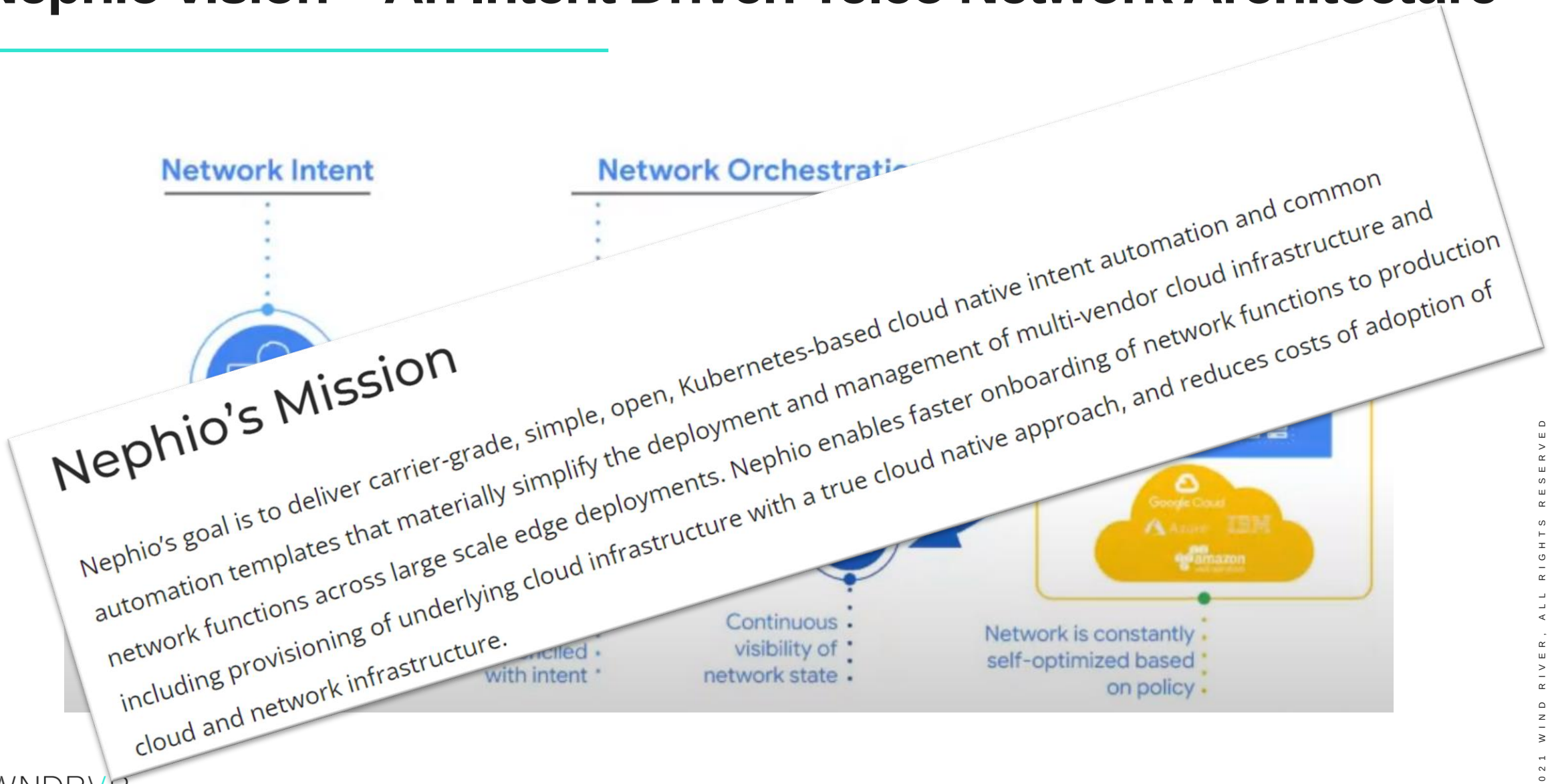
WNDRVR

ocloud-operator

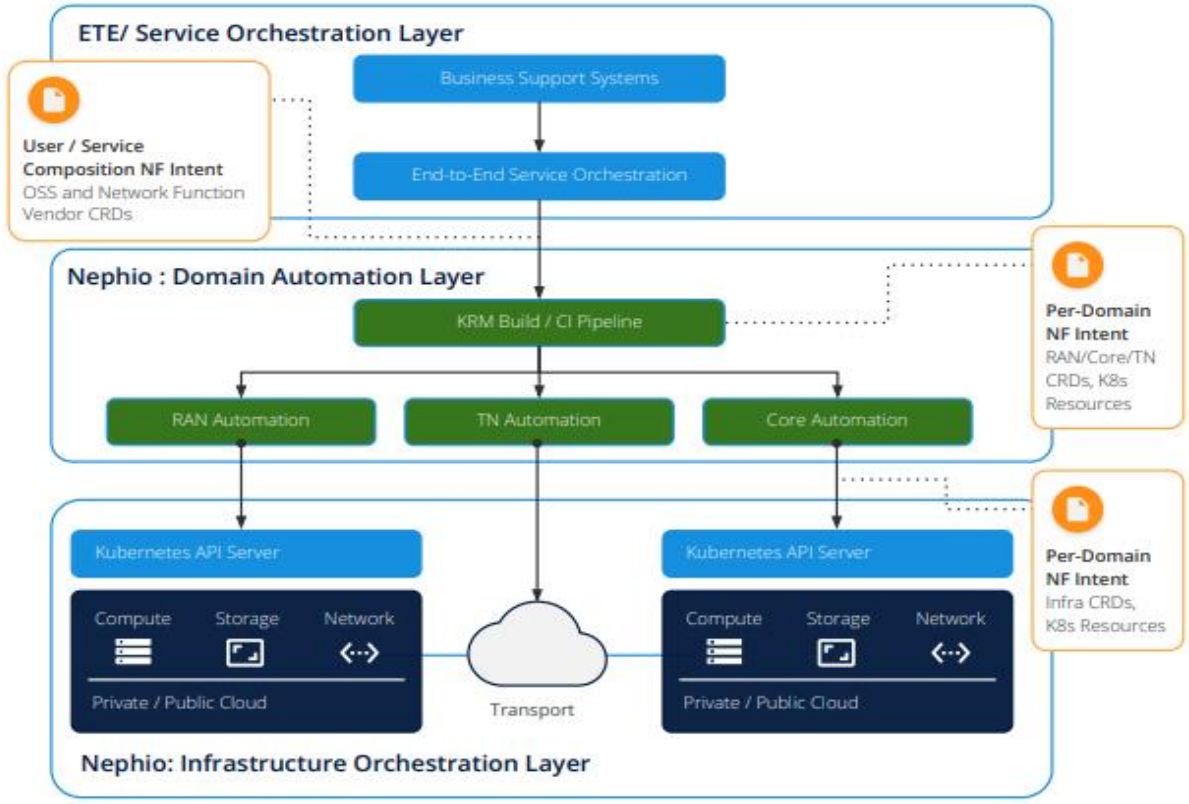
19/04/2023

Nephio vision – An Intent Driven Telco Network Architecture



Nephio Scope

Telecom Automation layers & Nephio Scope



External: Service orchestration layer

- Accepts user requirements
- Composes functions and supports end-to-end n/w slicing configuration
- Pushes intent to domain orchestration layer
- PNF orchestration

Nephio: Domain orchestration layer

- Accepts service composition
- Calculates domain and cluster-specifics
- Pushes Kubernetes manifests

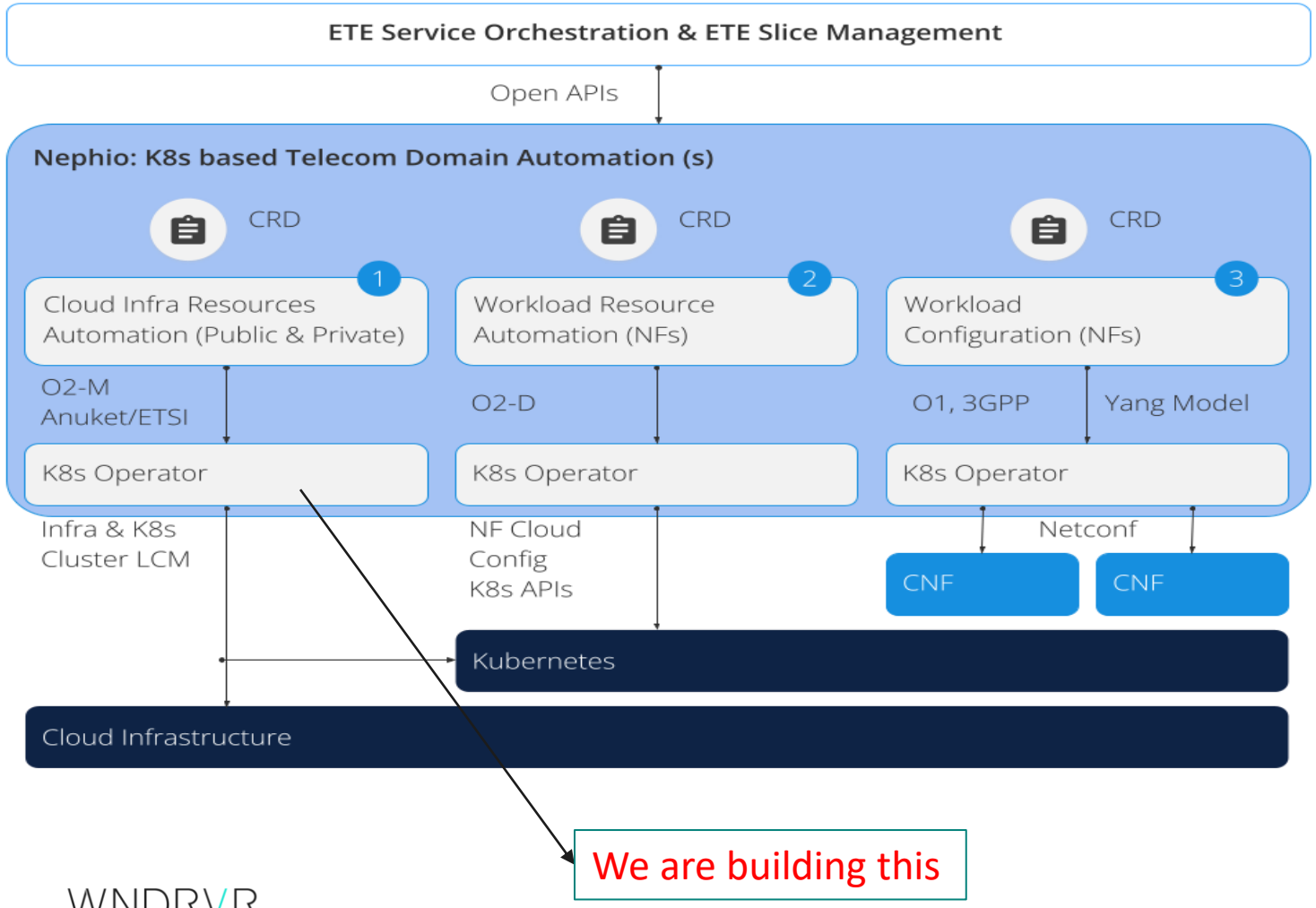
Nephio: Infrastructure orchestration layer

- Applies per-cluster Kubernetes manifests
- Actuates infrastructure resources
- Results in running network functions

THE LINUX FOUNDATION

WNDRVR

How Nephio Works



Utilizing Kubernetes as the automation control plane at each layer of the stack simplifies the overall automation and enables declarative management with active reconciliation for the entire stack.

We can broadly think of three layers in the stack,

1) cloud infrastructure,
2) workload (network function) resources, and
3) workload (network function) configuration.

Nephio is establishing open, extensible Kubernetes Custom Resource Definition (CRD) models for each layer of the stack, in conformance to the **3GPP & O-RAN standard.**

What are Kubernetes Operators

- Custom controllers that extend the functionality of Kubernetes.
- Developed using the Kubernetes Operator SDK and are built on top of the Kubernetes API.
- Watch for changes to Kubernetes objects, such as pods, services, or deployments, and take actions in response to those changes.
- Automate a variety of tasks, such as deploying and scaling applications, managing databases, monitoring and logging, and more.
- They are designed to simplify and automate complex operations, reduce human error, and increase the reliability and efficiency of Kubernetes applications.
- Operators can be created using different programming languages and can be installed on a Kubernetes cluster using Kubernetes manifests. There are also public operator catalogs, such as the Operator Hub, where users can find and install operators created by the community.

Helm charts vs Kubernetes Operators

- Helm is a package manager for Kubernetes.
- Very popular amongst the K8s users across the globe.
- An open-source tool, maintained by the Cloud Native Computing Foundation (CNCF), templates Kubernetes components and bundles them into packages -- known as charts -- that can be versioned and shipped to clusters for deployment.

- Operators represent an evolution of package manager, taking it a step further to include the stages of the application lifecycle after initial deployment.
- As such, Kubernetes operators complement Helm charts.
- Users can build operators from Helm charts without writing any code by using the Kubernetes operator software development kit (SDK).

WNDRVR

Benefits of using operators

- Simplify complex tasks
- Increased reliability and efficiency
- Standardize operations
- Better resource utilization
- Easier upgrades and maintenance

How do operators work

- Kubernetes operators work by extending the Kubernetes API with custom resources and controllers.

  Here's how operators typically work:

  - Custom resources: Operators define custom resources, which are extensions of the Kubernetes API
  - Watch for changes: Controller watches for changes to the custom resources using Kubernetes' built-in event-driven architecture
  - Take action: controller takes action based on the desired state of the custom resource.
  - Report back: controller reports back to the Kubernetes API server on the status of the custom resource.

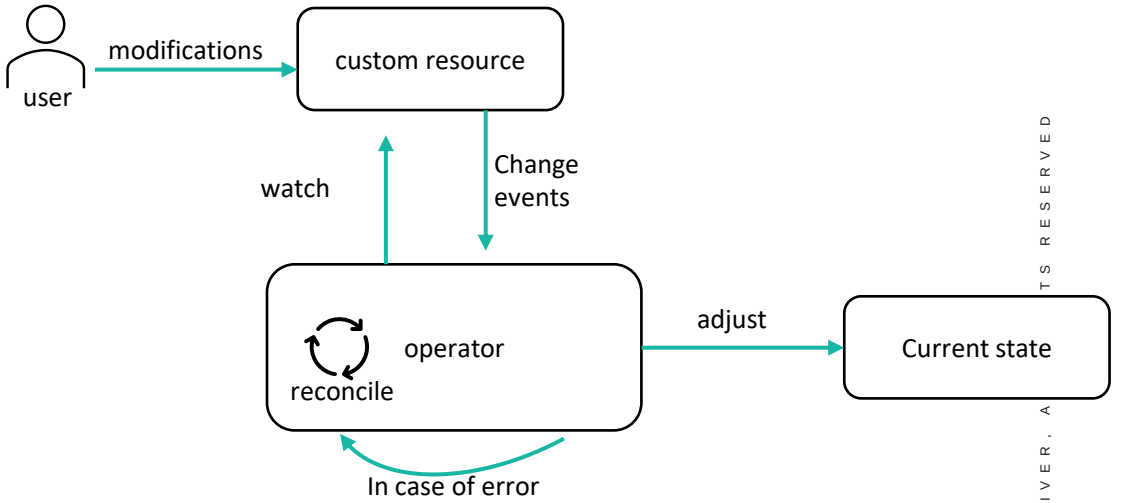
Custom Resource Definition (CRD)

```
apiVersion: <k8sGroup>/<k8sApiVersion>
kind: CustomResourceDefinition
metadata:
  name: <plural>.<group>
spec:
  group: <group>
  names:
    kind: <kind>
    listKind: <listKind>
    plural: <plural>
    singular: <singular>
  scope: <scope>
  version: <version>
```

Custom Resource (CR)

```
apiVersion: <group>/<version>
kind: <kind>
metadata:
  name: <name>
spec:
  <key>: <value>
```

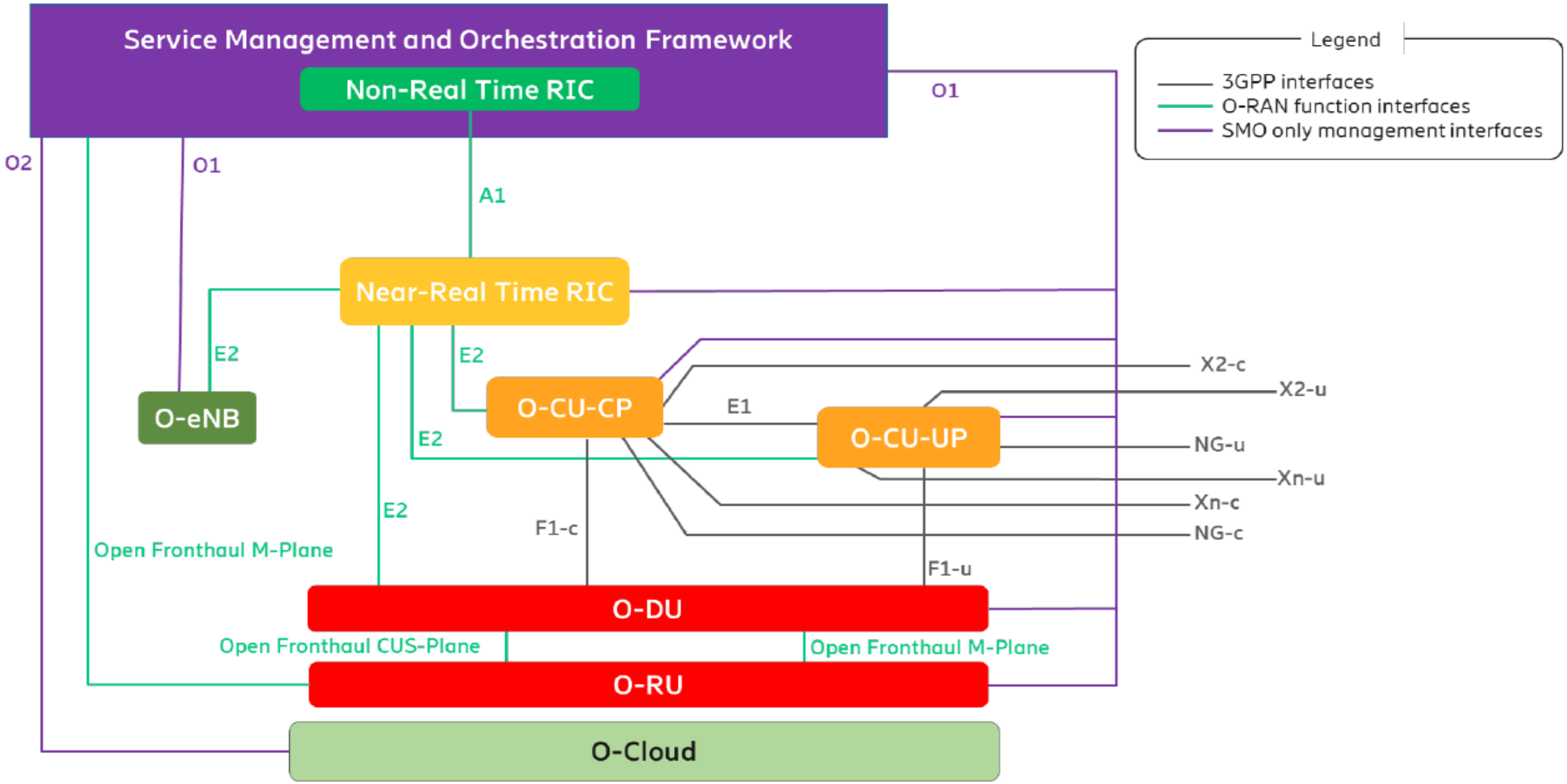
Unique name for custom resource instance



Operator Development

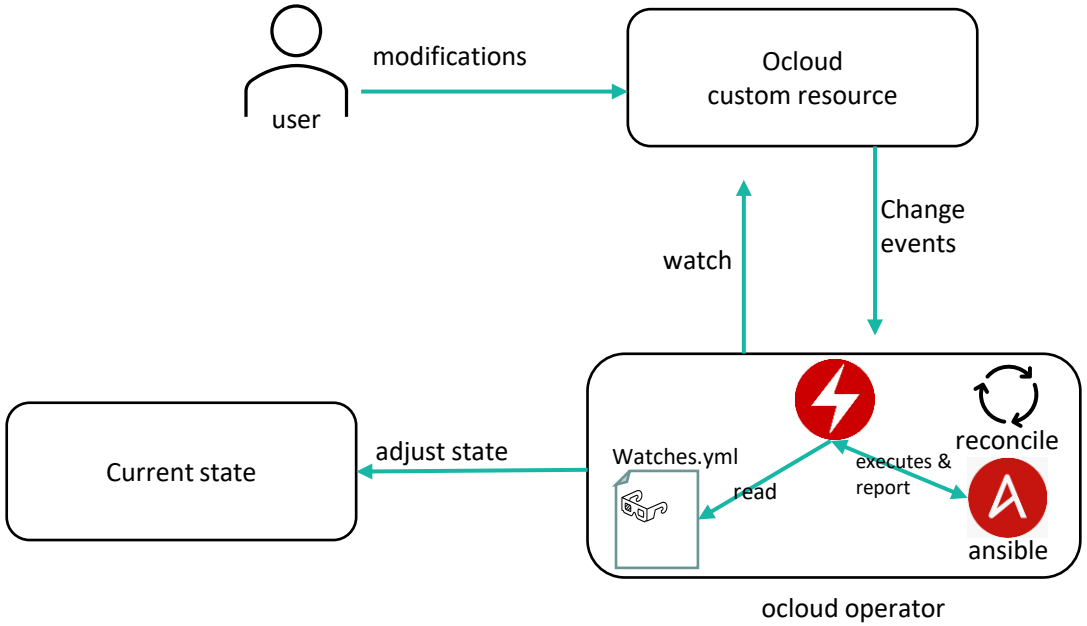
- Identify the task to automate
- Choose an operator framework
- Define custom resources
- Write a controller
- Test operator
- Deploy operator
- Monitor and maintain operator

ORAN Architecture

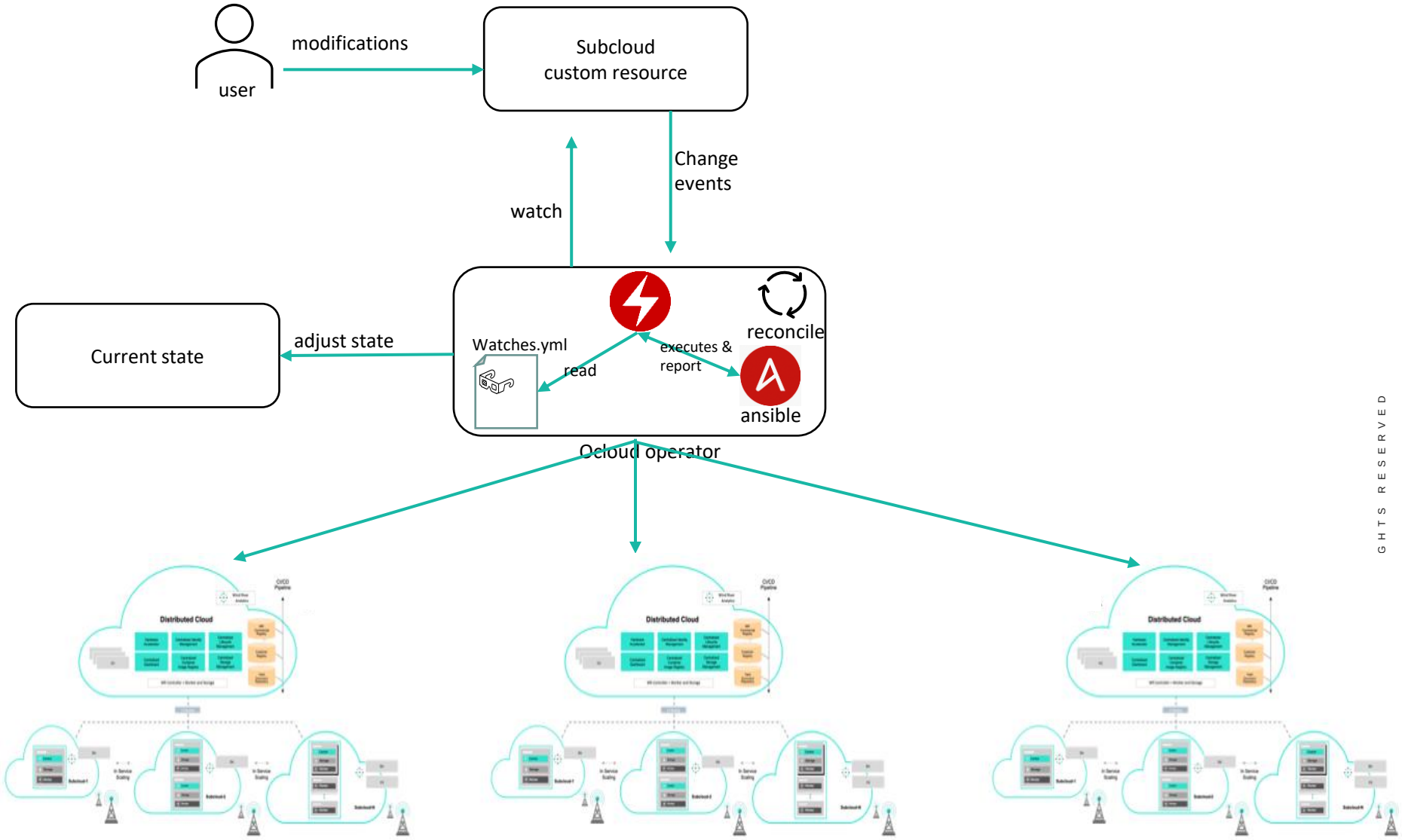


Ocloud Operator

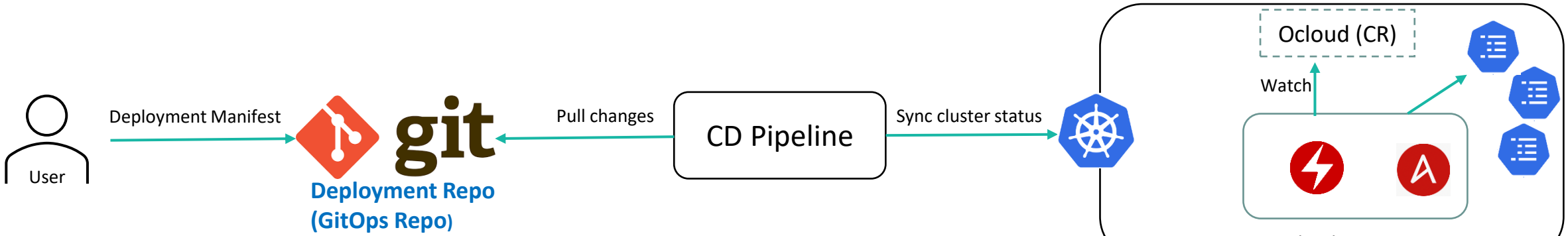
- Ocloud operator provides infrastructure automation, allows user to provision subclouds across multiple edge locations.
- Developed using Ansible operator-sdk
- Define Ocloud Custom Resource Definition (CRD)
- Define Ocloud Custom Resource (CR)
- Watch for changes to the custom resources and takes actions to manage the objects.



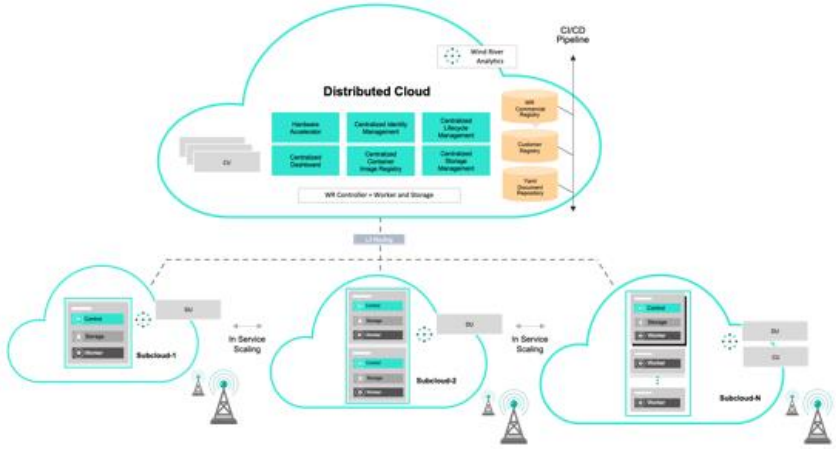
Deployment



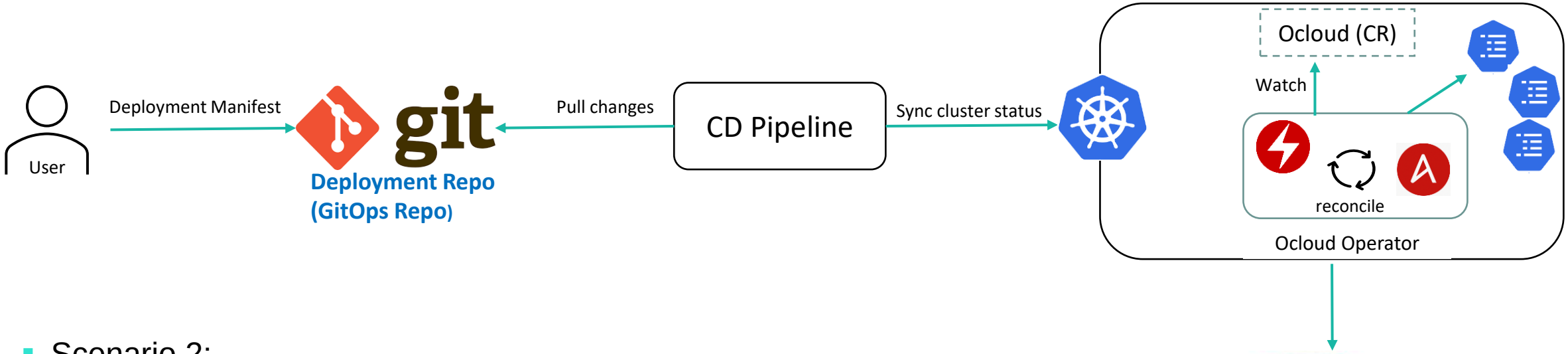
GitOps



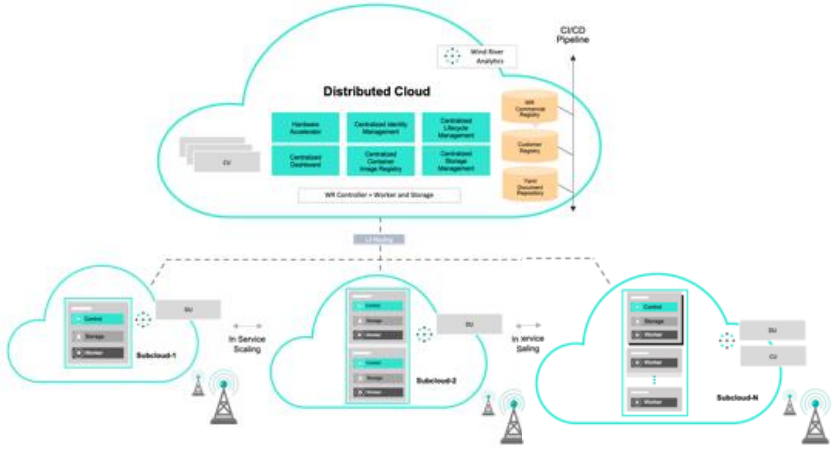
- Scenario 1:
  - User push manifest to Git repository
  - CD pipeline, pull changes and sync to the ks8 cluster
  - Ocloud operator watch Ocloud CR and provision Ocloud.



GitOps



- Scenario 2:
  - Delete Ocloud
  - Ocloud operator reconcile in order to maintain the desired state.



DEMO

Develop Ocloud operator

- Initialize Operator With Ansible
  - operator-sdk new ocloud-operator --api-version=ocloud.operator.com/v1alpha1 --kind=Ocloud --type=ansible
- Automate With Ansible
  - Create new roles and playbooks or reuse an existing one to deploy Ocloud
- Define a watches file
  - Map a Kubernetes object to your Ansible content
- Build Ocloud Operator
  - operator-sdk build Ocloud-operator:v0.0.1
- Deploy Ocloud Operator to a Kubernetes Cluster

Custom Resource

```yaml
---
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: oclouds.cache.ocloud.operator.com
spec:
  group: cache.ocloud.operator.com
  names:
    kind: Ocloud
    listKind: OcloudList
    plural: oclouds
    singular: ocloud
  scope: Namespaced
  versions:
  - name: v1alpha1
    schema:
      openAPIV3Schema:
        description: Ocloud is the Schema for the oclouds API
        properties:
          apiVersion:
            description: 'APIVersion defines the versioned schema of
              of an object. Servers should convert recognized schemas
              internal value, and may reject unrecognized values. Mor
            type: string
          kind:
            description: 'Kind is a string value representing the RES
              object represents. Servers may infer this from the endp
              submits requests to. Cannot be updated. In CamelCase. M
            type: string
          metadata:
            type: object
          spec:
            description: Spec defines the desired state of Ocloud
            type: object
            x-kubernetes-preserve-unknown-fields: true
          status:
            description: Status defines the observed state of Ocloud
            type: object
            x-kubernetes-preserve-unknown-fields: true
        type: object
    served: true
    storage: true
    subresources:
      status: {}
```

Custom resource (CRD)

```yaml
apiVersion: cache.ocloud.operator.com/v1alpha1
kind: Ocloud
metadata:
  labels:
    app.kubernetes.io/name: ocloud
    app.kubernetes.io/instance: ocloud-sample
    app.kubernetes.io/part-of: ocloud-operator
    app.kubernetes.io/managed-by: kustomize
    app.kubernetes.io/created-by: ocloud-operator
  name: ocloud-sample
spec:
  name: subcloud1
  system_controller_address: 128.224.248.160
  cloud_url_type: http
  system_mode: simplex
  timezone: Europe/Berlin
  location: Ismaning
  dns_servers:
       - 128.224.200.11
       - 147.11.57.128
  external_oam_subnet: 128.224.248.0/23
  external_oam_gateway_address: 128.224.248.1
  external_oam_floating_address: 128.224.248.54

  management_subnet: 10.10.17.0/24
  management_start_address: 10.10.17.11
  management_end_address: 10.10.17.50
  management_gateway_address: 10.10.17.1

  bootstrap_address: 128.224.248.54

  systemcontroller_gateway_address: 10.10.25.1

  docker_registries:
    k8s.gcr.io:
      url: registry.central:9001/k8s.gcr.io
    gcr.io:
      url: registry.central:9001/gcr.io
    quay.io:
      url: registry.central:9001/quay.io
    docker.io:
      url: registry.central:9001/docker.io
    docker.elastic.co:
      url: registry.central:9001/docker.elastic.co
    ghcr.io:
      url: registry.central:9001/ghcr.io
    defaults:
      username: sysinv
      password: 0v**0F5*lmx+sx9W
      type: docker

  wipe_ceph_osds: false
```

Custom resource (CR)

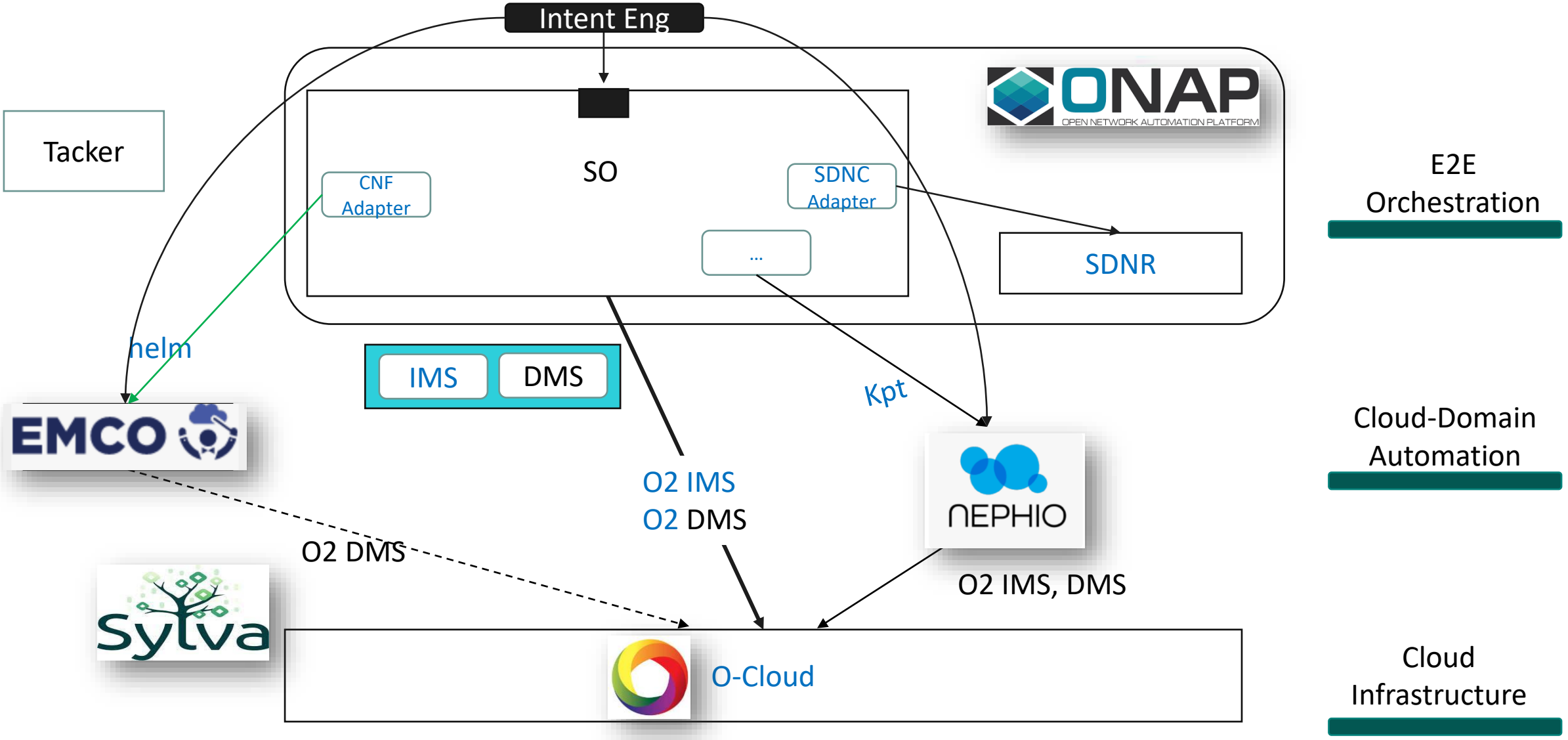
Mapping Kubernetes events to Ansible

- The “watches” file (watches.yaml) maps a Kubernetes object to the Ansible automation.
  - Associates the Kubernetes Group, Version, Kind (GVK) to an Ansible Role or Playbook.
  - The Operator SDK binary watches the cluster for matching events defined in the watches.yml
  - Executes the associated Ansible content when an event occurs

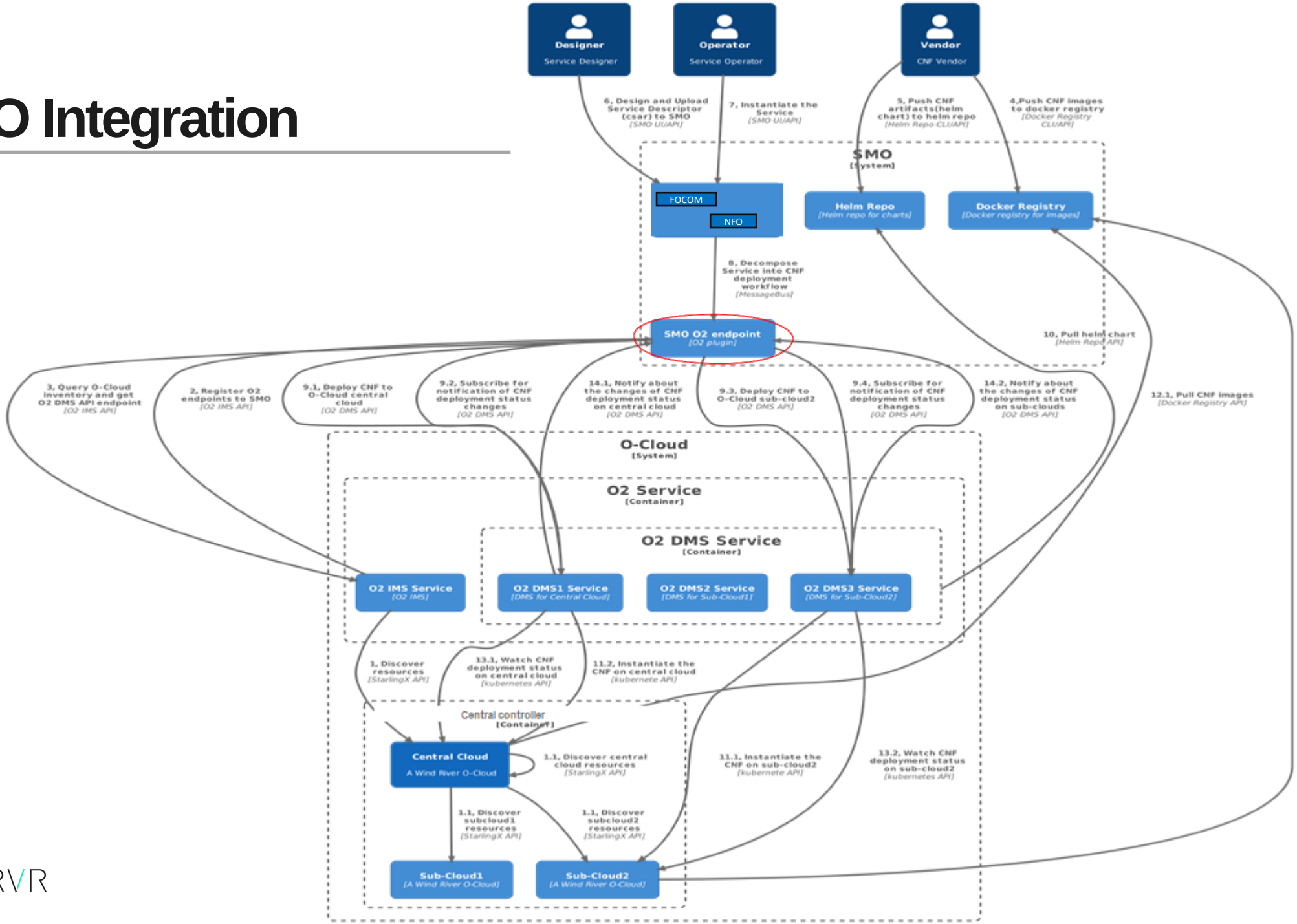
```
# watches.yaml
---
version: v1alpha1
group: ocloud.example.com
kind: Ocloud
playbook: /path/to/playbook
```

Next Steps

Integration with other Codes



SMO Integration



WNDRVR

Future Developments

- Ocloud DMS
- Extend to LCM operations
- Git-Ops driven Automation
- Intent driven Orchestration
- Control loop
- Bring it to larger audience and participation
- Integration to the broader communities
- Demos in ORAN, Nephio, ONAP…

Thanks

Backup slides

Nephio – A quick Intro

What is Nephio?

Nephio is a Kubernetes-based intent-driven automation of network functions and the underlying infrastructure that supports those functions. It allows users to

- express high-level intent, and provides intelligent,
- declarative automation that can set up the cloud and edge infrastructure,
- render initial configurations for the network functions, and
- then deliver those configurations to the right clusters to get the network up and running.

What Problem Does It Solve?

Nephio is intended to address the initial provisioning of the network functions and the underlying cloud infrastructure, and also provide Kubernetes-enabled reconciliation to ensure the network stays up through failures, scaling events, and changes to the distributed cloud.

Nephio 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.

Ocloud Operator, Development and Deployment