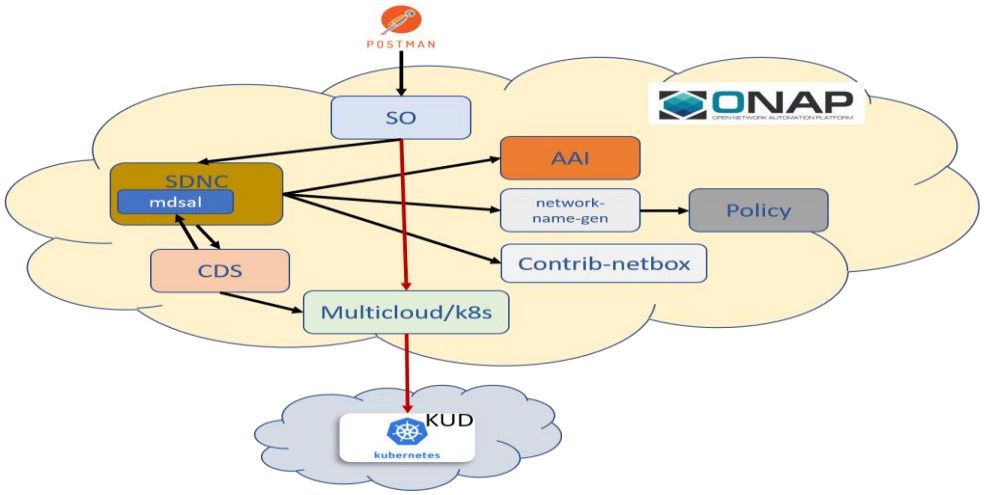
# ONAP E2E flow

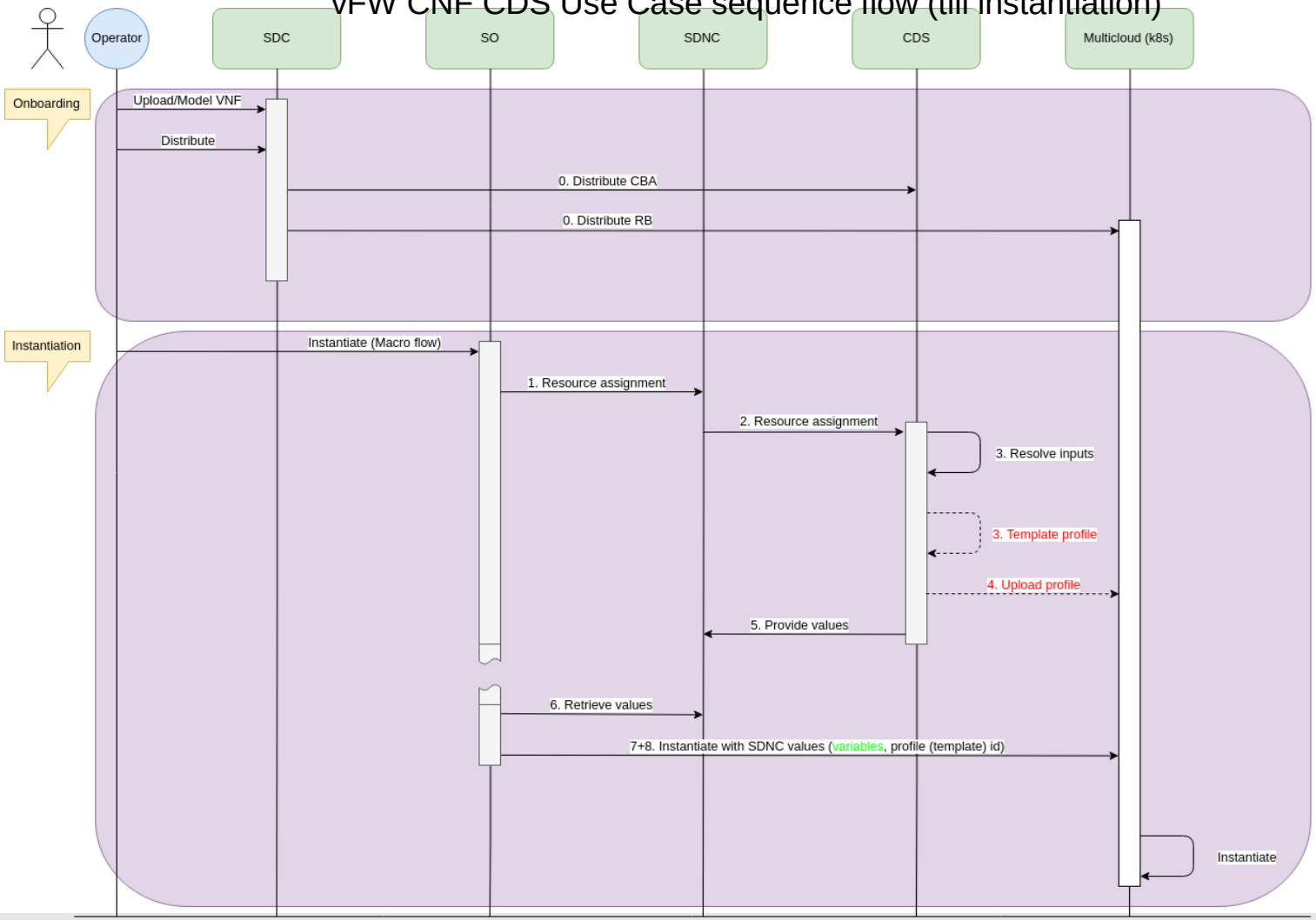
# ONAP Specific Workflows

## CNF Use Case - vFW CNF with CDS

vFW CNF CDS Use Case Runtime interactions.




vFW CNF CDS Use Case sequence flow (till instantiation)



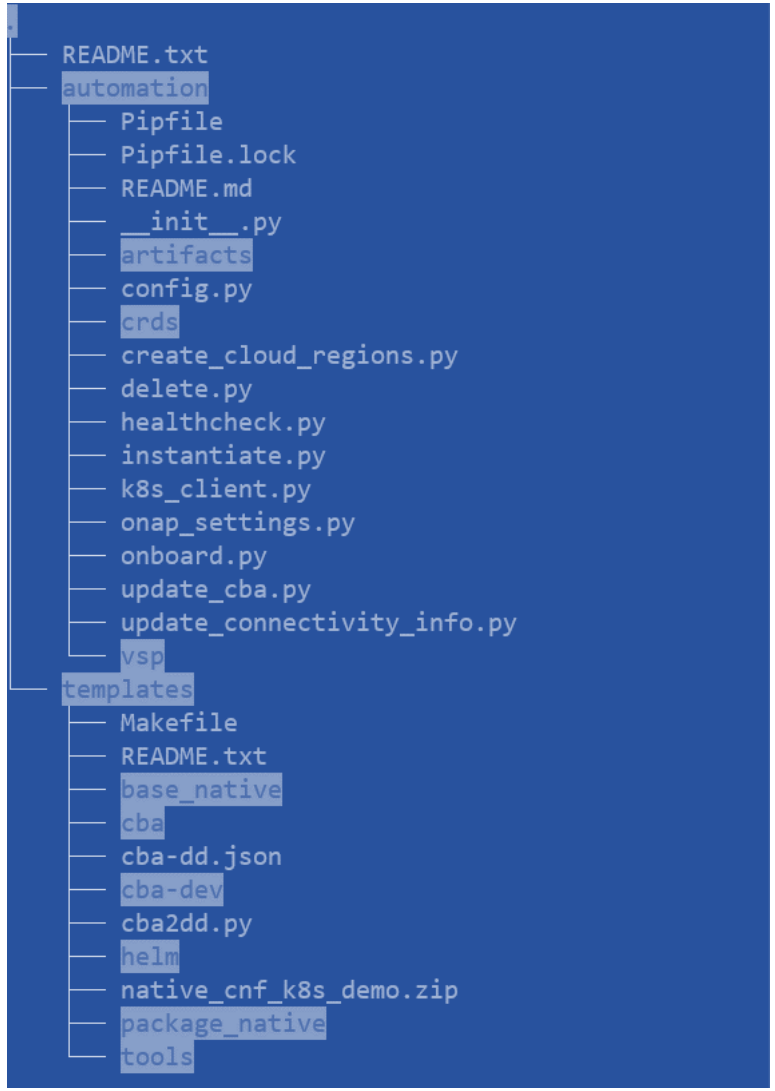
# Automation for CNF Use Cases – from different sources

CDS CBA Structure:

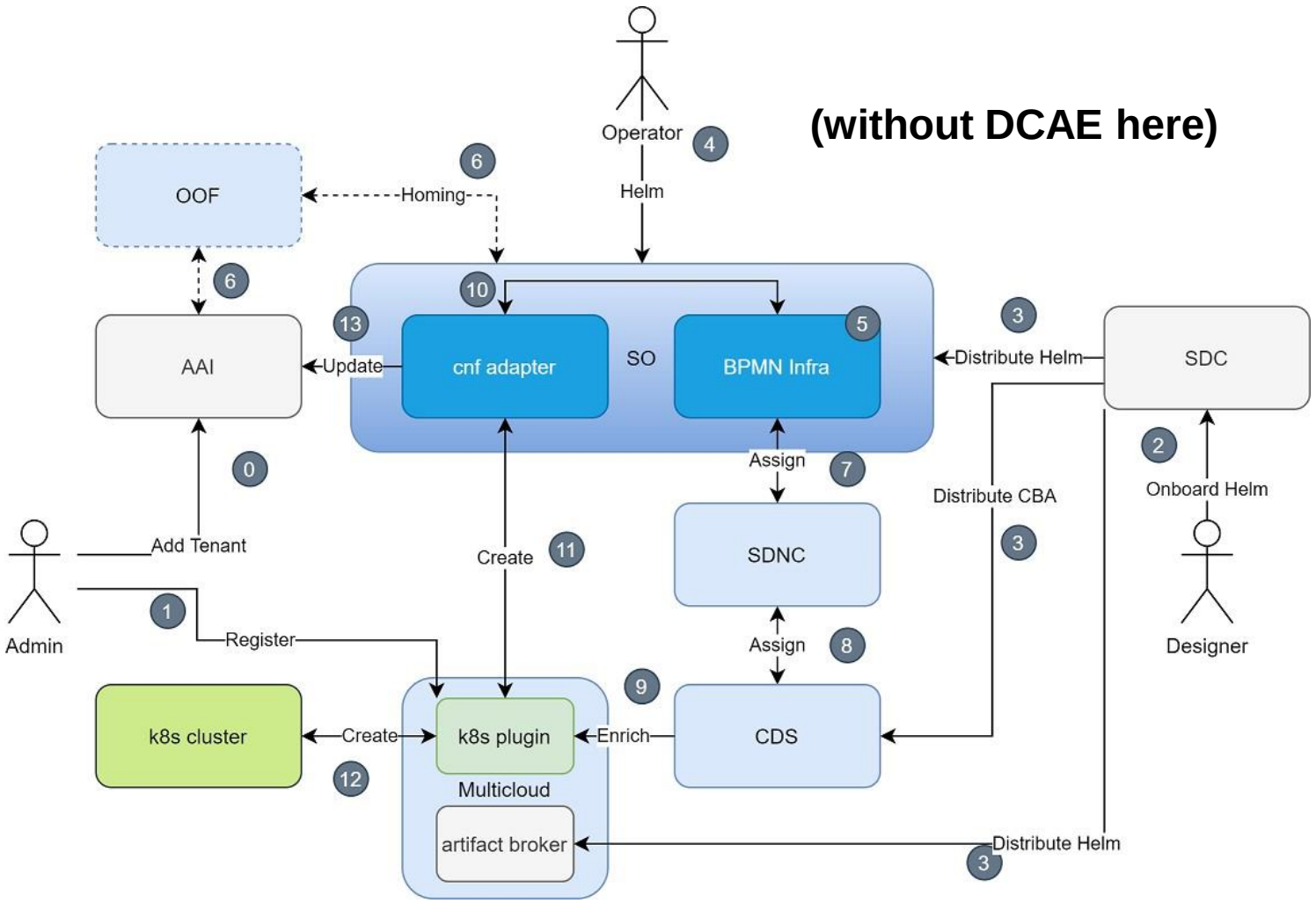
- Templates Folder
  - Build and Test CBA
  - Build VSP with make
- Automation Folder
  - Step-by-step README
  - Create K8S Region
  - Onboard Service
  - Instantiate Service
  - Delete Service
  - Check Health of CNF

https://git.onap.org/demo/tree/heat/vFW_CNF_CDS/templates

https://git.onap.org/demo/tree/heat/vFW_CNF_CDS/automation



# E2E Flow Details




- **Helm Onboarding** **[SDC]**
- **Day 0/1 Customization** **[CDS/MC]**
- **Instantiation** **[SO/MC]**
- **Day 2 Configuration** **[CDS/MC]**
- **State Synchronization** **[SO/AAI/MC]**
- **Healthcheck** **[SO/CDS/MC]**
- **Upgrade** **[SO/CDS/MC]**
- **Multi-cluster deployment** **[SO/MC]**
- **VNF/PNF Integration** **[ALL]**
- **E2E Service Automation** **[SDK]**

# CDS Blueprint Design Steps

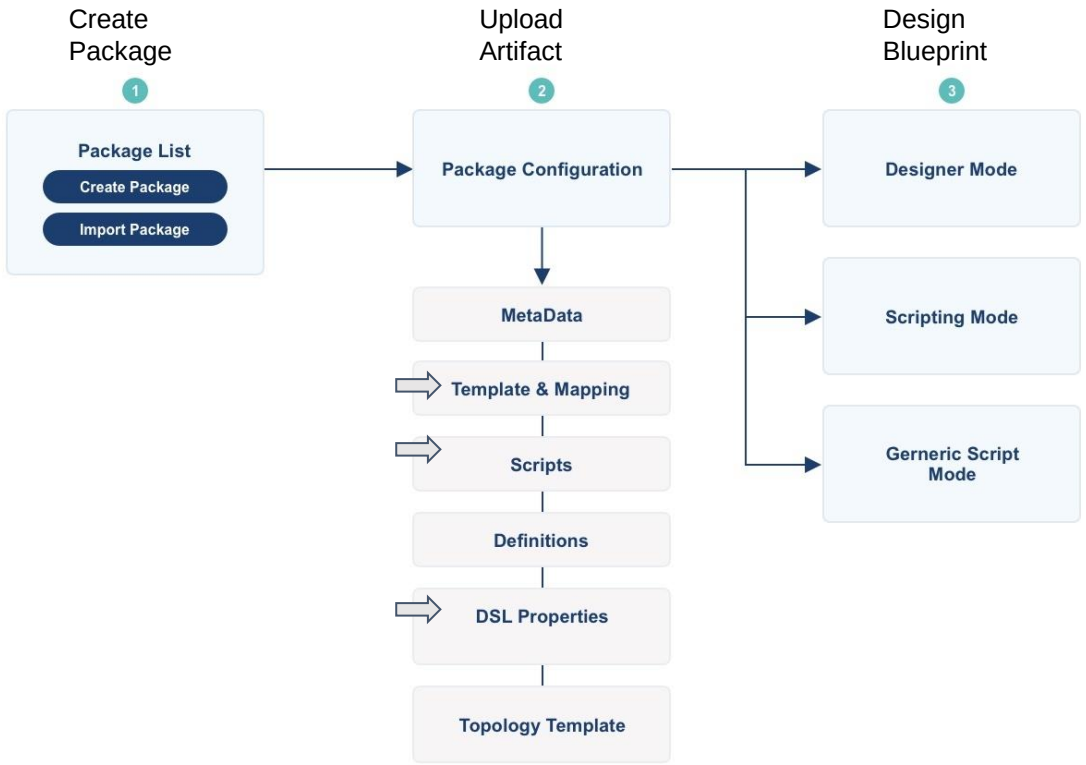
ccsdk-cds/components/model-catalog/definition-type/starter-type/node_type at master · onap/ccsdk-cds · GitHub

**Step 1: Create required Data dictionaries**

- A data dictionary models the how a specific resource can be resolved.
- A resource is a variable/parameter in the context of the service. It can be anything, but it should not be confused with SDC or Openstack resources.
- A data dictionary can have multiple sources to handle resolution in different ways.
- The main goal of data dictionary is to define re-usable entity that could be shared.

| Property | Description | Scope |
|---|---|---|
| updated-by | The creator | Mandatory |
| tags | Information related | Mandatory |
| sources | List of resource source instance | Mandatory |
| property | Defines type and description, as nested JSON | Mandatory |
| name | Data dictionary name | Mandatory |

**Step 2: Create CBA including following Steps**

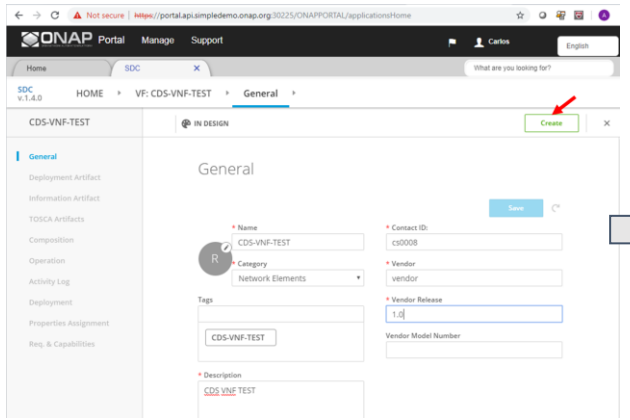



https://git.onap.org/demo/tree/heat/vFW_CNF_CDS

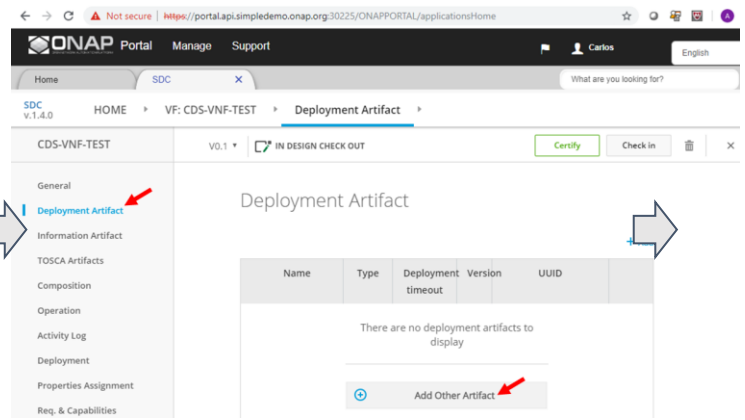
# CDS Distribution with SDC

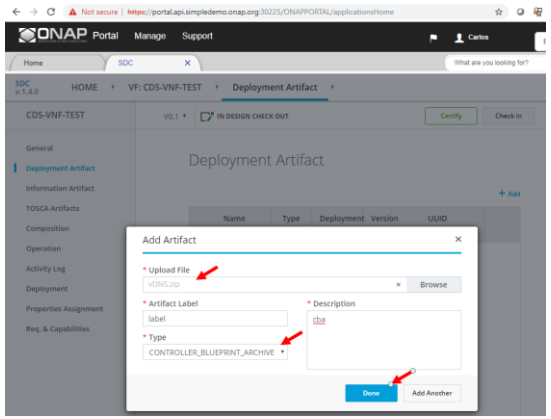
https://wiki.onap.org/display/DW/User+Guide#UserGuide-1737652736

Create the VF resource

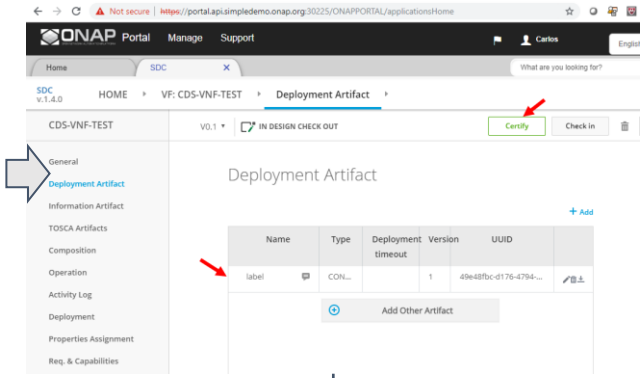


Click on **Deployment Artifact**, then **Add other arifacts**, and select you CBA

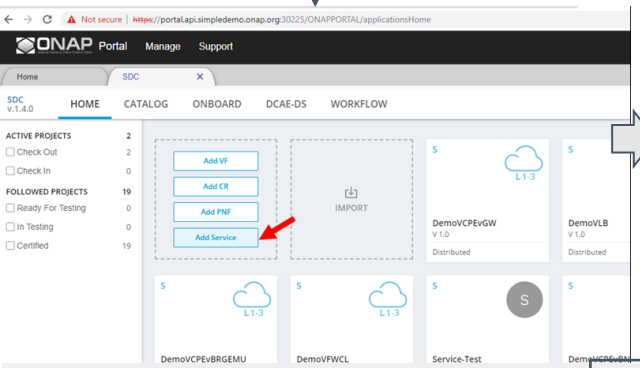




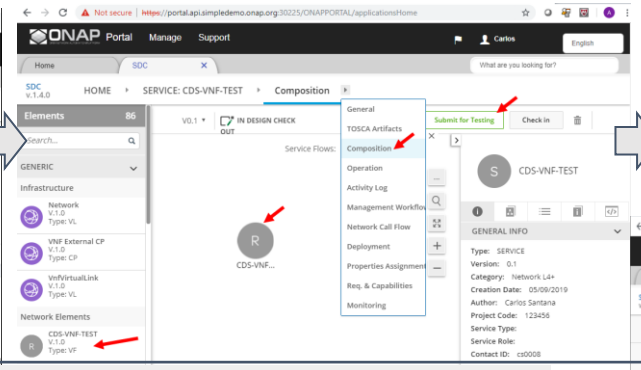
Check the artifact is uploaded OK, and click on **Certify**.



Create a new service model, and add the newly created VF (including CBA artifact) to the new service model. Click on "Add Service"



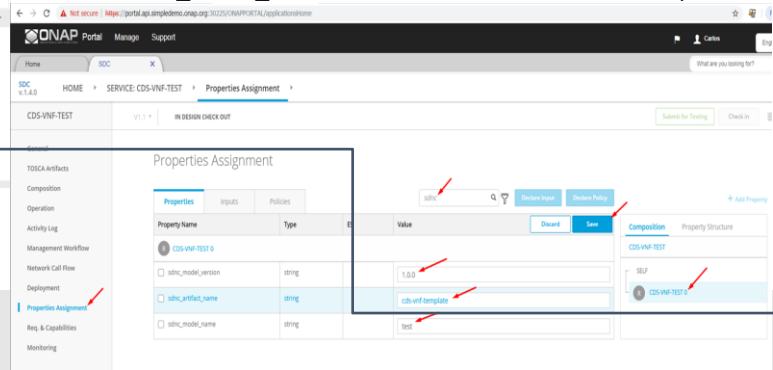
Click on "Composition", and drag the VF we created from the palette on the left onto the canvas in the middle. Then, click on "Submit for Testing".



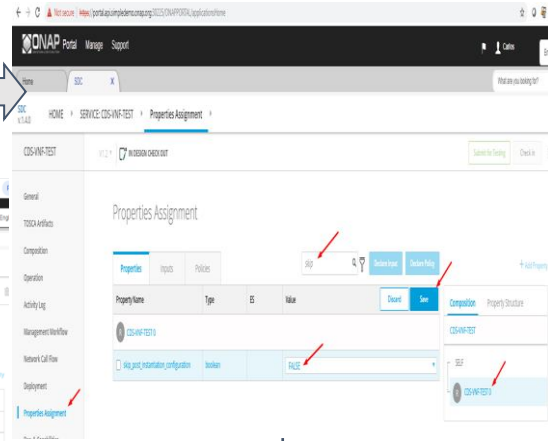
Click on Properties Assignments, then click on the service name, e.g. "CDS-VNF-TEST" from the right bar.

Type "sdnc" in the filter box, and add the **sdnc_model_name**, **sdnc_model_version**, and **sdnc_artifact_version**, and click "Save".

- sdnc_model_name - This is the name of the blueprint (e.g. CBA name)
- sdnc_model_version - This is the version of the blueprint



Type "skip" in the filter box, and set "skip post instantiation" to FALSE, then click "Save".



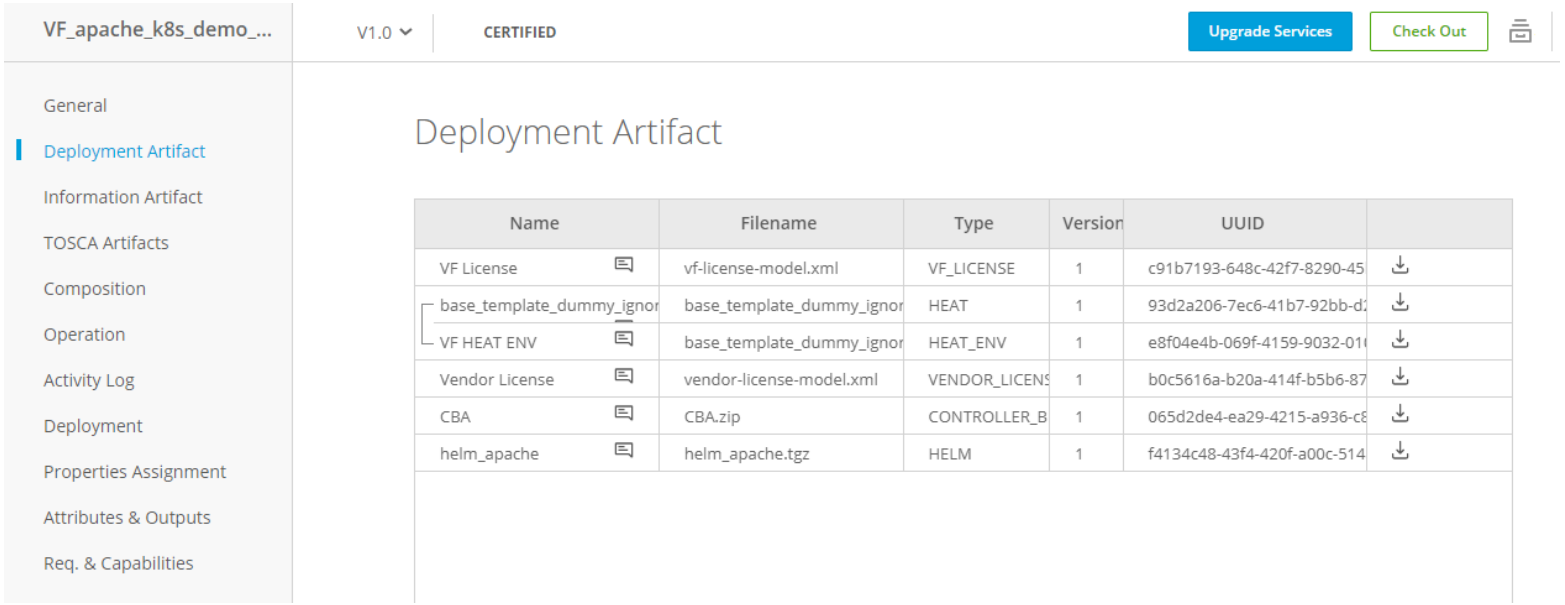
Login as Tester (jm0007/demo123456!) and accept the new service.

Login as Governor (gv0001/demo123456!) and approve for distribution.

Login as Operator (op0001/demo123456!) and click on "Distribute"

# CNFO Onboarding

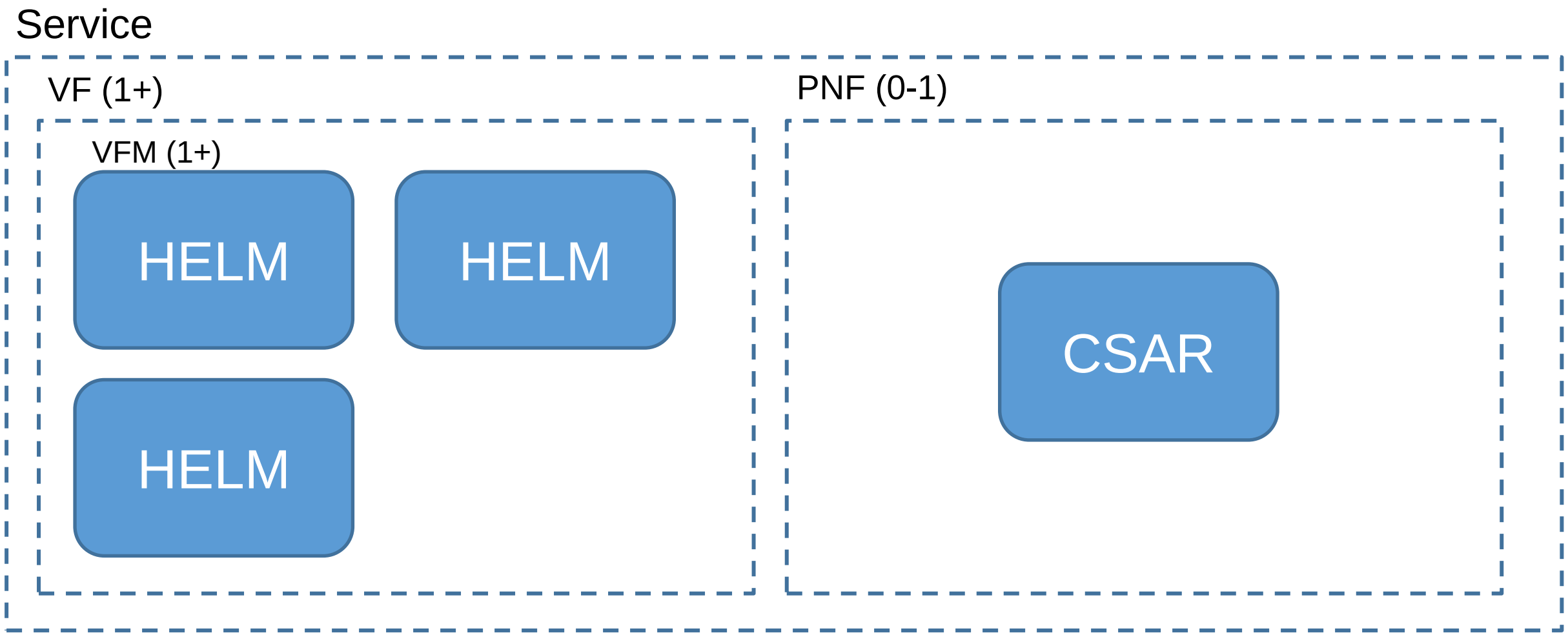
```
{
    "name": "simpleCNF",
    "description": "",
    "data": [
        {
            "file": "CBA.zip",
            "type": "CONTROLLER_BLUEPRINT_ARCHIVE"
        },
        {
            "file": "helm_apache.tgz",
            "type": "HELM",
            "isBase": "true"
        }
    ]
}
```



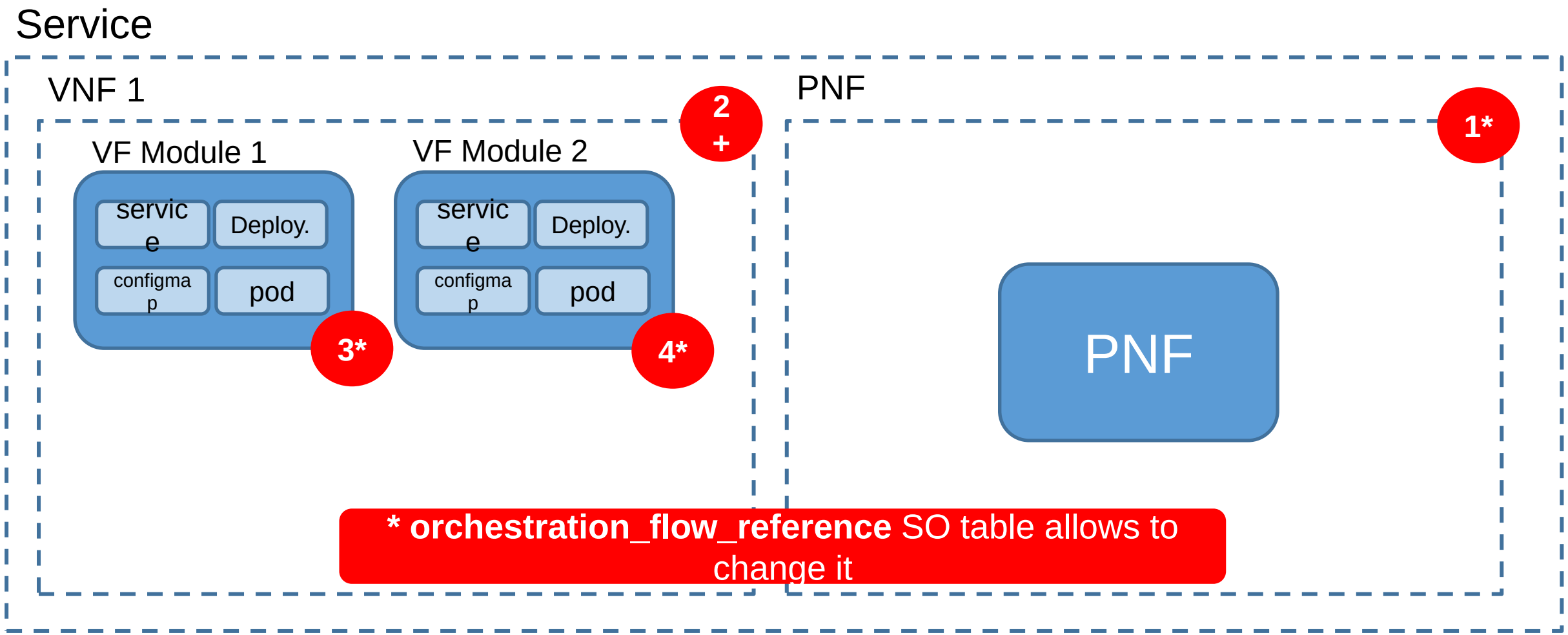

- Standard Simple VSP Package (ZIP)
- CBA is crucial and mandatory for CNFO
- In the future may be replaced with ASD

We will see the complete package of a CBA in the coming slides

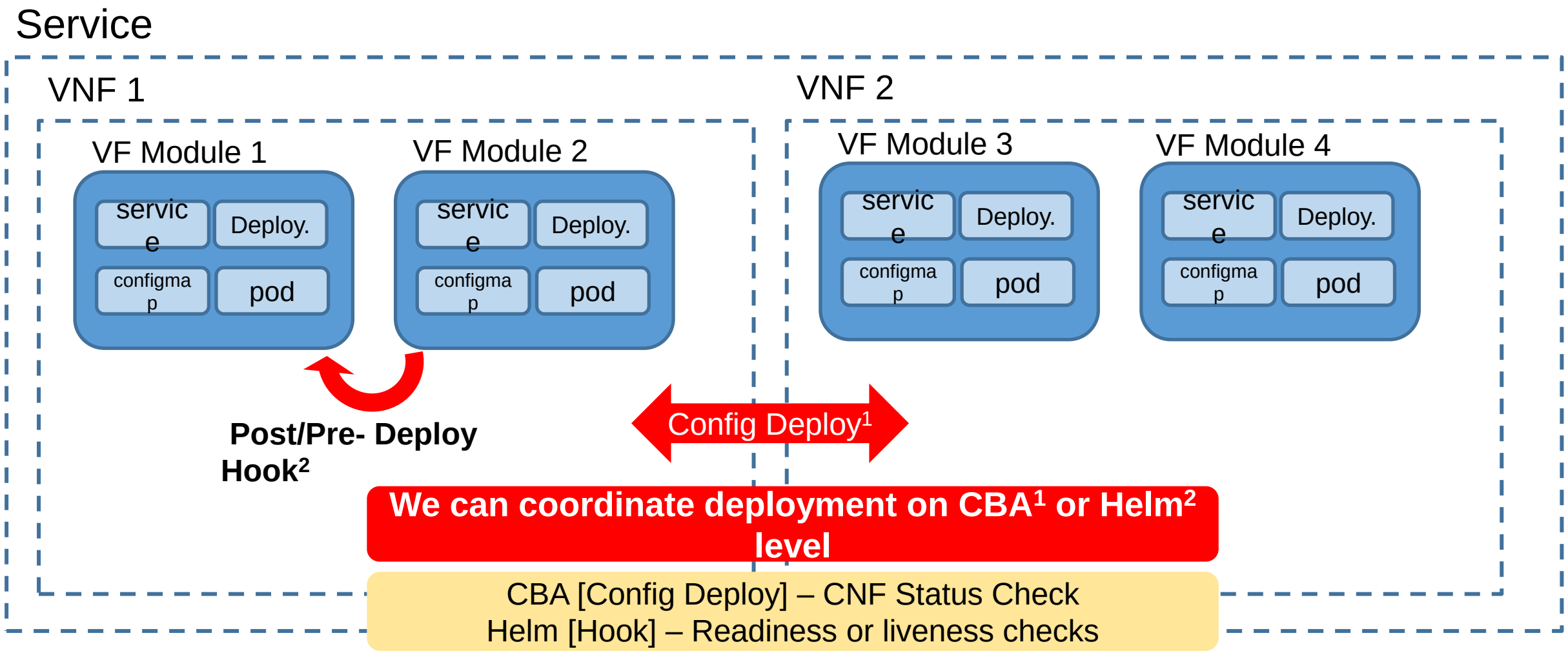
# ONAP modeling concept (SDC)



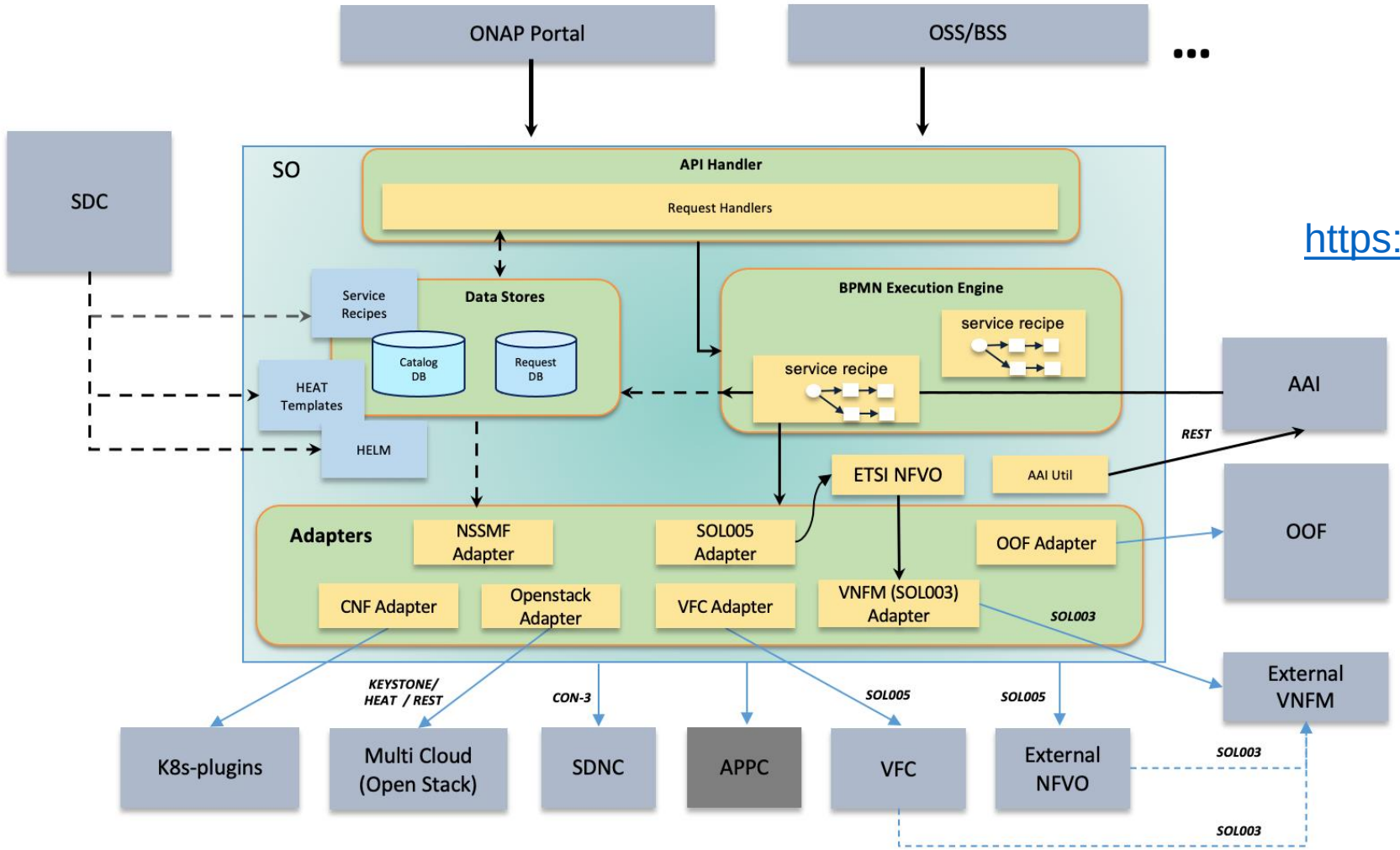
# CNF/PNF Coordination (1)



# CNF/PNF Coordination (3)

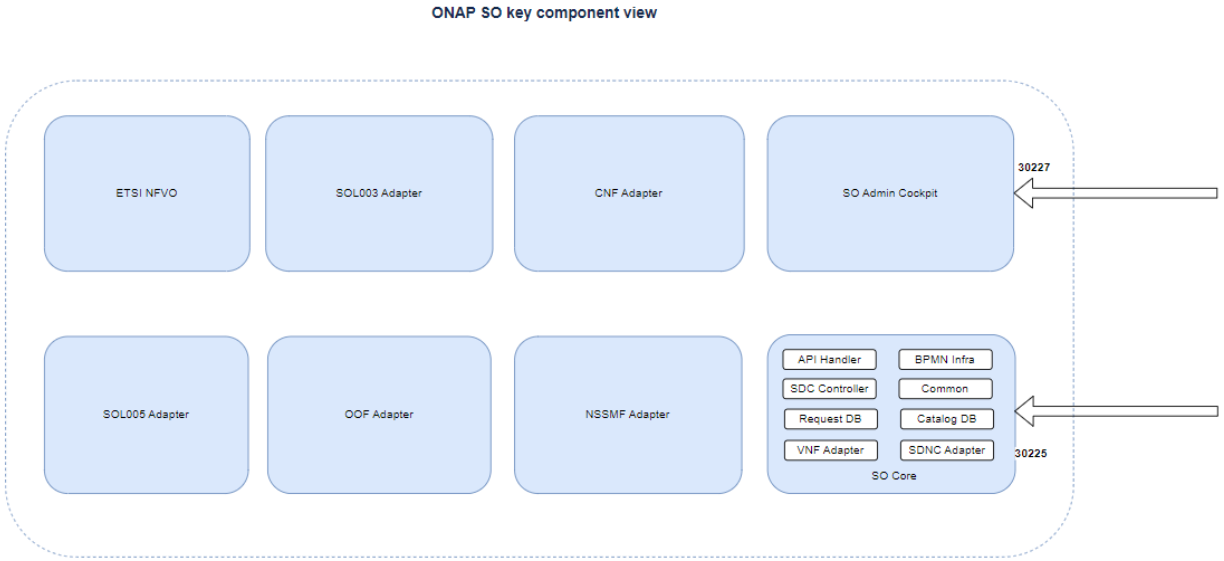


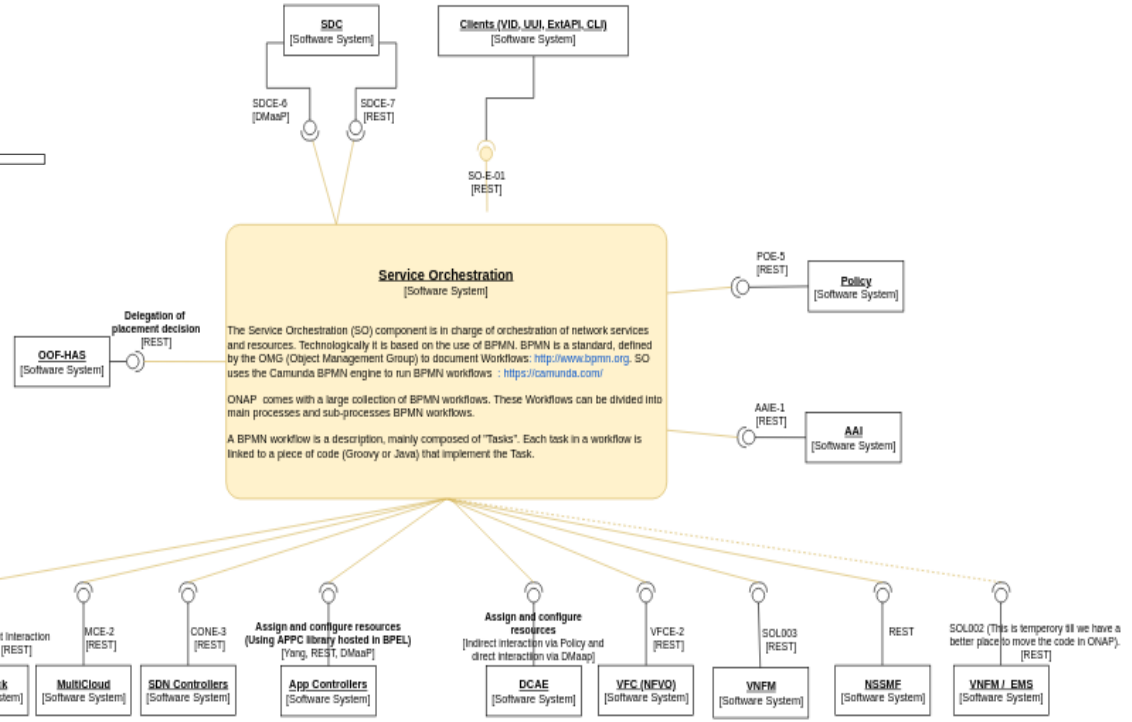
# Understand SO




https://github.com/onap/so

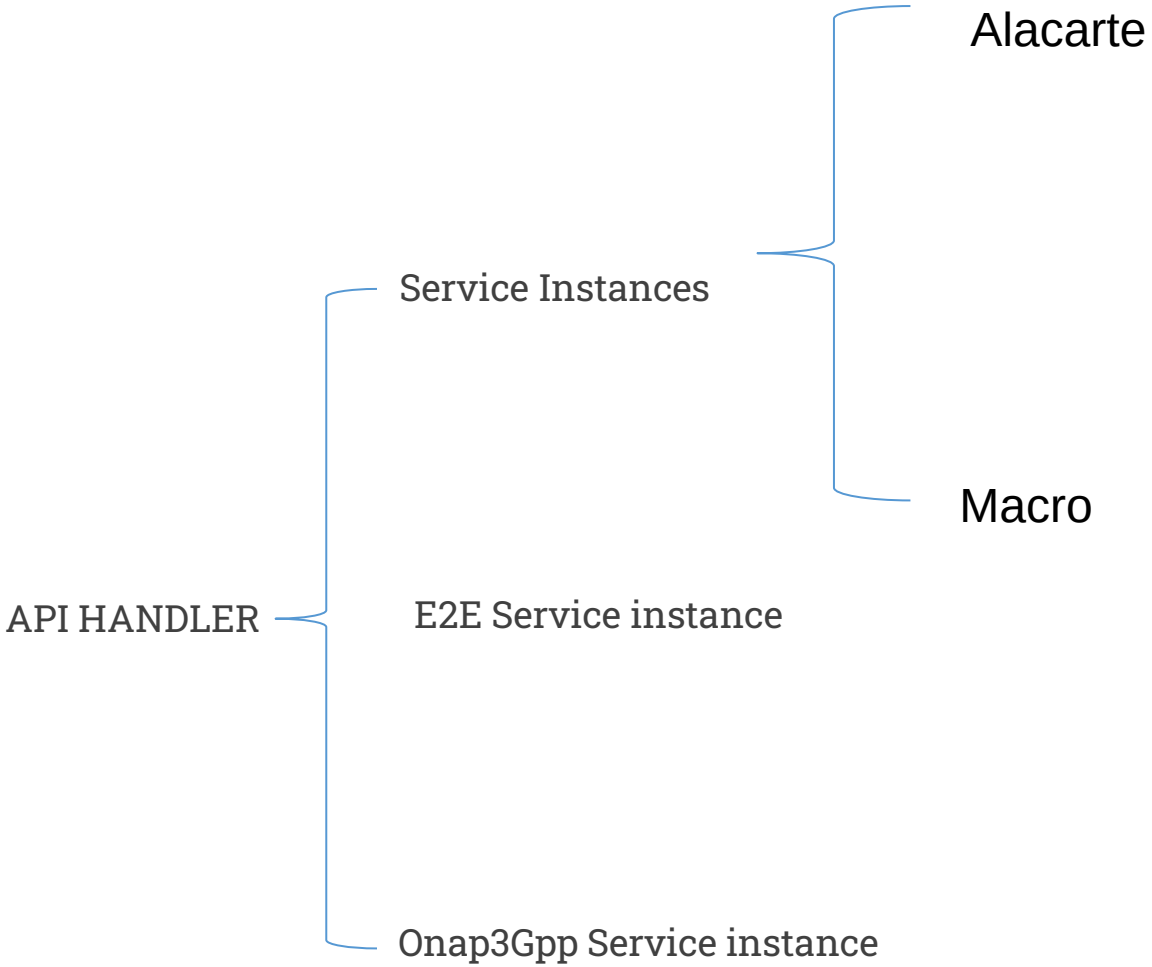
# Understand SO cont




https://wiki.onap.org/display/DW/ARC+Service+Orchestrator+Component+Description+-+London-R12

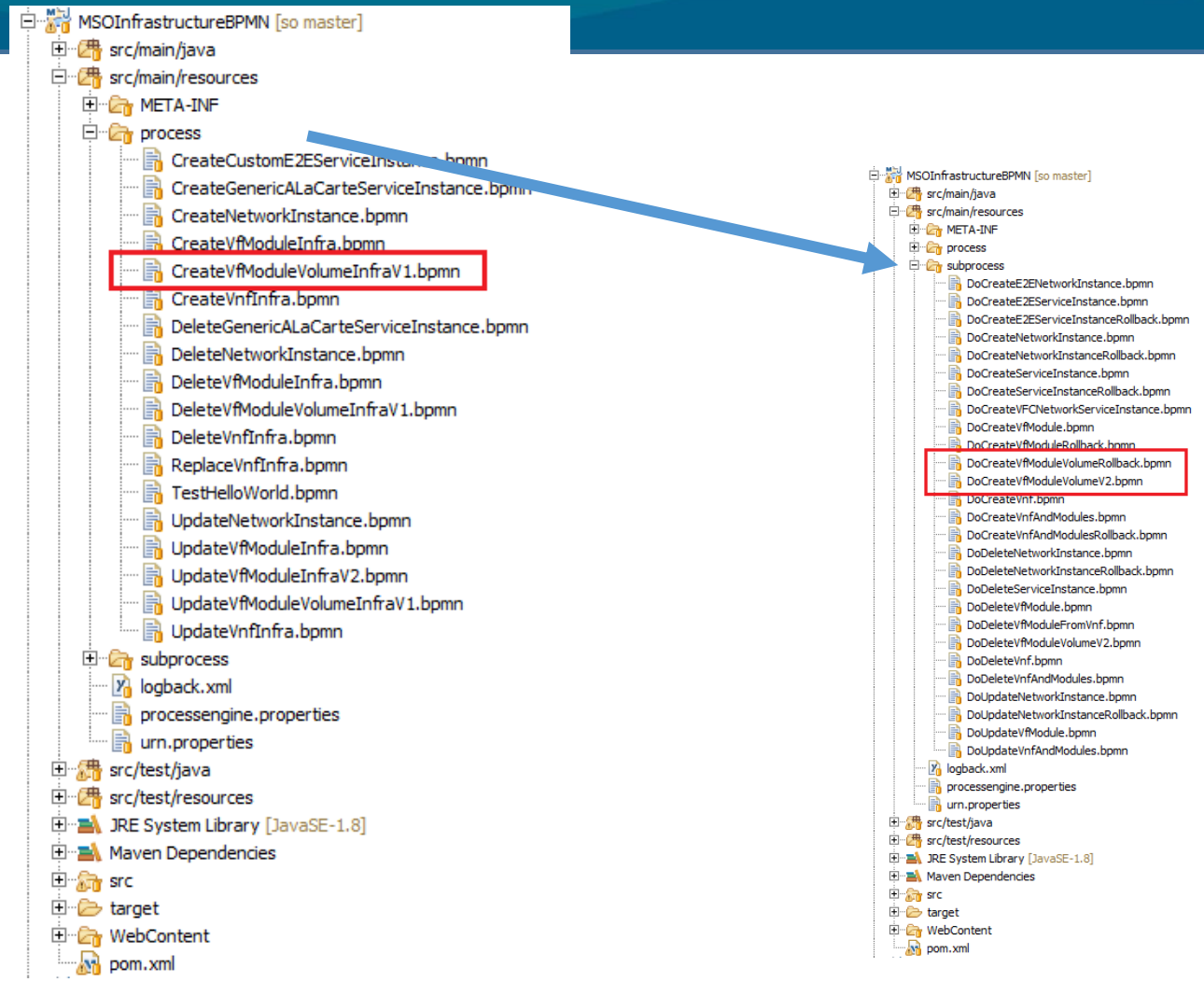
# Key Flows in SO



# Key Flows in SO

a unique requestid is generated corresponding to every request, and is checked whether that particular requestid exists in request db

1.Checks whether the service already exists in catalogdb **http://catalog-db-adapter:8082/service/{modelNameVersionId}** (GET).
2.Checks the service recipe table with the modelNameVersionId and the action to be performed **http://catalog-db-adapter:8082/serviceRecipe/search/findFirstByServiceModelUUIDAndAction?serviceModelUUID={modelNameVersionId}&action=createInstance**




https://docs.onap.org/projects/onap-so/en/latest/developer_info/BPMN_Project_Structure.html
https://github.com/onap/so/tree/master/bpmn/so-bpmn-infrastructure-flows/src/main/resources/process

# Macro Flow – PNF PNP as an Example

## orchestration_flow_reference table in the SO catalog table

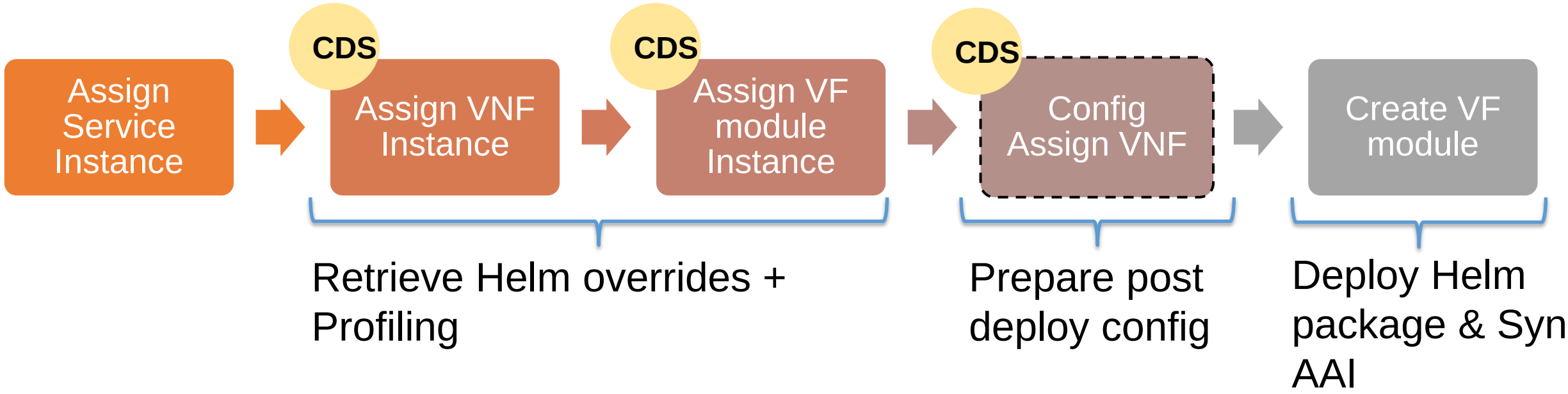
Sequence in Service-Macro-Create flow

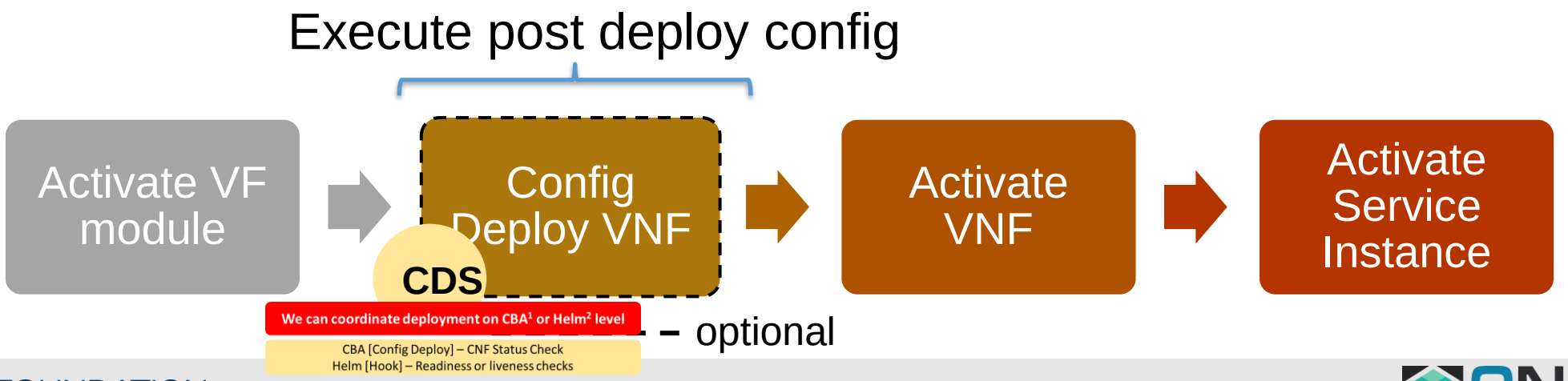
1.AssignServiceInstanceBB
2.CreateNetworkCollectionBB
3.AssignNetworkBB
4.AssignVnfBB
5.AssignVolumeGroupBB
6.AssignVfModuleBB
7.AssignPnfBB
8.WaitForPnfReadyBB
9.ControllerExecutionBB (action: configAssign, scope: pnf)
10.ControllerExecutionBB (action: configDeploy, scope: pnf)
11.ActivatePnfBB
**12.ConfigAssignVnfBB**
13.CreateNetworkBB
14.ActivateNetworkBB
15.CreateVolumeGroupBB
16.ActivateVolumeGroupBB
17.CreateVfModuleBB
18.ActivateVfModuleBB
**19.ConfigDeployVnfBB**
20.ActivateVnfBB
21.ActivateNetworkCollectionBB
22.ActivateServiceInstanceBB

https://github.com/onap/so/tree/master/bpmn/so-bpmn-building-blocks/src/main/resources/subprocess/BuildingBlock

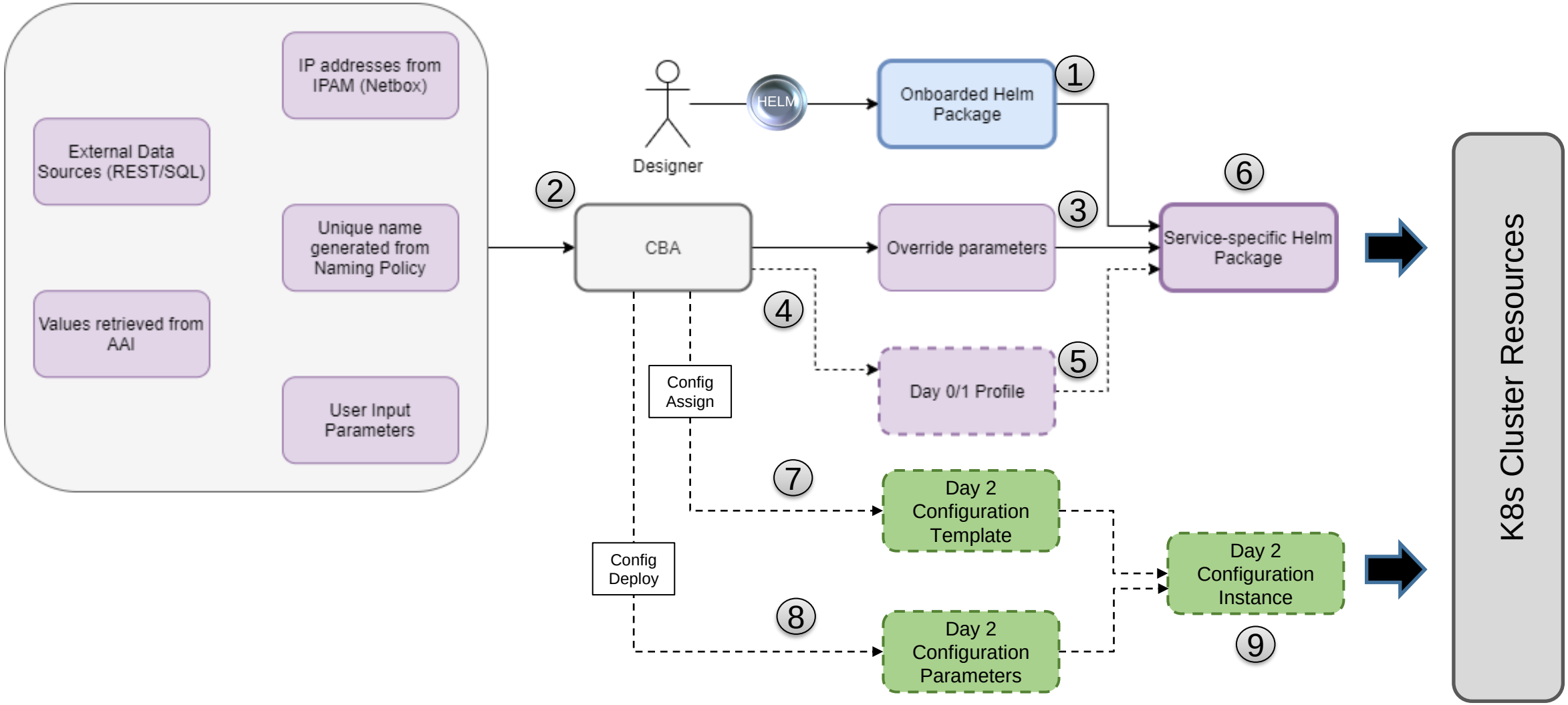
https://docs.onap.org/projects/onap-so/en/latest/developer_info/pnf_pnp_workflow_migration_to_BB/Building_Block_based_PNF_PnP_flows.html

# CNF Instantiation (macro mode)



# Helm Package Day 0/1 + Day2



# CNF Day 0 – Helm Enrichment

```
"resource-assignment": {
    "steps": {
        "resource-assignment": {
            "description": "Resource Assign Workflow",
            "target": "resource-assignment",
            "activities": [
                {
                    "call_operation": "ResourceResolutionComponent.process"
                }
            ],
            "on_success": [
                "profile-upload"
            ]
        },
        "profile-upload": {
            "description": "Generate and upload K8s Profile",
            "target": "k8s-profile-upload",
            "activities": [
                {
                    "call_operation": "K8sProfileUploadComponent.process"
                }
            ]
        }
    },
```

- CNF instance based
- Modifies Helm package from VSP
- K8s Profile Creation & Upload
  - Native mechanisms in CDS
  - Customizable by CBA
- Modification of Helm values
- Customization of labels
- Selection of k8s namespace
- Modification of Helm templates
- Provisioning of new Helm templates
- New k8s-resource types to

# CNF Day 2 – Config Preparation

```
"config-assign": {
    "steps": {
        "config-setup": {
            "description": "Gather necessary input for config template upload",
            "target": "config-setup-process",
            "activities": [
                {
                    "call_operation": "ResourceResolutionComponent.process"
                }
            ],
            "on_success": [
                "config-template"
            ]
        },
        "config-template": {
            "description": "Generate and upload K8s config template",
            "target": "k8s-config-template",
            "activities": [
                {
                    "call_operation": "K8sConfigTemplateComponent.process"
                }
            ]
        }
    },
```

- CNF instance based
- Config Template (CFT)
  - Helm package
  - Build or modified by CDS
  - We can use VSP Helm as a template
- CFT preparation may be a part of Config-Assign in CDS
- Native mechanisms in CDS
  - Customizable by CBA
- Config Setup merges data
  - CBA
  - AAI i.e. vf-modules info
  - MDSAL – i.e. resolved Day 0
  - K8s – i.e. k8s resource status info
  - Kotlin, Python, REST
  - Complex JSON

# CNF Day 2 – Config Creation

```
"config-deploy": {
    "steps": {
        "config-setup": {
            "description": "Gather necessary input for config init and status verification",
            "target": "config-setup-process",
            "activities": [
                {
                    "call_operation": "ResourceResolutionComponent.process"
                }
            ],
            "on_success": [
                "config-apply"
            ],
            "on_failure": [
                "handle_error"
            ]
        },
        "config-apply": {
            "description": "Activate K8s config template",
            "target": "k8s-config-apply",
            "activities": [
                {
                    "call_operation": "K8sConfigTemplateComponent.process"
                }
            ],
            "on_success": [
                "status-verification-script"
            ]
        },
```

- CNF instance based
- Config Instance (CFI)
  - Instantiates CFT
  - Provides overrides for CFT
- CFI creation is part of Config-Deploy in CDS
  - Creates new k8s resources
  - Modifies k8s resources of existing CNF instance
- Native mechanisms in CDS
  - Customizable by CBA
- In vFW CNF Use Case followed by simple Status Check
  - Checks Pod Status until „Running”
  - Fails after 30 retries

# Resource Assignment (CNF/PNF)

```
"steps": {
    "resource-assignment": {
        "description": "Resource Assign Workflow",
        "target": "resource-assignment",
        "activities": [
            {
                "call_operation": "ResourceResolutionComponent.process"
            }
        ],
        "on_success": [
            "profile-upload"
        ]
    },
    "profile-upload": {
        "description": "Generate and upload K8s Profile",
        "target": "k8s-profile-upload",
        "activities": [
            {
                "call_operation": "ComponentScriptExecutor.process"
            }
        ]
    }
},
```

- **Resource Assignment**:
  - First of the ways to enrich Helm package
  - Resolves overrides for Helm instantiation
  - It is supplemented by profiling
  - We use it to gather inputs and prepare for profiling
  - Result is stored in MDSAL and can be easily used during Day2 operations
- **ResourceResolutionComponent** used

# Resource Assignment (CNF/PNF)

```
"steps": {
    "resource-assignment": {
        "description": "Resource Assign Workflow",
        "target": "resource-assignment",
        "activities": [
            {
                "call_operation": "ResourceResolutionComponent.process"
            }
        ],
        "on_success": [
            "profile-upload"
        ]
    },
    "profile-upload": {
        "description": "Generate and upload K8s Profile",
        "target": "k8s-profile-upload",
        "activities": [
            {
                "call_operation": "ComponentScriptExecutor.process"
            }
        ]
    }
},
```

- **Profiling** mechanism allows to also parametrize complex overrides values
  - values.yaml file is taken from the profile
  - original helm chart **is not modified**
  - There are two types of profiles
    - **static –** predefined in CBA
    - **dynamic –** generated and templated during instantiation
  - CBA may have many profils with predefined overrides.
- **K8sProfileUploadComponent** is used

# Config Deploy: PNF Registration

```
"pnf-registration": {
    "description": "Register UERANSIM as a PNF",
    "target": "pnf-registration-request",
    "activities": [
        {
            "call_operation": "ComponentScriptExecutor.process"
        }
    ],
    "on_success": [
        "status-verification-script"
    ],
    "on_failure": [
        "handle_error"
    ]
},
```

- Service model is composed of PNF and CNF
- PNF is simulated by UERANSIM solution
- In order to register PNF in ONAP **PNF Plug and Play** procedure is used
- This step sends PNF registration event to PRH component of DCAE
- CNF Core instantiation waits until PNF Registration finishes

# Config Deploy: Status Verification

```
"status-verification-script": {
    "description": "Simple status verification script",
    "target": "simple-status-check",
    "activities": [
        {
            "call_operation": "ComponentScriptExecutor.process"
        }
    ],
    "on_success": [
        "pnf-reconfiguration"
    ],
    "on_failure": [
        "handle_error"
    ]
},
```

- Procedure verifies if CNF is up and running
- All k8s resources created must have „Running” state to continue
- Script calls k8sPlugin Status API
- Instance status verification checks value of **ready** flag:
  - **False** means deployment in progress
  - **True** means deployment is finished
- **ComponentScriptExecutor** operation used

# Config Deploy: PNF Reconfiguration

```
"pnf-reconfiguration": {
    "description": "Reconfigure UERANSIM - call ue-reconfiguration workflow",
    "target": "ran-reconfiguration-request",
    "activities": [
        {
            "call_operation": "ComponentScriptExecutor.process"
        }
    ],
    "on_success": [
        "collect-results"
    ],
    "on_failure": [
        "handle_error"
    ]
},
```

- Aim at configuration of PNF base on the configuration resolved from the CNF
- Request sent towards UERANSIM component contains parameters required during subscription, eg:
  - PLMN ID
  - UE ID
- **ComponentScriptExecutor** operation used

# Resource Reconfiguration: Config Apply

```
"config-apply": {
    "description": "Activate UE reconfiguration template",
    "target": "k8s-config-apply",
    "activities": [
        {
            "call_operation": "K8sConfigTemplateComponent.process"
        }
    ],
    "on_failure": [
        "handle_error"
    ]
}
```

- K8sPlugin instantiates the configuration uploaded during config-upload step
- As a result:
  - new gnb pod is created with modified parameters
  - The old instance is deleted
- **K8sConfigValuesComponent** component is utilized

# ONAP Specific Workflows

## VNF/VF-Module - Instantiation & Post Instantiation

https://wiki.onap.org/display/DW/SO+Building+blocks
https://wiki.onap.org/display/DW/User+Guide#Designtime--1690278344
https://wiki.onap.org/pages/viewpage.action?pageId=36966186
https://wiki.onap.org/pages/viewpage.action?pageId=64006314#E2ERunTime-2095048582

The following workflows are contracts established between SO, SDNC and CDS to cover the **instantiation** and the **post-instantiation** use cases.

User -> SO (Macro Service Create)
SO -> **Assign**BB (service, vnf, vf-module)
->CDS (resource-assignment workflow)
SO -> **ConfigAssign**BB - day0 config assign
-> CDS (config-assign workflow)
**Assignment** for Instantiation & Post Instantiation configuration

SO -> CreateBB (VF-Module)
-> OpenStack adapter / Multi-Cloud
**Instantiation**

SO -> **ConfigDeploy**BB - day0 config push
-> CDS (config-deploy workflow)
Post Instantiation **Configuration**

The **third step** is to perform assignment. Assignment will be performed per the orchestration plan - and will start from the service-level, then iterating through the various resources contained within the service. Assignment can involve different systems.

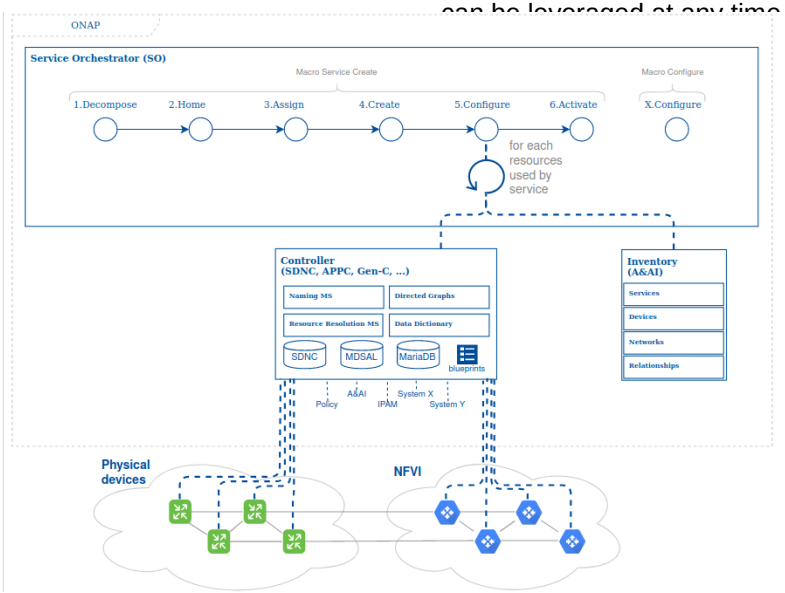
For example: for a '*management_ip*' property on a specific VNF component, representing the management interface address, the system may have to reach out to an IPAM system, pulling information from a specific subnet (either rules-based, leveraging a database such as the controller data store, or provided through input).

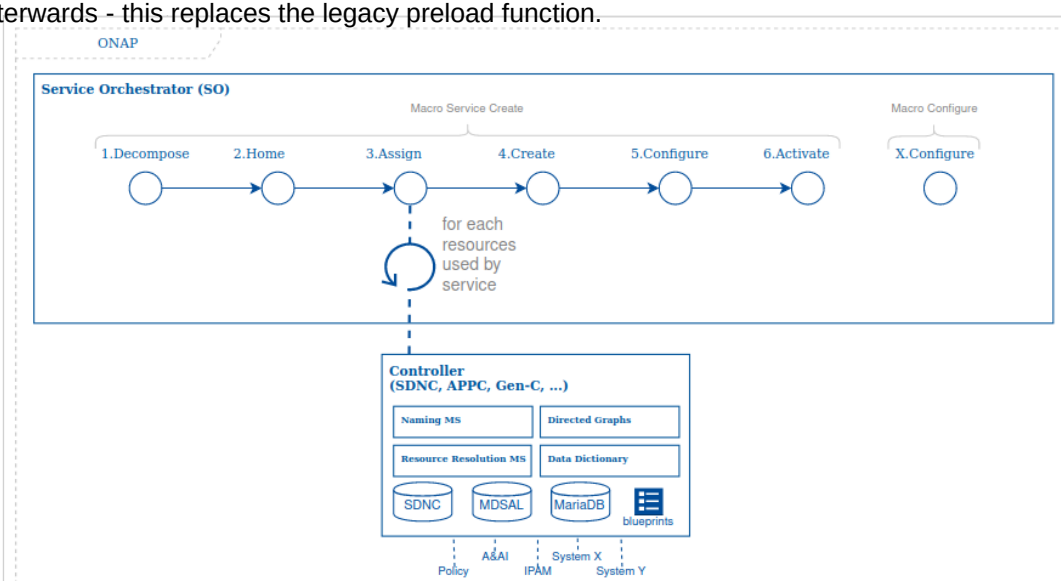
The ONAP Controller Design Studio (CDS) initiative implements an exhaustive framework to tackle this (through data dictionary, controller blueprints or other means as it evolves).

These assigned values will be stored in the service context, inside the controller's data store (MDSAL). These can be leveraged at any time afterwards - this replaces the legacy preload function.

The **fifth function** is the configuration of the device - essentially applying service configuration or application type configuration to the device so it can become operational.

This is done involving the right controllers, and again leverages the service & resource context stored in the controller data store, the directed graphs and/or Controller Design Studio blueprints artifacts (which can include DGs, code, etc.). It will then transformed all the assigned values into configuration payload for the device, using the right protocol (Netconf/Restconf or just Rest APIs), and when triggered through CDS Blueprints will use Velocity templating for transformation/mapping. This applies to PNFs or VNFs - it is purely network device configuration. If any aspect of the configuration needs to be represented in the inventory, it will perform such updates.



# AAI model: k8s resource object

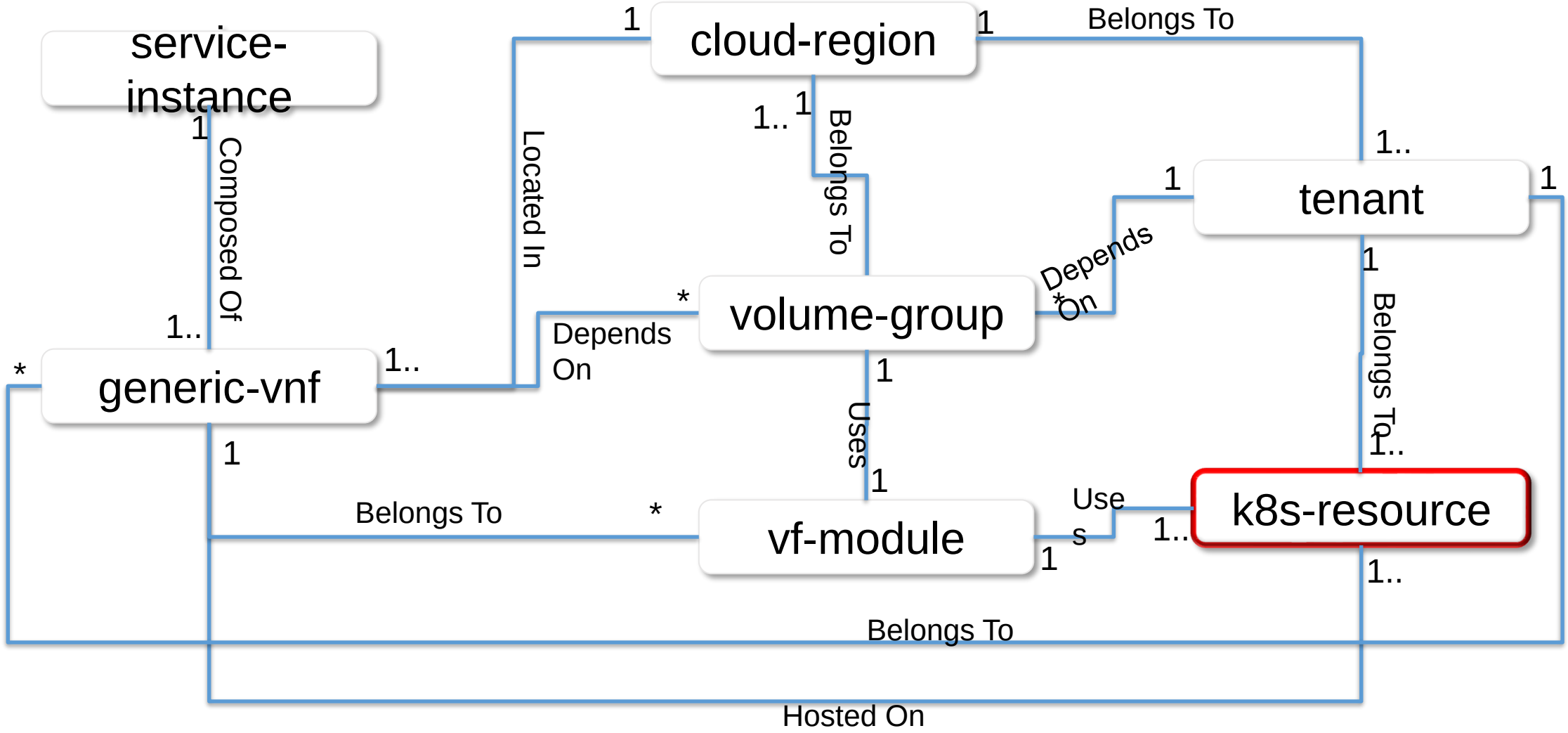
| Attribute | Type | Mandatory |
|---|---|---|
| id | UUID | Yes (PK) |
| name | String | Yes |
| group | String | Yes |
| version | String | Yes |
| kind | String | Yes |
| labels | List of strings | No |
| namespace | String | Yes |
| selflink | URI | Yes |

K8s resource is basic AAI entity to model resources created in K8s cluster.

It plays similar role as vserver resource for standard VNFs.

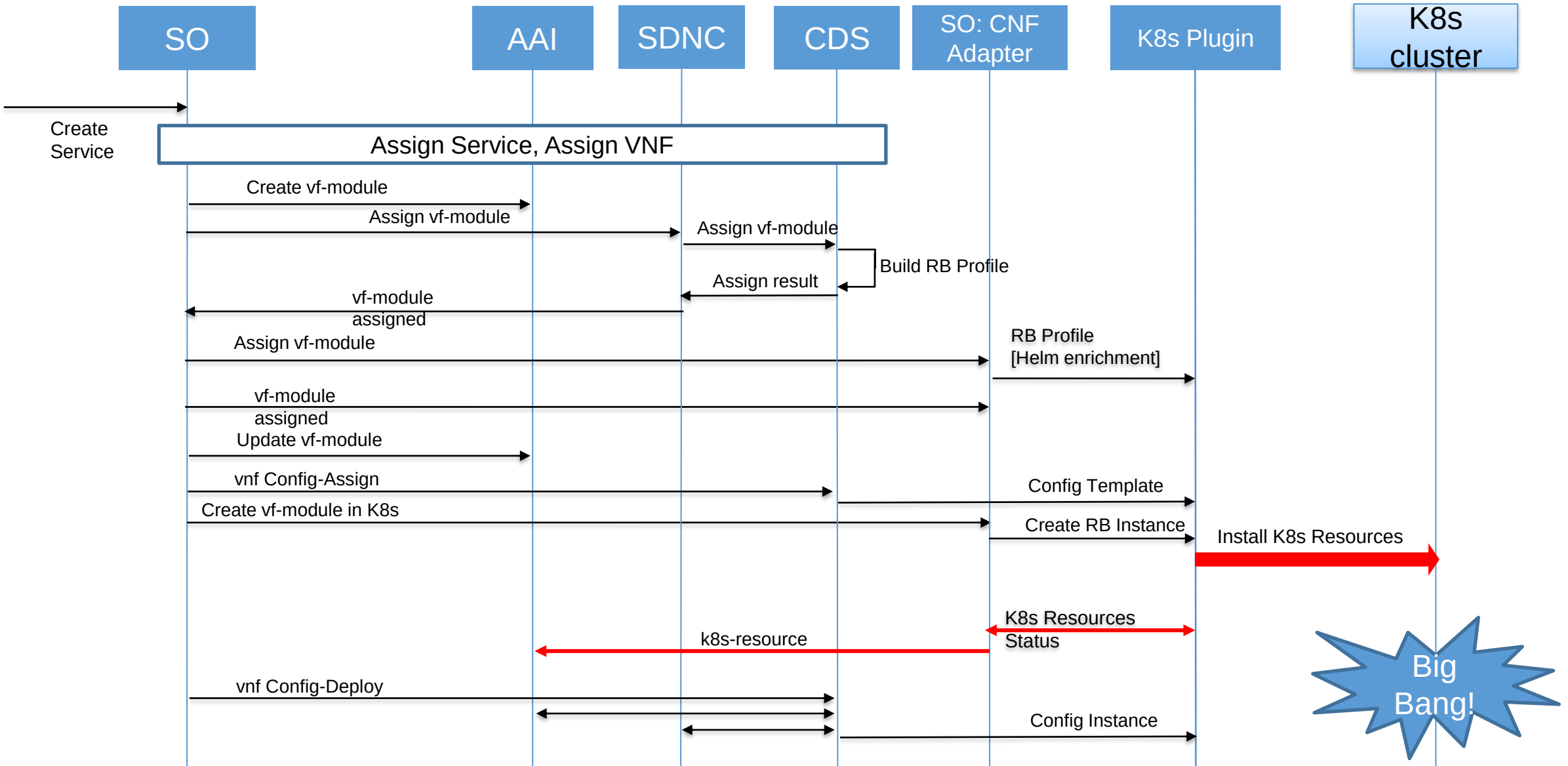
Self-link allows to access full and actual details of the k8s resource

# AAI model: relations

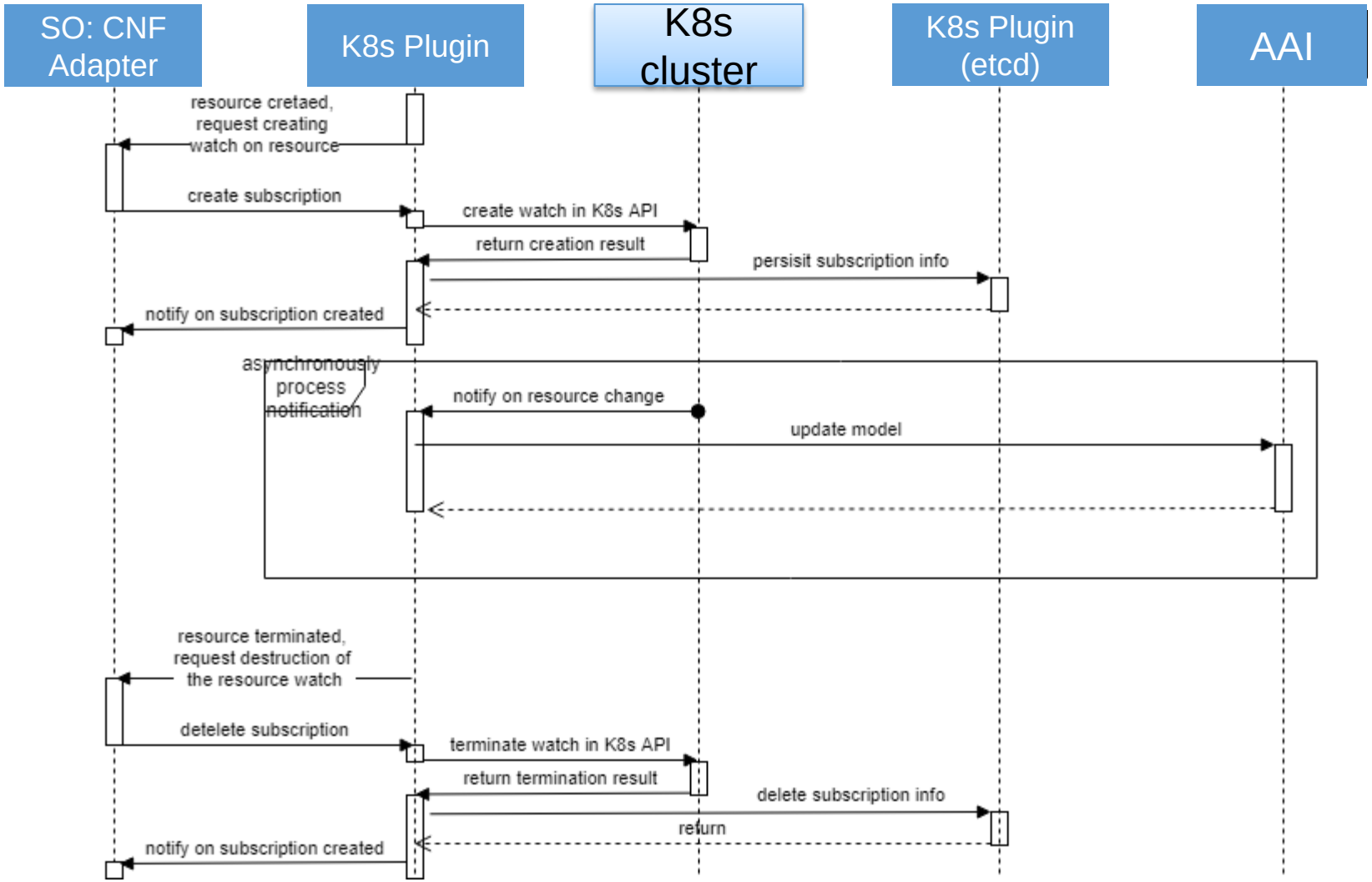



https://wiki.onap.org/display/DW/Simplified+K8S+Resource+Model+-+IM

# Instantiation of the Helm Chart – Istanbul



# CNF AAI Update - Jakarta




1. CNF Adapter creates status notification subscription
2. K8s Notifies on Resource's change
3. K8sPlugin Sends Subscription Notification
4. CNF Adapter Determines type of change
   - Create new k8s-resource
   - Deletes k8s-resource
   - Update K8s resource version

Thank You