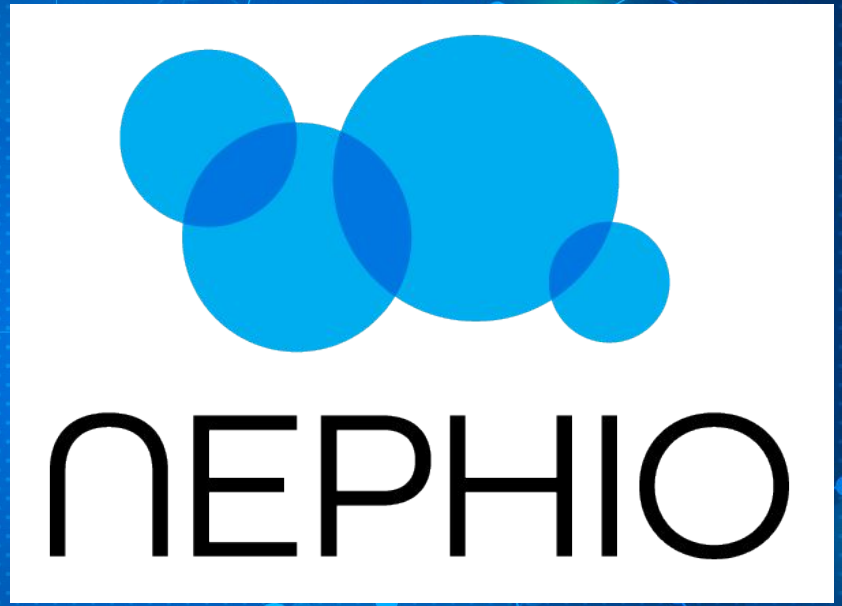
R1 Live Workshop

October 9, 2023

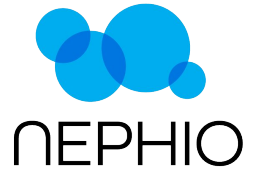
- John Belamaric, Google
- Anh Thu Vo
- Ravi Ravindran, F5 Networks
- Sandeep Sharma, Aarna
- Victor Morales, Samsung
- Vish Jayaraman, Red Hat

https://nephio.org/learn



THE LINUX FOUNDATION

Our Journey Today



- Introduction [5m]
- Installation [5m]
  - https://github.com/nephio-project/docs/tree/v1.0.1/install-guide
- Exercises - Theory & Practice [60m]
  - https://github.com/nephio-project/docs/blob/v1.0.1/user-guide/exercises.md
  - Create regional cluster
  - Deploy edge clusters
  - Deploy regional free5gc control plane
  - Deploy free5gc operator to the regional and edge clusters
  - Deploy AMF, SMF, UPF
  - Deploy UERANSIM (not shown)
  - Change in capacity required ⇒ change in CPU and memory
- Q & A [15m]

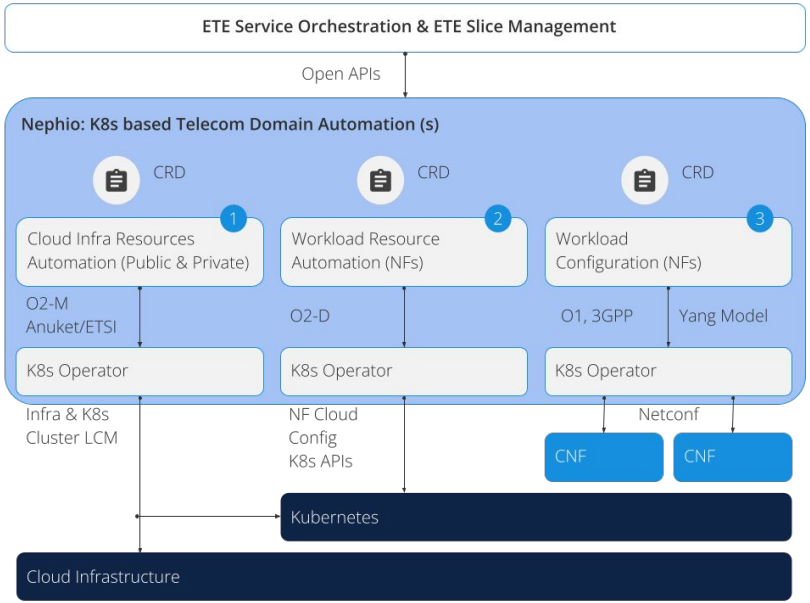
Nephio Scope

“Swimlanes”

1. Infrastructure
2. Workload (network function)
3. Workload configuration

R1 Demonstrates Some of Each

1. Cluster provisioning
2. Network function provisioning
3. NF config file generation in operator



Very High Level Nephio Architecture

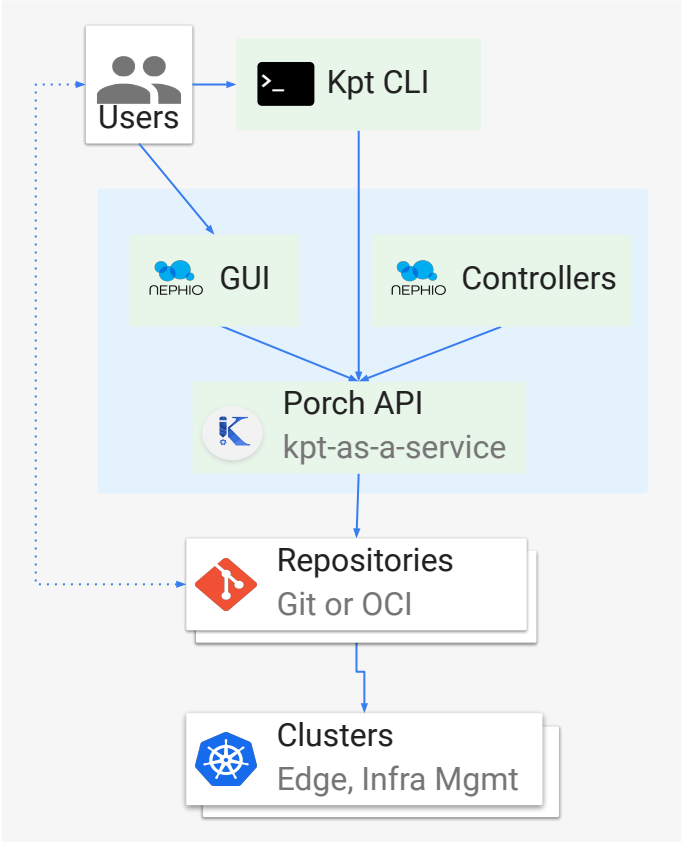
Platform enabling users and automation agents to cooperatively interact with and deploy configuration

Central Nephio K8s cluster houses GUI service, Porch APIs, and Nephio Controllers

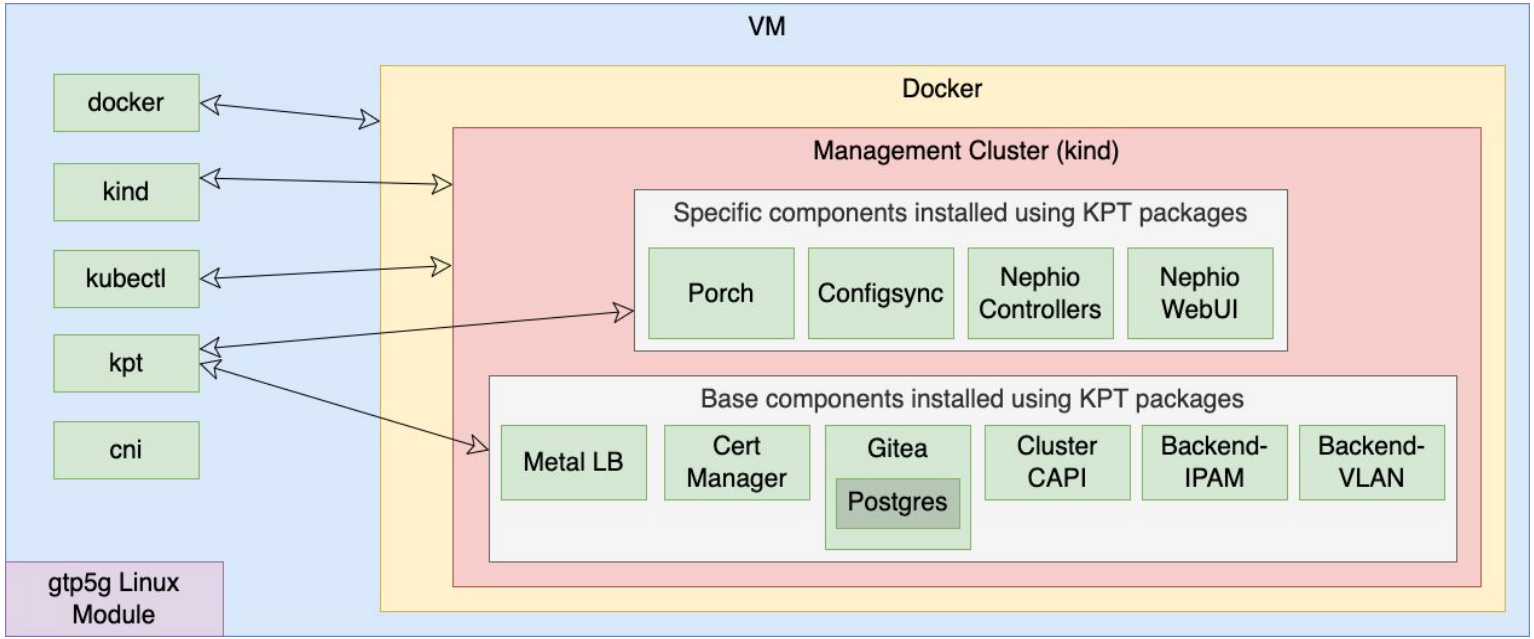
Users manipulate config packages which Porch pushes to Git or OCI repositories

Downstream clusters consume those via Config Sync, which applies them to the K8s API server

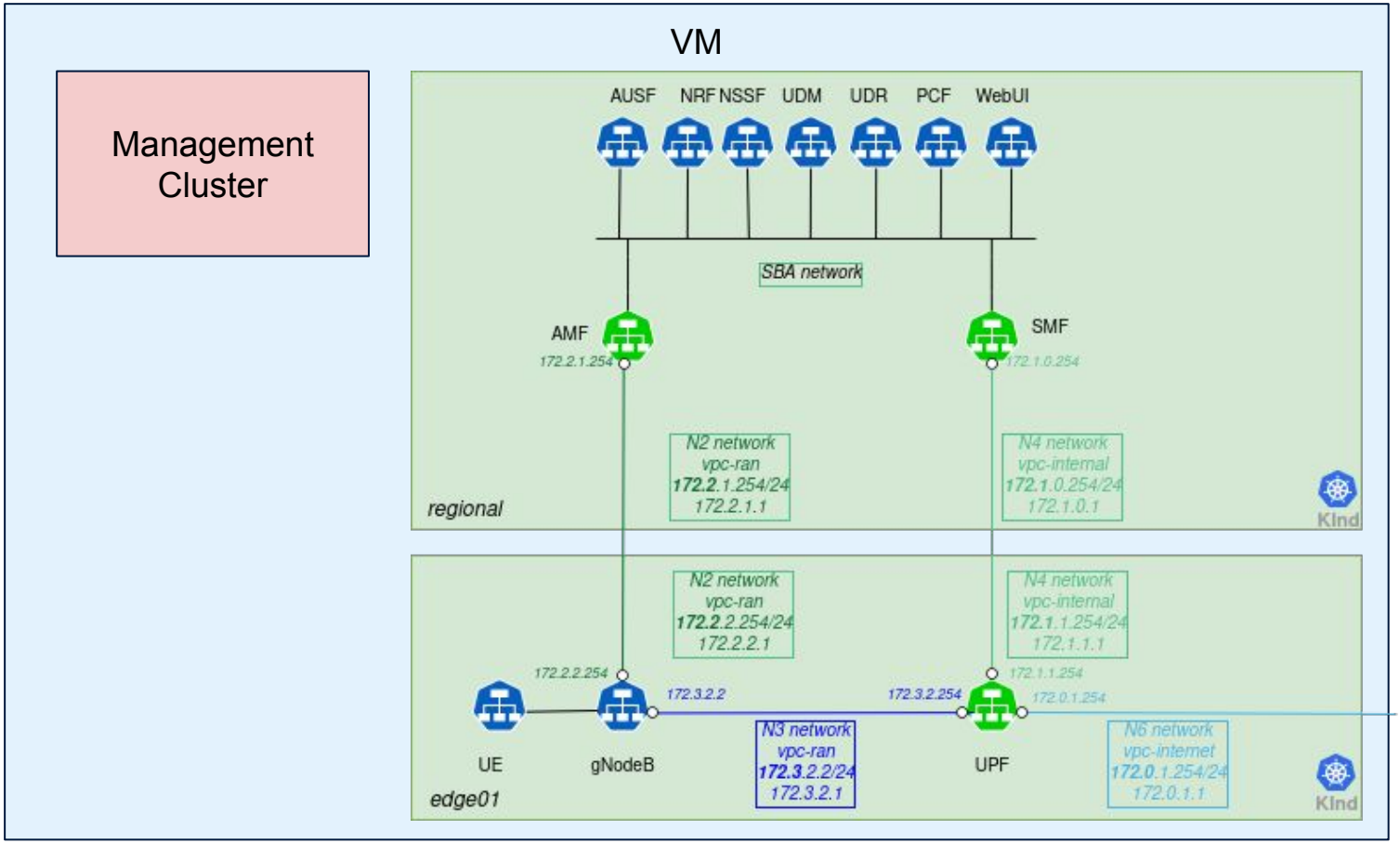
Same process for infrastructure - using for example KCC to create clusters, as for workloads

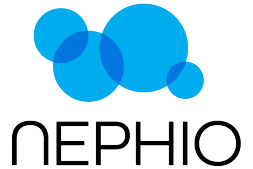


Starting Place



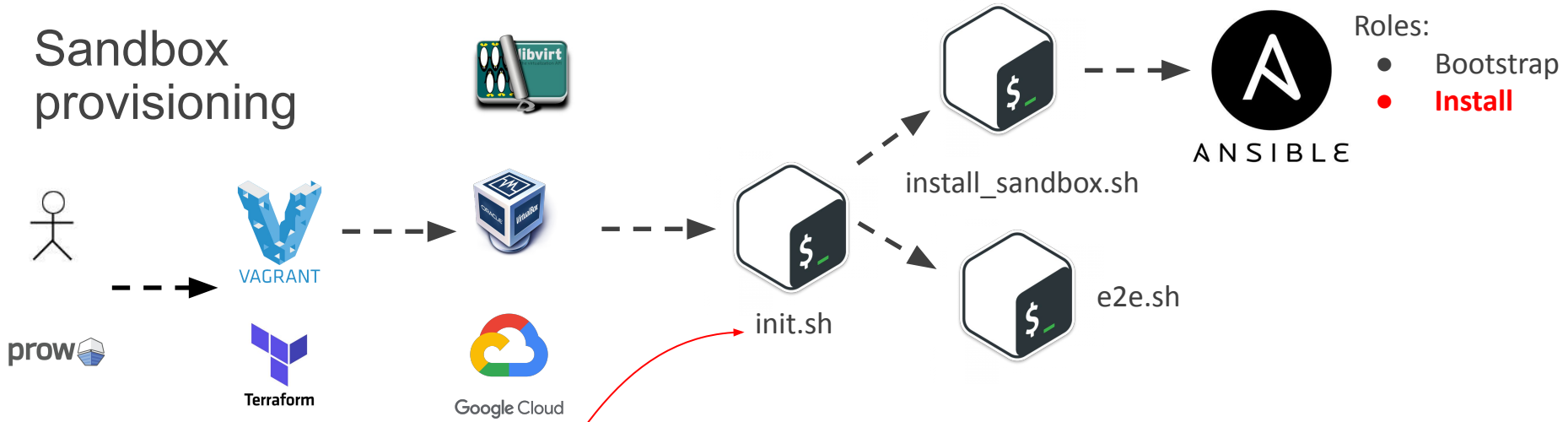
End Result


Management Cluster
VM
AUSF
NRF
NSSF
UDM
UDR
PCF
WebUI
SBA network
AMF
172.2.1.254
SMF
172.1.0.254
N2 network
vpc-ran
172.2.1.254/24
172.2.1.1
N4 network
vpc-internal
172.1.0.254/24
172.1.0.1
regional
Kind
N2 network
vpc-ran
172.2.2.254/24
172.2.2.1
N4 network
vpc-internal
172.1.1.254/24
172.1.1.1
172.2.2.254
172.3.2.2
172.3.2.254
172.1.1.254
172.0.1.254
internet
N3 network
vpc-ran
172.3.2.2/24
172.3.2.1
N6 network
vpc-internet
172.0.1.254/24
172.0.1.1
UE
gNodeB
UPF
edge01
Kind



Install

Sandbox provisioning




```
# Provision test bed
config.vm.provision 'shell', privileged: true do |sh|
  sh.env = {
    NEPHIO_DEBUG: ENV.fetch('DEBUG', true),
    NEPHIO_DEPLOYMENT_TYPE: ENV.fetch('DEPLOYMENT_TYPE', 'r1'),
    NEPHIO_RUN_E2E: ENV.fetch('RUN_E2E', false),
    NEPHIO_USER: ENV.fetch('NEPHIO_USER', 'vagrant'),
    NEPHIO_REPO_DIR: '/opt/test-infra',
    E2EDIR: '/opt/test-infra/e2e'
  }
  sh.inline = <<-SHELL
    set -o errexit
    set -o pipefail

    cd /opt/test-infra/e2e/provision/
    ./init.sh | tee ~/init.log
  SHELL
end
```

Requirements:

- Vagrant CLI (https://developer.hashicorp.com/vagrant/downloads)
- Providers:
  - Libvirt provider (https://github.com/vagrant-libvirt/vagrant-libvirt)
  - GCE Provider(https://github.com/mitchellh/vagrant-google)

https://github.com/nephio-project/test-infra/blob/main/e2e/provision/Vagrantfile

$ NEPHIO_DEBUG=false **GOOGLE_PROJECT_ID**=pure-faculty-367518 **GOOGLE_JSON_KEY_LOCATION**=~/.config/gcloud/pure-faculty-367518-f1afca4feb6b.json NEPHIO_USER=$USER RUN_E2E=true vagrant up --provider google

Nephio Installation in a Nutshell

The following kpt packages are needed:

- Porch - https://github.com/nephio-project/nephio-example-packages/tree/main/porch-dev
- Nephio Controllers - https://github.com/nephio-project/nephio-example-packages/tree/main/nephio-controllers
- Management Cluster GitOps Tool - https://github.com/nephio-project/nephio-example-packages/tree/main/configsync
- Nephio Stock Repositories (optional) - https://github.com/nephio-project/nephio-example-packages/tree/main/nephio-stock-repos

```
for nephio_pkg in porch-dev nephio-controllers configsync nephio-stock-repos; do
    kpt pkg get --for-deployment "https://github.com/nephio-project/nephio-example-packages.git/$nephio_pkg@v1.0.1"
    kpt fn render "$nephio_pkg"
    kpt live init "$nephio_pkg"
    kpt live apply "$nephio_pkg" --reconcile-timeout=15m --output=table
done
```

https://github.com/nephio-project/docs/blob/main/install-guide/common-components.md

- **Concepts**
  - Package
  - Package Management
  - Automation & Scaling Package Deployment
- **Demo**
  - Create Regional Cluster
  - Deploy Edge Clusters
  - Deploy Regional Free5gc Control Plane (not including AMF and SMF)
  - Deploy Free5gc Operator

What is Kubernetes Resource Model (KRM)

- A way to create a declarative configuration file in a readable format to define the desired system state using code.
- Encourages separation of concerns by supporting multiple distinct configuration sources and preserving declarative intent while allowing automatically set attributes

KRM Function Specification

- Client side functions that operate on K8S declarative configurations are referred to as KRM functions.
- Enables creating small, interoperable and language-independent executable programs packaged as containers that can be chained together as part of a configuration management pipeline.
- Pipeline functions execution results in configurations that can be applied to a control plane

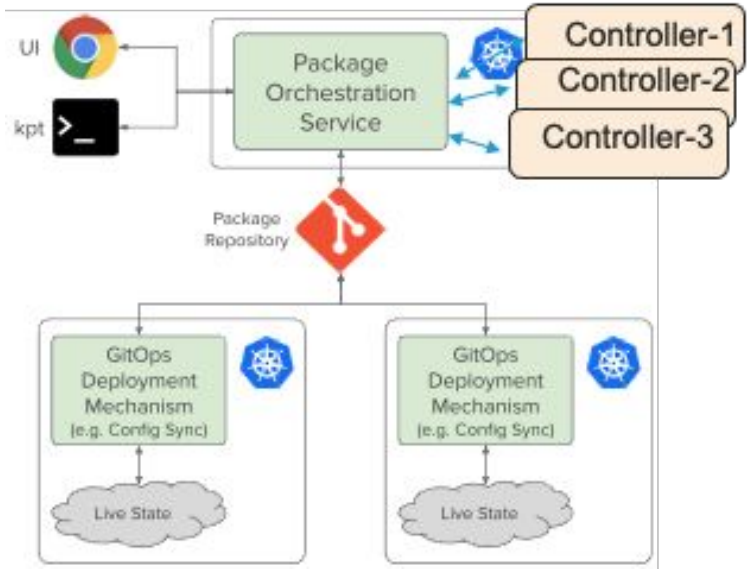
KRM Functions - Use Cases

KRM functions enable shift-left-practices(client-side) through

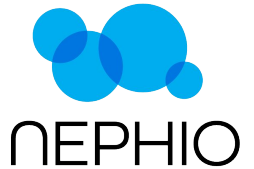
- *Pre-commit / delivery validation and linting of configuration*
  - e.g. Fail if any containers don't have CPU / Memory limits
- *Implementation of abstractions as client actuated APIs*
  - e.g. Create a client-side *"CRD"* for generating configuration checked into git
- *Injection of cross-cutting configuration*
  - e.g. T-Shirt size containers by annotating resources with `small`, `medium`, `large` and inject the cpu and memory resources into containers accordingly.
  - e.g. Inject `init` and `side-car` containers into resources based off of resource type, annotations, etc.

Packages and Package Management

- **What is a Package ?**
  - Bundle of KRM
  - Some metadata
  - Function pipeline
- **How to Manage Packages ?**
  - Create, Modify, Deploy, and Delete
- ***Porch* makes Packages available as K8s APIs using**
  - PackageRevision CR
  - PackageRevisionResources CR
- **Higher level services**
  - Package Repository Management
  - Package Discovery, Authoring and Lifecycle Management



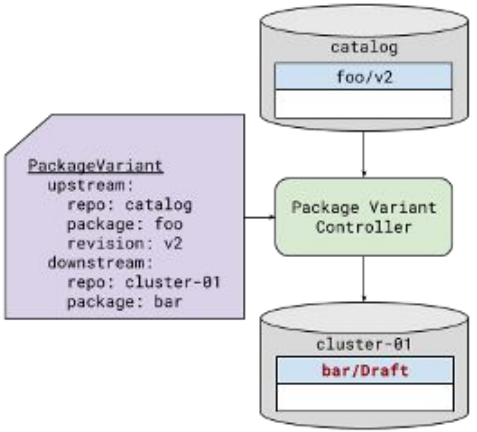
Scaling and Automating Package Management



- **Designed to address several different dimensions of Scalability**
  - Number of different workloads for a given cluster
  - Number of clusters across with those workloads are deployed
  - Different types or characteristics of those clusters
  - Complexity of the organizations deploying those workloads
  - Changes to those workloads over time
- **Using PackageVariant and PackageVariantSet**
  - Apply any Context or Resource outside the Package's Context, e.g.Cluster, Infra, Service etc.
- **Allows for automating the creation and lifecycle management of package variants.**
  - E.g; Across a fleet of clusters, it may be necessary to modify workload configuration for a specific cluster
- **Manages the deviation of a variant of a package from the original source package, and manage the evolution of that variant over time.**

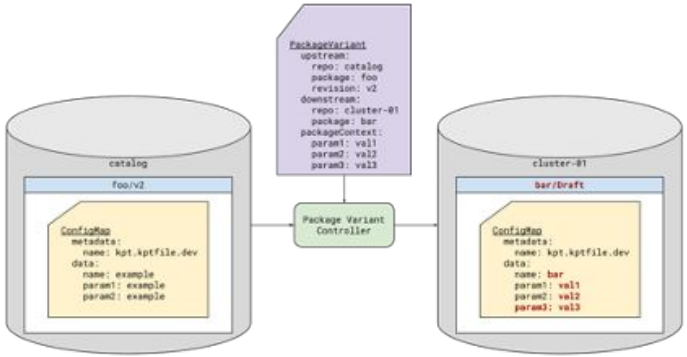
PackageVariant CR / PackageVariant Controller

- **Takes instance of the Upstream Blueprint package and clones as draft Deployment Repo applying several transformations**



- **Affect KPT Pipeline in the original Package**

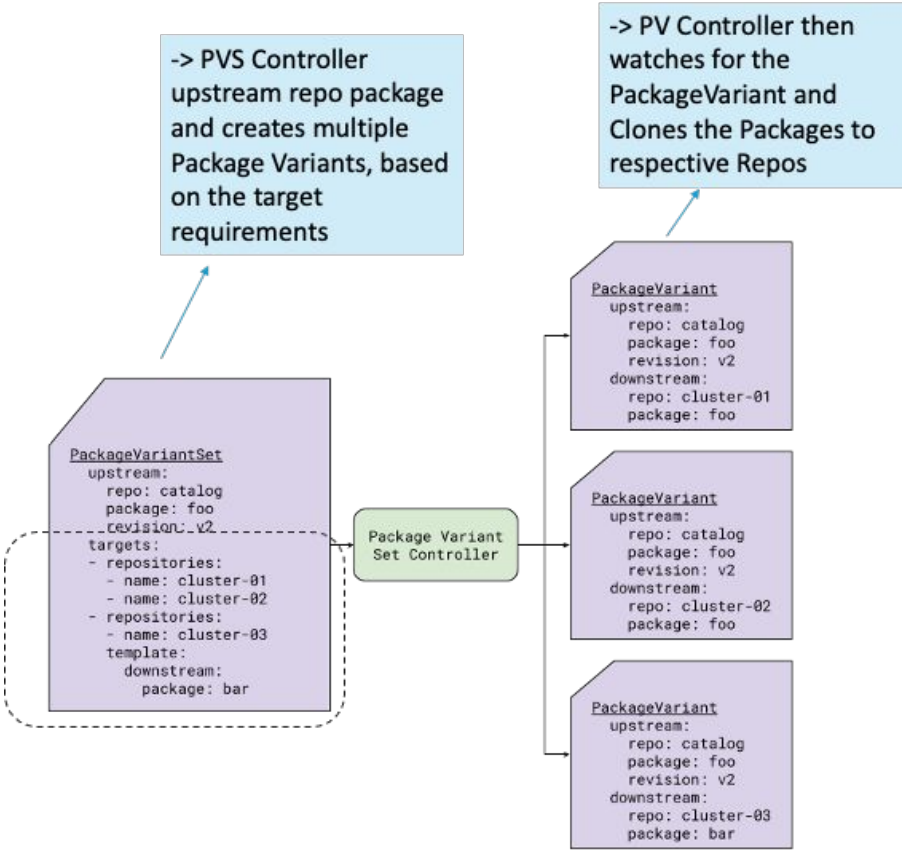
- **Manage the Package Context (Configmap)**
  - Customized Key:Value pairs



- **Specify Injectors**
  - Inject cluster KRM into the new Copy of the Blueprint Package

PackageVariantSet CR and PVS Controller

- **Handles large scale Fan Out scenarios**
- **The PVS Controller uses the reference to the Upstream Blueprint Repository Package in the CR and creates *multiple Package Variants***
- **Targeting criterion in the PVS**
  - An explicit list of repositories and package names
  - A label selector for Repository objects
  - An arbitrary object selector
  - Customize Injectors and Kpt pipelines for each Package Variant

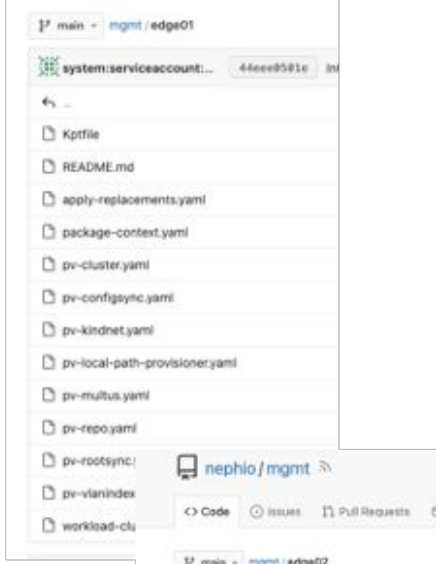


Example of a PVS and PV

```yaml
apiVersion: config.porch.kpt.dev/v1alpha2
kind: PackageVariantSet
metadata:
  name: edge-clusters
spec:
  upstream:
    repo: nephio-example-packages
    package: nephio-workload-cluster
    revision: v9
  targets:
  - repositories:
    - name: mgmt
      packageNames:
      - edge01
      - edge02
    template:
      annotations:
        approval.nephio.org/policy: initial
      pipeline:
        mutators:
        - image: gcr.io/kpt-fn/set-labels:v0.2.0
          configMap:
            nephio.org/site-type: edge
            nephio.org/region: us-west1
```

Package Variant – Edge01-Cluster

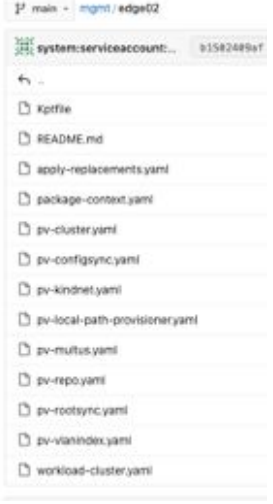
PackageVariant Controller



{ PackageVariantSet Controller }

Package Variant – Edge02-Cluster

PackageVariant Controller



- Kpt function pipeline and data that has to be injected to the Package Variant

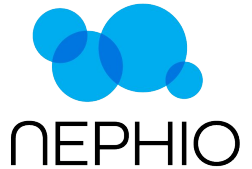
Demo

- **Creation of a Regional Cluster**
  - Using Manual KPT Commands

- **Creation of Edge Clusters**
  - Using PackageVariantSet CR

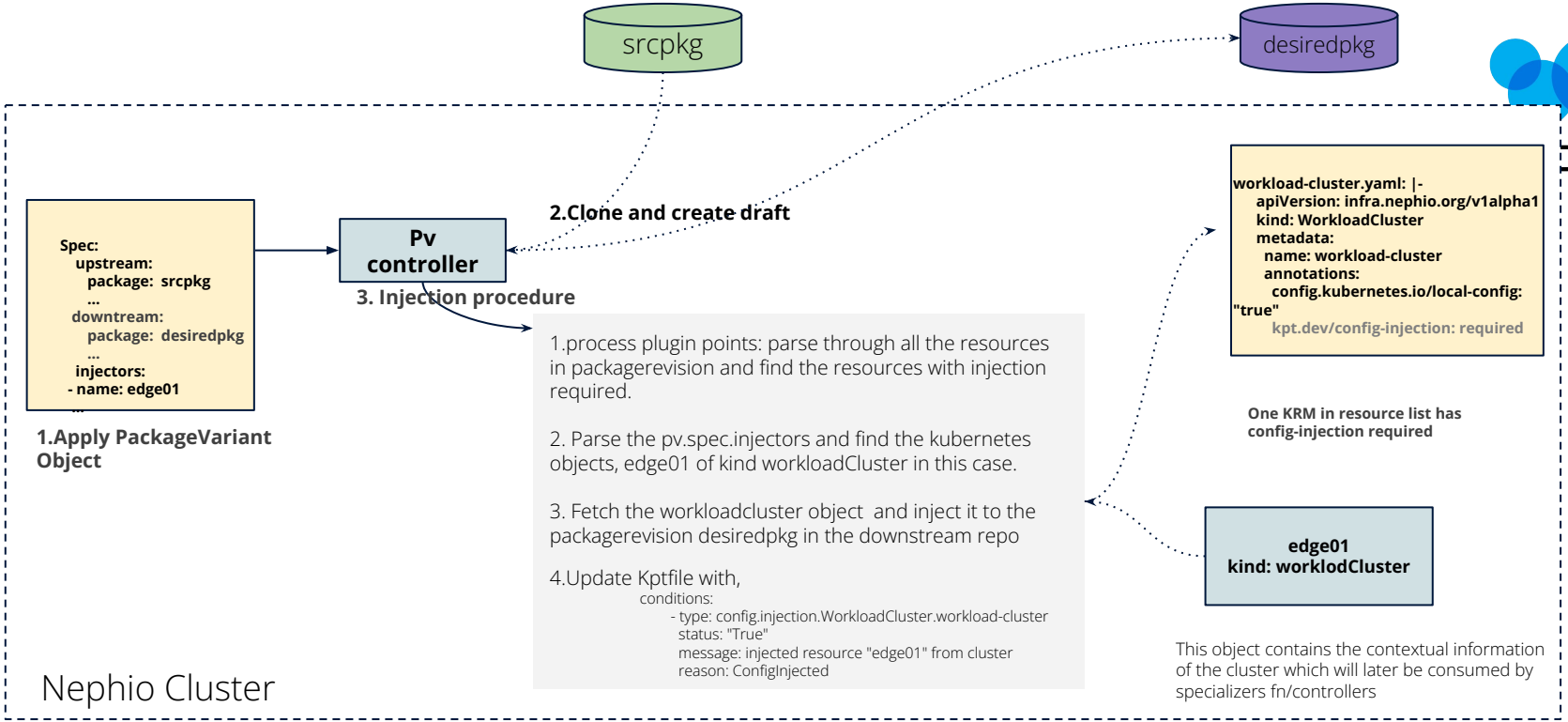
- **Deployment of Free5gc Control Plane**
  - Using Nephio WebUI

- **Deployment of Free5gc Operator**
  - Using PackageVariantSet CR



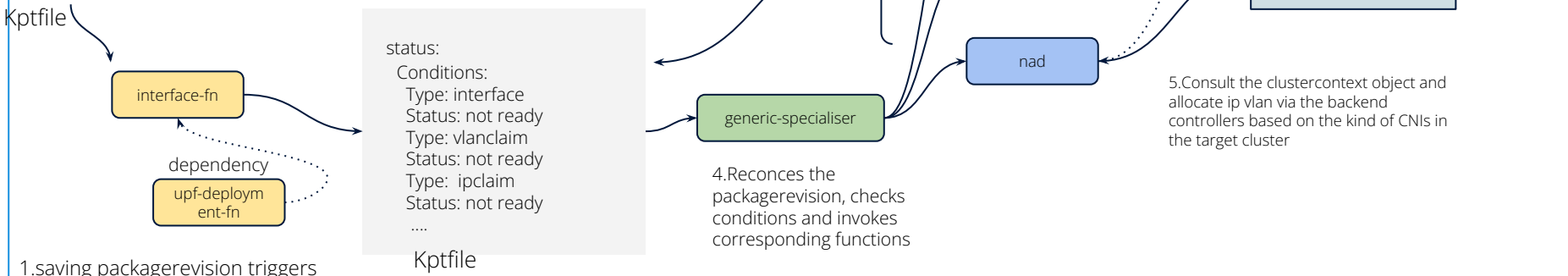
- **Concepts**
  - Config Injection
  - Package Specialization
  - Condition Choreography
- **Demo**
  - Deploy AMF, SMF, UPF

Config Injection



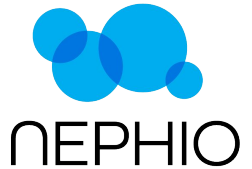
Specialization Pipeline and Condition Choreography

```
pipeline:
    mutators:
      - image: gcr.io/kpt-fn/apply-replacements:v0.1.1
        configPath: apply-replacements-namespace.yaml
      - image: docker.io/nephio/upf-deploy-fn:v1.0.1
      - image: docker.io/nephio/interface-fn:v1.0.1
```



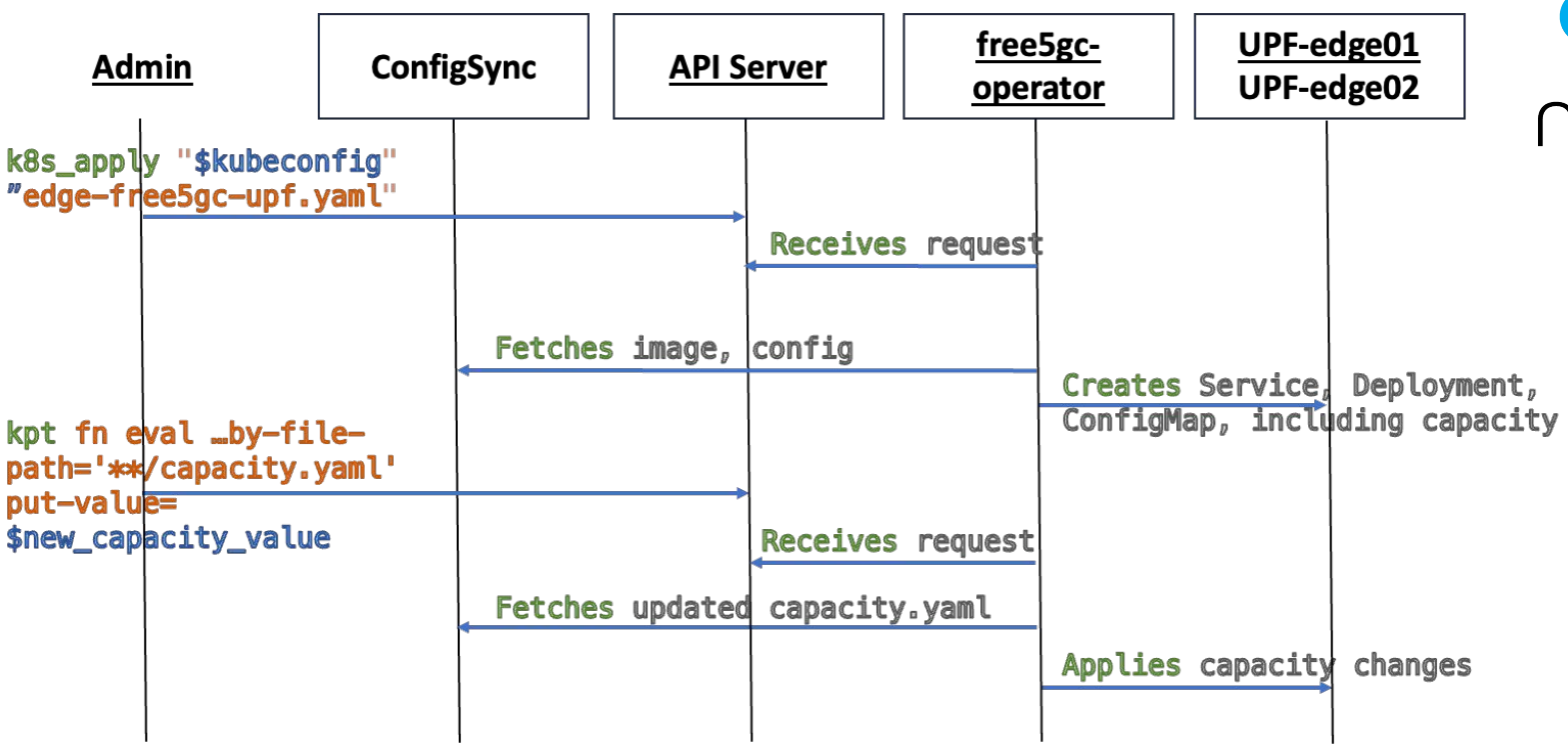
1.saving packagerevision triggers pipeline.
2. Functions create inventory of for, owns and watch
3.Functions Update kptfile conditions and save.

Eventually all resources all allocated, and all conditions in the KPTfile are set to Ready by the condition choreography done by KRM fns/controllers. The result is the deployment of UPFDeployment curated CR in the edge01 cluster



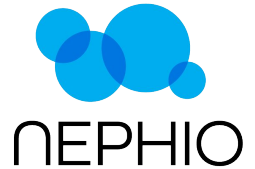
- **Concepts**
  - Specialization Output to Workload Clusters
  - free5GC operators
  - Operators translate user API to K8s API
- **Demo**
  - Capacity Changes

Creating and Updating UPF-edge01 and UPF-edge02



Sample UPF resources.go for Resource Mgmt (Optional)

github.com/nephio-project/free5gc/blob/main/controllers/upf/resources.go



func createResourceRequirements(upfDeploymentSpec nephiov1alpha1.UPFDeploymentSpec) (int32, *apiv1.ResourceRequirements, error) {…

```
        if upfDeploymentSpec. Capacity.MaxDownlinkThroughput.Value() > downlink.Value() {
                cpuLimit = "1000m"
                memoryLimit = "1Gi"
                cpuRequest = "1000m"
                memoryRequest = "1Gi"
        } else {
                cpuLimit = "500m"
                memoryLimit = "512Mi"
                cpuRequest = "500m"
                memoryRequest = "512Mi"
        }

resources := apiv1.ResourceRequirements{
                Limits: apiv1.ResourceList{
                        apiv1.ResourceCPU:
                        resource.MustParse(cpuLimit),
                        apiv1.ResourceMemory:
                        resource.MustParse(memoryLimit),
                },
                Requests: apiv1.ResourceList{
                        apiv1.ResourceCPU:
                        resource.MustParse(cpuRequest),
                        apiv1.ResourceMemory:
                        resource.MustParse(memoryRequest),
},}
return replicas, &resources, nil
}
```

Everyone

Q&A