf-657688dd8c-lnrcl           2/2     Running   0          21h
f5gc-udm-655c4f568-grtld            2/2     Running   0          21h
f5gc-udr-6fdd649c6f-9tl6p           2/2     Running   0          21h
f5gc-upf-5dc67d5f98-t886f           2/2     Running   0          21h
f5gc-webui-78769967-hcnb6           2/2     Running   11         21h
nwdaf-646db467cd-zrqmp              1/1     Running   0          19h
```

2. Verify logs of AF pod and ensure the following events are streaming.

   kubectl logs analytics-af-6657f98875-xkb7h

   2021/12/09 10:07:11 Starting a new run of runAnalytics
   2021/12/09 10:07:11 Preparing and sending discover NF request to NRF endpoint http://f5gc-nrf:29510
   2021/12/09 10:07:11 Response : &{200 OK 200 HTTP/1.1 1 1 map[Content-Length:[41] Content-Type:[application/json] Date:[Thu, 09 Dec 2021 10:07:11 GMT]] {{"validityPeriod":100,"nfInstances":null}} 41 [] false false map[] 0xc0000cb300 <nil>}

CBA onboarding

CDS component installed as part AMCOP. This section explains how to onboard the scale-out CBA. Please refer to the section on Life cycle management for more information on CBA design and onboarding.

Download the CBA from post-operation-cba.zip , extract and copy *post-operation* folder to ~/aarna-stream/cds-blueprints/

```
# Bootstrap cds, this step is a one time activity.
cd ~/aarna-stream/cds-blueprints/k8s-utility-scripts/
```

```
bash -x ./bootstrap-cds.sh

# Load the data dictionary to the Database
bash -x ./dd-microk8s.sh ~/aarna-stream/cds-blueprints/post-operation/Scripts/dd.json

# Create cba zip folder
cd ~/aarna-stream/cds-blueprints/post-operation
zip -r  CBA_FOLDER.zip *

# Enrich the CBA
cd ~/aarna-stream/cds-blueprints/k8s-utility-scripts/
bash -x ./enrich-and-download-cds-blueprint.sh
~/aarna-stream/cds-blueprints/post-operation/CBA_FOLDER.zip

# Save the Enrichment
bash -x ./save-enriched-blueprint.sh /tmp/CBA/ENRICHED-CBA.zip
bash -x ./get-cds-blueprint-models.sh
```