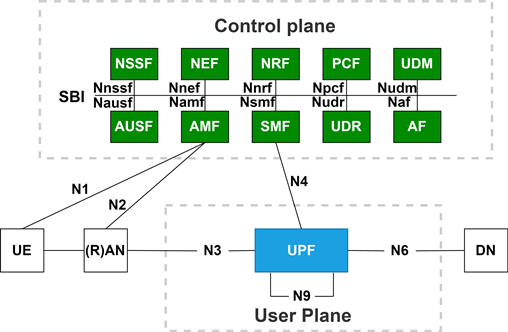
**5G Architecture & NEPHIO**

**5g Network Architecture:**



**UE**: User Equipment

**NG-RAN (NEW GENERATION RAN or gNODEB or NR):**

* A gNodeB, also called NR in 5G, is like a special tower for your mobile devices. Its job is to send and receive signals to connect your phone, tablet, or other gadgets to the internet.
* The gNodeB is responsible for providing radio access to user devices. It transmits and receives radio signals, establishing wireless links with devices in its coverage area.
* 5G supports a wide range of frequency bands, including sub-6 GHz and millimeter-wave (mmWave) frequencies. The gNodeB is designed to operate in various frequency bands to provide different coverage and capacity options.
* The gNodeB can employ beamforming techniques to improve signal quality and coverage.
* gNodeBs often feature advanced antenna systems with multiple transmit and receive antennas
* gNodeBs play a crucial role in ensuring low-latency communication in 5G networks, which is essential for applications like augmented reality, virtual reality, and real-time industrial automation.
* gNodeBs are connected to the core network, which includes elements like the AMF, SMF, and UPF. This connection allows gNodeBs to route traffic and establish end-to-end connections for user devices.

**AMF- Access and Mobility Function:**

* The AMF is a core network function responsible for managing user access to the 5G network and ensuring seamless mobility between different cells and network areas.
* It handles functions such as user authentication, authorization, and accounting (AAA), registration, and session management.
* AMF plays a crucial role in setting up and maintaining connections for mobile devices as they move within the network, providing continuous connectivity.

**SMF- Session Management Function:**

* The SMF is responsible for managing the data sessions for user devices in 5G networks.
* It handles session establishment, modification, and termination for both the user plane and control plane.
* SMF plays a central role in optimizing data transmission, ensuring low latency, and providing Quality of Service (QoS) management for various types of services, including enhanced mobile broadband, massive machine-type communication, and ultra-reliable low-latency communication.

**UPF- User Plane Function:**

* The UPF is responsible for handling the user data plane in 5G networks.
* It plays a crucial role in routing user data packets efficiently between the user device and external services or the internet.
* UPF is responsible for data traffic optimization, traffic shaping, security, and ensuring that data packets are transmitted with minimal delay and maximum throughput.
* It is a key component in delivering the low-latency and high-speed data services promised by 5G networks.

**Call flow in 5g in simple terms:**

* The UE (user device) powers on and establishes an initial connection with the nearest gNodeB.
* The UE initiates a registration request to the gNodeB.
* The gNodeB forwards the request to the AMF.
* AMF verifies the UE's identity and authorizes its access.
* When the UE wants to access a data service (e.g., internet browsing), it requests a data session.
* The request goes to the SMF, which sets up the session parameters.
* When data is sent or received by the UE, the UPF routes the data packets between the UE and the external data network.
* If the UE moves and connects to a different gNodeB, the AMF manages the handover process, ensuring uninterrupted service.
* When the UE finishes using the service, the SMF terminates the data session.
* Finally, the UE may initiate a release request, and the AMF and gNodeB release the resources allocated for the UE.

**O-RAN(OPEN-RAN):**

* It is an open and flexible approach to building radio access networks. It consists of components like:
* O-RU (Open Radio Unit): The O-RU is the radio unit responsible for transmitting and receiving radio signals. It's essentially the hardware that communicates wirelessly with user devices.
* O-DU (Open Distributed Unit): The O-DU is responsible for processing and forwarding radio signals. It's the counterpart to the gNodeB in traditional 5G networks.
* O-CU (Open Centralized Unit): The O-CU is responsible for centralized functions like baseband processing, which includes functions like modulation/demodulation and encoding/decoding of data.
* O-RAN Controller: This component acts as the brain of the O-RAN architecture. It manages the O-RUs, O-DUs, and O-CUs, making decisions about how to allocate resources and optimize network performance.
* X2 Interface: The X2 interface connects different O-RAN elements to facilitate communication and coordination between them.

**O-RAN vs GNodeB:**

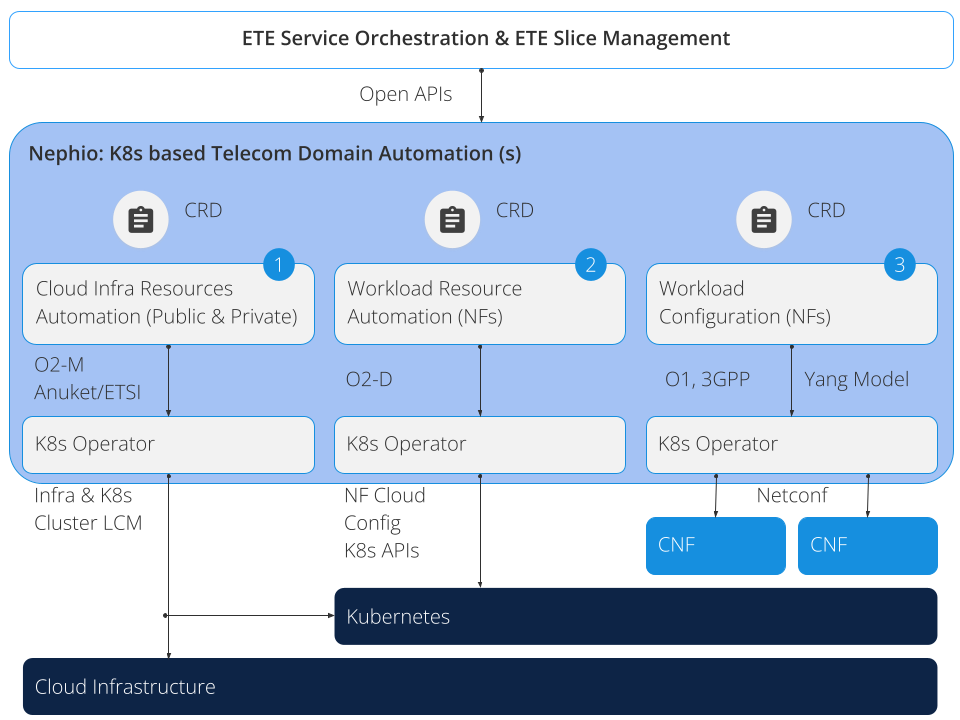
* O-RAN is designed to be more open and interoperable. It encourages the use of standardized interfaces, making it easier for operators to mix and match equipment from different vendors. In contrast, gNodeB in traditional 5G networks often relies on proprietary interfaces.
* It promotes vendor neutrality, allowing operators to choose hardware and software components from different vendors. This can lead to cost savings and increased flexibility. In traditional 5G, gNodeBs are often tied to specific vendors.
* It decomposes the radio access network into modular components (O-RU, O-DU, O-CU), which can be flexibly configured and combined. In contrast, gNodeB in traditional 5G combines several functions into a single unit.
* O-RAN leverages Software-Defined Networking and Network Functions Virtualization principles, allowing network functions to be implemented in software and dynamically controlled. Traditional gNodeBs may have more static hardware-based configurations.
* It aims to lower infrastructure costs and promote innovation by fostering a competitive ecosystem of hardware and software providers. Traditional 5G gNodeBs can be more closed and cost- prohibitive.

**NEPHIO:**

* Nephio is a Kubernetes-based intent-driven automation tool for network functions.
* It allows users to express high-level intent, that can set up the desired infrastructure and render initial configurations for the network functions in right clusters to get the network up and running.
* It uses Kubernetes to handle changes and issues in the cloud, ensuring your network stays up and running, even in complex setups.
* It uses new approaches that can handle the complexity of provisioning and managing a multi-vendor, multi-site deployment of interconnected network functions across on-demand distributed cloud.

**Nephio breaks down the larger problem into two primary areas:**

* Using Kubernetes as control plane to control everything in each site to configure distributed cloud and network functions.
* Using an automation framework with Kubernetes declarative approach to make complex configurations easier to handle.

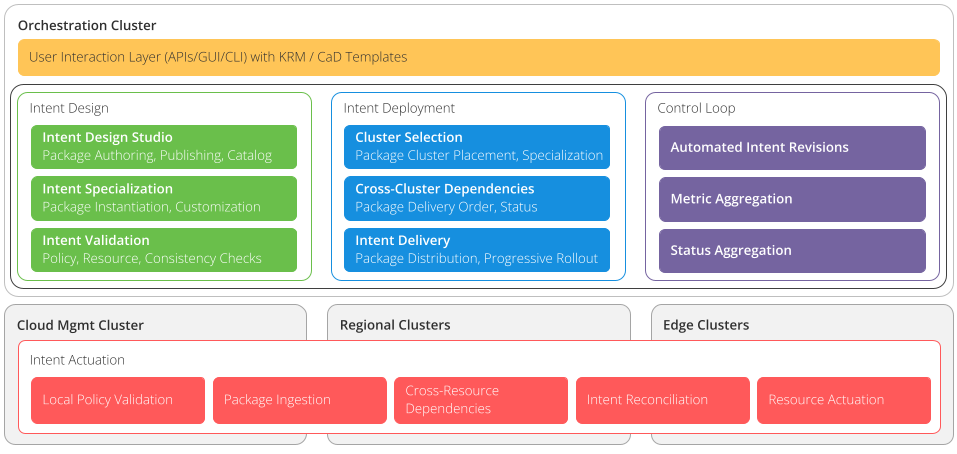


Using Kubernetes simplifies automation across three layers: **Cloud Infrastructure**, **Workload Resources**, and **Workload Configuration.** Nephio creates Kubernetes Custom Resource Definitions (CRDs) for each layer, following 3GPP & O-RAN standards.

* For **Cloud Infrastructure (1)**, Nephio offers Kubernetes CRDs and operators for public and private clouds, ensuring industry standard compliance.
* **Workload resource automation (2)** includes configuring network function containers and related requirements. Instead of complex templates, Nephio uses Kubernetes CRDs with well-structured schemas for standardized automation.
* **Workload configuration (3)** integration involves tooling for vendors to align with standards. Ultimately, these models should transition to Kubernetes CRDs for seamless inter-layer automation. Nephio provides tooling for automation at each layer.

**IN Nephio R1:**

* **Cloud infrastructure (1)** lane is used for cluster provisioning
* **Workload resource (2)** lane for network functions provisioning and supporting workload for network functions
* **Workload configuration (3)** lane we are building a Kubernetes operator that takes intent (kubernetes resource) which is basic configuration that gets delivered to workload clusters and the operator running on that workload cluster uses that intent to render the deployment,services and configmap for network functions.



**From the above figure:**

* **Intent Actuation** layeris a Uniform Automation control plane running on site clusters.
* **Orchestration Cluster** is a Kubernetes cluster for housing the automation framework.
* **Nephio's Automation** **Framework** uses Google's **kpt** and **ConfigSync** for Configuration-as-Data, allowing users to create, customize, and deploy network functions easily without conflicts and enabling seamless upgrades.

**KPT:**

kpt is a package manager for k8s alternative to our traditional helm charts built by Google. kpt lets you reuse and share packages across all environments. We can perform operations like Package k8s configs, pull those from GitHub, with a single command update all files in the package, and again update the latest files in GitHub.

**Config sync:**

Config sync refers to a configuration management approach or tool used to ensure that the desired configuration for your Kubernetes cluster is in sync with the actual state of the cluster. Config sync ensures that the configuration of your Kubernetes resources, such as pods, services, and ConfigMaps, matches the desired state specified in your configuration files or manifests. It helps you define and declare your desired state and then enforces it on the cluster.

**Nephio R1 Demo:**

We are using a single VM environment for demo whose prerequisites are:

* Linux Flavour: Ubuntu-20.04-focal
* 8 cores
* 32 GB memory
* 200 GB disk size
* Default user with sudo passwordless permissions

**To insatll Nephio on your VM simply use this command**

wget -O - https://raw.githubusercontent.com/nephio-project/test-infra/v1.0.1/e2e/provision/init.sh | \

sudo NEPHIO\_DEBUG=false \

NEPHIO\_BRANCH=v1.0.1 \

NEPHIO\_USER=ubuntu \

bash

**After installation, to browse Nephio web UI and gitea UI we need to use port forwarding.**

Use the below command after changing the user and ip address.

ssh <user>@<vm-address> \

-L 7007:localhost:7007 \

-L 3000:172.18.0.200:3000 \

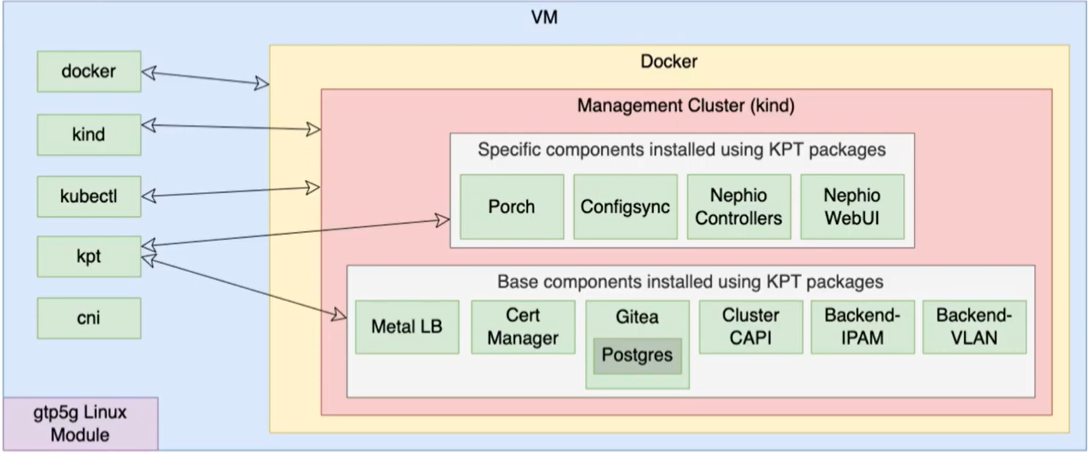
kubectl port-forward --namespace=nephio-webui svc/nephio-webui 7007

After port forwarding, use the following url’s

**Nephio web UI:** http://localhost:7007/config-as-data

**Gitea UI:**  <http://localhost:3000/nephio>

After the deployment a management cluster gets created. The services in the management cluster are shown in the figure below.

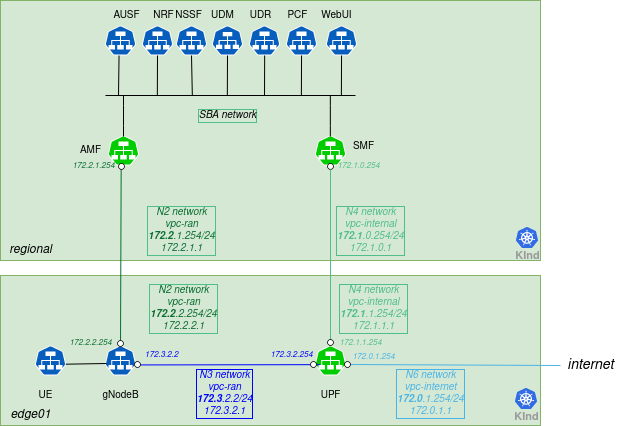


Some other services like “metalLB(load balancer), gitea(self-hosted Git server), cluster API (for cluster provisioning) and the backend resource allocators (IPAM and VLAN for ip allocation automation)” can be replaced with external services.

**After the nephio deployment we are we will set up the following:**

* A Regional cluster
* Two Edge clusters
* Repositories for each cluster, registered with Nephio, and with Config Sync set up to pull from those repositories.
* Inter-cluster networking between those clusters
* A complete free5gc deployment including:
* AUSF, NRF, NSSF, PCF, UDM, UDR running on the regional cluster and communicating via the Kubernetes default network
* SMF, AMF running on the regional cluster and attached to the secondary Multus networks as needed
* UPF running on the Edge clusters and attached to the secondary Multus networks as needed
* The free5gc WebUI and MongoDB as supporting services
* A registered subscriber in the free5gc core
* UERANSIM running on the edge01 cluster and simulating a gNB and the subscriber's UE

**The network architecture of the free5gc and UERANSIM deployment using Nephio looks like:**



Note: Only one edge cluster is represented above. Edge02 is also created in general.

After installing nephio to start the Nephio R1 demo we need to following steps:

## **Create the Regional Cluster**

First, check to make sure that both the mgmt and mgmt-staging repositories are in the Ready state. The mgmt repository is used to manage the contents of the Management cluster via Nephio; the mgmt-staging repository is just used internally during the cluster bootstrapping process.

**# kubectl get repositories**

Output is similar to:

NAME TYPE CONTENT DEPLOYMENT READY ADDRESS

free5gc-packages git Package false True https://github.com/nephio-project/free5gc-packages.git

mgmt git Package true True http://172.18.0.200:3000/nephio/mgmt.git

mgmt-staging git Package false True http://172.18.0.200:3000/nephio/mgmt-staging.git

nephio-example-packages git Package false True <https://github.com/nephio-project/nephio-example-packages.git>

Since those are Ready, you can deploy a package from the nephio-example-packages repository into the mgmt repository. To do this, you retrieve the Package Revision name using kpt alpha rpkg get, clone that specific Package Revision via the kpt alpha rpkg clone command, then propose and approve the resulting package revision. You want to use the latest revision of the nephio-workload-cluster package, which you can get with the command below (your latest revision may be different):

**# kpt alpha rpkg get --name nephio-workload-cluster**

Output is similar to

NAME PACKAGE WORKSPACENAME REVISION LATEST LIFECYCLE REPOSITORY

nephio-example-packages-05707c7acfb59988daaefd85e3f5c299504c2da1 nephio-workload-cluster main main false Published nephio-example-packages

nephio-example-packages-781e1c17d63eed5634db7b93307e1dad75a92bce nephio-workload-cluster v1 v1 false Published nephio-example-packages

nephio-example-packages-5929727104f2c62a2cb7ad805dabd95d92bf727e nephio-workload-cluster v2 v2 false Published nephio-example-packages

nephio-example-packages-cdc6d453ae3e1bd0b64234d51d575e4a30980a77 nephio-workload-cluster v3 v3 false Published nephio-example-packages

nephio-example-packages-c78ecc6bedc8bf68185f28a998718eed8432dc3b nephio-workload-cluster v4 v4 false Published nephio-example-packages

nephio-example-packages-46b923a6bbd09c2ab7aa86c9853a96cbd38d1ed7 nephio-workload-cluster v5 v5 false Published nephio-example-packages

nephio-example-packages-17bffe318ac068f5f9ef22d44f08053e948a3683 nephio-workload-cluster v6 v6 false Published nephio-example-packages

nephio-example-packages-0fbaccf6c5e75a3eff7976a523bb4f42bb0118ce nephio-workload-cluster v7 v7 false Published nephio-example-packages

nephio-example-packages-7895e28d847c0296a204007ed577cd2a4222d1ea nephio-workload-cluster v8 v8 false Published nephio-example-packages

nephio-example-packages-48cea934a3bd876b775099ab59e7c12456888ffd nephio-workload-cluster v9 v9 true Published nephio-example-packages

Then, use the NAME from that in the clone operation, and the resulting PackageRevision name to perform the propose and approve operations:

**# kpt alpha rpkg clone -n default nephio-example-packages-48cea934a3bd876b775099ab59e7c12456888ffd --repository mgmt regional**

Next, you will want to ensure that the new Regional cluster is labeled as regional. Since you are using the CLI, you will need to pull the package out, modify it, and then push the updates back to the Draft revision. You will use kpt and the set-labels function to do this.

To pull the package to a local directory, use the rpkg pull command:

**# kpt alpha rpkg pull -n default mgmt-08c26219f9879acdefed3469f8c3cf89d5db3868 regional**

The package is now in the regional directory. So you can execute the set-labels function against the package imperatively, using kpt fn eval:

**# kpt fn eval --image gcr.io/kpt-fn/set-labels:v0.2.0 regional -- "nephio.org/site-type=regional" "nephio.org/region=us-west1"**

If you wanted to, you could have used the --save option to add the set-labels call to the package pipeline. This would mean that function gets called whenever the server saves the package. If you added new resources later, they would also get labeled.

In any case, you now can push the package with the labels applied back to the repository:

**# kpt alpha rpkg push -n default mgmt-08c26219f9879acdefed3469f8c3cf89d5db3868 regional**

Finally, you propose and approve the package.

**# kpt alpha rpkg propose -n default mgmt-08c26219f9879acdefed3469f8c3cf89d5db3868**

**# kpt alpha rpkg approve -n default mgmt-08c26219f9879acdefed3469f8c3cf89d5db3868**

**Check the Regional Cluster Installation**

**# kubectl get cl**

NOTE: cluster may take 5 to 10 minutes to get created please wait

Out is similar to

NAME PHASE AGE VERSION

regional Provisioned 52m v1.26.3

To access the API server of that cluster, you need to retrieve the kubeconfig file by pulling it from the Kubernetes Secret and decode the base64 encoding:

# **kubectl get secret regional-kubeconfig -o jsonpath='{.data.value}' | base64 -d > $HOME/.kube/regional-kubeconfig**

**# export KUBECONFIG=$HOME/.kube/config:$HOME/.kube/regional-kubeconfig**

You can then use it to access the Workload cluster directly:

**# kubectl get ns --context regional-admin@regional**

The output is similar to:

NAME STATUS AGE  
config-management-monitoring Active 3h35m  
config-management-system Active 3h35m  
default Active 3h39m  
kube-node-lease Active 3h39m  
kube-public Active 3h39m  
kube-system Active 3h39m

You should also check that the KinD cluster has come up fully with kubectl get machinesets. You should see READY and AVAILABLE replicas.

**# kubectl get machinesets**

The output is similar to:

NAME CLUSTER REPLICAS READY AVAILABLE AGE VERSION  
regional-md-0-zhw2j-58d497c498xkz96z regional 1 1 1 3h58m v1.26.3

**Deploy two Edge clusters**

Next, you can deploy two Edge clusters by applying the PackageVariantSet that can be found in the tests directory:

**# kubectl apply -f test-infra/e2e/tests/002-edge-clusters.yaml**

You should also check that the KinD cluster has come up fully with kubectl get machinesets. You should see READY and AVAILABLE replicas.

**# kubectl get machinesets**

NOTE: cluster may take 5 to 10 minutes to get created please wait

The output is similar to:

NAME CLUSTER REPLICAS READY AVAILABLE AGE VERSION  
edge01-md-0-p5vwv-98cb4b55cx58l8l edge01 1 1 1 114m v1.26.3  
edge02-md-0-4nfpb-797dc6ddd7x8fc56 edge02 1 1 1 114m v1.26.3

To access the API server of these clusters, you will need to get the kubeconfig file. To retrieve the file, you pull it from the Kubernetes Secret and decode the base64 encoding:

**# kubectl get secret edge01-kubeconfig -o jsonpath='{.data.value}' | base64 -d > $HOME/.kube/edge01-kubeconfig**

**# kubectl get secret edge02-kubeconfig -o jsonpath='{.data.value}' | base64 -d > $HOME/.kube/edge02-kubeconfig**

**# export KUBECONFIG=$HOME/.kube/config:$HOME/.kube/regional-kubeconfig:$HOME/.kube/edge01-kubeconfig:$HOME/.kube/edge02-kubeconfig**

To retain the KUBECONFIG environment variable permanently across sessions for the user, add it to the ~/.bash\_profile and source the ~/.bash\_profile file

**# echo "export KUBECONFIG=$HOME/.kube/config:$HOME/.kube/regional-kubeconfig:$HOME/.kube/edge01-kubeconfig:$HOME/.kube/edge02-kubeconfig" >> ~/.bash\_profile**

**# source ~/.bash\_profile**

Once the Edge clusters are ready, it is necessary to connect them. For now you are using the containerlab tool Eventually, the inter-cluster networking will be automated as well.

**# ./test-infra/e2e/provision/hacks/inter-connect\_workers.sh**

You will also need to configure the nodes for the VLANs. Again, this will be automated in a future release that addresses node setup and inter-cluster networking. For now, you must run a script that creates them in each of the worker nodes.

**# ./test-infra/e2e/provision/hacks/vlan-interfaces.sh**

Finally, you want to configure the resource backend to be aware of these clusters. The resource backend is an IP address and VLAN index management system. It is included for demonstration purposes to show how Nephio package specialization can interact with external systems to fully configure packages. But it needs to be configured to match our topology.

First, you will apply a package to define the high-level networks for attaching our workloads. The Nephio package specialization pipeline will determine the exact VLAN tags and IP addresses for those attachments based on the specific clusters. There is a predefined PackageVariant in the tests directory for this:

**# kubectl apply -f test-infra/e2e/tests/003-network.yaml**

Then you will create appropriate Secret to make sure that Nephio can authenticate to the external backend.

**# kubectl apply -f test-infra/e2e/tests/003-secret.yaml**

The predefined PackageVariant package defines certain resources that exist for the entire topology. However, you also need to configure the resource backend for our particular topology. This will likely be automated in the future, but for now you can just directly apply the configuration you have created that matches this test topology. Within this step also the credentials and information is provided to configure the network device, that aligns with the topology.

**# ./test-infra/e2e/provision/hacks/network-topo.sh**

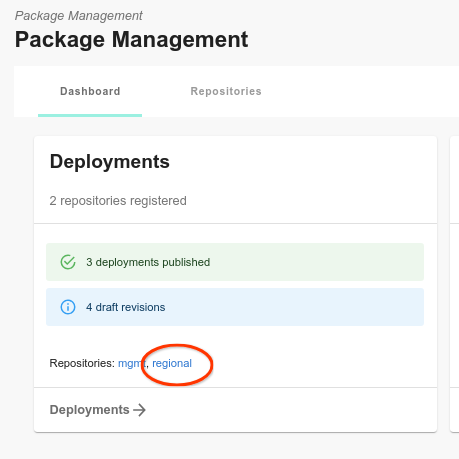
**# kubectl apply -f test-infra/e2e/tests/003-network-topo.yaml**

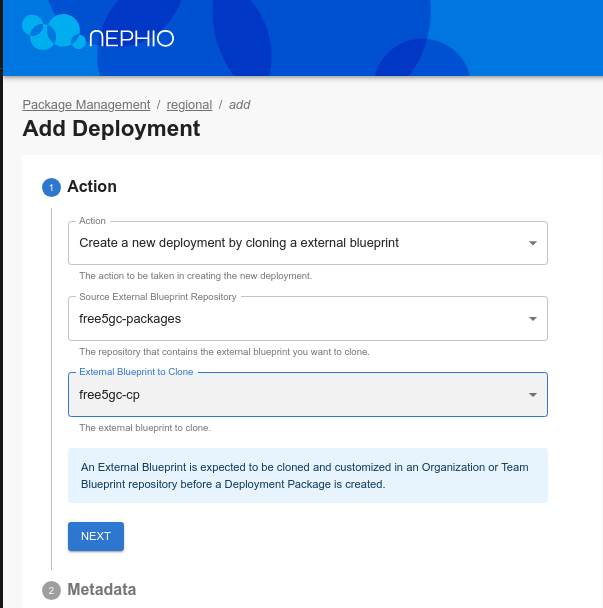
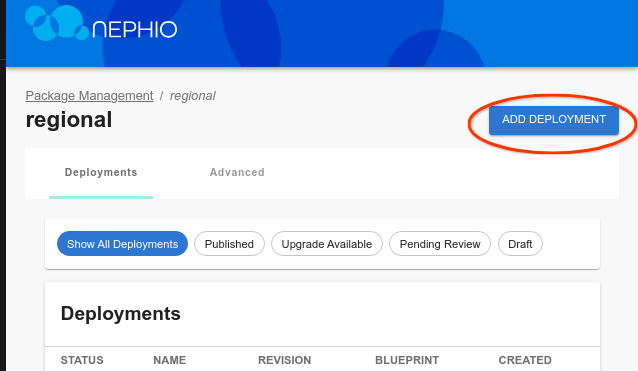
The output is similar to:

rawtopology.topo.nephio.org/nephio created

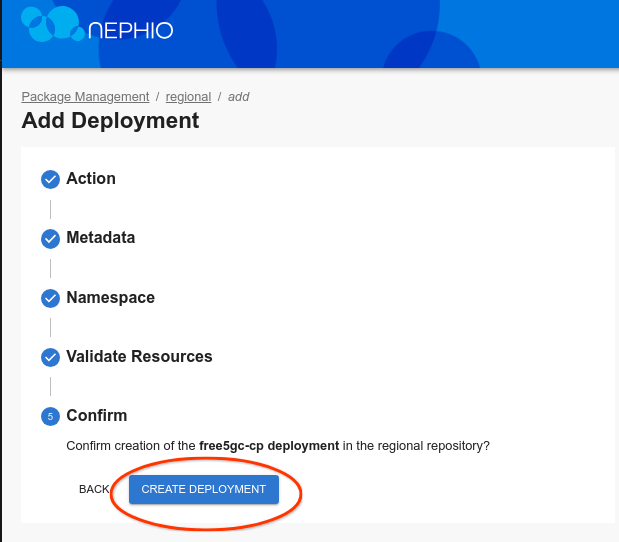
**Deploy Free5GC Control Plane Functions**

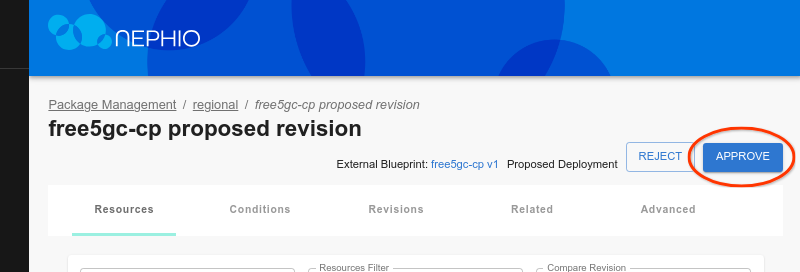
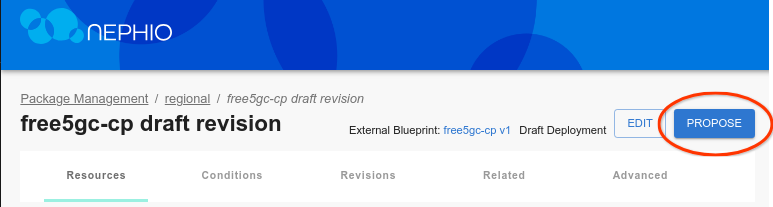
While the Edge clusters are deploying (which will take 5-10 minutes), you can install the free5gc functions other than SMF, AMF, and UPF. For this, you will use the Regional cluster. Since these are all installed with a single package, you can use the UI to pick the free5gc-cp package from the free5gc-packages repository and clone it to the regional repository (you could have also used the CLI).





Click through the "Next" button until you are through all the steps, then click "Add Deployment". On the next screen, click "Propose", and then "Approve".





Now in cli use the following commands, you should see free5gc-cp in the cluster namespace:

**# kubectl get ns --context regional-admin@regional**

The output is similar to:

NAME STATUS AGE  
config-management-monitoring Active 28m  
config-management-system Active 28m  
default Active 28m  
free5gc-cp Active 3m16s  
kube-node-lease Active 28m  
kube-public Active 28m  
kube-system Active 28m  
local-path-storage Active 28m  
resource-group-system Active 27m

And the actual workload resources:

**# kubectl -n free5gc-cp get all --context regional-admin@regional**

The output is similar to:

NAME READY STATUS RESTARTS AGE  
pod/free5gc-ausf-7d494d668d-k55kb 1/1 Running 0 3m31s  
pod/free5gc-nrf-66cc98cfc5-9mxqm 1/1 Running 0 3m31s  
pod/free5gc-nssf-668db85d54-gsnqw 1/1 Running 0 3m31s  
pod/free5gc-pcf-55d4bfd648-tk9fs 1/1 Running 0 3m31s  
pod/free5gc-udm-845db6c9c8-54tfb 1/1 Running 0 3m31s  
pod/free5gc-udr-79466f7f86-wh5bt 1/1 Running 0 3m31s  
pod/free5gc-webui-84ff8c456c-g7q44 1/1 Running 0 3m31s  
pod/mongodb-0 1/1 Running 0 3m31s  
  
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE  
service/ausf-nausf ClusterIP 10.131.151.99 <none> 80/TCP 3m32s  
service/mongodb ClusterIP 10.139.208.189 <none> 27017/TCP 3m32s  
service/nrf-nnrf ClusterIP 10.143.64.94 <none> 8000/TCP 3m32s  
service/nssf-nnssf ClusterIP 10.130.139.231 <none> 80/TCP 3m31s  
service/pcf-npcf ClusterIP 10.131.19.224 <none> 80/TCP 3m31s  
service/udm-nudm ClusterIP 10.128.13.118 <none> 80/TCP 3m31s  
service/udr-nudr ClusterIP 10.137.211.80 <none> 80/TCP 3m31s  
service/webui-service NodePort 10.140.177.70 <none> 5000:30500/TCP 3m31s  
  
NAME READY UP-TO-DATE AVAILABLE AGE  
deployment.apps/free5gc-ausf 1/1 1 1 3m31s  
deployment.apps/free5gc-nrf 1/1 1 1 3m31s  
deployment.apps/free5gc-nssf 1/1 1 1 3m31s  
deployment.apps/free5gc-pcf 1/1 1 1 3m31s  
deployment.apps/free5gc-udm 1/1 1 1 3m31s  
deployment.apps/free5gc-udr 1/1 1 1 3m31s  
deployment.apps/free5gc-webui 1/1 1 1 3m31s  
  
NAME DESIRED CURRENT READY AGE  
replicaset.apps/free5gc-ausf-7d494d668d 1 1 1 3m31s  
replicaset.apps/free5gc-nrf-66cc98cfc5 1 1 1 3m31s  
replicaset.apps/free5gc-nssf-668db85d54 1 1 1 3m31s  
replicaset.apps/free5gc-pcf-55d4bfd648 1 1 1 3m31s  
replicaset.apps/free5gc-udm-845db6c9c8 1 1 1 3m31s  
replicaset.apps/free5gc-udr-79466f7f86 1 1 1 3m31s  
replicaset.apps/free5gc-webui-84ff8c456c 1 1 1 3m31s  
  
NAME READY AGE  
statefulset.apps/mongodb 1/1 3m31s

**Deploy Free5GC Operator in Workload Clusters**

Now you will need to deploy the free5gc operator across all of the Workload clusters (regional and edge). To do this, you use another PackageVariantSet. This one uses an objectSelector to select the WorkloadCluster resources previously added to the Management cluster when you had deployed the nephio-workload-cluster packages (manually as well as via PackageVariantSet).

**# kubectl apply -f test-infra/e2e/tests/004-free5gc-operator.yaml**

The output is similar to:

packagevariantset.config.porch.kpt.dev/free5gc-operator created

**Check Free5GC Operator Deployment**

Within five minutes of applying the free5gc Operator YAML file, you should see free5gc namespaces on your regional and edge clusters:

**# kubectl get ns --context edge01-admin@edge01**

The output is similar to:

NAME STATUS AGE  
config-management-monitoring Active 3h46m  
config-management-system Active 3h46m  
default Active 3h47m  
free5gc Active 159m  
kube-node-lease Active 3h47m  
kube-public Active 3h47m  
kube-system Active 3h47m  
resource-group-system Active 3h45m

**# kubectl -n free5gc get all --context edge01-admin@edge01**

The output is similar to:

NAME READY STATUS RESTARTS AGE  
pod/free5gc-operator-controller-controller-58df9975f4-sglj6 2/2 Running 0 164m  
  
NAME READY UP-TO-DATE AVAILABLE AGE  
deployment.apps/free5gc-operator-controller-controller 1/1 1 1 164m  
  
NAME DESIRED CURRENT READY AGE  
replicaset.apps/free5gc-operator-controller-controller-58df9975f4 1 1 1 164m

**Deploy UPF, AMF and SMF NFs**

Finally, you can deploy the individual network functions which the operator will instantiate. For now, you will use individual PackageVariants targeting the Regional cluster for each of the AMF and SMF NFs and a PackageVariantSet targeting the Edge clusters for the UPF NFs. In the future, you could put all of these resources into yet-another-package - a "topology" package - and deploy them all as a unit. Or you can use a topology controller to create them. But for now, let's do each manually.

**# kubectl apply -f test-infra/e2e/tests/005-edge-free5gc-upf.yaml**

**# kubectl apply -f test-infra/e2e/tests/006-regional-free5gc-amf.yaml**

**# kubectl apply -f test-infra/e2e/tests/006-regional-free5gc-smf.yaml**

Free5gc requires that the SMF and AMF NFs be explicitly configured with information about each UPF. Therefore, the AMF and SMF packages will remain in an "unready" state until the UPF packages have all been published.

### **Check UPF deployment**

You can check the UPF logs in edge01 cluster:

**# UPF1\_POD=$(kubectl get pods -n free5gc-upf -l name=upf-edge01 --context edge01-admin@edge01 -o jsonpath='{.items[0].metadata.name}')**

**# kubectl -n free5gc-upf logs $UPF1\_POD --context edge01-admin@edge01**

The output is similar to:

2023-07-15T09:05:51Z [INFO][UPF][Cfg] ==================================================  
2023-07-15T09:05:51Z [INFO][UPF][Main] Log level is set to [info] level  
2023-07-15T09:05:51Z [INFO][UPF][Main] starting Gtpu Forwarder [gtp5g]  
2023-07-15T09:05:51Z [INFO][UPF][Main] GTP Address: "172.3.0.254:2152"  
2023-07-15T09:05:51Z [INFO][UPF][Buff] buff server started  
2023-07-15T09:05:51Z [INFO][UPF][Pfcp][172.1.1.254:8805] starting pfcp server  
2023-07-15T09:05:51Z [INFO][UPF][Pfcp][172.1.1.254:8805] pfcp server started  
2023-07-15T09:05:51Z [INFO][UPF][Main] UPF started  
2023-07-15T09:10:45Z [INFO][UPF][Pfcp][172.1.1.254:8805] handleAssociationSetupRequest  
2023-07-15T09:10:45Z [INFO][UPF][Pfcp][172.1.1.254:8805][rNodeID:172.1.0.254] New node

### **Check AMF deployment**

You can check the AMF logs:

**# AMF\_POD=$(kubectl get pods -n free5gc-cp -l name=amf-regional --context regional-admin@regional -o jsonpath='{.items[0].metadata.name}')**

**# kubectl -n free5gc-cp logs $AMF\_POD --context regional-admin@regional**

The output is similar to:

2023-07-15T09:08:55Z [INFO][AMF][App] AMF version:  
 free5GC version: v3.2.1  
 build time: 2023-06-09T16:40:22Z  
 commit hash: a3bd5358  
 commit time: 2022-05-01T14:58:26Z  
 go version: go1.20.5 linux/amd64  
2023-07-15T09:08:55Z [INFO][AMF][Init] Server started  
2023-07-15T09:08:55Z [INFO][AMF][Util] amfconfig Info: Version[1.0.3] Description[AMF initial local configuration]  
2023-07-15T09:08:55Z [INFO][AMF][NGAP] Listen on 172.2.2.254:38412

### **Check SMF deployment**

You can check the SMF logs:

**# SMF\_POD=$(kubectl get pods -n free5gc-cp -l name=smf-regional --context regional-admin@regional -o jsonpath='{.items[0].metadata.name}')**

**# kubectl -n free5gc-cp logs $SMF\_POD --context regional-admin@regional**

The output is similar to:

2023-07-15T09:10:45Z [INFO][SMF][CTX] smfconfig Info: Version[1.0.2] Description[SMF configuration]  
2023-07-15T09:10:45Z [INFO][SMF][CTX] Endpoints: [172.3.0.254]  
2023-07-15T09:10:45Z [INFO][SMF][CTX] Endpoints: [172.3.1.254]  
2023-07-15T09:10:45Z [INFO][SMF][Init] Server started  
2023-07-15T09:10:45Z [INFO][SMF][Init] SMF Registration to NRF {7011c946-4ca4-45ff-bac6-32116bd93934 SMF REGISTERED 0 0xc0001da168 0xc0001da198 [] [] [smf-regional] [] <nil> [] [] <nil> 0 0 0 area1 <nil> <nil> <nil> <nil> 0xc0000b81c0 <nil> <nil> <nil> <nil> <nil> map[] <nil> false 0xc0001da060 false false []}  
2023-07-15T09:10:45Z [INFO][SMF][PFCP] Listen on 172.1.0.254:8805  
2023-07-15T09:10:45Z [INFO][SMF][App] Sending PFCP Association Request to UPF[172.1.1.254]  
2023-07-15T09:10:45Z [INFO][LIB][PFCP] Remove Request Transaction [1]  
2023-07-15T09:10:45Z [INFO][SMF][App] Received PFCP Association Setup Accepted Response from UPF[172.1.1.254]  
2023-07-15T09:10:45Z [INFO][SMF][App] Sending PFCP Association Request to UPF[172.1.2.254]  
2023-07-15T09:10:45Z [INFO][LIB][PFCP] Remove Request Transaction [2]  
2023-07-15T09:10:45Z [INFO][SMF][App] Received PFCP Association Setup Accepted Response from UPF[172.1.2.254]

NOTE: deployment may take 5 to 10 minutes to get created please wait

**Deploy UERANSIM**

The UERANSIM package can be deployed to the edge01 cluster, where it will simulate a gNodeB and UE. Just like our other packages, UERANSIM needs to be configured to attach to the correct networks and use the correct IP address. Thus, you use our standard specialization techniques and pipeline to deploy UERANSIM, just like the other network functions.

However, before you do that, let us register the UE with free5gc as a subscriber. You will use the free5gc Web UI to do this. To access it, you need to open another port forwarding session. Assuming you have the regional-kubeconfig file you created earlier in your home directory, you need to establish another ssh session from your workstation to the VM, port forwarding port 5000.

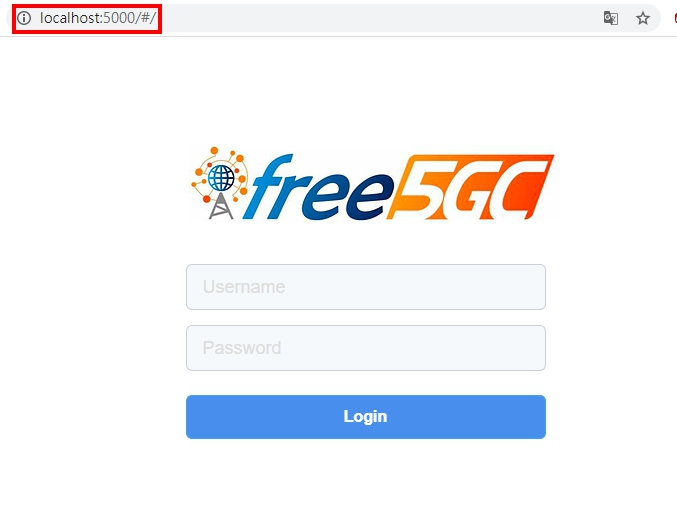
Before moving on to the new terminal, let's copy regional-kubeconfig to the home directory:

**# cp $HOME/.kube/regional-kubeconfig .**

Then, use the same form of the command you did for other sessions. For example, for the generic Ubuntu VM, the command would be:

**# ssh <user>@<vm-address> \**  
 **-L 5000:localhost:5000 \**  
 **kubectl --kubeconfig regional-kubeconfig \**  
 **port-forward --namespace=free5gc-cp svc/webui-service 5000**

You should now be able to navigate to[**http://localhost:5000**](http://localhost:5000)and access the free5gc WebUI. The test subscriber is the same as the standard free5gc default subscriber.



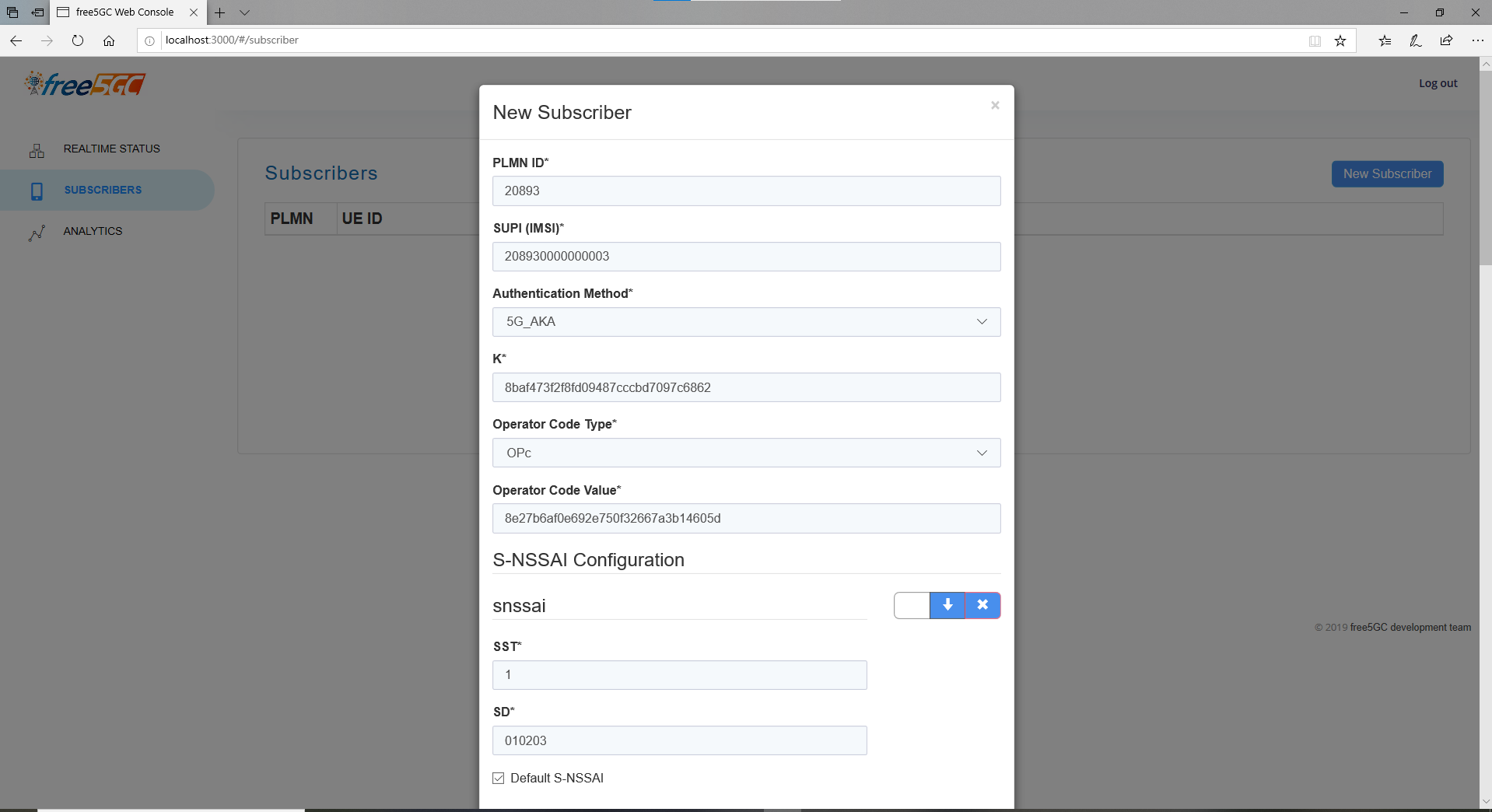
Login with the below credentials

**Username**: admin

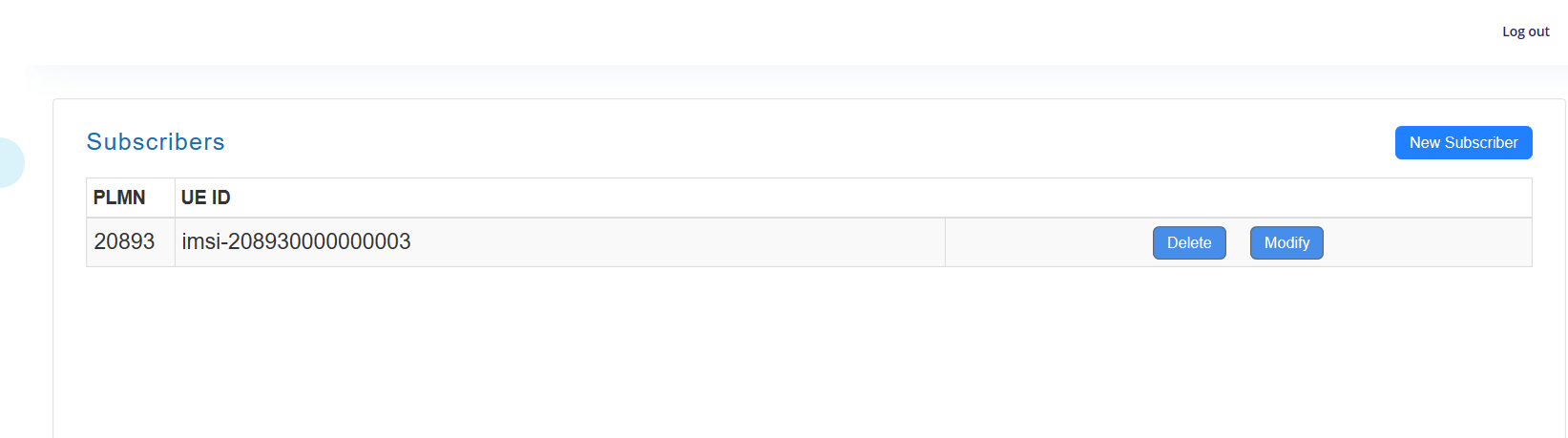
**Password**: free5gc

### Add new subscriber

* Choose **SUBSCRIBERS** in the left side and press **New Subscriber** button
* Fill the data and press **Submit** button



### New subscriber is added successfully



Once the subscriber is registered, you can deploy UERANSIM:

**# kubectl apply -f test-infra/e2e/tests/007-edge01-ueransim.yaml**

You can check to see if the simulated UE is up and running by checking the UERANSIM deployment. First, you can use the get\_capi\_kubeconfig shortcut to retrieve the kubeconfig for the edge01 cluster, and then query that cluster for the UERANSIM pods.

**# get\_capi\_kubeconfig edge01**

**# kubectl --kubeconfig edge01-kubeconfig -n ueransim get po**

The output is similar to:

NAME READY STATUS RESTARTS AGE  
ueransimgnb-edge01-748b45f684-sbs8h 1/1 Running 0 81m  
ueransimue-edge01-56fccbc4b6-h42k7 1/1 Running 0 81m

Let's see if you can simulate the UE pinging out to our DNN.

**# UE\_POD=$(kubectl --kubeconfig edge01-kubeconfig get pods -n ueransim -l app=ueransim -l component=ue -o jsonpath='{.items[0].metadata.name}')**

**# kubectl --kubeconfig edge01-kubeconfig -n ueransim exec -it $UE\_POD -- /bin/bash -c "ping -I uesimtun0 172.0.0.1"**

The output is similar to:

PING 172.0.0.1 (172.0.0.1) from 10.1.0.2 uesimtun0: 56(84) bytes of data.  
64 bytes from 172.0.0.1: icmp\_seq=1 ttl=63 time=7.01 ms  
64 bytes from 172.0.0.1: icmp\_seq=2 ttl=63 time=6.28 ms  
64 bytes from 172.0.0.1: icmp\_seq=3 ttl=63 time=8.89 ms  
64 bytes from 172.0.0.1: icmp\_seq=4 ttl=63 time=8.15 ms  
^C  
--- 172.0.0.1 ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3003ms  
rtt min/avg/max/mdev = 6.280/7.586/8.896/1.011 ms

Note that our DNN does not actually provide access to the internet, so you won't be able to reach the other sites.

**Change the Capacities of the UPF and SMF NFs**

In this step, you will change the capacity requirements for the UPF and SMF, and see how the operator reconfigures the Kubernetes resources used by the network functions.

The capacity requirements are captured in a custom resource (capacity.yaml) within the deployed package. You can edit this value with the CLI, or use the web interface. Both options lead to the same result, but using the web interface is faster. First, you will vertically scale the UPF using the CLI.

To create a new package revision, you need to start by retrieving the PackageVariant downstream target name.

**# kubectl config use kind-kind**

**# kubectl get packagevariant edge-free5gc-upf-edge01-free5gc-upf -o jsonpath='{.status.downstreamTargets[0].name}'**

This way you retrieve the downstream target name of the package. You can also retrieve this information by using the `kubectl describe` command on the UPF PackageVariant.

Next create a new package revision from the existing UPF package.

**# kpt alpha rpkg copy -n default edge01-e827e1b4d5ea1d76d1514de20d1ee27bf884c72e --workspace upf-scale-package**

The output contains the package revision of our newly cloned upf package. Pull the package to a local directory of your choice (in the example you can use /tmp/upf-scale-package).

**# kpt alpha rpkg pull -n default edge01-40c616e5d87053350473d3ffa1387a9a534f8f42 /tmp/upf-scale-package**

You can inspect the contents of the package in the chosen directory. The UPF configuration is located in the capacity.yaml file.

**# cat /tmp/upf-scale-package/capacity.yaml**

The output is similar to:

apiVersion: req.nephio.org/v1alpha1  
kind: Capacity  
metadata: # kpt-merge: /dataplane  
 name: dataplane  
 annotations:  
 config.kubernetes.io/local-config: "true"  
 specializer.nephio.org/owner: workload.nephio.org/v1alpha1.UPFDeployment.upf-edge01  
 specializer.nephio.org/namespace: free5gc-upf  
 internal.kpt.dev/upstream-identifier: 'req.nephio.org|Capacity|default|dataplane'  
spec:  
 maxUplinkThroughput: 5G  
 maxDownlinkThroughput: 5G

The contents of the package will be mutated using kpt functions to adjust the UPF configuration, however you can also manually edit the file. Apply the kpt functions to the contents of the kpt package with a new value for the throughputs of your choice.

**# kpt fn eval --image gcr.io/kpt-fn/search-replace:v0.2.0 /tmp/upf-scale-package -- by-path='spec.maxUplinkThroughput' by-file-path='\*\*/capacity.yaml' put-value=10**

**# kpt fn eval --image gcr.io/kpt-fn/search-replace:v0.2.0 /tmp/upf-scale-package -- by-path='spec.maxDownlinkThroughput' by-file-path='\*\*/capacity.yaml' put-value=10**

The output is similar to:

[RUNNING] "gcr.io/kpt-fn/search-replace:v0.2.0"  
[PASS] "gcr.io/kpt-fn/search-replace:v0.2.0" in 6.3s  
 Results:  
 [info] spec.maxUplinkThroughput: Mutated field value to "10"

Observe the changes to the UPF configuration using the kpt pkg diff command.

**# kpt pkg diff /tmp/upf-scale-package | grep linkThroughput**

The output is similar to:

From <https://github.com/nephio-project/free5gc-packages> \* tag pkg-example-upf-bp/v3 -> FETCH\_HEAD  
Adding package "pkg-example-upf-bp".  
< maxUplinkThroughput: 10G  
< maxDownlinkThroughput: 10  
> maxUplinkThroughput: 5G  
> maxDownlinkThroughput: 5G

Next, progress through the package lifecycle stages by pushing the changes to the package to its repository, proposing the changes and approving them.

**# kpt alpha rpkg push -n default edge01-40c616e5d87053350473d3ffa1387a9a534f8f42 /tmp/upf-scale-package**

**# kpt alpha rpkg propose -n default edge01-40c616e5d87053350473d3ffa1387a9a534f8f42**

**# kpt alpha rpkg approve -n default edge01-40c616e5d87053350473d3ffa1387a9a534f8f42**

The output is similar to:

edge01-40c616e5d87053350473d3ffa1387a9a534f8f42 proposed  
edge01-40c616e5d87053350473d3ffa1387a9a534f8f42 approved

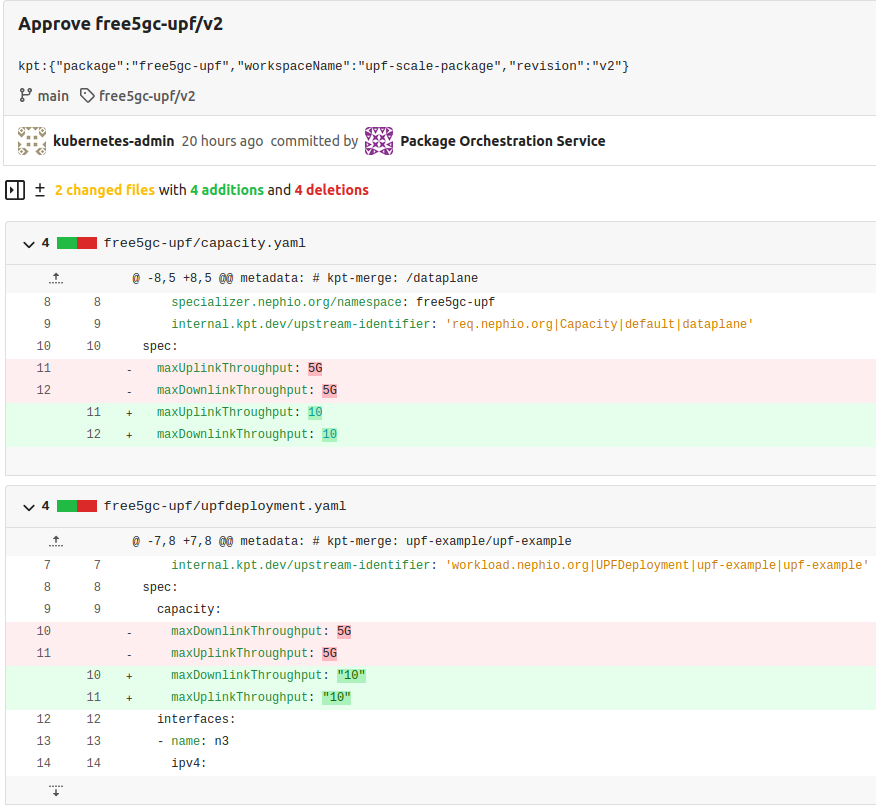
You can check the current lifecycle stage of a package using the kpt alpha rpkg get command.

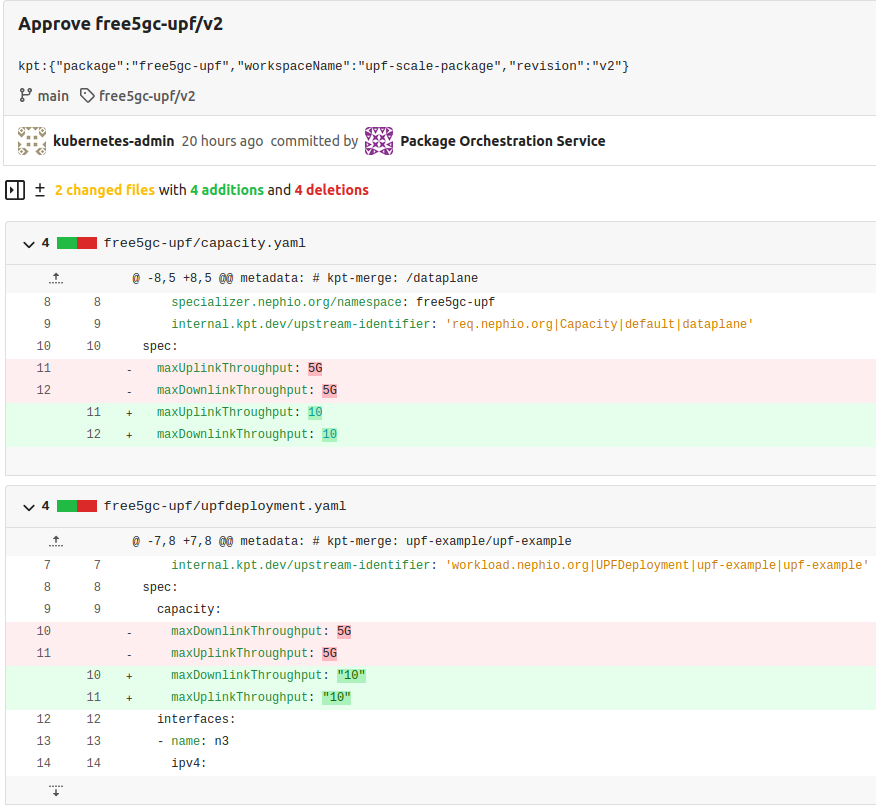
**# kpt alpha rpkg get**

The output is similar to:

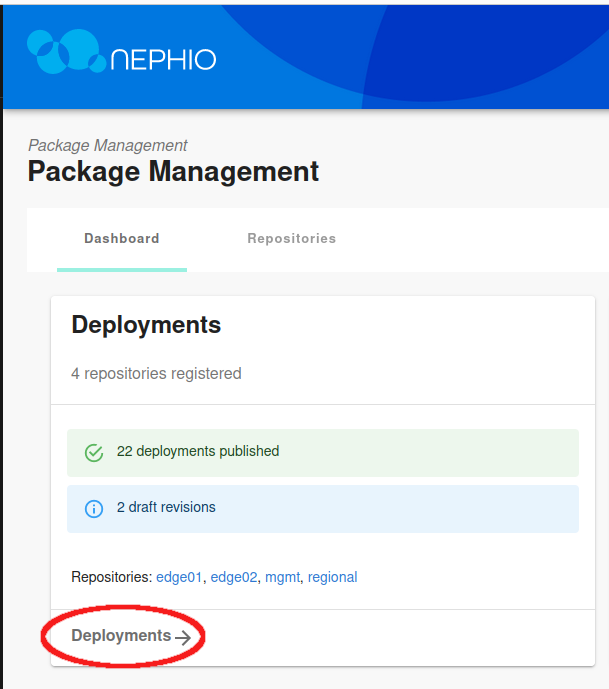
NAME PACKAGE WORKSPACENAME REVISION LATEST LIFECYCLE REPOSITORY  
edge01-e72d245b864db0fd234d9b4ead2f96edcf6bb3e4 free5gc-operator packagevariant-1 main false Published edge01  
edge01-7c9bf9f43768ecd2b45a8be84698763cdd2593b6 free5gc-operator packagevariant-1 v1 true Published edge01  
edge01-40c616e5d87053350473d3ffa1387a9a534f8f42 free5gc-upf upf-scale-package v2 true Published edge01

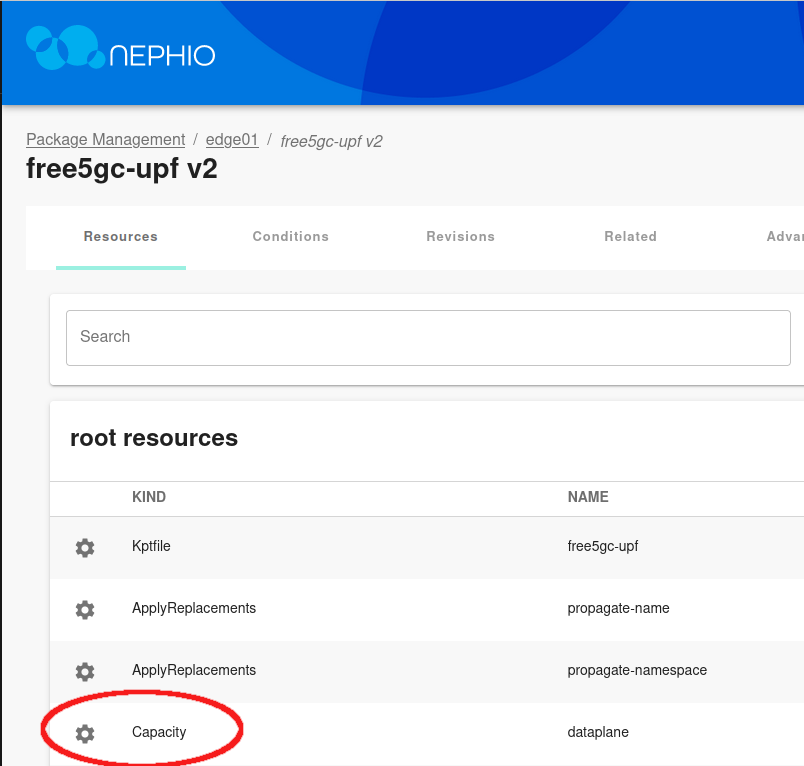
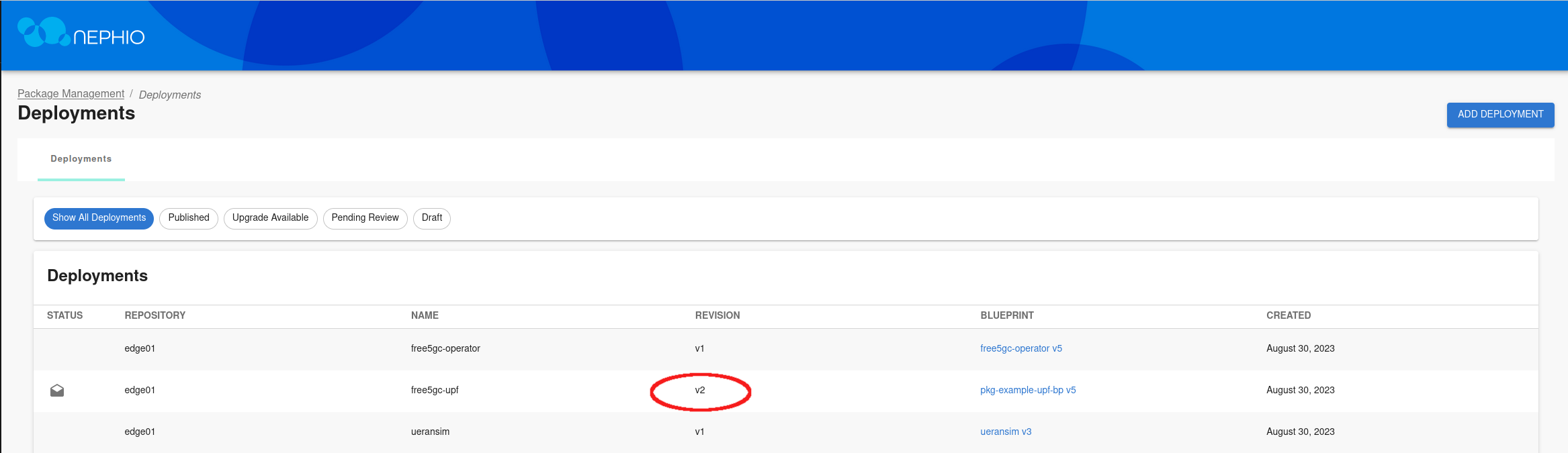
Additionally you can check the Gitea edge01 repository (accessible at[**http://localhost:3000/nephio/edge01**](http://localhost:3000/nephio/edge01)) for new commits to see how Porch interacts with packages stored in Git repositories.

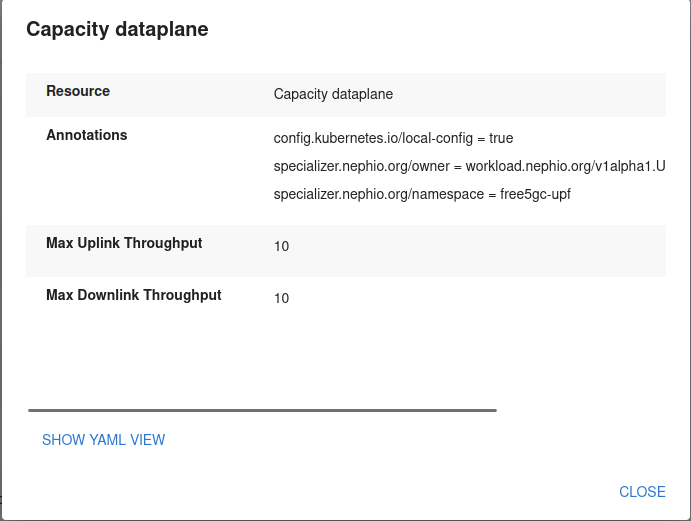




After the package is approved, the results can be observed in Nephio Web UI. Head over to[**http://localhost:7007/config-as-data**](http://localhost:7007/config-as-data) ([port forwarding](https://github.com/nephio-project/docs/blob/main/install-guide/README.md#access-to-the-user-interfaces/) must be running).

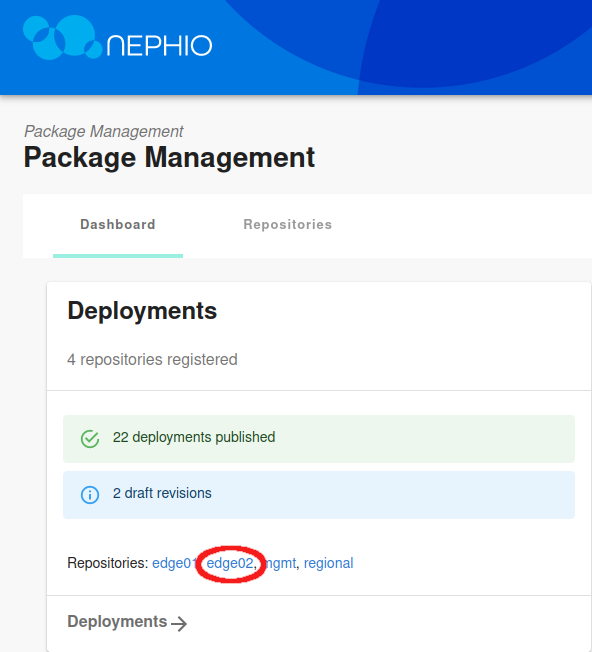




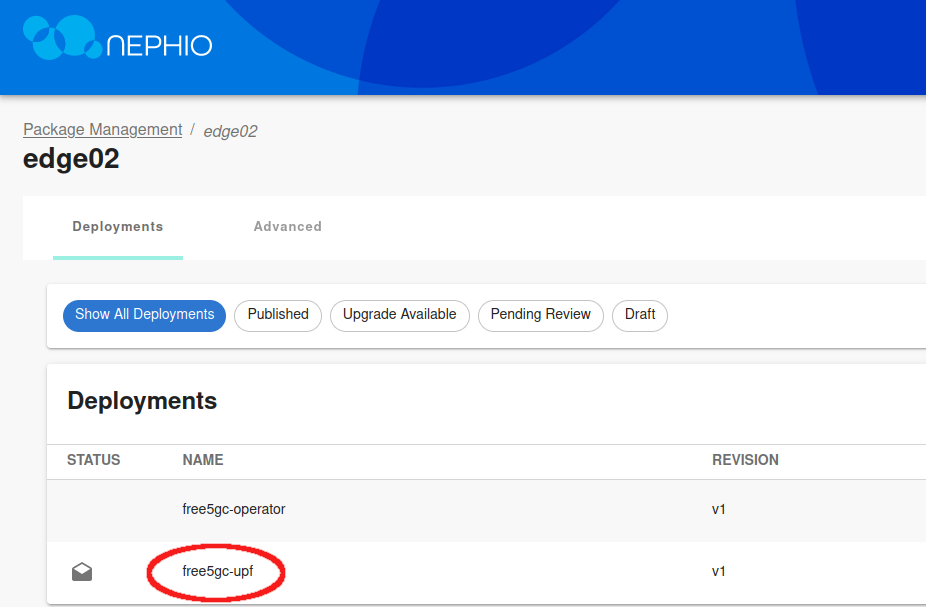


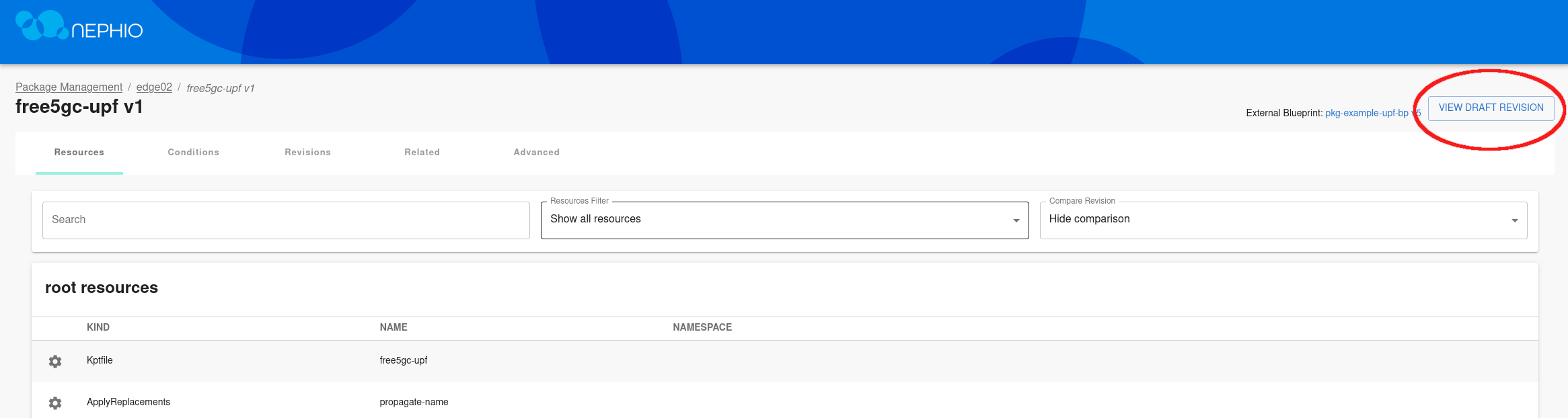
Inside the package, you can see that the throughput values for UPF have been modified, reflecting the changes you made with the CLI.

You can also scale NFs vertically using the Nephio Web UI. For practice you can scale the UPF on the second edge cluster. Once again, navigate to the Web UI and choose the edge02 repository in the Deployments section.

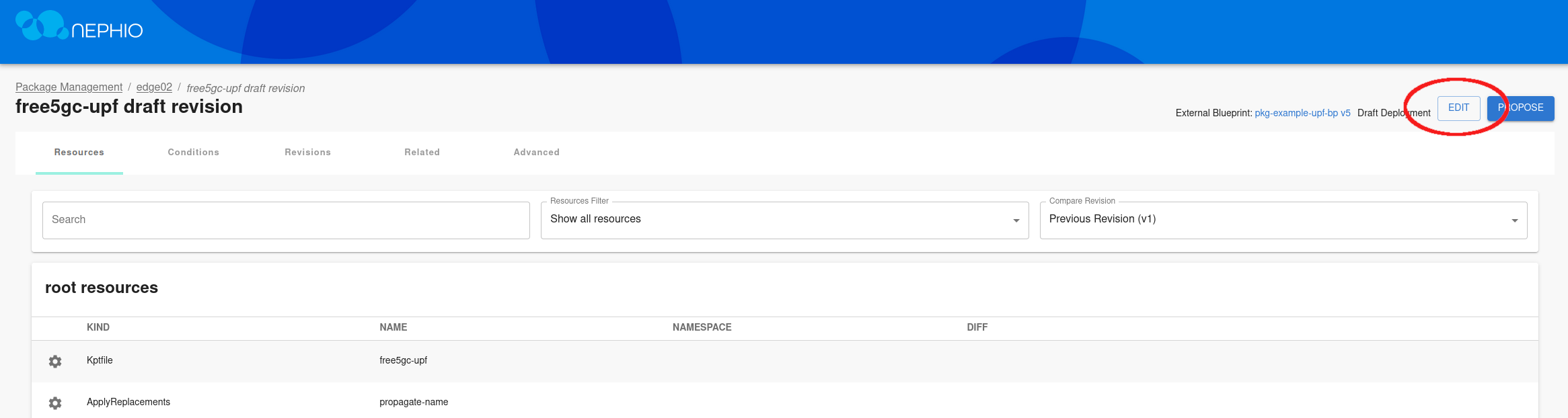


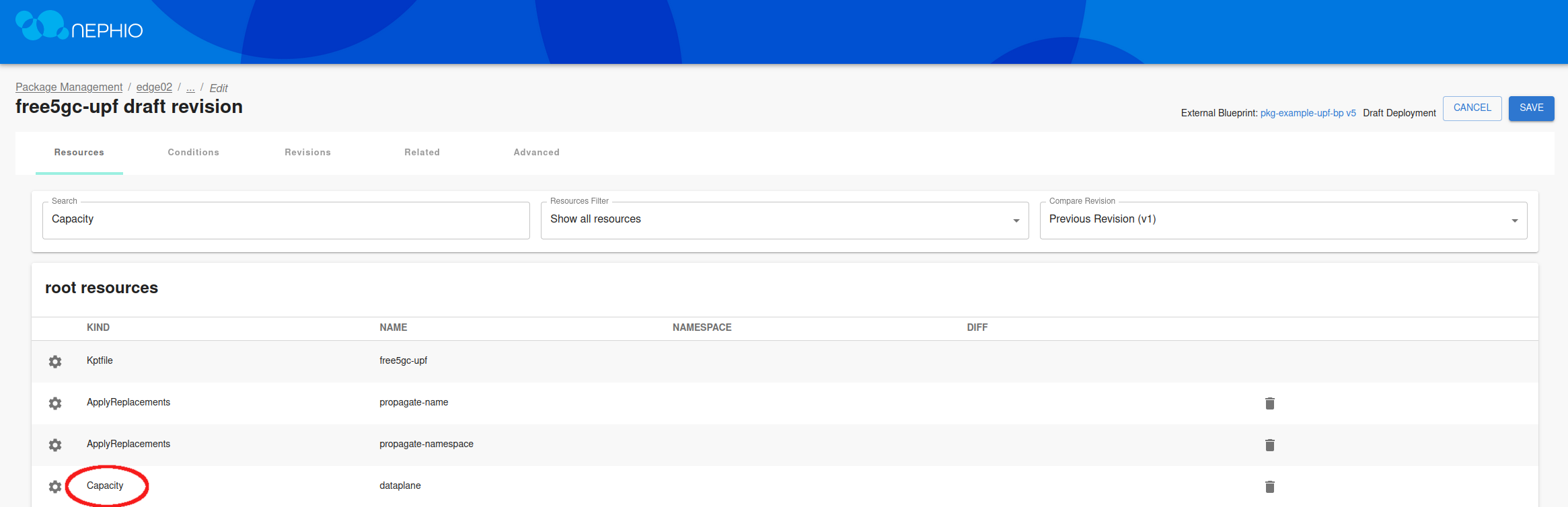
Select the free5gc-upf deployment, and then View draft revision.



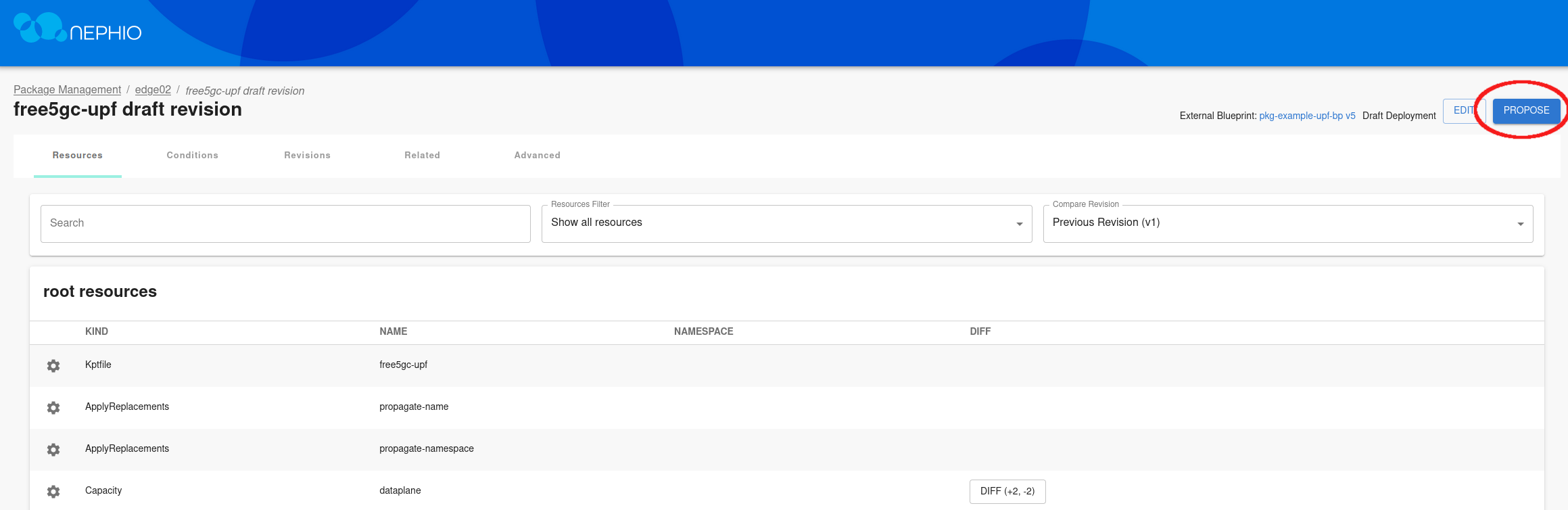


Edit the draft revision, and modify the Capacity.yaml file.

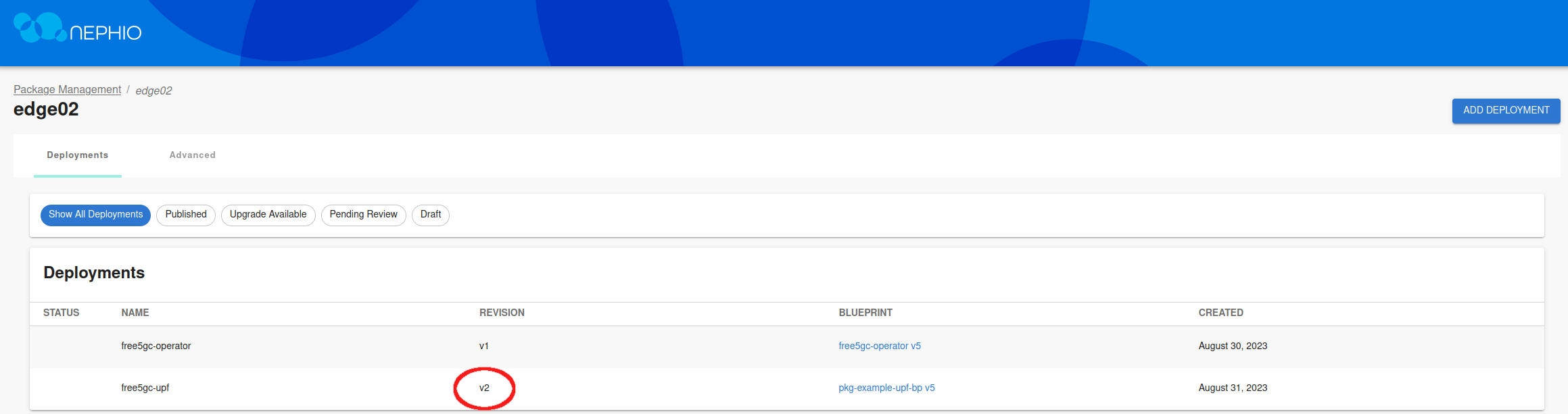








After saving the changes to the file, propose the draft package and approve it.



After a few minutes, the revision for the UPF deployment will change, and the changes will be reflected in the edge-02 cluster.