**5G ENM SMF/AMF NODE ADDITION/DELETION**

**USER DOCUMENT - LLD**

Vijay Daniel

**Table of Contents**

1. [Introduction](#introduction)-------------------------------------------------------------------------------------------------------3
2. [Synopsis](#synopsis)------------------------------------------------------------------------------------------------------------3
3. [Prerequisites](#prerequisites)------------------------------------------------------------------------------------------------------3
4. [Manual Job Execution](#manual-job-execution)------------------------------------------------------------------------------------------4
5. [Prerequisites](#prerequisites) of Variable--------------------------------------------------------------------------------------10
6. Understanding of Pipeline Execution steps --------------------------------------------------------------11
7. Pipeline Flow ----------------------------------------------------------------------------------------------------12
8. Procedure for Onboarding for a New APP with new TARGET CLUSTER in production---------14
9. DATA FLOW & SEQUENCE DIAGRAM-----------------------------------------------------------------------18
10. Junit Tests---------------------------------------------------------------------------------------------------------24

**Appendix A: Best Practices -----------------------------------------------------------------------------------------25**

**Appendix B: Guardrails----- -----------------------------------------------------------------------------------------28**

1. **Introduction**

This document provides detailed instructions for setting up and using a GitLab CI/CD pipeline for the automation of SMF and AMF node addition and deletion tasks with command sets into ENM. The pipeline ensures tasks are executed in a controlled and repeatable manner, integrating Ansible playbooks and custom Python scripts. Guidelines and best practices/guardrails and outline also provided to onboard for new apps as well.

**2. Synopsis**

The GitLab CI/CD pipeline is designed to manually execute the following Jobs:

1. **Addition**: Executes the SMF and AMF node addition tasks using an Ansible playbook and returns the output of commands executed for the addition of network element.
2. **Deletion**: Executes the SMF and AMF node deletion tasks using an Ansible playbook and returns the output of commands executed for the decommissioning of network element.

The pipeline ensures that all necessary dependencies are installed before executing the tasks and provides artifacts for later use.

1. **Prerequisites**

Before setting up the pipeline, ensure the following prerequisites are met:

1. GitLab Lab instance is accessible. In case if we move to production then production instance should be accessible
2. GitLab runner with the tag nfvrunner is configured and running. If nfvrunner is not shared runner or nfvrunner as tag name is different then, that runner should have been defined proper tag and should be in running state.
3. Ansible along with Kubernetes, python package and libraries should have been enabled in one of ansible image from gitlab container registry / Tools. This image will be used for all ENM related playbooks for the execution of Addition/Deletion jobs

**Manual Job Execution**

**Triggering the Pipeline**

1. Navigate to GitLab project using [**https://gitlab01.nfvdev.teluslabs.net/cicd-nfvso/team\_vegas\_automation/pcc\_nf\_automation/-/pipelines/new**](https://gitlab01.nfvdev.teluslabs.net/cicd-nfvso/team_vegas_automation/pcc_nf_automation/-/pipelines/new) **# This is lab instance, you should get production instance url along with project name**
2. Select the ACTION as ADDITION/DELETION by default it will be “ADDITION”

A screenshot of a computer

Description automatically generated

1. Select the NODE either SMF/AMF, by default it is selected as SMF.

A screenshot of a computer

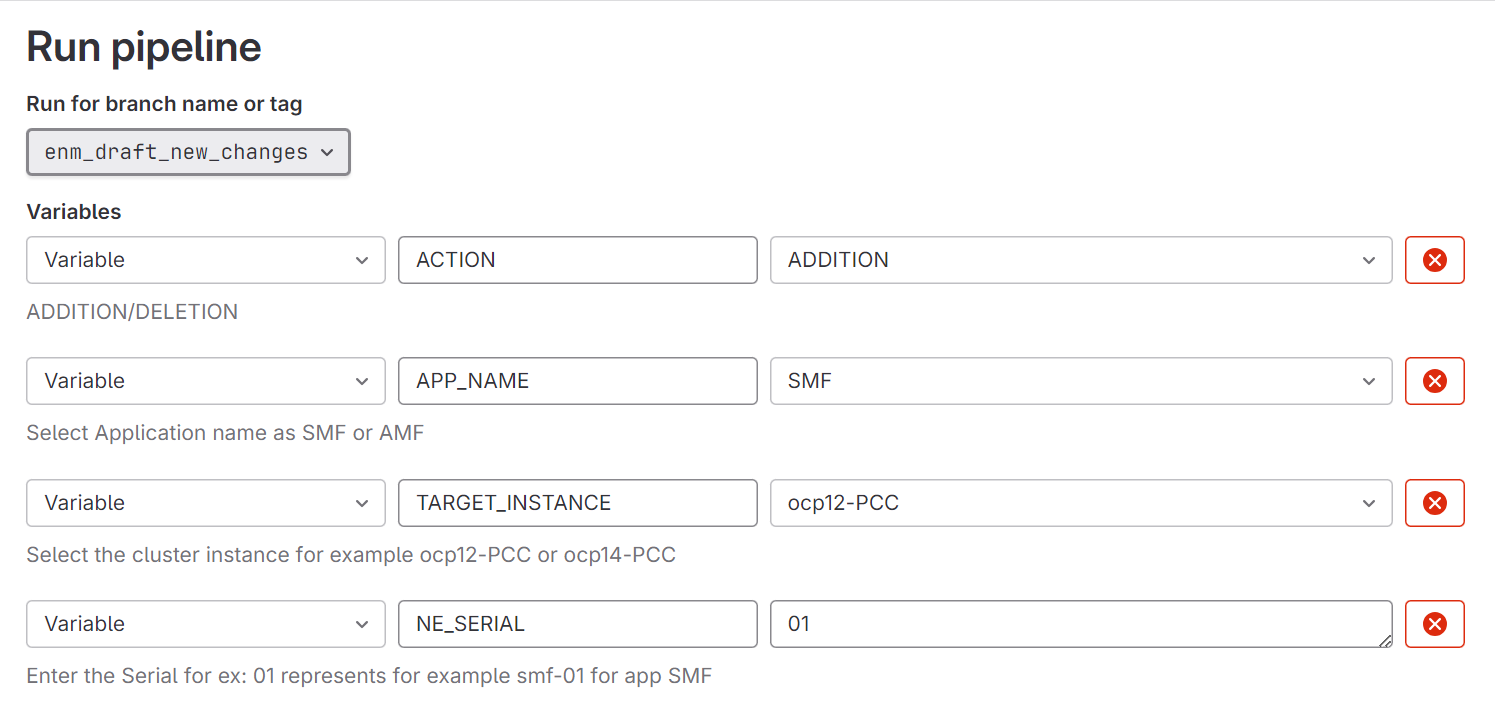
Description automatically generated

1. Select the TARGET\_INSTANCE

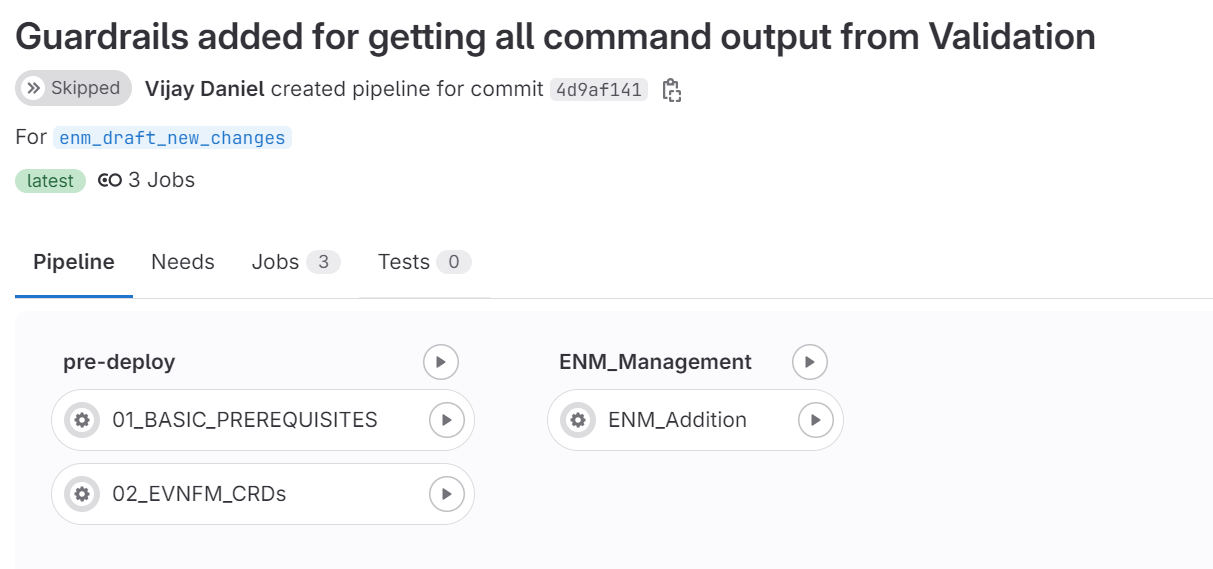
A screenshot of a computer

Description automatically generated

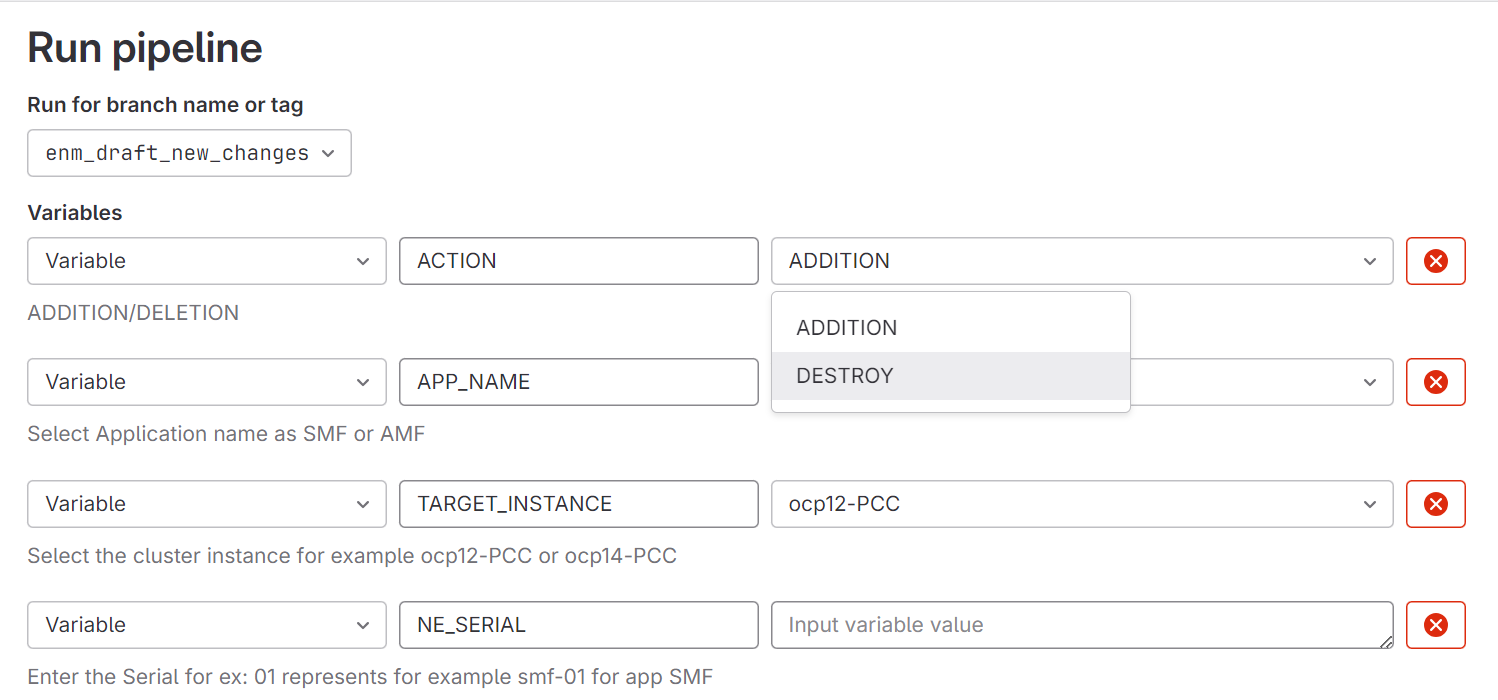
1. Type the desired network element serial ie: NE\_SERIAL 01 or 02 or 03 and so on to process individual network element. Based on below it is referring **srbhong-smf-01** network element

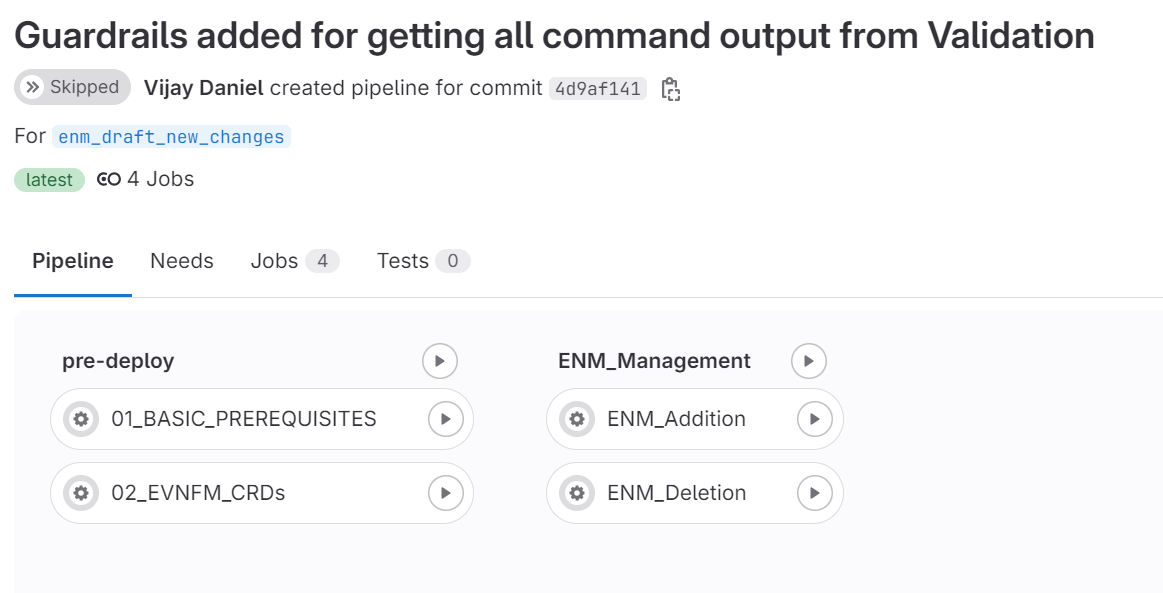


1. Once everything is set then click on Run pipeline and you will land to next page with stages. Click on ENM\_Management -> ENM\_Addition if you want to perform Addition and click start button.

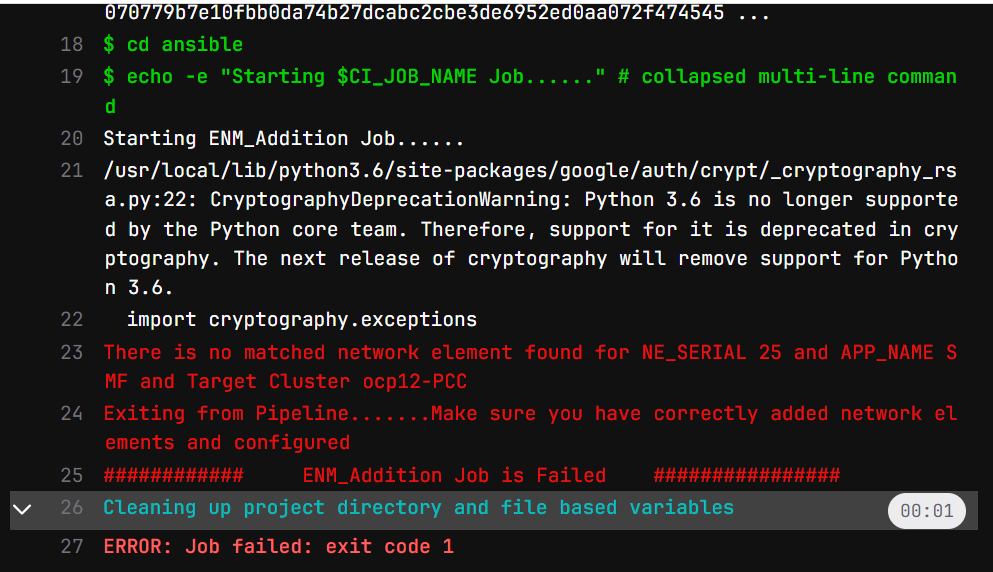


1. What if you would have selected ACTION as DESTROY, you will land to following page.





1. What if you would have entered wrong NE\_SERIAL. Here I am trying to onboard srbhong-smf-25 for SMF Application from ocp12 TARGET CLUSTER. Showing pipeline console output for Addition

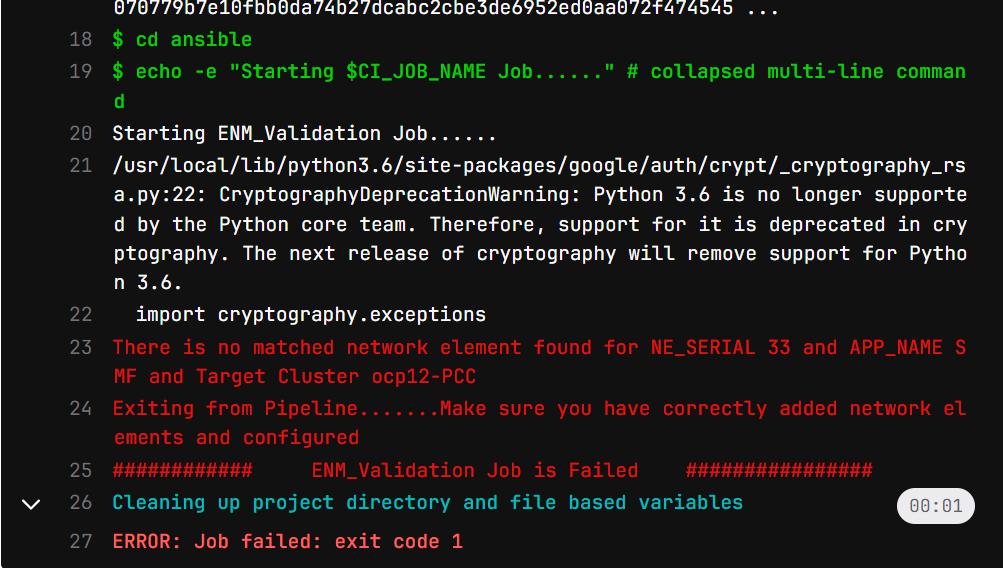


Here, 25 is the NE\_SERIAL I provided from gitlab UI with APP\_NAME=SMF, for TARGET\_INSTANCE=ocp12 it did not find srbhong-smf-25 network element from ocp12-PCC.yml file. This is the file which holds specific network element reference of it. So simply we cannot add any network element or even delete from ENM.

***ocp12-PCC.yml***

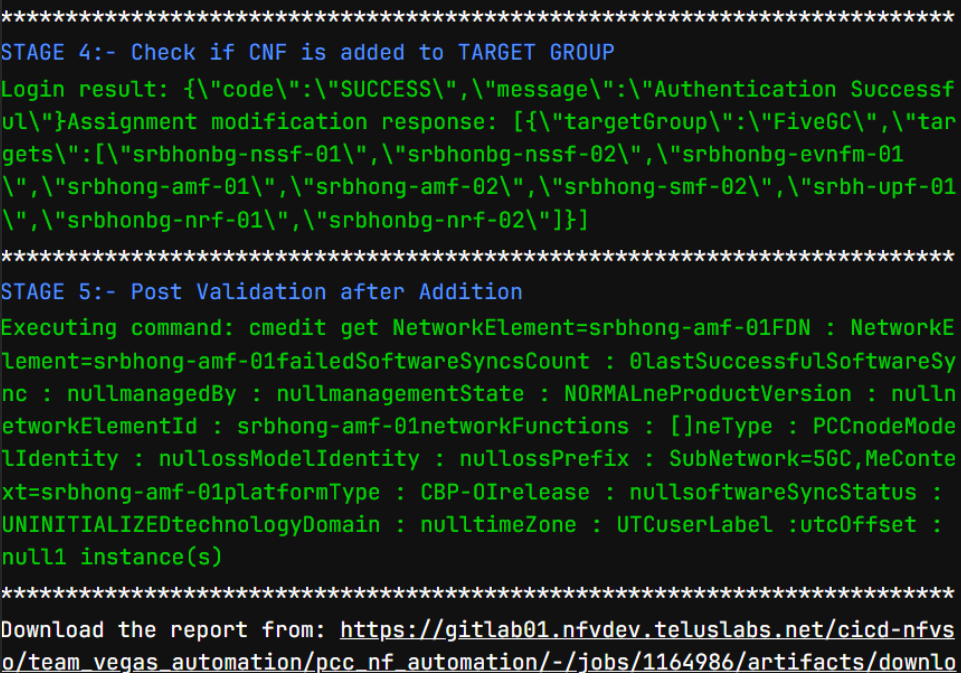
***applications:  
 - name: SMF  
 release\_namespace: srbhsmf01  
 context: /api-system:serviceaccount:default:dzs-saocp12-nfvdev-tlabs-ca:6443/  
 prerequisites:  
 create\_namespace: srbhsmf01  
 namespace\_node\_selector: nodetype=5g-amf-smf  
 label\_nodes:   
 - nodes: wb01  
 labels:  
 pcc-sm-pod: controller  
 nodetype: 5gc-otwaonxh-ctl01-all  
 - nodes: wb02, wb03  
 labels:  
 pcc-sm-pod: non-controller  
 nodetype: 5gc-otwaonxh-ctl01-all  
 ENM\_integration:  
 network\_element: srbhong-smf-01  
 app\_name: smf  
 namespace: srbhsmf01***

***Failure Scenario: Picture-1***

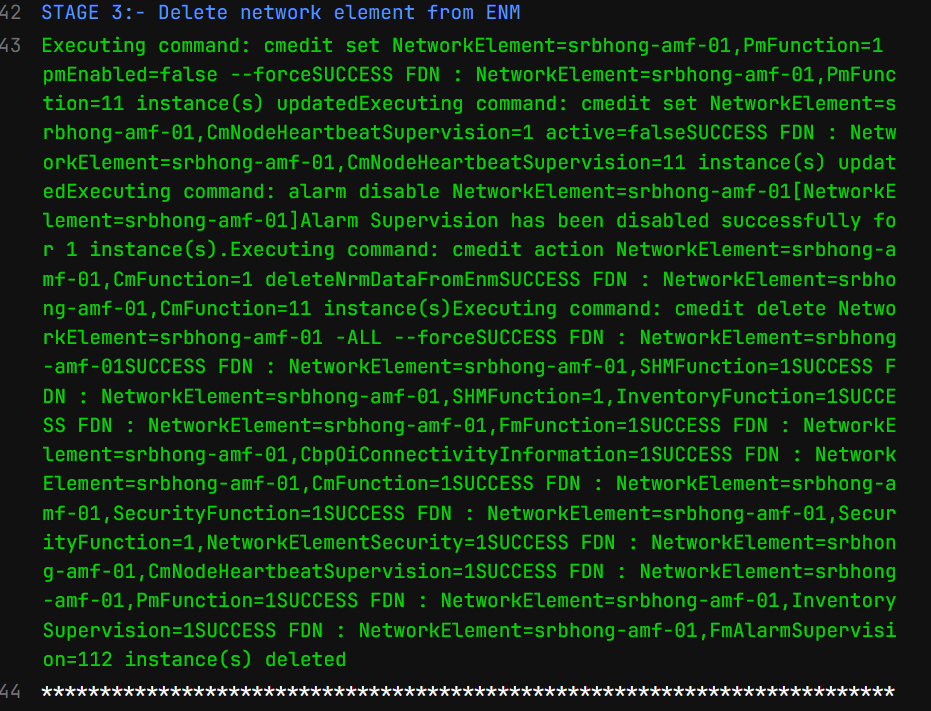


**Success Scenarios: -**

**For Addition: -**

****

**For Deletion:-**

****

**Prerequisites for Variables: -**

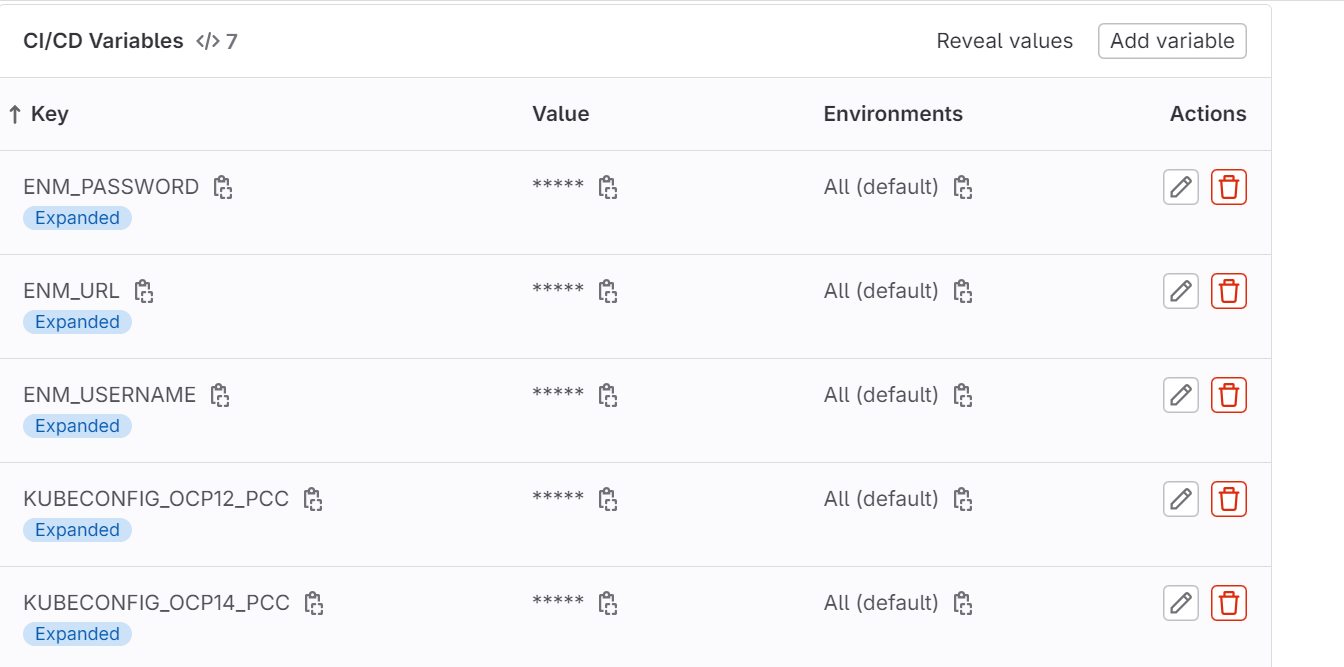
Make sure following variables should get created for ENM\_URL, ENM\_USERNAME, ENM\_PASSWORD,TARGETGROUP,smf\_password,amf\_password. Follow best practices from Appendix A section as well.

KUBECONFIG\_OCP12\_PCC & KUBECONFIG\_OCP14\_PCC these are created based on lab environment.

You can replace **OCP12, OCP14** with your production instances cluster naming conventions.

KUBECONFIG\_OCP12\_PCC variables holds the kubeconfig file contents of srbhong-smf-01 network element with dz service content. Since smf01, amf01 deployed on ocp12 cluster we pasted the contents of one of network\_element kubeconfig file to authentication. You can also follow the same while you migrate to prod instances. Similarly ocp14 cluster smf02, amf02 network elements are deployed. So we pasted one of network\_element in that variable section. Follow the same approach while you migrate to prod environments.

***CI/CD Variables: Picture-2***

****

**Understanding of Pipeline Execution steps: -**

**Overview**

This document provides an overview of the CI/CD pipeline used for performing ENM Management Addition and Deletion of 5G SMF/AMF network instances. The pipeline includes two stages and jobs designed to handle prerequisites, Addition, Deletion. Here’s what you need to know:

**Pipeline Configuration**

**1. Docker Image**

* **Image Used:** registry.nfvdev.teluslabs.net/nfvi/tools/ansible-openshift-helm-collection-ubi:4.10.50
  + This Docker image contains the necessary tools for Ansible and Helm operations.
  + **While moving to production make sure image contains ansible, Kubernetes client libraries and python3 installed**

**2. Pipeline Variables**

* **GIT\_SSL\_NO\_VERIFY:** "1" – Disables SSL verification for Git.
* **ANSIBLE\_FORCE\_COLOR:** "True" – Forces Ansible to use colored output.
* **ACTION:** Defines the action to perform: ADDITION, or DESTROY (default: ADDITION). # **These three actions remain same when pipeline codebase moved to production**
* **APP\_NAME:** Application name: SMF or AMF (default: SMF). **# APP Name can be changed if any new app will be integrated**
* **TARGET\_INSTANCE:** Kubernetes cluster instance: ocp12-PCC or ocp14-PCC (default: ocp12-PCC). **# Since LAB instance cluster name ocp12 and ocp14 so TARGET\_INSTANCE variable is created based on that and KUBECONFIG\_ocp12\_PCC, KUBECONFIG\_ocp14\_PCC is created in CI/CD variable sections to hold respective cluster network elements kubeconfig file content. The contents will be modified or copied based on production network elements. Refer Section Prerequisites for Variables**
* **NE\_SERIAL:** Serial number for the network element. **# This represents the individual network element series. For example, if it is given from gitlab UI as 32 for the APP SMF for the target instance ocp12-PCC then it represents srbhong-smf-32.** Make sure it exists in TARGET\_INSTANCE.yml file. **Refer Section Prerequisites for Variables**
* **ROLE\_DIR:** Directory path where Ansible roles are located for ENM Jobs ${CI\_PROJECT\_DIR}/ansible/roles/enm\_management.

***Pipeline Flow: -***

**1. Pre-deploy Stage**

* **01\_BASIC\_PREREQUISITES**
  + **Purpose:** Set up prerequisites for the deployment.
  + **Script:**
    - Switches to the ansible directory.
    - Configures Kubernetes access using a base64 encoded configuration.
    - Runs the configure\_prerequisites.yml playbook.
  + **Execution:** Manual trigger.
  + **Tag:** nfvrunner
* **02\_EVNFM\_CRDs**
  + **Purpose:** Configure Custom Resource Definitions (CRDs) for ENM management.
  + **Script:**
    - Switches to the ansible directory.
    - Configures Kubernetes access.
    - Runs the configure\_crds.yml playbook.
  + **Execution:** Manual trigger.
  + **Tag:** nfvrunner

**2. ENM Management Stage**

* **ENM\_Addition**
  + **Purpose:** Add a new network element to the system.
  + **Script:**
    - Retrieves IP address using a Python script and saves to ip\_address.env.
    - Exports environment variables from ip\_address.env.
    - Runs the enm\_addition.yml playbook with the retrieved IP address.
  + **Artifacts:** report\_output.html – The output report from the job.
  + **Execution:** Manual trigger.
  + **Tag:** nfvrunner
* **ENM\_Deletion**
  + **Purpose:** Remove a network element from the system.
  + **Script:**
    - Retrieves IP address using a Python script and saves to ip\_address.env.
    - Exports environment variables from ip\_address.env.
    - Runs the enm\_destroy.yml playbook with the retrieved IP address.
  + **Rules:** Executes only if ACTION is set to DESTROY.
  + **Artifacts:** report\_output.html – The output report from the job.
  + **Artifacts:** ValidationResults.txt – The output report of commands before deletion.
  + **Execution:** Manual trigger.
  + **Tag:** nfvrunner

**User Actions**

1. **Trigger Jobs Manually:** Jobs in the pipeline are set to be triggered manually. Ensure to select the correct action ( ***ADDITION, DESTROY )*** before running the pipeline.
2. **Provide Necessary Variables:** Ensure that the below pipeline variables

**ACTION**

***APP\_NAME***

***TARGET\_INSTANCE***

***NE\_SERIAL***

are set correctly according to the requirements. [ ADDITION/DELETION]

1. **Review Artifacts:** After job execution, review the ***report\_output.html*** artifacts to check the results and check ValidationResults.txt file for backup of command outputs. Review if any errors encountered during the process.
2. **Kubernetes Access:** Ensure Kubernetes configurations are correctly set up for accessing the target clusters.

**Troubleshooting**

* **Failed Jobs:** Check the job logs for errors. Common issues might include configuration errors or missing environment variables. Refer ***Failure Scenario: Picture-1***
* **Environment Variables:** Ensure that ip\_address.env contains the correct IP address and network element information. Make sure you always trying valid NE\_SERIAL for the Application along with TARGET CLUSTER information.
* **CI/CD Variables:** You need to properly copy paste the kubeconfig file contents for the KUBECONFIG\_TARGET\_INSTANCE [ Here TARGET\_INSTANCE referred to ocp12 or ocp14]. **Refer Prerequisites for Variables**

**Procedure for Onboarding for a New APP with new TARGET CLUSTER in production**

* 1. Copy the code base and move to development/production gitlab environment to feature branch other than main branch and perform initial commit.
  2. As of now, though the code is generic, and any new APP can be onboarded. You will only need to identify below
     1. TARGET INSTANCE or TARGET CLUSTER [ currently ocp12 & ocp14 have been identified as our TARGET Clusters in lab environment and network elements smf-01, amf-01 are running on ocp12 cluster and smf-02, amf-02 are running on ocp14 ].
     2. So, we created ocp14-PCC.yml and ocp14.yml to differentiate. So that when user specifies from gitlab UI, it will pick corresponding file and pickup network elements based on APP Name [ SMF or AMF based on currently available application from lab environment ]

For PROD, identify your TARGET CLUSTERS. Based on number of TARGET CLUSTERS create TARGET INSTANCE.yml file.

For example, if you identified in future with below example.

* + - * ***PCRF Application*** is running on TARGET CLUSTER called ***ABC*** Cluster and its network instances **somesite-*pcrf-01/*somesite-*pcrf-04/*somesite-*pcrf-06***.
      * ***NRF Application*** is running on ***BAC*** cluster with network instances **somesite-nrf-02/somesite-nrf-06/somesite-nrf-08** and if have any other cluster which is holding another application for that also you need to create ***TARGET\_INSTANCE.yml*** file
      * You will create ***ABC.yml***, ***BAC.yml***and you will paste respective network elements configuration as below.
      * For ***PCRF APP*** with its network elements and for NRF APP with its network elements which is running on same cluster called ***ABC.yml***.

---  
  
**instance\_name: ABC.yml  
  
*# Provided at pipeline kickoff to indicate which applications should be installed by name  
# Example APP\_NAME = PCRF  
# For now only one may be provided at a time*application\_to\_install: "{{lookup('env', 'APP\_NAME') | default([])}}"  
  
*# The details of each application, which may have many or one helm chart*applications:  
 - name: PCRF  
 release\_namespace: pcrf01 namespace # Paste original namespace  
 context: : copy paste context information # Paste original context  
 prerequisites:  
 create\_namespace: pcrf01 namespace  
 namespace\_node\_selector: nodetype=5g-pcrf-nrf  
 label\_nodes:   
 - nodes: wb01  
 labels:  
 pcc-sm-pod: controller  
 nodetype: 5gc-otwaonxh-ctl01-all  
 - nodes: wb02, wb03  
 labels:  
 pcc-sm-pod: non-controller  
 nodetype: 5gc-otwaonxh-ctl01-all  
 ENM\_integration:  
 network\_element: srbhong-pcrf-01 # Paste original network element  
 app\_name: pcrf  
 namespace: pcrf01 namespace # Paste original namespace  
  
  
 - name: NRF  
 release\_namespace: nrfnamespace # Paste original namespace  
 context: copy paste context information # Paste original context  
 prerequisites:  
 create\_namespace:  
 namespace\_node\_selector:  
 label\_nodes:  
 - nodes: wb01  
 labels:  
 pcc-sm-pod: controller  
 nodetype: 5gc-otwaonxh-ctl01-all  
 - nodes: wb02, wb03  
 labels:  
 pcc-sm-pod: non-controller  
 nodetype: 5gc-otwaonxh-ctl01-all  
 ENM\_integration:  
 network\_element: srbhong-nrf-01 # Paste original network element  
 app\_name: nrf  
 namespace: nrfnamespace # Paste original namespace**

* + - * Make sure these cluster target instance.yml file must be stored under ${CI\_PROJECT\_DIR}/ansible/inventory/host\_vars directory. Don’t change the path or don’t store in different directory.
      * Likewise, you must create for any other cluster.yml as above if you want to perform ENM Jobs for those clusters
      * After above step finished, Refer section **Prerequisites for Variables**

**Override below variables values from .gitlab-ci.yml as well**

APP\_NAME:  
 description: "Select Application name as SMF or AMF"  
 options:  
 - SMF # you can change to PCRF  
 - AMF # you can change to NRF  
 value: "SMF" *# Default value can be PCRF*TARGET\_INSTANCE:  
 description: "Select the cluster instance for example ocp12-PCC or ocp14-PCC"  
 options:  
 - ABC # Change this according to new cluster instance for new APP  
 - BAC # Change this according to new cluster instance for new APP  
 value: "ABC" *# Default value can any new TARGET CLUSTER*

**Restricting ansible task to look for only -smf- or -amf- patterns [ Additional Guardrails added only for Destroy]**

*In the* ***${CI\_PROJECT\_DIR}/ansible/enm\_destroy.yml****, we have put restriction not to look for any other network elements other than* ***-smf- or -amf-*** *patterns. You will have to change for your new app.*

- name: Check if network element is either SMF or AMF and print success [Only added for Deletion]  
 debug:  
 msg: "The network element '{{ network\_element }}' matches the required pattern (.\*-smf-.\* or .\*-amf-.\*)."  
 when: network\_element is match('.\*-smf-.\*') or network\_element is match('.\*-amf-.\*')  
  
- name: Fail if network element does not match SMF or AMF [It will abort the deletion job here itself if not matched]  
 fail:  
 msg: "The network element '{{ network\_element }}' does not match the required patterns **(.\*-smf-.\* or .\*-amf-.\***). Aborting playbook execution."  
 when: network\_element is not match('**.\*-smf-.\***') and network\_element is not match(**'.\*-amf-.\*'**)

After setting above prerequisites, add your local changes and push to feature branch and do merge request. Review the code and resolve any conflicts if appears then finally push to main branch.

**DATA FLOW & SEQUENCE DIAGRAM:-**

A diagram of a diagram

Description automatically generated

A white sheet of paper with black lines

Description automatically generated

**Steps for ADDITION Action**

1. Start:
   * The process begins when the User interacts with the GitLab UI.
2. User selects ACTION (ADDITION):
   * The user chooses "ADDITION" as the action to be performed.
3. User selects APP NAME (SMF/AMF):
   * The user is prompted to select the application name, which could be SMF or AMF, depending on the type of network element they are working with.
4. User selects TARGET INSTANCE:
   * The user selects the target instance where the network element is to be added.
5. User selects NE SERIAL:
   * The user selects the NE SERIAL, which identifies the specific network element (e.g., 01, 02, 03).
6. User clicks Run Pipeline:
   * After all selections are made, the user triggers the pipeline to start the process.
7. Run get\_ip\_address.py:
   * The pipeline executes the get\_ip\_address.py script.
   * This script retrieves the necessary environment variables (KUBECONFIG and APP\_NAME).
   * The script fetches network\_elements and namespace from the target instance configuration.
8. Retrieve IP Address from Kubernetes API:
   * The script uses the Kubernetes API to retrieve the IP address of the service corresponding to the selected network element.
   * The retrieved IP address is stored in the pipeline.
9. Is IP\_ADDRESS retrieved?:
   * The pipeline checks whether the IP address was successfully retrieved.
   * If not, the pipeline exits; if yes, the process continues.
10. Check if IP belongs to *-smf-* or *-amf-*?:
    * The pipeline verifies if the retrieved IP address corresponds to the expected patterns (\*-smf-\* or \*-amf-\*).
    * If the IP address does not match, the pipeline exits; otherwise, it continues.
11. Validate if Network Element is managed by EVNFM:
    * The pipeline checks if the network element is managed by EVNFM.
    * If it is managed by EVNFM, the pipeline exits; otherwise, it proceeds.
12. Deploy & Validate:
    * The pipeline deploys the network element and validates the deployment to ensure it was successful.
    * The deployment command's output is stored.
13. Retrieve Output from Automation:
    * The pipeline retrieves the output of the deployment and validation process.
14. Provide Option to Download Report:
    * The pipeline generates a report of the actions performed and provides the user with a link to download the report.
15. End:
    * The pipeline process ends after all actions are completed, and the report is made available to the user.

**Steps for DELETION Action**

1. Start:
   * The process begins similarly, with the User interacting with the GitLab UI.
2. User selects ACTION (DELETION):
   * The user chooses "DELETION" as the action to be performed.
3. User selects APP NAME (SMF/AMF):
   * The user is prompted to select the application name (SMF or AMF) for the network element to be deleted.
4. User selects TARGET INSTANCE:
   * The user selects the target instance from which the network element is to be deleted.
5. User selects NE SERIAL:
   * The user selects the NE SERIAL corresponding to the network element to be deleted.
6. User clicks Run Pipeline:
   * The user initiates the pipeline to begin the deletion process.
7. Run get\_ip\_address.py:
   * The pipeline runs the get\_ip\_address.py script, which retrieves the necessary environment variables (KUBECONFIG and APP\_NAME).
   * The script fetches network\_elements and namespace from the target instance configuration.
8. Retrieve IP Address from Kubernetes API:
   * The script uses the Kubernetes API to retrieve the IP address of the network element to be deleted.
   * The retrieved IP address is stored in the pipeline.
9. Is IP\_ADDRESS retrieved?:
   * The pipeline checks if the IP address was successfully retrieved.
   * If not, the pipeline exits; if yes, the process continues.
10. Check if IP belongs to *-smf-* or *-amf-*?:
    * The pipeline verifies whether the retrieved IP address matches the expected patterns (\*-smf-\* or \*-amf-\*).
    * If the IP address does not match, the pipeline exits; otherwise, it continues.
11. Validate if Network Element is managed by EVNFM:
    * The pipeline checks if the network element is managed by EVNFM.
    * If it is managed by EVNFM, the pipeline exits; otherwise, it continues.
12. Backup Network Element Configuration:
    * Before proceeding with the deletion, the pipeline takes a backup of the network element configuration using cmedit get ${network\_element} NetworkElement,\*.
    * The backup confirmation is stored in the pipeline.
13. Delete & Validate:
    * The pipeline deletes the network element and validates the deletion to ensure it was successful.
    * The deletion command's output is stored.
14. Retrieve Output from Automation:
    * The pipeline retrieves the output of the deletion and validation process.
15. Provide Option to Download Report:
    * The pipeline generates a report of the actions performed and provides the user with a link to download the report.
16. End:
    * The pipeline process ends after all actions are completed, and the report is made available to the user.

**Summary of Key Differences:**

* Backup Step: Only present in the DELETION process, where the network element configuration is backed up before deletion.
* Deployment vs. Deletion: The core action differs depending on whether the user selects ADDITION or DELETION, with respective deployment or deletion of network elements.
* Validation Checks: Common to both actions, including checks for IP retrieval, pattern matching, EVNFM management, and whether the element is already added or deleted.

**Junit Tests:-**

|  |  |
| --- | --- |
| ***Test Case*** | ***Description*** |
| Right Network Element Validation | This test checks whether a network element matching the given  NE\_SERIAL, APP\_NAME, and TARGET\_INSTANCE is found.  If no match is found, the pipeline exits with a failure. |
| Check IP Address File Check | This test verifies if the ip\_address.env file exists. If the file is missing,  the pipeline reports an error and exits. |
| Check IP Address Assignment | This test ensures that the IP\_ADDRESS  and NETWORK\_ELEMENT variables are set  in the ip\_address.env file. If either is missing, the test fails. |
| Check IP Address Retrieval | This test validates that the IP address was successfully retrieved  from the ip\_address.env file for the network element. |
| Check Ansible Playbook Execution | This test verifies the successful execution of the Ansible playbook.  If any errors or failures occur during execution,  it captures the error details and logs the failure. |
| Stage 1: EVNFM Team Management | This test verifies if the EVNFM team is managing the  elements by processing and checking output  from the log for "Stage 1" and logging the result. |
| Stage 2: Check if Network Element Already Deleted ( For Deletion Action ) | This test checks whether the network element  has already been deleted.  It validates the absence of network elements  in the log, marking the test as passed or failed. |
| Stage 2: Check if Network Element Already Added ( For Addition Action ) | This test checks whether the network element  has already been added.  It validates the presence of network elements in the log,  marking the test as passed or failed. |
| Stage 3: Delete Network Element from ENM ( For Deletion Action ) | This test confirms that  the network element was deleted from ENM by verifying  the playbook log for Stage 3. |
| Stage 3: Add Network Element to ENM ( For Addition Action ) | This test confirms that the network element  was successfully added to ENM by verifying the playbook  log for Stage 3. |
| Stage 4: Post-Modification Checks and TargetGroup Added | This test validates the successful update or modification of network elements by verifying that the log contains post-modification details and ensuring no errors occurred. For Addition Action, it also validates whether CNF has been added to FiveGC targetgroup. |
| Stage 5: Final Network Element Validation | This test ensures that the network element is functioning correctly after being added or modified, by performing a final check in the log and confirming the pipeline's success. |

**Appendix A: Best Practices**

**Overview:**  
This appendix outlines best practices and guardrails for managing ENM jobs, with a focus on security, maintainability, efficiency, observability, upgradability, scalability, and backup strategies.

**A.1 Security Considerations**

**A.1.1 Sensitive Data Handling:**

* Ensure that KUBECONFIG data, containing sensitive cluster access information, is securely stored and encrypted, especially in production environments.
* Limit permissions associated with KUBECONFIG files to only what's necessary, avoiding overly privileged service accounts.

**A.1.2 Environment Variables Security:**

* Mask sensitive information in environment variables within the CI/CD pipeline logs. If masking isn’t possible, ensure it’s not displayed in logs and that it’s decoded if stored and referenced. If used only once, delete it from the pipeline runner after use.
* Utilize GitLab’s protected variables feature to restrict access, ensuring availability only in specific branches or pipelines.

**A.1.3 Project-Level Audit Logs:**

* **Purpose:** Monitor actions within specific projects, such as changes to CI/CD variables and pipeline executions.
* **Steps:**
  1. Navigate to the project to audit.
  2. Go to **Settings > Audit Events** to view logs of project-related activities.

**A.1.4 Monitoring and Filtering Audit Logs:**

* **Filter by Date:** Narrow logs to specific timeframes.
* **Filter by User:** Audit actions by specific users.
* **Filter by Event Type:** Focus on specific actions like CI/CD variable changes or project setting modifications.

**A.1.5 Reviewing Key Audit Events:**

* **CI/CD Variables Changes:** Monitor events related to creation, modification, or deletion of CI/CD variables.
* **Pipeline Executions:** Track who triggered pipelines, especially on protected branches.
* **Access Changes:** Review changes to user roles and permissions, particularly administrative roles.

**A.1.6 Responding to Unauthorized Access or Changes:**

* Investigate suspicious activities and, if necessary, revoke access or restore original settings.
* Implement additional security measures like MFA or stricter RBAC based on audit findings.

**A.1.7 Exporting Audit Logs:**

* **Purpose:** For external analysis or archival.
* **Steps:** Use the **Export** button in the audit logs interface to download logs in CSV format.

**A.2 Maintainability**

**A.2.1 Modularization of Ansible Playbooks:**

* Ensure playbooks are modular and reusable. Common tasks (e.g., KUBECONFIG setup, IP address retrieval) should be separated into roles or tasks reusable across multiple playbooks.

**A.2.2 Dynamic Inventory Management:**

* Use dynamic inventories in Ansible to fetch the latest cluster details, avoiding hardcoding in inventory files.

**A.2.3 Centralized Configuration:**

* Store and version control configuration files (e.g., TARGET\_INSTANCE.yml) centrally. Maintain consistency and documentation across different clusters.

**A.2.4 Git Version Control Commands:**

* Example commands for managing configuration changes:

git checkout <your dev branch>

git tag -a v1.0 -m "Initial configuration version for my dev branch"

git push origin v1.0

git tag

git ls-remote --tags origin

git reset --hard <tag-name>

**A.3 Efficiency and Performance**

**A.3.1 Parallel Execution:**

* Parallelize independent tasks to speed up pipeline execution using Ansible’s strategy: free.

**A.3.2 Cache Management:**

* Implement caching for repeated tasks or data fetching operations to improve pipeline speed.

**A.3.3 Conditional Execution:**

* Use conditional execution to skip unnecessary stages or jobs, improving pipeline efficiency.

**A.4 Observability and Logging**

**A.4.1 Detailed Logging:**

* Ensure detailed logging in Ansible playbooks for better troubleshooting, especially for tasks involving NETWORK\_ELEMENT, IP\_ADDRESS, APP\_NAME, and TARGET\_INSTANCE.

**A.4.2 Monitoring and Alerts:**

* Set up monitoring and alerts on critical pipeline stages to be promptly notified of failures or unusual behavior.

**A.5 Upgradability and Scalability**

**A.5.1 Versioning:**

* Version Ansible playbooks and Docker images used in the pipeline to facilitate rollback in case of failure during upgrades.

**A.5.2 Scalability:**

* Use templating in GitLab CI/CD to dynamically handle different clusters or environments. Ensure the pipeline scales appropriately as more clusters or applications are added.

**A.6 Backup**

**A.6.1 Backup Strategy:**

* Regularly back up local changes and store them in a separate path or folder to ensure data recovery and continuity.

**Appendix B: GitLab Pipeline Guardrails**

**B1. Validating EVNFM Management for SMF/AMF Nodes**

* **Description**:
  + Before any action (ADDITION or DELETION) is performed on network elements, the pipeline validates whether the network element, whether it's an SMF or AMF node (e.g., smf-01, amf-01), is managed by the EVNFM (Ericsson Virtual Network Function Manager) team.
* **Purpose**:
  + Ensures that only network elements managed by Telus are modified, preventing unintended changes to elements managed by EVNFM.

**B2. No Action if Managed by EVNFM**

* **Description**:
  + If the pipeline determines that the network element is managed by EVNFM, it will not proceed with any actions, such as ADDITION or DELETION, and will exit early.
* **Purpose**:
  + Protects network elements managed by EVNFM from unauthorized modifications, maintaining the integrity of the network.

**B3. Actions Allowed Only if Not Managed by EVNFM**

* **Description**:
  + If the EVNFM management query returns 0 instances, meaning the network element is not managed by EVNFM, the pipeline allows Telus to proceed with the ADDITION or DELETION of that element.
* **Purpose**:
  + Ensures that Telus only modifies network elements under its management, preventing accidental changes to elements managed by other teams.

**B4. Backup Before Deletion for DELETION Action**

* **Description**:
  + Before performing a DELETION action, the pipeline runs a backup command: cmedit get ${network\_element} NetworkElement,\*. This captures the network element's configuration and state before deletion.
* **Purpose**:
  + Provides a safety net by preserving the current data and configuration of the network element, enabling recovery or analysis if needed.

**B5. Storing Backup Output for User Verification**

* **Description**:
  + The output from the backup command during the DELETION process is stored in separate files for each network element, available for user verification, audit, or historical reference.
* **Purpose**:
  + Ensures there is a traceable record of the network element’s configuration before deletion, facilitating verification and maintaining a history for audits.

**B6. Deletion Based on Matching Patterns Only**

* **Description**:
  + The pipeline proceeds with the deletion of network elements only if the IP address matches the expected patterns for SMF (\*-smf-\*) or AMF (\*-amf-\*). If the IP does not match, the pipeline exits without performing the deletion.
* **Purpose**:
  + Ensures that only the correct network elements are targeted for deletion, reducing the risk of accidental deletions and maintaining network integrity.

**B7. IP Address Validation for ADDITION Action**

* **Description**:
  + Before proceeding with an ADDITION action, the pipeline checks that the command execution includes a valid IP address. If the IP is missing or invalid, the pipeline exits without performing the addition.
* **Purpose**:
  + Ensures that the ADDITION process only continues when a valid IP address is present, preventing misconfigurations and potential network issues.

This appendix provides a comprehensive overview of the guardrails in place within the GitLab pipeline, explaining their function and purpose in ensuring safe and controlled modifications to network elements.