Aarna Networks Multi-Cluster Orchestration and Automation Platform (AMCOP)

User Guide

Product Version: 2.2.10052021

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:

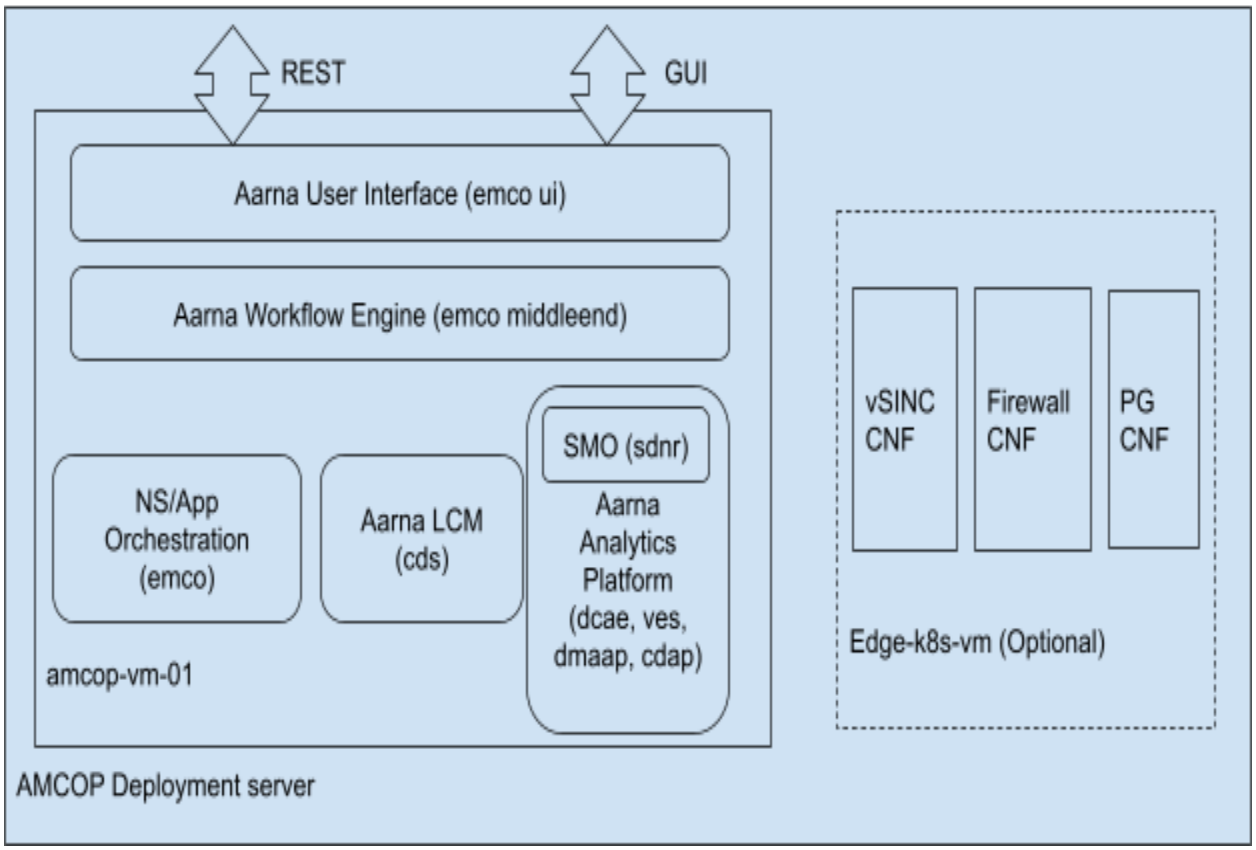


Figure 1: Single Server deployment with KuD cluster

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

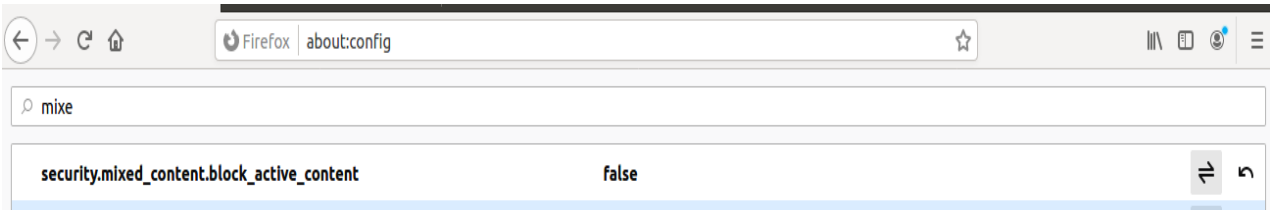
Configuring Desktop/Laptop to access AMCOP portal

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

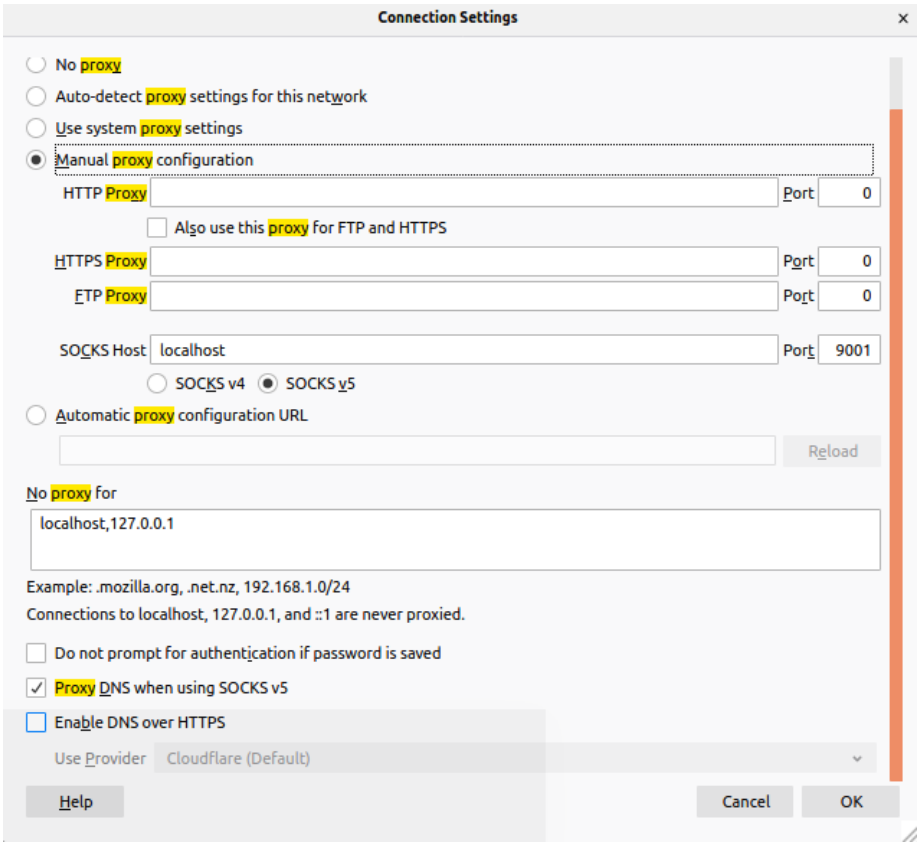
Browser Settings

- 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 Portal

- 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
Clm     NodePort   10.3.241.49     <none>       9061:31856/TCP    28m    app=clm
Emcoui  NodePort   10.3.240.124    <none>       9080:30480/TCP    28m    app=emcoui
```

- 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.,

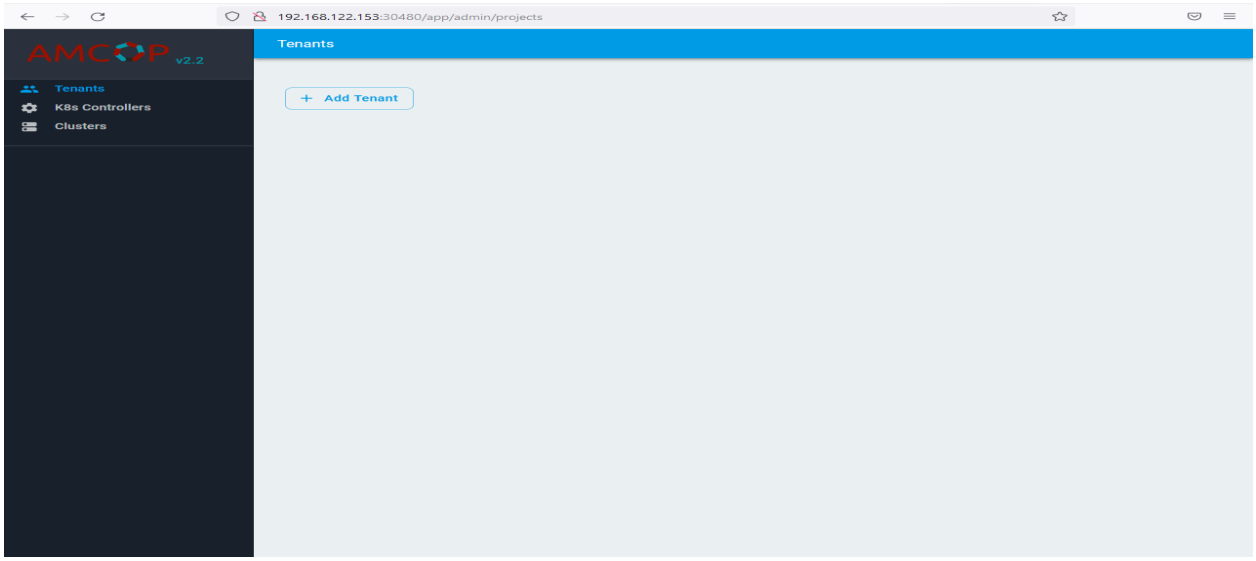
30480). Make sure the Firefox browser settings are done, and a tunnel is created and is running.

For example, the URL will look like: 192.168.122.74:30480

The AMCOP 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://git.onap.org/multicloud/k8s/

# 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.

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:

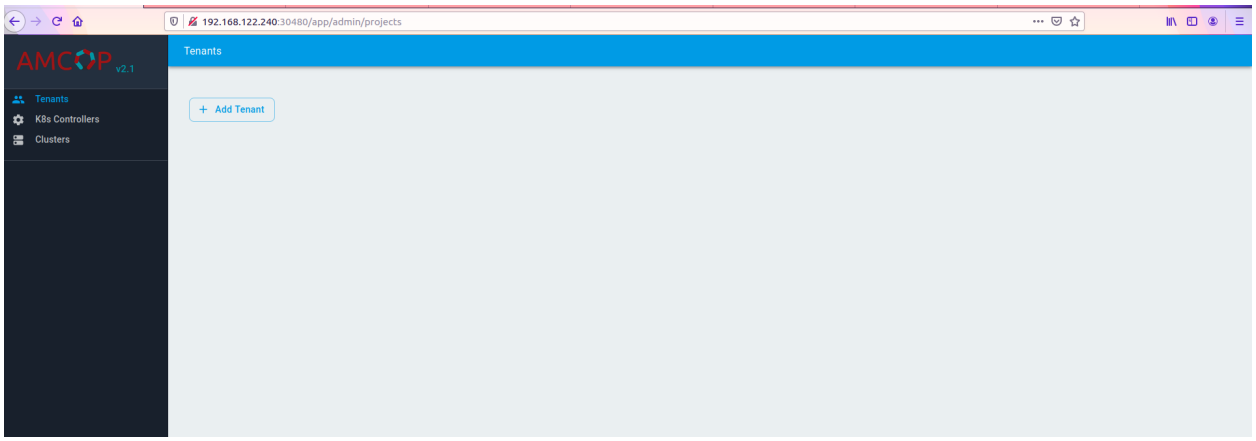
http://<amcop-master-vm-ip>:30480

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

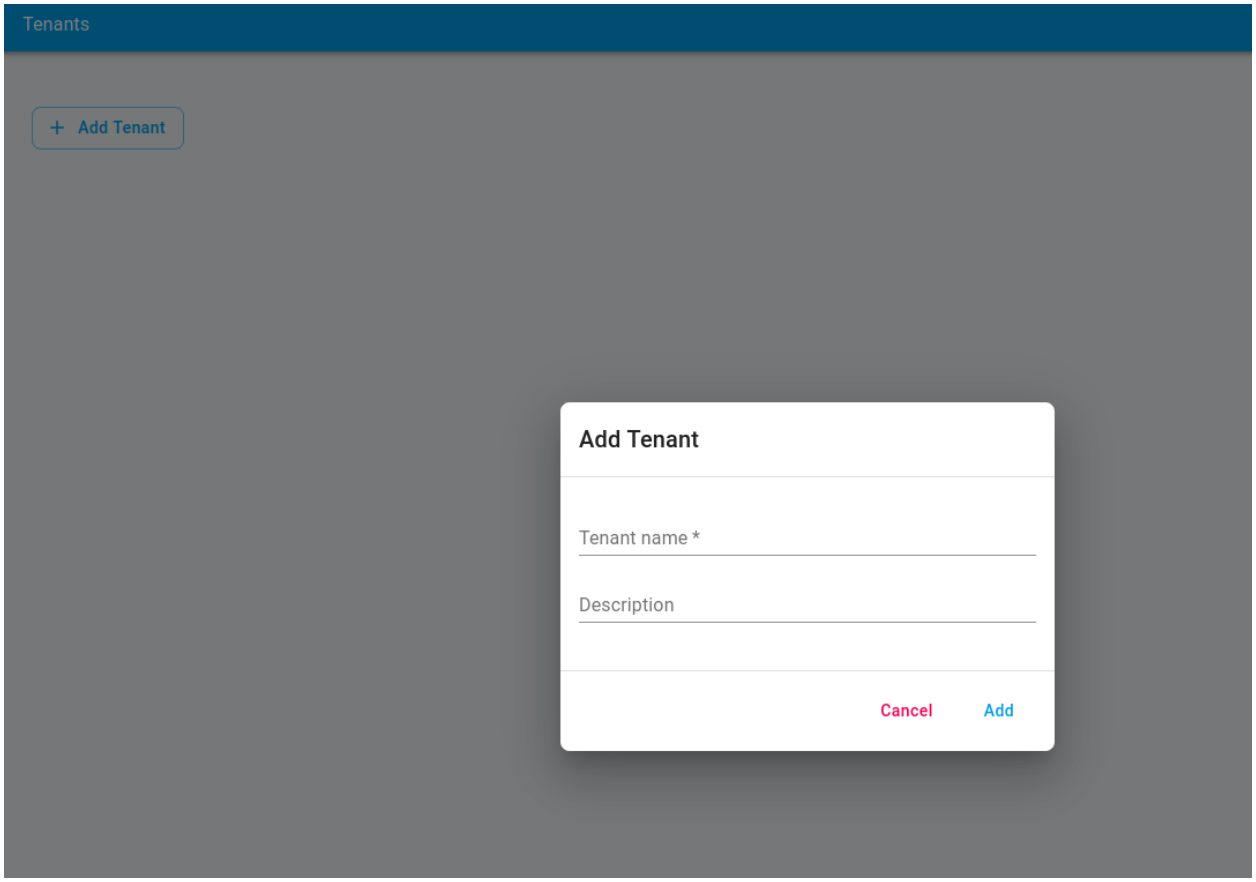
The user interface of AMCOP GUI is divided into two parts. One is for admin related functionalities like adding projects, onboarding clusters, adding controllers etc and the other for the service designer related functionalities like Creating service, instantiating service etc.

Admin User

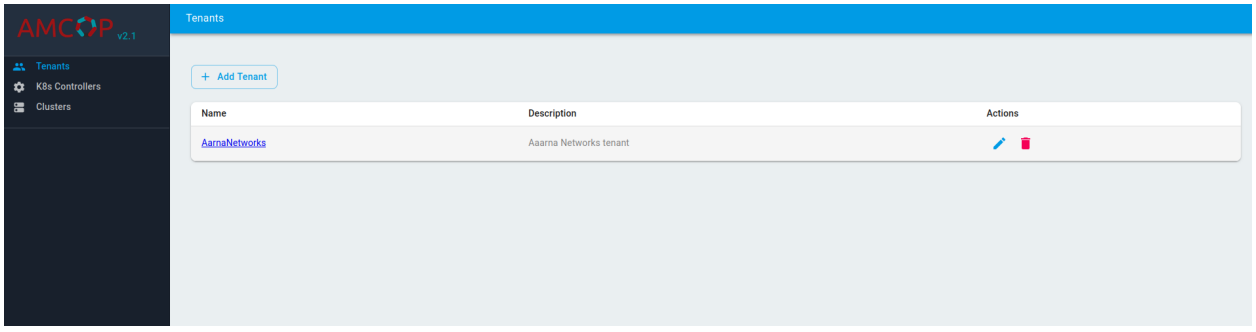
Once the GUI is launched, the tenants page will be displayed.



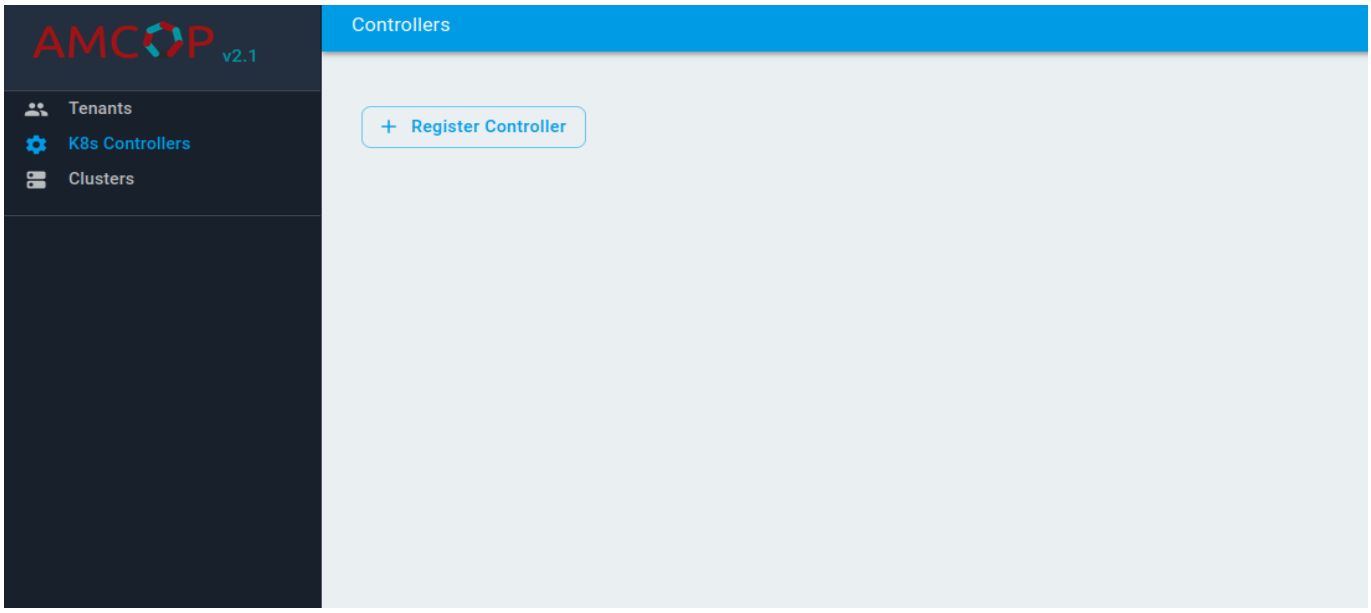
1. Add a Tenant
    a. To start with, add a tenant by clicking on the Add Tenant button and filing the required fields and click Add.



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.

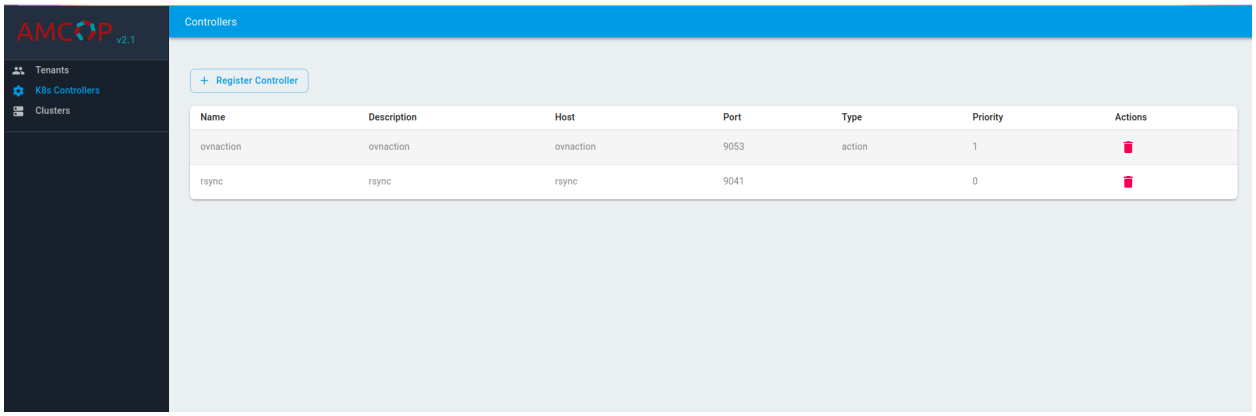
For the vFW service, you need the following two controllers. Hence, these two controllers are bundled with AMCOP deployment, with the below configuration.

1. rsync:

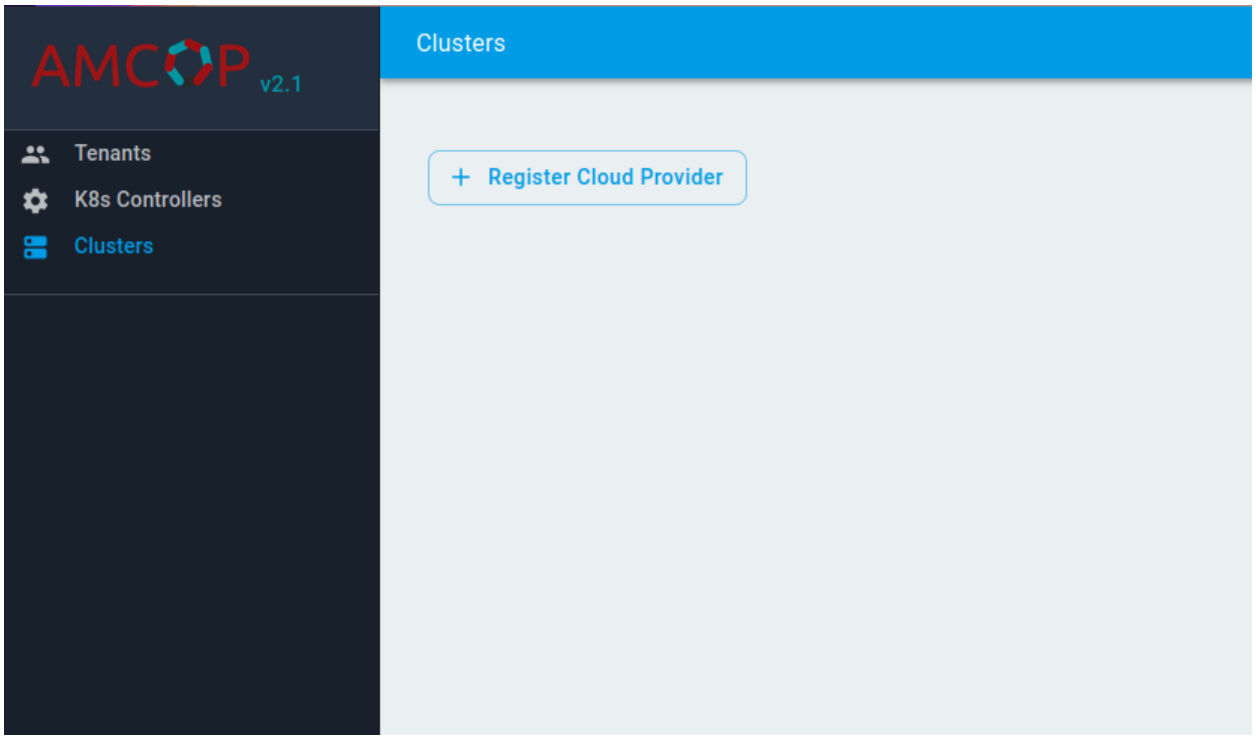
```
Name: rsync
host : rsync
port: 9041
type: < leave blank, not required.>
priority: < leave blank, not required>
```

2. ovnaction:

```
Name: ovnaction
host: ovnaction
port: 9053
type: action
priority: 1
```



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.



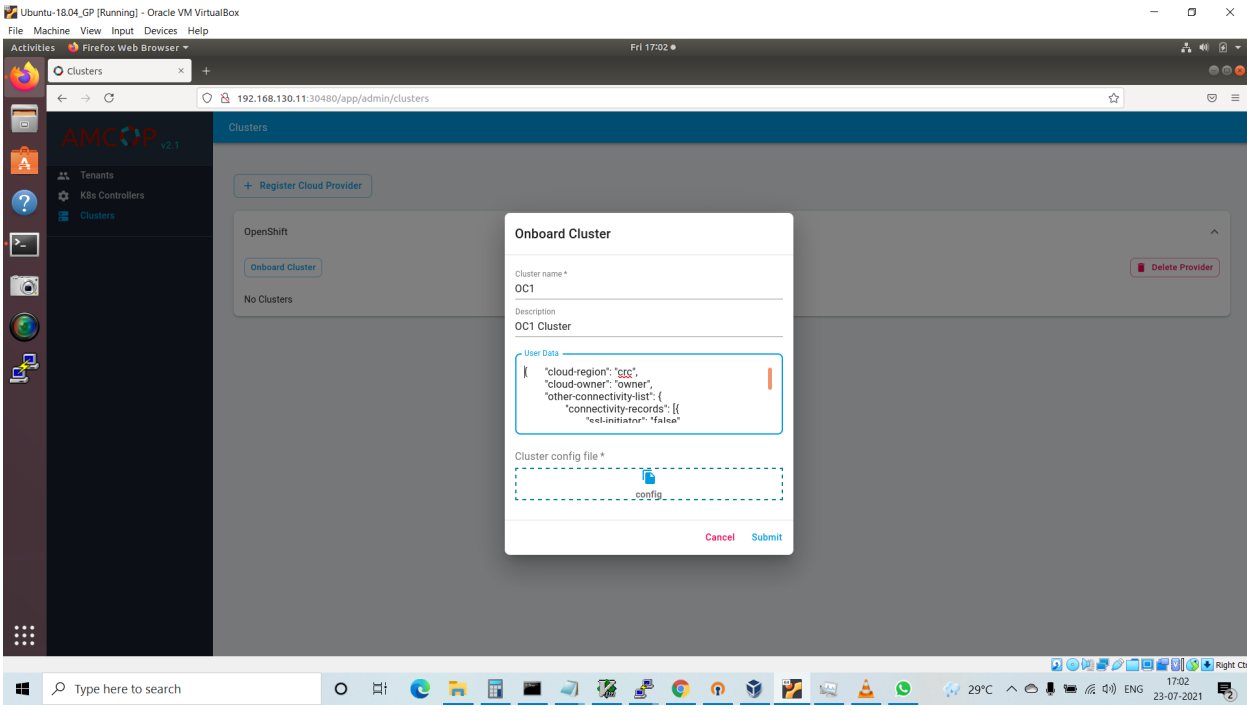
    b. Download the target cluster's k8s config file to your local workstation. It will be required in the next step.
    Once a cloud provider is registered it will appear as an expandable row as shown below. Click on the row to expand it. Once the row is expanded click on the *OnboardCluster* button to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

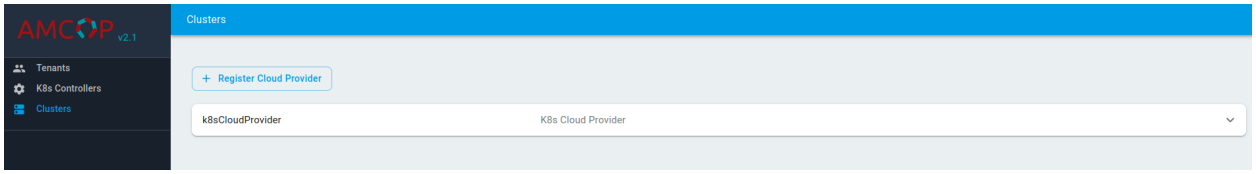
the target cluster's kube config file. If your cluster needs some additional information such as username, password etc, add them to the user data field.

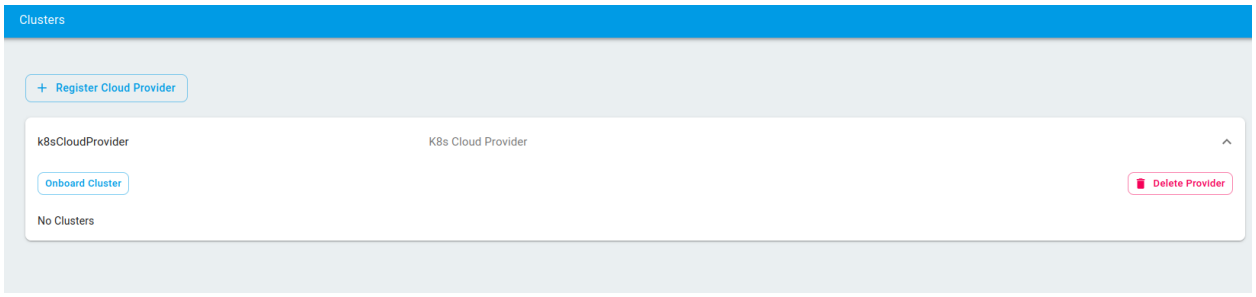
Note:

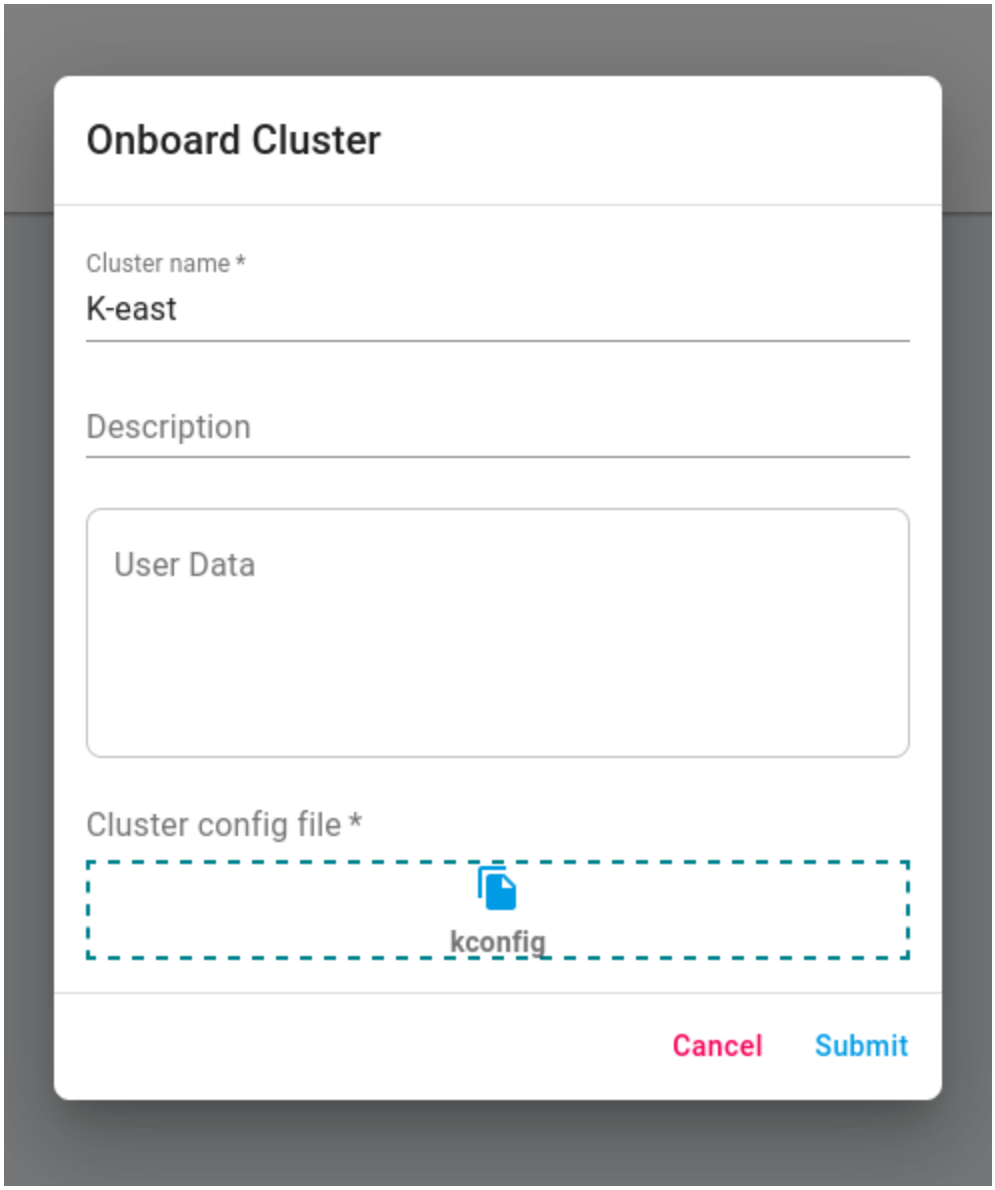
For other platforms such as Openshift, more parameters need to be input, which are mentioned below.

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

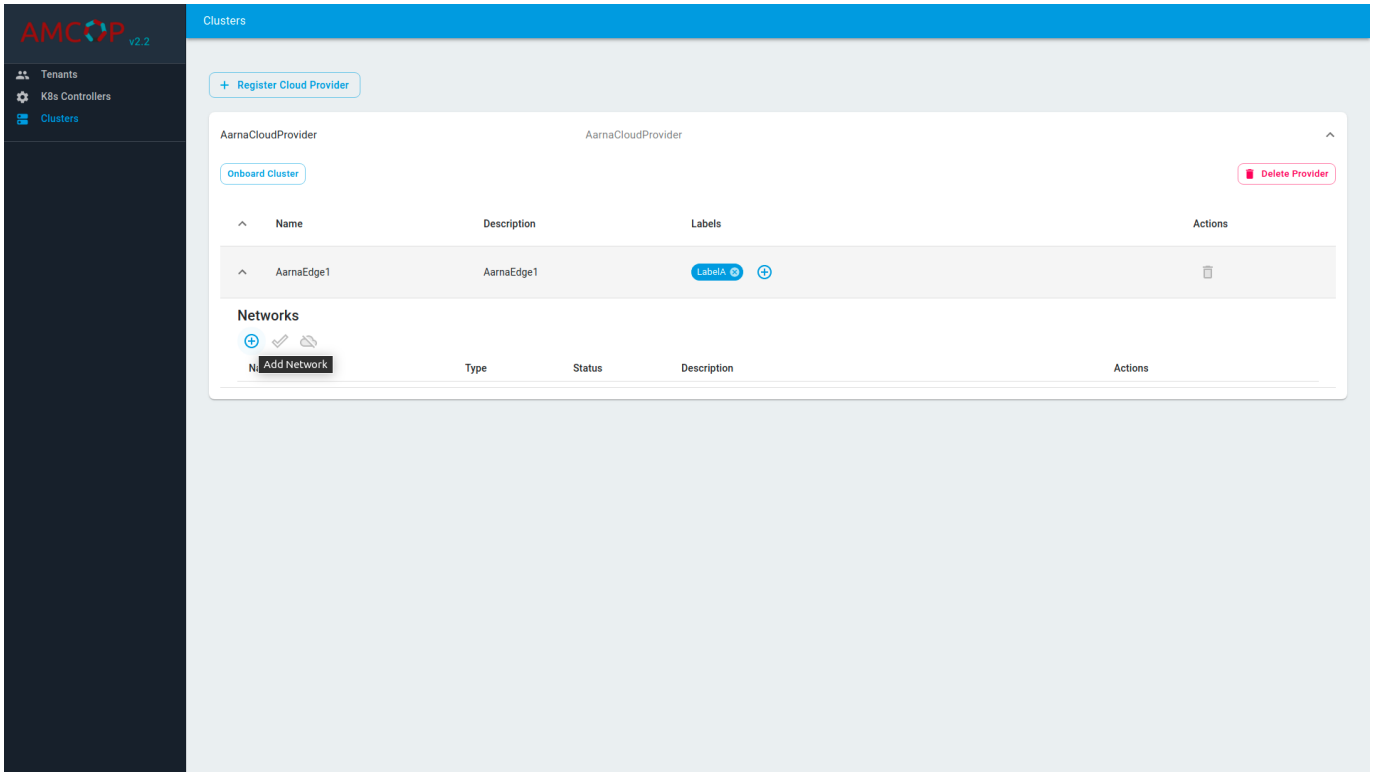



AMCOP v2.1
Tenants
K8s Controllers
Clusters
Clusters
Register Cloud Provider
k8sCloudProvider
K8s Cloud Provider


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.
d. To add networks to the cluster, click on the “*+ network”* button in the actions column.



e. 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*,

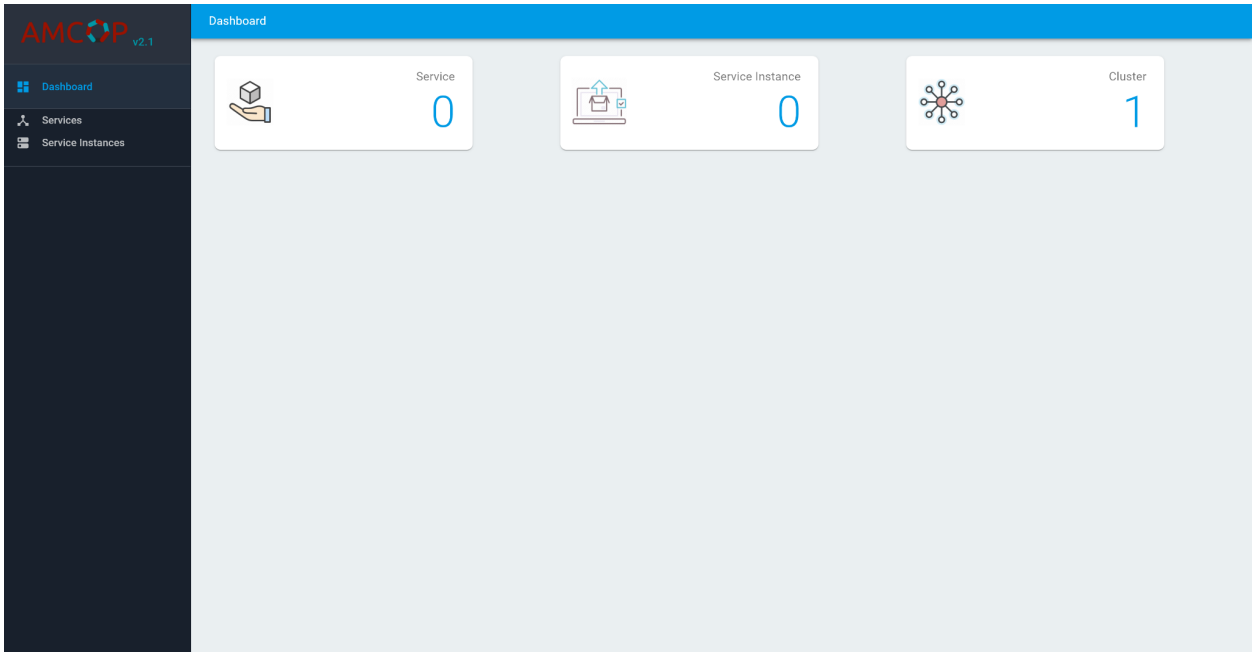
```
{
      "cniType": "ovn4nfv",
      "ipv4Subnets": [
            {
                  "subnet": "192.168.20.0/24",
                  "name": "subnet1",
```

```
                                        "gateway":  "192.168.20.100/32"
                                    }
                                ]
                    }
```

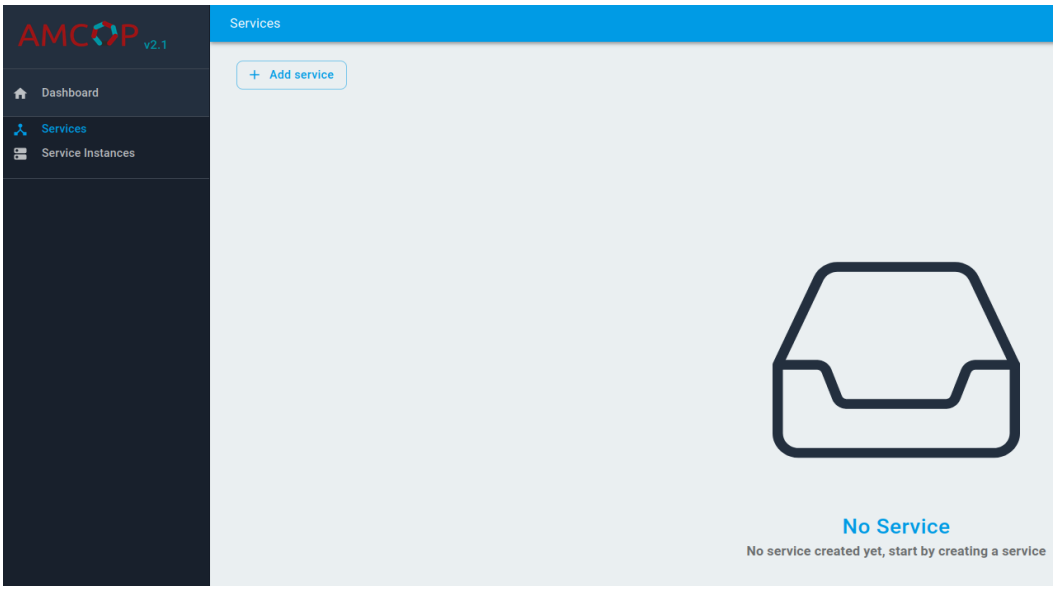
Service Designer User

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.

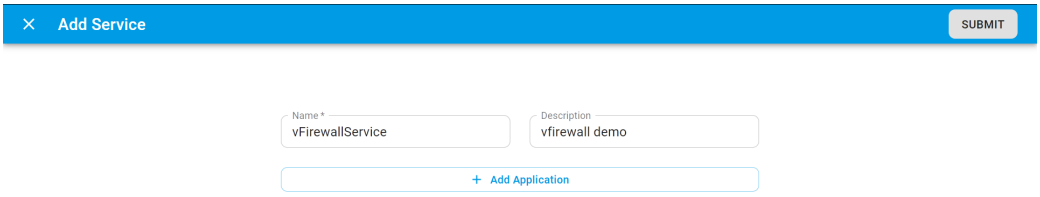
Once a tenant name is clicked, the tenant dashboard is displayed.
The dashboard gives an overview of the tenant resources like total service count, service instance count and available cluster count.



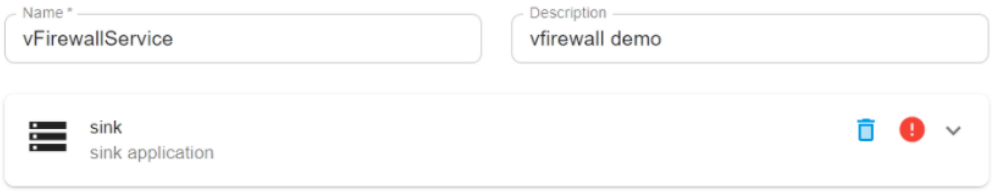
1. Add a Service
    a. To add a service first click on the Services tab from the left-hand side menu.



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

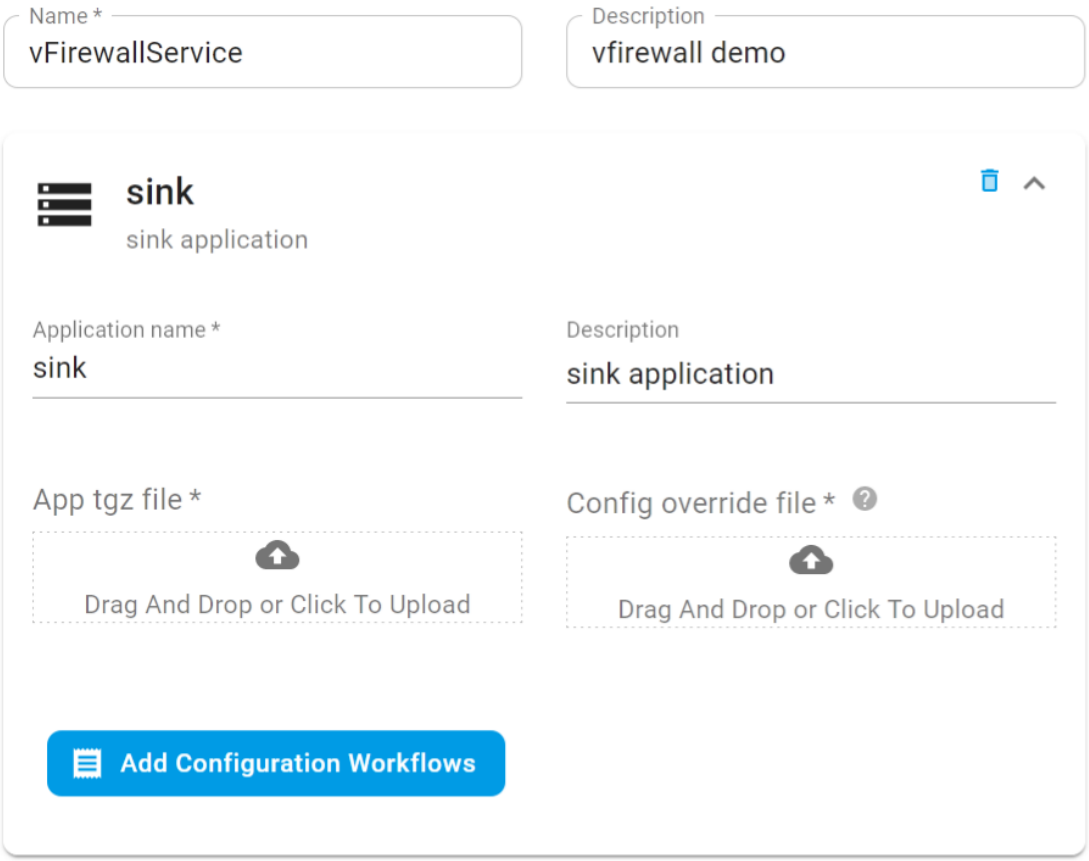


d. In the form to add the application, fill in the name and description of the application and click *Create*.
NOTE: The application should match the helm chart name in the package. For example in case of vFW the **app names should be sink, packetgen and firewall** as the helm charts are sink.tgz, packetgen.tgz 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.



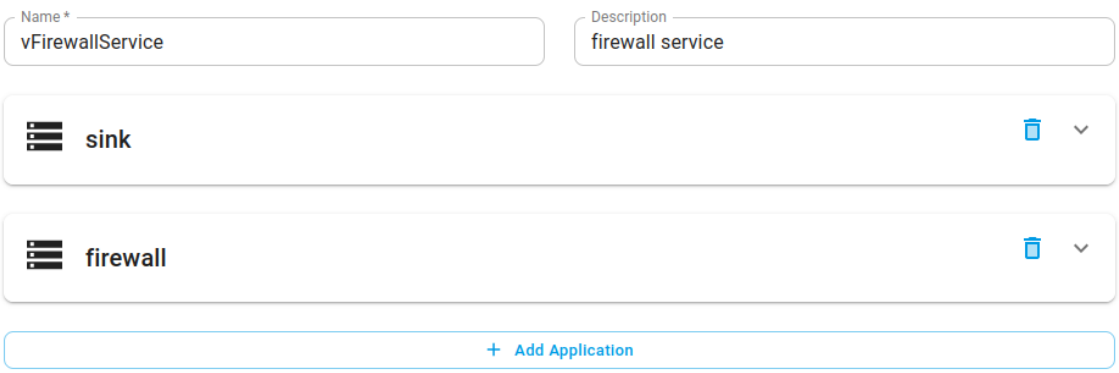
f. Click on the app row to expand it. Now upload the app .tgz package file and profile .gz file which contains the override files by clicking on the upload button or

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/ directory.
The profile bundle is present in /home/<user>/aarna-stream/cnf/payload/. The name of the bundle is profile.tar.gz

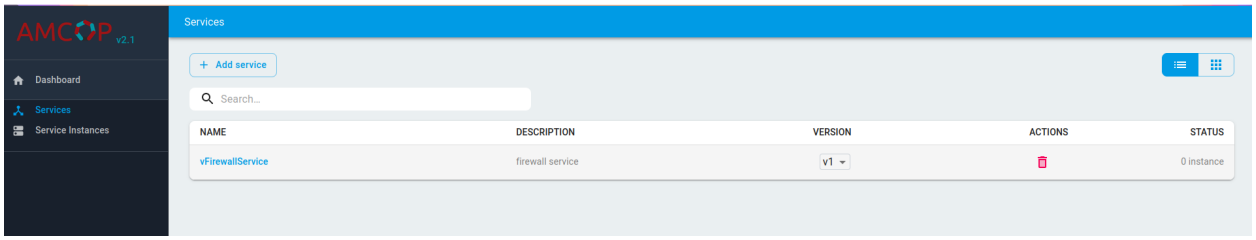


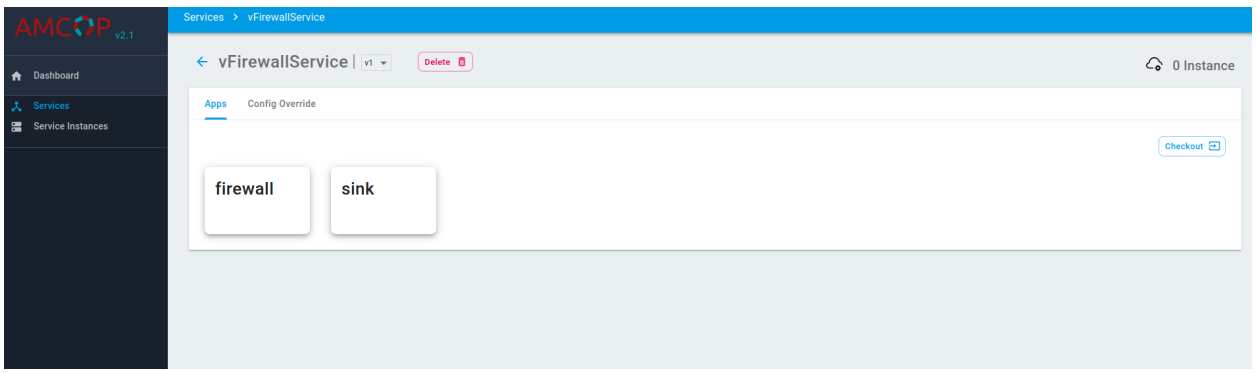
g. For vfirewall, you need to add a total of three applications, sink, packetgen and firewall. So repeat the above step to add the other 2 applications.

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.





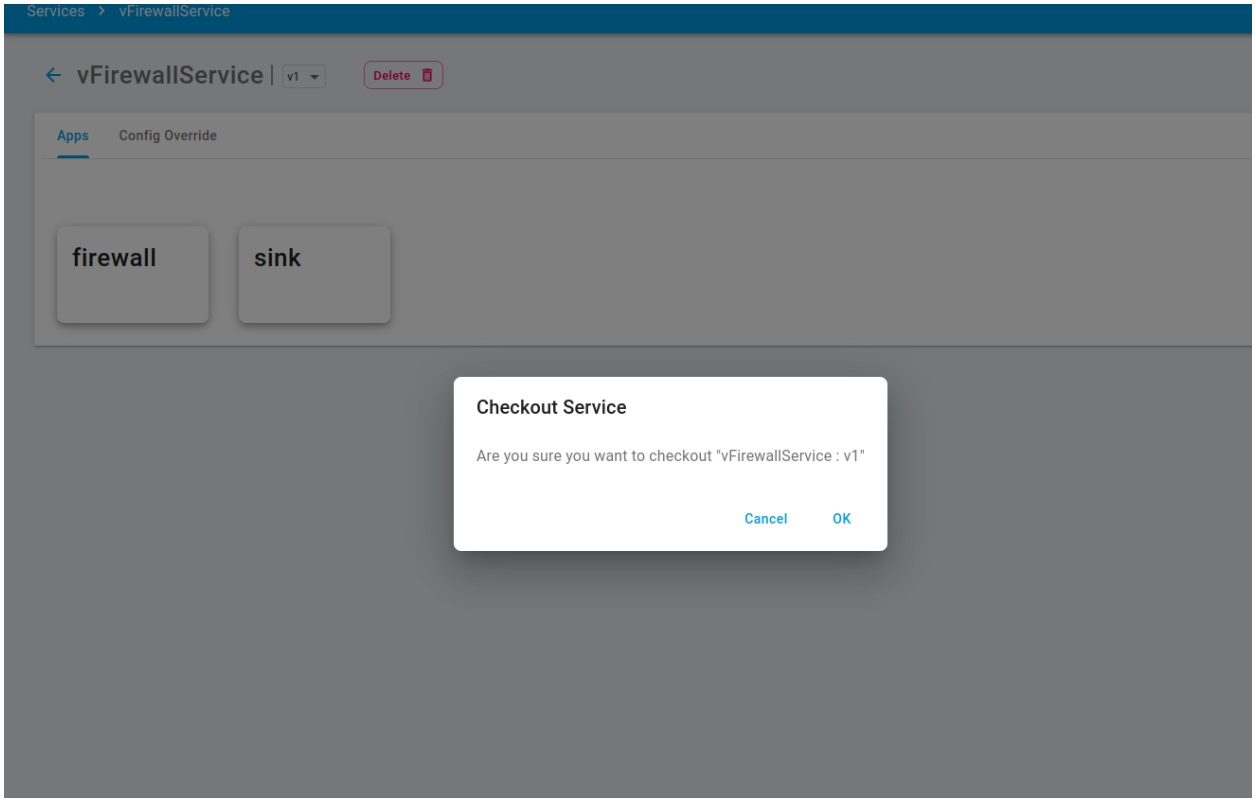
2. Checkout and Edit a service

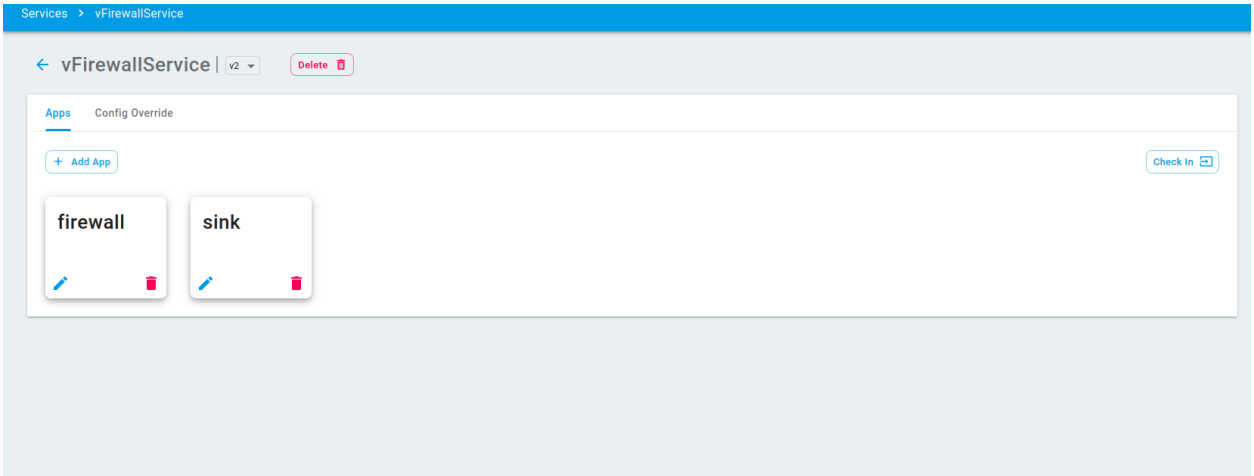
   You can perform the following operations on the created service.

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

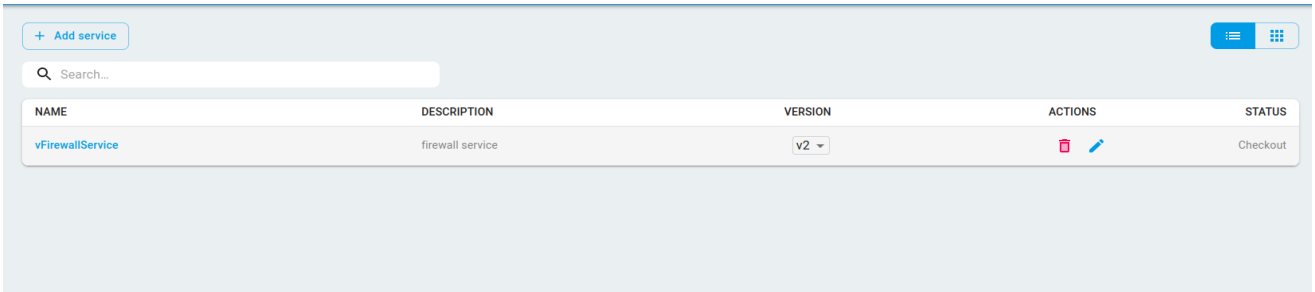
c. Add new applications to the service.

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.

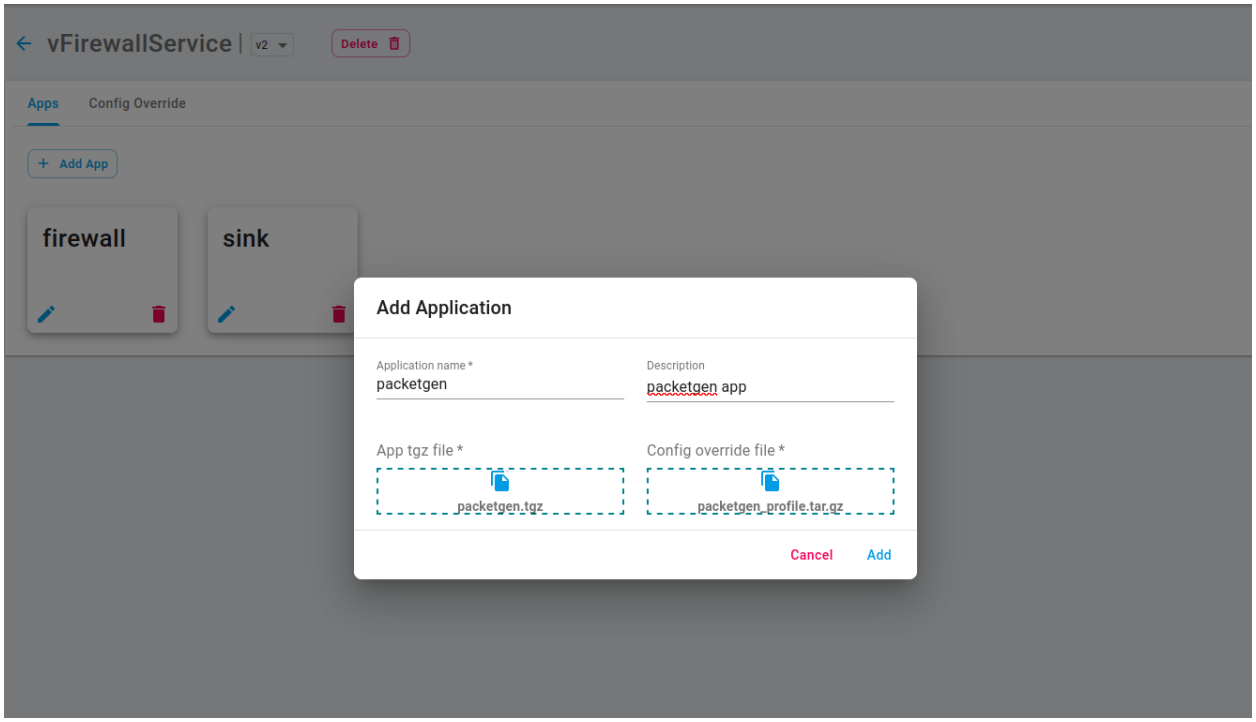




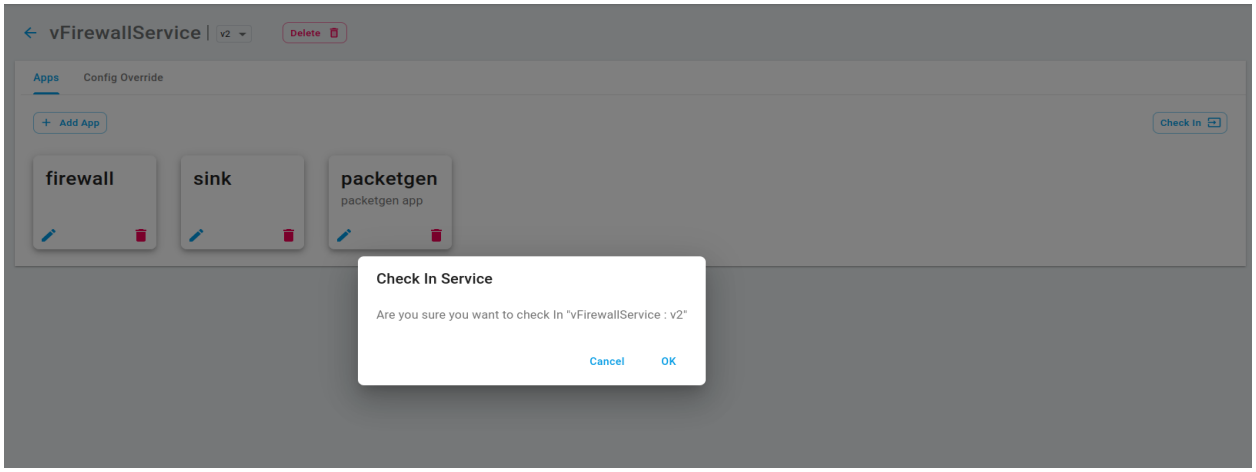
2. If needed, you can return to the main page and take up the modification work later. The status of the service remains in the checkout state.

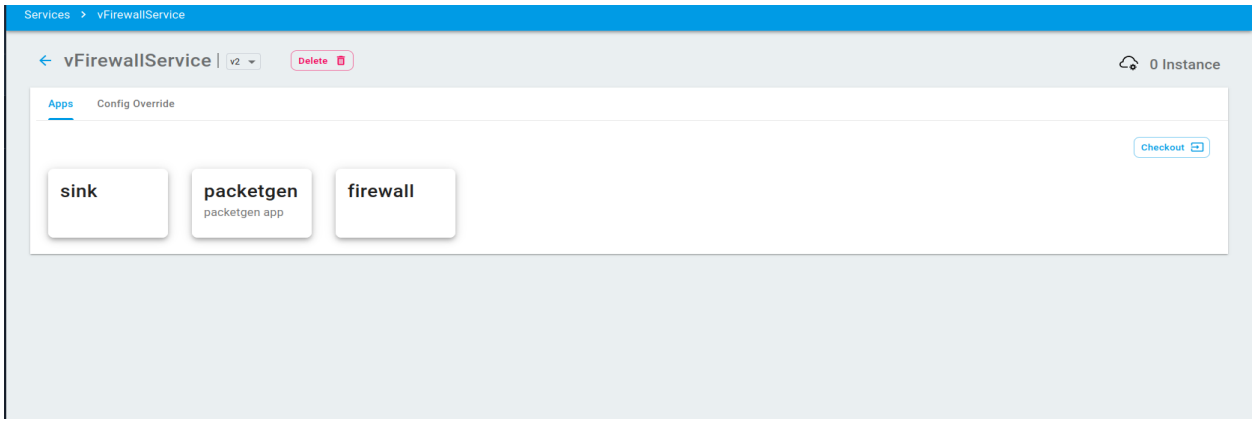


3. Add a new application to the checked-out service.



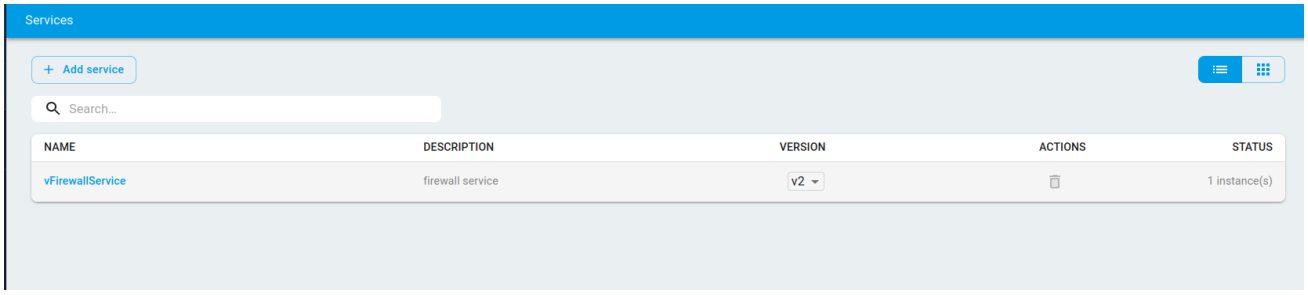
4. Check in.





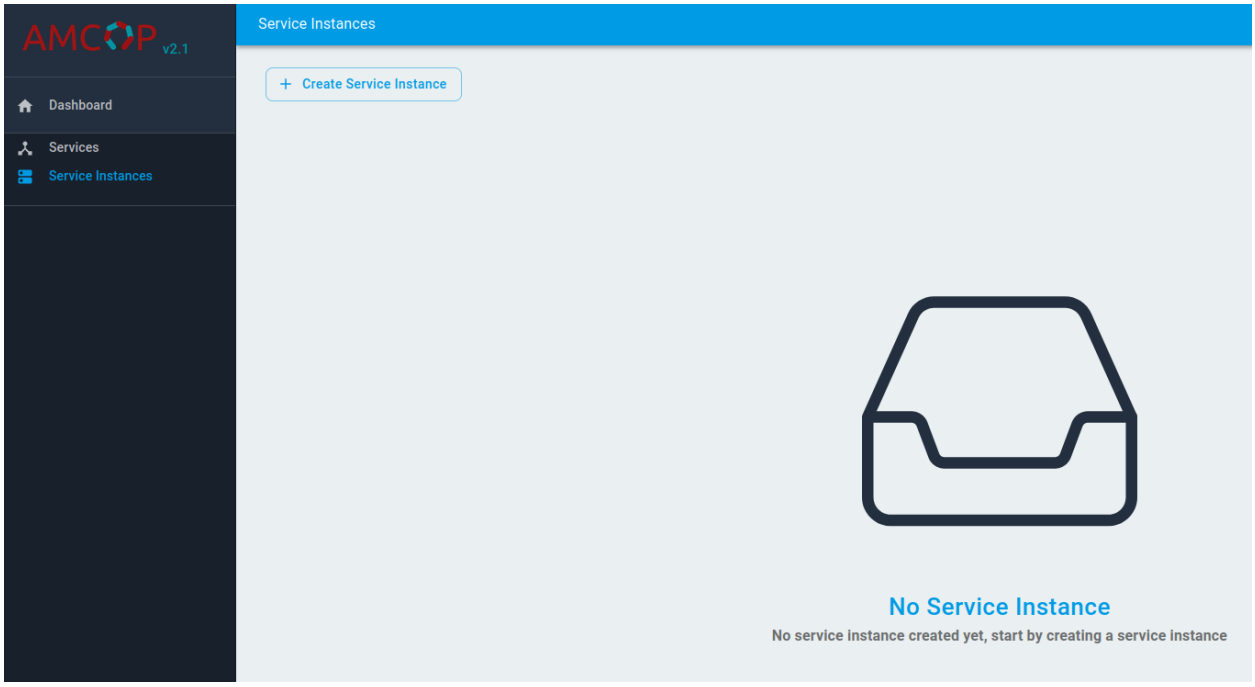
On the service page, the status of the service means the following,

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

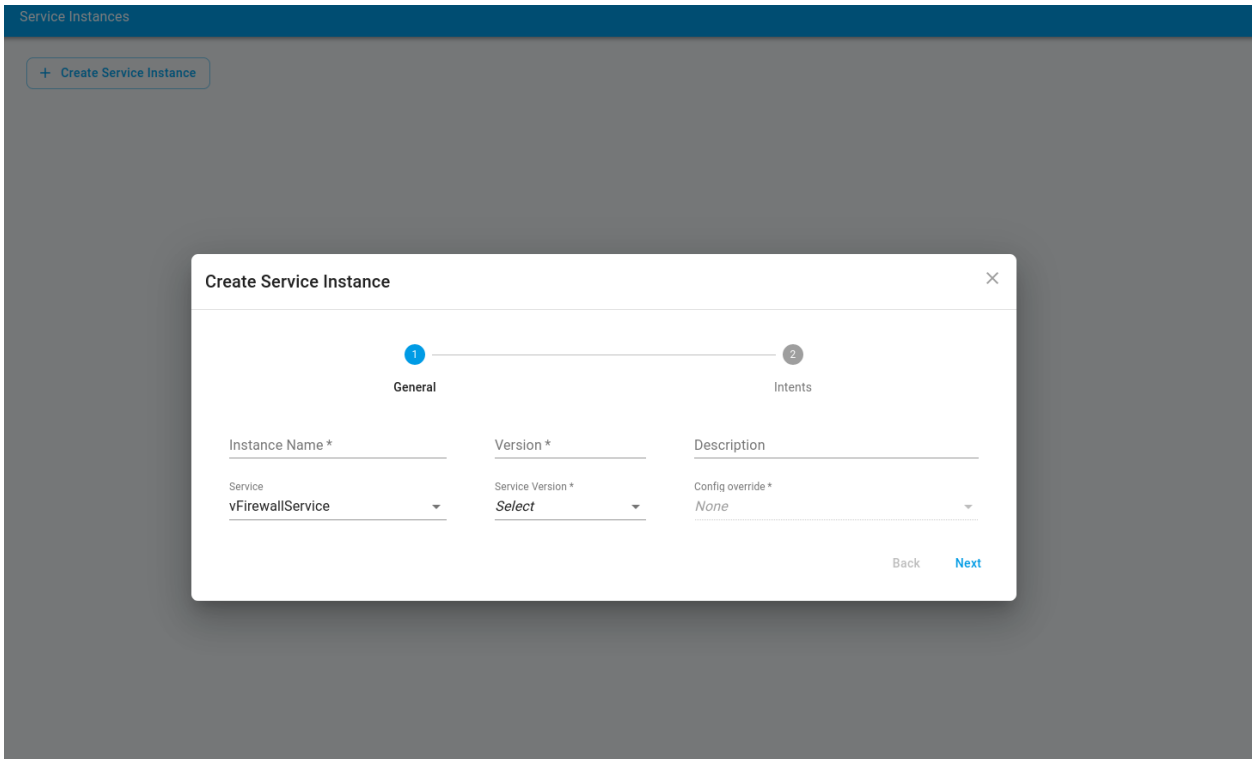


3. Create a service instance.

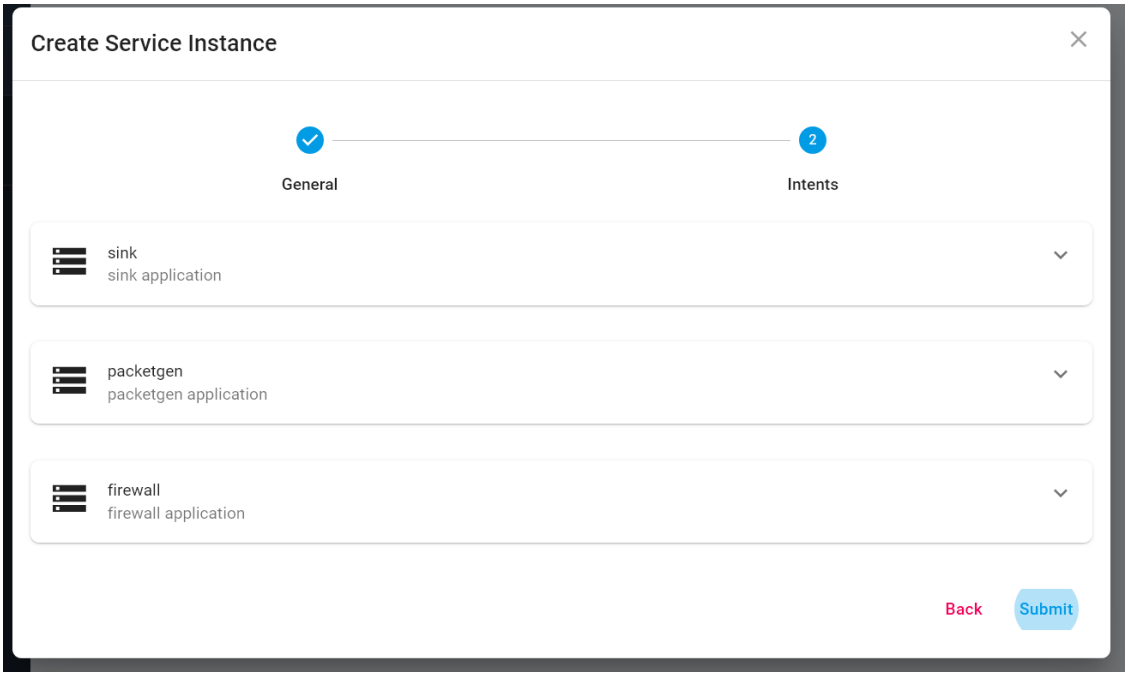
    a. To create a service instance, go to the service instances screen from the left-hand side navigation and then click on create *service instances* button, this will open the service instance form.



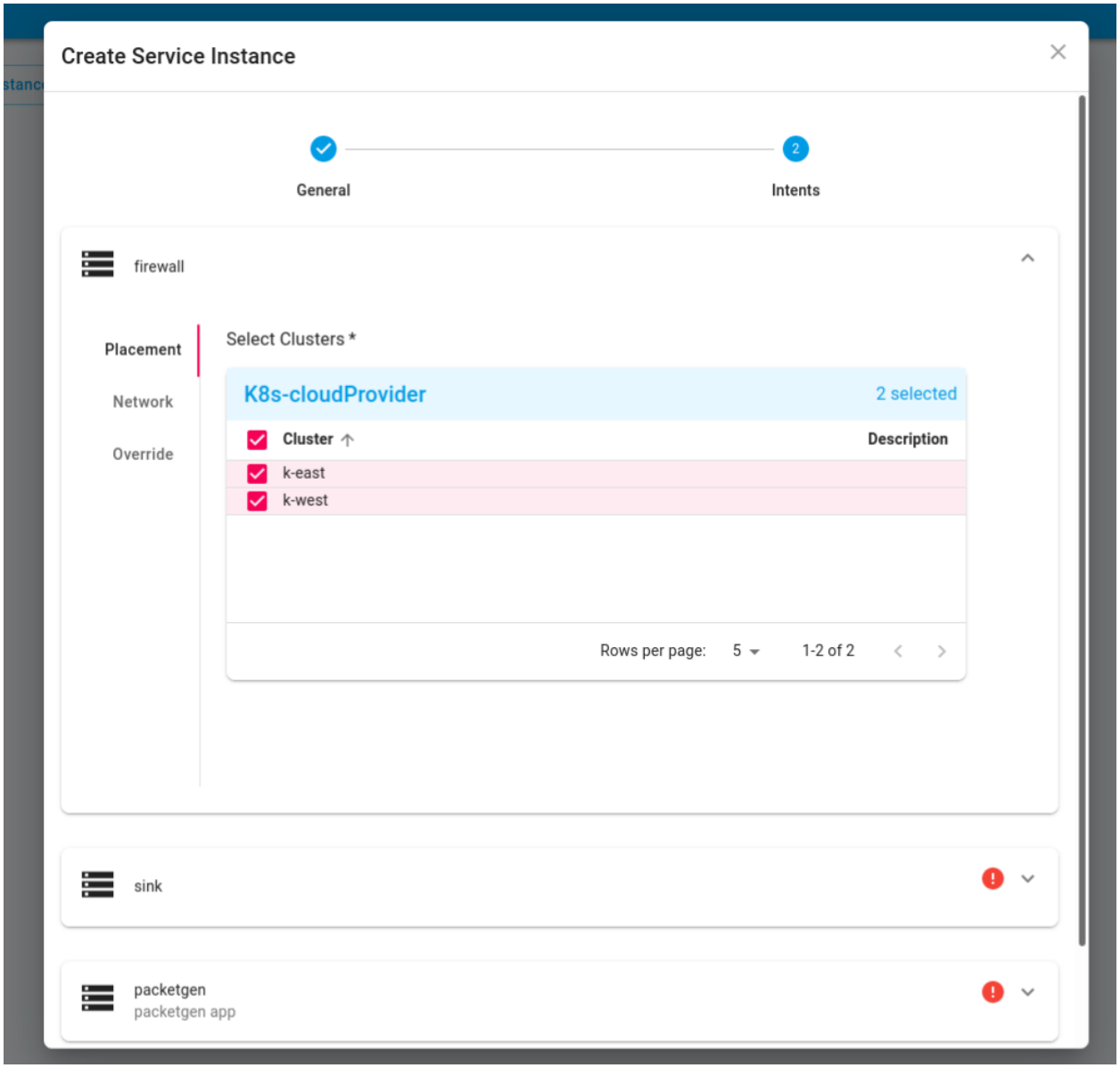
b. In the service instance form, fill in the details like service instance name, version, description, etc. Also, select the service from the dropdown for which the instance needs to be created. In this case select vfirewall Service.



c. When a service is selected, it's corresponding override file is automatically selected.
d. You can also give 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.

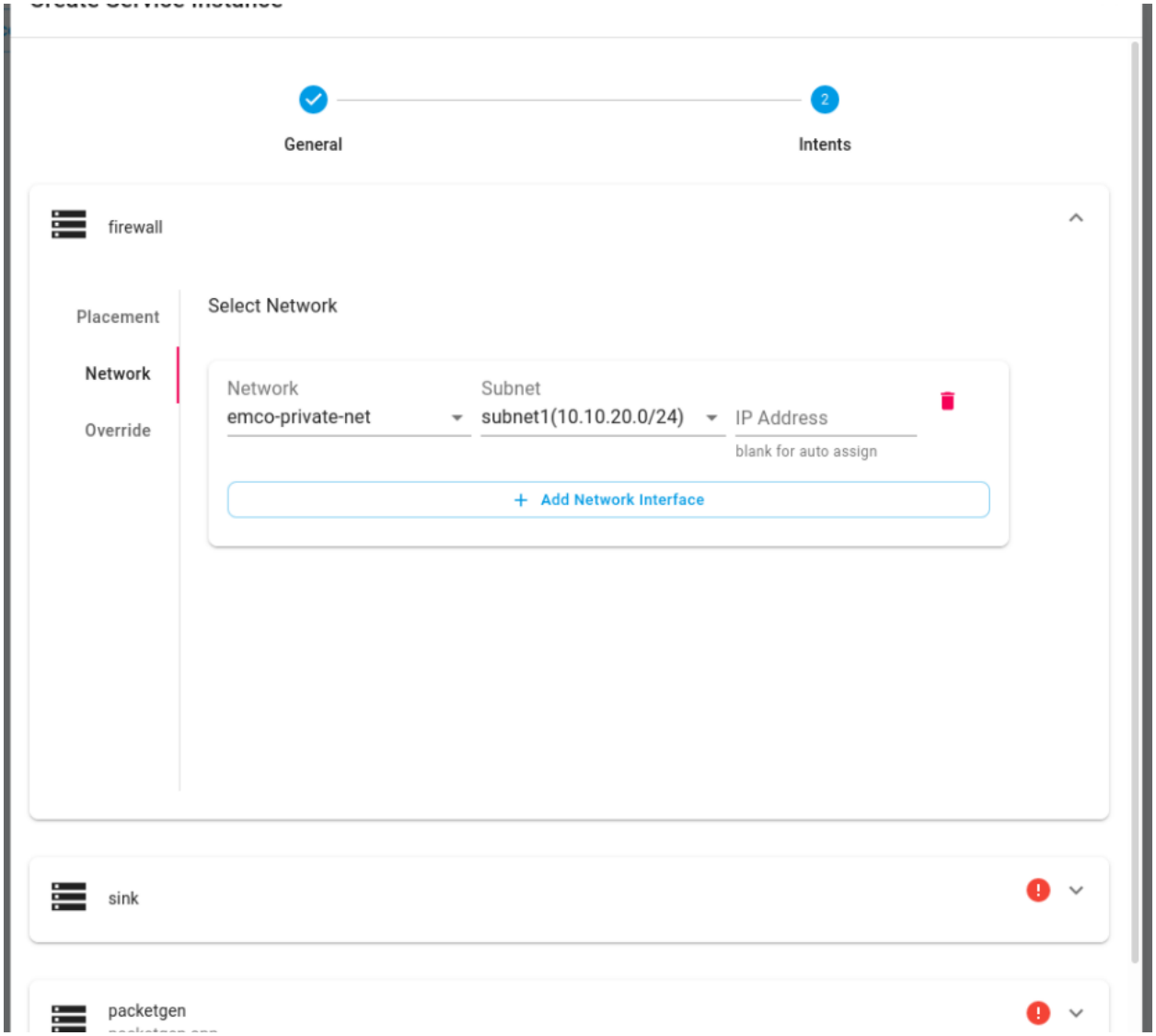


g. Once the application row is expanded, you can see two tabs: *placement and network*. One is for placement and the other is for adding the network interfaces. In the placement tab select the clusters in which you want your app to be deployed.

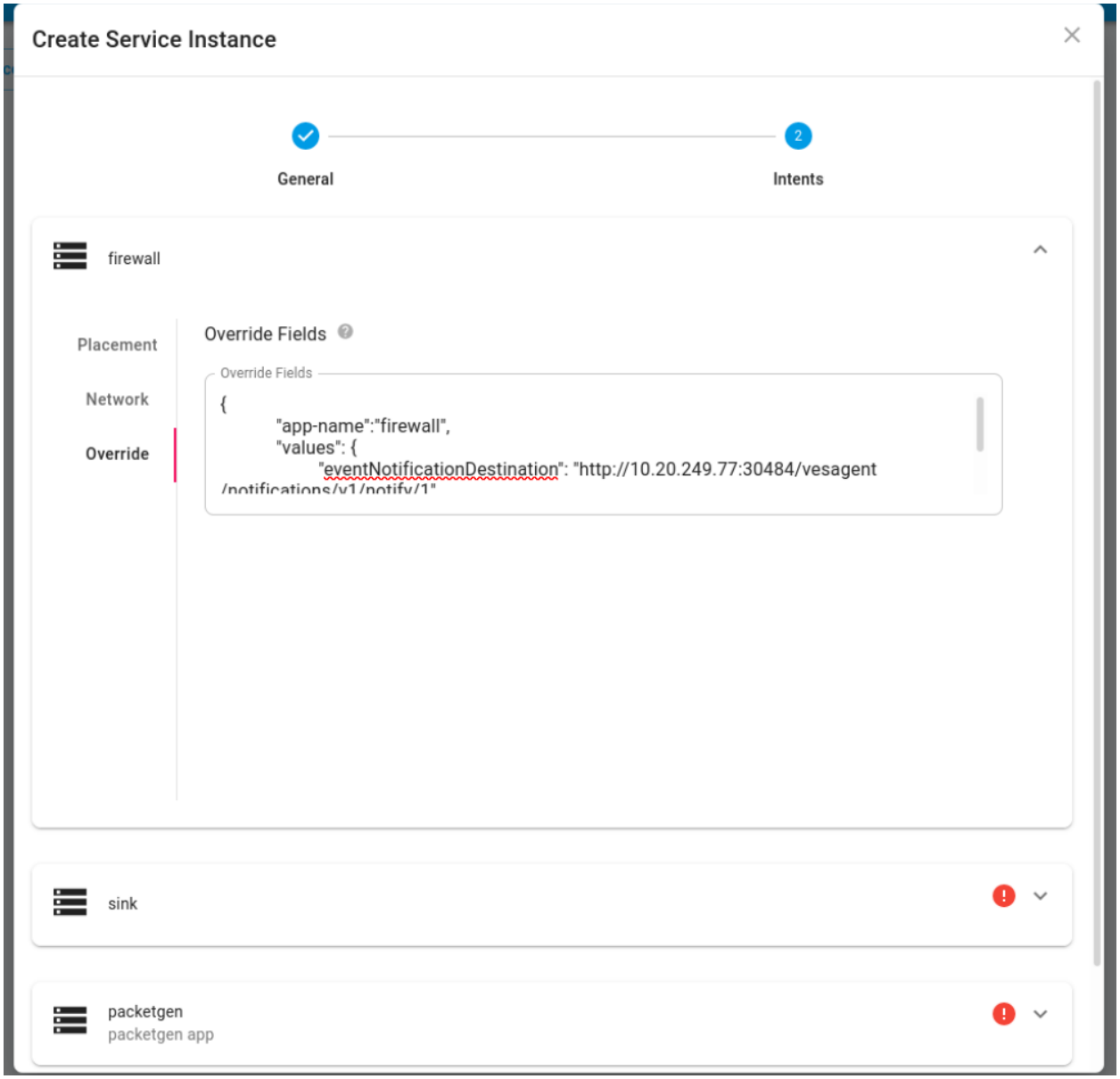


h. Now click on the *network* tab and then click on *Add Network Interface*. Then select the network from the drop down menu and then the subnet. Leave the IP field blank to auto assign the IP address or fill in the IP address if required. Repeat this step to add more network interfaces.
For the vfw following network interfaces are to be added for each application,
    1. sink:
       protected-private-net, ip: 192.168.20.3
       emco-private-net, ip: 10.10.20.4
    2. packetgen:
       unprotected-private-net, ip: 192.168.10.2
       emco-private-net, ip: 10.10.20.2
    3. firewall:

```
protected-private-net, ip: 192.168.20.2
emco-private-net, ip: 10.10.20.3
unprotected-private-net, ip: 192.168.10.3
```



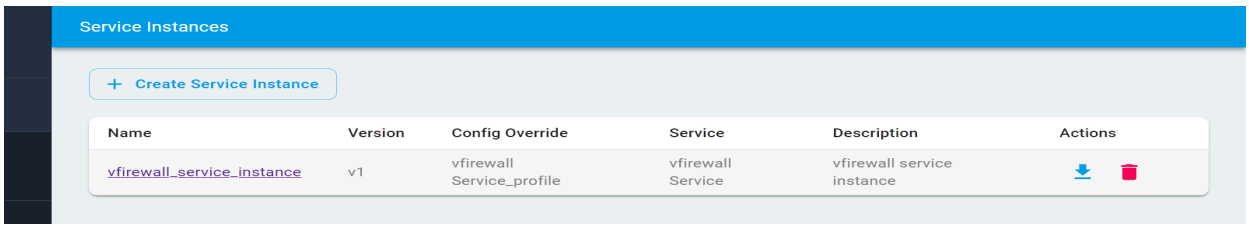
i. If there are any Day 0 override values for an application, they can be added in the *Override* tab. It is optional to add the override values. For documentation purpose following is the screenshot of how the override values can be added,



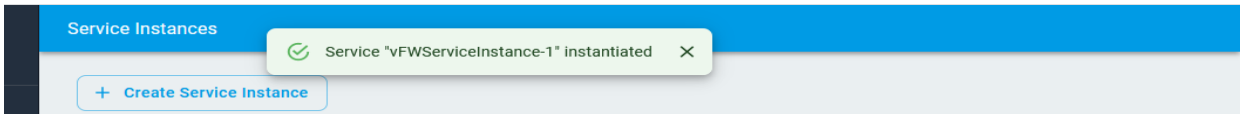
.

j. Now click on the submit button.

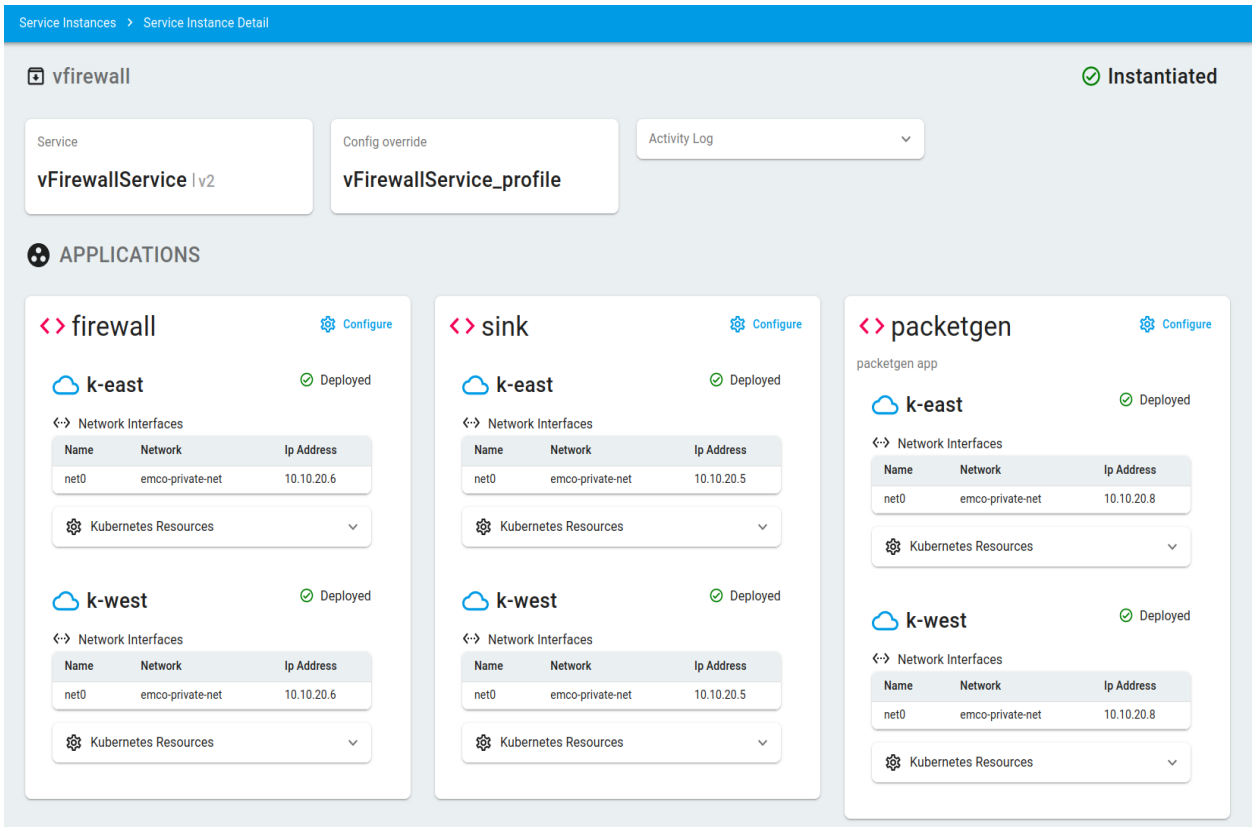
k. Now you can see the service instance. To instantiate the service instance, click on the instantiate button' ' in the actions column.



l. On the 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.

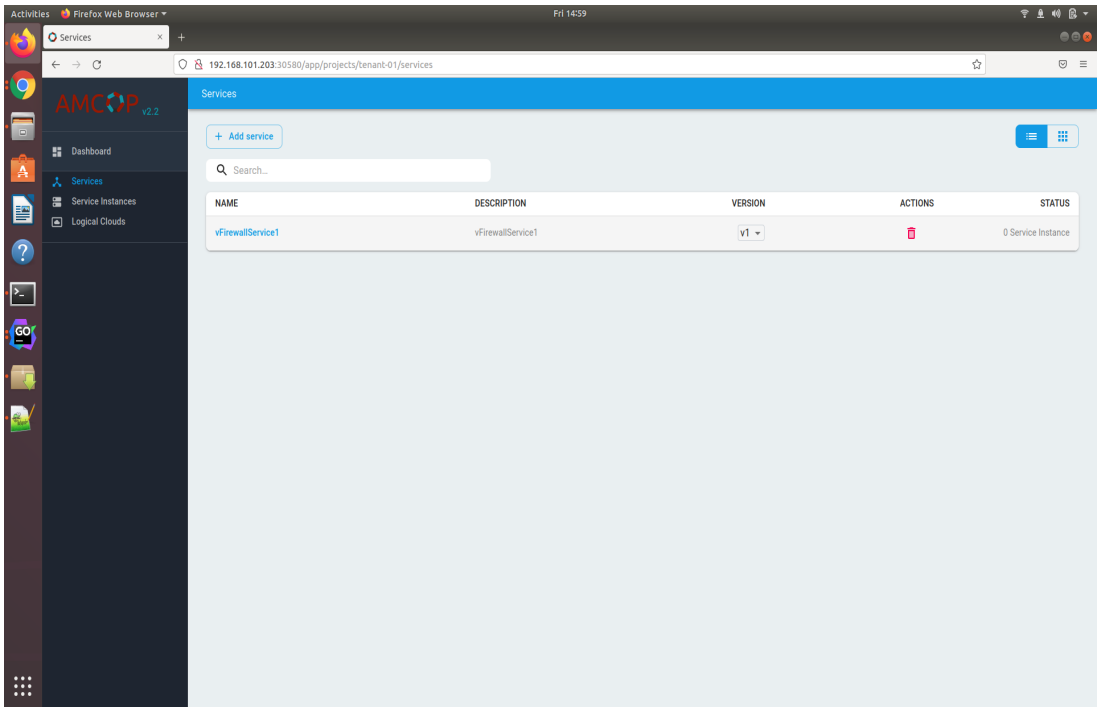


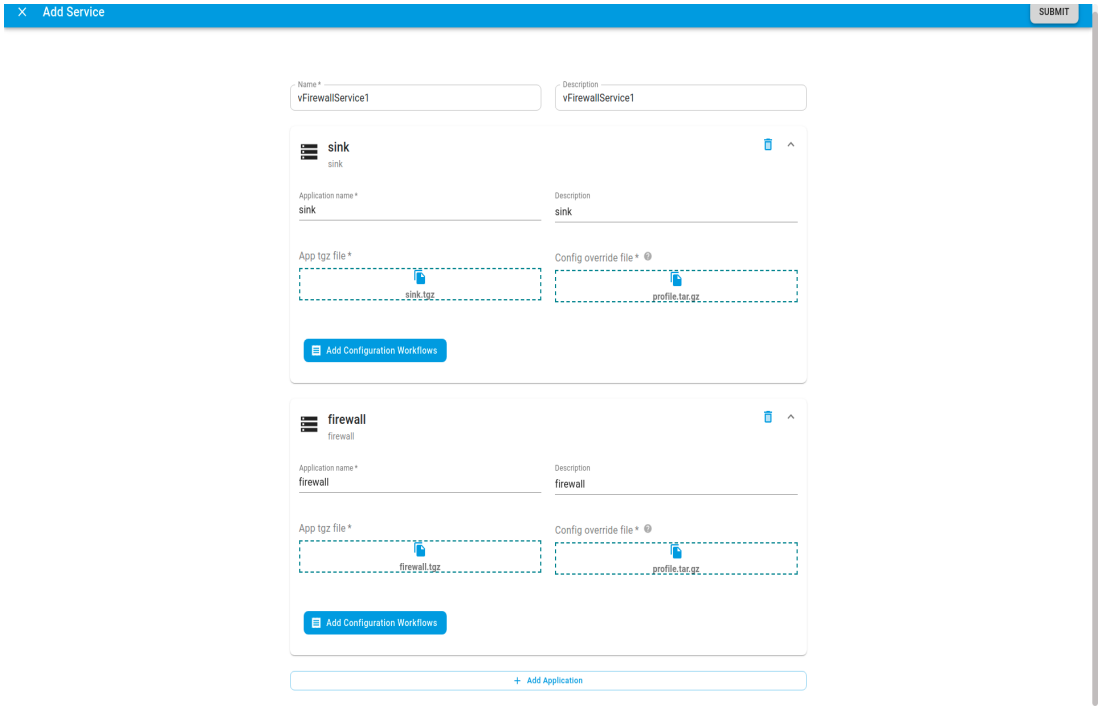
Service Instance Update

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

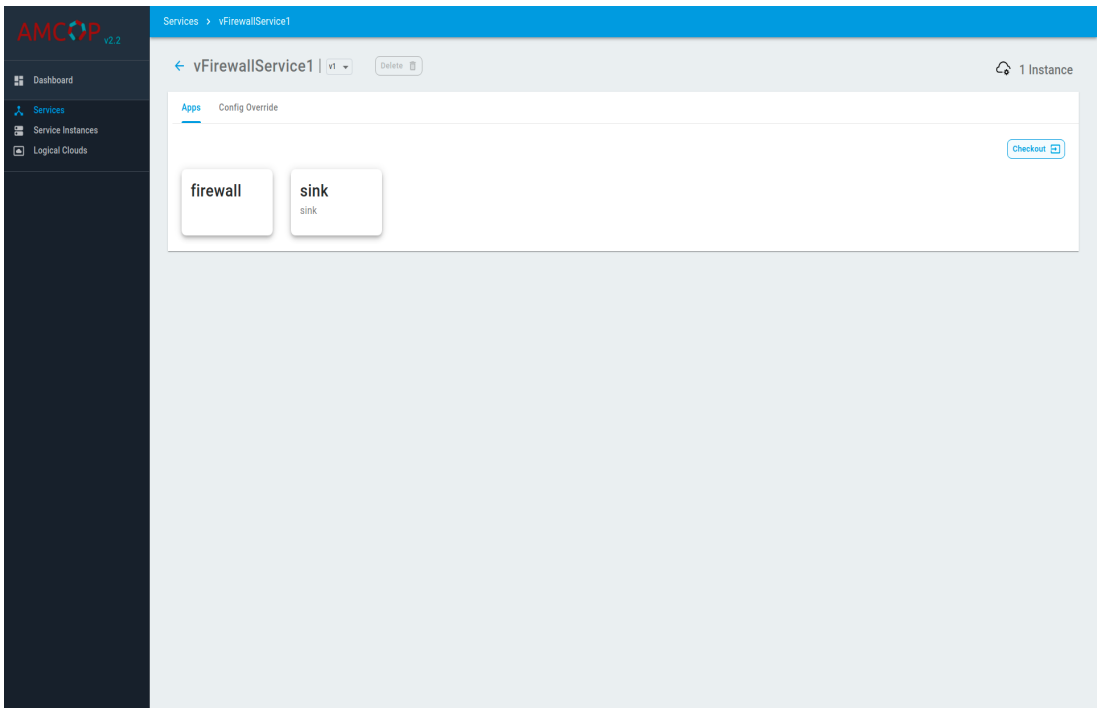
AMCOP supports updating an existing Service instance. Follow the steps given below to perform Service instance update of vFirewall application (shown as an example).

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.

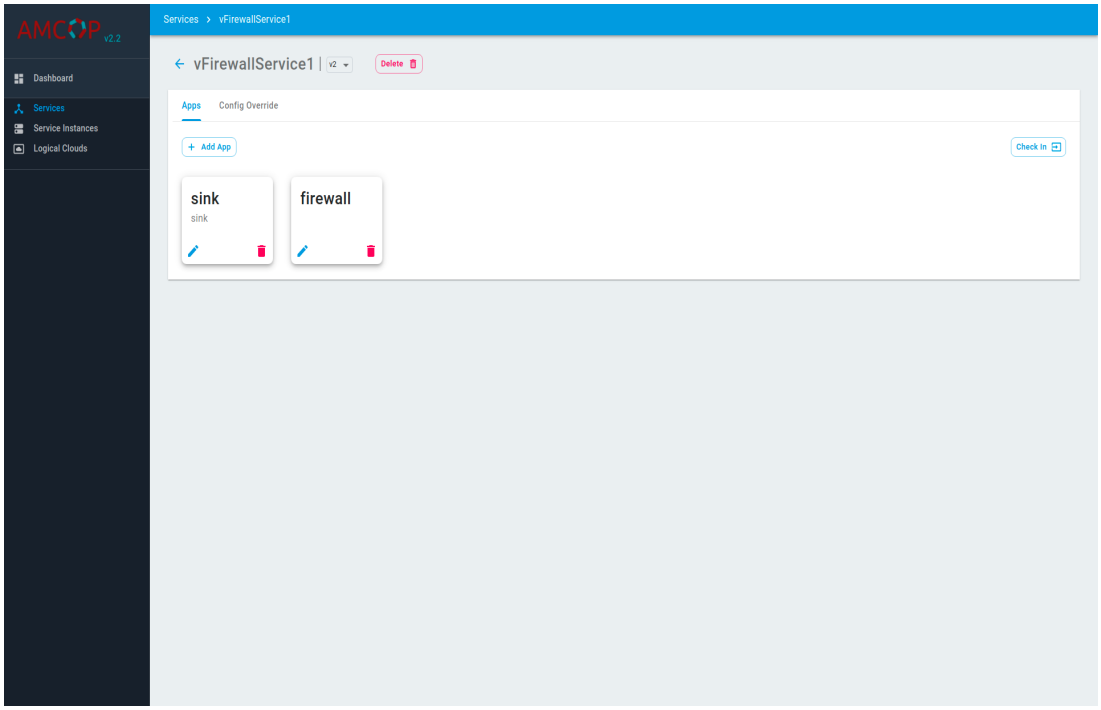


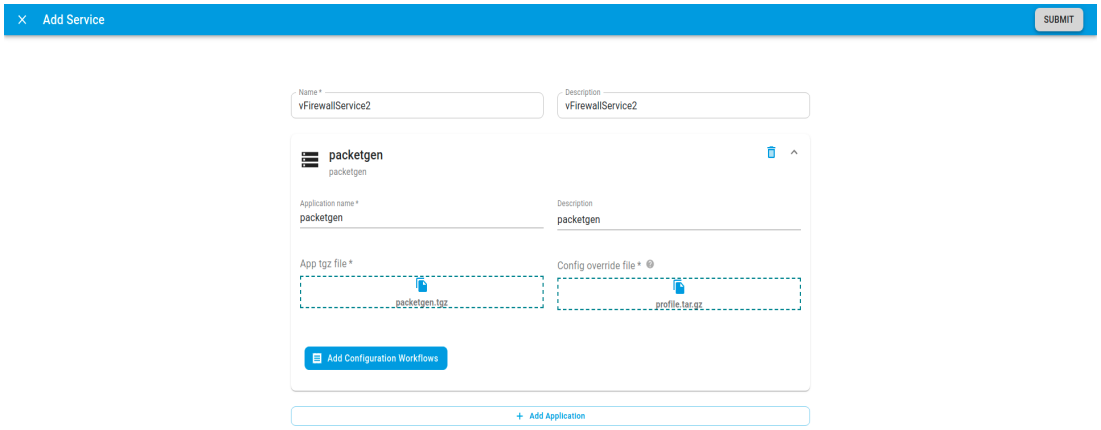


2. As an update to the v1 service add one more app (say ***packetgen***, as shown below) by selecting the ***Checkout*** option in the below screenshot.

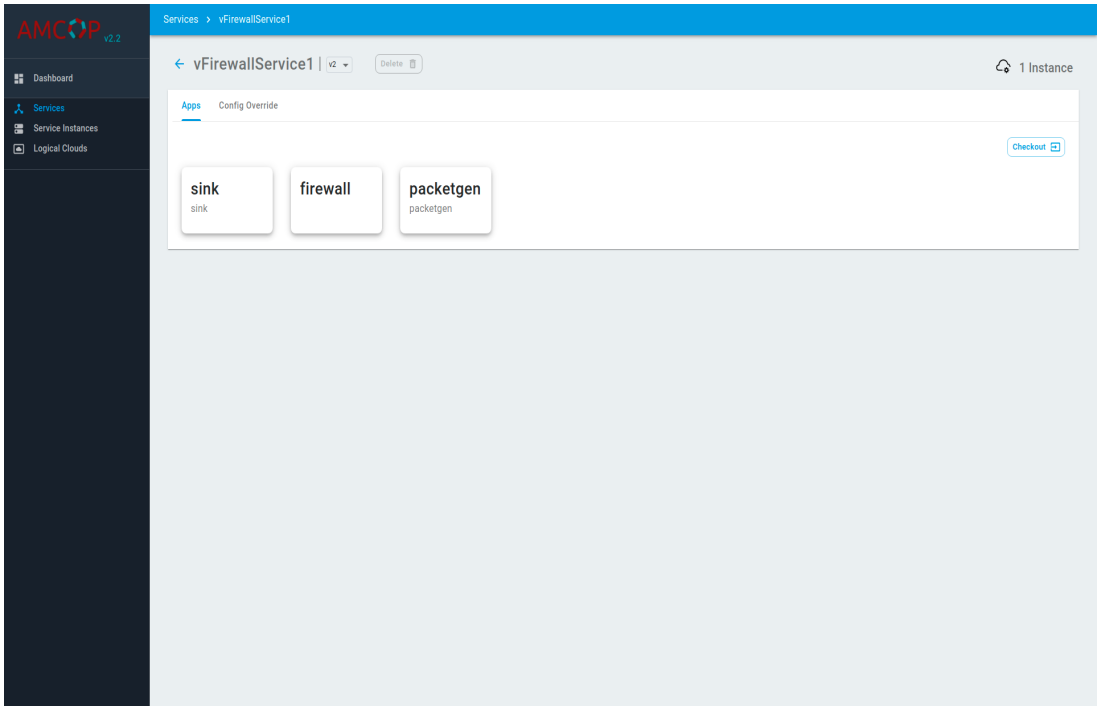


3. Select ***Add App*** to add the ***packetgen*** app and then select the Check-In option to update the service.

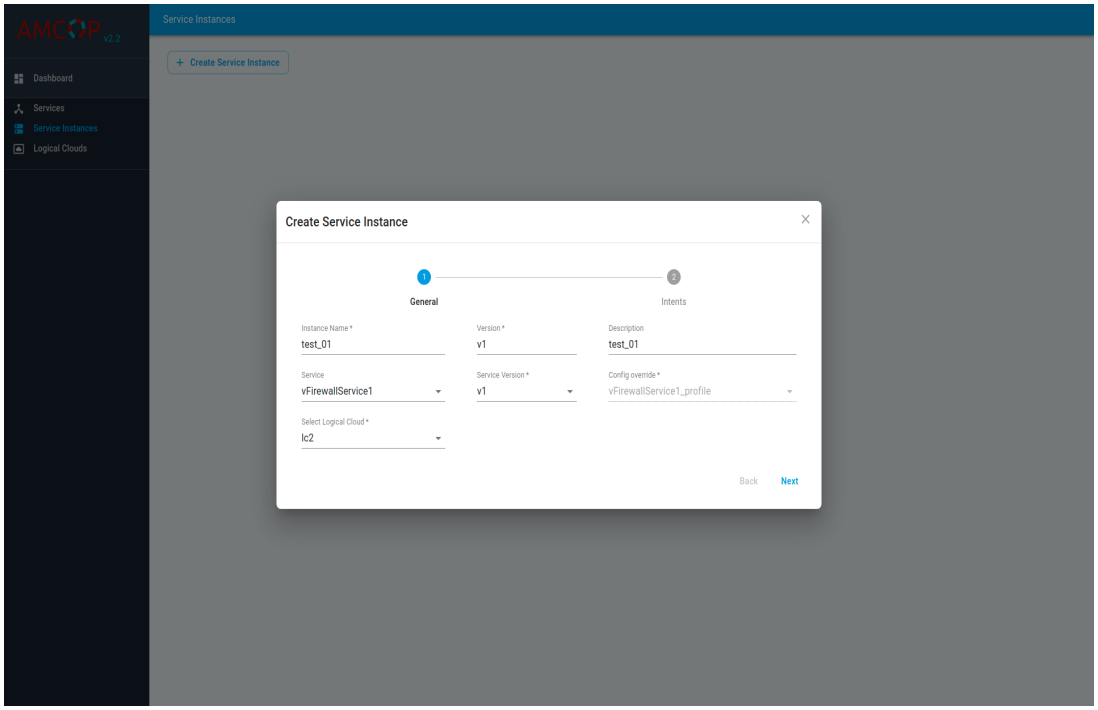




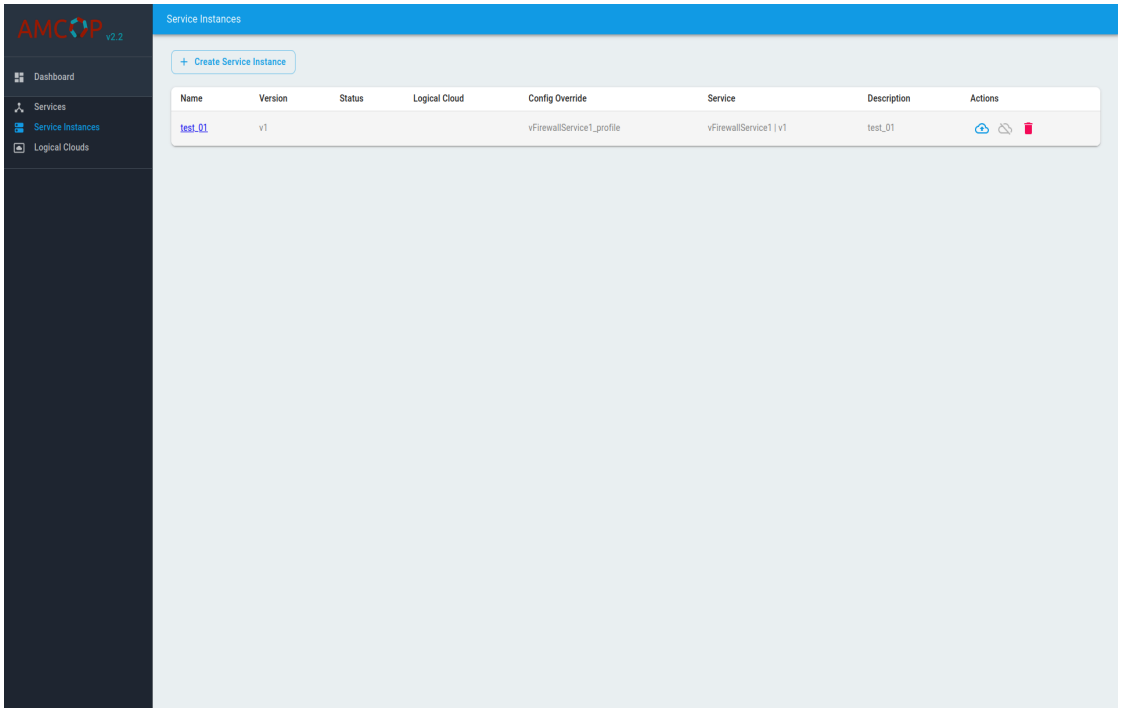
Then click on the submit button on the top right corner to update the changes as version v2 of the service.



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.

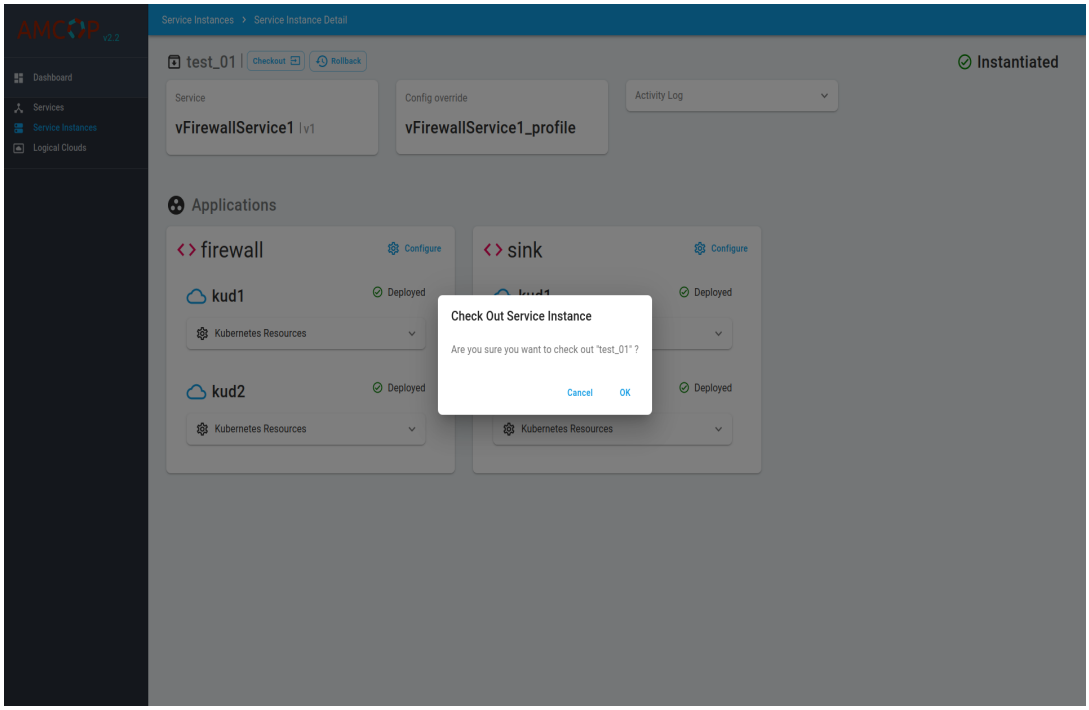


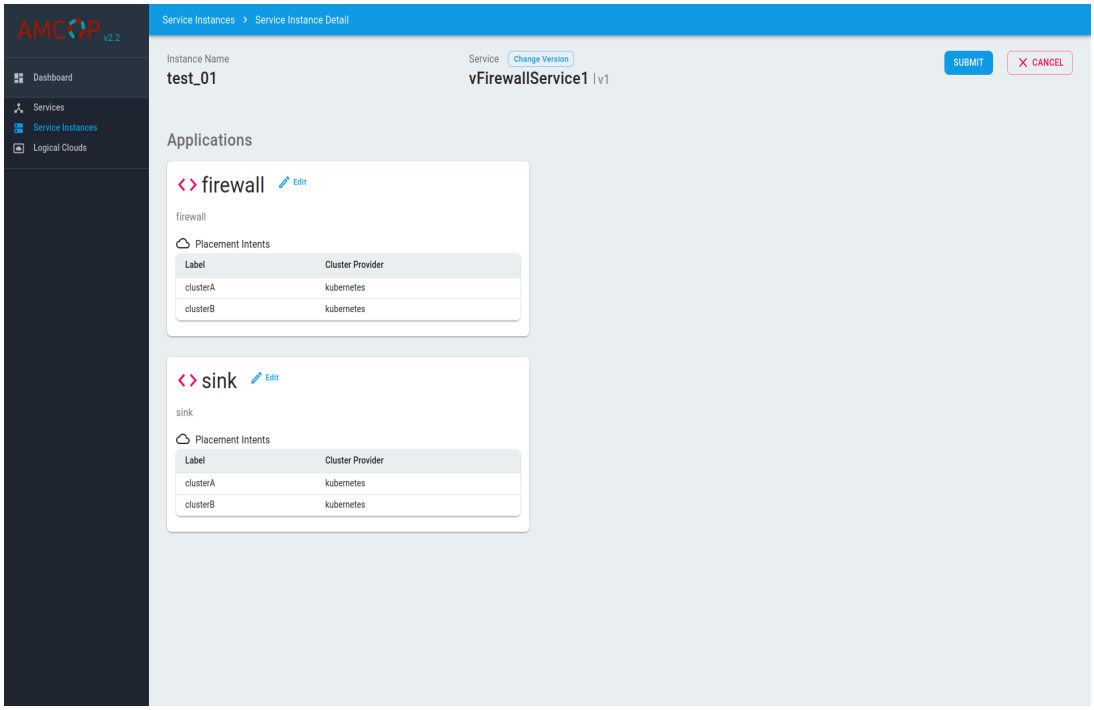
5. Complete the instantiation process as shown below.



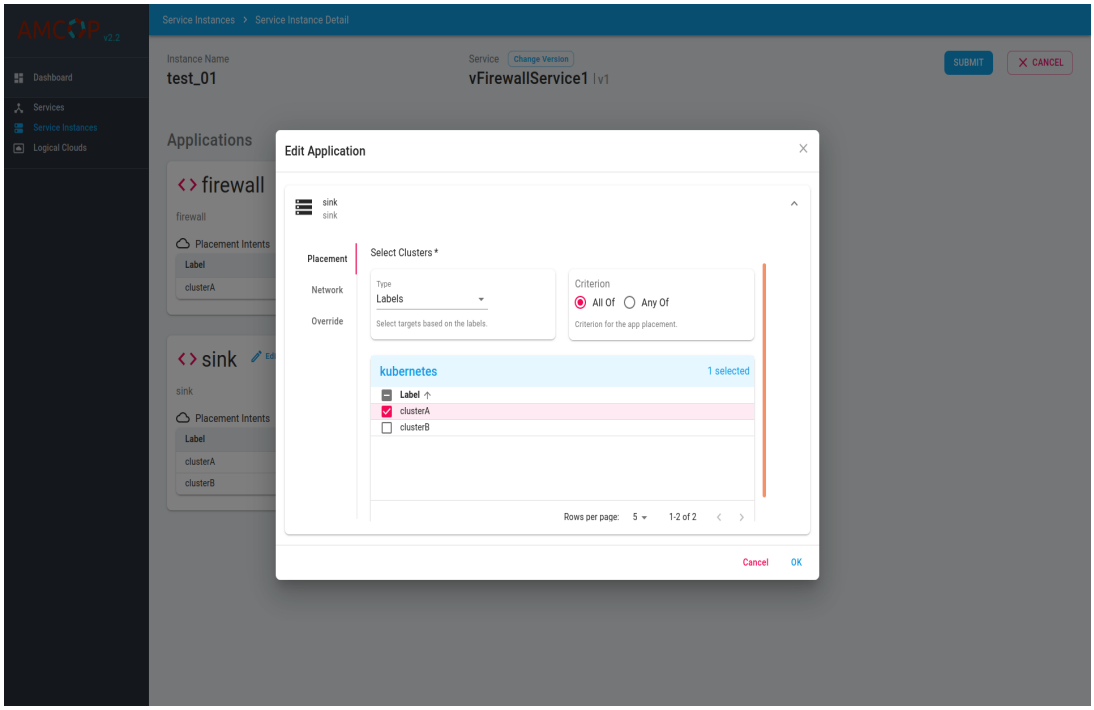
Service Instance "***test_01***" is now created using service "***vFirewallService1***" version v1.

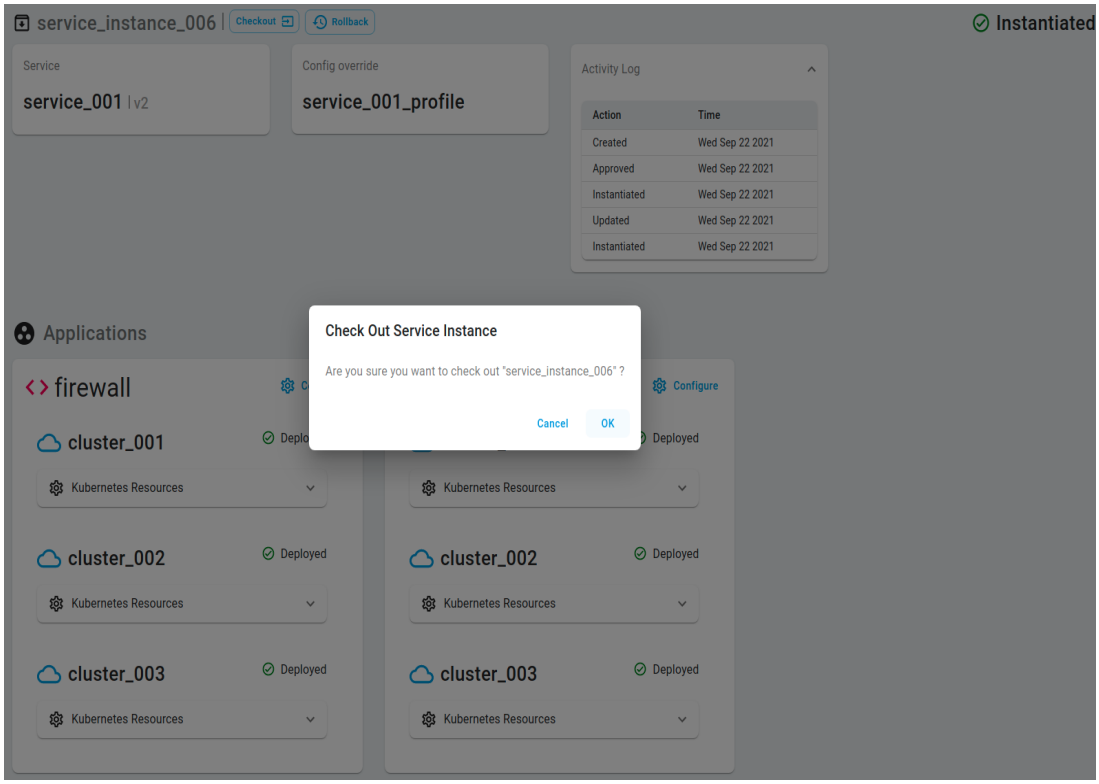
5. Select the ***Service Instances*** option from the left sidebar, Check out the service instance which was created in the above steps by clicking on the checkout button on the top left right ***Checkout*** button at the top, next to the service instance name ***test_01*** and select ***ok***.


AMCOP v2.2
Dashboard
Services
Service Instances
Logical Clouds
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

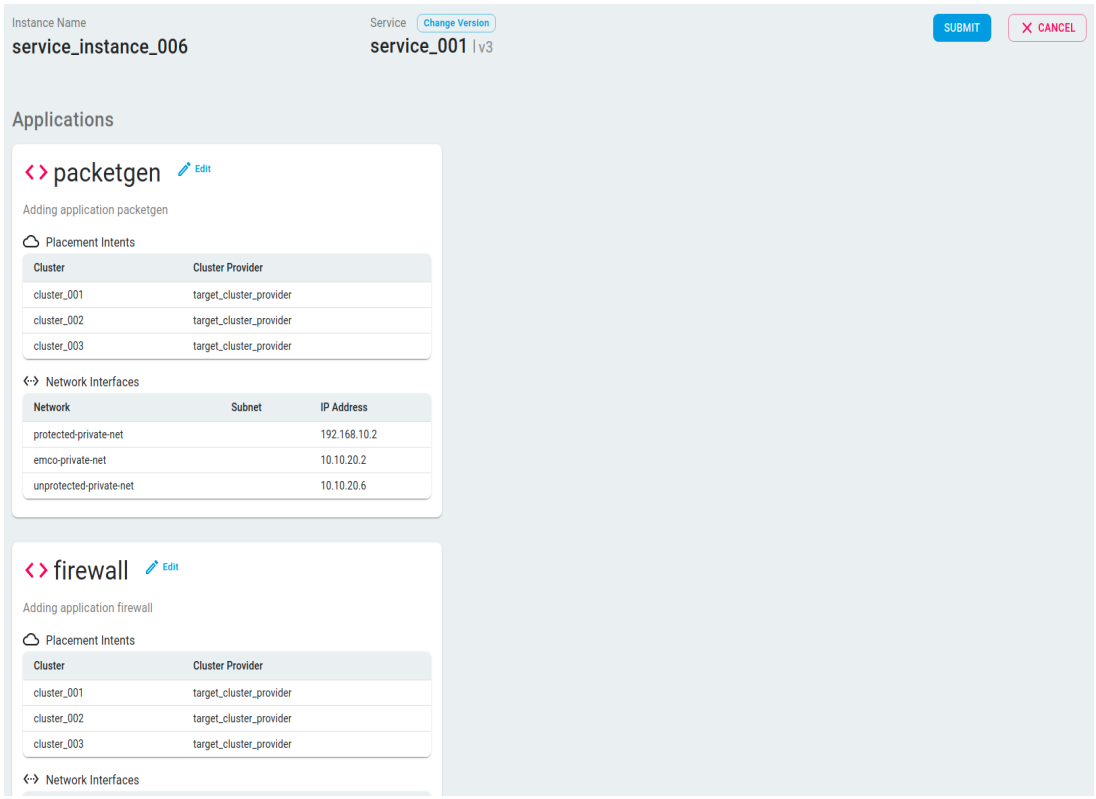

AMCOP v2.2
Dashboard
Services
Service Instances
Logical Clouds
Service Instances > Service Instance Detail
Instance Name
test_01
Service
Change Version
vFirewallService1 | v1
SUBMIT
CANCEL
Applications
firewall
Edit
firewall
Placement Intents
Label
Cluster Provider
clusterA
kubernetes
clusterB
kubernetes
sink
Edit
sink
Placement Intents
Label
Cluster Provider
clusterA
kubernetes
clusterB
kubernetes

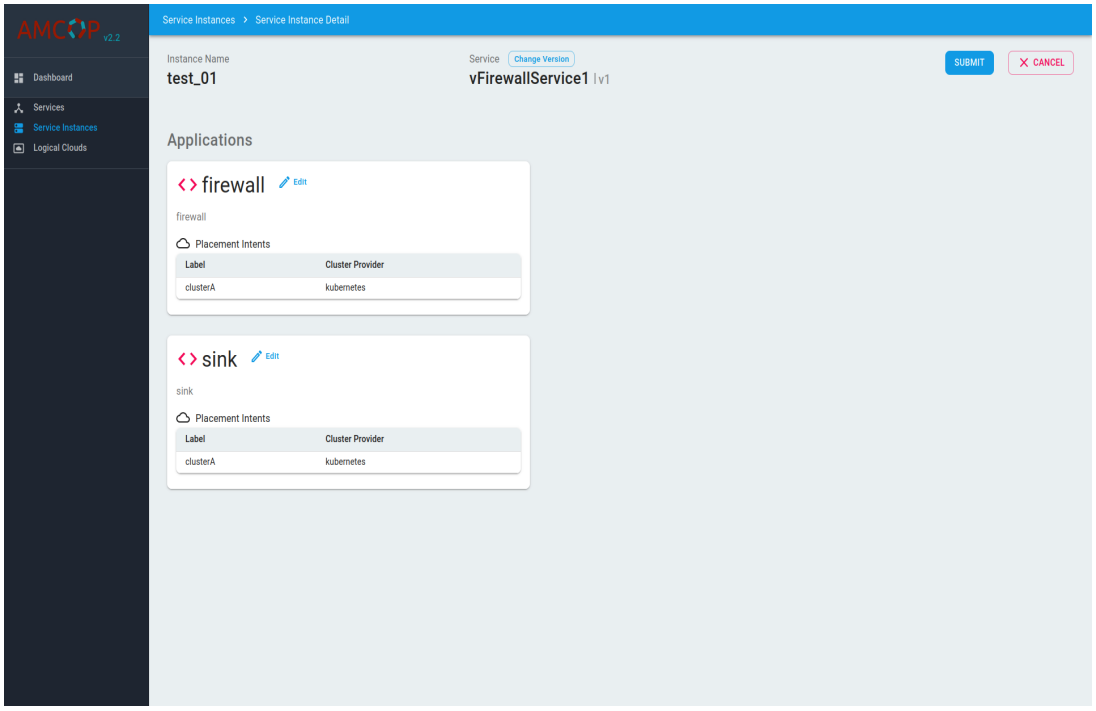
6. As part of the service instance update, Update placement intents of ***firewall*** and ***sink*** applications.

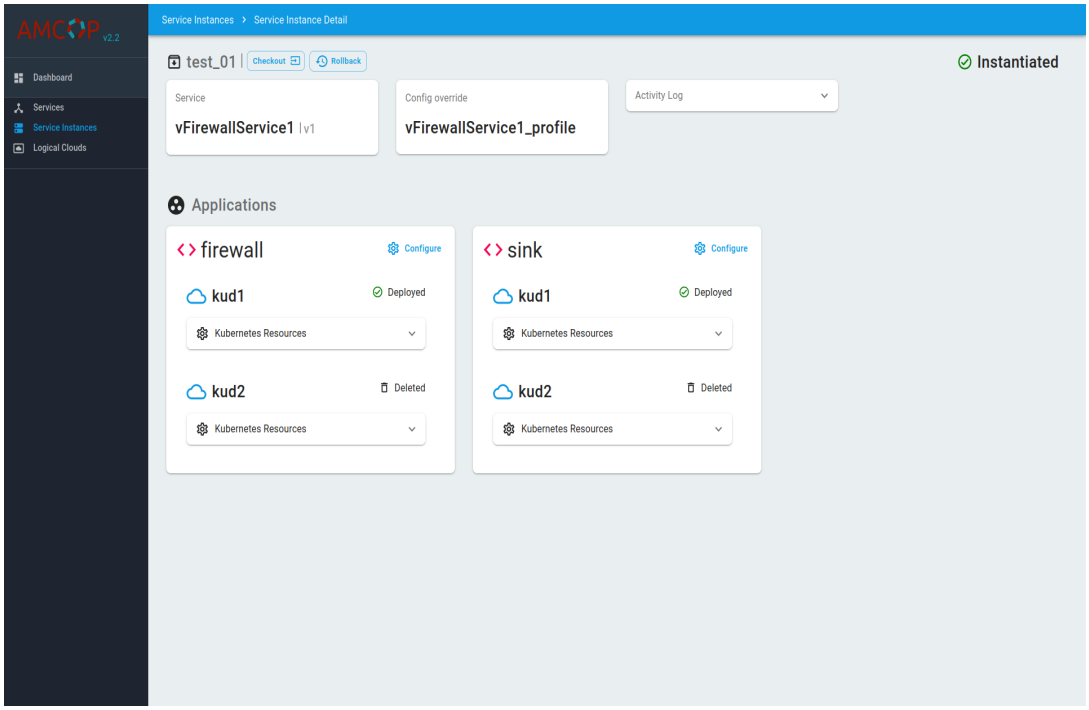




7. Click on the ***Submit*** button to ensure new placement intents are reflected on target clusters.


Instance Name
service_instance_006
Service
Change Version
service_001 | v3
SUBMIT
CANCEL
Applications
packetgen
Edit
Adding application packetgen
Placement Intents
Cluster
Cluster Provider
cluster_001
target_cluster_provider
cluster_002
target_cluster_provider
cluster_003
target_cluster_provider
Network Interfaces
Network
Subnet
IP Address
protected-private-net
192.168.10.2
emco-private-net
10.10.20.2
unprotected-private-net
10.10.20.6
firewall
Edit
Adding application firewall
Placement Intents
Cluster
Cluster Provider
cluster_001
target_cluster_provider
cluster_002
target_cluster_provider
cluster_003
target_cluster_provider
Network Interfaces


AMCOP v2.2
Dashboard
Services
Service Instances
Logical Clouds
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
sink
Edit
sink
Placement Intents
Label
Cluster Provider
clusterA
kubernetes

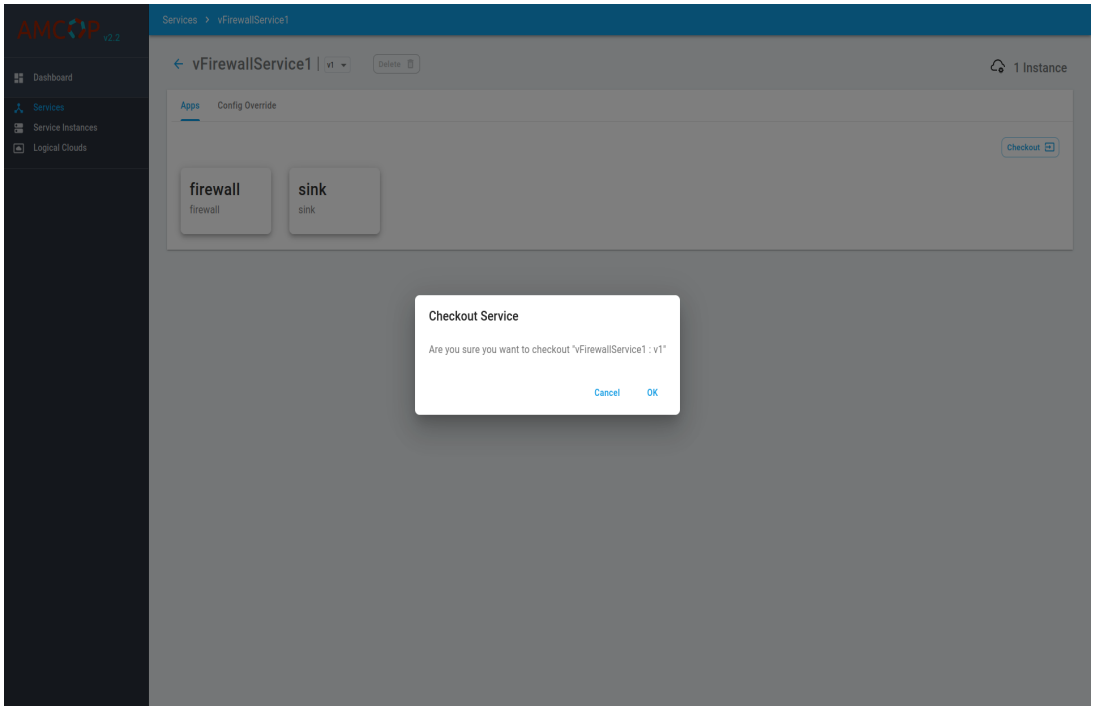


Service Instance Migration

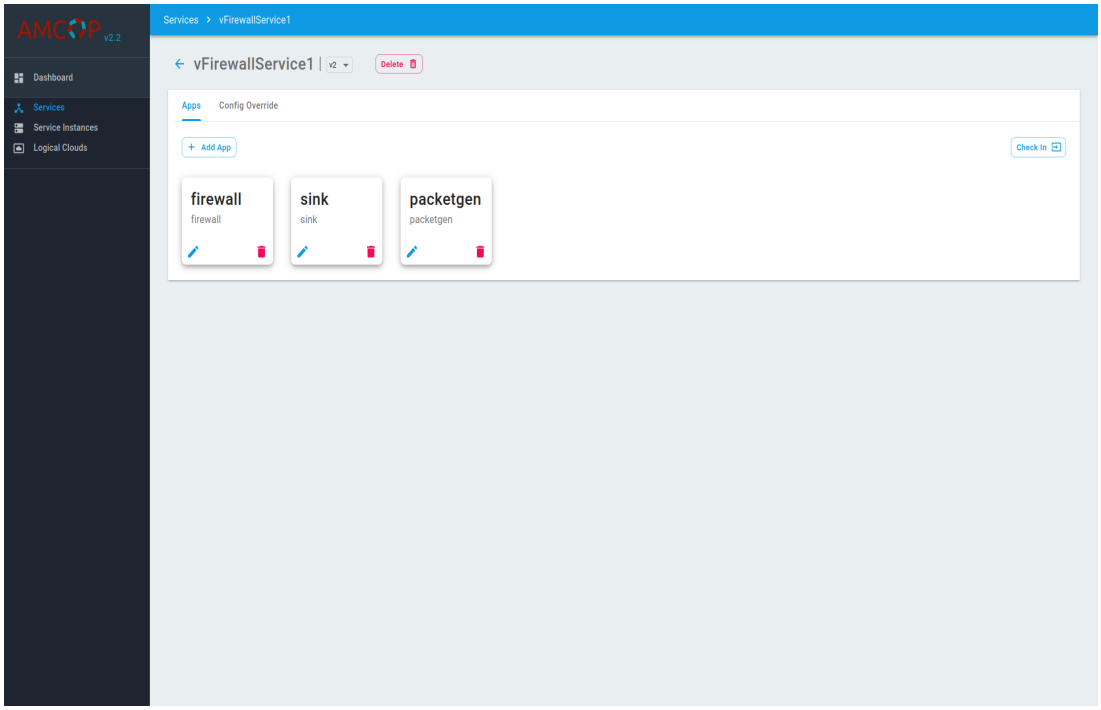
Service instance migration feature helps you to migrate to new versions (such as updated versions of the application helm charts, etc.,) or updated versions (such as adding a new application to the existing composite application) of the same service instance.

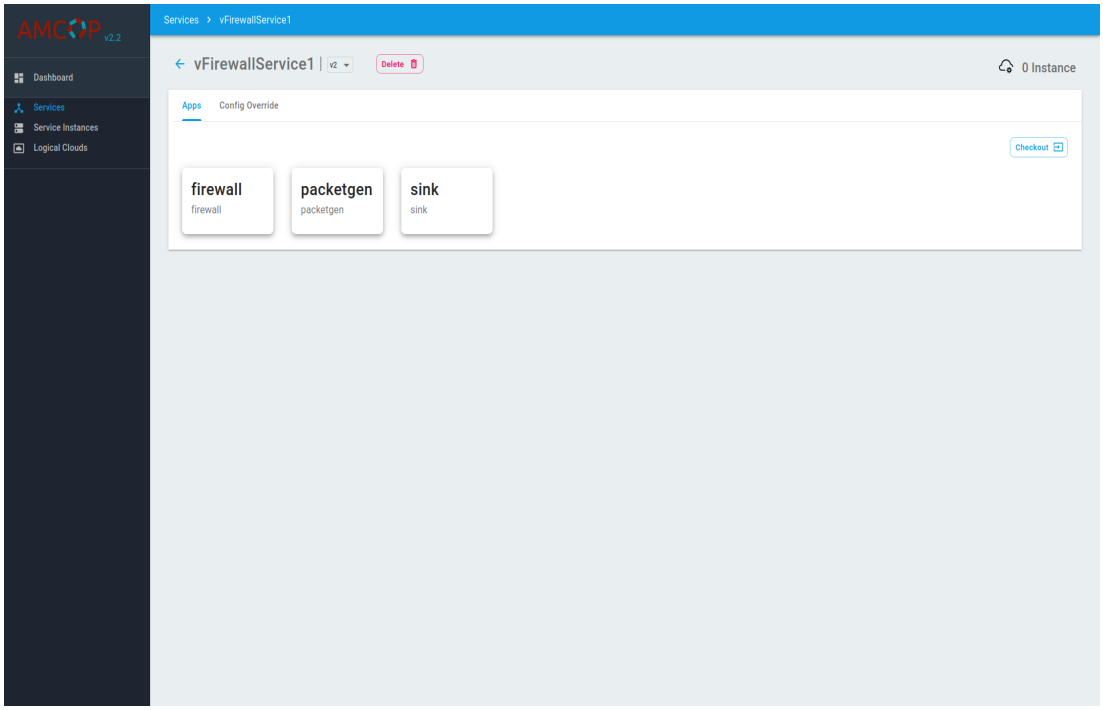
AMCOP supports service instance migration. For service instance migration, you need to first have a service (say vFW) with multiple versions as mentioned in previous sections of this document. You can follow the below steps to create multiple versions of vFW service.

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

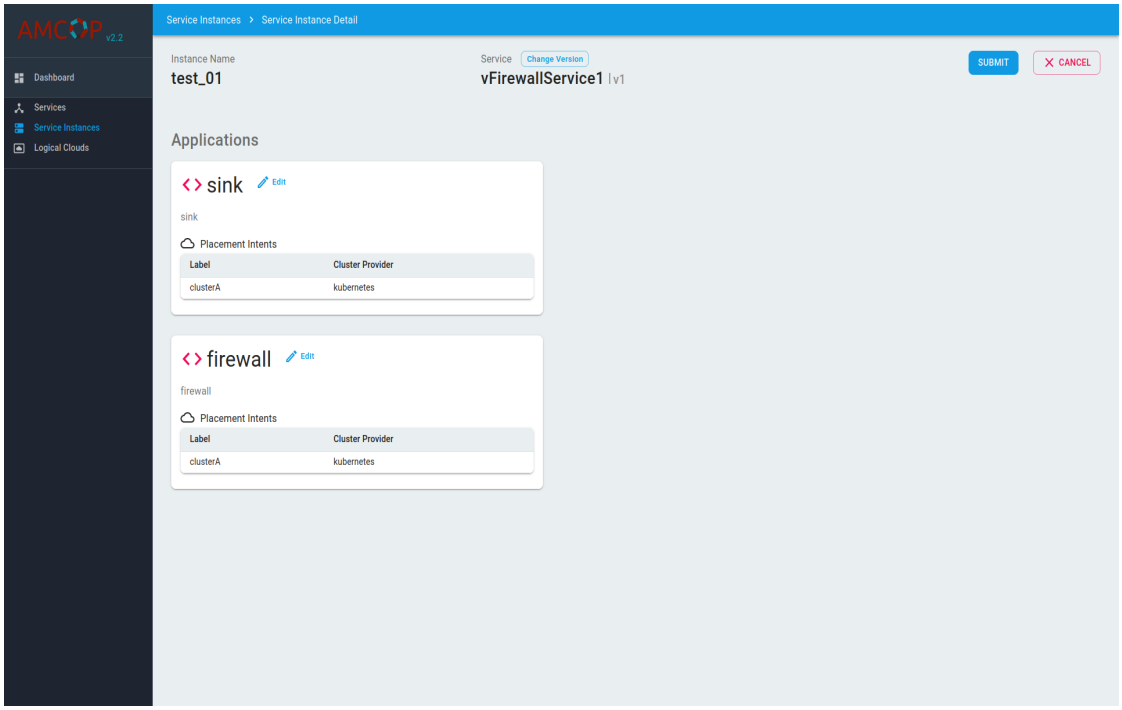


2. Create version v2 of ***vFirewallService1*** by adding a ***packetgen*** application to the existing ***firewall*** and ***sink*** applications.

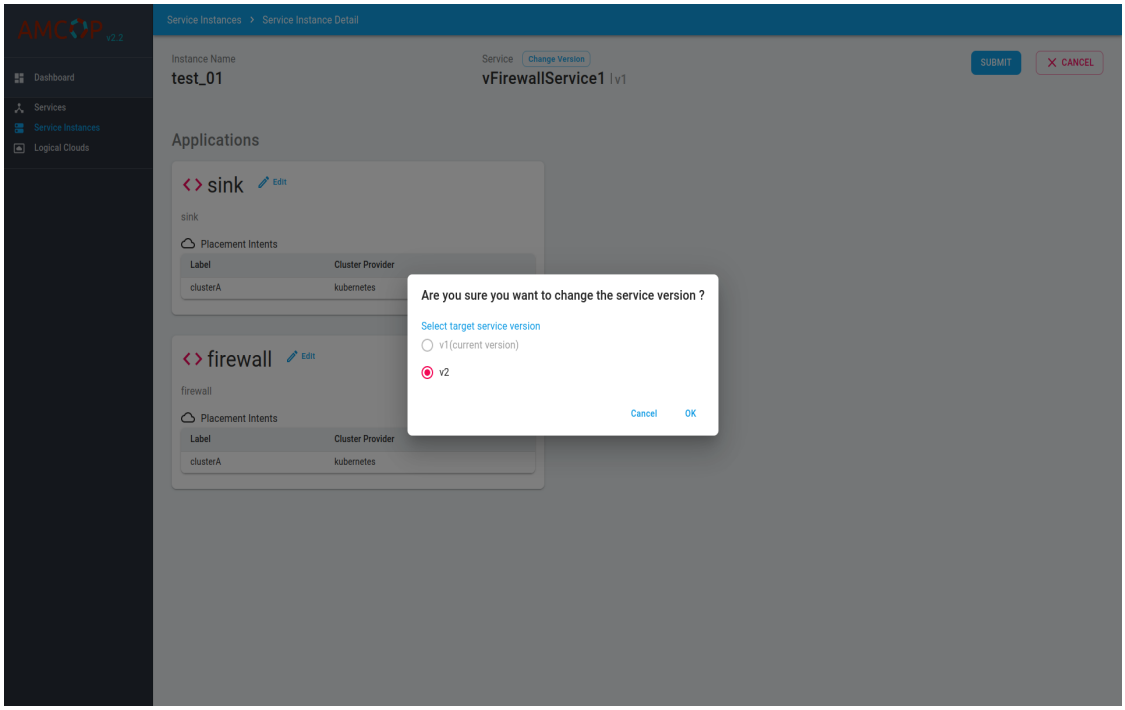




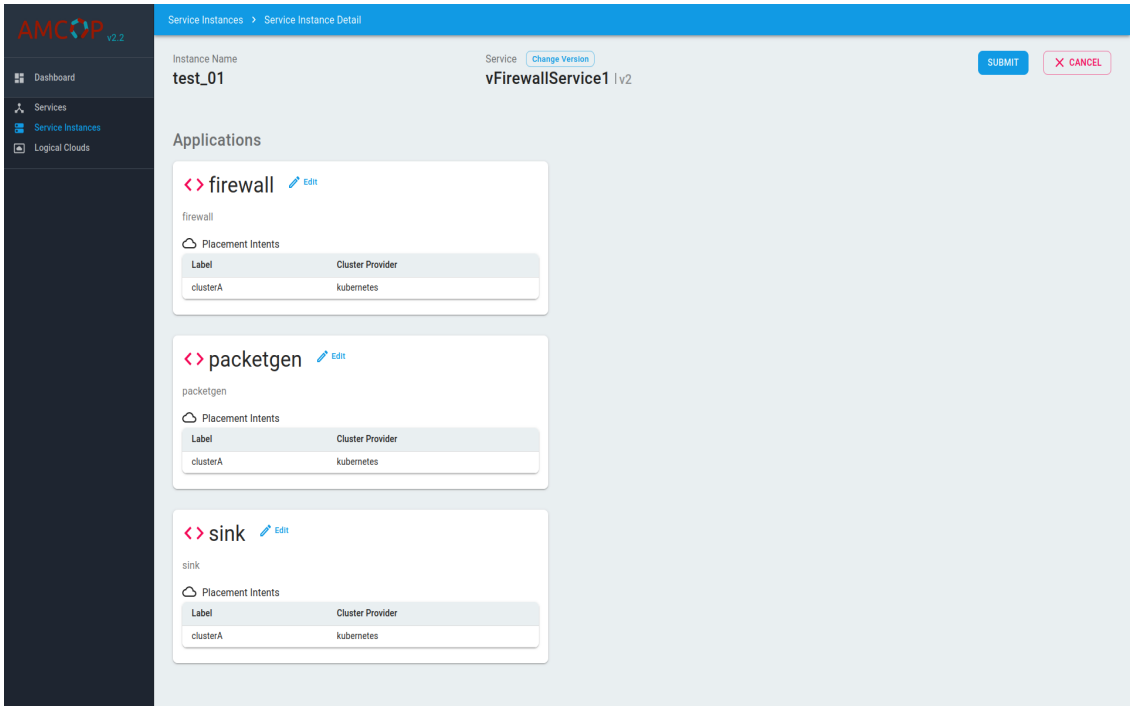
3. Select ***test_01*** service instance and perform a checkout.



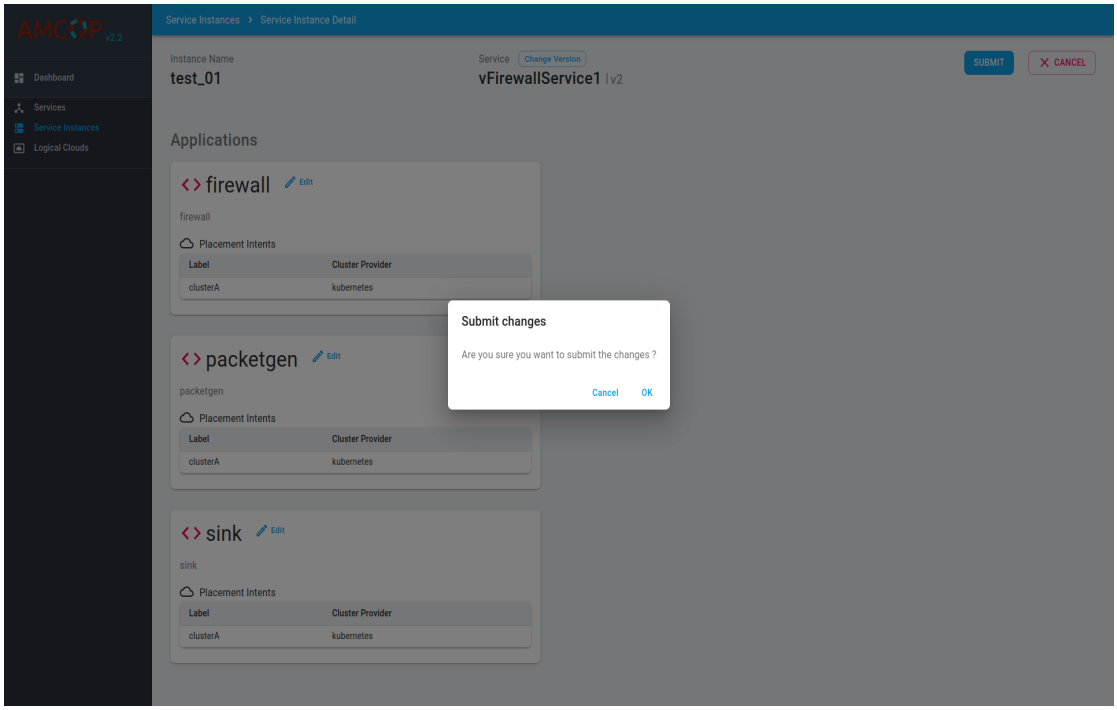
4, Click on the ***Change Version*** button for selecting version v2 of ***vFirewallService1*** to perform service instance migration. Then create a service instance, by selecting a version (say v2).

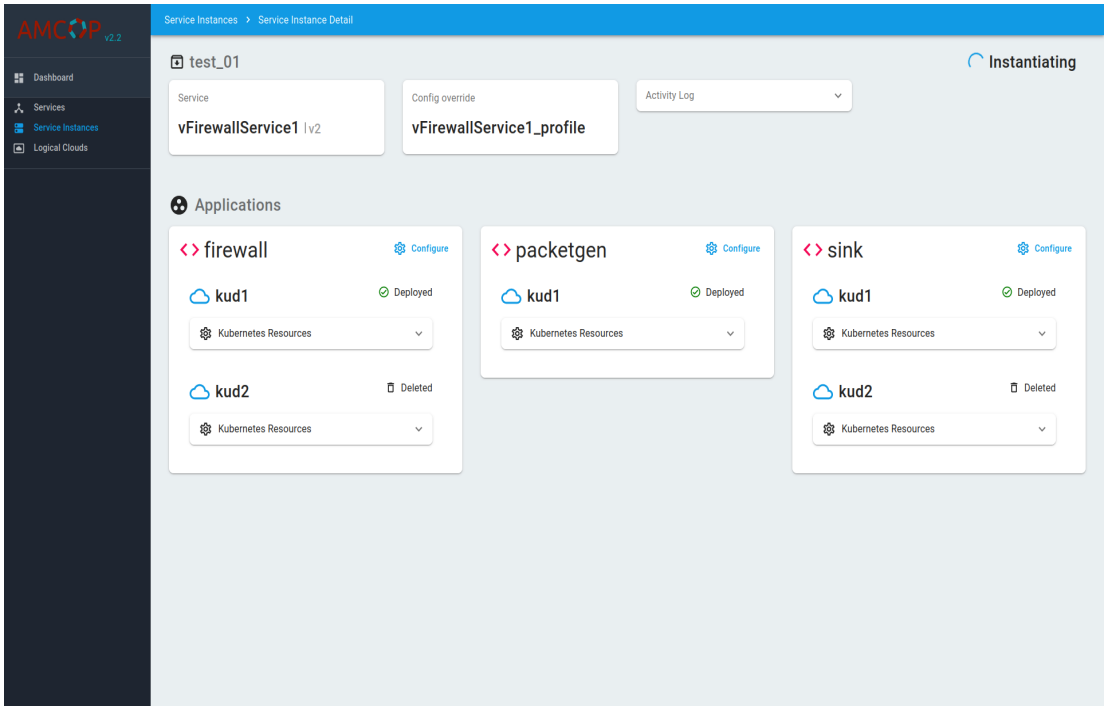


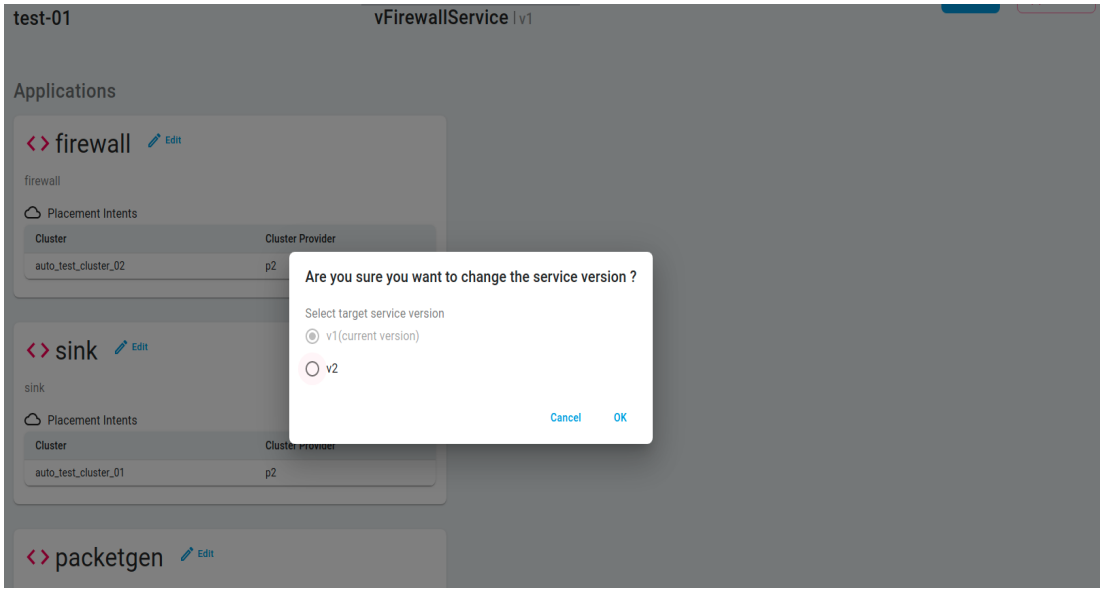
5. Select placement intent for ***packetgen*** application part of version v2 of ***vFirewallService1,*** Check out the service instance using a different version (say v2) for the service instance migration.



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







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.

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>:30480

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 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 gNb simulator environment

Admin User

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

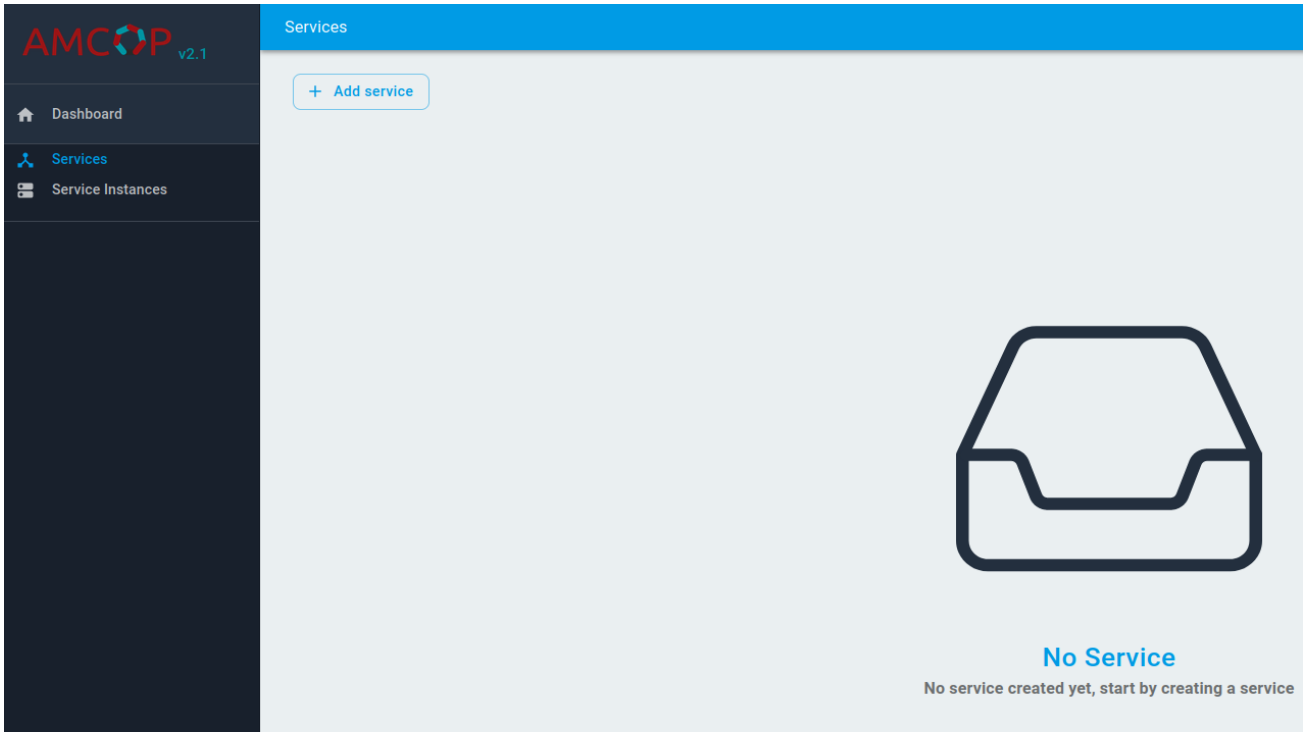
Refer to the steps mentioned in the section Orchestration of vFirewall using AMCOP GUI, to create Tenants, register controllers and onboard a cluster.

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

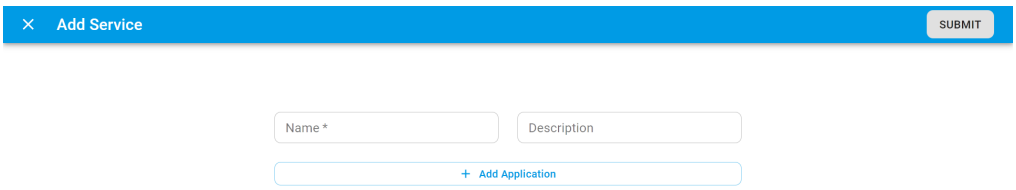
Note: Please make sure the rsync controller is registered, since this is necessary for Free5gc.

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.



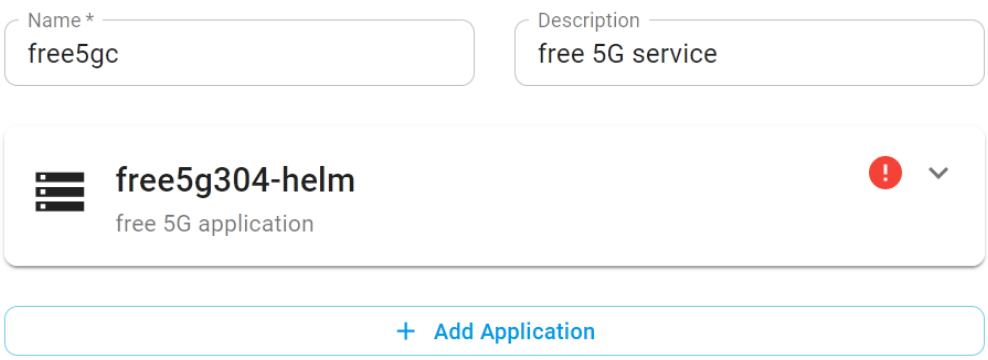
1. Add a Service
    a. Once the service designer view is opened, click on the *Add Service* button to add a new service.
    b. Fill in the basic information like name and description and then click on *Add Application* to add an application to the service.



    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 free5g304-helm** as the helm chart is free5g304-helm-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.

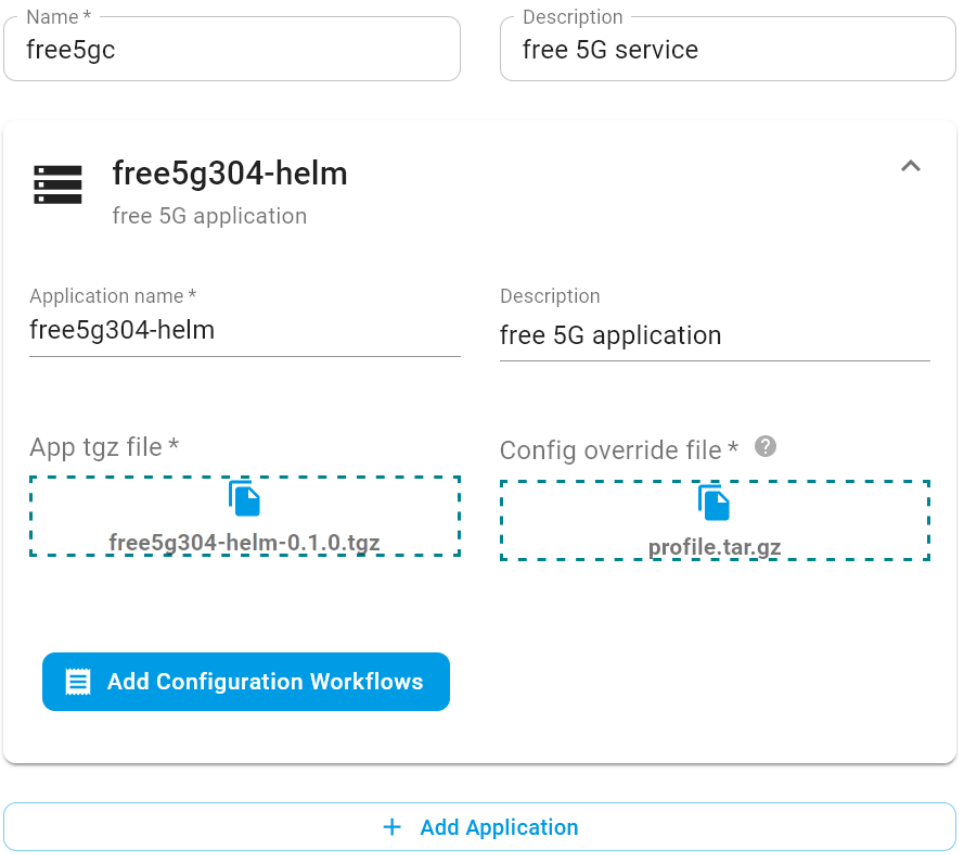


e. Click on the service row to expand it. Now upload the app .tgz package file and profile .gz file which contains the override files by clicking on the upload button or dragging and dropping in the upload area. App name and description can also be changed here.

   Note: The helm charts for the free5g are present in `/home/<user>/free5c_deploy` directory. The name of the bundle is free5g304-helm-0.1.0.tgz

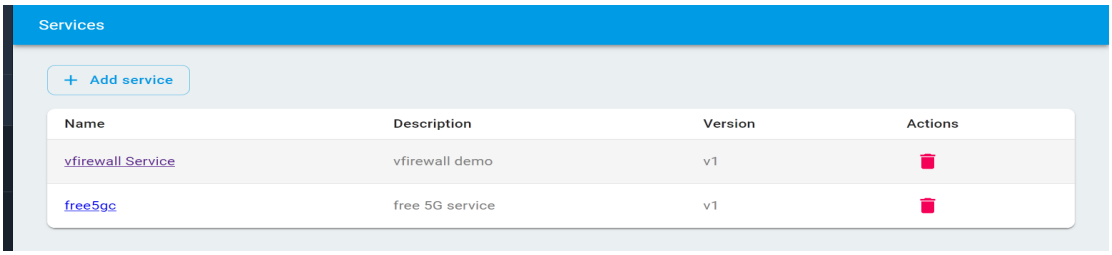
   Note: Before service design and instantiation of Free5GC, you need to set up the target environment as mentioned in the section Setting up Free5GC and gNb simulator environment

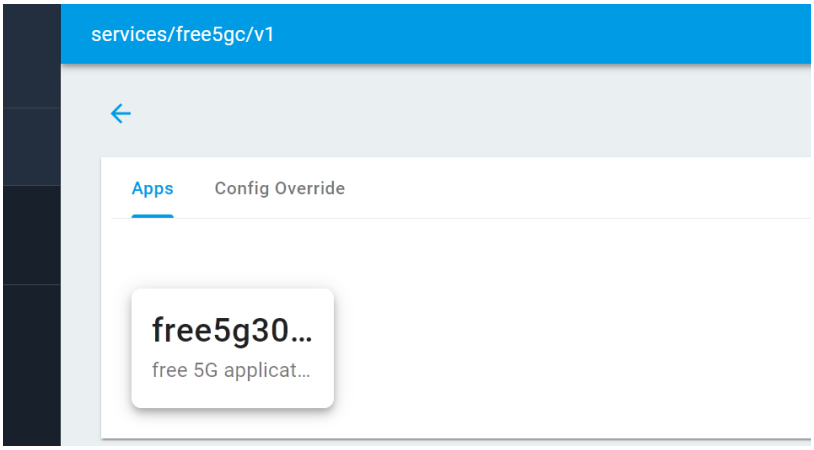
f. The profile bundle is present in `/home/<user>/free5c_deploy`. The name of the bundle is profile.tar.gz



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.

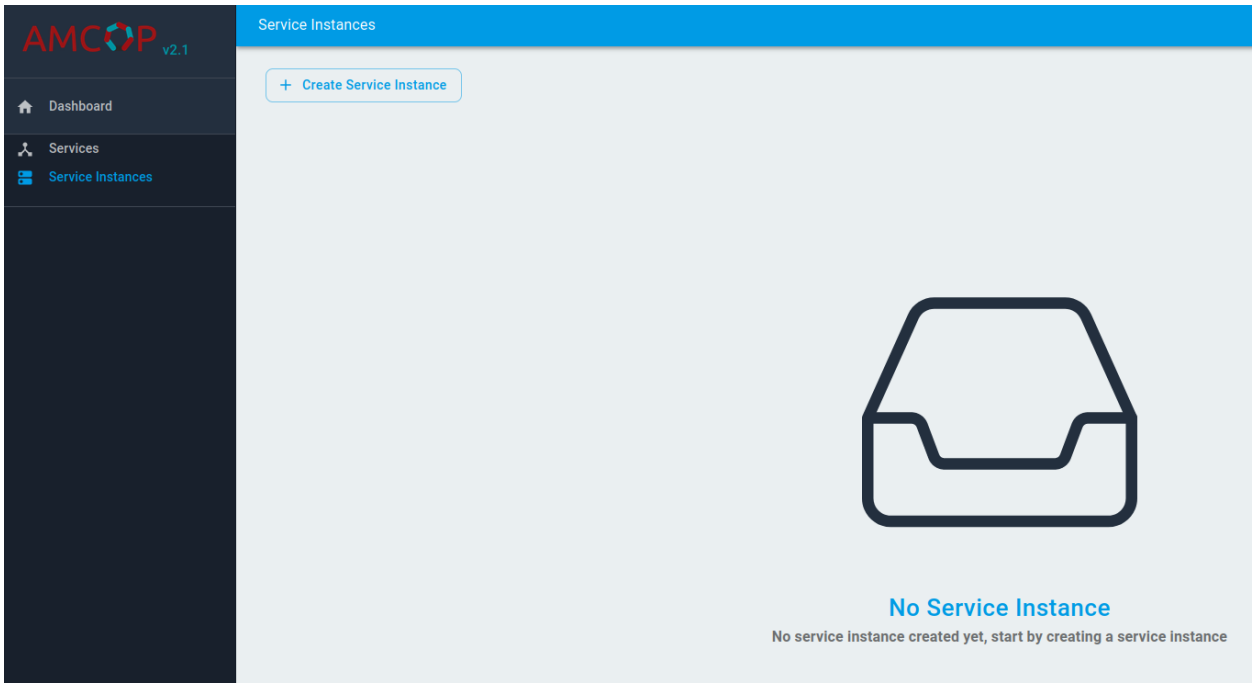
h. Now once the service is created, it will appear on the services page. We can look at the details of the service by clicking on the name of the service in the services table.



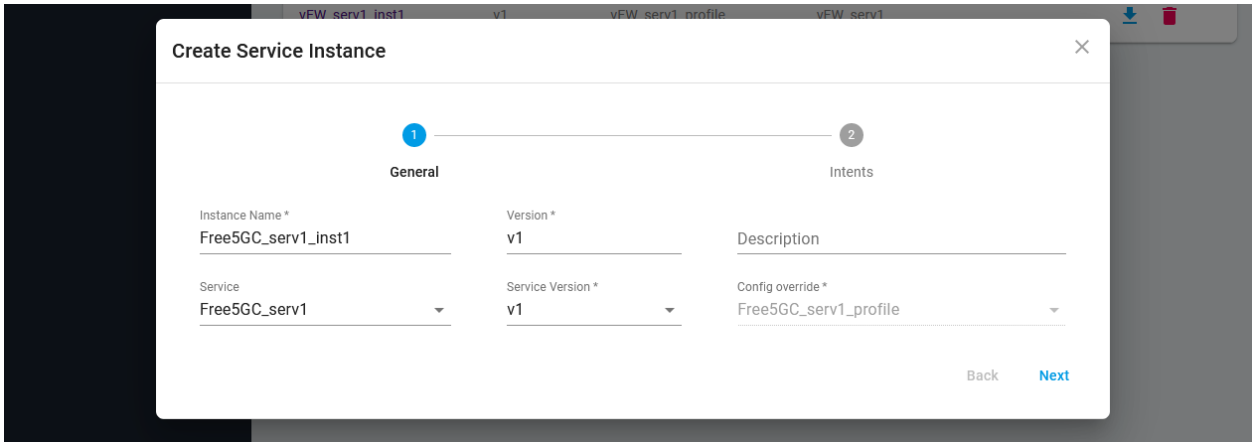


2. Create a service instance.

   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.

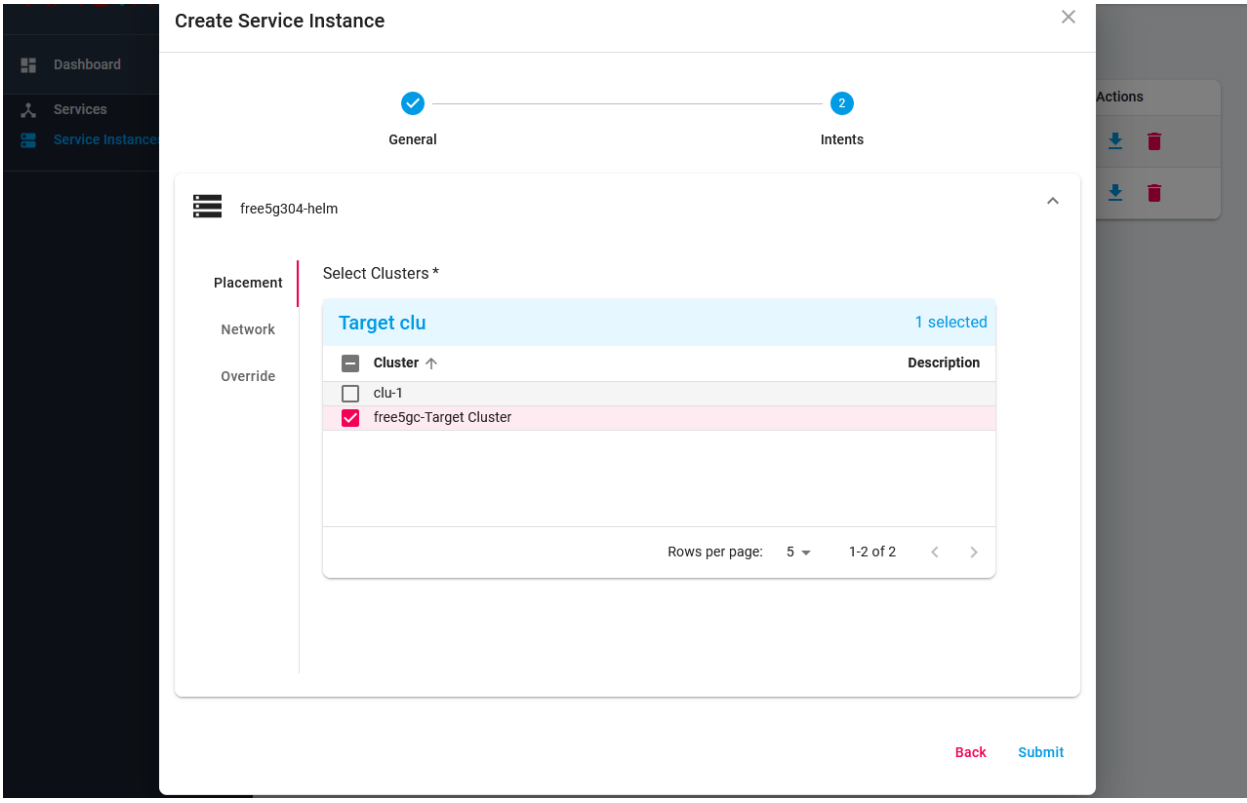


   b. In the service instance form, fill in the details like service instance name, version, description etc. Also select the service from the dropdown for which the instance needs to be created. In this case select free5gc Service

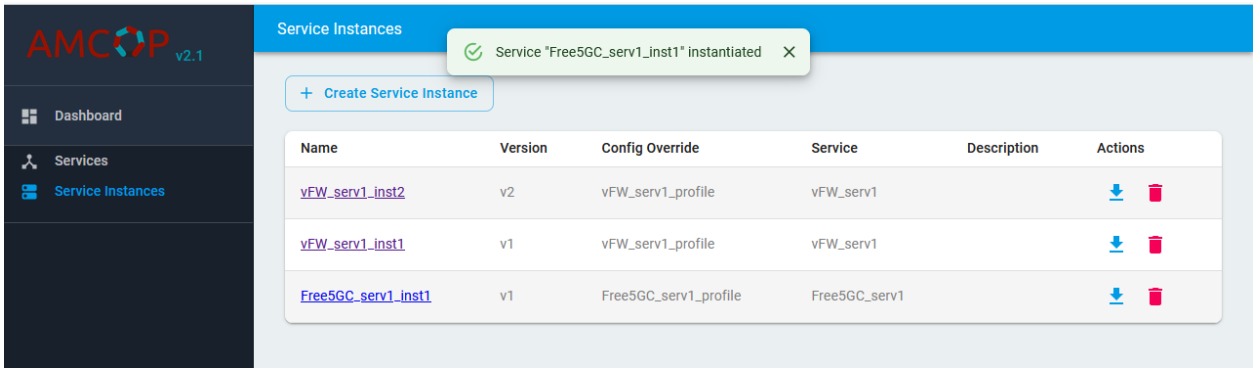


c. When a service is selected, its corresponding override file is automatically selected.
d. We can also provide override values if we need to override any value in the service instance at runtime. For free5gc, leave it blank.
e. Once all the required fields are filled, click on next.
f. Now you will 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
```

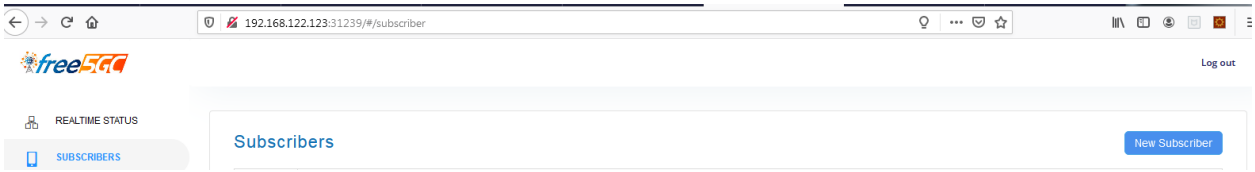
Free5GC Validation using gNb Simulator

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:

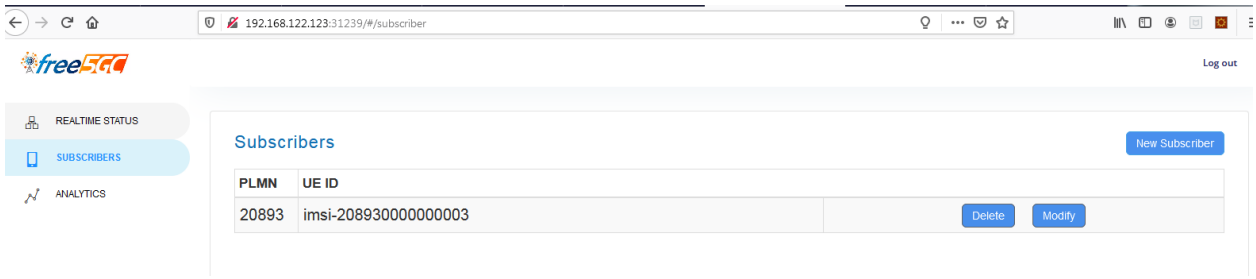
1. Login to the webui. Run the below command to identify the port number. The highlighted port is the webui port

   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. Go to **$HOME/gnbsim/example** directory to edit the UE details
6. Open **example.json** file and edit the following properties
    a. imeisv : replace the value with UE ID created above.
       Eg: "imeisv": "208930000000003",
    b. Msin: replace it with the last 10 digit value of the UE ID
       Eg: "msin": "0000000003",
    c. GTPuAddr: replace it with gnbsim VM_IP (VM where gnbsim is checked out)
       Eg: "GTPuAddr": "192.168.122.123",
    d. GTPuIFname: replace it with the interface name
       Eg: "GTPuIFname": "ens3"

    Save file and exit.
7. Rename example.go file as follows
    a. cd gnbsim/example
    b. Rename example.go file to example.go.bk
8. Copy /home/<user>/free5c_deploy/example.go file (on deployment host) to $HOME/gnbsim/example directory of the current VM.
9. Run “make”
10. Run the simulator with the AMF IP. To identify AMF IP, login to AMF pod and run “IP a” command. For gnbsimulator, 172.16.10.20 is the IP of AMF

    sudo ./example -ip 172.16.10.20 -test registerUE
    sudo ./example -ip 172.16.10.20 -test setupPDUSession -gtp gtp0

11. To run the setupPDUSession multiple times, increment the gtp parameter value from gtp0 to 1...n
12. Simulator logs (terminal console logs) will have the output of the above execution.
13. AMF, AUSF, SMF and UPF logs will have more information.

The simulator will do the following

1. NGSetup: Perform the handshake between the 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.

AMCOP 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 30481 30415 30461 30477
```

- 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
```

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
```

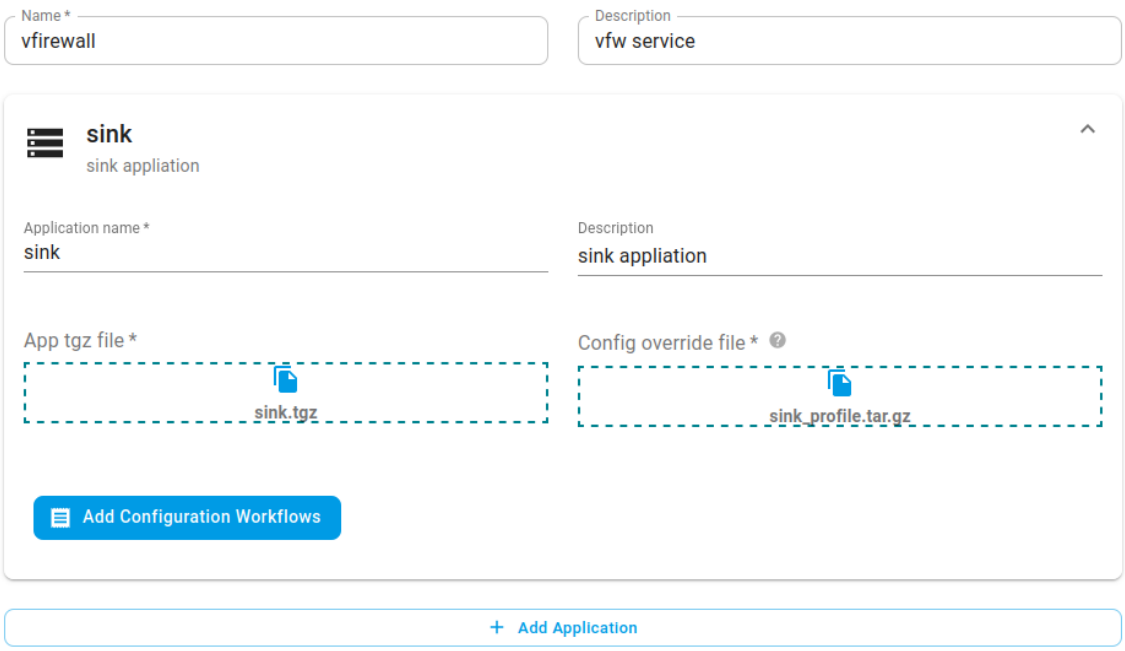
With this profile the value replicaCount in the chart Values.yaml will be overridden with 4.

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

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.

```
cd ~/aarna-stream/cds-blueprints/k8s-utility-scripts/
bash -x ./deploy-cba.sh <CBA_FOLDER>
```

For example, in case of vfw_netconf CBA:

```
cd ~/aarna-stream/cds-blueprints/vfw_netconf/Templates
vi stream-count-config-edit-schema-template.vtl

# Replace "localhost" with "<Target Cluster IP>"
# where vPG will be deployed.

cd ~/aarna-stream/cds-blueprints/k8s-utility-scripts/
bash -x ./deploy-cba.sh vfw_netconf
```

Note: The <CBA_FOLDER> for vfw_netconf is present under: "~/aarna-stream/cds-blueprints". For any other CBAs, users need to copy the CBA folder under: "~/aarna-stream/cds-blueprints" after logging into 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.

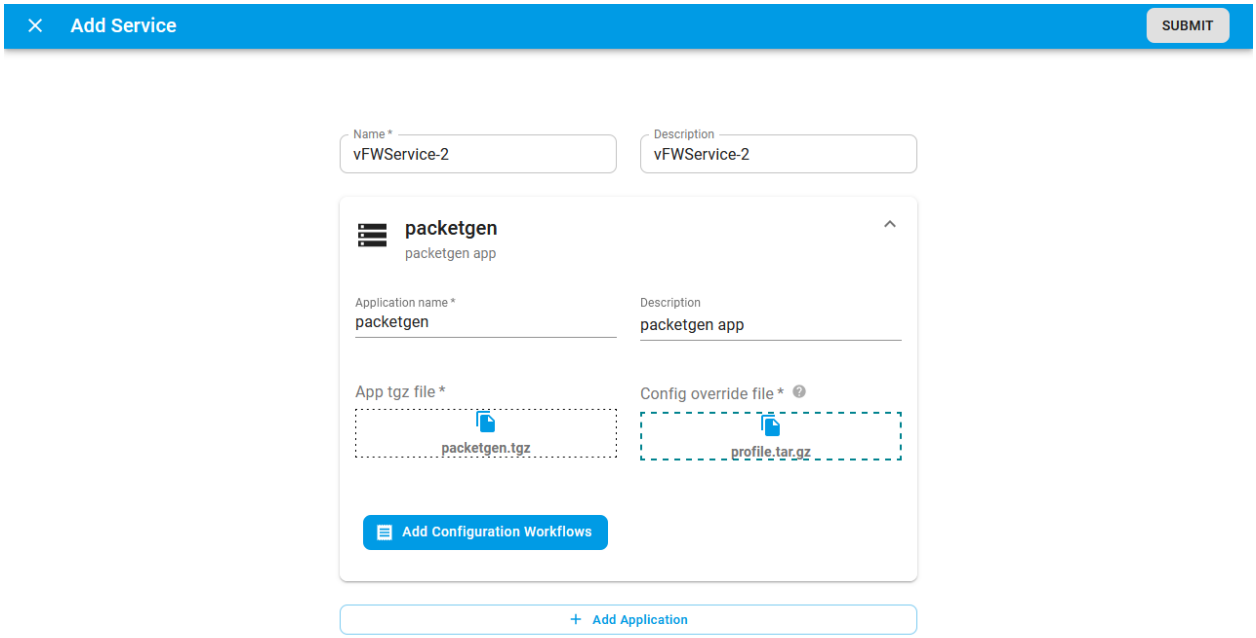
Configuration using AMCOP GUI

Design time

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

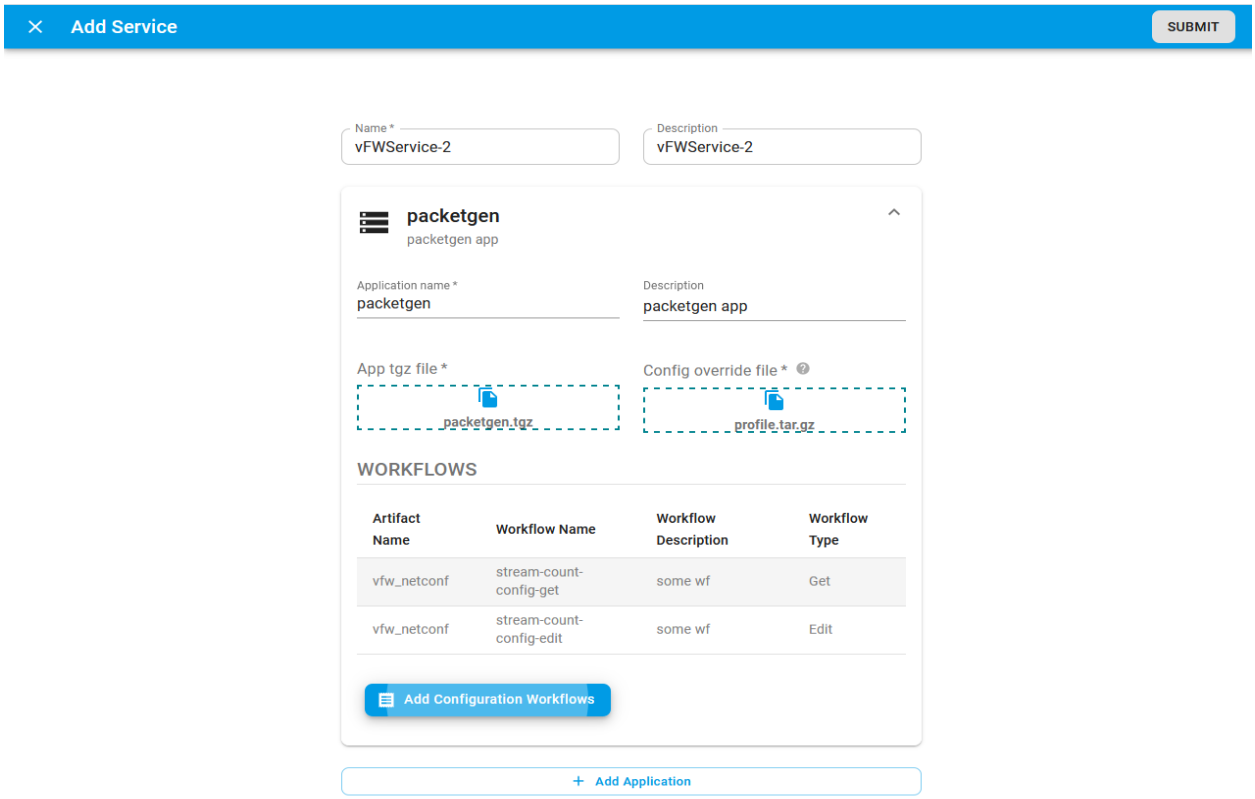
Click on *Add Configuration Workflows* in the app form. This shows all the workflows that have been onboarded to AMCOP through the corresponding CBAs. Then select the workflows which need to be tagged with the application. Select the *packetgen*app under the deployed vFW service and then select "*Add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

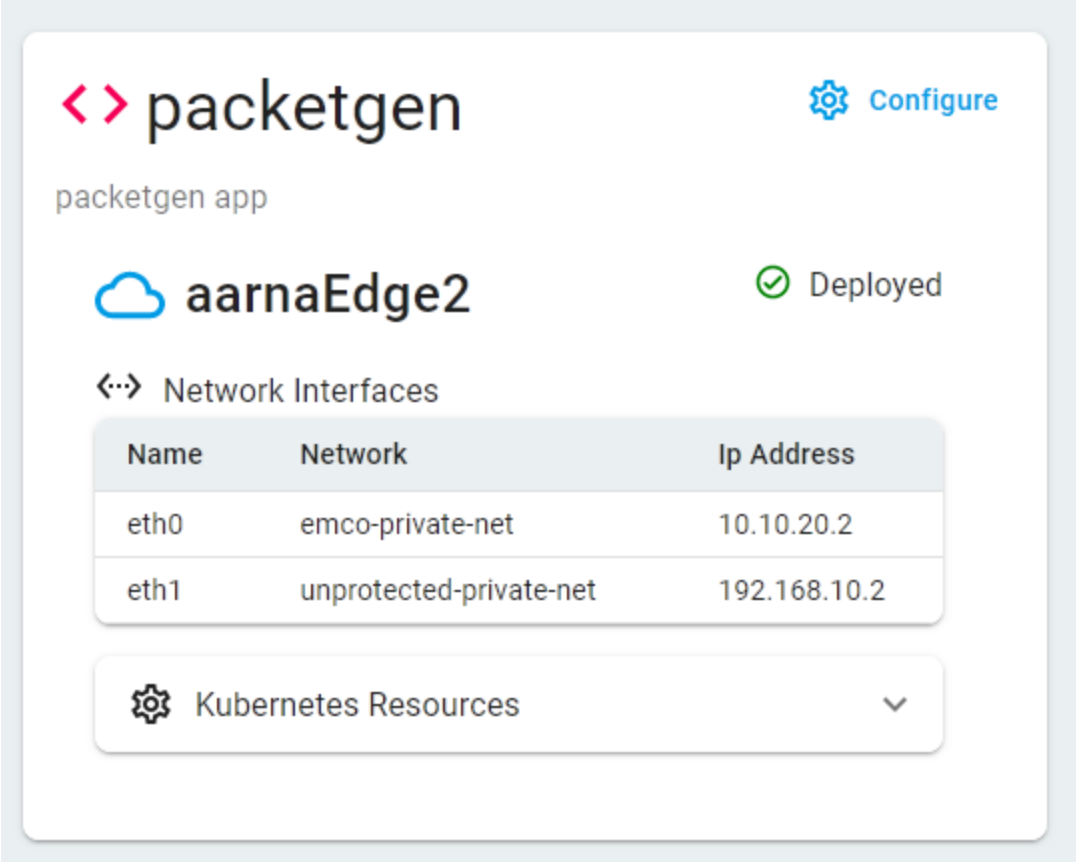
Note: Users need to wait for ~15 mins before the packetgen app cloud init to complete. Then the user can perform the Config GET workflow.

Run time

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

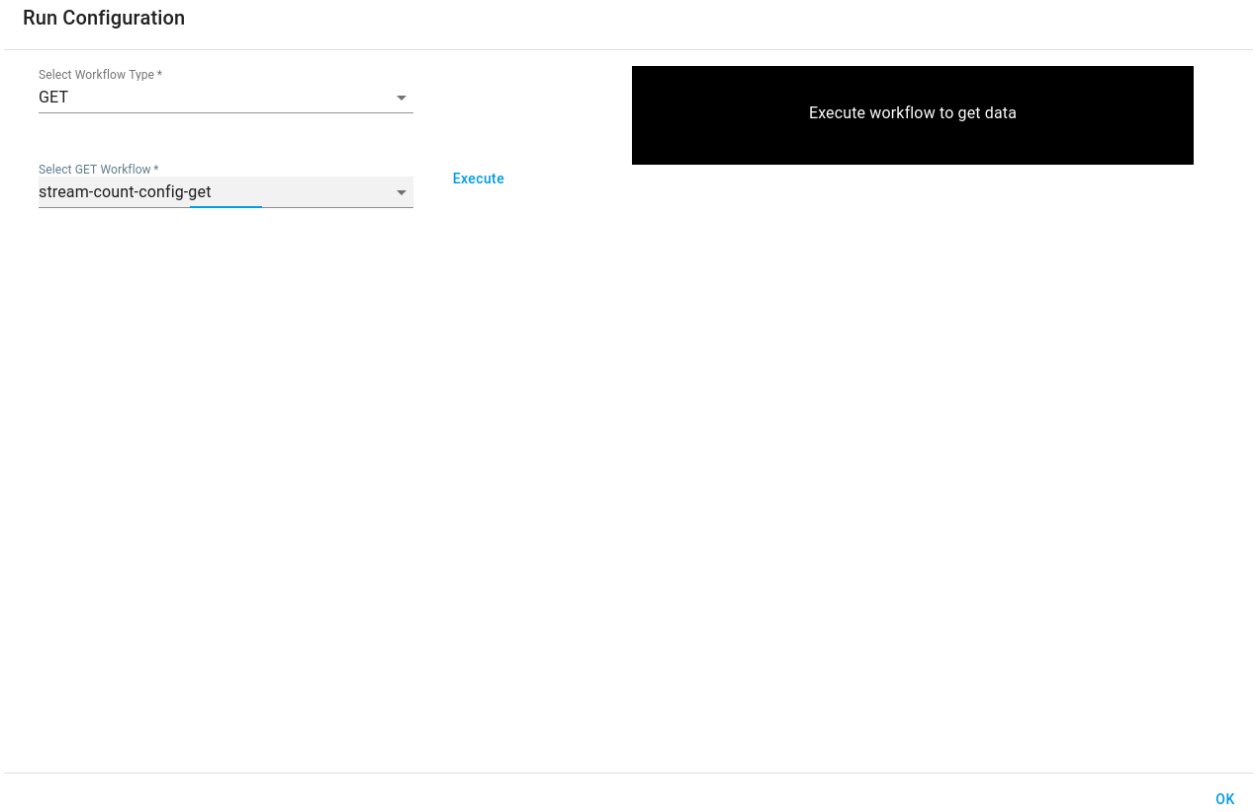
To configure an application, go to service details by clicking on the service instance name.

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.

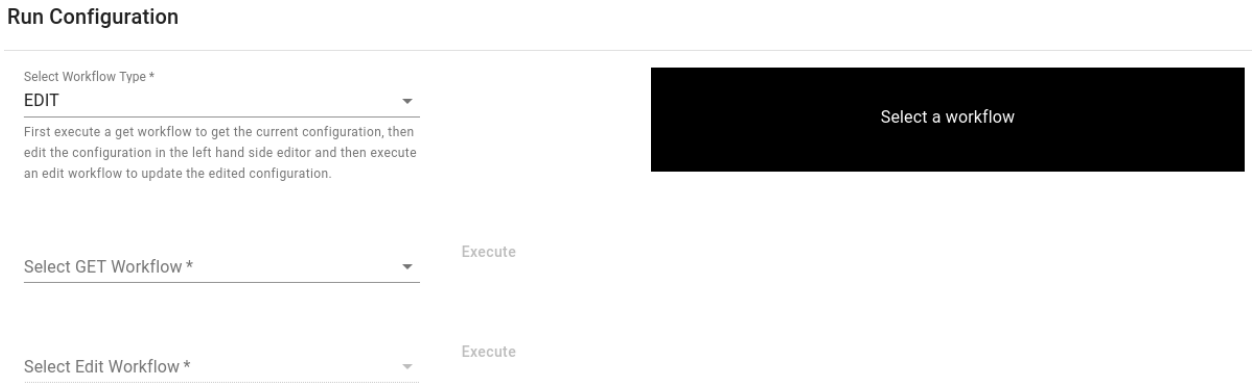


Clicking execute will trigger the workflow. In this example, we are triggering the CONFIG GET workflow, so as expected we will see the running configuration of the packetgen. We are fetching the pg streams config of the packet generator.

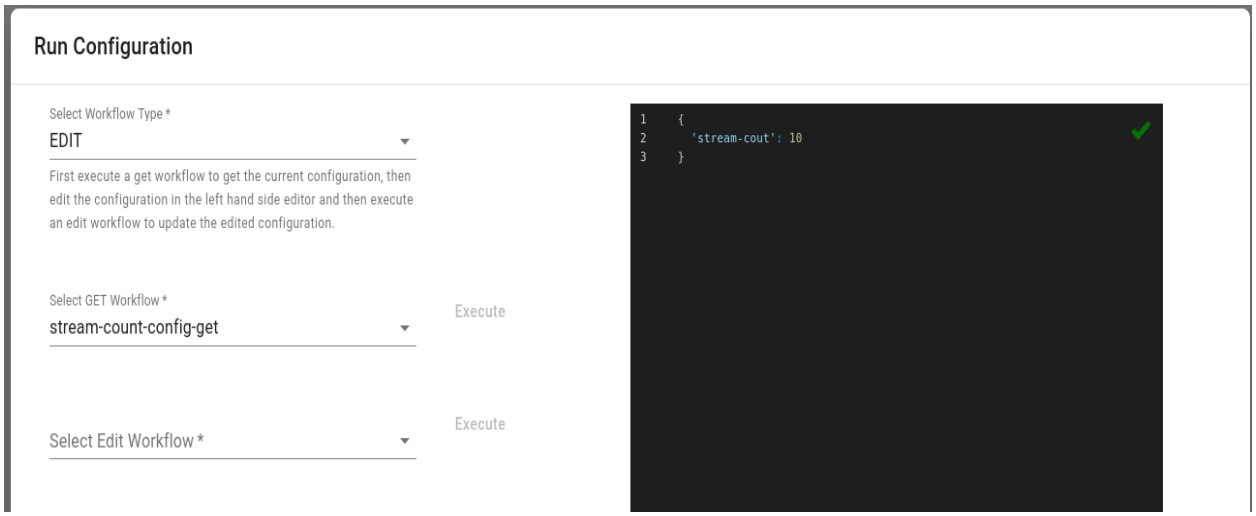
Config EDIT Workflow

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,

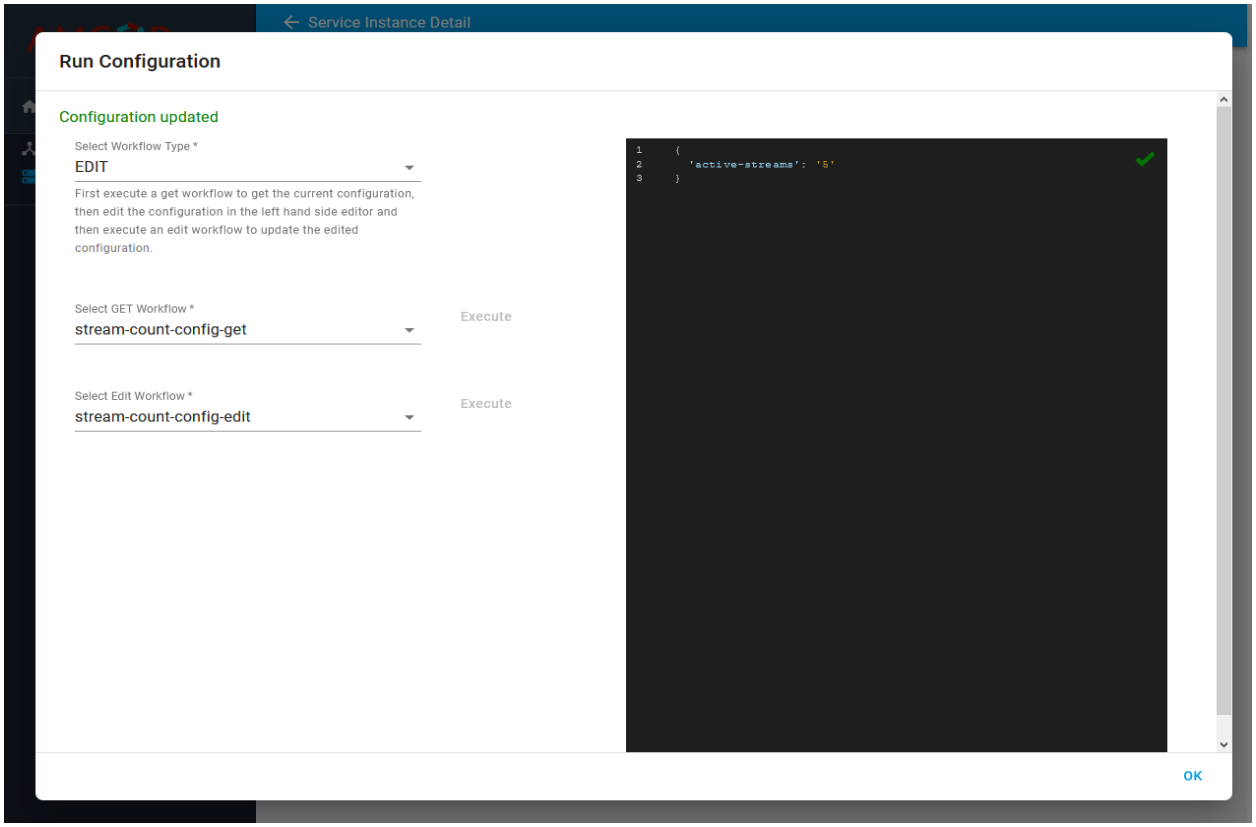
Select EDIT, it will show the options as shown in the image,



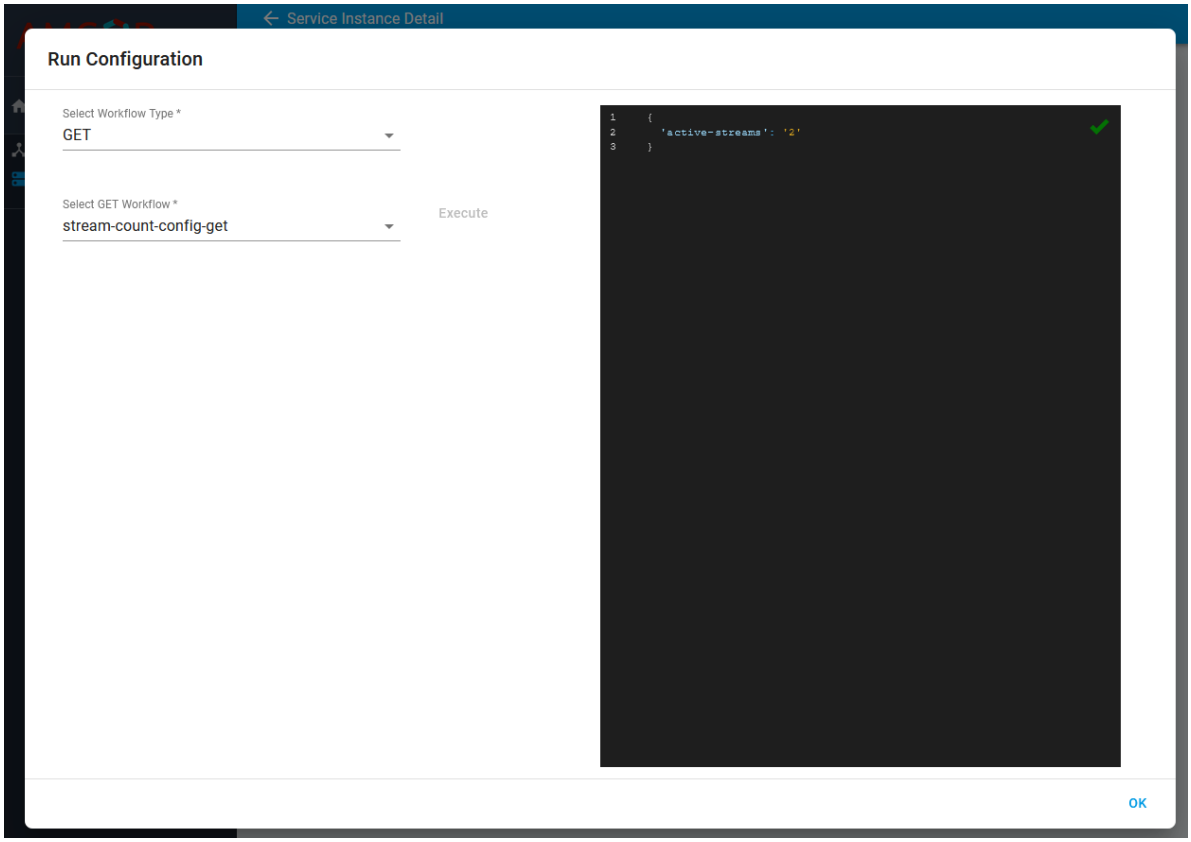
Select the GET, and execute to get the current config, “**stream-count”: 10** in this example



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.
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.

Develop CDAP Analytics application

The CDAP applications (based on CDAP version 5.x) can be onboarded using either the CDAP GUI or CLI (CDAP REST APIs). This section describes the components of the CDAP application, and a link to the sample application code is provided for reference.

You can download the complete application from the following link, and make the necessary changes for your use case.

https://drive.google.com/file/d/1TgyeGLFdiskcg2NUJj3x3KA82x62b2eW/view?usp=sharing

AbstractApplication Class

This is the support class from CDAP, using which you can implement the configue() method to define the analytics application. In the configure() method, initialize the CDAP application

components that are required by the application and configure them with the application load time configuration.

The components of the CDAP application are:

AppConfig: Call the method getConfig() in order to get the configuration provided during the application creation. Use this information for configuring the subsequent components.

- DataSets: These are the hive tables that will be created by the CDAP.

```
// ========== Dataset Setup
final DatasetProperties vesEventTableProperties = getDatasetProperties(
                VESEvent.class,
                "VES Event Table",
                appConfig.getMessageStatusTableTTLSeconds());
createDataset(appConfig.getEventTableName(), ObjectMappedTable.class,
vesEventTableProperties);
        LOG.info("Creating Table: {} with TTL:
{}",appConfig.getEventTableName(),
appConfig.getMessageStatusTableTTLSeconds());
```

- Streams: These are the primary means for pushing the data into the CDAP system. For AMCOP analytics use cases, the streams usually connect the DMAAP(Kafka) topics with the Application flowlets which process the data.

```
final Stream subscriberOutputStream = new
Stream(subscriberOutputStreamName,
subscriberOutputStreamDescription);
         addStream(subscriberOutputStream);
```

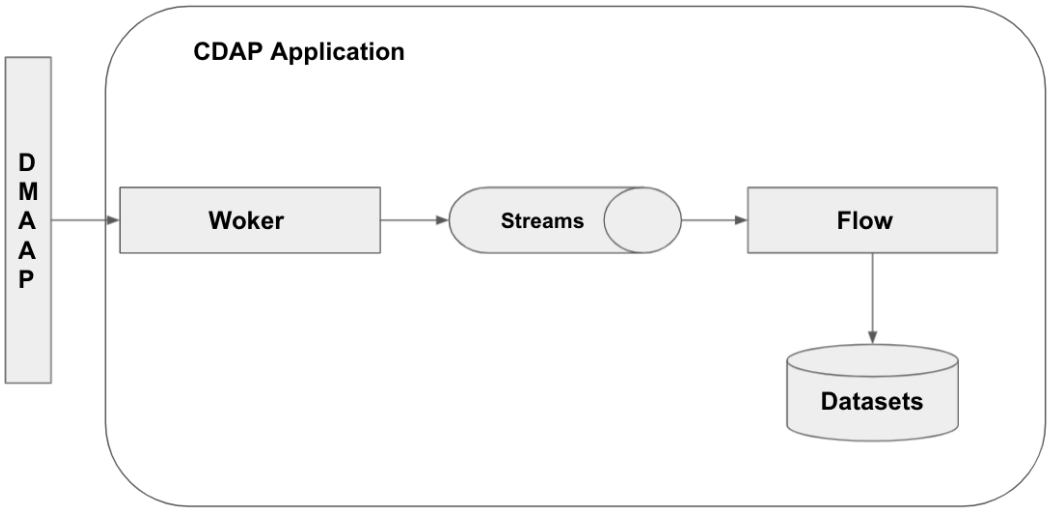
- Worker: A worker node in the application, in the sample application a worker has been created for polling the messages from DMAAP and passing them on the Stream.

```
 // ========= Workers Setup
        LOG.info("creating DMAAP Subscriber Worker");
        addWorker(new
KafkaSubscriberWorker(appConfig.getSubscriberOutputStreamName()));
```

- Flows: A flow defines the data flow in the Analytics application. It enables the application developers to create and define flowlets and connect them in order to define the data flow.

```
// ========= Flow Setup
         addFlow(new VesCollectorFlow(appConfig));
```

At a high level the confure() method creates/configures a CDAP as shown in the diagram below,



AbstractFlow Class

This is the support class from CDAP, which enables you to add Flowlets and connect them in order to define the data flow of the application. The Flow is explained in the above section, which is created in the configure() method of AbstractApplication Class.

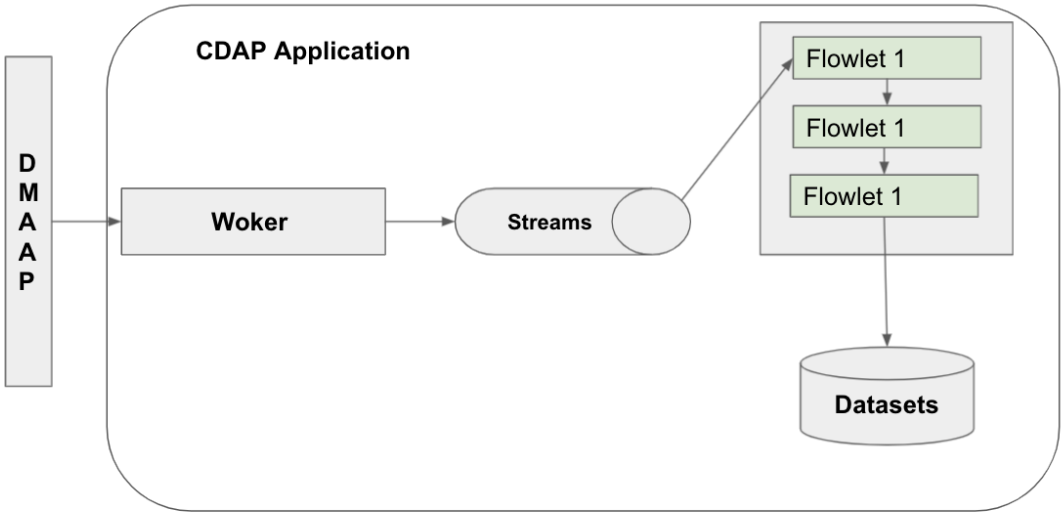
- Flowlet: The flowlets parse, analyze and emit the processed data and one or all of the following based on the required application logic,
    - Emit processed data to the next flowlet in the data flow.
    - Store data in the Dataset
    - ***Take a service auto corrective action*** which includes either calling the REST API of CDS or an external application. Such action concludes the Close loop.
    - The CDS or external application endpoints are configured through the application preference json, which is uploaded during application instantiation.

```
final AlertsActionFlowlet alertsActionFlowlet = new
AlertsActionFlowlet();
addFlowlet(alertsActionFlowlet);
```

- Connect: This method connects the flowlets, or streams to flowlets.

```
connectStream(appConfig.getSubscriberOutputStreamName(),
messageRouterFlowlet);
connect(messageRouterFlowlet, vesEventProcessorFlowlet);
```

With the flowlets in place, the application is complete,



CDAP Application Deployment

Following are the steps and artifacts needed for deploying an analytics application in AMCOP.

- Set the environment variables

```
cd ~/aarna-stream/cdap_application/

CDAP_HOST=${CDAP_HOST:-"<AMCOP VM IP address>:31015"}
```

- Create namespace

```
curl -X PUT http://$CDAP_HOST/v3/namespaces/cdap_closedloop
```

- Load artifact

```
curl -X POST --data-binary @CDAPAnalyticApplication-1.0.jar
http://$CDAP_HOST/v3/namespaces/cdap_closedloop/artifacts/CDAPAnalyticAp
plication
```

- Create application

```
curl -X PUT -d @app_config.json
http://$CDAP_HOST/v3/namespaces/cdap_closedloop/apps/AnalyticApplication
```

Format of app_config.json

```
{ "artifact": { "name": "CDAPAnalyticApplication", "version": "1.0.0",
"scope": "USER" } }
```

- Load app_preferences : Application preferences are the place for specifying the policies and threshold values (in case of TCA). These preferences can be modified during the runtime either through the CDAP GUI or CLI (CDAP REST API)

```
curl -X PUT -d @app_preferences.json
http://$CDAP_HOST/v3/namespaces/cdap_closedloop/apps/AnalyticApplication
/preferences
```

Format of app_preferences.json

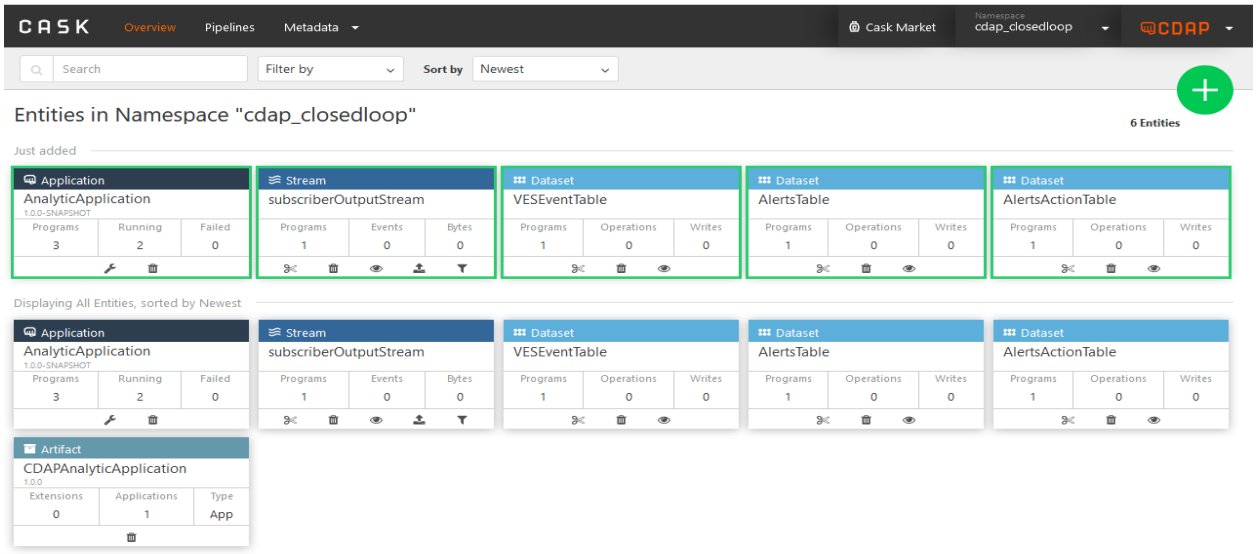
```
{ "subscriberHosts":"kafka1.onap4k8s.svc.cluster.local:9092",
"subscriberTopics":"unauthenticated.VES_NOTIFICATION_OUTPUT",
"pcfEndpoint": "10.20.249.52:39001",
"cdsEndpoint":"cds-blueprints-processor-http.onap.svc.cluster.local:8080
",
"alertPolicies":"{\"PDU_SessionRegistrationFailed_Alert\":{\"domain\":\"
notification\",\"eventName\":\"Notification_PDU_SessionRegistrationFaile
d\",\"changeType\":\"config-unavailable\"}}",
"alertActionPolicies":"{\"PDU_SessionRegistrationFailed_Alert\":
{\"action\": \"PCF_Subscription_policy\",\"description\":\"Do perform
subscription policy on the PCF\"}}" }
```

- Start programs

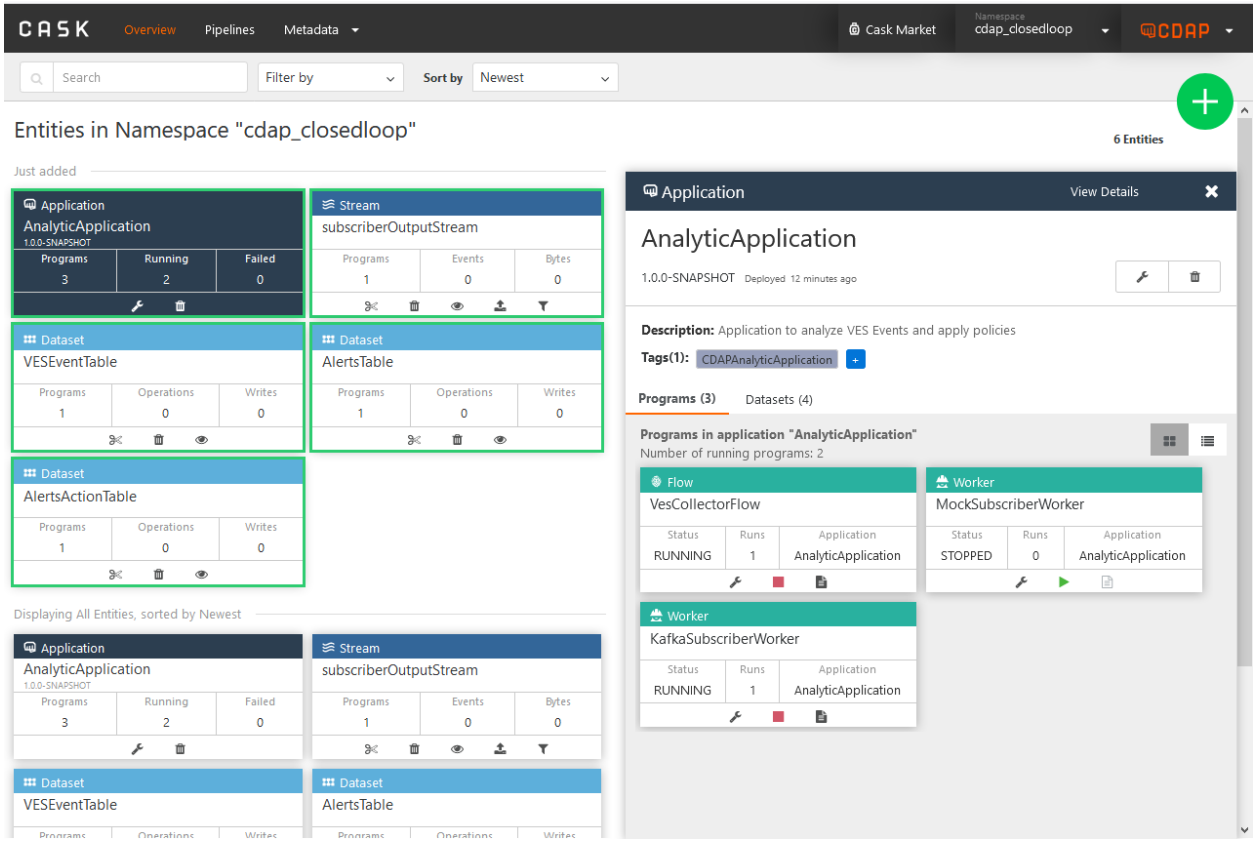
```
curl -X POST
http://$CDAP_HOST/v3/namespaces/cdap_closedloop/apps/AnalyticApplication
/workers/KafkaSubscriberWorker/start

curl -X POST
http://$CDAP_HOST/v3/namespaces/cdap_closedloop/apps/AnalyticApplication
/flows/VesCollectorFlow/start
```

- CDAP UI can be accessed at: “<AMCOP VM IP address>:31011”

- Below screenshots shows the namespace *cdap_closedloop (*top right corner*)* deployed on the cdap platform



- The below screenshot shows the programs '*VesCollectorFlow*', '*KafkaSubscriberWorker*' (right-hand side of the screen) running in the '*AnalyticApplication*' deployed in above steps.



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).

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.

Following packages are required for enabling monitoring in AMCOP:

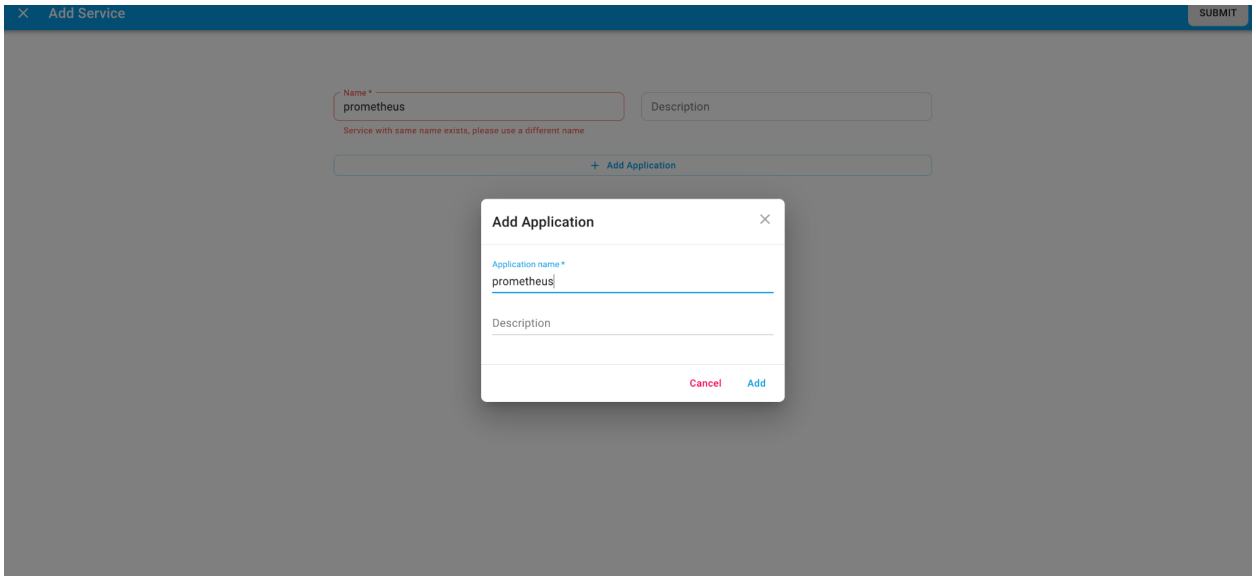
- Prometheus.tgz: Prometheus is a metrics collector application
- Grafana.tgz: Grafana is a visualization tool for monitoring that supports Prometheus as a data source.
- Since Prometheus follows the pull model for collecting metrics, each application is required to expose the metrics in a specific format. If the application/system does not support this, an adapter plugin termed as ‘exporters’ is required. For example, the plugin ‘nodeexporter’ exposes OS-related metrics in Prometheus format. For monitoring Kubernetes clusters following exporters are useful:
    - ksmetrics.tgz (Kubestate metrics, provides metrics on Kubernetes objects like pods, nodes, deployments etc)
    - nodeexporter.tgz (Node exporter, in context of Kubernetes, gives OS-level metrics of the host machine)

The helm charts for above mentioned packages can be downloaded from below URL. Please note that these helm charts are different from the community helm charts. For example the Prometheus helm chart has the configurations for scraping ***cadvisor*** and ***nodeexporter*** metrics.
https://drive.google.com/drive/folders/1EcfRw0TvMnzxdzRddU5JMd-UjUnfCkX7?usp=sharing

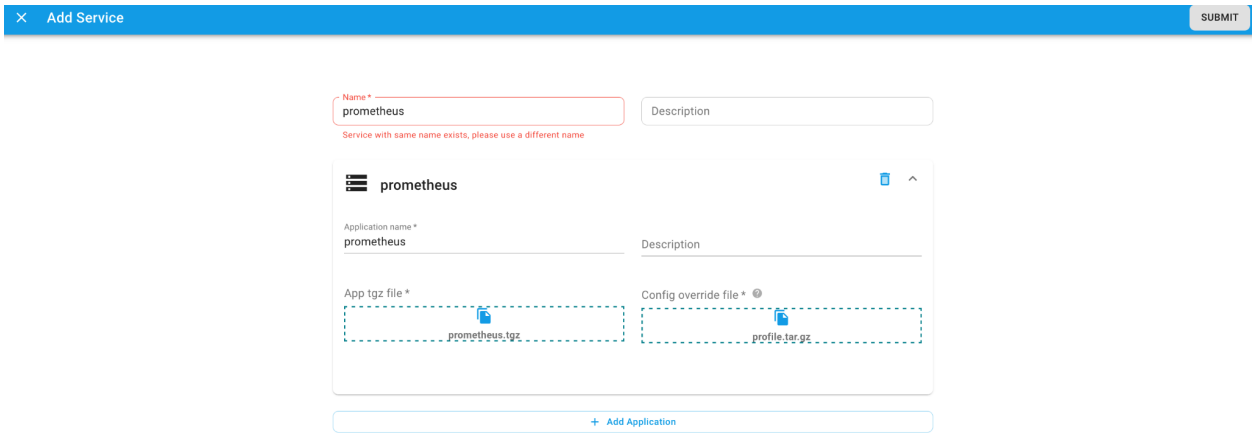
Note: This exercise is a demonstration of how a telemetry service can be deployed using AMCOP. It is not recommended for production class use cases and not all the applications are deployed with persistent storage, secrets etc. Please refer to the community helm chart documentation for more details:
https://github.com/helm/charts/tree/master/stable/prometheus

After deploying AMCOP, and you have created a tenant, add services for Prometheus, Grafana and exporters (Or a single service as per users requirement), follow the steps to add service and application.

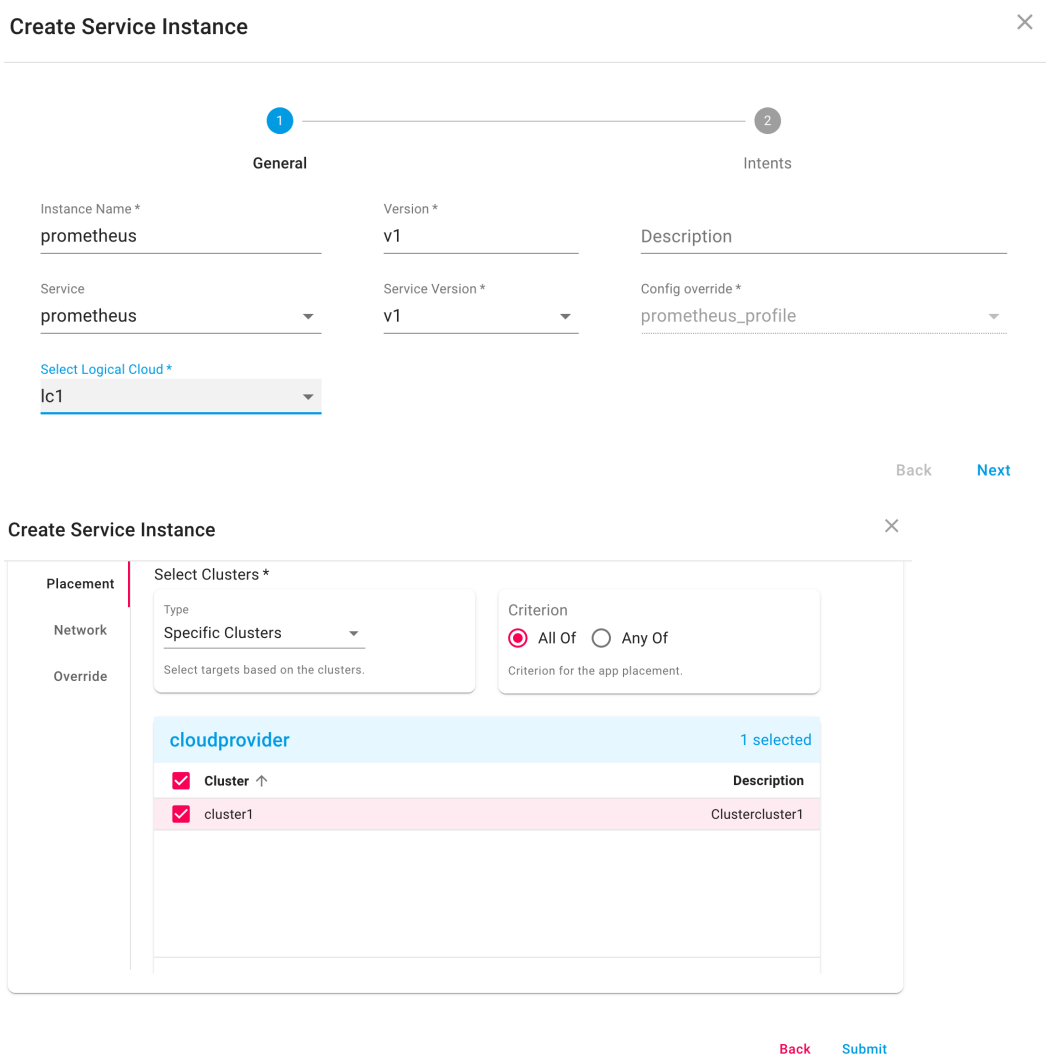
Step 1: Add prometheus service. Add an application named ‘prometheus’



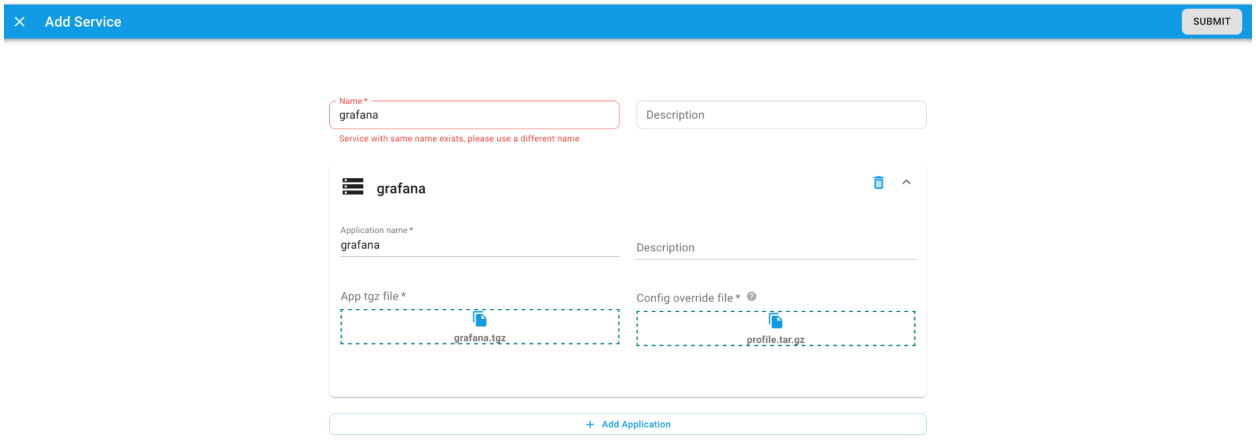
Step 2: Upload prometheus.tgz

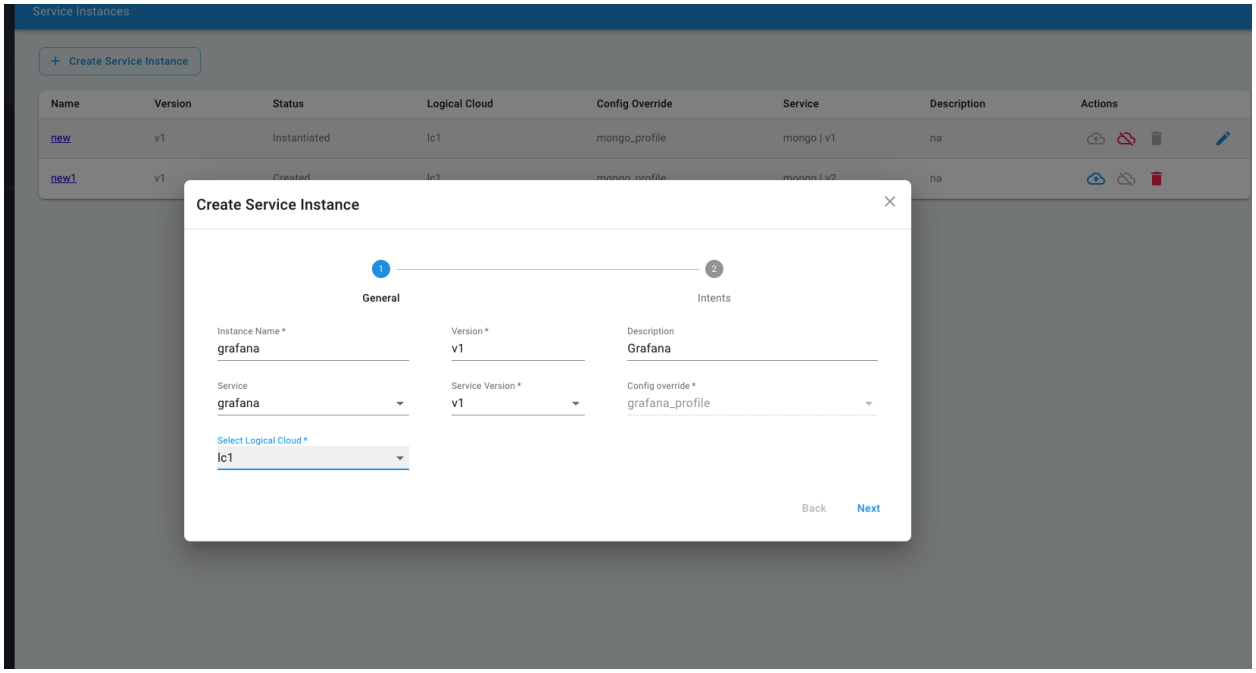


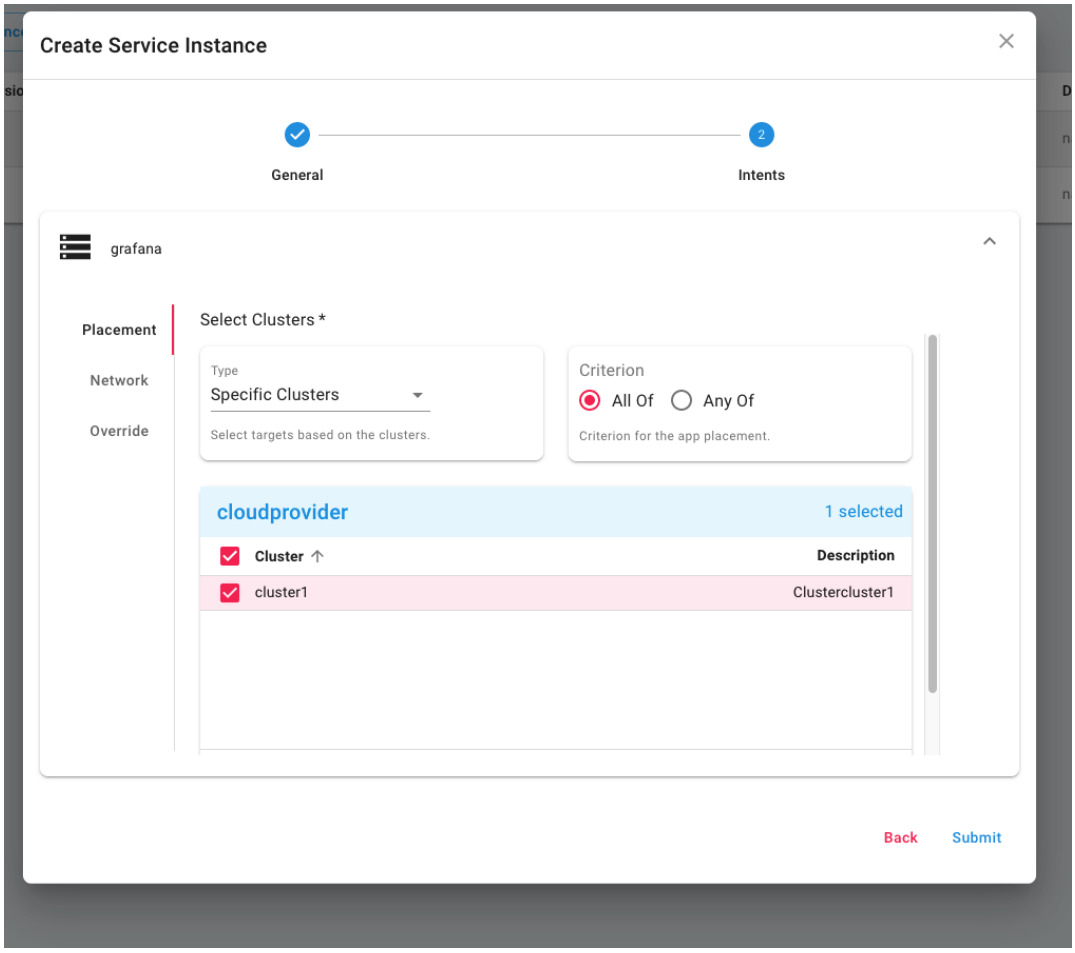
Step 3: Create a service instance of prometheus



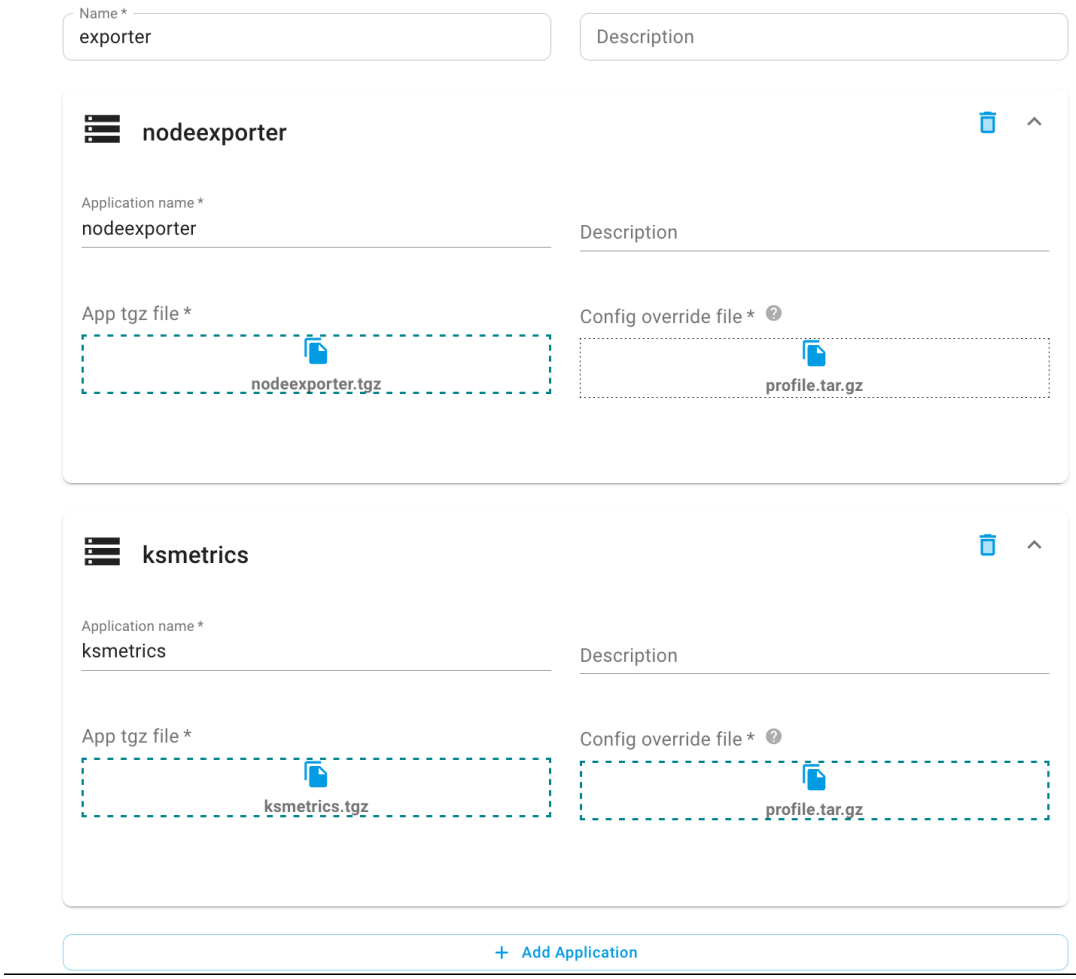
Step 4: Similarly add a new service ‘grafana’ and an application ‘grafana’. Also, create a service instance of newly added service.

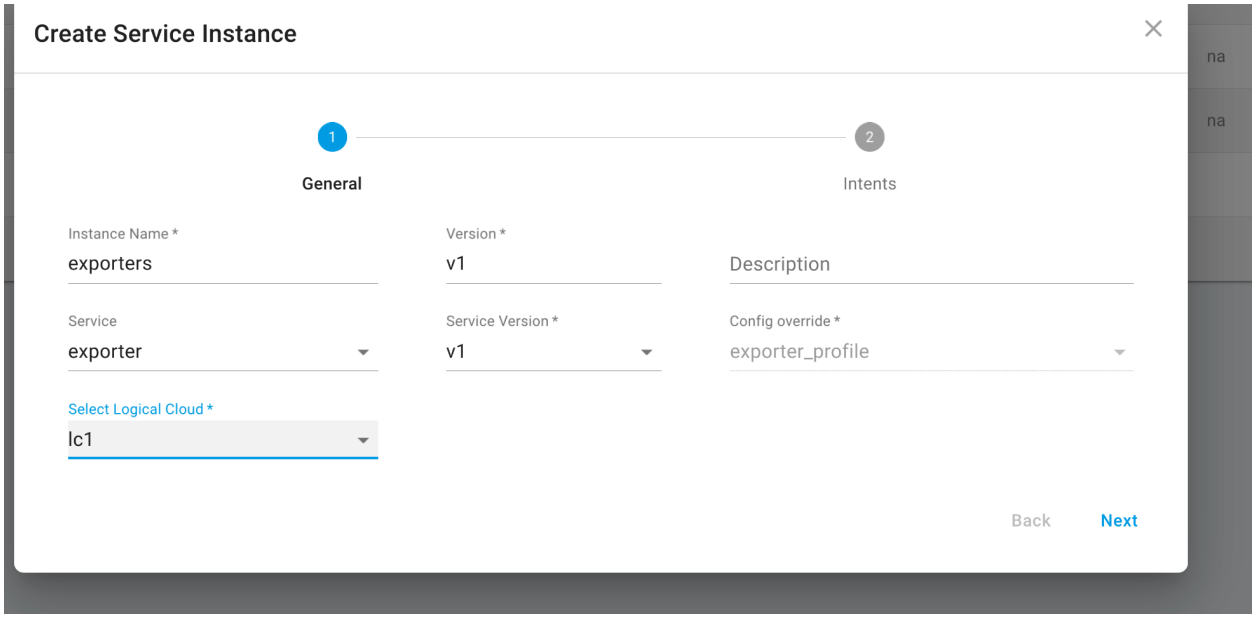

Add Service
SUBMIT
Name *
grafana
Service with same name exists, please use a different name
Description
grafana
Application name *
grafana
Description
App tgz file *
grafana.tgz
Config override file *
profile.tar.gz
+ Add Application


Service Instances
+ Create Service Instance
Name
Version
Status
Logical Cloud
Config Override
Service
Description
Actions
new
v1
Instantiated
lc1
mongo_profile
mongo | v1
na
new1
v1
Created
lc1
mongo_profile
mongo | v2
na
Create Service Instance
1
General
2
Intents
Instance Name *
grafana
Version *
v1
Description
Grafana
Service
grafana
Service Version *
v1
Config override *
grafana_profile
Select Logical Cloud *
lc1
Back
Next

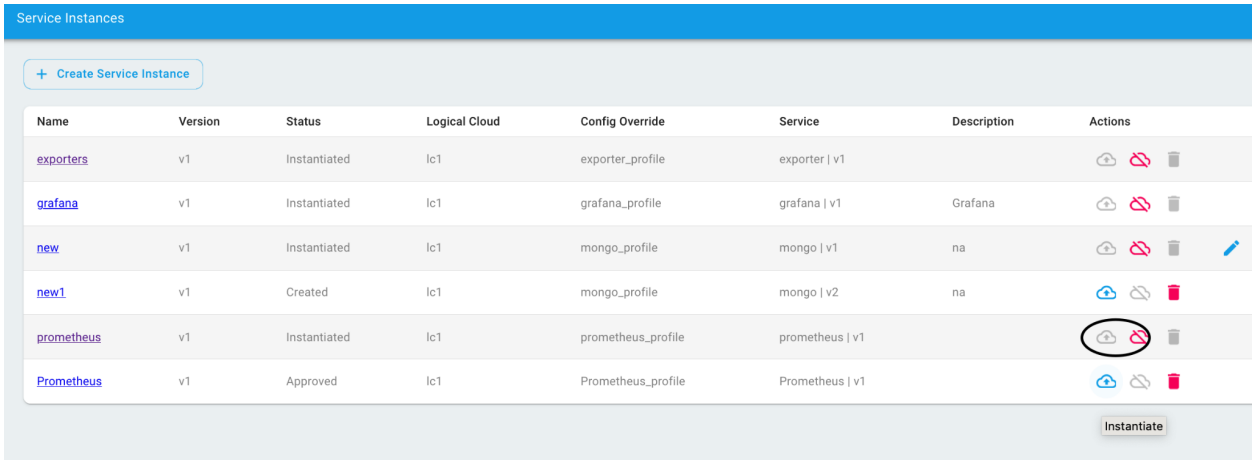


Step 5: Add a service ‘exporters’ and add the applications “nodeexporter” and “ksmetrics” as below. Follow steps to create service instances as in previous steps.


Name *
exporter
Description
nodeexporter
Application name *
nodeexporter
Description
App tgz file *
nodeexporter.tgz
Config override file *
profile.tar.gz
ksmetrics
Application name *
ksmetrics
Description
App tgz file *
ksmetrics.tgz
Config override file *
profile.tar.gz
+ Add Application


Create Service Instance
1
General
2
Intents
Instance Name *
exporters
Version *
v1
Description
Service
exporter
Service Version *
v1
Config override *
exporter_profile
Select Logical Cloud *
lc1
Back
Next

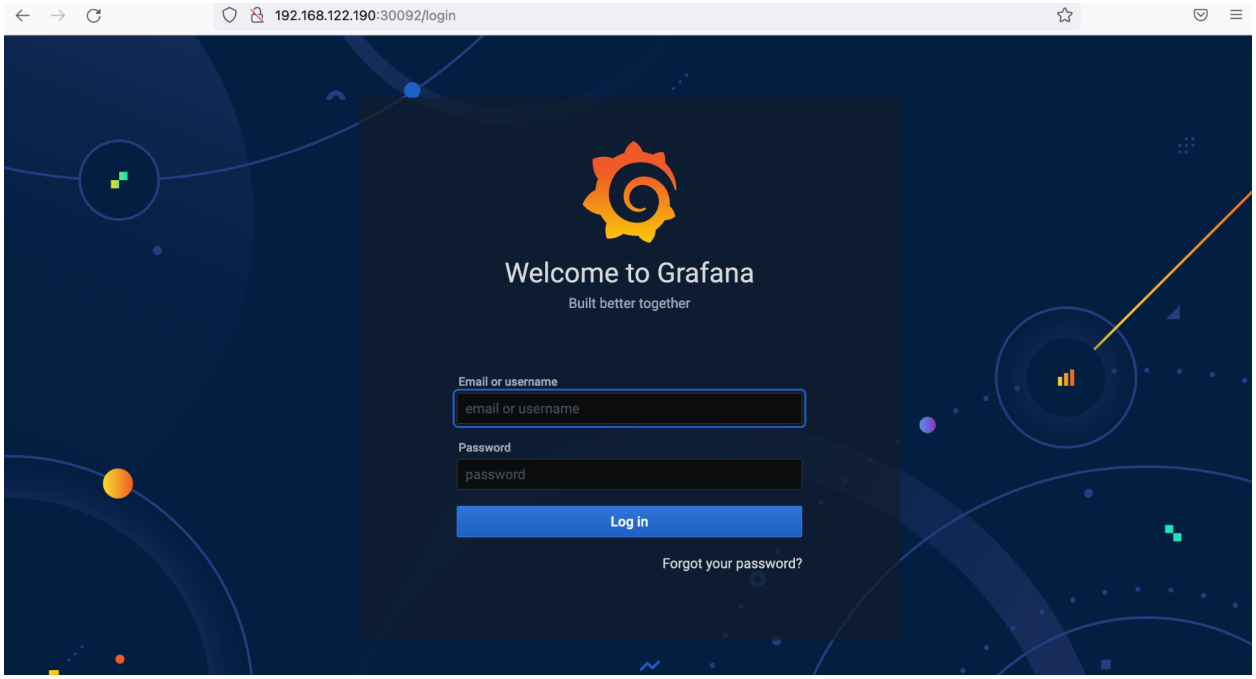
Instantiate all new created service instances:



Once the pods are created, get the grafana port using the following command. By default, all services will be created in ‘default’ namespace:

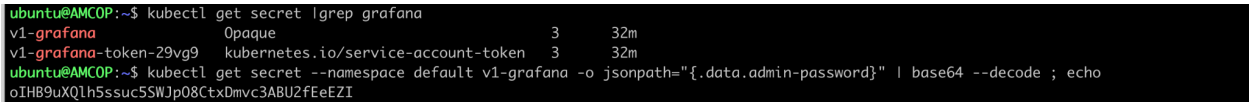
```
ubuntu@AMCOP:~$ kubectl get svc |grep grafana
v1-grafana    NodePort    10.99.135.91     <none>        80:30092/TCP    30m
```

Access the grafana at <host-ip>:<grafana-port> . For example: http://192.168.122.190:30092



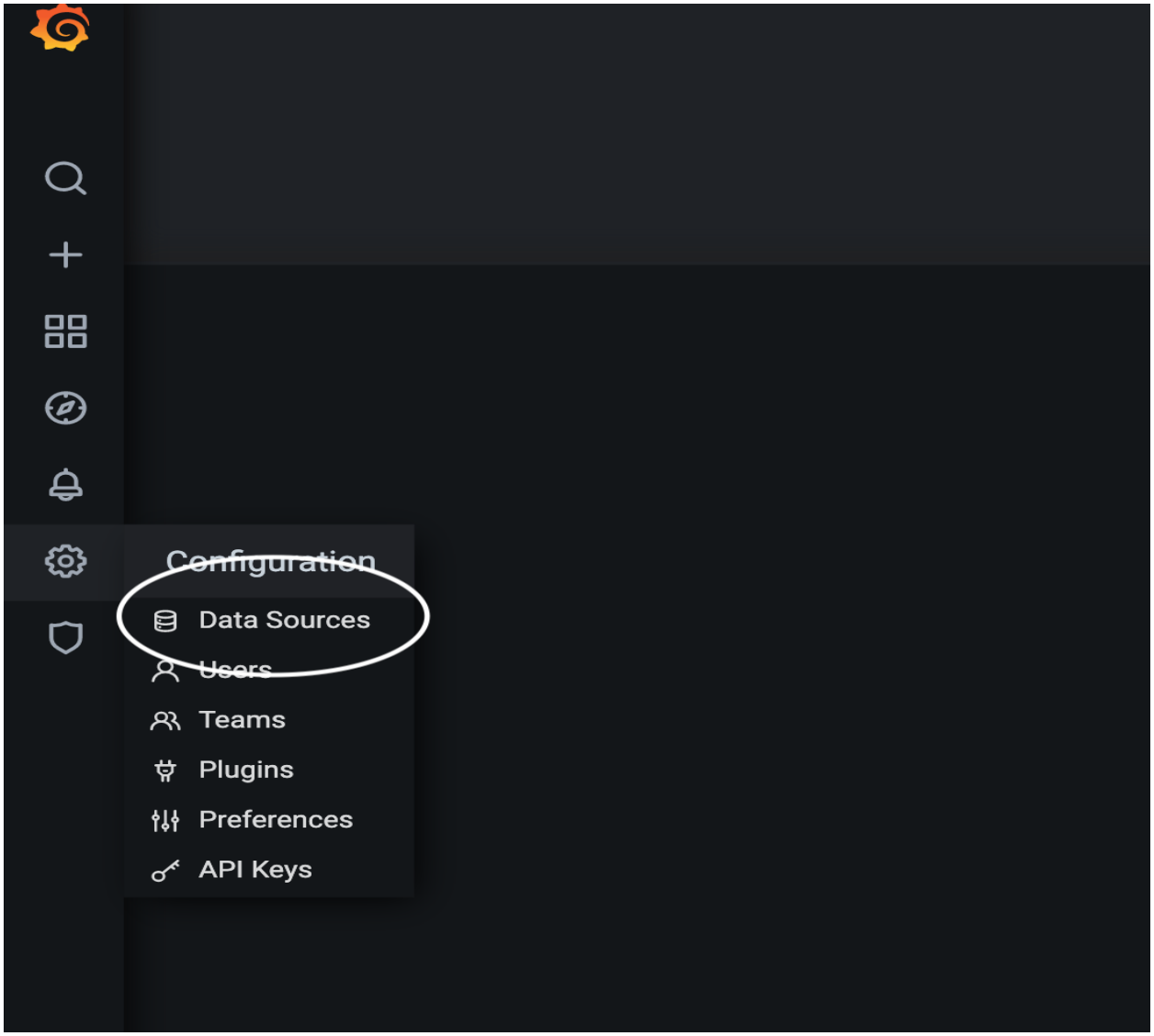
Run the following commands to get the password for the username ‘admin’.

```
kubectl get secret | grep grafana

kubectl get secret --namespace default v1-grafana -o
jsonpath="{.data.admin-password}" | base64 --decode ; echo
```

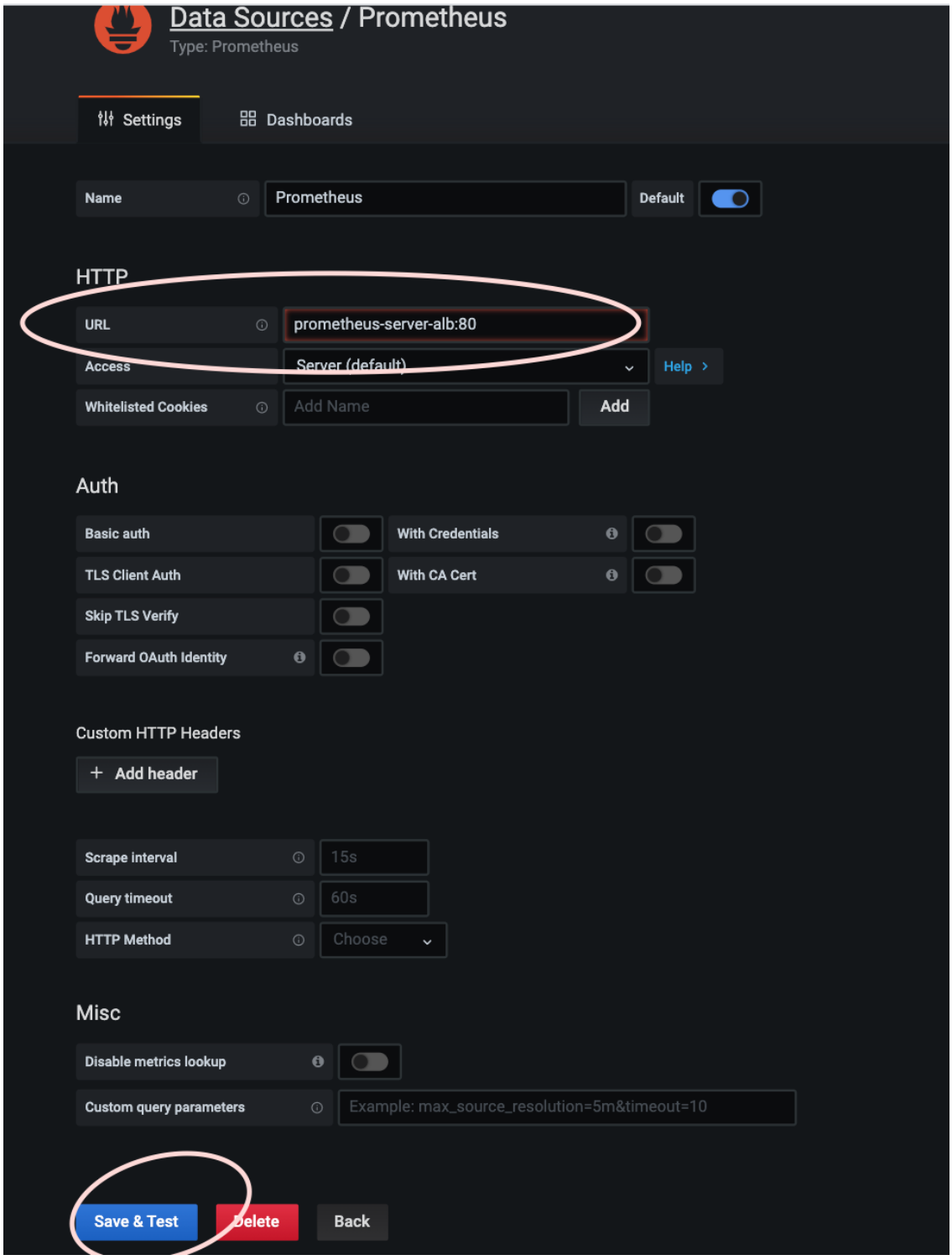


Login to grafana using the “admin” user credentials.

After login, we need to add the Prometheus application which was deployed using AMCOP as a data source for grafana. Follow the below steps for the same:

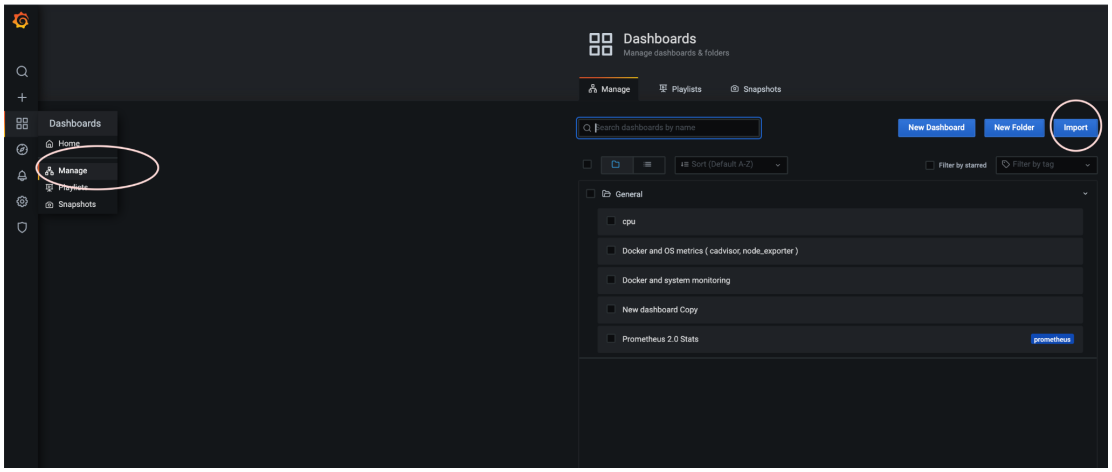


From the data sources, select Prometheus option and provide the service end point of prometheus as URL and try ‘save and test’. If everything is fine, this step will succeed. An error message during this test indicates that the deployment has issues.

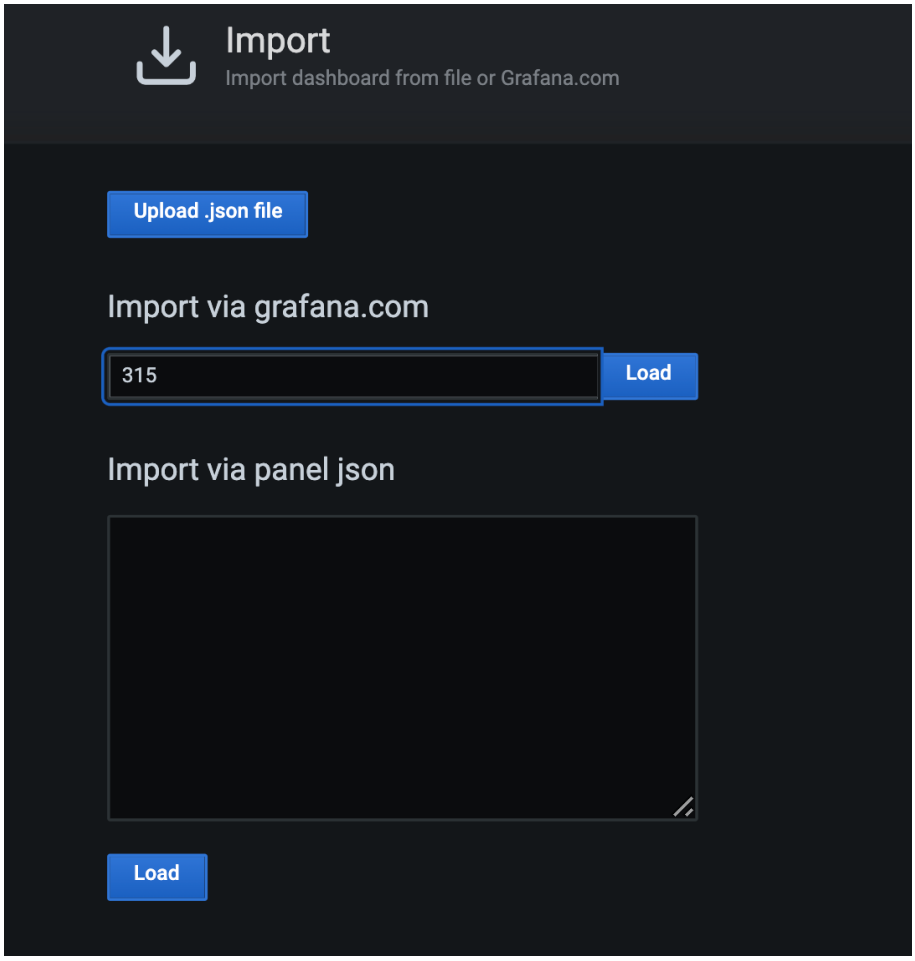


Now we need to add a dashboard. There are sample dashboards available on grafana.com website or you can build your own dashboard. Following steps illustrate adding a standard dashboard. (Dashboard No: 315)

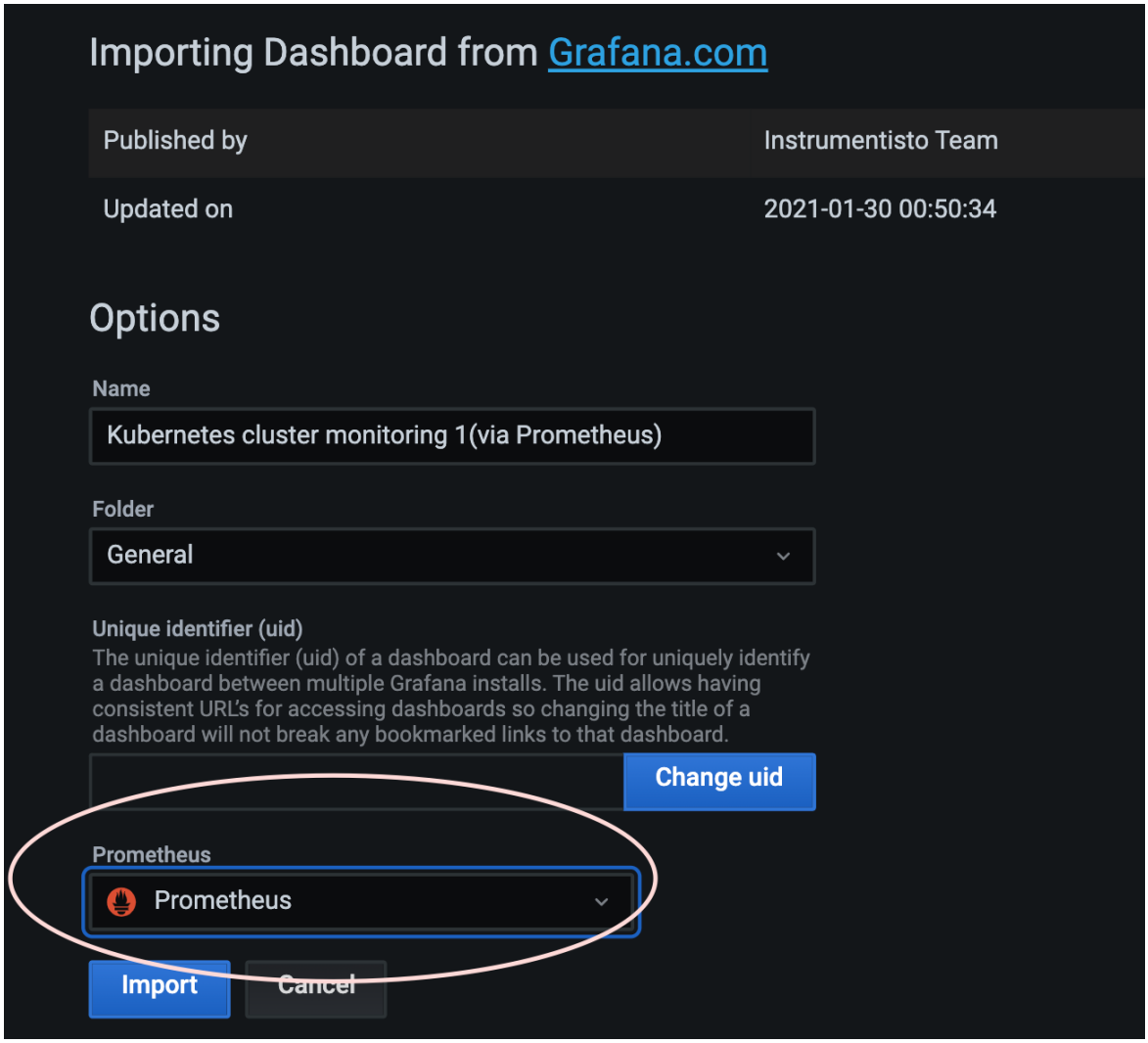
Select Dashboard tab from left of UI and select ‘manage’. Click on ‘import’ as shown below:



Type dashboard number (315 in this case), and select Load next to the text box.



Select Prometheus as a data source, and select the Import button.



Now you will be able to see a dashboard like below

Prometheus metrics to Kafka (DMaap) bus

> **Note:**
>
> **This is for the advanced users of Prometheus. It requires you to modify the helm charts of Prometheus and prometheus-kafka-adapter to point to the update endpoints.**

For forwarding Prometheus metrics to DMaap, you need to use the plugin 'prometheus-kafka-adapter'. This adapter helm chart can be downloaded from https://drive.google.com/drive/folders/1EcfRw0TvMnzxdzRddU5JMd-UjUnfCkX7?usp=sharing

In the 'prometheus-kafka-adapter' chart, edit values.yaml to point KAFKA_BROKER_LIST to point to your dmaap endpoint (Replace message-router.onap.svc.cluster.local:3904 with dmaap endpoint you wish to send). Also update KAFKA_TOPIC with the kafka/dmaap topic in the same values.yaml file.

Now in prometheus, in template/deployment.yaml (section scrape_configs) add the following line to add a remote endpoint.

```
 remote_write:
      - url: <prometheus-kafka-adapter receive endpoint>
For example:
 remote_write:
      - url: "http://kfadapter-prometheus-kafka-adapter:80/receive"
```

At this point, the metrics scraped by Prometheus will be available at the KAFKA_TOPIC.

An example use case is to configure the CDAP application worker to poll metrics at this topic and stream it into the data processing flowlets. This enables the analytics applications in the closed loop to analyze these metrics and take any specific actions.

Service Management & Orchestrator (SMO)

This section explains the SMO functionality supported in AMCOP. SDNR 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.

SDNR supports GUI and swagger RESTconf interfaces for device management. Following are the default credentials, applicable for both SDNR GUI and RESTconf.

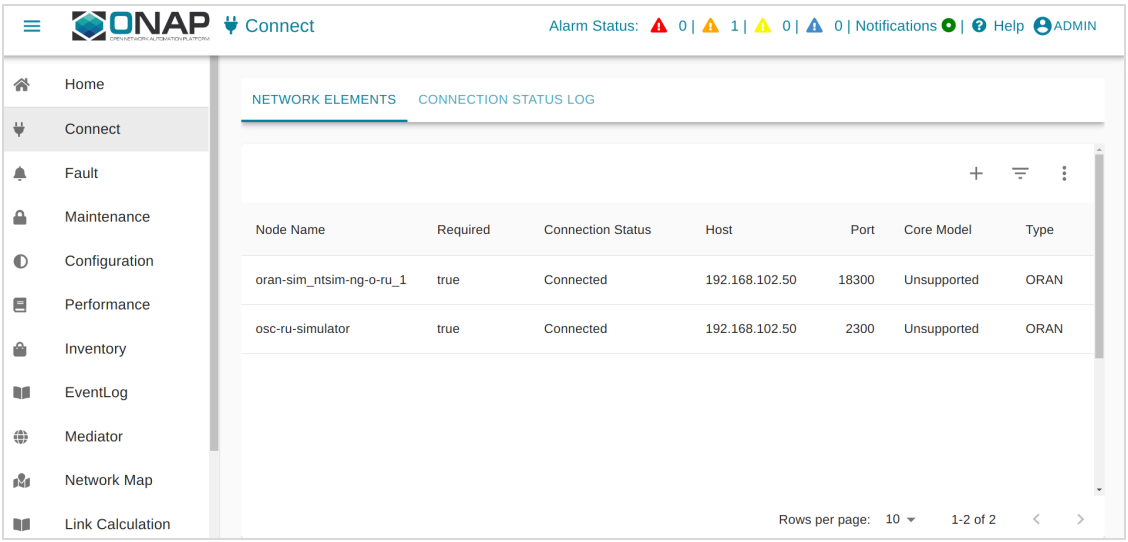
AMCOPGUI

- GUI is accessible at http://<ancop-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 Following provides more information on each UI page and its capabilities.

Connect

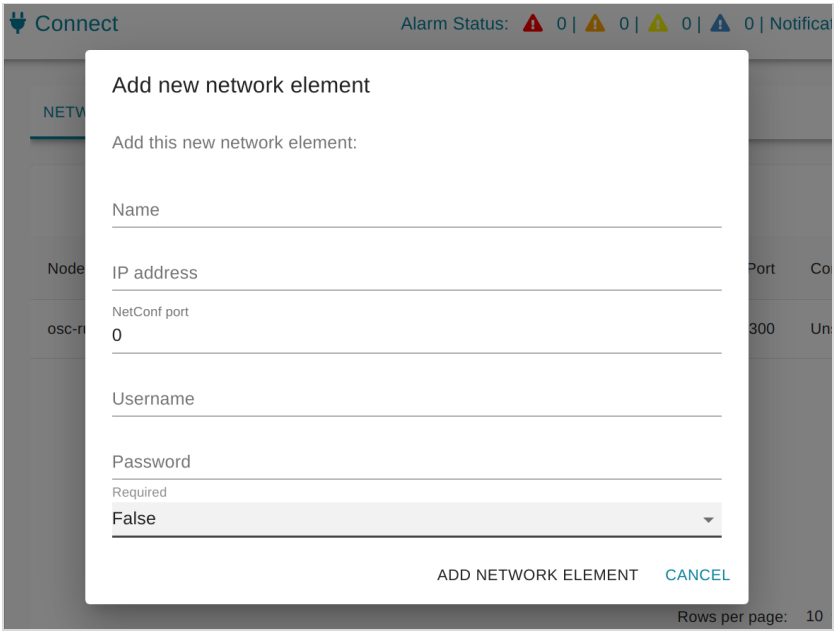
This “Connect” page on the left-hand side of the SDNR GUI shows all the connected devices and their status.



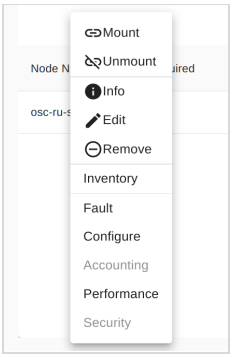
To add a new network element, click on the button at the top right corner, it will open the popup to enter the various details of the device. The system will mount and connect if all the given

details are correct. If you don't have any real RAN device, follow **Appendix B: O1 simulator installation** to create O1 simulators which can be added using SDNR GUI.

Note: To experience manual steps of adding O1 simulator devices, refer to option-2 and option-3 in **Appendix B: O1 simulator installation**, whereas option-1 will automatically connect the O1 simulators to the SDNR. With option-1, users can view the O1 simulator devices automatically getting added under the "Connect" page.

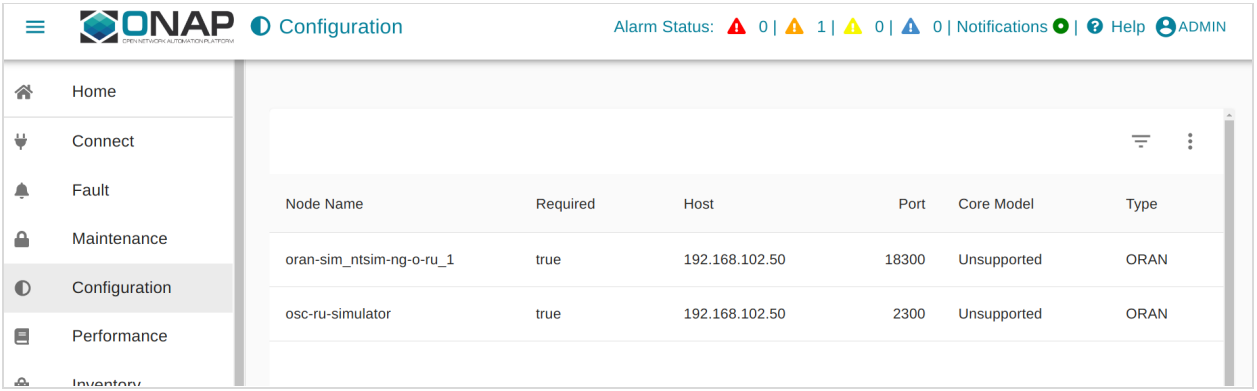


From the "Connect" page, you can go to the other pages with the help of shortcuts. Just right click on the device. It will list down all the shortcuts.

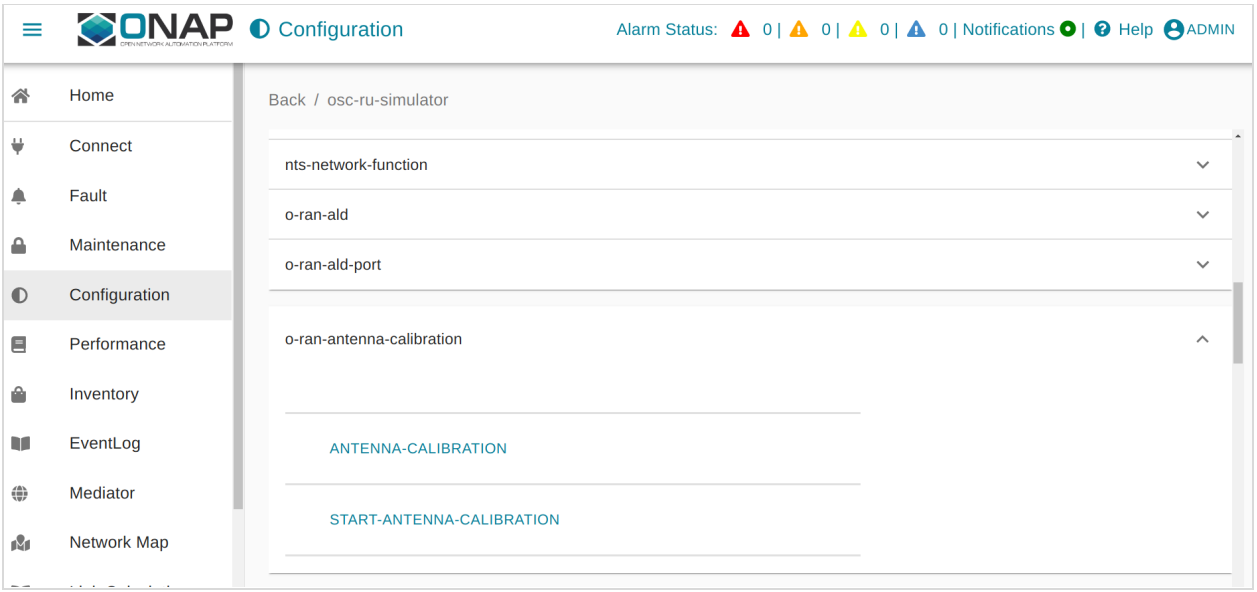


Configuration

Configuration page on the left hand side of the SDNR GUI shows all the connected devices and users can retrieve and edit all the associated configurations of the devices.

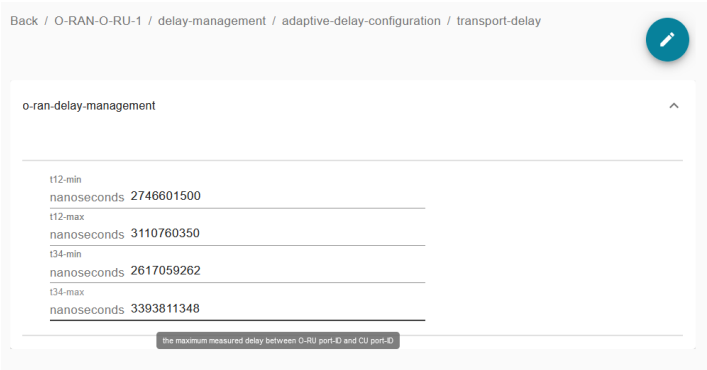


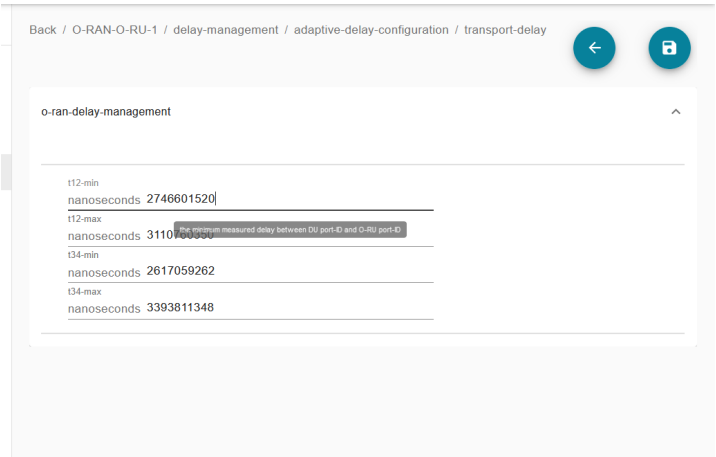
Clicking on any of the listed devices takes to a page where it lists all the Yang models inside.



Clicking on the Yang model shows the current configurations.

Users can click on the edit button  to update, change the configuration and click on the save button  to save the changes.

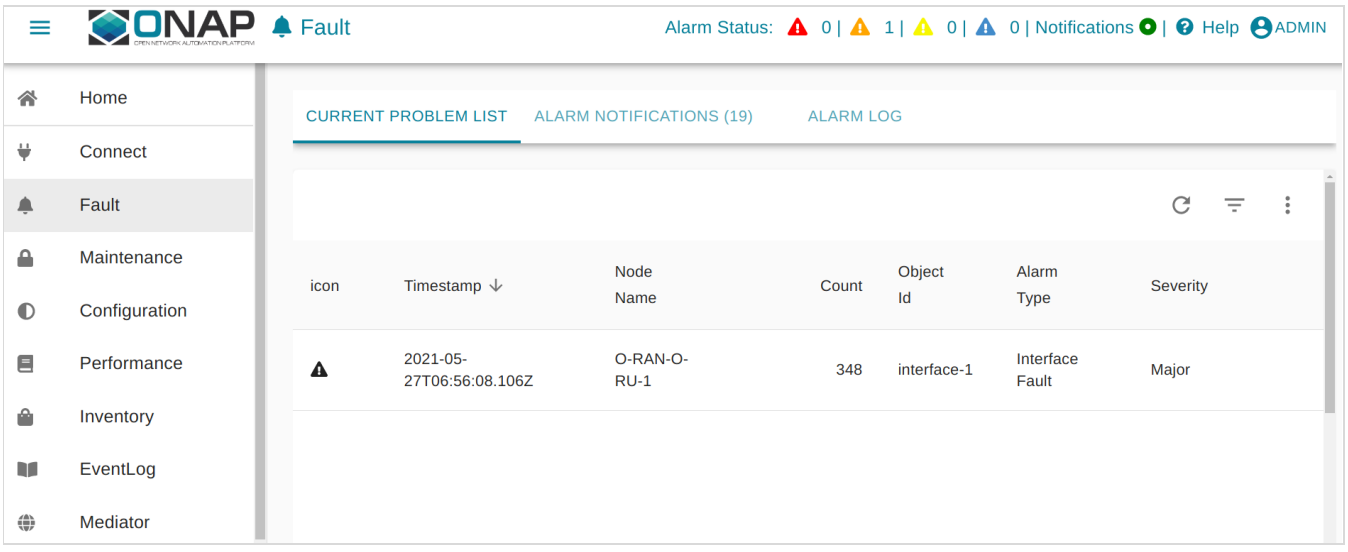




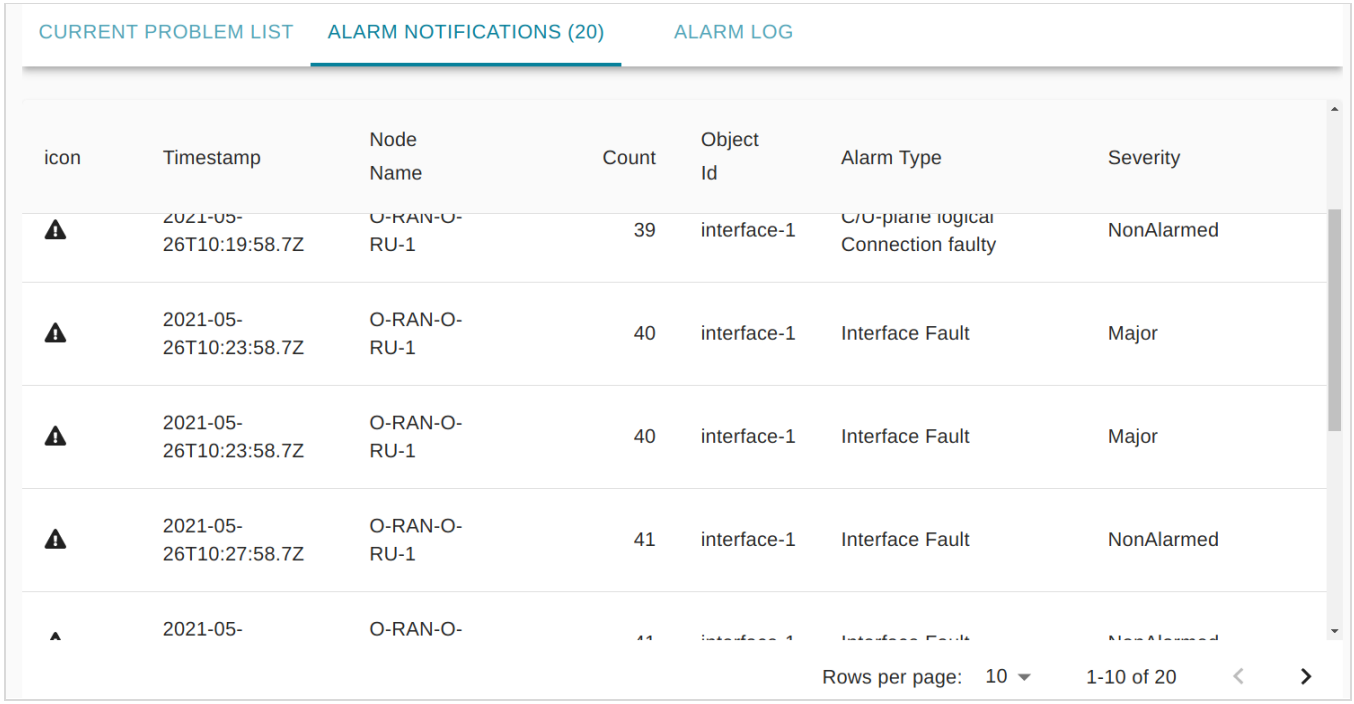
Fault

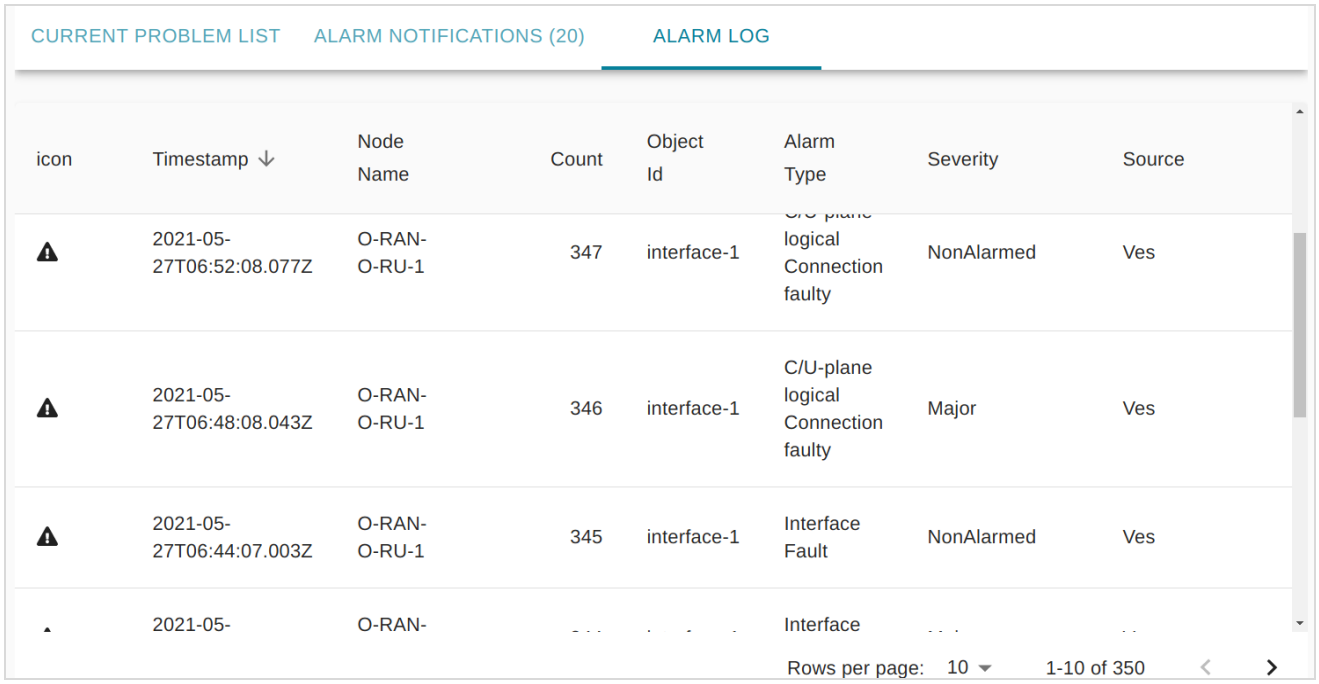
The fault page on the left-hand side of the SDNR GUI shows all the notifications and alarms.

CURRENT PROBLEM LIST tab shows the most recent alerts.



ALARM NOTIFICATIONS and ***ALARM LOG*** shows all the alerts.

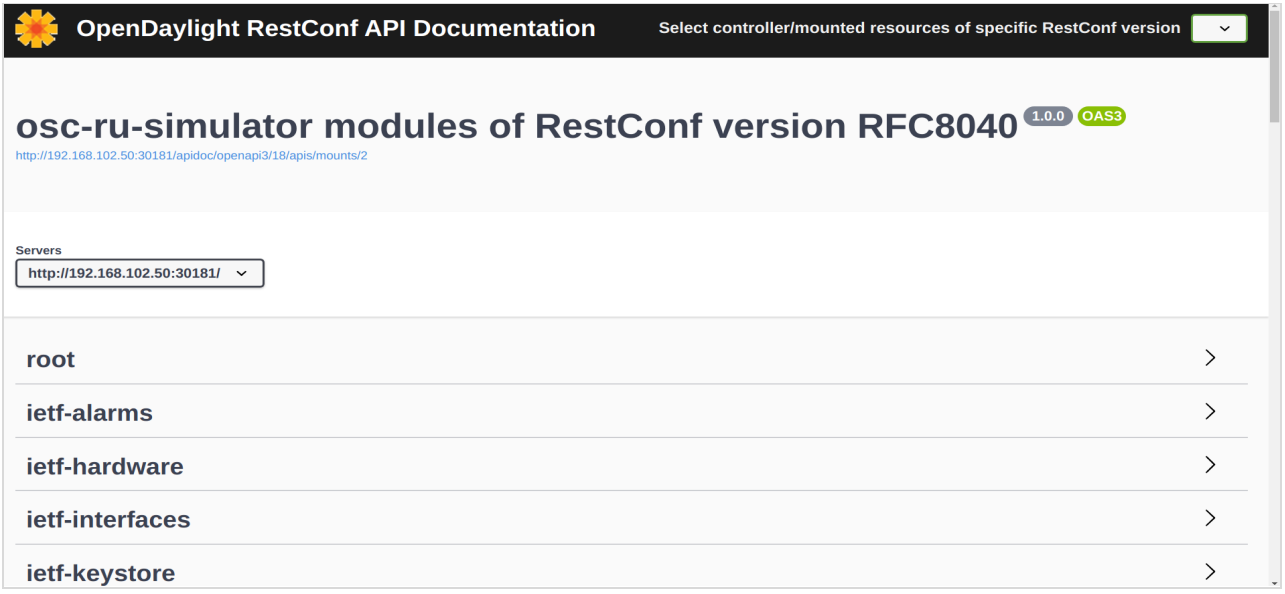




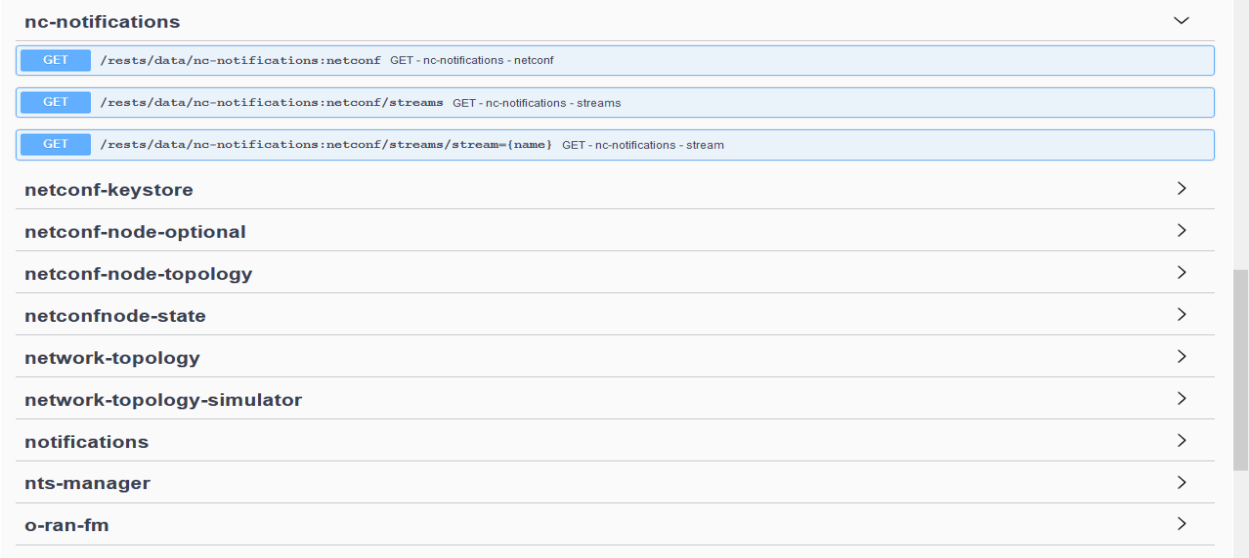
Swagger RESTconf

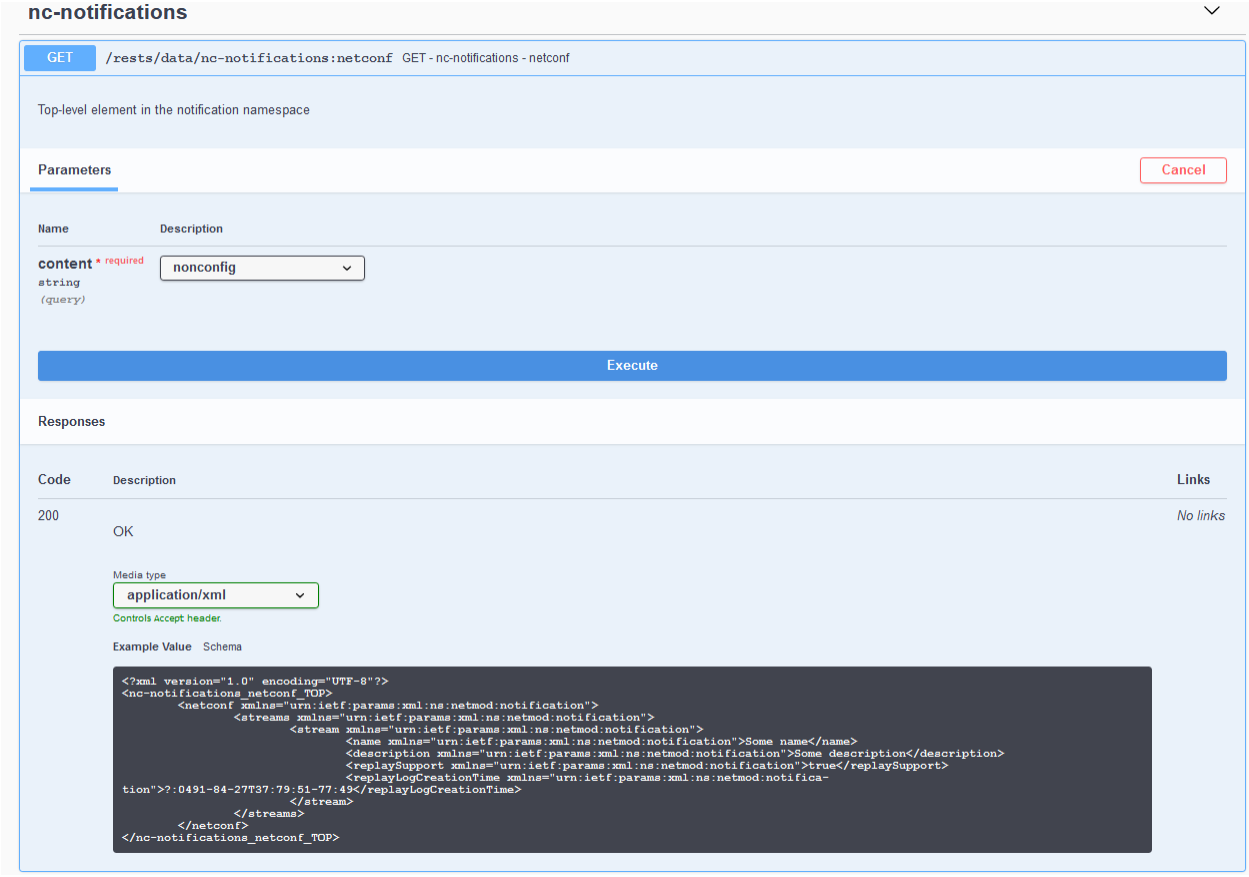
Restconf swagger interface is available at
http://<amcop-vm-ip>:30181/apidoc/explorer/index.html

Select a device from the top right corner to display all the Yang models of the device.



Select the specific Yang model to see the supported operations. Users can perform all the supported operations like GET, POST, PUT, DELETE.




nc-notifications
GET
/rests/data/nc-notifications:netconf GET - nc-notifications - netconf
Top-level element in the notification namespace
Parameters
Cancel
Name
Description
content * required
string
(query)
nonconfig
Execute
Responses
Code
Description
Links
200
OK
No links
Media type
application/xml
Controls Accept header.
Example Value
Schema
<?xml version="1.0" encoding="UTF-8"?>
<nc-notifications_netconf_TOP>
<netconf xmlns="urn:ietf:params:xml:ns:netmod:notification">
<streams xmlns="urn:ietf:params:xml:ns:netmod:notification">
<stream xmlns="urn:ietf:params:xml:ns:netmod:notification">
<name xmlns="urn:ietf:params:xml:ns:netmod:notification">Some name</name>
<description xmlns="urn:ietf:params:xml:ns:netmod:notification">Some description</description>
<replaySupport xmlns="urn:ietf:params:xml:ns:netmod:notification">true</replaySupport>
<replayLogCreationTime xmlns="urn:ietf:params:xml:ns:netmod:notifica-
tion">?:0491-84-27T37:79:51-77:49</replayLogCreationTime>
</stream>
</streams>
</netconf>
</nc-notifications_netconf_TOP>

Configure ELK for debugging

We can deploy the ELK stack in the target clusters in order to collect/filter/collate logs from the pods that are deployed in the target cluster. In this section we are going to deploy and set up ELK using the AMCOP platform, and set up a sample index_pattern and log filter on the Kibana dashboard.

We are going to create a service call ELK from the AMCOP GUI and add the following applications under the service,

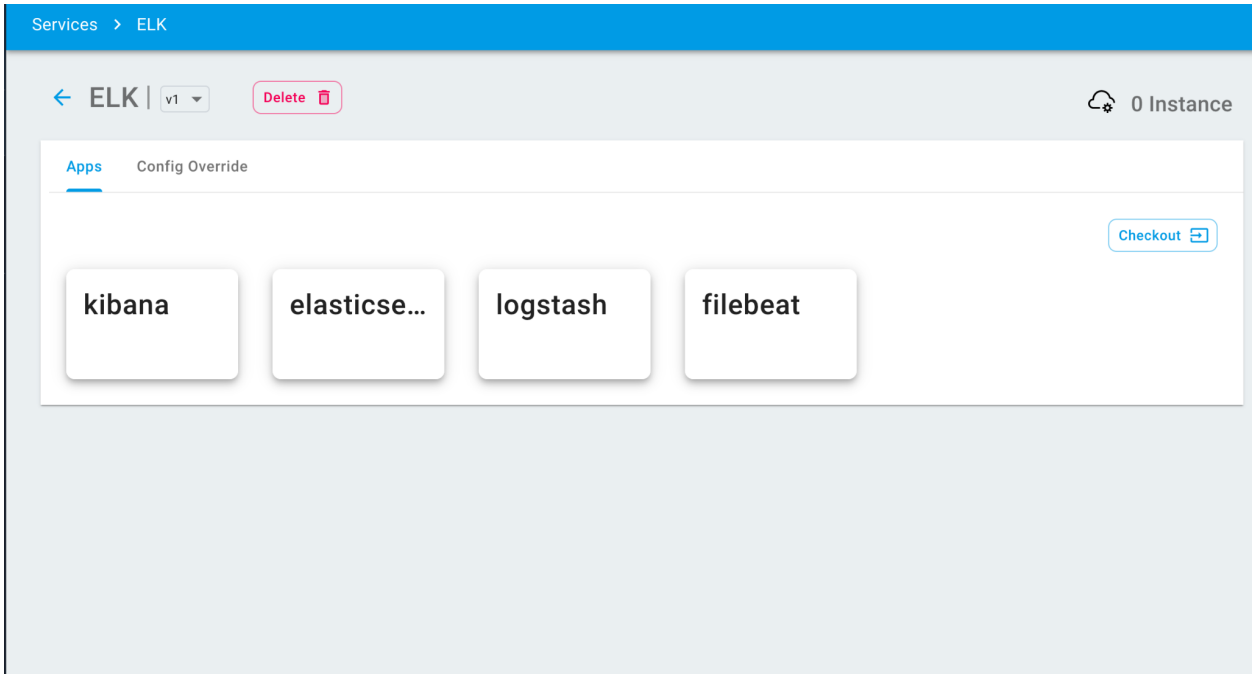
- Elasticsearch
- Logstash
- Filebeat (daemonset, that ships the logs to logstash)
- Kibana

The helm charts contain the required rbac yamls for creating the serviceaccounts. Also all the applications are referring to each other with the servicename:port.
The helm charts can be downloaded from
https://drive.google.com/drive/folders/1EcfRw0TvMnzxdzRddU5JMd-UjUnfCkX7?usp=sharing

Note: This exercise is a demonstration of how a logging infrastructure can be deployed using AMCOP. It is not recommended for production class use cases and not all the applications are deployed with persistent storage, secrets etc. Please refer to the community helm chart documentation for more details:
https://github.com/elastic/helm-charts

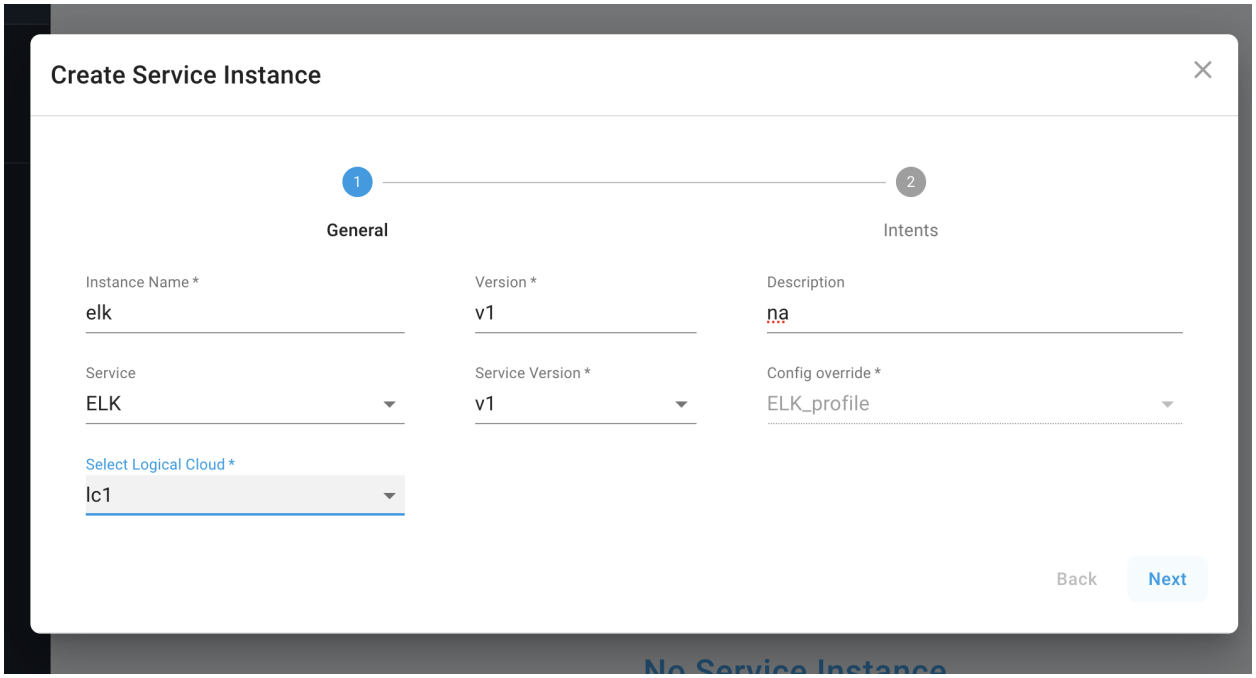
The Service

A service is created using the above helm charts in AMCOP UI. The following picture shows the service created.

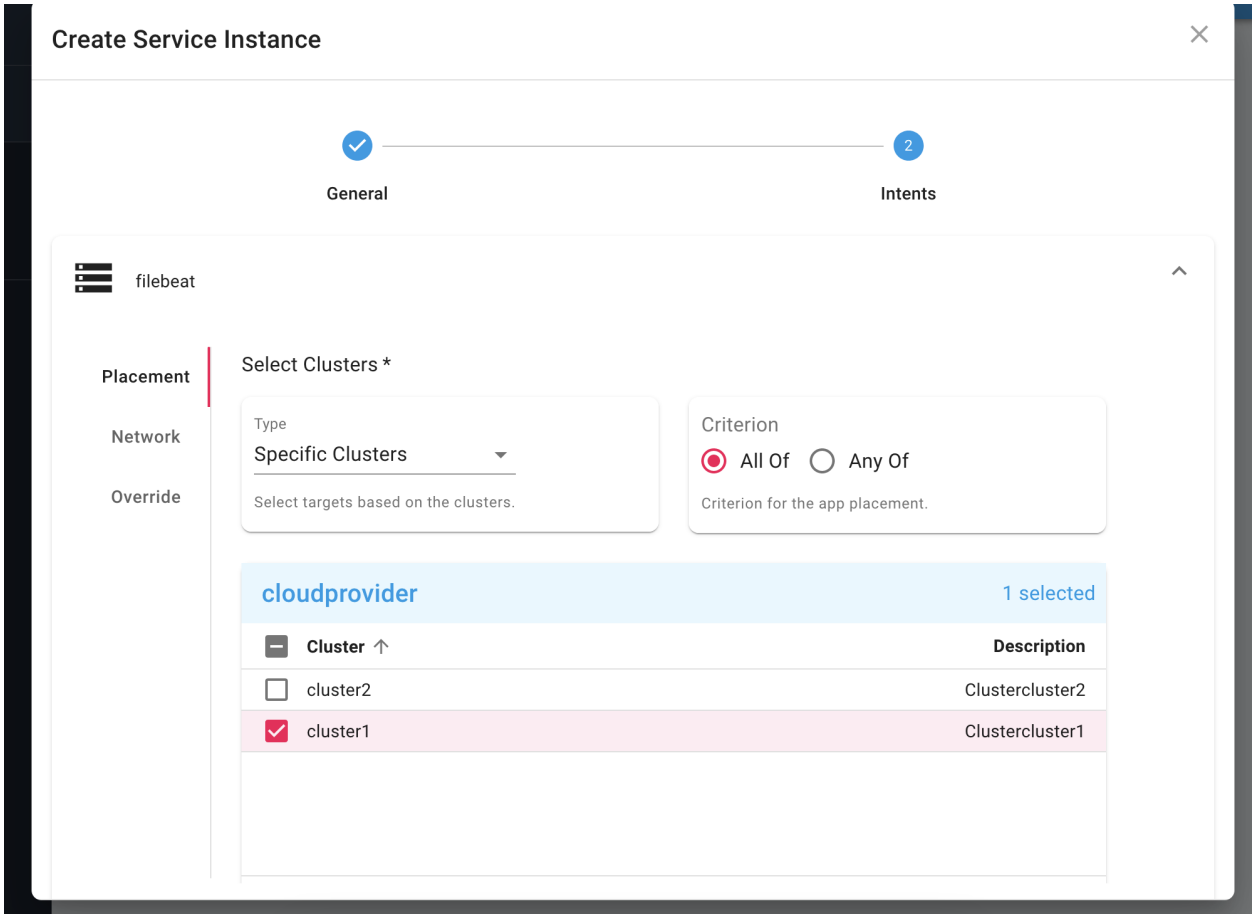


Service Instantiation

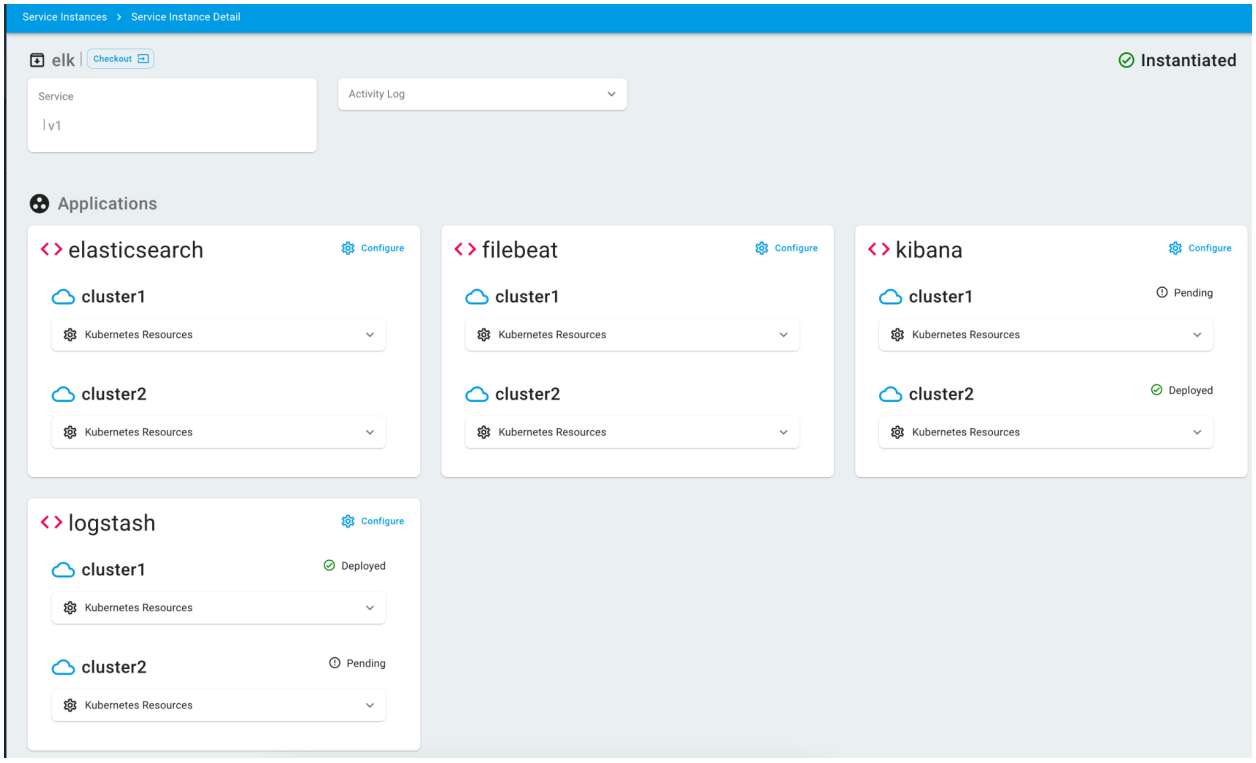
Create a service instance and select the ELK service that has been created,



Specify the placement intents,



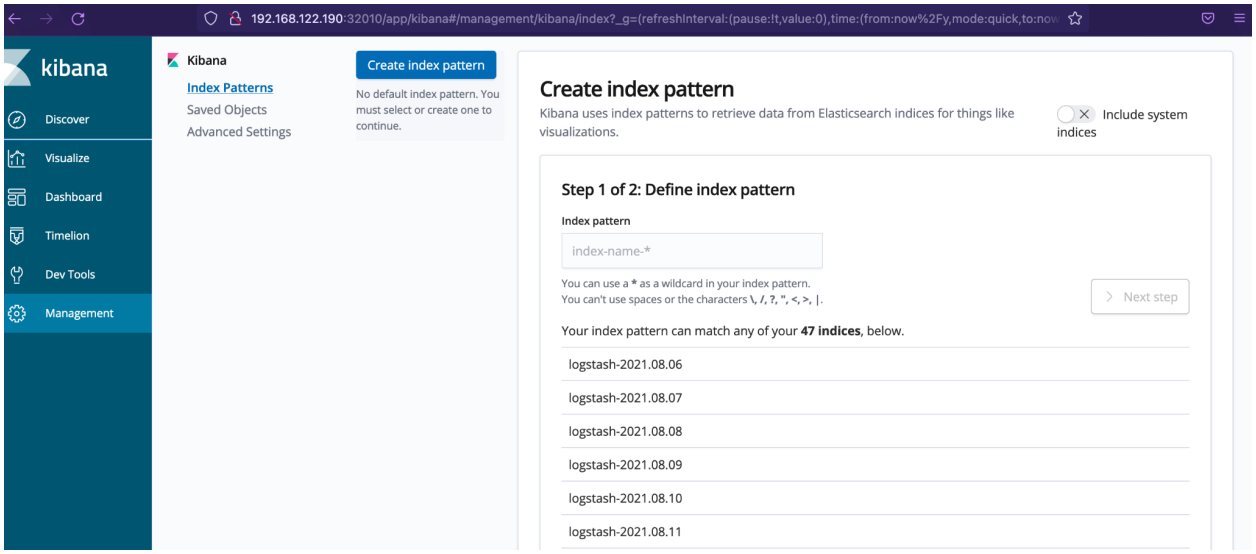
Submit and then click instaintate.



All the pods and service should be up and running as shown below.

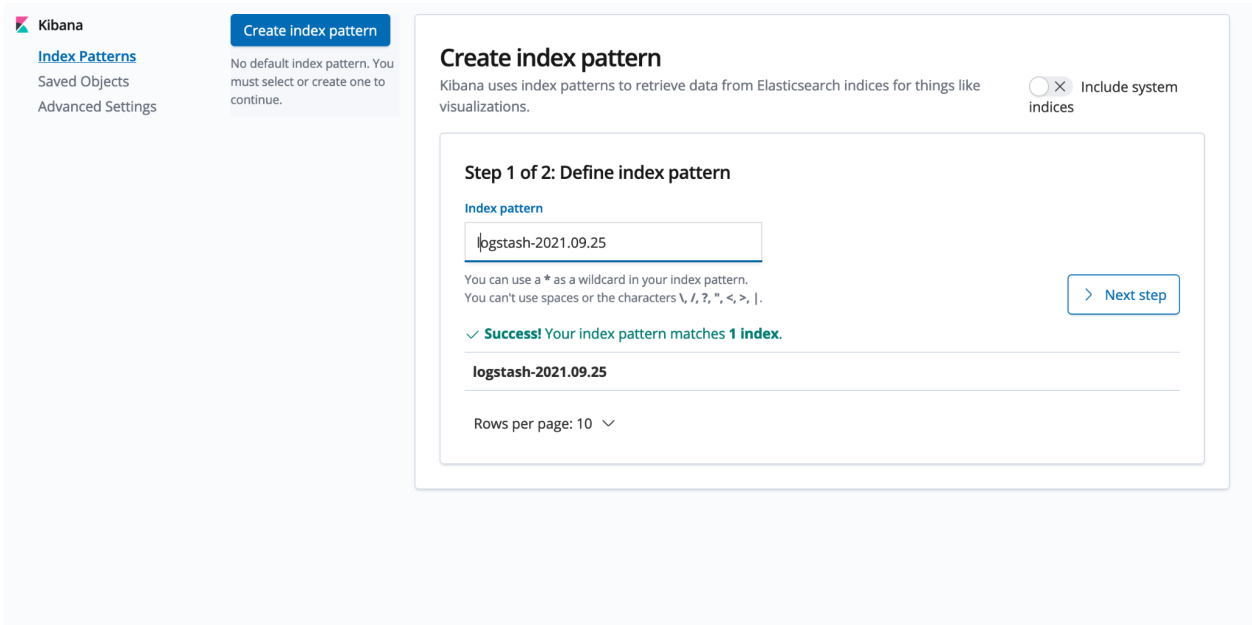
```
ubuntu@AMCOP:~/snadeep/elk/taas$ kubectl get all
NAME                                      READY   STATUS    RESTARTS   AGE
pod/elasticsearch-logging-0               1/1     Running   0          27s
pod/elasticsearch-logging-1               1/1     Running   0          23s
pod/filebeat-ssfhm                        1/1     Running   0          26s
pod/kibana-logging-546b56bf8c-zgcr5       1/1     Running   0          28s
pod/logstash-deployment-7f7f6f9bfb-m5cmg  1/1     Running   0          27s
pod/mongo-86567d97d6-hhnjr                1/1     Running   0          7d2h
pod/nginx-796d9485c7-m7f54                1/1     Running   0          6d7h
pod/v1-grafana-59f475cf94-lxxtd           1/1     Running   0          6d5h

NAME                            TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
service/elasticsearch-logging   ClusterIP   10.102.222.168   <none>        9200/TCP         27s
service/kibana-logging          NodePort    10.99.75.45      <none>        5601:32010/TCP   28s
service/kubernetes              ClusterIP   10.96.0.1        <none>        443/TCP          64d
service/logstash-service        ClusterIP   10.108.204.252   <none>        5044/TCP         28s
service/mongo                   ClusterIP   10.103.137.19    <none>        27017/TCP        8d
service/nginx                   ClusterIP   10.100.159.247   <none>        80/TCP           6d7h
service/v1-grafana              NodePort    10.99.135.91     <none>        80:30092/TCP     6d5h

NAME                         DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
daemonset.apps/filebeat      1         1         1       1            1           <none>          26s

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/kibana-logging          1/1     1            1           28s
deployment.apps/logstash-deployment     1/1     1            1           28s
deployment.apps/mongo                   1/1     1            1           8d
deployment.apps/nginx                   1/1     1            1           6d7h
deployment.apps/v1-grafana              1/1     1            1           6d5h

NAME                                               DESIRED   CURRENT   READY   AGE
replicaset.apps/kibana-logging-546b56bf8c          1         1         1       28s
replicaset.apps/logstash-deployment-7f7f6f9bfb     1         1         1       27s
replicaset.apps/mongo-854d54bc76                   0         0         0       8d
replicaset.apps/mongo-86567d97d6                   1         1         1       7d2h
replicaset.apps/nginx-796d9485c7                   1         1         1       6d7h
replicaset.apps/v1-grafana-59f475cf94              1         1         1       6d5h

NAME                                     READY   AGE
statefulset.apps/elasticsearch-logging   2/2     27s
ubuntu@AMCOP:~/snadeep/elk/taas$
```

You can see that the kibana service is configured as nodeport and the port exposed is 32010. You can login to the kibana dashboard and should see the indices that are created in the elasticsearch on clicking Discover. With these indices you can create an index pattern, and apply a filter to view specific logs.

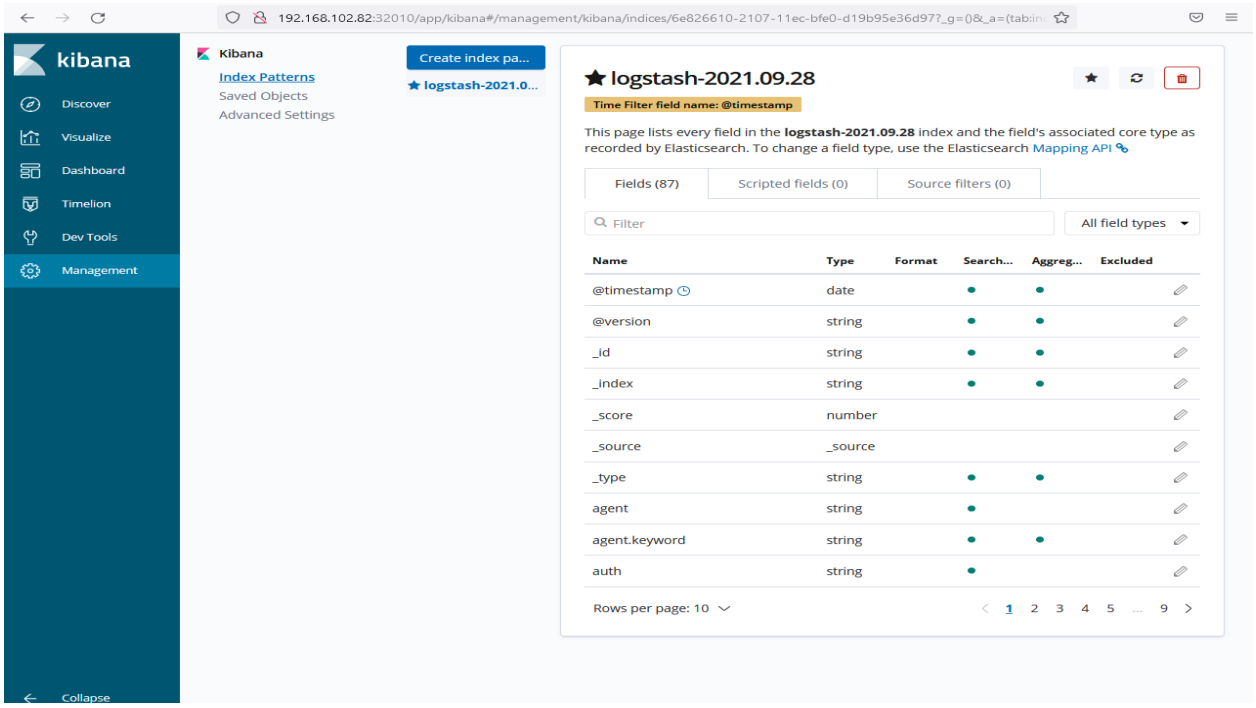


Kibana - Create Index pattern

Select an index from the list, like logstash-2021.09.25 has been selected in this example. And click Next step, select @timestamp in the ‘time filter field name’.



This should create our index pattern as shown below.



Appendix A - Free5GC & gNBSim Installation

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

gNbsim: gNbsim is 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 gNb simulator environment

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

- Download and extract the required files for gNb simulator.

```
mkdir -p /home/<user>/free5gc_deploy
cd /home/<user>/free5gc_deploy

# Download the install package using the following link:
free5gc304_gnbsim_deployment_files.zip

# Extract the .zip package

unzip  free5gc304_gnbsim_deployment_files.zip in the deployment host
```

- The following files will be extracted under `/home/<user>/free5c_deploy` directory
  - create_qem_vm.sh
  - free5g304-helm-0.1.0.tgz
  - free5g_install_prereq.sh
  - networkcrddef.yaml
  - example.go
  - profile.tar.gz
- 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>
```

- Check the kernel version. It should be 5.4. This is important for gNb simulator to run successfully.

```
uname -a
```

Install prerequisite

- Copy free5g_install_prereq.sh, networkcrddef.yaml and example.go files (from the host machine) onto the newly created VM under $HOME directory. You can use scp command to copy the files.

```
scp <file> ubuntu@<VM_IP>:/home/ubuntu
ssh ubuntu@<VM_IP>
```

- Log in to the newly created VM and perform the below steps.

- Open free5g_install_prereq.sh and modify the the networkcrddef.yaml file path (5th line) as follows:

```
CRD_FILE_PATH=$HOME/aarna-stream/cnf/cnfs-helm/networkcrddef.yaml
To
CRD_FILE_PATH=$HOME/networkcrddef.yaml

# Save the file
```

- Identify the network interface by running the following command
```
ip a
```

```
# This command will return all the interfaces available.
# VMs created by create_qem_vm.sh script will have "ens3"
# interface created by default
```

- Run free5g_install_prereq.sh as following

  ```
  bash -x ./free5g_install_prereq.sh ens3
  ```

  Note: this step will take ~5 minutes depending on the network bandwidth.

- Run `kubectl get pods --all-namespaces` to make sure the cluster is up and running.

The above steps will install the required packages and CNIs for Free5GC and gNb simulator. The CNIs used are Weave and Multus.

Free5gc Orchestration

This section provides steps to download and orchestrate Free5GC using AMCOP.

- Copy free5g304-helm-0.1.0.tgz and profile.tar.gz files (extracted from zip file in the deployment host) onto the machine where AMCOP UI is accessed from.
- Now users can go back here to design and instantiate free5g304.
- Reference link: Open source code base link is as follows: https://github.com/free5gc/free5gc

gNb Simulator deployment

This section provides steps to download, configure and run gNb Simulator as a standalone utility

- Make sure the example.go file is copied onto the target cluster server(the newly created VM under $HOME directory).
- Rename example.go file in the gNb simulator director as follows:

  ```
  cd gnbsim/example
  # Rename example.go file to example.go.bk
  ```
- Copy $HOME/example.go file to $HOME/gnbsim/example directory
- Follow the steps provided in AMCOP_2.0_User_Guide under section **Free5GC Validation using gNb Simulator** to run the simulator
- Reference link: Open source code base link is as follows
  - https://github.com/H21lab/gnbsim

Troubleshooting tips

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

1. Issue: Connection timeout issue. Logs will be something similar given below.
    a. `[gnbsim]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_1.0_User_Guide to populate the corresponding fields in example.json.

4. Issue: Resource busy. Logs will be something similar given below.
    a. `[gnbsim]2021/01/23 14:38:42.784491 example.go:321: failed to ADD tun device=gtp0: Tuntap IOCTL TUNSETIFF failed [0], errno device or resource busy`

    Possible root cause: The same gtp tunnel is used multiple times.
    Resolution: while running the command sudo ./example -ip 172.16.10.20 -test setupPDUSession -gtp **gtp0,** increment the gtp tunnel name that is highlighted in the command.

Appendix B - O1 Simulator Installation

This section describes the steps to deploy open-source O1 Simulators and their configurations which can be used to try out the MCOP SMO capabilities of AMCOP. The simulators are from the ORAN-SC community and currently include RU and DU simulators. You can pick any one of the below O1 simulator deployment options as per your requirements.

Option-1: O1 Simulator with configurations

This section explains the required steps to bring up RU and DU simulators with more features compared to the below options. Using this option, the O1 simulators automatically connect to SDNR after instantiation (plug-n-play). O1 simulators send VES notifications like PNF registrations and faults to the configured VES endpoint. So, you can try the SMO capability of AMCOP through O1 simulator plug n play and O1 simulator fault management.

Note: In case of nested deployment, where AMCOP VM is running inside another (base) VM, Use this option to deploy O1 simulators either on the same (base) host where AMCOP VM is running (or) inside the nested VM, where AMCOP is deployed. If you deployed AMCOP with skip VM tag (--skip-tags vm) option, then the O1 simulators can be deployed remotely either on any server reachable to the VM where AMCOP is deployed or on the server where AMCOP is deployed. Below table represents the details explained here:

	O1 simulator running on base VM support	O1 simulator running on AMCOP VM support	O1 simulator running on a remote server support
Nested VM AMCOP deployment	Yes	Yes	No
Optional AMCOP deployment with (--skip-tags vm)	Not Applicable	Yes	Yes

Prerequisite: docker, docker-compose

```
# To install supported docker-compose version (1.27.0), execute below command:
curl -L
"https://github.com/docker/compose/releases/download/1.27.0/docker-compose-$(u
name -s)-$(uname -m)" > ./docker-compose
```

```
sudo mv ./docker-compose /usr/bin/docker-compose

sudo chmod +x /usr/bin/docker-compose


# Download the zip
o1_simulators.zip

# Extract the zip file. It contains 2 files docker-compose.yaml and .env
unzip o1_simulators.zip

# Update .env file to have the correct IP addresses and configuration
#parameters as below. If o1 simulators are deployed either inside VM where
#AMCOP VM is running or inside the AMCOP VM (where AMCOP components are
#deployed), update "NTS_NF_MOUNT_POINT_ADDRESSING_METHOD" to "docker-mapping"
#and "SDN_CONTROLLER_IP" and "VES_ENDPOINT_IP" should be AMCOP VM IP (where
#AMCOP components are deployed).

# execute below command to bring up the simulators
docker-compose up -d

# simulator status check
docker-compose ps
```

If SDNR and VES endpoints are configured correctly, then simulators will connect automatically to SMO after instantiation and alerts appear on the Fault page of SMO GUI.

Option-2: Standalone O1 simulator

This section explains how to bring up the standalone O1 simulators with sample configurations. Netconf credentials to use for these simulators are: "*netconf/netconf!*"

Prerequisite: docker

```
# RU Simulator
docker run -d -p2300:830 -e
NTS_NF_STANDALONE_START_FEATURES="datastore-populate"
nexus3.o-ran-sc.org:10004/o-ran-sc/nts-ng-o-ran-ru-fh:1.3.1

# DU Simulator
docker run -d -p3300:830 -e
NTS_NF_STANDALONE_START_FEATURES="datastore-populate"
nexus3.o-ran-sc.org:10004/o-ran-sc/nts-ng-o-ran-du:1.3.1
```

Once the simulator containers are up and running, these O1 simulator containers can be manually added to SMO GUI using IP Address (VM IP where the O1 simulators are deployed), port (port exposed by the O1 docker containers, for DU element, it is 3300, for RU element, it is 2300) and netconf credentials (netconf/netconf!). So using this option, you can experience the SMO capability of AMCOP by manually adding O1 simulator interfaces through SMO UI.

	O1 simulator docker container running on base VM support	O1 simulator docker container running on AMCOP VM support	O1 simulator docker container running on a remote server support
Nested VM AMCOP deployment	Yes	Yes	Yes
Optional AMCOP deployment with (--skip-tags vm)	Not Applicable	Yes	Yes

Option-3: Standalone O1 simulator (using helm charts)

This section explains how to bring up the standalone O1 simulators with sample configurations using helm charts for O1 simulators. Netconf credentials to use for these simulators are: "*netconf/netconf!*"

Prerequisite: Target cluster (Kud) or any Kubernetes cluster.

```
# Download the helm charts

o1-sim.zip
# Untar the helm charts
unzip o1-sim.zip

# Deploy both DU and RU O1 Simulators after creating namespace o1sim

kubectl create ns o1sim
cd o1-sim/

helm install --namespace o1sim .
```

Once the simulator pods are up and running, these O1 simulators can be manually added to SMO GUI using IP Address (VM IP where the O1 simulators are deployed), port (port exposed

by the O1 simulator pod) and netconf credentials (netconf/netconf!). So using this option, you can experience the SMO capability of AMCOP by manually adding O1 simulator interfaces through SMO UI.

Note: You need to edit the deployed O1 simulator services to make these simulator pods expose external node ports which can be used to configure as part of the SMO GUI.

References

Following are some of the references on O1 simulators and corresponding documentation.

- https://wiki.o-ran-sc.org/pages/viewpage.action?pageId=33292606
- https://docs.o-ran-sc.org/projects/o-ran-sc-sim-o1-interface/en/latest/index.html
- https://gerrit.o-ran-sc.org/r/admin/repos/sim/o1-interface
- https://github.com/Melacon/ntsim/tree/master/ntsimulator