

CN2 Pipeline for GitOps Guide

Published
2023-09-28

Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, California 94089
USA
408-745-2000
www.juniper.net

Juniper Networks, the Juniper Networks logo, Juniper, and Junos are registered trademarks of Juniper Networks, Inc. in the United States and other countries. All other trademarks, service marks, registered marks, or registered service marks are the property of their respective owners.

Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.

CN2 Pipeline for GitOps Guide

Copyright © 2023 Juniper Networks, Inc. All rights reserved.

The information in this document is current as of the date on the title page.

YEAR 2000 NOTICE

Juniper Networks hardware and software products are Year 2000 compliant. Junos OS has no known time-related limitations through the year 2038. However, the NTP application is known to have some difficulty in the year 2036.

END USER LICENSE AGREEMENT

The Juniper Networks product that is the subject of this technical documentation consists of (or is intended for use with) Juniper Networks software. Use of such software is subject to the terms and conditions of the End User License Agreement ("EULA") posted at <https://support.juniper.net/support/eula/>. By downloading, installing or using such software, you agree to the terms and conditions of that EULA.

Table of Contents

1

Install and Manage

CN2 Pipeline | 2

- CN2 Pipeline Overview | 2
- CN2 Configuration | 3
- CN2 and GitOps | 4
- Pipeline Installation and Setup | 7
- Prerequisites | 8
- Download CN2 Pipeline | 9
- Create Service Account and Token in Kubernetes | 9
- Install Helm | 17
- Install CN2 Pipeline Helm Chart | 17
- Verify CN2 Pipeline Helm Chart Installation | 20
- Argo CD and Helm Configuration | 21
- Argo Log In | 22
- CN2 and Workflows | 23
- CN2 Pipeline Service | 24
- CN2 Pipeline Configurations | 24
- Create Custom Workflows for CN2 Pipeline | 26

Explanation of Parameters for values.yaml | 30

2

Architecture and Design

CN2 Pipeline Solution Test Architecture and Design | 36

- Overview | 36
- Use Case | 36

Workflows | 37

Profile | 38

Test Environment Configuration | 45

Logging and Reporting | 54

Test Cases

CN2 Pipeline Test Management | 56

Trigger the CN2 Pipeline Test | 56

Change Commit Threshold Trigger | 56

Change Test Profiles | 57

Identify CN2 Pipeline Test Trigger | 58

Access Test Results | 58

Uninstall the CN2 Pipeline | 59

Test Cases for No vMX Setup per Profile | 59

Test Cases for a Standard Setup per Profile | 64

CN2 Pipeline Test Case Descriptions | 70

Architect Onboard | 70

Architect Execute | 71

Architect Teardown | 73

SRE Onboard | 73

SRE Execute | 73

SRE teardown | 78

1

CHAPTER

Install and Manage

[CN2 Pipeline](#) | 2

[Explanation of Parameters for values.yaml](#) | 30

CN2 Pipeline

SUMMARY

Juniper Cloud-Native Contrail® Networking™ (CN2) pipeline supports GitOps-based network configuration management. Use this document to review the pipeline GitOps configuration and install the pipeline in environments using CN2 Release 23.1 or later.

IN THIS SECTION

- [CN2 Pipeline Overview | 2](#)
- [CN2 Configuration | 3](#)
- [CN2 and GitOps | 4](#)
- [Pipeline Installation and Setup | 7](#)
- [Prerequisites | 8](#)
- [Download CN2 Pipeline | 9](#)
- [Create Service Account and Token in Kubernetes | 9](#)
- [Install Helm | 17](#)
- [Install CN2 Pipeline Helm Chart | 17](#)
- [Verify CN2 Pipeline Helm Chart Installation | 20](#)
- [Argo CD and Helm Configuration | 21](#)
- [Argo Log In | 22](#)
- [CN2 and Workflows | 23](#)
- [CN2 Pipeline Service | 24](#)
- [CN2 Pipeline Configurations | 24](#)
- [Create Custom Workflows for CN2 Pipeline | 26](#)

CN2 Pipeline Overview

GitOps is a deployment methodology centralized around a Git repository, where the GitOps workflow pushes a configuration through testing, staging, and production. Many customers run a staging environment or staging cluster. Supporting GitOps for CN2 allows automatic configuration deployment and testing of CN2 network configurations using parameters in test case YAML files.

After configuring the CN2 pipeline and the GitOps tool, CN2:

- Syncs with the GitOps repository and auto-provisions CN2 configurations across multiple Kubernetes clusters.
- Provisions with the capability to test and verify the deployed CN2 configurations in each Kubernetes cluster.
- Provides auto-revision monitoring and updates.

CN2 Configuration

IN THIS SECTION

- [GitOps | 3](#)

In CN2, the configurations are Kubernetes Custom Resource Definitions (CRDs), written in YAML or JSON format. These CRDs are stored and managed in the Git repository, which makes the Git repository the source of truth for all of the network configurations.

See [CN2 Sample Configurations](#).

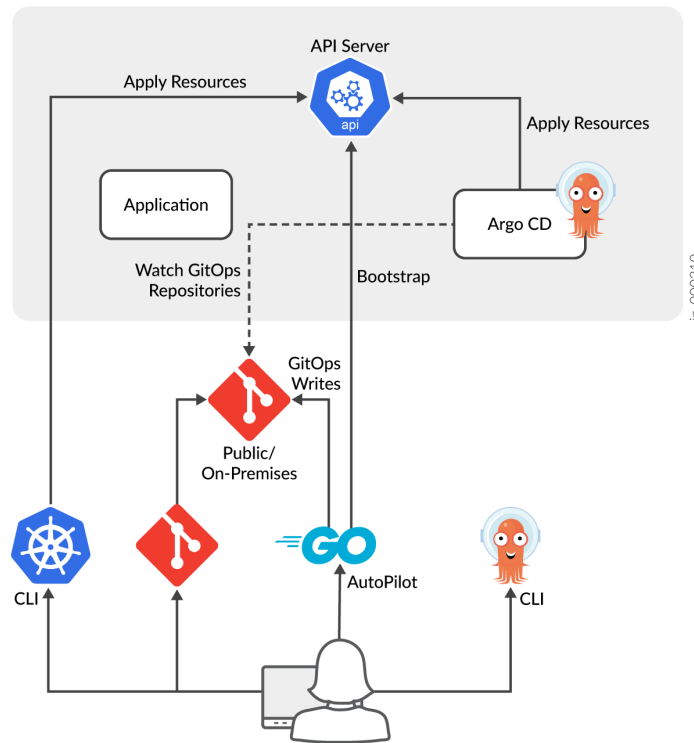
GitOps

"GitOps is a paradigm or a set of practices that empowers developers to perform tasks which typically fall under the purview of IT operations. GitOps requires us to describe and observe systems with declarative specifications that eventually form the basis of continuous everything." Quote is from *CloudBees*.

In order to achieve the GitOps mode of operation, the Argo CD application is used. The CN2 Git repository is configured to be used as the Argo CD application.

The GitOps engine watches for changes in the Git repository and applies the changes to Kubernetes whenever there are any changes made to the system. The GitOps engine also runs a repository server that caches all the applications files from the Git repository for verification against the Git repository to monitor for any changes being made in the Git repository.

Figure 1: Argo CD with Git Repository and Kubernetes



CN2 and GitOps

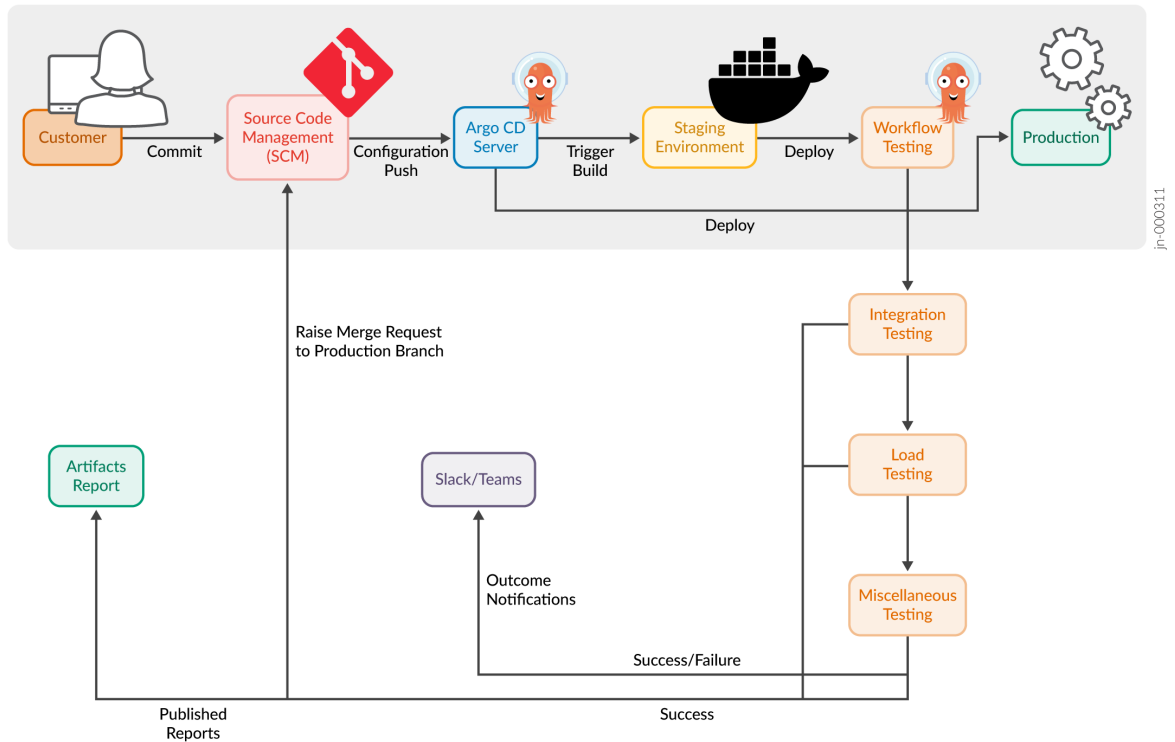
IN THIS SECTION

- [CN2 Pipeline Configuration Flow | 5](#)
- [GitOps Server | 6](#)
- [Workflow and Tests | 6](#)

The primary benefit of supporting GitOps for CN2 is to achieve automatic configuration deployment and testing of CN2 network configurations. CN2 configurations are custom resource definitions (CRDs) which are maintained in a Git repository. These CRDs are applied to the CN2 cluster whenever there is a

change to the CN2 configurations present in the Git repository. In order to test and apply these changes, the GitOps applications Argo CD and Argo Workflows are deployed.

Figure 2: GitOps Pipeline Workflow

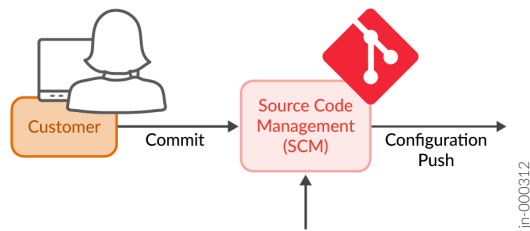


CN2 Pipeline Configuration Flow

Your CN2 configurations are maintained in the CN2 Git repository. The CN2 Git repository is configured to be used as the Argo CD application. The CN2 pipeline configuration flow is:

1. CN2 configurations are initially pushed to the staging repository by you (the administrator).
2. Any changes to the configurations present in the repository triggers a Git synchronization to the GitOps server.
3. The GitOps server looks for any changes required by comparing the current configuration and the new configuration fetched from the CN2 Git repository.
4. If any new changes are pushed, the GitOps server applies these new changes to the CN2 environment.

Figure 3: CN2 Configurations in Customer Git Repository



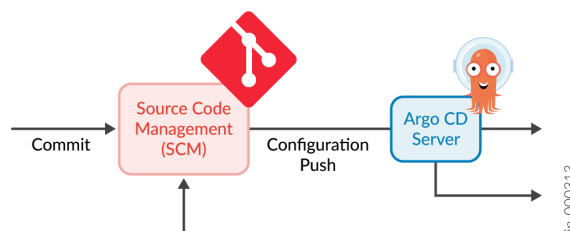
GitOps Server

The GitOps server confirms the configuration in the CN2 environment is always synchronized with what is present in the Git repository. CN2 pipeline supports two branches:

- One for the staging environment.
- One for the production environment.

The staging branch is the branch where you push any configurations required to be pushed to the staging CN2 cluster. These configurations are then tested by the workflow engine before the configurations are merged to the production branch.

Figure 4: GitOps Server



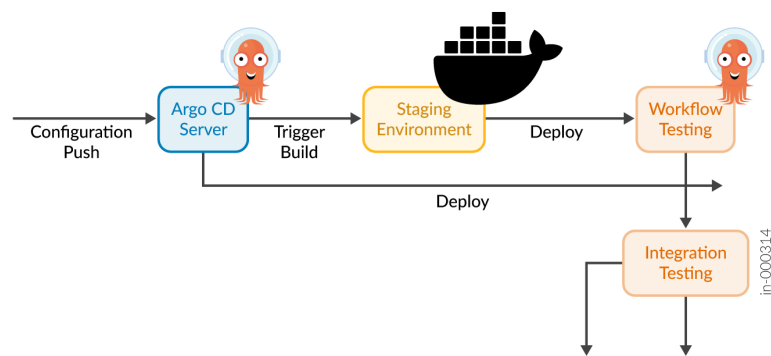
Workflow and Tests

The GitOps server pushes all of the changes to the CN2 setup.

1. This push triggers the workflow cycle to run test cases to validate and verify the CN2 setup against the configurations that were applied in the staging setup.

2. With successful performance of the test cases, you are notified about the test completion and a merge request is presented to the production branch.
3. At this point, you need to validate the changes in the merge request and approve that the changes be merged to the production branch.
4. After the new configurations are merged to the production branch, the GitOps server synchronizes the configurations and applies the configurations to the CN2 production cluster.

Figure 5: Workflow and Tests



Pipeline Installation and Setup

IN THIS SECTION

- [Components | 7](#)

Components

The pipeline components are:

- Git Server
- Argo CD
- Argo Workflow

- Argo Events
- CN2 Pipeline Service
- CN2 Solution Test Suite

CN2 Components

All CN2 pipeline components are installed and configured as part of the CN2 Pipeline Helm chart installation. Argo CD is installed as part of this initial system setup in an external cluster (cluster where CN2 is not running). Argo CD is configured with the following details during the initial setup:

- CN2 cluster environment details
- Git repository access details
- CN2 GitOps engine application configuration

See ["Install CN2 Pipeline Helm Chart" on page 17](#).

Kubernetes

You can use any native Kubernetes or Red Hat OpenShift with CN2 or another Container Network Interface (CNI) to provision the CN2 pipeline.

Prerequisites

Prerequisites for installing the CN2 pipeline are:

- Management Kubernetes cluster
- CN2 Kubernetes cluster
- Connectivity from the management Kubernetes cluster to the CN2 cluster
- GitLab repository CN2 configuration folder with sample ConfigMap file
- Connectivity from the management Kubernetes cluster to outside to access Argo CD, Argo workflow, and test results

NOTE: Install ingress from the `files/ingress/openshift/public` on the OpenShift cluster.
Contrail pipeline needs GitLab or GitLab Open Source as event source.

Download CN2 Pipeline

To download the CN2 Pipeline tar file:

1. Download CN2 Pipelines Deployer files from [Juniper Networks Downloads](#).
2. Untar the downloaded files to the management server.

Create Service Account and Token in Kubernetes

IN THIS SECTION

- [For Kubernetes Version 1.23 or Earlier | 9](#)
- [For Kubernetes Version 1.24 or Later | 13](#)
- [Apply Ingress and Update /etc/hosts for OpenShift Deployment Type | 16](#)

Creating a service account and token are important to give the Kubernetes cluster access within and outside of the CN2 pipeline using APIs. This topic describes how to create the service account, token, role, and role bindings.

NOTE: Throughout these procedures, cn2pipelines is used as an example.

For Kubernetes Version 1.23 or Earlier

Perform these steps on the CN2 cluster.

To create a service account and token:

1. Create the namespace if one does not already exist.

```
kubectl create ns cn2pipelines
```

2. Create a service account named cn2pipelines.

```
kubectl create sa cn2pipelines -n cn2pipelines
```

3. Run the describe command to fetch the token.

```
kubectl describe sa cn2pipelines -n cn2pipelines
```

Output:

```
Name: cn2pipelines
Namespace: cn2pipelines
Labels:          <none>
Annotations:     <none>
Image pull secrets: <none>
Mountable secrets: cn2pipelines-token-5szb6
Tokens:          cn2pipelines-token-5szb6
Events:          <none>
```

By default, Kubernetes version 1.23 or earlier creates the token as a secret by default when you create the service account.

4. For CN2 with Kubernetes, retrieve the Mountable secrets from the Step 3 output. Run the describe secret command to get the bearerToken for the service account. The bearerToken is needed when you update the values.yaml file.

```
kubectl describe secret cn2pipelines-token-5szb6 -n cn2pipelines
```

Output:

```
Name:          cn2pipelines-token-5szb6
Namespace:     cn2pipelines
Labels:        <none>
Annotations:   kubernetes.io/service-account.name: cn2pipelines
               kubernetes.io/service-account.uid: e5059023-7269-482a-870c-5c4ff175ba00
Type:          kubernetes.io/service-account-token
Data
====
```

```
namespace: 10 bytes
token:
eyJhbGciOiJSUzI1NiIsImtpZCI6InZQMkx0cWl0Qjg5MElySutiWGpPTWJVVGZNR3FQS3hnUDhyTDFHZjd3VFkifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJjaS1qZW5raW5zIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6ImNpLWplbmtbnMtdG9rZW4tNXN6YjYiLCJrdWJlcm5ldGVzLm1vL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC5uYW1lIjoiy2kta
mVua2lucyIsImt1YmVybmV0ZXMuaW8vc2VydmljZWJjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6ImU1MDU5MDIzLTc
yNjktNDgyYS04NzBjLTVjNGZmMTc1YmEwMCIsInN1YiI6InN5c3RlbnR1bTzZXJ2aWNlYWNjb3VudDpjaS1qZW5raW5zOmnPl
WplbmtbnMifQ.DeAySlkf7dW6xzUH5bLeXc2lRPa_RMZ2bG4zGktPhyA2eDdM-nliCTpwhuBPBZ2fNeiaZb3Tl8h-
MJNF7IygwXEHjW8ALfvUv4nBnmSMj9JW44PoPeMSCAnrtIXucy8hcGZN4K6i1w2n6ASSYAXyifwMOLy3-
KfbY9PYEr0b0eC34-cHKP-
TQoV0o4ncA58kw0wut2DmkIKfH3gs0AY445w04_WUeYuq0_JU0uQpyPaCR09sLDhM1Vcnp0TI7hvZu_DbVyRhy4b8QqJEj
3h08j0lPGvFhvmCcUqTSLXbVtV9o62cqhd1q9pcFq5yAxmYpuwWjkOP8KuIsf71U070_w
ca.crt: 1099 bytes
```

5. For CN2 with Red Hat OpenShift, you use the image to pull secrets as a secret value and then run the following command to get the bearerToken.

- a. Get the bearerToken.

```
kubectl get secret cn2pipelines-dockercfg-445hx -n cn2pipelines -o yaml
```

Output:

```
apiVersion: v1
data:
  .dockercfg: e30=
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: cn2pipelines
    kubernetes.io/service-account.uid: 38b98d44-334e-4fce-ba90-afe6eae1f644
    openshift.io/token-secret.name: cn2pipelines-token-n5qwb
    openshift.io/token-secret.value:
eyJhbGciOiJSUzI1NiIsImtpZCI6ImZ5aVpVpQRURJUZU1YThqV3ZUME43UGxiX1JhR0hoYnhZd25GMkpBX2g3UzAifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJjaS1qZW5raW5zIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6ImNpLWplbmtbnMtdG9rZW4tbnR1bTzZXJ2aWNlYWNjb3VudC5uYW1lIjoiy2kta
mVua2lucyIsImt1YmVybmV0ZXMuaW8vc2VydmljZWJjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6ImU1MDU5MDIzLTc
yNjktNDgyYS04NzBjLTVjNGZmMTc1YmEwMCIsInN1YiI6InN5c3RlbnR1bTzZXJ2aWNlYWNjb3VudDpjaS1qZW5raW5zOmnPl
WplbmtbnMifQ.Zj5Bs8Y8h4GL0o-p7rhnJcPeYdbcoVpfM0oRMHky3KUQuTab5ZjwV3o-
```

```

h0e-
hZlQC_TpI4kNotijEoFiwKU_mYPr9bY36EBngUZp41BiqSwiY_5qG_wYDd6Dg_Xh6C5n4eagBN80Ai9IXlM3SYH9hgG
mEx-
dQoXh1GdxCht_JPWODXbKq0eFU_mtUlqjMU0p__g1VoQ1svlRCUvRsfI80xIM5jd7qPC3NqkpHlK1I5BHQaScWdjiha
To70pK-zkcgSj8820kw4UxsttFgJZ5iF7hHLCMwtVi-
pX4SVl2pdHi8H7DxD4YDiZD3xUJapRpRvnHqNsDvoXXonWB0skW4JE86t95Z5Z7lIHNPpvftajxc3qky6hBW0-1yfpG
K36Df2g30GGrVm16S31w11K6y7oUu6Py5B4BM5qcge7J9wNTNRMezuomt38SyqyuaCt8SBL-
dtc_8bAhKLnMZ7Vr_kGHCD5GBe0_7BaH9dqhb85-
oyC_mKHA8F_5xmC4wZ7bBDhMRN9lAoKePK6p1toz1Ca395_w83ib5zGxMfD9C-
hskYNrkHCPJwS00_s_QXQdXnnzhc0_C2K9KqZk1qE8A2zm1baxtcP_PrMMhS5H0bs2i8n88kZ074H7AmPk8HRx0oLG5
Ue80h8F5x9Ua5M4WuZSfmN5jXlSVdCvtqQY
  creationTimestamp: "2023-03-17T17:32:11Z"
  name: cn2pipelines-dockercfg-445hx
  namespace: cn2pipelines
  ownerReferences:
  - apiVersion: v1
    blockOwnerDeletion: false
    controller: true
    kind: Secret
    name: cn2pipelines-token-n5qwb
    uid: a02b6e19-db50-4d27-9b29-b33a60ad47c9
  resourceVersion: "1132721"
  uid: 723f8808-2fd5-49e6-8012-9fbab8962b47
type: kubernetes.io/dockercfg

```

6. Create a ClusterRole and ClusterRoleBinding to give the service account appropriate permissions.

a. Create a ClusterRole and name the file clusterrole-cn2pipelines.yaml.

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    name: cn2pipelines
  name: cn2pipelines
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'

```



```
- nonResourceURLs:
  - '*'
verbs:
  - '*'
```

- b.** Apply the `clusterrole-cn2pipelines.yaml` file that you just created.

```
kubectl apply -f clusterrole-cn2pipelines.yaml -n cn2pipelines
```

- c.** Create a `ClusterRoleBinding` file and name the file `clusterrolebinding-cn2pipelines.yaml`.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    name: cn2pipelines
  name: cn2pipelines
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cn2pipelines
subjects:
- kind: ServiceAccount
  name: cn2pipelines
  namespace: cn2pipelines
```

- d.** Apply the `clusterrolebinding-cn2pipelines.yaml` file that you just created.

```
kubectl apply -f clusterrolebinding-cn2pipelines.yaml -n cn2pipelines
```

You now have full permissions to all resources in cluster.

For Kubernetes Version 1.24 or Later

For Kubernetes version 1.24 and later, creating a service account does not create a secret automatically. Use the following procedure to manually create a token for that service account.

1. Create the namespace if one does not already exist.

```
kubectl create ns cn2pipelines
```

2. Create a service account named cn2pipelines.

```
kubectl create sa cn2pipelines -n cn2pipelines
```

3. Create a token for the service account cn2pipelines.

```
kubectl create token cn2pipelines -n cn2pipelines --duration=999999h
```

4. Create a ClusterRole and ClusterRoleBinding to give the service account appropriate permissions.

- a. Create a ClusterRole and name the file clusterrole-cn2pipelines.yaml.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    name: cn2pipelines
  name: cn2pipelines
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
- nonResourceURLs:
  - '*'
  verbs:
  - '*'
```

- b. Apply the clusterrole-cn2pipelines.yaml file that you just created.

```
kubectl apply -f clusterrole-cn2pipelines.yaml -n cn2pipelines
```

- c. Create a ClusterRoleBinding file and name the file clusterrolebinding-cn2pipelines.yaml.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    name: cn2pipelines
  name: cn2pipelines
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cn2pipelines
subjects:
- kind: ServiceAccount
  name: cn2pipelines
  namespace: cn2pipelines
```

- d. Apply the clusterrolebinding-cn2pipelines.yaml file that you just created.

```
kubectl apply -f clusterrolebinding-cn2pipelines.yaml -n cn2pipelines
```

Your token is created and you now have full permissions to all resources in cluster.

5. Run the describe command to fetch the token.

```
kubectl describe sa cn2pipelines -n cn2pipelines
```

Output:

```
Name: cn2pipelines
Namespace: cn2pipelines
Labels:          <none>
Annotations:     <none>
Image pull secrets: <none>
Mountable secrets: cn2pipelines-token-5szb6
Tokens:          cn2pipelines-token-5szb6
Events:          <none>
```

6. Retrieve the Mountable secrets from the Step 5 output. Run the describe secret command to get the bearerToken for the service account.

```
kubectl describe secret cn2pipelines-token-5szb6 -n cn2pipelines
```

Output:

```
Name:          cn2pipelines-token-5szb6
Namespace:     cn2pipelines
Labels:        <none>
Annotations:   kubernetes.io/service-account.name: cn2pipelines
               kubernetes.io/service-account.uid: e5059023-7269-482a-870c-5c4ff175ba00
Type:          kubernetes.io/service-account-token
Data
====
namespace:    10 bytes
token:
eyJhbGciOiJSUzI1NiIsImtpZCI6InZQMkx0cWl0Qjg5MElySutiWGpPTWJVVGVZNR3FQS3hnUDhyTDFHZjd3VFkifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJjaS1qZW5raW5zIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6ImNpLWplbmtpbmMtdG9rZW4tNXN6YjYiLCJrdWJlcm5ldGVzLm1vL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC5uYW1lIjoieY2ktaWVua2lucyIsImt1YmVybmV0ZXMuaW8vc2Vydm1jZWFiY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6ImU1MDU5MDIzLTcyNjktNDgyYS04NzBjLTVjNGZmMTc1YmEwMCIsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDpjaS1qZW5raW5zOmNpLWplbmtpbmMifQ.DeAySlkf7dW6xzUH5bLeXc2lRPa_RMZ2bG4zGktpHyA2eDdM-nliCTpwhuBPbZ2fNeiaZb3Tl8h-MJNF7IygwXEHjW8ALfvUv4nBnmSMj9JW44PoPeMSCAnrtIXucy8hcGZN4K6i1w2n6ASSYAXyifwMOLy3-KfbY9PYErOb0eC34-cHKP-TQoV0o4ncA58kwOwut2DmkIKfH3gs0AY445wO4_WUeYuqQ_JU0uQpyPaCR09sLDhM1Vcnp0TI7hvZu_DbVyRhy4b8QqJEj3h08j0lPGvFhvmCcUqTSLXbVtV9o62cqhd1q9pcFq5yAxmYpuwWjkOP8KuIsf71U070_w
ca.crt:       1099 bytes
```

Apply Ingress and Update /etc/hosts for OpenShift Deployment Type

Perform these steps on the CN2 cluster.

1. Check that /etc/hosts contains entries from the OpenShift cluster. For example:

```
192.167.19.571 api.ocp-ss-571.net
```

2. Run the following command to locate the files to apply.

```
cd files/public/openshift
```

3. Apply the following YAML files.

```
kubectl apply -f clusterRoleBindings.yml  
kubectl apply -f haproxy-ingress-controller.yaml
```

4. Wait for all the pods to come up, then apply these two YAMLs.

```
kubectl apply -f contour.yaml  
kubectl apply -f nginx.yaml
```

The /etc/hosts for OpenShift is updated.

Install Helm

Before installing the CN2 pipeline chart, you need to install Helm 3 in the management cluster. Helm helps you manage Kubernetes applications. Helm Charts help you define, install, and upgrade even the most complex Kubernetes application.

Run the following command to download and install the latest version of Helm 3.

```
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3  
chmod 700 get_helm.sh  
./get_helm.sh
```

Install CN2 Pipeline Helm Chart

The CN2 pipeline Helm chart is used to install and configure the CN2 pipeline management cluster.

To install the CN2 pipeline Helm chart on your management cluster:

1. In your downloaded CN2 Pipelines Deployer files, locate the `values.yaml` in folder **contrail-pipelines-x.x.x/values**.
2. Input the chart values. For `values.yaml` parameters descriptions, see ["Explanation of Parameters for values.yaml" on page 30](#).

Example `values.yaml` for the management cluster:

```
#####
#           Common Configuration (global vars)           #
#####
global:
  docker_image_repo: docker.io # Global docker registry for non Juniper images
  registry: enterprise-hub.juniper.net/contrail-container-prod/ # Global image registry to
pull Juniper artifacts
  registry_authentication: true # true or false
  imagePullSecret: <base64 imagePullSecret> # Image pull secret for authenticated registry
  deployment_type: 'k8s' # deployment_type: k8s for CN2 kubernetes cluster (or)
deployment_type: "openshift" for CN2 openshift cluster
  nginx_nodeport: 30552 # Nodeport for reports, leave as default

  gitServer:
    access_token: <base64 access_token> # eg: eTE1Y0p1Mlo4TGhiWFpfLTFSVEg= (git access token
should be in base64 format)
    gitlabBaseURL: <gitlabBaseURL> # eg: https://cnf-gitlab.net
    project: <project> # eg: devops/cn2/cn2-pipelines
    folderName: <folderName> # eg: cn2networkconfig
    branch: <branch> # eg: master
    deploy_username: <deploy_username> # eg: gitlab+deploy-token-154 (git repository deploy
token username)
    deploy_token: <deploy_token> # eg: syDXrL-tNjiYo9WUwbNy (git repository deploy token )

  cn2ClusterDetails:
    name: <cluster name> # CN2 cluster name
    server: <kubeAPI IP> # CN2 cluster kubeapi server (should be reachable from management
server)
    bearerToken: "<bearerToken>" # CN2 cluster Service Account's Bearer Token
    kubeconfig: cn2-cluster-kubeconfig # CN2 kubeconfig name, leave as default
    mountpath: /opt/cn2_workflows # CN2 test profile folder location

  argo-events:
    managementServer: <managementServer> # CN2 pipeline management server IP
```

```

# Leave below section as default
pipeline_service:
  image: pipeline_service
  tag: "v6"
  imagePullPolicy: Always
pipeline_service_prebuild:
  image: pipeline_service_postinstall
  tag: "v6"
  imagePullPolicy: Always

workflow-objects:
  image: cn2-ptk
  tag: "23.1.0.282"
  imagePullPolicy: IfNotPresent
  topology: test_config_k8s.yaml # Test config file name - values: test_config_k8s.yaml (or)
test_config_ocp.yaml
  ssl_enabled: True # True if CN2 cluster deployed with SSL enabled else it is False
  ## ''' Enable below OCP keys only for deployment_type is openshift ''' ##
  #ocp_api_host_ip: <ocp_api_host_ip> # eg: '192.167.19.571'
  #ocp_api_host_name: <ocp_api_host_name> # eg: 'api.ocp-ss-571.net'
  argocd_retry_interval: "120" # retry interval check in seconds ; Both the parameters needs
to be set based on avg config apply time
  argocd_check_retry: "5" # number of retries before timeout

# Leave below section as default
argo-cd:
  argocdExpose:
    enabled: true
    nodePort: 30551
  replicas: 1

argo-workflows:
  argowfExpose:
    enabled: true
    nodePort: 30550
  replicas: 1

```

3. Run the following command to install the CN2 pipeline Helm Chart with the release name cn2-pipeline:

```
$ helm install cn2-pipeline .
```

Verify CN2 Pipeline Helm Chart Installation

To verify the CN2 Pipeline Helm Chart Installation, run the following commands:

1. List the Helm release in the current namespace.

```
helm ls
```

Output:

NAME	NAMESPACE	REVISION	UPDATED
STATUS	CHART	APP VERSION	
cn2-pipeline	default	1	2023-16-10 11:44:29.380155158 +0000 UTC
deployed	cn2-pipeline-1	22.2.0	

2. Display all pods in all namespaces.

```
kubectl get pods -A
```

Output:

NAMESPACE	NAME	READY	STATUS
RESTARTS	AGE		
argo-events	controller-manager-844d44-vf6r8	1/1	Running
0	2m23s		
argo-events	eventbus-default-stan-0	2/2	Running
0	2m18s		
argo-events	eventbus-default-stan-1	2/2	Running
0	2m6s		
argo-events	eventbus-default-stan-2	2/2	Running
0	2m4s		
argo-events	events-webhook-64dc49f456-p6rmw	1/1	Running
0	2m23s		
argo-events	gitlab-eventsource-qhnz8-74c4c785dc-ggmp	1/1	Running
ago)	2m17s		2 (118s
argo-events	gitlab-sensor-xc5s6-74c65564b8-m5cld	1/1	Running
ago)	2m17s		3 (115s
argo	argo-server-65566599f8-tv99s	1/1	Running
0	2m23s		
argo	workflow-controller-77c44779bf-9b42k	1/1	Running

0	2m23s			
argocd	argocd-application-controller-0	1/1	Running	
0	2m23s			
argocd	argocd-dex-server-76d5bc7dc6-r5rnw	1/1	Running	1 (2m15s
ago) 2m23s				
argocd	argocd-notifications-controller-5ff9495c68-8z581	1/1	Running	
0	2m23s			
argocd	argocd-redis-857ddfd67b-21fd2	1/1	Running	
0	2m23s			
argocd	argocd-repo-server-6dcd4856d4-hjv95	1/1	Running	
0	2m23s			
argocd	argocd-server-7cf45b4594-cntd5	1/1	Running	
0	2m23s			

Argo CD and Helm Configuration

This topic lists the Argo components and configurations that are automated as part of the CN2 pipeline Helm chart install.

- **Argo CD External Service**—Creates a Kubernetes service with service type as NodePort or LoadBalancer. This creates the Argo CD external service, providing access to the Argo CD API server and the Argo CD GUI.
- **Register Git Repository with CN2 Configurations**—Configures repository credentials and connects your Git repository to Argo CD. Argo CD is configured to your Git repository in order to watch and pull the configuration changes from your Git repository. This Git repository should only contain Kubernetes resources. Argo CD does not understand any other type of YAML or files.
- **Register Kubernetes Clusters**—Registers a Kubernetes cluster to Argo CD. This process configures Argo CD to provision the Kubernetes resources in any Kubernetes cluster. Multiple Kubernetes clusters can be configured in Argo CD.
- **Create an Argo CD Application**—Creates an application using the Argo CD GUI. Any application created in Argo CD needs to be associated with a Git repository and one Kubernetes cluster.

Argo Log In

IN THIS SECTION

- [Access Argo Workflow UI | 22](#)
- [Access Argo CD GUI | 22](#)

After installing the CN2 pipeline Helm chart, you have access to the Argo Workflow GUI and the Argo CD GUI.

Access Argo Workflow UI

To access the Argo CD GUI, you need connectivity from the management cluster to access the GUI using the NodePort service. The Argo CD GUI is accessed using the management server IP address and port 30550.

1. Access the Argo CD GUI from your browser.

```
https://<management-server-ip-address>:30550
```

2. On the management node, run the following command to receive the token.

```
kubectl -n argo exec $(kubectl get pod -n argo -l 'app=argo-server' -o  
jsonpath='{.items[0].metadata.name}') -- argo auth token
```

Access Argo CD GUI

To access the Argo CD GUI, you need connectivity from the management cluster to access the GUI using the NodePort service. The Argo CD GUI is accessed using the management server IP address and port 30551.

1. Access the Argo CD GUI from your browser.

```
https://<management-server-ip-address>:30550
```

2. On the management node, run the following command to receive the token. The username is **admin**.

```
kubect1 -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" |  
base64 -d
```

CN2 and Workflows

IN THIS SECTION

- [Why Workflows Are Needed | 23](#)
- [How Workflows Work and How CN2 Uses Workflows | 23](#)

Argo Workflows is an open source container-native workflow engine for orchestrating parallel jobs on Kubernetes. You can define workflows where each step in the workflow is a container. And you can model multi-step workflows as a sequence of tasks or capture dependencies between tasks using a directed acyclic graph (DAG).

Why Workflows Are Needed

Workflows are used to invoke and run CN2 test cases after provisioning CN2 resources by using the GitOps engine. These workflows qualify the CN2 application configurations and generates test results for the configuration that is being deployed.

How Workflows Work and How CN2 Uses Workflows

Workflows are triggered whenever a CN2 resource is provisioned by the GitOps engine. Each of the CN2 resources or a group of CN2 resources are mapped to a specific workflow test DAG. After successful completion of these test suites, the CN2 configurations are qualified to be promoted to production environments from the Staging or Test environments.

CN2 Pipeline Service

The pipeline service listens for notifications from Argo Events for any changes in Kubernetes resources. The pipeline service exposes a service which is used by Argo Events to consume and trigger the service with the data related to the CN2 configuration that you applied. It is the responsibility of the CN2 pipeline service to identify the test workflow to be triggered for the type of CN2 configuration that you applied. Workflows change dynamically depending on the objects being notified. The CN2 pipeline listener service invokes the respective workflow dependent on the CN2 configuration that gets applied.

CN2 Pipeline Configurations

IN THIS SECTION

- [Pipeline Configuration | 24](#)
- [Test Workflow Template Parameter Configuration | 25](#)
- [Workflow to Kind Map | 25](#)
- [Kubeconfig Secret for the CN2 Cluster | 26](#)

This topic shows examples for the CN2 pipeline configurations.

Pipeline Configuration

The pipeline configuration is used by the pipeline engine and includes:

- Pipeline commit threshold
- Config map: cn2pipeline-configs
- Namespace: argo-events

Example pipeline configuration:

```
apiVersion: v1
data:
  testcase_trigger_threshold: "10"
kind: ConfigMap
```

```
labels:
  app.kubernetes.io/managed-by: Helm
name: cn2pipeline-configs
namespace: argo-events
```

Test Workflow Template Parameter Configuration

All workflow template inputs are stored as configuration maps. These configuration maps are dynamically selected during the execution by the pipeline service.

Workflow to Kind Map

This mapping configuration contains the workflow template to CN2 resource kind mapping. Only one template is selected for execution and the first map that matches has the higher priority. An asterisk (*) in kind: ['*'] has higher priority than any other kind matches and overrides every mapping.

A workflow template for a CN2 resource kind mapping template includes:

- Config map: cn2tmpl-to-kind-map
- Namespace: argo-events

Following is an example configuration for the workflow template to CN2 resource kind mapping. Note the asterisk (*) in kind: ['*'] kindmap.

```
apiVersion: v1
data:
  kindmap: |
    - workflow: it-cloud-arch1-sre2
      kind: ['*']
    - workflow: custom-cnf-sample-test
      kind: ['namespace']
    - workflow: it-cloud-arch1-sre1
      kind: ['service']
    - workflow: it-cloud-arch2-sre3
      kind: ['service']
    - workflow: it-cloud-arch2-sre4
      kind: ['service']
    - workflow: it-cloud-arch2-sre5
      kind: ['namespace']
kind: ConfigMap
metadata:
```

```

labels:
  app.kubernetes.io/managed-by: Helm
name: cn2tmpl-to-kind-map
namespace: argo-events

```

Kubeconfig Secret for the CN2 Cluster

A base64 encoded kubeconfig for the CN2 cluster is created as a secret. Kubeconfig is a YAML file with all the Kubernetes cluster details, certificate, and secret token to authenticate the cluster.

- Secret: cn2-cluster-kubeconfig
- Namespace: argo-events

NOTE: Do not change the cn2-cluster-kubeconfig name.

Following is an example kubeconfig secret for CN2 cluster:

```

apiVersion: v1
kind: Secret
metadata:
  name: cn2-cluster-kubeconfig
  namespace: argo-events
data:
  config: <base64 value of your cn2cluster kubeconfig>
type: Opaque

```

Create Custom Workflows for CN2 Pipeline

You can create custom workflows tests to test your container network functions (CNFs).

In order to create a custom workflow, you can use the example custom test workflow templates provided with the CN2 pipeline files. Every workflow has a set of input parameters, volume mounts, container creation, and so on. To understand the workflow template creation see [Argo Workflows](#).

The following example custom test workflow templates are provided:

- Input parameters to workflow

- Mount volumes
- Create Kubernetes resource using workflow (Template name: create-cnf-tf and create-cnf-service-tf)
- Embedded code in workflow (Template name: test-access-tf)
- Pull external code and execute within a container (Template name: test-service-tf)

In order to automate the inputs to the workflow during the pipeline run, a workflow parameter configuration map is created which has the inputs for the workflow. The configuration map needs to have the same name as the workflow template. In the following example, the template name is `custom-cnf-sample-test` and a configuration map is created automatically with the same name. As a part of the pipeline run, the pipeline service looks for the configuration map with the template name, gets the inputs, which are then automatically added to the workflow when the pipeline triggers the workflow.

Another update that happens in the test case which triggers the custom workflow is to change the configuration map name to `cn2tmpl-to-kind-map`.

```
- workflow: custom-cnf-sample-test
  kind: ['namespace']
```

The following is an example workflow configuration.

`custom-cnf-sample-test`

```
apiVersion: argoproj.io/v1alpha1
kind: WorkflowTemplate          # new type of k8s spec
metadata:
  name: custom-cnf-sample-test  # name of the workflow spec
  namespace: argo-events
spec:
  serviceAccountName: operate-workflow-sa
  entrypoint: cnf-test-workflow # invoke the workflows template
  hostNetwork: true
  arguments:
    parameters:
      - name: image              # the qf path to a test docker image
        value: not_provided
      - name: kubeconfig_secret  # eg: kubeconfig-989348
        value: not_provided
      - name: report_dir         # eg: /root/SolutionReports
        value: not_provided
  volumes:
```

```

- name: kubeconfig
  secret:
    secretName: "{{ `{{workflow.parameters.kubeconfig_secret}}` }}"
- name: reportdir
  hostPath:
    path: "{{ `{{workflow.parameters.report_dir}}` }}"
templates:
- name: create-cnf-tf
  resource:
    action: apply
    #successCondition: status.succeeded > 0
    #failureCondition: status.failed > 3
    manifest: |
      apiVersion: v1
      kind: Pod
      metadata:
        name: webapp-cnf
        namespace: argo-events
        labels:
          app.kubernetes.io/name: proxy
      spec:
        containers:
          - name: nginx
            image: "{{ .Values.global.docker_image_repo }}/nginx:stable
            ports:
              - containerPort: 80
                name: http-web-svc
- name: create-cnf-service-tf
  resource:
    action: apply
    #successCondition: status.succeeded > 0
    #failureCondition: status.failed > 3
    manifest: |
      apiVersion: v1
      kind: Service
      metadata:
        name: webapp-service
        namespace: argo-events
      spec:
        selector:
          app.kubernetes.io/name: proxy
        ports:
          - name: webapp-http

```



```

        protocol: TCP
        port: 80
        targetPort: http-web-svc
- name: test-access-tf
  script:
    image: "{{ `{{workflow.parameters.image}}` }}"
    command: [python]
    source: |
      import time
      print('--Test access to CNF--')
      url = 'webapp-service.argo-events.svc.cluster.local'
      retry_max = 3
      retry_cnt = 0
      while retry_cnt < retry_max:
        print('Response status code: {}'.format('200'))
        time.sleep(1)
        retry_cnt += 1
        print('Monitoring access count: {}'.format(retry_cnt))
      print('Completed')
- name: test-service-tf
  inputs:
    artifacts:
      - name: pyrunner
        path: /usr/local/src/cn2_py_runner.py
        mode: 0755
      http:
        url: https://raw.githubusercontent.com/roshpr/argotest/main/cn2-experiments/
cn2_py_runner.py
  script:
    image: "{{ `{{workflow.parameters.image}}` }}"
    command: [python]
    args: ["/usr/local/src/cn2_py_runner.py", "4"]
- name: cnf-test-workflow
  dag:
    tasks:
      - name: create-cnf
        template: create-cnf-tf
      - name: create-cnf-service
        template: create-cnf-service-tf
      - name: test-connectivity
        template: test-access-tf
        dependencies: [create-cnf-service]
      - name: test-load

```

```
template: test-service-tf
dependencies: [create-cnf-service]
```

Explanation of Parameters for values.yaml

Table 1: Parameters for values.yaml

Name	Description	Value	Accepted Values
Global Parameters			
global.registry	Global image registry to pull Juniper artifacts	""	enterprise-hub.juniper.net/contrail-container-prod/
global.docker_image_repo	Global docker registry for non-Juniper images	""	docker.io
global.registry_authentication	True if registry required registry_authentication or false	false	true, false
global.imagePullSecret	Image pull secret for authenticated registry in base64 format	""	base64 formatted secret
global.deployment_type	CN2 cluster installed on K8 or OpenShift	""	k8, openshift
global.nginx_nodeport	NGINX port for test result	30552	30552
Global GitServer Parameters			

Table 1: Parameters for values.yaml (Continued)

Name	Description	Value	Accepted Values
global.gitServer.access_token	GitLab personal access token in base64 format	""	base64 formatted token
global.gitServer.gitlabBaseURL	Base server for your GitLab server	""	Example: https://cnf-gitlab.net
global.gitServer.project	Repository/Project name in GitLab	""	Example: devops/cn2config/
global.gitServer.folderName	Folder name where all the CN2 configurations exist	""	Example: cn2networkconfig
global.gitServer.branch	CN2 configuration's Git branch name	""	Example: master
global.gitServer.deploy_username	GitLab repository deploy token username	""	Example: gitlab+deploy-token-154
global.gitServer.deploy_token	GitLab repository deploy token	""	Example: syDXrL-tNjiYo9WUwbNy
Global CN2ClusterDetails Parameters			
global.cn2ClusterDetails.name	CN2 cluster name from kubeconfig file	""	Example: cluster.local
global.cn2ClusterDetails.server	CN2 cluster Kubernetes API server IP address, which is accessible from the management cluster	""	Example: 1.1.2.3
global.cn2ClusterDetails.bearerToken	CN2 cluster Service Account's Bearer Token	""	base64 formatted token

Table 1: Parameters for values.yaml (Continued)

Name	Description	Value	Accepted Values
global.cn2ClusterDetails.kubeconfig	CN2 kubeconfig name	""	cn2-cluster-kubeconfig
global.cn2ClusterDetails.mountpath	CN2 test profile folder location	""	Example: /opt/cn2
Argo Events Parameters			
argo-events.managementServer	Management server Kubernetes API IP address	""	Example: 192.168.1.4
argo-events.pipeline_service.image	Pipeline service image name	""	pipeline_service
argo-events.pipeline_service.tag	Pipeline service prebuild image tag	""	R23.1.272
argo-events.pipeline_service.imagePullPolicy	Image pull policy	""	Always, IfNotPresent
argo-events.pipeline_service_prebuild.image	Pipeline service prebuild image name	""	pipeline_service_postinstall
argo-events.pipeline_service_prebuild.tag	Pipeline service prebuild image tag	""	R23.1.272
argo-events.pipeline_service_prebuild.imagePullPolicy	Image pull policy	""	Always, IfNotPresent

Table 1: Parameters for values.yaml (Continued)

Name	Description	Value	Accepted Values
Workflow Object Parameters			
workflow-objects.image	Test image name	""	cn2-ptk
workflow-objects.tag	Test image tag	""	R23.1.272
workflow-objects.imagePullPolicy	Image pull policy	""	Always, IfNotPresent
workflow-objects.topology	Test configuration file name in mountPath folder	""	test_config_ocp.yaml, test_config_k8s.yaml
workflow-objects.ssl_enabled	If SSL is enabled true, otherwise false	false	false, true
workflow-objects.ocp_api_host_ip	Only for Red Hat OpenShift Container Platform (OCP) Kubernetes API IP address	""	Example: 192.167.19.571
workflow-objects.ocp_api_host_name	Only for OCP Kubernetes API name	""	Example: api.ocp-ss-571.net
workflow-objects.argocd_retry_interval	Retry interval check in seconds	""	60
workflow-objects.argocd_check_retry	Number of retries before timeout	""	5
ArgoCD and ArgoWorkflow Parameters			

Table 1: Parameters for values.yaml (Continued)

Name	Description	Value	Accepted Values
argo-cd.argocdExpose.enabled	Enable exposing outside	""	true
argo-cd.argocdExpose.nodePort	NodePort for Argo CD	""	30551
argo-cd.replicas	Number of replicas	""	1
argo-workflows.argowfExpose.enabled	Enable exposing outside	""	true
argo-workflows.argowfExpose.nodePort	NodePort for Argo Workflow	""	30551
argo-workflows.replicas	Number of replicas	""	1

2

CHAPTER

Architecture and Design

CN2 Pipeline Solution Test Architecture and Design | 36

CN2 Pipeline Solution Test Architecture and Design

SUMMARY

IN THIS SECTION

- [Overview | 36](#)
- [Use Case | 36](#)
- [Workflows | 37](#)
- [Profile | 38](#)
- [Test Environment Configuration | 45](#)
- [Logging and Reporting | 54](#)

Overview

Solution Test Automation Framework (STAF) is a common platform developed for automating and maintaining solution use cases mimicking the real-world production scenarios.

- STAF can granularly simulate and control user-personas, actions, timing at scale and thereby exposing the software to all real-world scenarios with long-running traffic.
- STAF architecture can be extended to allow the customer to plug-in GitOps artifacts and create custom test workflows.
- STAF is implemented in Python and Pytest test framework.

Use Case

STAF emulates Day 0, Day 1, and Day-to-Day operations in a customer environment. Use case tests are performed as a set of test workflows by user-persona. Each user-persona has its own operational scope.

- Operator—Performs global operations, such as cluster setup and maintenance, CN2 deployment, and so on.
- Architect—Performs tenant related operations, such as onboarding, teardown, and so on.

- Site Reliability Engineering (SRE)—Performs operations in the scope of a single tenant only.

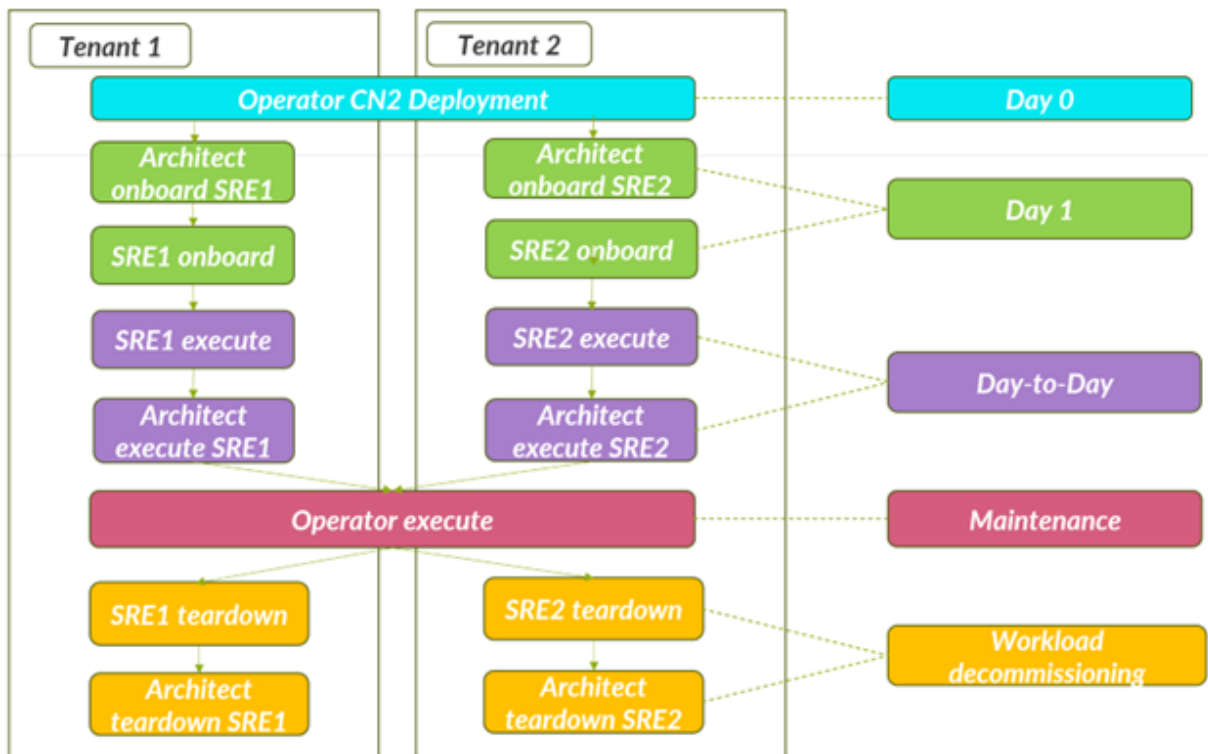
Currently, STAF supports IT cloud webservice and telco use cases.

Workflows

Workflows for each tenant are executed sequentially only. Several tenants' workflows can be executed in parallel, with the exclusion of Operator tests.

Day 0 operation or CN2 deployment is currently independent from test execution. The rest of the workflows are executed as Solution Sanity Tests. In Pytest each workflow is represented by a test suite.

Figure 6: Typical Use Case Scenario



For test descriptions, see ["CN2 Pipeline Test Case Descriptions"](#) on page 70.

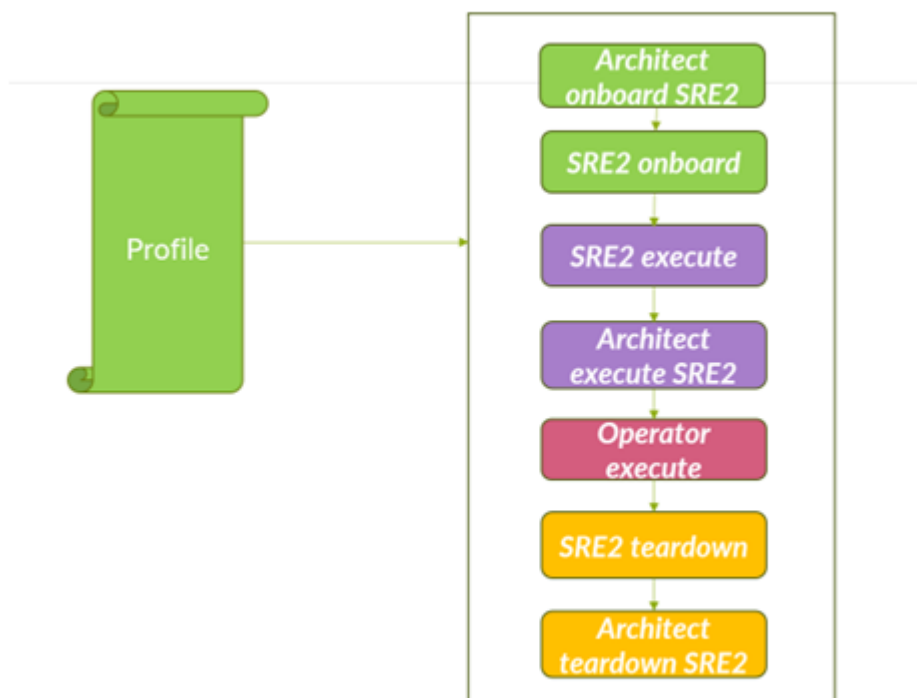
Profile

IN THIS SECTION

- [Isolated LoadBalancer Profile | 39](#)
- [Isolated NodePort Profile | 40](#)
- [Multi-Namespace Contour Ingress LoadBalancer Profile | 41](#)
- [Multi-Namespace Isolated LoadBalancer Profile | 42](#)
- [Non-Isolated Nginx Ingress LoadBalancer Profile | 43](#)

Workflow is executed for a use case instance described in a profile YAML file. The profile YAML describes parameters for namespaces, application layers, network policies, service type, and so on.

Figure 7: Profile Workflow



The profile file is mounted outside of a test container to give you flexibility with choice of scale parameters. For CN2 Pipeline Release 23.1, you can update the total number of pods only.

The complete set of profiles can be accessed from the downloaded CN2 pipeline tar file in folder **charts/workflow-objects/templates**.

The following sections have examples profiles.

Isolated LoadBalancer Profile

The IsolatedLoadBalancerProfile.yml configures a three-tier webservice profile

- Frontend pods are launched using deployment with a replica count of two (2). These frontend pods are accessed from outside of the cluster through the LoadBalancer service.
- Middleware pods are launched using deployment with a replica count of two (2) and allowed address pair is configured on both the pods. These pods are accessible through the ClusterIP service from the frontend pods.
- Backend pods are deployed with a replica count of two (2). Backend pods are accessible from middleware pods through the ClusterIP service.
- Policies are created to allow only specific ports for each tier.

IsolatedLoadbalancerProfile.yml

```
isl-lb-profile:
  WebService:
    isolated_namespace: True
    count: 1
    frontend:
      external_network: custom
      n_pods: 2
      services:
        - service_type: LoadBalancer
      ports:
        - service_port: 21
          target_port: 21
          protocol: TCP
    middleware:
      n_pods: 2
      aap: active-standby
      services:
        - service_type: ClusterIP
```

```

    ports:
      - service_port: 80
        target_port: 80
        protocol: TCP
  backend:
    n_pods: 2
    services:
      - service_type: ClusterIP
        ports:
          - service_port: 3306
            target_port: 3306
            protocol: UDP

```

Isolated NodePort Profile

The `IsolatedNodePortProfile.yml` configures a three-tier webservice profile.

- Frontend pods are launched using deployment with a replica count of two (2). These frontend pods are accessed from outside of the cluster using haproxy node port ingress service.
- Middleware pods are launched using deployment with a replica count of two (2) and allowed address pair is configured on both the pods. These pods are accessible through the ClusterIP service from the frontend pods.
- Backend pods are deployed with a replica count of two (2). Backend pods are accessible from middleware pods through the ClusterIP service.
- Policies are created to allow only specific ports for each tier. Isolated namespace is enabled in this profile.

```

isl-np-web-profile-w-haproxy-ingress:
  WebService:
    count: 1
    isolated_namespace: True
    frontend:
      n_pods: 2
      anti_affinity: true
      liveness_probe: HTTP
      ingress: haproxy_nodeport
      services:
        - service_type: NodePort
          ports:
            - service_port: 443

```

```

        target_port: 443
        protocol: TCP
      - service_port: 80
        target_port: 80
        protocol: TCP
    middleware:
      n_pods: 2
      liveness_probe: command
      services:
        - service_type: ClusterIP
          ports:
            - service_port: 80
              target_port: 80
              protocol: TCP
    backend:
      n_pods: 2
      services:
        - service_type: ClusterIP
          ports:
            - service_port: 3306
              target_port: 3306
              protocol: UDP

```

Multi-Namespace Contour Ingress LoadBalancer Profile

The MultiNamespaceContourIngressLB.yml configures a three-tier webservice profile.

- Frontend pods are launched using deployment with a replica count of two (2). These frontend pods are accessed from outside of the cluster using haproxy node port ingress service.
- Middleware pods are launched using deployment with a replica count of two (2) and allowed address pair is configured on both the pods. These pods are accessible through the ClusterIP service from the frontend pods.
- Backend pods are deployed with a replica count of two (2). Backend pods are accessible from middleware pods through the ClusterIP service.
- Policies are created to allow only specific ports for each tier. Isolated namespace is enabled in this profile.

```

multi-ns-contour-ingress-profile:
  WebService:
    isolated_namespace: True

```

```

multiple_namespace: True
fabric_forwarding: True
count: 1
frontend:
  n_pods: 2
  ingress: contour_loadbalancer
  services:
    - service_type: ClusterIP
      ports:
        - service_port: 6443
          target_port: 6443
          protocol: TCP
middleware:
  n_pods: 2
  services:
    - service_type: ClusterIP
      ports:
        - service_port: 80
          target_port: 80
          protocol: TCP
backend:
  n_pods: 2
  services:
    - service_type: ClusterIP
      ports:
        - service_port: 3306
          target_port: 3306
          protocol: UDP

```

Multi-Namespaced Isolated LoadBalancer Profile

The MultiNamespacedIsolatedLB.yml profile configures a three-tier webservice profile.

- Frontend pods are launched using deployment with a replica count of two (2). These frontend pods are accessed from outside of the cluster using a LoadBalancer service.
- Middleware pods are launched using deployment with a replica count of two (2) and allowed address pair is configured on both the pods. These middleware pods are accessible through the ClusterIP service from the frontend pods.
- Backend pods are deployed with a replica count of two (2). Backend pods are accessible from middleware pods through the ClusterIP service.

- Policies are created to allow only specific ports for each tier. Isolated namespace is enabled in this profile in addition to multiple namespace for frontend, middleware, and backend deployments.

```
multi-ns-lb-profile:
  WebService:
    isolated_namespace: True
    multiple_namespace: True
    count: 1
    frontend:
      n_pods: 2
      services:
        - service_type: LoadBalancer
          ports:
            - service_port: 443
              target_port: 443
              protocol: TCP
            - service_port: 6443
              target_port: 6443
              protocol: TCP
    middleware:
      n_pods: 2
      aap: active-standby
      services:
        - service_type: ClusterIP
          ports:
            - service_port: 80
              target_port: 80
              protocol: TCP
    backend:
      n_pods: 2
      services:
        - service_type: ClusterIP
          ports:
            - service_port: 3306
              target_port: 3306
              protocol: UDP
```

Non-Isolated Nginx Ingress LoadBalancer Profile

The NonIsolatedNginxIngressLB .yaml profile configures a three-tier webservice profile.

- Frontend pods are launched using deployment with a replica count of two (2). These frontend pods are accessed from outside of the cluster using a NGINX ingress LoadBalancer service.
- Middleware pods are launched using deployment with a replica count of two (2) and allowed address pair is configured on both the pods. These middleware pods are accessible through the ClusterIP service from the frontend pods.
- Backend pods are deployed with a replica count of two (2). Backend pods are accessible from middleware pods through the ClusterIP service.
- Policies are created to allow only specific ports for each tier.

```
non-isl-nginx-ingress-lb-profile:
```

```
  WebService:
```

```
    isolated_namespace: False
```

```
    count: 1
```

```
  frontend:
```

```
    ingress: nginx_loadbalancer
```

```
    n_pods: 2
```

```
    liveness_probe: HTTP
```

```
    services:
```

```
      - service_type: ClusterIP
```

```
        ports:
```

```
          - service_port: 80
```

```
            target_port: 80
```

```
            protocol: TCP
```

```
  middleware:
```

```
    n_pods: 2
```

```
    services:
```

```
      - service_type: ClusterIP
```

```
        ports:
```

```
          - service_port: 443
```

```
            target_port: 443
```

```
            protocol: TCP
```

```
  backend:
```

```
    n_pods: 2
```

```
    is_deployment: False
```

```
    liveness_probe: command
```

```
    services:
```

```
      - service_type: ClusterIP
```

```
        ports:
```

```
          - service_port: 3306
```



```
target_port: 3306
protocol: UDP
```

Test Environment Configuration

IN THIS SECTION

- [Kubernetes Environment | 45](#)
- [OpenShift Environment | 49](#)
- [Kubeconfig File | 53](#)

To configure the test environment, you deploy a YAML file which contains parameters that describe the test execution environment. This topic shows example YAML files for both Kubernetes and OpenShift test environment configuration.

Kubernetes Environment

Following are example YAML files used to deploy and configure a Kubernetes test environment.

CN2 in Kubernetes Environment — No vMX and No VM

The following is an example YAML for configuring a Kubernetes test environment that does not have a Juniper Networks® MX Series 3D Universal Edge Router (vMX) and does not have an external virtual machine (VM) setup in the test environment.

```
k8s_clusters:
  central:
    authentication:
      type: cn2_client_cert
      kubeconfig_file: /root/.kube/config
    metadata:
      kubemanager: contrail-k8s-kubemanager
      multus: False
      ingress:
        haproxy_nodeport:
```

```

    ingress_class_name: haproxy-nodeport
    namespace: haproxy-controller
    service_name: haproxy-ingress
    service_port: 80
  nginx_loadbalancer:
    ingress_class_name: nginx-loadbalancer
    namespace: ingress-nginx
    service_name: ingress-nginx-controller
    service_port: 80
  contour_loadbalancer:
    ingress_class_name: contour
    namespace: projectcontour
    service_name: envoy
    service_port: 80

computes:
  cn2-test-node-1:
    username: username_is_here
    password: password_is_here
    mgmt_ip: ip_is_here
    data_ip: ip_is_here
  cn2-test-node-2:
    username: username_is_here
    password: password_is_here
    mgmt_ip: ip_is_here
    data_ip: ip_is_here
  cn2-test-node-3:
    username: username_is_here
    password: password_is_here
    mgmt_ip: ip_is_here
    data_ip: ip_is_here
  cn2-test-node-4:
    username: username_is_here
    password: password_is_here
    mgmt_ip: ip_is_here
    data_ip: ip_is_here
  cn2-test-node-5:
    username: username_is_here
    password: password_is_here
    mgmt_ip: ip_is_here
    data_ip: ip_is_here
  cn2-test-node-6:
    username: username_is_here

```

```

password: password_is_here
mgmt_ip: ip_is_here
data_ip: ip_is_here

test_config:
  deployment:
    type: k8s
  traffic_image: enterprise-hub.jnpr.net/contrail-container-internal/ubuntu-traffic:v2.0.1
  registry:
    type: authenticated

```

CN2 in Kubernetes Environment — Standard Test Setup

The following YAML is a standard test setup for configuring a Kubernetes test environment.

```

k8s_clusters:
  central:
    authentication:
      type: cn2_client_cert
      kubeconfig_file: /root/.kube/config
    metadata:
      kubemanager: contrail-k8s-kubemanager
      multus: False
    dc_gw:
      - vmx-1
    rr:
      - vmx-1
    public_endpoint:
      - cn2-sanity-public-endpoint-1
    ingress:
      haproxy_nodeport:
        ingress_class_name: haproxy-nodeport
        namespace: haproxy-controller
        service_name: haproxy-ingress
        service_port: 80
      nginx_loadbalancer:
        ingress_class_name: nginx-loadbalancer
        namespace: ingress-nginx
        service_name: ingress-nginx-controller
        service_port: 80
      contour_loadbalancer:

```

```

        ingress_class_name: contour
        namespace: projectcontour
        service_name: envoy
        service_port: 80
devices:
  vmx-1:
    mgmt_ip: ip_is_here
    type: mx
    user_name: username_is_here
    password: password_is_here
    labels:
      - rr
      - gw
    attributes:
      public_interface: ge-0/0/1
      loopback: 10.1.1.1
      public_vrf: sanity-default-public-network
      config_group_name: DCGW
      bgp_group_name: Sanity-Cluster
      public_rt: "target:6000:99"

computes:
  cn2-sanity-public-endpoint-1:
    mgmt_ip: ip_is_here
    attributes:
      public_address: ip_is_here
    interfaces:
      ens192:
        peer: vmx-1

  cn2-test-node-1:
    username: username_is_here
    password: password_is_here
    mgmt_ip: ip_is_here
    data_ip: ip_is_here
  cn2-test-node-2:
    username: username_is_here
    password: password_is_here
    mgmt_ip: ip_is_here
    data_ip: ip_is_here
  cn2-test-node-3:
    username: username_is_here
    password: password_is_here

```

```

    mgmt_ip: ip_is_here
    data_ip: ip_is_here
cn2-test-node-4:
    username: username_is_here
    password: password_is_here
    mgmt_ip: ip_is_here
    data_ip: ip_is_here
cn2-test-node-5:
    username: username_is_here
    password: password_is_here
    mgmt_ip: ip_is_here
    data_ip: ip_is_here
cn2-test-node-6:
    username: username_is_here
    password: password_is_here
    mgmt_ip: ip_is_here
    data_ip: ip_is_here

test_config:
  deployment:
    type: k8s
  traffic_image: enterprise-hub.jnpr.net/contrail-container-internal/ubuntu-traffic:v2.0.1
  registry:
    type: authenticated

```

OpenShift Environment

Following are example YAML files used to deploy and configure a Red Hat OpenShift test environment.

CN2 in OpenShift Environment — No vMX and No VM Setup

The following is an example YAML for configuring an OpenShift test environment that does not have a Juniper Networks® MX Series 3D Universal Edge Router (vMX) and does not have an external virtual machine (VM) setup in the test environment.

```

k8s_clusters:
  central:
    authentication:
      type: cn2_client_cert
      kubeconfig_file: /root/.kube/config
    metadata:

```

```

kubemanager: contrail-k8s-kubemanager
multus: True
ingress:
  haproxy_nodeport:
    ingress_class_name: haproxy-nodeport
    namespace: haproxy-controller
    service_name: haproxy-ingress
    service_port: 80
  nginx_loadbalancer:
    ingress_class_name: nginx
    namespace: nginx-ingress
    service_name: nginxingress-sample-nginx-ingress
    service_port: 80
  contour_loadbalancer:
    ingress_class_name: contour
    namespace: projectcontour
    service_name: envoy
    service_port: 80

test_config:
  deployment:
    type: openshift
  traffic_image: enterprise-hub.jnpr.net/contrail-container-internal/ubuntu-traffic:v2.0.1
  registry:
    type: authenticated
  ocp_infra:
    ocp_api_host:
      ip: ip_is_here
      name: api.ocp.domain_is_here
    ocp_native_apps_host:
      ip: ip_is_here

computes:
  master1:
    username: username_is_here
    password: password_is_here
    mgmt_ip: ip_is_here
    data_ip: ip_is_here
  master2:
    username: username_is_here
    password: password_is_here

```

```

    mgmt_ip: ip_is_here
    data_ip: ip_is_here
master3:
    username: username_is_here
    password: password_is_here
    mgmt_ip: ip_is_here
    data_ip: ip_is_here
worker1:
    username: username_is_here
    password: password_is_here
    mgmt_ip: ip_is_here
    data_ip: ip_is_here
worker2:
    username: username_is_here
    password: password_is_here
    mgmt_ip: ip_is_here
    data_ip: ip_is_here

```

CN2 in OpenShift Environment – Standard Test Setup

The following YAML is a standard test setup for configuring an OpenShift test environment.

```

k8s_clusters:
  central:
    authentication:
      type: cn2_client_cert
      kubeconfig_file: /root/.kube/config
    metadata:
      kubemanager: contrail-k8s-kubemanager
      multus: True
      dc_gw:
        - vmx1
      rr:
        - vmx1
    public_endpoint:
      - cn2-sanity-public-endpoint-1
    ingress:
      haproxy_nodeport:
        ingress_class_name: haproxy-nodeport
        namespace: haproxy-controller
        service_name: haproxy-ingress

```

```

        service_port: 80
    nginx_loadbalancer:
        ingress_class_name: nginx
        namespace: nginx-ingress
        service_name: nginxingress-sample-nginx-ingress
        service_port: 80
    contour_loadbalancer:
        ingress_class_name: contour
        namespace: projectcontour
        service_name: envoy
        service_port: 80

devices:
    vmx1:
        mgmt_ip: ip_is_here
        type: mx
        user_name: user_name_is_here
        password: password_is_here
        labels:
            - rr
            - gw
        attributes:
            public_interface: ge-0/0/1
            loopback: 10.1.1.1
            public_vrf: sanity-default-public-network
            config_group_name: DCGW
            bgp_group_name: Sanity-Cluster
            public_rt: "target:64512:1000"

test_config:
    deployment:
        type: openshift
    traffic_image: enterprise-hub.jnpr.net/contrail-container-internal/ubuntu-traffic:v2.0.1
    registry:
        type: authenticated
    ocp_infra:
        ocp_api_host:
            ip: ip_is_here
            name: api.ocp.domain_is_here
        ocp_native_apps_host:
            ip: ip_is_here

computes:

```



```

master1:
  username: username_is_here
  password: password_is_here
  mgmt_ip: ip_is_here
  data_ip: ip_is_here
master2:
  username: username_is_here
  password: password_is_here
  mgmt_ip: ip_is_here
  data_ip: ip_is_here
master3:
  username: username_is_here
  password: password_is_here
  mgmt_ip: ip_is_here
  data_ip: ip_is_here
worker1:
  username: username_is_here
  password: password_is_here
  mgmt_ip: ip_is_here
  data_ip: ip_is_here
worker2:
  username: username_is_here
  password: password_is_here
  mgmt_ip: ip_is_here
  data_ip: ip_is_here

cn2-sanity-public-endpoint-1:
  mgmt_ip: ip_is_here
  username: username_is_here
  password: password_is_here
  attributes:
    public_address: ip_is_here
  interfaces:
    ens160:
      peer: vmx1

```

Kubeconfig File

The kubeconfig file data is used for authentication. The kubeconfig file is stored as a secret on the Argo host Kubernetes cluster.

Required Data:

- Server: Secret key should point to either server IP address or host name.
- For Kubernetes setups, point to the master node IP address: server:

```
https://xx.xx.xx.xx:6443
```

- For OpenShift setups, point to the OpenShift Container Platform API server, extension, and server:

```
https://api.ocp.xxxx.com:6443
```

- Client certificate: Kubernetes client certificate.

Logging and Reporting

Two types of log files are created during each test run:

- Pytest session log file—One per session
- Test suite log file—One per test suite

Default file size is 50 MB. Log file rotation is supported.

3

CHAPTER

Test Cases

CN2 Pipeline Test Management | 56

Test Cases for No vMX Setup per Profile | 59

Test Cases for a Standard Setup per Profile | 64

CN2 Pipeline Test Case Descriptions | 70

CN2 Pipeline Test Management

SUMMARY

This section covers tasks that are specific to managing the test cases.

IN THIS SECTION

- [Trigger the CN2 Pipeline Test | 56](#)
- [Change Commit Threshold Trigger | 56](#)
- [Change Test Profiles | 57](#)
- [Identify CN2 Pipeline Test Trigger | 58](#)
- [Access Test Results | 58](#)
- [Uninstall the CN2 Pipeline | 59](#)

Trigger the CN2 Pipeline Test

Any valid commit automatically triggers the CN2 pipeline test. For the commit to be valid, you need to do the following:

1. Apply the commit to the correct branch mentioned in the values.yaml.
2. Verify that Argo CD is in sync state.
3. Verify the commit will cross the threshold trigger.

You can verify the threshold trigger setting in the configuration map named cn2-pipeline-configs.

Change Commit Threshold Trigger

Use this procedure to change the commit threshold trigger.

1. To access the configuration map named cn2-pipeline-configs, run this command:

```
kubectl get cm -n argo-events cn2pipeline-configs -o yaml
```

Output:

```
apiVersion: v1
data:
  argocd_check_retry: "5"
  argocd_retry_interval: "120"
  testcase_trigger_threshold: "0"
```

2. Change the `testcase_trigger_threshold` value for the number of commits you want to ignore. By default, this is set to 0.

Change Test Profiles

Use this procedure to change test profiles. This mapping configuration contains the workflow template to CN2 resource kind mapping. Only one template is selected for execution and the first map that matches has the higher priority. An asterisk (*) in kind: ['*'] has higher priority than any other kind matches and overrides every mapping.

1. To access the configuration map named `cn2tmpl-to-kind-map`, run this command:

```
kubectl get cm -n argo-events cn2tmpl-to-kind-map -o yaml
```

Output:

```
apiVersion: v1
data:
  kindmap: |
    - workflow: it-cloud-arch1-sre2
      kind: ['*']
    - workflow: custom-cnf-sample-test
      kind: ['namespace']
    - workflow: it-cloud-arch1-sre1
      kind: ['service']
    - workflow: it-cloud-arch2-sre3
      kind: ['service']
    - workflow: it-cloud-arch2-sre4
      kind: ['service']
    - workflow: it-cloud-arch2-sre5
```

```
kind: ['namespace']
kind: ConfigMap
```

CN2 pipeline will trigger the test where the `kind:['*']` value exists. By default, the profile `it-cloud-arch1-sre2` is triggered. If we want to trigger any other profile

2. To trigger another profile, change profile kind to `kind:['*']`.

NOTE: Only one workflow can have `*as kind:` value.

Identify CN2 Pipeline Test Trigger

After installing CN2 pipeline, every valid commit in a given branch and folder will trigger the CN2 test. There is a commit processing workflow triggered after every commit that starts from `resource-*`, for example where the `kind:['*']` value exists. This processes the commit and validates the commit against the threshold value and Argo CD synchronize.

- The commit processing workflow looks like this:

resource-workflow-5547v-3345460271	0/2	Completed	0	11h
------------------------------------	-----	-----------	---	-----

- The actual test workflow starts from CN2 and looks like this:

cn2-test-workflow-zfhtf-4049010958	0/2	Completed	0	13h
------------------------------------	-----	-----------	---	-----

Access Test Results

You can access the test results in HTML format from the GUI.

- To access test results, enter the following in your browser:

```
https://<management_api_ip_address>:30552
```

Uninstall the CN2 Pipeline

Use this procedure to uninstall and delete the CN2 pipeline.

- To uninstall CN2 pipeline, run this command:

```
helm uninstall cn2-pipeline
```

Output:

```
kubectl patch -n argo-events eventsource/gitlab -p '{"metadata":{"finalizers":[]}}' --type=merge
kubectl patch -n argo-events sensor/gitlab -p '{"metadata":{"finalizers":[]}}' --type=merge
kubectl patch -n argo-events eventbus/default -p '{"metadata":{"finalizers":[]}}' --type=merge
kubectl patch -n argocd applications.argoproj.io/cn2networkconfig -p '{"metadata":{"finalizers":[]}}' --type=merge
```

Test Cases for No vMX Setup per Profile

Please note that test case steps related to an external traffic verification or Juniper Networks® vMX Virtual Router (vMX) configuration updates are not applicable for this type of a setup.

Listed in the table are the profiles with their test suites and test cases run for that profile.

Table 2: Test Cases for No VMX Setup per Profile.

Profile	Test Case Steps
<i>IsolatedNodePortProfile</i>	
Test Suite	Test Case
architect1_onboard_sre1	create_namespaces
sre1_onboard	onboard_services

Table 2: Test Cases for No VMX Setup per Profile. *(Continued)*

Profile	Test Case Steps
sre1_execute	modify_liveness_probe
sre1_execute	update_cluster_ip_service
sre1_execute	update_nodeport_service
sre1_execute	update_ingress_network_policy
sre1_execute	update_egress_network_policy
sre1_execute	update_network_policy_policy_types
sre1_execute	update_ingress_service
architect1_execute_sre1	validate_link_local_service
sre1_tearardown	teardown_services
architect1_tearardown_sre1	teardown_namespaces
<i>IsolatedLoadbalancerProfile</i>	
Test Suite	Test Case
architect1_onboard_sre2	create_namespaces
sre2_onboard	onboard_services
sre2_execute	update_cluster_ip_service

Table 2: Test Cases for No VMX Setup per Profile. *(Continued)*

Profile	Test Case Steps
sre2_execute	update_service_type
sre2_execute	update_ingress_network_policy
sre2_execute	update_egress_network_policy
sre2_execute	update_network_policy_policy_types
sre2_execute	validate_allowed_address_pair_failover
sre2_execute	validate_allowed_address_pair_update
architect1_execute_sre2	validate_link_local_service
sre2_tearardown	teardown_services
architect1_tearardown_sre2	teardown_namespaces
<i>NonIsolatedNginxIngressLB</i>	
Test Suite	Test Case
architect2_onboard_sre3	create_namespaces
sre3_onboard	onboard_services
sre3_execute	modify_liveness_probe
sre3_execute	update_cluster_ip_service

Table 2: Test Cases for No VMX Setup per Profile. *(Continued)*

Profile	Test Case Steps
sre3_execute	update_ingress_network_policy
sre3_execute	update_egress_network_policy
sre3_execute	update_network_policy_policy_types
sre3_execute	update_ingress_service
sre3_execute	update_label_of_pods
architect2_execute_sre3	validate_link_local_service
sre3_tearardown	teardown_services
architect2_tearardown_sre3	teardown_namespaces
<i>MultiNamespaceContourIngressLB</i>	
Test Suite	Test Case
sre4_onboard	onboard_services
sre4_execute	update_cluster_ip_service
sre4_execute	update_ingress_network_policy
sre4_execute	update_egress_network_policy
sre4_execute	update_network_policy_policy_types

Table 2: Test Cases for No VMX Setup per Profile. *(Continued)*

Profile	Test Case Steps
architect2_execute_sre4	validate_link_local_service
architect2_execute_sre4	update_namespace_label
architect2_execute_sre4	update_forwarding_mode_on_namespace
architect2_onboard_sre4	create_namespaces
sre4_tearardown	teardown_services
architect2_tearardown_sre4	teardown_namespaces
<i>MultiNamespacelolatedLB</i>	
Test Suite	Test Case
architect2_onboard_sre5	create_namespaces
sre5_onboard	onboard_services
sre5_execute	update_cluster_ip_service
sre5_execute	update_service_type
sre5_execute	update_ingress_network_policy
sre5_execute	update_egress_network_policy
sre5_execute	update_network_policy_policy_types

Table 2: Test Cases for No VMX Setup per Profile. *(Continued)*

Profile	Test Case Steps
sre5_execute	validate_allowed_address_pair_failover
sre5_execute	validate_allowed_address_pair_update
architect2_execute_sre5	validate_link_local_service
architect2_execute_sre5	update_namespace_label
sre5_tearardown	teardown_services
architect2_tearardown_sre5	teardown_namespaces

Test Cases for a Standard Setup per Profile

Table 3: Test Cases for a Standard Setup per Profile

Profile	Test Case Steps
<i>IsolatedNodePortProfile</i>	
Test Suite	Test Case
architect1_onboard_sre1	create_namespaces
sre1_onboard	onboard_services
sre1_execute	validate_flow_stickiness

Table 3: Test Cases for a Standard Setup per Profile *(Continued)*

Profile	Test Case Steps
sre1_execute	modify_liveness_probe
sre1_execute	update_cluster_ip_service
sre1_execute	update_nodeport_service
sre1_execute	update_ingress_network_policy
sre1_execute	update_egress_network_policy
sre1_execute	update_network_policy_policy_types
sre1_execute	update_ingress_service
architect1_execute_sre1	validate_link_local_service
sre1_tearardown	teardown_services
architect1_tearardown	teardown_namespaces
<i>IsolatedLoadbalancerProfile</i>	
Test Suite	Test Case
architect1_onboard_sre2	create_namespaces
sre2_onboard	onboard_services
sre2_execute	update_cluster_ip_service

Table 3: Test Cases for a Standard Setup per Profile *(Continued)*

Profile	Test Case Steps
sre2_execute	update_service_type
sre2_execute	update_ingress_network_policy
sre2_execute	update_egress_network_policy
sre2_execute	update_network_policy_policy_types
sre2_execute	update_loadbalancer_service_general_properties
sre2_execute	validate_allowed_address_pair_failover
sre2_execute	validate_allowed_address_pair_update
sre2_execute	update_lb_service_static_public_vn
sre2_execute	update_label_of_public_network
architect1_execute_sre2	validate_link_local_service
sre2_tearardown	teardown_services
architect1_tearardown_sre2	teardown_namespaces
<i>NonIsolatedNginxIngressLB</i>	
Test Suite	Test Case
architect2_onboard_sre3	create_namespaces

Table 3: Test Cases for a Standard Setup per Profile *(Continued)*

Profile	Test Case Steps
sre3_onboard	onboard_services
sre3_execute	modify_liveness_probe
sre3_execute	update_cluster_ip_service
sre3_execute	update_ingress_network_policy
sre3_execute	update_egress_network_policy
sre3_execute	update_network_policy_policy_types
sre3_execute	update_ingress_service
sre3_execute	update_label_of_pods
sre3_execute	validate_port_mirror
architect2_execute_sre3	validate_link_local_service
sre3_tearardown	teardown_services
architect2_tearardown_sre3	teardown_namespaces
<i>MultiNamespaceContourIngressLB</i>	
Test Suite	Test Case
architect2_onboard_sre4	create_namespaces

Table 3: Test Cases for a Standard Setup per Profile *(Continued)*

Profile	Test Case Steps
sre4_onboard	onboard_services
sre4_execute	update_cluster_ip_service
sre4_execute	update_ingress_network_policy
sre4_execute	update_egress_network_policy
sre4_execute	update_network_policy_policy_types
sre4_execute	update_ingress_service
architect2_execute_sre4	validate_link_local_service
architect2_execute_sre4	update_namespace_label
architect2_execute_sre4	validate_mesh_virtual_network_router
architect2_execute_sre4	validate_hub_spoke_virtual_network_router
architect2_execute_sre4	update_forwarding_mode_on_namespace
sre4_tearardown	teardown_services
architect2_tearardown_sre4	teardown_namespaces
<i>MultiNamespacelolatedLB</i>	
Test Suite	Test Case

Table 3: Test Cases for a Standard Setup per Profile *(Continued)*

Profile	Test Case Steps
architect2_onboard_sre5	create_namespaces
sre5_onboard	onboard_services
architect2_execute, 1, sre5	validate_link_local_service
architect2_execute, 1, sre5	update_namespace_label
architect2_execute, 1, sre5	validate_mesh_virtual_network_router
architect2_execute, 1, sre5	validate_hub_spoke_virtual_network_router
architect2_execute, 1, sre5	update_forwarding_mode_on_namespace
architect2_execute, 1, sre5	update_fabric_forwarding_on_external_vn
sre5_execute	update_cluster_ip_service
sre5_execute	update_service_type
sre5_execute	update_ingress_network_policy
sre5_execute	update_egress_network_policy
sre5_execute	update_network_policy_policy_types
sre5_execute	update_loadbalancer_service_general_properties
sre5_execute	validate_allowed_address_pair_failover
sre5_execute	validate_allowed_address_pair_update

Table 3: Test Cases for a Standard Setup per Profile *(Continued)*

Profile	Test Case Steps
sre5_execute	update_lb_service_static_public_vn
sre5_tearardown	teardown_services
architect2_tearardown_sre5	teardown_namespaces

CN2 Pipeline Test Case Descriptions

SUMMARY

This section provides the complete list of solution test cases with descriptions of what the test case is performing.

IN THIS SECTION

- [Architect Onboard | 70](#)
- [Architect Execute | 71](#)
- [Architect Teardown | 73](#)
- [SRE Onboard | 73](#)
- [SRE Execute | 73](#)
- [SRE teardown | 78](#)

Architect Onboard

test_create_namespaces: : all profiles

1. For each profile in the use case profiles, create a webservice instance.
2. Inside the webservice instance, create all the required namespaces.

Architect Execute

test_validate_link_local_service

1. Update GlobalVrouterConfig (GVC) with the localhost service.
2. Update GVC with multiple services and with one service having multiple fabric IP addresses.
3. Delete the newly added LLS service.
4. Restore the original LLS set.

test_validate_mesh_virtual_network_router

1. Remove all imports from all virtual network routers (VNR)s. Routes should not be advertised.
2. Add all the other VNRs to each other VNRs. Creates mesh of all three layers.
3. Update NetworkPolicy so that backend can talk to frontend.
4. Cascading VNR, remove the direct import of VNRs between frontend and backend.
5. Negative test of updating the VNR type from mesh to hub.
6. Update the virtual network (VN) label selector of middleware service VNR.
7. Remove the added VN label selector.
8. Update the VNR label of middleware service VNR.
9. Reset the VNRs as specified by the profile.

test_update_namespace_label

1. Update labels of namespace from name=ns1 to name=ns2.
2. Update the corresponding VirtualNetworkRouter objects.

3. Update namespaceSelector in the backend NetworkPolicy rule.
4. Reset the Namespace label, VNR selectors, and NetworkPolicy selector.

test_validate_hub_spoke_virtual_network_router

1. Convert middleware to hub VNR. Convert frontend and backend to spoke VNR.
2. Create new mesh VNRs and interconnect all.
3. Delete the middleware hub VNR and recreate the same later in the process.
4. Add dummy VN label to backend spoke VNR (Negative).
5. Revert back to the proper VN label.
6. Update backend VNR as hub VNR (Negative).
7. Update backend VNR as mesh VNR (Negative).
8. Delete the newly created mesh VNR.
9. Create a new hub VNR for middleware pod network without any import statements.
10. Remove middleware pod network label from original hub VNR.
11. Update the duplicate hub VNR with the import of the backend spoke VNR.
12. Update backend spoke VNR with new middleware pod hub VNR.
13. Change metadata label on new middleware VNR.
14. Update import statements of backend spoke VNR
15. Create a custom VN matching the backend VN label.
16. Update custom VN label to dummy so traffic should fail.
17. Remove the labels from backend VNR so all the VNs in the namespace are selected.
18. Reset the configurations to the baseline profile defined.

test_update_forwarding_mode_on_namespace

1. Set annotations forwarding mode to false (no ip-fabric or fabric-snat).
2. Set forwarding mode to ip-fabric.

3. Set forwarding mode to fabric-snat.
4. Reset forwarding mode to original forwarding mode.

test_update_fabric_forwarding_on_external_vn

1. Enable fabric forwarding on external VN.
2. Reset fabric forwarding on external VN.

Architect Teardown

test_teardown_namespaces: all profiles

1. For each profile in the use case profiles, teardown the namespaces.

SRE Onboard

test_onboard_services: all profiles

1. For each profile in use case profiles, create number of instances mentioned by count.
2. Setup deployment, services, and traffic generator for each instance.

SRE Execute

test_modify_liveness_probe

1. Validate HTTP liveness probe failure.
2. Validate Exec liveness probe failure.

test_update_cluster_ip_service

1. Create new middleware pods as a replica of existing ones.
2. Update selector in NetworkPolicy and Service.
3. Update NetworkPolicy with additional ports.
4. Update the target port from 9091 to 9092.
5. Update the service port from 9091 to 9090.
6. Update protocol of ClusterIP from TCP to UDP.
7. Remove the newly added service UDP/9090/9092.
8. Multiple services mapping to the same podSelector label.
9. Delete the newly created set of middleware pods and set the ClusterIP selector.

test_update_nodeport_service

1. Create new frontend pods as a replica of existing ones.
2. Update selector with new labels.
3. Update NetworkPolicy with additional ports.
4. Update the target port from 9091 to 9092.
5. Update the service port from 9091 to 9090.
6. Update session affinity of the ClientIP service to a specific pod.
7. Update external policy to local.
8. Remove the newly added service TCP/9090/9092.
9. Multiple services mapping to the same podSelector label.
10. Update NodePort port from x to x+1 and validate the traffic.
11. Delete the newly created set of frontend pods and set the ClusterIP selector.

test_update_service_type

1. Update service type from LoadBalancer to NodePort.
2. Reset the service type back to LoadBalancer.
3. Update the service type from ClusterIP to NodePort.
4. Reset the service type back to ClusterIP.

test_update_ingress_network_policy

1. Change the NetworkPolicy ingress rules from match on podLabel to namespaceLabel.
2. Delete all the ingress rules. Frontend to middleware traffic will be dropped - deny all.
3. Update the ingress to list of Empty dictionaries (Allow all traffic).
4. Add back the ingress PolicyType and ingress rules.
5. Add ingress rule with ipBlock - CIDR information. (Update podSelector to ipBlock alone - /16).
6. New NetworkPolicy with ip_block cidr and exception rule.
7. Update the rule such as exception for a specific IP address. (Deny from one of the frontend pods alone).
8. Update the rule so that rule has all of the three filters namespaceSelector, podSelector, and AddressBlock.
9. Reset the rules to rules specified by the profile.

test_update_egress_network_policy

1. Change the NetworkPolicy egress rules from match on ip_block to namespaceLabel.
2. Delete all the egress rules. (Middleware to backend traffic will be dropped - deny all).
3. Update the egress to list of Empty dictionaries. (Allow all traffic).
4. Add back the egress PolicyType and egress rules.
5. Update the rule such as except a specific IP address. (Deny to backend service).
6. New NetworkPolicy for same podSelector but allow the except address.
7. Update the rule so that rule has all of the three filters namespaceSelector, podSelector, and AddressBlock.

8. Reset the rules to rules specified by the profile.

test_update_network_policy_policy_types

1. Modify PolicyTypes to have ingress alone. (Deny all outgoing.)
2. Modify PolicyTypes to have egress alone. (Deny all incoming.)
3. Modify PolicyTypes to have none of the policy types. (Deny all incoming and outgoing.)
4. Reset the rules to rules specified by the profile.

test_update_loadbalancer_service_general_properties

1. Create new frontend pods as a replica of existing ones.
2. Update selector with new labels.
3. Update NetworkPolicy with additional ports.
4. Update the target port from 9091 to 9092.
5. Update the service port from 9091 to 9090.
6. Update session affinity of ClientIP to a specific pod.
7. Update external policy to local.
8. Remove the newly added service TCP/9090/9092.
9. Multiple services mapping to the same podSelector label.
10. Delete the newly created set of frontend pods and reset selectors to previously existing labels as a test case teardown.

test_validate_allowed_address_pair_failover

1. Trigger Virtual Router Redundancy Protocol (VRRP) master switchover.
2. Configure Allowed Address Pairs (AAP) mode as active-active.
3. Reset AAP mode to active-backup.
4. Reset VRRP master status.

test_validate_allowed_address_pair_update

1. Update AAP IP from x to y .
2. Update AAP IP to have multiple addresses.

test_update_lb_service_static_public_vn

1. Create a VN (`new-public-vn`) under the service namespace with custom public RT1 assigned. Also, configure Juniper Networks® MX Series 5G Universal Routing Platform (MX) with routing instance and route targets with the same VN properties.
2. Create a VN (`new-public-vn`) under default namespace with custom public RT2 assigned and configure MX accordingly.
3. Update the namespace annotations `ExternalNetwork` to default `new-public-vn`.
4. Create a new LoadBalancer service with annotations `ExternalNetwork` and value as `new-public-vn`.
5. Update `ExternalIP` on the LoadBalancer service (both IPv4 and IPv6).
6. Delete the LoadBalancer service and validate.
7. Create a service without any annotation specified and validate traffic.
8. Exhaust the IP address on the IPv4 IP addresses on the public subnet by creating dummy LoadBalancer services.
9. Create one more service and check if service is in pending state as the IP address pool is exhausted.
10. Delete one of the dummy LoadBalancer services.
11. Validate the service which was created in Step 9.
12. Update Namespace annotation `external-virtual-network` to `new-public-vn` (without namespace).
13. Create a service without any annotation specified and validate traffic.
14. Reset the namespace annotations to original value.
15. Delete all services.

test_update_ingress_service

1. Create additional service for frontend.

2. Update the service selector label of ingress backend from old service to new service.
3. Have multiple service paths mentioned in the ingress specification.
4. Remove one of the paths.
5. Delete the ingress service.
6. Recreate the ingress service.

test_update_label_of_public_network

1. Update custom public networks label from `local==public-test` to `local==unselect-public-vn`. Existing service should not be affected.
2. Create a new LoadBalancer service with `service.contrail.juniper.net/externalNetworkSelector=custom-external-in-service-namespace` annotation set. New service moves to pending state.
3. Reset the label to `local==public-test`. The new service created in Step 2 will get a public IP address and is accessible from the Internet endpoint.
4. Delete the new service.

test_update_label_of_pods

1. Update pod label, corresponding service, and NetworkPolicy selectors.
2. Reset pod label, corresponding service, and NetworkPolicy selectors.

SRE teardown

test_teardown_services

1. Delete SRE objects created during the SRE onboard phase.