OCTOBER 2023

NSX CERTIFICATES MANAGEMENT COOKBOOK

Software Version 4.1.1

NSX Certificates Management Cookbook

Table of Contents

1 Introduction

1.1 Scope of the document

Starting with NSX version 4.1 many more certificates are visible in NSX. Those certificates have always been present on the platform, even in previous versions, but it was not possible to lifecycle them. This document will help the reader in understanding the purpose of all the certificates part of the NSX platform and will provide examples covering common certificate related tasks an NSX administrator may be tackling while administering NSX.

To make these example reproducible, they are presented in the form of bash scripts. We opted to use bash for maximum portability. The scripts mainly use curl to perform API calls to the NSX API and use the jq to process the returned JSON data structures. You must install jq on your system to run the sample scripts. You can use your system package manager (i.e., apt or homebrew)

The scripts are provided for educational purpose only. You should perform your validations before leveraging them on production systems.

1.2 Additional Resources

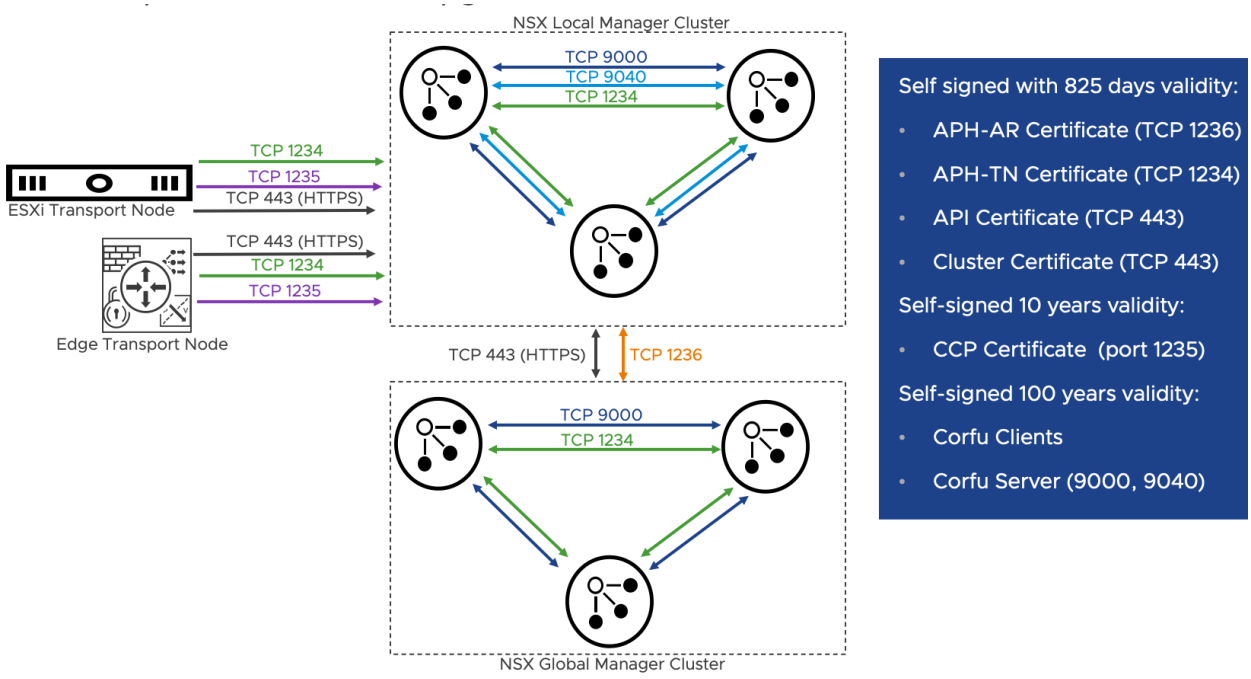
The NSX official documentation is the most authoritative source of information for any NSX related topic:

HTTPS://DOCS.VMWARE.COM/EN/VMWARE-NSX/4.1/ADMINISTRATION/GUID-C47D8468-6A91-42EA-B8AC-63743F86C1E4.HTML

2 Overview of the NSX Certificates

NSX is full-stack L2-L7 networking and security solution. The NSX platform architecture is distributed in nature and depends on the secure communication between its components. In version 4.1.x and later all communication channels are encrypted via TLS 1.2 and they are authenticated via X.509 certificates. RSA and Elliptic Curve certificates are supported. In a fresh installation all the certificates are RSA self-signed certificates, but they can be replaced with other self-signed or CA signed certificates by an NSX user with Enterprise Admin role.

The diagram below summarizes the TLS flows in an NSX deployment with and without federation. Subsequent sections will provide a brief description of the different types of certificates protecting those flows.



Replacing a certificate for a specific service is a two-step process:

- Generating or uploading a new certificate to the NSX Truststore (This step can be performed via the UI or the API)
- Apply the certificate to the certificate profile, which maps to the service that will use the certificate (This step must be performed via the API)

2.1 API/UI Certificates

The NSX API/UI certificates are presented to the end user when interacting with the UI or API. In NSX Manager cluster, by default each of the three nodes has an individual self-signed certificate that is presented to the users who interactively explicitly with that node. In addition, a separate self-sign certificate is shared among the 3 UA nodes and presented to the users who connect to the redundant VIP. The VIP resides on a single node at that time, and moves in case of a failure of the node where it is running.

API/UI are usually the only certificates we may need to replace in a fresh deployment. The reason is that these certificates are exposed to the end users. Leveraging certificates trusted by the clients because signed by a trusted public or private Certificate Authority is a general best practice. All other NSX certificates are used for the internal platform communications and self-signed certificates are usually enough for this purpose unless specific compliance considerations apply to the deployment.

Users interact with the UI via a browser. Browsers implement additional security checks on the certificates presented by a web site (in this case, the web server is the NSX Manager UI), on top of verifying that the certificate is valid and has been signed by a trusted CA. Browsers check that the FQDN in the URL entered by the user matches one of the values in the SAN (Subject Alternative Names) extension in the certificate. For this reason, we need to ensure that the CSR (Certificate Signing Request) generated for this purpose includes the SANs with all the FQDNs and IP addresses users may use to connect to NSX Manager.

It is possible to share a single certificate for the API/UI service on each node plus the VIP rather than four individual certificates. Wildcard certificates are supported, but a better approach is using a regular certificate with EKU set to “SERVER” and the FQDNs and IPs of the cluster nodes and VIP in the SANs.

2.2 APH-TN, CCP, APH-AR Certificates

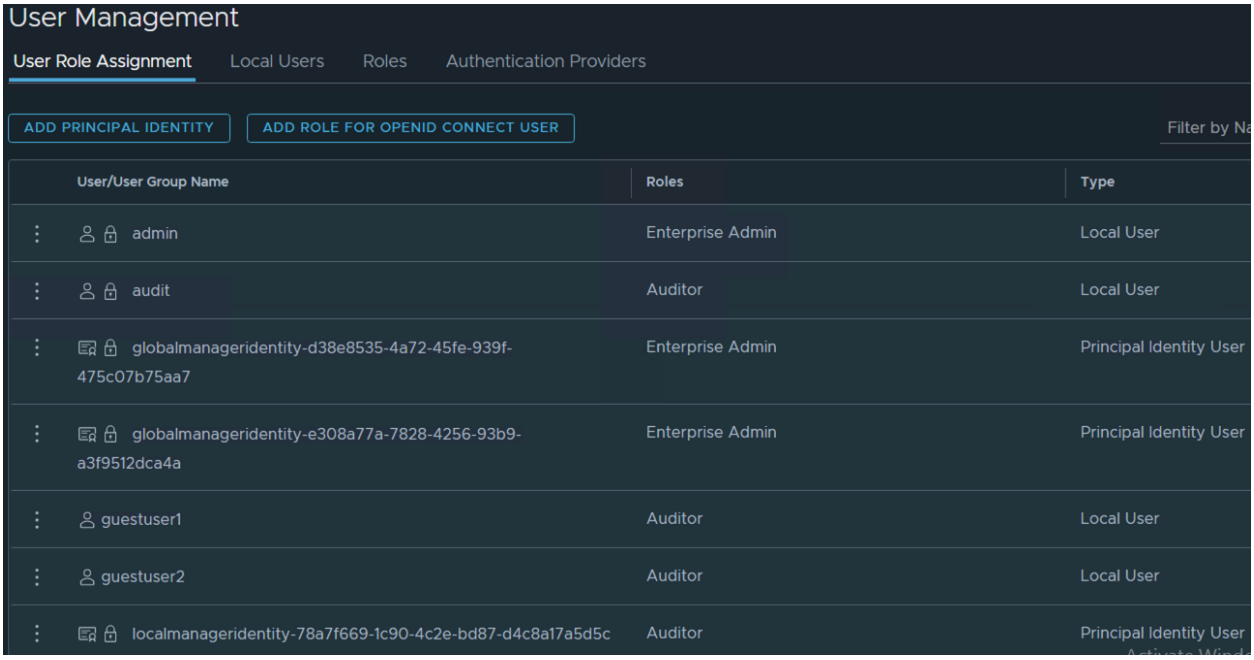
The APH-TN certificate is presented to transport nodes when connecting to the NSX UA nodes on port 1234 for management plane connectivity. The CCP certificate is presented to transport nodes when connecting to NSX UA nodes on port 1235 for control plane connectivity. The APH-

AR certificate is exposed on the NSX UA on port 1236 by the async-replicator service to synchronize control plane information across different locations in a federation deployment. The async-replicator is not used in non-federated deployments. There is no need to replace these certificates in a fresh deployment. When replacing them for compliance reasons or because they are approaching the end of their validity, the replacement operation can be performed in-band. The new certificate is passed to the clients on the exiting secure channel before switching to a new secure TLS connection based on the new certificate. CLI commands on the host allows to accept a new certificate if the in-band replacement process fails.

2.3 Local manager and Global Manager Certificates

These certificates are only used in federation deployments. They are associated with Principal Identities defined on all the LM and GM clusters part of the federation deployment. The active and standby GM cluster will have a PI with Enterprise Admin privileges defined on the LM clusters. These PIs will use the GLOBAL_MANAGER certificate of the GM cluster to authenticate on the LMs. The GMs uses these credentials to push configurations to the LMs. All the LM clusters part of the federation deployment will have a PI with Audit privileges defined on all the other LM and GM clusters. The LMs uses these credentials to retrieve status and statistics information from the other clusters.

In a deployment with active and standby GM clusters and two locations, the automatically created Principal identities will be as follow. On the LM:



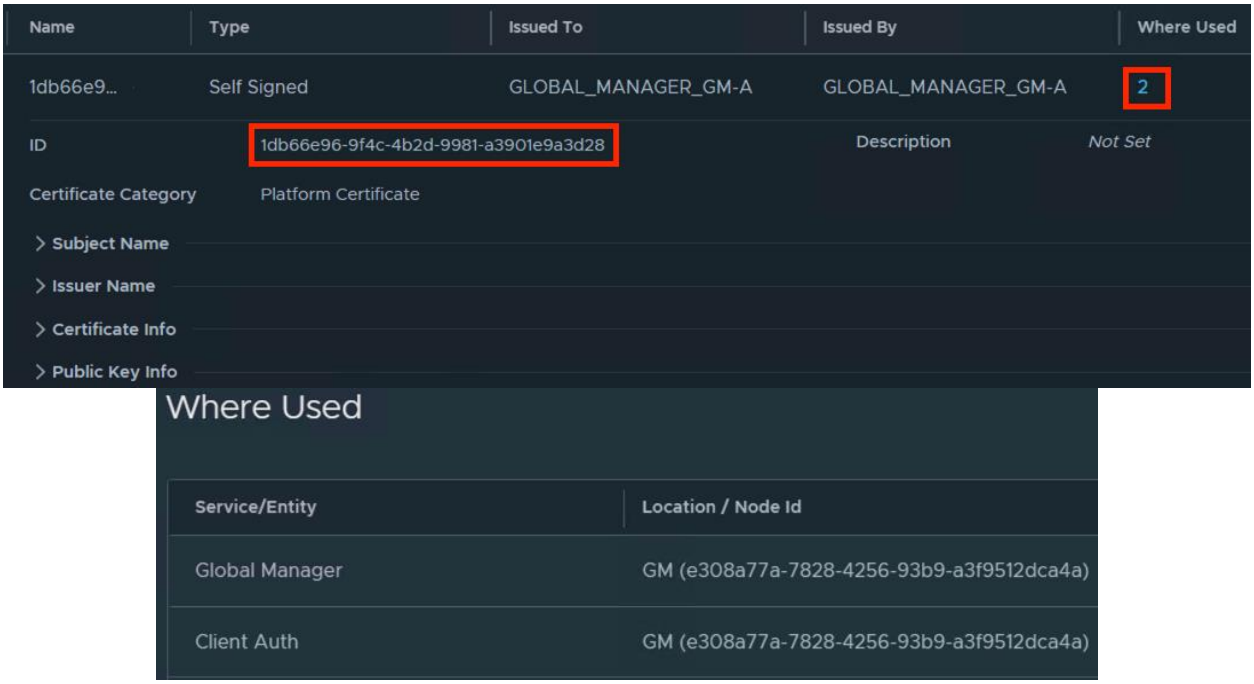
From the figure we can see three principal identities: globalmanageridentity-d38e8535-4a72-45fe-939f-475c07b75aa7, globalmanageridentity-e308a77a-7828-4256-93b9-a3f9512dca4a, and localmanageridentity-78a7f669-1c90-4c2e-bd87-d4c8a17a5d5c. Unfortunately, it is not possible to understand what is the certificate associated with those PIs from the UI. We can do it from the API. In the example below, we use jq to parse the json response and retrieve only the name and certificate_id fields for each PI.

```
NSX_MANAGER='192.168.110.17'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI="/api/v1/trust-management/principal-identities"
METHOD='GET'

curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" | jq '.results[] | {name,
certificate_id}'

{
  "name": "GlobalManagerIdentity-e308a77a-7828-4256-93b9-
a3f9512dca4a",
  "certificate_id": "1db66e96-9f4c-4b2d-9981-a3901e9a3d28"
```

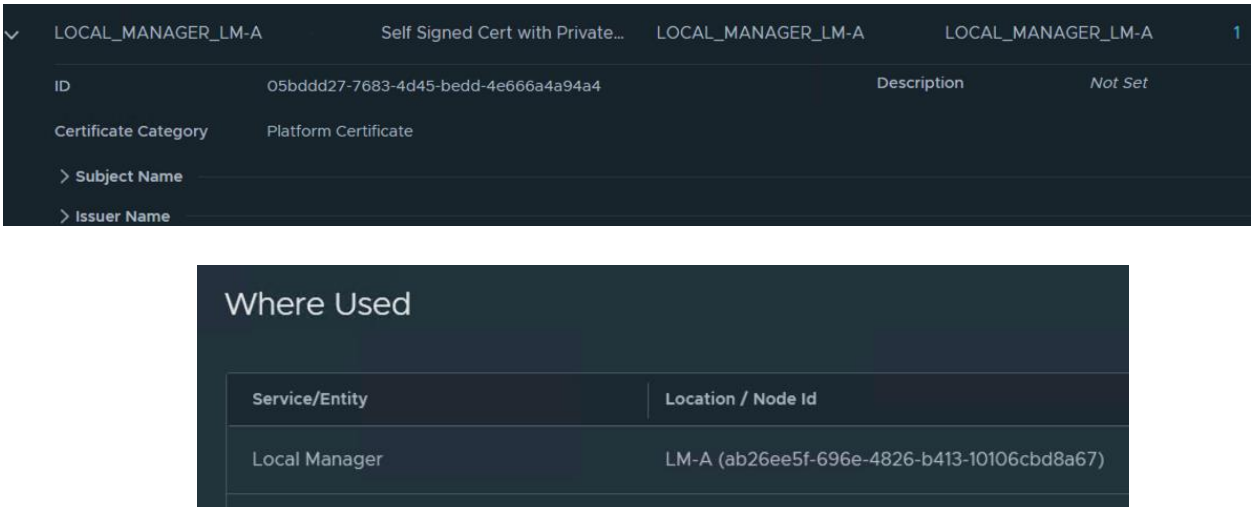
```
}
{
  "name": "LocalManagerIdentity-78a7f669-1c90-4c2e-bd87-d4c8a17a5d5c",
  "certificate_id": "889be17b-a051-4e80-862b-8f817310137b"
}
{
  "name": "GlobalManagerIdentity-d38e8535-4a72-45fe-939f-
475c07b75aa7",
  "certificate_id": "276bf82a-b9bb-42c2-87e1-36d63530c045"
}
```


Looking at the certificates page we can match the certificates that are used by the three PIs for client authentication. It is important to understand that these PI certificates does not need to be installed or life-cycled on this local manager. They have been automatically imported from the other GM or LM clusters. If we need to replace them, we need to do it on the originating cluster. Then automatically, they will be pushed to the other LM or GM clusters. We can identify the certificates coming form other clusters based on their used_by (in the API) or Where Used (in the UI) property. Certificates originating from other clusters will have two services associated to the them: Local Manager or Global Manager, and Client Auth.

	Name	Type	Issued To	Issued By	Where Used
>	1db66e9...	Self Signed	GLOBAL_MANAGER_GM-A	GLOBAL_MANAGER_GM-A	2
>	276bf82a...	Self Signed	GLOBAL_MANAGER_GM-B	GLOBAL_MANAGER_GM-B	2
>	889be17...	Self Signed	LOCAL_MANAGER_LM-B	LOCAL_MANAGER_LM-B	2



On the other hand, the Local or Global Manager certificate of the cluster we are browsing will have a single service associated to it (Local or Global manager) and no Client_Auth service because no PI is associated with it. If we connect to another cluster we will see the same certificate associated to a PI and to the Client_Auth service. When we replace this certificate on LM-A, it will be automatically pushed to the other NSX clusters part of the federation deployment.



2.4 CBM certificates

The NSX Unified Appliance (UA) uses CorfuDB for persistence. CorfuDB is a strongly consistent object store built over a shared log. CorfuDB is based ON PEER REVIEWED research and is an open source project on GITHUB.

CorfuDB operates in a clustered configuration with Corfu servers running on each NSX Unified Appliance (UA). These CorfuDB servers employ mutual TLS 1.2 for two crucial purposes: authenticating other UA services connecting to them and encrypting all communication with these services.

To establish trust, a service connecting to the CorfuDB server must have its certificate already known to the CorfuDB server. Likewise, for a service to authenticate the CorfuDB server, it needs to possess the CorfuDB server's certificate in advance. In the context of a UA cluster, which comprises three Unified Appliances, each UA hosts a Corfu server that establishes a TLS 1.2 connection with every other Corfu server in the cluster. Consequently, a service is required to communicate with all CorfuDB servers within a UA cluster.

Furthermore, each UA features a Cluster Boot Manager (CBM) service responsible for ensuring that each CorfuDB server possesses the certificates of all other services within that UA. Importantly, CBM does not engage in communication with UAs outside of its cluster.

CBM allows for the substitution of any of its managed certificates with customer-supplied certificates to address the following potential concerns:

- The default certificates, generated on the UA during the bootstrap process (referred to as Default Certificates), suffer from low entropy, violating DRG.2 requirements, and fall short of EAL4 standards.
- The default certificates lack the use of Elliptic Curve cryptography.
- Default certificates have an extended validity period, which may not be in line with industry practices.

- In the event of a CBM managed certificate being compromised, it becomes unusable.
- CBM managed certificates that are nearing expiration or have already expired must be replaced.

Upon the initial boot-up of a UA, it undergoes an initialization process where it is configured as a single-node cluster. The CBM's initialization script, executed with root privileges, generate an RSA key pair and a corresponding self-signed X.509 certificate for each service running on the UA. Subsequently, when the CBM server is launched, it identifies the various services operating on the UA and establishes TrustStores to facilitate secure TLS mutual authentication between them.

In total, there are 14 services hosted on the UA, namely: cbm, api, ccp, csm, proton, gm, ar, vmc, monitoring, idps-reporting, cm-inventory, messaging-manager, upgrade-coordinator, and site-manager. Consequently, the CBM generates a total of 14 certificates and key pairs, one for each service.

During the formation of UA clusters, a new UA joining the cluster acquires the certificates of all pre-existing UAs within the cluster. These public certificates of the new UA are disseminated to all existing UAs within the cluster. Since these certificates are public in nature, there is no need for specific protection measures during the propagation process.

CBM managed certificates do not support revocation. If a certificate is revoked, an alarm is raised to indicate the system is using a revoked certificate, but the system continues to work. The only way to ensure a revoked CBM managed certificate is not used by the system is to replace it with a valid certificate.

2.5 Transport Nodes Certificates

Transport nodes (ESXi and NSX Edges) use client certificates to perform mutual authentication with the management (APH_TN) and control plane (CCP) services running on the NSX Manager UA. By default, they use self-signed certificates with 825 days validity generated when NSX is first installed on the ESXi or the NSX edge deployed. The TN certificates can be replaced in-band via an API call to the NSX Manager cluster. In case the certificate expires, and the TN is

disconnected from the NSX Management cluster for this reason, it is possible to replace the TN certificate manually.

Transport node certificates are not managed or stored on the NSX Manager cluster. NSX Manager only provides a way to replace them without direct SSH access to the host and will generate an alarm when a TN certificate is approaching expiration or is expired. Key pairs, CSR and certificates for transport nodes should be generated outside of NSX (i.e., via OpenSSL). This is because it is not possible to export the private key of NSX generated certificates or CSRs. The private key of the certificate must be passed in the API call to replace a TN certificate.

2.6 Principal Identity Certificates

Principal Identities (PI) are unique users in NSX who can create an object and ensure that the object can only be modified or deleted by the same identity. Authentication of PIs is only supported via client certificate, authentication is local to NSX Manager (no external Identity Provider is required), and it's possible to assign a predefined RBAC role to the PI. Principal Identities are generally leveraged by third-party applications or cloud management platforms such as Openstack, NCP, Antrea, or vRA to ensure that an admin does not modify the NSX configuration generating a mismatch between their view of the NSX environment and its actual configuration.

We have already a special use case for Principal Identities in Federation deployments, where PIs are associated to the Global_Manager and Local_Manager certificate profiles to provide an authentication mechanism between the different cluster s in the deployment. The GMs uses a PI with Enterprise Admin privileges to push configurations to the LM, those configurations can be seen in the LM UI, but they are protected and can only be modified by the PI who generated them. LMs have instead PI with Audit privileges defined on all other clusters (GMs, and other LMs) to retrieve statistics and resource status. The lifecycle of the certificates associated with such PIs represent an exception and should follow the procedures outline in the dedicated section.

For other use cases (anything other than NSX Federation), we will follow the process outlined in section **3.5**. It will consist in generating a key pair and associated certificate outside of NSX,

then creating a PI user associated to the certificate. It is important to notice that the private key is not imported with the certificate, NSX manager will only have access to the public certificate, that will be used to verify that the client is in possession of the corresponding private key. Certificates for PIs cannot be created on NSX because in that case NSX will store the private key, and the private key cannot be exported.

Note about PI for the integration of Antrea with NSX:

Today, to register an Antrea cluster to NSX, we use a principal identity user. If the cert of the PI user expires, the Antrea cluster registration to NSX breaks. Just changing the PI cert in NSX is not sufficient. A user has to un-register the cluster, update the cert in the Antrea NSX Interworking config file and then re-register the cluster.

2.7 Summary of the certificates in a Local Manager NSX Cluster

UI Name	Issue to/Issue By	Validity	API certificate profile name/ **service-type**	UI Used By Field	Cluster or Node certificate	EKU
APH-AR certificate for Node <uuid node 1>	VMware-NSX-ApplProxyHub	825 Days	APH-AR/**APH**	APH	Node	SERVER
APH-TN certificate for node <uuid node 1>	VMware-NSX-ApplProxyHub	825 Days	APH-TN/**APH_TN**	policy.certificate.usedby.resourcetype.APH_TN	Node	SERVER
API certificate for node 1	<UA_hostname>	825 Days	API/**API**	API	Node	SERVER
AR-Corfu Client certificate for node <uuid node 1>	ar	100 years	AR-Corfu Client/**CBM_AR**	CBM AR	Node	CLIENT
CCP certificate for node 1	nsx-controller	10 Years	CCP/**CCP**	policy.certificate.usedby.resourcetype.CCP	Node	SERVER
CCP-Corfu Client certificate for node <uuid node 1>	ccp	100 years	CCP-Corfu Client/**CBM_CCP**	CBM CCP	Node	CLIENT
Cluster certificate for site <uuid site>	-cluster	825 days	Cluster/**MGMT_CLUSTER**	Management Cluster	Cluster	SERVER
Cluster Manager-Corfu Client certificate for	cluster-manager	100 years	Cluster Manager-Corfu Client/**CBM_CLUSTER_MANAGER**	CBM Cluster Manager	Node	CLIENT

node <uuid node 1>						
CM Inventory-Corfu Client certificate for node <uuid node 1>	cm-inventory	100 years	CM Inventory-Corfu Client/ **CBM_CM_INVENTORY**	CBM CM Inventory	Node	CLIENT
Corfu Server certificate for node <uuid node 1>	corfu	100 years	Corfu Server/**CBM_CORFU**	CBM Corfu	Node	SERVER
IDPS reporting-Corfu Client certificate for node <uuid node 1>	idps-reporting	100 years	IDPS reporting-Corfu Client/**CBM_IDPS_REPORTING**	CBM IDPS reporting	Node	CLIENT
LocalManager	local-manager	825 Days	Local Manager Principal Identity/**LOCAL_MANAGER**	Local Manager	Cluster	CLIENT
Messaging Manager-Corfu Client certificate for node <uuid node 1>	messaging-manager	100 years	Messaging Manager-Corfu Client/**CBM_MESSAGING_MANAGER**	CBM Messaging Manager	Node	CLIENT
Monitoring-Corfu Client certificate for node <uuid node 1>	monitoring	100 years	Monitoring-Corfu Client/**CBM_MONITORING**	CBM Monitoring	Node	CLIENT
MP-Corfu Client certificate for node <uuid node 1>	mp	100 years	MP-Corfu Client/**CBM_MP**	CBM MP	Node	CLIENT
Site Manager-Corfu Client certificate for node <uuid node 1>	site-manager	100 years	Site Manager-Corfu Client/**CBM_SITE_MANAGER**	CBM Site Manager	Node	CLIENT
Upgrade Coordinator-Corfu Client certificate for node <uuid node 1>	upgrade-coordinator	100 years	Upgrade Coordinator-Corfu Client/**CBM_UPGRADE_COORDINATOR**	CBM Upgrade Coordinator	Node	CLIENT

The table above shows a total of 17 certificates for a single node NSX manager cluster. Of those 17 certificates 15 are specific to the node, and 2 are cluster certificates. The implication is that a three node NSX manager cluster will have a total of 47 certificates, 15 for each node, plus two cluster certificates (LOCAL_MANAGER and MGMT_CLUSTER). 15 x 3 + 2 = 47. All the 47

certificates will be visible in the UI of any of the NSX Manager nodes in the cluster, as they are replicated and stored across the cluster nodes. The list of certificates that can be replaced in an NSX Local Manager cluster may vary between versions. Only certificates with a corresponding certificate profile can be replaced. An authoritative list of supported certificate profiles can be retrieved via the following API call.

```
NSX_MANAGER='nsxmgr-99a'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI='/api/v1/trust-management/certificate-profiles'
METHOD='GET'


curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    | jq .
```

```
{
  "results": [
    {
      "profile_name": "Cluster",
      "service_type": "MGMT_CLUSTER",
      "extended_key_usage": [
        "SERVER"
      ],
      "cluster_certificate": true,
      "unique_use": false,
      "node_type": [
        "global-manager",
        "nsx-manager",
        "nsx-shared"
      ]
    },
    {
      "profile_name": "CSM-Corfu Client",
      "service_type": "CBM_CSM",
      "extended_key_usage": [
        "CLIENT"
```

```
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "Cluster Manager-Corfu Client",
    "service_type": "CBM_CLUSTER_MANAGER",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "Message Bus Client for K8S Platform",
    "service_type": "K8S_MSG_CLIENT",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": true,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "Site Proxy-Client-Corfu Client",
    "service_type": "CBM_SITE_PROXY_CLIENT",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
```

```
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "GM-Corfu Client",
    "service_type": "CBM_GM",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "Corfu Server",
    "service_type": "CBM_CORFU",
    "extended_key_usage": [
      "SERVER"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "MP-Corfu Client",
    "service_type": "CBM_MP",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
```

```
},
{
  "profile_name": "CCP",
  "service_type": "CCP",
  "extended_key_usage": [
    "SERVER"
  ],
  "cluster_certificate": false,
  "unique_use": false,
  "node_type": [
    "nsx-manager",
    "nsx-shared"
  ]
},
{
  "profile_name": "APH-TN",
  "service_type": "APH_TN",
  "extended_key_usage": [
    "SERVER"
  ],
  "cluster_certificate": false,
  "unique_use": false,
  "node_type": [
    "global-manager",
    "nsx-manager",
    "nsx-shared"
  ]
},
{
  "profile_name": "Local Manager Principal Identity",
  "service_type": "LOCAL_MANAGER",
  "extended_key_usage": [
    "CLIENT"
  ],
  "cluster_certificate": true,
  "unique_use": false,
  "node_type": [
    "nsx-manager",
    "nsx-shared"
  ]
},
{
  "profile_name": "Site Manager-Corfu Client",
  "service_type": "CBM_SITE_MANAGER",
  "extended_key_usage": [
    "CLIENT"
```

```
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "CCP-Corfu Client",
    "service_type": "CBM_CCP",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "Monitoring-Corfu Client",
    "service_type": "CBM_MONITORING",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "APH-AR",
    "service_type": "APH",
    "extended_key_usage": [
      "SERVER"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
```

```
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "AR-Corfu Client",
    "service_type": "CBM_AR",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "Messaging Manager-Corfu Client",
    "service_type": "CBM_MESSAGING_MANAGER",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "API",
    "service_type": "API",
    "extended_key_usage": [
      "SERVER"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
```

```
    },
    {
      "profile_name": "IDPS reporting-Corfu Client",
      "service_type": "CBM_IDPS_REPORTING",
      "extended_key_usage": [
        "CLIENT"
      ],
      "cluster_certificate": false,
      "unique_use": false,
      "node_type": [
        "global-manager",
        "nsx-manager",
        "nsx-shared"
      ]
    },
    {
      "profile_name": "Upgrade Coordinator-Corfu Client",
      "service_type": "CBM_UPGRADE_COORDINATOR",
      "extended_key_usage": [
        "CLIENT"
      ],
      "cluster_certificate": false,
      "unique_use": false,
      "node_type": [
        "global-manager",
        "nsx-manager",
        "nsx-shared"
      ]
    },
    {
      "profile_name": "CM Inventory-Corfu Client",
      "service_type": "CBM_CM_INVENTORY",
      "extended_key_usage": [
        "CLIENT"
      ],
      "cluster_certificate": false,
      "unique_use": false,
      "node_type": [
        "global-manager",
        "nsx-manager",
        "nsx-shared"
      ]
    }
  ]
}
```

2.8 Summary of the certificates in a Global Manager NSX cluster

UI Name	Issue to/Issue By	Validity	API certificate profile name/ **service-type**	UI Used By Field	Cluster or Node certificate	EKU
APH-AR certificate for Node <uuid node 1>	VMware-NSX-ApplProxyHub	825 Days	APH-AR/**APH**	APH	Node	SERVER
APH-TN certificate for node <uuid node 1>	VMware-NSX-ApplProxyHub	825 Days	APH-TN/**APH_TN**	policy.certificate.usedby.resourcetype.APH_TN	Node	SERVER
API certificate for node <uuid node 1>	<UA_hostname>	825 Days	API/**API**	API	Node	SERVER
AR-Corfu Client certificate for node <uuid node 1>	ar	100 years	AR-Corfu Client/**CBM_AR**	CBM AR	Node	CLIENT
CCP-Corfu Client certificate for node <uuid node 1>	ccp	100 years	CCP-Corfu Client/**CBM_CCP**	CBM CCP	Node	CLIENT
Cluster certificate for site <uuid site>	-cluster	825 days	Cluster/**MGMT_CLUSTER**	Management Cluster	Cluster	SERVER
Cluster Manager-Corfu Client certificate for node <uuid node 1>	cluster-manager	100 years	Cluster Manager-Corfu Client/**CBM_CLUSTER_MANAGER**	CBM Cluster Manager	Node	CLIENT
CM Inventory-Corfu Client certificate for node <uuid node 1>	cm-inventory	100 years	CM Inventory-Corfu Client/ **CBM_CM_INVENTORY**	CBM CM Inventory	Node	CLIENT
Corfu Server certificate for node <uuid node 1>	corfu	100 years	Corfu Server/**CBM_CORFU**	CBM Corfu	Node	SERVER
IDPS reporting-Corfu Client certificate for node <uuid node 1>	idps-reporting	100 years	IDPS reporting-Corfu Client/**CBM_IDPS_REPORTING**	CBM IDPS reporting	Node	CLIENT
GlobalManager	local-manager	825 Days	Gloabll Manager Principal	Global Manager	Cluster	CLIENT

			Identity/**GLOBAL_MANAGER**			
GM-Corfu Client certificate for node <uuid node 1>	gm	100 years	GM-Corfu Client/**CBM_GM**	CBM GM	Node	CLIENT
Messaging Manager-Corfu Client certificate for node <uuid node 1>	messaging-manager	100 years	Messaging Manager-Corfu Client/**CBM_MESSAGING_MANAGER**	CBM Messaging Manager	Node	CLIENT
Monitoring-Corfu Client certificate for node <uuid node 1>	monitoring	100 years	Monitoring-Corfu Client/**CBM_MONITORING**	CBM Monitoring	Node	CLIENT
MP-Corfu Client certificate for node <uuid node 1>	mp	100 years	MP-Corfu Client/**CBM_MP**	CBM MP	Node	CLIENT
Site Manager-Corfu Client certificate for node <uuid node 1>	site-manager	100 years	Site Manager-Corfu Client/**CBM_SITE_MANAGER**	CBM Site Manager	Node	CLIENT
Upgrade Coordinator-Corfu Client certificate for node <uuid node 1>	upgrade-coordinator	100 years	Upgrade Coordinator-Corfu Client/**CBM_UPGRADE_COORDINATOR**	CBM Upgrade Coordinator	Node	CLIENT

The table below shows a total of 17 certificates for a single node NSX manager cluster. Of those 17 certificates 15 are specific to the node, and 2 are cluster certificates. The implication is that a three node NSX global manager cluster will have a total of 47 certificates, 15 for each node, plus two cluster certificates (GLOBAL_MANAGER and MGMT_CLUSTER). 15 x 3 + 2 = 47. All the 47 certificates will be visible in the UI of any of the NSX Manager nodes in the cluster, as they are replicated and stored across the cluster nodes. The list of certificates that can be replaced in an NSX Global Manager cluster may vary between versions. Only certificates with a corresponding certificate profile can be replaced. An authoritative list of supported certificate profiles can be retrieved via the following API call.

```
NSX_MANAGER='192.168.110.18'
```

```
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI='/api/v1/trust-management/certificate-profiles'
METHOD='GET'


curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    | jq .

{
  "results": [
    {
      "profile_name": "Cluster",
      "service_type": "MGMT_CLUSTER",
      "extended_key_usage": [
        "SERVER"
      ],
      "cluster_certificate": true,
      "unique_use": false,
      "node_type": [
        "global-manager",
        "nsx-manager",
        "nsx-shared"
      ]
    },
    {
      "profile_name": "CSM-Corfu Client",
      "service_type": "CBM_CSM",
      "extended_key_usage": [
        "CLIENT"
      ],
      "cluster_certificate": false,
      "unique_use": false,
      "node_type": [
        "global-manager",
        "nsx-manager",
        "nsx-shared"
      ]
    },
    {
      "profile_name": "Cluster Manager-Corfu Client",
      "service_type": "CBM_CLUSTER_MANAGER",
      "extended_key_usage": [
```

```
      "CLIENT"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "Message Bus Client for K8S Platform",
    "service_type": "K8S_MSG_CLIENT",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": true,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "Site Proxy-Client-Corfu Client",
    "service_type": "CBM_SITE_PROXY_CLIENT",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "GM-Corfu Client",
    "service_type": "CBM_GM",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": false,
    "unique_use": false,
```

```
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "Corfu Server",
    "service_type": "CBM_CORFU",
    "extended_key_usage": [
      "SERVER"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "MP-Corfu Client",
    "service_type": "CBM_MP",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "APH-TN",
    "service_type": "APH_TN",
    "extended_key_usage": [
      "SERVER"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
```

```
    ]
  },
  {
    "profile_name": "Site Manager-Corfu Client",
    "service_type": "CBM_SITE_MANAGER",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "CCP-Corfu Client",
    "service_type": "CBM_CCP",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "Monitoring-Corfu Client",
    "service_type": "CBM_MONITORING",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "APH-AR",
```

```
    "service_type": "APH",
    "extended_key_usage": [
      "SERVER"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "AR-Corfu Client",
    "service_type": "CBM_AR",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "Messaging Manager-Corfu Client",
    "service_type": "CBM_MESSAGING_MANAGER",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "API",
    "service_type": "API",
    "extended_key_usage": [
      "SERVER"
    ],
```

```
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "Global Manager Principal Identity",
    "service_type": "GLOBAL_MANAGER",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": true,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "IDPS reporting-Corfu Client",
    "service_type": "CBM_IDPS_REPORTING",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
      "nsx-shared"
    ]
  },
  {
    "profile_name": "Upgrade Coordinator-Corfu Client",
    "service_type": "CBM_UPGRADE_COORDINATOR",
    "extended_key_usage": [
      "CLIENT"
    ],
    "cluster_certificate": false,
    "unique_use": false,
    "node_type": [
      "global-manager",
      "nsx-manager",
```

```
        "nsx-shared"
      ]
    },
    {
      "profile_name": "CM Inventory-Corfu Client",
      "service_type": "CBM_CM_INVENTORY",
      "extended_key_usage": [
        "CLIENT"
      ],
      "cluster_certificate": false,
      "unique_use": false,
      "node_type": [
        "global-manager",
        "nsx-manager",
        "nsx-shared"
      ]
    }
  ]
}
```


2.9 NSX Certificates Best practices

- It is possible to use a single CA signed certificate for the nodes API/UI and VIP services. The certificate should be signed by a public CA, or an enterprise CA trusted by the clients connecting to NSX for management purposes. Include the FQDN, short names, IPs (and any possible way user will connect to the NSX UAs, i.e., CNAMEs) in the SAN property of the certificate. Use a single certificate when an external load balancer is part of the deployment. Wildcard certificates are supported for the Node API and VIP certificates.
- Use descriptive CN field and use the same string in the display_name property of the certificate for Local_Manager and Global_Manager certificates. This will help identify the certificates when they are pushed across clusters in Federation.
- Transport Nodes certificates MUST be unique.
- GM and LM principal identity certificates MUST be unique.
- Corfu server certificates MUST be unique.

- CBM client certificates do not need to be unique.
- Certificates MUST include the EKU property matching their certificate profile.
- Do not replace expired LM and PI certificate by updating the PI itself, use the procedure specific to those certificates.
- Keep all the certificates valid by replacing them with new self-signed or CA certificates before they expire.

2.10Impact of certificate expiration

The NSX platform relies on valid certificates for the internal communication between its components and for external clients' connections. A healthy system has none of its certificate expired. This section outlines known impacts of certificate expiration per certificate type. Additional unexpected or unknown impacts of expired certificates may arise in live systems.

Certificate Type	Certificate Profile	Impact of Expiration	Resolution
API/VIP	API/MGMT_CLUSTER	Low. External clients will receive an SSL error. Upstream consumption may break	Replace the certificates via the normal procedure
Transport Nodes Management and Control Plane (TCP 1234,1235)	APH_TN/CCP	High. Transport Nodes will be disconnected from the NSX cluster	Replace the certificates via the normal procedure. Then SSH to each individual transport node and retrieve the new certificate.
APH-AR (TCP 1236)	APH	High. In federation deployment the sync between GM and LM clusters will not be operational. In non-federation deployment the impact is none.	If GM and LM are still syncing (Sync status is green) use the normal procedure. If the sync is down, the LM must be off-boarded and on-boarded again.
Local and Global Manager Principal Identities	LOCAL_MANAGER/GLOBAL_MANAGER	None	Replace the certificates via the normal procedure
CBM Certificates	CBM_*	High. If the Corfu expiry check is enabled, the cluster will be inoperational. None, if Corfu expiry check is disabled.	Replace the certificates via the normal procedure if the cluster is operational. If the cluster is not operational, disable the Corfu expiry check on the 3 nodes and reboot the UAs, then replace the certs via the normal procedure
Transport Node certificates	N/A	High. TN will not be able to connect to the NSX Manager cluster.	Replace the certificate manually on the transport node (API call to NSX Manager will not work) and push the new certificate to the NSX Manager cluster via the ESXi nsxcli command "push host-certificate".

3 Recipes

3.1 Singleton NSX Manager

A singleton NSX Manager is an NSX Manager cluster comprised by a single NSX Unified Appliance. It is a supported configuration that may be appropriate in small deployments. See the EASY ADOPTION NSX DESIGN guide for sample reference Architecture.

3.1.1 Replacing API/UI Certificates with a self-signed certificate

In order to include the SAN extensions in these user facing certificates we will leverage the CSR extended API which is currently in experimental state. This API does not allow to generate the CSR and create a self-sign certificate in a single call, so after the CSR is created another API call perform the signing and generation of the certificate. The generated certificate will be used for the API service and the VIP UI/API access. Any client connecting to the API/UI will not trust the presented certificate and prompt an error. If this is not acceptable the CSR must be signed by an enterprise or public CA. See section 3.5 for some helper scripts addressing such use case.

```
NSX_MANAGER='nsxmgr-99a'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI='/api/v1/trust-management/csrs-extended'
METHOD='POST'
BODY=$(cat <<END
{
  "display_name": "API/VIP Certificate",
  "subject": {
    "attributes": [
      {
        "key": "CN",
        "value": "nsxmgr-99a"
      },
      {
        "key": "O",
        "value": "VMware"
      },
      {
        "key": "OU",
        "value": "NSBU"
```

```
      },
      {
        "key": "C",
        "value": "US"
      },
      {
        "key": "ST",
        "value": "CA"
      },
      {
        "key": "L",
        "value": "PA"
      }
    ]
  },
  "key_size": "2048",
  "algorithm": "RSA",
  "extensions": {
    "subject_alt_names": {
      "dns_names": [
        "nsxmgr-99a-vip",
        "nsxmgr-99a-vip.nsxdg.eng.vmware.com",
        "nsxmgr-99a",
        "nsxmgr-99a.nsxdg.eng.vmware.com"
      ]
    }
  }
}
END
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")


CSR_ID=$(echo $response | jq -r '.id')


NSX_MANAGER='nsxmgr-99a'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI="/api/v1/trust-
management/csrs/$CSR_ID?action=self_sign&days_valid=825"
```

```
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")


CERT_ID=$(echo $response | jq -r '.id')

#visualize certificate properties
openssl x509 -in <(echo $response | jq -r '.pem_encoded')   -text

#Find node UUID

NSX_MANAGER='nsxmgr-99a'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI="/api/v1/cluster"
METHOD='GET'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")

NODE_UUID=$(echo $response | jq -r '.nodes[].node_uuid')

#Apply certificate

URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=API&node_id=$NODE_UUID"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")

URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=MGMT_CLUSTER"
METHOD='POST'

response=$(curl -k -X $METHOD \
```

```
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")
```

3.1.2 Replacing Local Manager Certificate with a self-signed certificate

This certificate is not used in deployment without federation, but it should be kept valid to avoid alarms.

```
####LM Certificate

NSX_MANAGER='nsxmgr-99a'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI='/api/v1/trust-management/csrs?action=self_sign'
METHOD='POST'
BODY=$(cat <<END
{
  "display_name": "LOCAL_MANAGER_nsxmgr-99a",
  "subject":
  {
    "attributes":
    [
      {"key":"CN","value":"LOCAL_MANAGER_nsxmgr-99a"}
    ]
  },
  "key_size": "2048",
  "algorithm": "RSA",
  "days_valid": 825
}
END
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")


CERT_ID=$(echo $response | jq -r '.id')
```

```
URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=LOCAL_MANAGER"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")
```

3.1.3 Replacing APH-AR, APH-TN and CCP certificates with self-signed certificates

The APH-AR certificate is not used in a deployment without federation, but it should be kept valid to avoid alarms.

```
#Find node UUID

NSX_MANAGER='nsxmgr-99a'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI="/api/v1/cluster"
METHOD='GET'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")

NODE_UUID=$(echo $response | jq -r '.nodes[].node_uuid')

#APH

#Generate certificate

NSX_MANAGER='nsxmgr-99a'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI='/api/v1/trust-management/csrs?action=self_sign'
METHOD='POST'
BODY=$(cat <<END
{
  "display_name": "APH-AR for $NODE_UUID",
  "subject":
  {
```

```
    "attributes":
    [
      {"key":"CN","value":"APH-AR for $NODE_UUID"}
    ]
  },
  "key_size": "2048",
  "algorithm": "RSA",
  "days_valid": 825
}
END
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")


CERT_ID=$(echo $response | jq -r '.id')


#Apply certificate

SERVICE_TYPE="APH"
URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=$SERVICE_TYPE&node_id=$NODE_UUID"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")


#APH_TN

#Generate certificate

NSX_MANAGER='nsxmgr-99a'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI='/api/v1/trust-management/csrs?action=self_sign'
METHOD='POST'
BODY=$(cat <<END
```

```
{
  "display_name": "APH-TN for $NODE_UUID",
  "subject":
  {
    "attributes":
    [
      {"key":"CN","value":"APH-TN for $NODE_UUID"}
    ]
  },
  "key_size": "2048",
  "algorithm": "RSA",
  "days_valid": 825
}
END
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")


CERT_ID=$(echo $response | jq -r '.id')


#Apply certificate

SERVICE_TYPE="APH_TN"
URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=$SERVICE_TYPE&node_id=$NODE_UUID"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")


#CCP

NSX_MANAGER='nsxmgr-99a'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI='/api/v1/trust-management/csrs?action=self_sign'
```

```
METHOD='POST'
BODY=$(cat <<END
{
  "display_name": "CCP for $NODE_UUID",
  "subject":
  {
    "attributes":
    [
      {"key":"CN","value":"CCP for $NODE_UUID"}
    ]
  },
  "key_size": "2048",
  "algorithm": "RSA",
  "days_valid": 825
}
END
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")


CERT_ID=$(echo $response | jq -r '.id')


#Apply certificate

SERVICE_TYPE="CCP"
URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=$SERVICE_TYPE&node_id=$NODE_UUID"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")
```

3.1.4 Replace Corfu Client Certificates

All the corfu client certificates are replaced quickly upon invocation of the replacement API call

except for the CBM_MP certificates, which require multiple services to restart, and the API/UI to become unavailable for 10-15 minutes. For this reason, it is the last one we replace in the script.

```
##CBM Certificates

NSX_MANAGER='nsxmgr-99a'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'


#Find node UUID

URI="/api/v1/cluster"
METHOD='GET'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")

NODE_UUID=$(echo $response | jq -r '.nodes[].node_uuid')


#Replace CBM certificates

CBM_PROFILES=("CBM_AR" "CBM_CCP" "CBM_CLUSTER_MANAGER"
"CBM_CM_INVENTORY" "CBM_IDPS_REPORTING" "CBM_MESSAGING_MANAGER"
"CBM_MONITORING" "CBM_SITE_MANAGER" "CBM_UPGRADE_COORDINATOR"
"CBM_MP")

for i in ${!CBM_PROFILES[@]};
do
sleep 5
CBM_PROFILE=${CBM_PROFILES[$i]}
URI='/api/v1/trust-management/csrs?action=self_sign'
METHOD='POST'
BODY=$(cat <<END
{
  "display_name": "$CBM_PROFILE for node $NODE_UUID",
  "subject":
   {
   "attributes":
   [
     {"key":"CN","value":"$CBM_PROFILE for node $NODE_UUID"}
```

```
    ]
  },
  "key_size": "2048",
  "algorithm": "RSA",
  "days_valid": 825
}
END
)

  response=$(curl -k -X $METHOD \
      "https://$NSX_MANAGER$URI" \
      -u "$NSX_USER:$NSX_PASSWORD" \
      -H "content-type: application/json" \
      -d "$BODY")
  echo "Response for generating the certificate of $CBM_PROFILE:"
  echo $response

  CERT_ID=$(echo $response | jq -r '.id')

  #Apply certificate

  URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=$CBM_PROFILE&node_id=$NODE_UUID"
  METHOD='POST'

  response=$(curl -k -X $METHOD \
      "https://$NSX_MANAGER$URI" \
      -u "$NSX_USER:$NSX_PASSWORD" \
      -H "content-type: application/json")

  echo "Response for applying the certificate of $CBM_PROFILE:"
  echo $response

done
```

3.1.5 Replace Corfu Server Certificate

Replacing the corfu server certificate has minimal impact and risks in singleton NSX deployment. Please refer to the appropriate section if you a replacing it on a 3 nodes UA cluster.

```
#Find node UUID
```

```
NSX_MANAGER='nsxmgr-99a'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI="/api/v1/cluster"
METHOD='GET'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")

NODE_UUID=$(echo $response | jq -r '.nodes[].node_uuid')

#APH

#Generate certificate

NSX_MANAGER='nsxmgr-99a'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI='/api/v1/trust-management/csrs?action=self_sign'
METHOD='POST'
BODY=$(cat <<END
{
  "display_name": "Corfu-server for $NODE_UUID",
  "subject":
  {
    "attributes":
    [
      {"key":"CN","value":"Corfu-server for $NODE_UUID"}
    ]
  },
  "key_size": "2048",
  "algorithm": "RSA",
  "days_valid": 825
}
END
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")
```

```
CERT_ID=$(echo $response | jq -r '.id')


#Apply certificate

SERVICE_TYPE="CBM_CORFU"
URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=$SERVICE_TYPE&node_id=$NODE_UUID"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")
```

3.1.6 Delete Unused certificates.

NSX will block the deletion of any used certificate. This script will collect the UUIDs of the certificates with an empty used_by property, and then it will delete them one by one.

```
NSX_MANAGER='nsxmgr-99a'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI='/api/v1/trust-management/certificates'
METHOD='GET'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")

NOTUSED_CERTIDs=$(echo $response | jq '.results[] | select(.used_by ==
[])' | jq -r '.id')

# Initialize an empty array
NOTUSED_CERTIDs_array=()

# Use a while loop to split the string into an array
while read -r -a temp_array; do
    NOTUSED_CERTIDs_array+=("${temp_array[@]}")
done <<< "$NOTUSED_CERTIDs"
```

```
for i in ${!NOTUSED_CERTIDs_array[@]};
do
  CERT_ID=${NOTUSED_CERTIDs_array[$i]}
  URI="/api/v1/trust-management/certificates/$CERT_ID"
  METHOD='DELETE'

  response=$(curl -k -X $METHOD \
      "https://$NSX_MANAGER$URI" \
      -u "$NSX_USER:$NSX_PASSWORD" \
      -H "content-type: application/json")
done
```

3.2 Three node cluster NSX Manager

The recipes presented in this section are applicable to the most common NSX deployments which include a NSX Manager comprised of three unified appliances and not federated.

3.2.1 Replacing API/UI Certificates with a self-signed certificate

To include the SAN extensions in these user facing certificates we will leverage the CSR extended API which is currently in experimental state. This API does not allow to generate the CSR and create a self-sign certificate in a single call, so after the CSR is created another API call perform the signing and generation of the certificate. The generated certificate will be used for the API service on each node and the VIP UI/API access. Any client connecting to the API/UI will not trust the presented certificate and prompt an error. If this is not acceptable the CSR must be signed by an enterprise or public CA. See section 3.5 for some helper scripts addressing such use case.

```
#API/UI

NSX_MANAGER='192.168.110.17'
NSX_USER='admin'
URI='/api/v1/trust-management/csrs-extended'
METHOD='POST'
BODY=$(cat <<END
{
  "display_name": "API/VIP Certificate",
```

```
"subject": {
  "attributes": [
    {
      "key": "CN",
      "value": "LM-A"
    },
    {
      "key": "O",
      "value": "VMware"
    },
    {
      "key": "OU",
      "value": "NSBU"
    },
    {
      "key": "C",
      "value": "US"
    },
    {
      "key": "ST",
      "value": "CA"
    },
    {
      "key": "L",
      "value": "PA"
    }
  ]
},
"key_size": "2048",
"algorithm": "RSA",
"extensions": {
  "subject_alt_names": {
    "ip_addresses": [
      "192.168.110.11",
      "192.168.110.12",
      "192.168.110.13",
      "192.168.110.17"
    ],
    "dns_names": [
      "lm-01a",
      "lm-02a",
      "lm-03a",
      "lm-vip-a"
    ]
  }
}
```

```
}
END
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")


CSR_ID=$(echo $response | jq -r '.id')


URI="/api/v1/trust-
management/csrs/$CSR_ID?action=self_sign&days_valid=825"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")


CERT_ID=$(echo $response | jq -r '.id')

#visualize certificate properties
openssl x509 -in <(echo $response | jq -r '.pem_encoded')   -text

#Find node UUIDs

URI="/api/v1/cluster"
METHOD='GET'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")

NODE_UUID_01=$(echo $response | jq -r '.nodes[0].node_uuid')
NODE_UUID_02=$(echo $response | jq -r '.nodes[1].node_uuid')
NODE_UUID_03=$(echo $response | jq -r '.nodes[2].node_uuid')



#Apply certificate node 01
```

```
URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=API&node_id=$NODE_UUID_01"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")

#Apply certificate node 02

URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=API&node_id=$NODE_UUID_02"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")


#Apply certificate node 03

URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=API&node_id=$NODE_UUID_03"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")



#Apply certificate to VIP

URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=MGMT_CLUSTER"
METHOD='POST'

response=$(curl -k -X $METHOD \
```

```
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")
```

3.2.2 Replacing Local Manager Certificate with a self-signed certificate

This certificate is not used in deployment without federation, but it should be kept valid to avoid alarms.

```
####LM Certificate


NSX_MANAGER='192.168.110.17'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI='/api/v1/trust-management/csrs?action=self_sign'
METHOD='POST'
BODY=$(cat <<END
{
  "display_name": "LOCAL_MANAGER_LM-A ",
  "subject":
  {
    "attributes":
    [
      {"key":"CN","value":"LOCAL_MANAGER_LM-A"}
    ]
  },
  "key_size": "2048",
  "algorithm": "RSA",
  "days_valid": 825
}
END
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")


CERT_ID=$(echo $response | jq -r '.id')
```

```
URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=LOCAL_MANAGER"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")
```

3.2.3 Replacing APH-AR, APH-TN and CCP certificates with self-signed certificates

The APH-AR certificate is not used in a deployment without federation, but it should be kept valid to avoid alarms.

```
#Find node UUIDs

NSX_MANAGER='192.168.110.17'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI="/api/v1/cluster"
METHOD='GET'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")

NODE_UUID_01=$(echo $response | jq -r '.nodes[0].node_uuid')
NODE_UUID_02=$(echo $response | jq -r '.nodes[1].node_uuid')
NODE_UUID_03=$(echo $response | jq -r '.nodes[2].node_uuid')


NODE_UUIDS_ARRAY=("$NODE_UUID_01" "$NODE_UUID_02" "$NODE_UUID_03" )


##start loop accross the 3 nodes

for i in ${NODE_UUIDS_ARRAY[@]}; do
  NODE_UUID=$i
```

```
#APH

#Generate certificate

URI='/api/v1/trust-management/csrs?action=self_sign'
METHOD='POST'
BODY=$(cat <<END
{
  "display_name": "APH-AR for $NODE_UUID",
  "subject":
  {
    "attributes":
    [
      {"key":"CN","value":"APH-AR for $NODE_UUID"}
    ]
  },
  "key_size": "2048",
  "algorithm": "RSA",
  "days_valid": 825
}
END
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")


CERT_ID=$(echo $response | jq -r '.id')


#Apply certificate

SERVICE_TYPE="APH"
URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=$SERVICE_TYPE&node_id=$NODE_UUID"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")
```

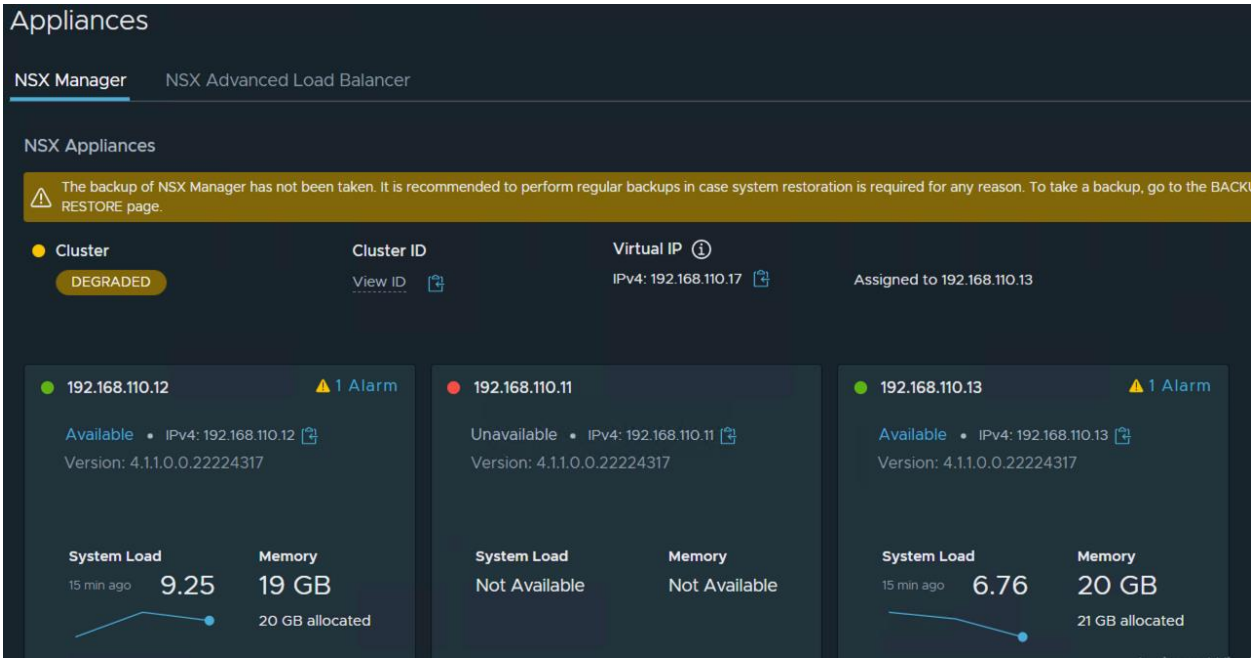
```
#APH_TN

#Generate certificate

URI='/api/v1/trust-management/csrs?action=self_sign'
METHOD='POST'
BODY=$(cat <<END
{
  "display_name": "APH-TN for $NODE_UUID",
  "subject":
  {
    "attributes":
    [
      {"key":"CN","value":"APH-TN for $NODE_UUID"}
    ]
  },
  "key_size": "2048",
  "algorithm": "RSA",
  "days_valid": 825
}
END
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")


CERT_ID=$(echo $response | jq -r '.id')


#Apply certificate

SERVICE_TYPE="APH_TN"
URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=$SERVICE_TYPE&node_id=$NODE_UUID"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
```

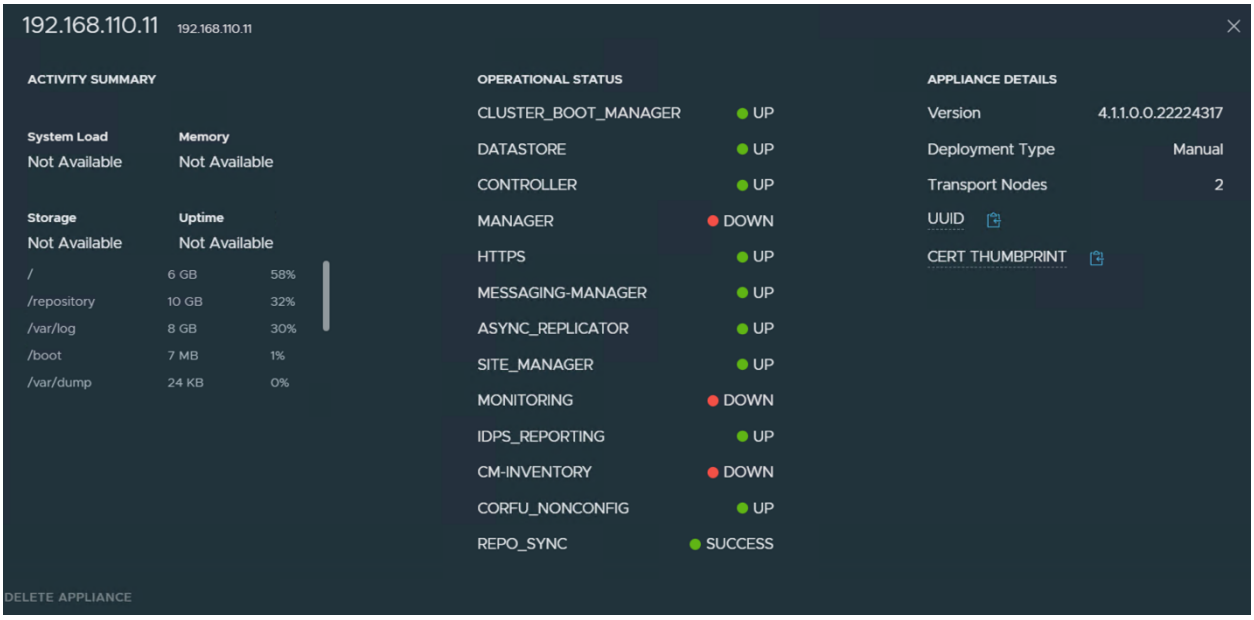
```
    -H "content-type: application/json")


#CCP


URI='/api/v1/trust-management/csrs?action=self_sign'
METHOD='POST'
BODY=$(cat <<END
{
  "display_name": "CCP for $NODE_UUID",
  "subject":
  {
    "attributes":
    [
      {"key":"CN","value":"CCP for $NODE_UUID"}
    ]
  },
  "key_size": "2048",
  "algorithm": "RSA",
  "days_valid": 825
}
END
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")


CERT_ID=$(echo $response | jq -r '.id')


#Apply certificate

SERVICE_TYPE="CCP"
URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=$SERVICE_TYPE&node_id=$NODE_UUID"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
```

```
    -H "content-type: application/json")
done
```

3.2.4 Replacing Corfu Client certificates

All the corfu client certificates are replaced quickly upon invocation of the replacement API call except for the CBM_MP certificates, which require multiple services to restart, and the API/UI to become unavailable for 10-15 minutes. For this reason, it is the last one we replace for each node in the script. It is also important to avoid starting the replacement of the certificates on another node until the previous one is fully operational. The script will pause and will wait for user input. The script should not be resumed until all the cluster members are green and all services operational. The screenshot below shows node 192.168.110.11 after the CBM_MP certificate has been replaced; UI/API is not available as multiple services are down (red). Once all the indicators are green the script can be resumed.






```
##CBM Certificates


NSX_MANAGER='192.168.110.17'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI="/api/v1/cluster"
METHOD='GET'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")

NODE_UUID_01=$(echo $response | jq -r '.nodes[0].node_uuid')
NODE_UUID_02=$(echo $response | jq -r '.nodes[1].node_uuid')
NODE_UUID_03=$(echo $response | jq -r '.nodes[2].node_uuid')


NODE_UUIDS_ARRAY=("$NODE_UUID_01" "$NODE_UUID_02" "$NODE_UUID_03" )


##start loop accross the 3 nodes

for i in ${NODE_UUIDS_ARRAY[@]}; do
  NODE_UUID=$i
```

```
#Replace CBM certificates

CBM_PROFILES=("CBM_AR" "CBM_CCP" "CBM_CLUSTER_MANAGER"
"CBM_CM_INVENTORY" "CBM_IDPS_REPORTING" "CBM_MESSAGING_MANAGER"
"CBM_MONITORING" "CBM_SITE_MANAGER" "CBM_UPGRADE_COORDINATOR"
"CBM_MP")

for i in ${!CBM_PROFILES[@]};
do
sleep 5
CBM_PROFILE=${CBM_PROFILES[$i]}
URI='/api/v1/trust-management/csrs?action=self_sign'
METHOD='POST'
BODY=$(cat <<END
{
  "display_name": "$CBM_PROFILE for node $NODE_UUID",
  "subject":
   {
   "attributes":
   [
     {"key":"CN","value":"$CBM_PROFILE for node $NODE_UUID"}
   ]
  },
  "key_size": "2048",
  "algorithm": "RSA",
  "days_valid": 825
}
END
)

  response=$(curl -k -X $METHOD \
      "https://$NSX_MANAGER$URI" \
      -u "$NSX_USER:$NSX_PASSWORD" \
      -H "content-type: application/json" \
      -d "$BODY")
  echo "Response for generating the certificate of $CBM_PROFILE for
node $NODE_UUID:"
  echo $response

  CERT_ID=$(echo $response | jq -r '.id')

  #Apply certificate
```

```
  URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=$CBM_PROFILE&node_id=$NODE_UUID"
  METHOD='POST'

  response=$(curl -k -w "%{http_code}" -X $METHOD \
      "https://$NSX_MANAGER$URI" \
      -u "$NSX_USER:$NSX_PASSWORD" \
      -H "content-type: application/json")

  echo "Response for applying the certificate of $CBM_PROFILE for node
$NODE_UUID:"
  echo $response

done
# Pause and wait for user input
read -n 1 -s -r -p "Press any key to continue when the cluster is
stable..."

# Continue with the script
echo "Continuing with the script..."
done
```

3.2.5 Replace Corfu Server Certificate

Replacing the corfu server certificate in a three nodes clusters requires the datastore service to restart and resync with the other members of the cluster. It is important to avoid replacing the corfu server certificate on a different node until the operation on the previous one is complete. The screenshot below shows the datastore service resynching after the certificate has been replaced. The process should last few minutes at most, but proceeding with the replacement of the corfu server certificate on another node before the previous one is fully synced carries the risk of making the cluster inoperable. The script will pause after replacing the corfu server certificate on each node, the user should resume it only after checking the status of the datatore service (must be green)


```
#Find node UUID

NSX_MANAGER='192.168.110.17'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI="/api/v1/cluster"
METHOD='GET'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")

NODE_UUID_01=$(echo $response | jq -r '.nodes[0].node_uuid')
NODE_UUID_02=$(echo $response | jq -r '.nodes[1].node_uuid')
NODE_UUID_03=$(echo $response | jq -r '.nodes[2].node_uuid')


NODE_UUIDS_ARRAY=("$NODE_UUID_01" "$NODE_UUID_02" "$NODE_UUID_03" )


##start loop across the 3 nodes

for i in ${NODE_UUIDS_ARRAY[@]}; do
  NODE_UUID=$i


#Generate certificate

URI='/api/v1/trust-management/csrs?action=self_sign'
METHOD='POST'
BODY=$(cat <<END
```

```
{
  "display_name": "Corfu-server for $NODE_UUID",
  "subject":
  {
    "attributes":
    [
      {"key":"CN","value":"Corfu-server for $NODE_UUID"}
    ]
  },
  "key_size": "2048",
  "algorithm": "RSA",
  "days_valid": 825
}
END
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")


CERT_ID=$(echo $response | jq -r '.id')


#Apply certificate

SERVICE_TYPE="CBM_CORFU"
URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=$SERVICE_TYPE&node_id=$NODE_UUID"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")

# Pause and wait for user input
sleep 10
read -n 1 -s -r -p "Press any key to continue when the cluster is
stable..."

# Continue with the script
echo "Continuing with the script..."
```

```
done
```

3.2.6 Delete Unused certificates.

```
NSX_MANAGER='192.168.110.17'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI='/api/v1/trust-management/certificates'
METHOD='GET'


response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")

NOTUSED_CERTIDs=$(echo $response | jq '.results[] | select(.used_by ==
[])' | jq -r '.id')

# Initialize an empty array
NOTUSED_CERTIDs_array=()

# Use a while loop to split the string into an array
while read -r -a temp_array; do
    NOTUSED_CERTIDs_array+=("${temp_array[@]}")
done <<< "$NOTUSED_CERTIDs"


for i in ${!NOTUSED_CERTIDs_array[@]};
do
  CERT_ID=${NOTUSED_CERTIDs_array[$i]}
  URI="/api/v1/trust-management/certificates/$CERT_ID"
  METHOD='DELETE'

  response=$(curl -k -X $METHOD \
      "https://$NSX_MANAGER$URI" \
      -u "$NSX_USER:$NSX_PASSWORD" \
      -H "content-type: application/json")
done
```

3.3 Global Manager 3 node cluster

3.3.1 Replacing API/UI Certificates with a self-signed certificate

```
#API/UI

NSX_MANAGER='192.168.110.18'
NSX_USER='admin'
URI='/api/v1/trust-management/csrs-extended'
METHOD='POST'
BODY=$(cat <<END
{
  "display_name": "API/VIP Certificate",
  "subject": {
    "attributes": [
      {
        "key": "CN",
        "value": "GM-A"
      },
      {
        "key": "O",
        "value": "VMware"
      },
      {
        "key": "OU",
        "value": "NSBU"
      },
      {
        "key": "C",
        "value": "US"
      },
      {
        "key": "ST",
        "value": "CA"
      },
      {
        "key": "L",
        "value": "PA"
      }
    ]
  },
  "key_size": "2048",
  "algorithm": "RSA",
  "extensions": {
    "subject_alt_names": {
```

```
      "ip_addresses": [
        "192.168.110.14",
        "192.168.110.15",
        "192.168.110.16",
        "192.168.110.18"
      ],
      "dns_names": [
        "gm-01a",
        "gm-02a",
        "gm-03a",
        "gm-vip-a"
      ]
    }
  }
}
END
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")


CSR_ID=$(echo $response | jq -r '.id')

#Self-sign the CSR

URI="/api/v1/trust-
management/csrs/$CSR_ID?action=self_sign&days_valid=825"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")


CERT_ID=$(echo $response | jq -r '.id')

#visualize certificate properties
openssl x509 -in <(echo $response | jq -r '.pem_encoded')   -text

#Find node UUIDs
```

```
URI="/api/v1/cluster"
METHOD='GET'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")

NODE_UUID_01=$(echo $response | jq -r '.nodes[0].node_uuid')
NODE_UUID_02=$(echo $response | jq -r '.nodes[1].node_uuid')
NODE_UUID_03=$(echo $response | jq -r '.nodes[2].node_uuid')



#Apply certificate node 01

URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=API&node_id=$NODE_UUID_01"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")

#Apply certificate node 02

URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=API&node_id=$NODE_UUID_02"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")


#Apply certificate node 03

URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=API&node_id=$NODE_UUID_03"
METHOD='POST'
```

```
response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")



#Apply certificate to VIP

URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=MGMT_CLUSTER"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")
```

3.3.2 Replacing Global Manager Certificate with a self-signed certificate

```
####GM Certificate


NSX_MANAGER='192.168.110.18'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI='/api/v1/trust-management/csrs?action=self_sign'
METHOD='POST'
BODY=$(cat <<END
{
  "display_name": "GLOBAL_MANAGER_GM-A",
  "subject":
  {
    "attributes":
    [
      {"key":"CN","value":"GLOBAL_MANAGER_GM-A "}
    ]
  },
  "key_size": "2048",
  "algorithm": "RSA",
  "days_valid": 825
}
END
```

```
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")


CERT_ID=$(echo $response | jq -r '.id')


URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=GLOBAL_MANAGER"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")
```

3.3.3 Replacing APH-AR and APH-TN certificates with self-signed certificates

```
#Find node UUID

NSX_MANAGER='192.168.110.18'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI="/api/v1/cluster"
METHOD='GET'


echo “Discovering NSX UAs UUIDs”

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")

NODE_UUID_01=$(echo $response | jq -r '.nodes[0].node_uuid')
NODE_UUID_02=$(echo $response | jq -r '.nodes[1].node_uuid')
```

```
NODE_UUID_03=$(echo $response | jq -r '.nodes[2].node_uuid')


NODE_UUIDS_ARRAY=("$NODE_UUID_01" "$NODE_UUID_02" "$NODE_UUID_03")


##start loop accross the 3 nodes

for i in ${NODE_UUIDS_ARRAY[@]}; do
  NODE_UUID=$i


#APH

#Generate certificate

URI='/api/v1/trust-management/csrs?action=self_sign'
METHOD='POST'
BODY=$(cat <<END
{
  "display_name": "APH-AR for $NODE_UUID",
  "subject":
  {
    "attributes":
    [
      {"key":"CN","value":"APH-AR for $NODE_UUID"}
    ]
  },
  "key_size": "2048",
  "algorithm": "RSA",
  "days_valid": 825
}
END
)

echo “Generating APH Certificate for node $NODE_UUID”

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")


CERT_ID=$(echo $response | jq -r '.id')
```

```
#Apply certificate

SERVICE_TYPE="APH"
URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=$SERVICE_TYPE&node_id=$NODE_UUID"
METHOD='POST'

echo “Applying APH Certificate for node $NODE_UUID”

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")


#APH_TN

#Generate certificate

URI='/api/v1/trust-management/csrs?action=self_sign'
METHOD='POST'
BODY=$(cat <<END
{
  "display_name": "APH-TN for $NODE_UUID",
  "subject":
  {
    "attributes":
    [
      {"key":"CN","value":"APH-TN for $NODE_UUID"}
    ]
  },
  "key_size": "2048",
  "algorithm": "RSA",
  "days_valid": 825
}
END
)

echo “Generating APH-TN Certificate for node $NODE_UUID”

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
```

```
    -d "$BODY")


CERT_ID=$(echo $response | jq -r '.id')


#Apply certificate

SERVICE_TYPE="APH_TN"
URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=$SERVICE_TYPE&node_id=$NODE_UUID"
METHOD='POST'

echo “Applying APH-TN Certificate for node $NODE_UUID”

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")

sleep 10
done
```

3.3.4 Replacing corfu clients certificates

All the corfu client certificates are replaced quickly upon invocation of the replacement API call except for the CBM_MP certificates, which require multiple services to restart, and the API/UI to become unavailable for 10-15 minutes. For this reason, it is the last one we replace for each node in the script. It is also important to avoid starting the replacement of the certificates on another node until the previous one is fully operational. The script will pause and will wait for user input. The script should not be resumed until all the cluster members are green and all services operational. The screenshot below shows node 192.168.110.11 after the CBM_MP certificate has been replaced; UI/API is not available as multiple services are down (red). Once all the indicators are green the script can be resumed. Besides the CMB_MP certificate, other certificate replacement operations may cause the API to become temporarily unavailable. This script has been augmented with an error handling component which will retry an API call after receiving an error message. The same strategy could be applied to other scripts if deemed

necessarily.

```
##CBM Certificates


NSX_MANAGER='192.168.110.18'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI="/api/v1/cluster"
METHOD='GET'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")

NODE_UUID_01=$(echo $response | jq -r '.nodes[0].node_uuid')
NODE_UUID_02=$(echo $response | jq -r '.nodes[1].node_uuid')
NODE_UUID_03=$(echo $response | jq -r '.nodes[2].node_uuid')


NODE_UUIDS_ARRAY=("$NODE_UUID_01" "$NODE_UUID_02" "$NODE_UUID_03" )


##start loop accross the 3 nodes

for i in ${NODE_UUIDS_ARRAY[@]}; do
  NODE_UUID=$i


#Replace CBM certificates

CBM_PROFILES=("CBM_AR" "CBM_CCP" "CBM_CLUSTER_MANAGER" "CBM_CSM"
"CBM_GM" "CBM_CM_INVENTORY" "CBM_SITE_PROXY_CLIENT"
"CBM_IDPS_REPORTING" "CBM_MESSAGING_MANAGER" "CBM_MONITORING"
"CBM_UPGRADE_COORDINATOR" "CBM_SITE_MANAGER" "CBM_MP")

for i in ${!CBM_PROFILES[@]};
do
sleep 5
CBM_PROFILE=${CBM_PROFILES[$i]}
URI='/api/v1/trust-management/csrs?action=self_sign'
METHOD='POST'
BODY=$(cat <<END
{
```

```
  "display_name": "$CBM_PROFILE for node $NODE_UUID",
  "subject":
   {
   "attributes":
   [
     {"key":"CN","value":"$CBM_PROFILE for node $NODE_UUID"}
   ]
  },
  "key_size": "2048",
  "algorithm": "RSA",
  "days_valid": 825
}
END
)

  while true; do
  response=$(curl -k -X $METHOD \
      "https://$NSX_MANAGER$URI" \
      -u "$NSX_USER:$NSX_PASSWORD" \
      -H "content-type: application/json" \
      -d "$BODY")
  echo "Response for generating the certificate of $CBM_PROFILE for
node $NODE_UUID:"
  if [[ $response =~ "error" ]]; then
        echo "Error in response: $response"
        sleep 10  # Wait for a few seconds before the next attempt
    else
        echo "Success! Response: $response"
        break  # Exit the loop when there is no 'error' in the
response
    fi
  done

  CERT_ID=$(echo $response | jq -r '.id')

  #Apply certificate

  URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=$CBM_PROFILE&node_id=$NODE_UUID"
  METHOD='POST'

  while true; do
  response=$(curl -k -X $METHOD \
      "https://$NSX_MANAGER$URI" \
      -u "$NSX_USER:$NSX_PASSWORD" \
```

```
      -H "content-type: application/json")
  if [[ $response =~ "error" ]]; then
        echo "Error in response: $response"
        sleep 10  # Wait for a few seconds before the next attempt
    else
        echo "Success! Response: $response"
        break  # Exit the loop when there is no 'error' in the
response
    fi
  done


  echo "Response for applying the certificate of $CBM_PROFILE for node
$NODE_UUID:"
  echo $response

done
# Pause and wait for user input
sleep 10
read -n 1 -s -r -p "Press any key to continue when the cluster is
stable..."

# Continue with the script
echo "Continuing with the script..."
done
```

3.3.5 Replace Corfu Server Certificate

Replacing the corfu server certificate in a three nodes clusters requires the datastore service to restart and resync with the other members of the cluster. It is important to avoid replacing the corfu server certificate on a different node until the operation on the previous one is complete. The process should last few minutes at most, but proceeding with the replacement of the corfu server certificate on another node before the previous one is fully synced carries the risk of making the cluster inoperable. The script will pause after replacing the corfu server certificate on each node, the user should resume it only after checking the status of the datatore service (must be green)

```
#Find node UUID

NSX_MANAGER='192.168.110.18'
```

```
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI="/api/v1/cluster"
METHOD='GET'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")

NODE_UUID_01=$(echo $response | jq -r '.nodes[0].node_uuid')
NODE_UUID_02=$(echo $response | jq -r '.nodes[1].node_uuid')
NODE_UUID_03=$(echo $response | jq -r '.nodes[2].node_uuid')


NODE_UUIDS_ARRAY=("$NODE_UUID_01" "$NODE_UUID_02" "$NODE_UUID_03" )


##start loop across the 3 nodes

for i in ${NODE_UUIDS_ARRAY[@]}; do
  NODE_UUID=$i


#Generate certificate

URI='/api/v1/trust-management/csrs?action=self_sign'
METHOD='POST'
BODY=$(cat <<END
{
  "display_name": "Corfu-server for $NODE_UUID",
  "subject":
  {
    "attributes":
    [
      {"key":"CN","value":"Corfu-server for $NODE_UUID"}
    ]
  },
  "key_size": "2048",
  "algorithm": "RSA",
  "days_valid": 825
}
END
)

response=$(curl -k -X $METHOD \
```

```
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")


CERT_ID=$(echo $response | jq -r '.id')

#Apply certificate

SERVICE_TYPE="CBM_CORFU"
URI="/api/v1/trust-
management/certificates/$CERT_ID?action=apply_certificate&service_type
=$SERVICE_TYPE&node_id=$NODE_UUID"
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")

# Pause and wait for user input
sleep 10
read -n 1 -s -r -p "Press any key to continue when the cluster is
stable..."

# Continue with the script
echo "Continuing with the script..."

done
```

3.3.6 Delete Unused certificates.

NSX will block the deletion of any used certificate. This script will collect the UUIDs of the certificates with an empty used_by property, and then it will delete them one by one.

```
NSX_MANAGER='192.168.110.18'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI='/api/v1/trust-management/certificates'
METHOD='GET'


response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
```

```
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")

NOTUSED_CERTIDs=$(echo $response | jq '.results[] | select(.used_by ==
[])' | jq -r '.id')

# Initialize an empty array
NOTUSED_CERTIDs_array=()

# Use a while loop to split the string into an array
while read -r -a temp_array; do
    NOTUSED_CERTIDs_array+=("${temp_array[@]}")
done <<< "$NOTUSED_CERTIDs"


for i in ${!NOTUSED_CERTIDs_array[@]};
do
  CERT_ID=${NOTUSED_CERTIDs_array[$i]}
  URI="/api/v1/trust-management/certificates/$CERT_ID"
  METHOD='DELETE'

  response=$(curl -k -X $METHOD \
      "https://$NSX_MANAGER$URI" \
      -u "$NSX_USER:$NSX_PASSWORD" \
      -H "content-type: application/json")
done
```

3.4 Transport Node Certificates

3.4.1 Replacing a TN certificate via the NSX API

```
NSX_MANAGER='192.168.110.17'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
TN_UUID='f5465f1d-2144-4121-aacf-28b597dfe19d'
URI="/api/v1/trust-management/certificates/action/replace-host-
certificate/$TN_UUID"
METHOD='POST'

openssl req -newkey rsa:2048 -keyout $TN_UUID'.key' -out
$TN_UUID'.csr' -subj '/CN='$TN_UUID -nodes
```

```
openssl x509 -req -in $TN_UUID'.csr' -signkey $TN_UUID'.key' -out
$TN_UUID'.crt' -days 825 -extfile <(printf
"extendedKeyUsage=clientAuth")


BODY=$(cat <<END
  {
    "pem_encoded": "$(awk '{printf "%s\\n", $0}' $TN_UUID'.crt' )",
    "private_key": "$(awk '{printf "%s\\n", $0}' $TN_UUID'.key' )"
  }
END
)

response=$(curl -w "%{http_code}" -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")
```

To check the certificate has been applied, ssh to the transport node a verify the new certificate is there, you can check that the CN=”Transport Node UUID” and that the validity starts immediately and last 825 days.

```
[root@esx-02a:~] openssl x509 -in /etc/vmware/nsx/host-cert.pem -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:

27:7d:a3:3f:fb:01:e8:61:18:fe:c0:2a:0d:3a:b8:ab:1a:85:11:3d
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=f5465f1d-2144-4121-aacf-28b597dfe19d
        Validity
            Not Before: Sep 29 14:34:59 2023 GMT
            Not After : Jan  1 14:34:59 2026 GMT
        Subject: CN=f5465f1d-2144-4121-aacf-28b597dfe19d
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:9e:b6:6a:bc:2a:c3:6e:57:73:51:e9:74:5e:52:
                    e0:36:b1:02:79:57:96:35:40:12:dc:2e:8b:13:10:
                    88:85:a5:db:fc:6a:71:51:68:1d:f4:d3:22:f5:88:
                    e2:11:97:8e:ab:da:d7:f1:5a:77:15:0e:52:9b:76:
```

```
                    39:69:63:23:0b:54:ff:3d:91:4a:1a:5a:7a:96:c4:
                    db:8f:4b:d3:86:ed:89:8e:63:37:2e:d8:1d:d3:d2:
                    ff:76:9e:1c:2b:70:90:2a:3b:37:1a:ab:95:bc:b4:
                    05:20:3d:39:6b:b6:76:fc:58:70:aa:40:29:b7:2d:
                    b0:d2:cd:0b:8c:38:9f:7d:c1:cc:27:a0:cb:7a:98:
                    fb:e8:cb:03:4d:d4:7b:55:21:d8:78:17:68:7a:6f:
                    a9:28:5e:98:38:f1:ff:7e:29:1b:63:cb:c1:be:e5:
                    47:c7:91:72:19:43:da:7c:9f:c4:99:6a:e6:69:5b:
                    f9:7d:df:ad:da:00:2d:14:7b:77:a4:3f:14:7e:4c:
                    76:52:78:a7:47:d8:24:7d:97:43:9e:d9:cc:bf:f9:
                    6d:b4:13:95:b0:a0:a1:53:de:97:f2:e7:25:c9:ee:
                    0b:2e:da:02:b1:86:43:67:0c:8f:5d:ac:6a:cc:fc:
                    a5:ac:da:7a:3e:78:00:04:1f:3b:04:a4:1e:62:a8:
                    d3:a7
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Extended Key Usage:
                TLS Web Client Authentication
            X509v3 Subject Key Identifier:

FA:A4:F9:1E:D4:82:2A:72:83:2A:D6:7B:AF:D2:78:18:E6:05:C0:9F
    Signature Algorithm: sha256WithRSAEncryption
         92:3a:df:2c:e6:fe:77:1b:2b:dc:cd:48:ed:be:a0:75:36:04:
         8f:a6:76:c9:be:96:8d:f3:fe:f8:c7:24:6f:a4:51:05:4a:4d:
         d3:57:80:ef:55:74:9c:00:7d:1d:25:95:37:61:91:cb:d7:0a:
         12:85:33:31:70:29:76:d6:0c:41:8a:dc:22:30:03:a3:be:39:
         9c:61:dc:62:a7:07:c4:46:34:02:2c:03:f5:24:31:16:a2:96:
         61:25:42:dc:3e:33:3c:be:d5:08:9d:ba:3a:79:1a:40:e0:aa:
         dd:48:1c:59:11:c1:16:11:c4:30:45:17:24:20:4b:81:24:ba:
         d4:2e:b7:e8:53:ed:a1:55:a6:19:b9:d2:23:f3:50:86:46:ef:
         aa:5a:a1:b0:b5:6f:34:0c:a3:19:5b:9e:b5:d3:63:8e:38:e5:
         16:60:75:22:12:00:f9:69:8c:3e:e5:c8:11:25:ae:c2:02:a6:
         b0:e8:bf:fd:ea:16:fe:b6:51:10:fe:f0:e7:5e:6f:2a:e9:d4:
         71:02:c6:2b:35:e6:58:08:99:4a:f4:a0:98:fa:b8:cf:78:9d:
         76:52:5b:21:48:9c:77:48:2a:cf:c3:84:82:67:0a:92:c8:30:
         f5:31:1e:6f:3b:88:ea:95:a3:6c:76:a6:56:6d:6f:a7:53:cb:
         4a:06:cb:17
-----BEGIN CERTIFICATE-----
MIIDIjCCAgqgAwIBAgIUJ32jP/sB6GEY/sAqDTq4qxqFET0wDQYJKoZIhvcNAQEL
BQAwLzEtMCsGA1UEAwwkZjU0NjVmMWQtMjE0NC00MTIxLWFhY2YtMjhiNTk3ZGZl
MTlkMB4XDTIzMDkyOTE0MzQ1OVoXDTI2MDEwMTE0MzQ1OVowLzEtMCsGA1UEAwwk
ZjU0NjVmMWQtMjE0NC00MTIxLWFhY2YtMjhiNTk3ZGZlMTlkMIIBIjANBgkqhkiG
9w0BAQEFAAOCAQ8AMIIBCgKCAQEAnrZqvCrDbldzUel0XllgNrECeVeWNUAS3C6L
ExCIhaXb/GpxUWgd9NMi9YjiEZeOq9rX8Vp3FQ5Sm3Y5aWMjC1T/PZFKGlp6lsTb
j0vThu2JjmM3Ltgd09L/dp4cK3CQKjs3GquVvLQFID05a7Z2/FhwqkApty2w0s0L
jDiffcHMJ6DLepj76MsDTdR7VSHYeBdoem+pKF6YOPH/fikbY8vBvuVHx5FyGUPa
```

```
fJ/EmWrmaVv5fd+t2gAtFHt3pD8Ufkx2UninR9gkfZdDntnMv/lttBOVsKChU96X
8uclye4LLtoCsYZDZwyPXaxqzPylrNp6PngABB87BKQeYqjTpwIDAQABozYwNDAT
BgNVHSUEDDAKBggrBgEFBQcDAjAdBgNVHQ4EFgQU+qT5HtSCKnKDKtZ7r9J4GOYF
wJ8wDQYJKoZIhvcNAQELBQADggEBAJI63yzm/ncbK9zNSO2+oHU2BI+mdsm+lo3z
/vjHJG+kUQVKTdNXgO9VdJwAfR0llTdhkcvXChKFMzFwKXbWDEGK3CIwA6O+OZxh
3GKnB8RGNAIsA/UkMRailmElQtw+Mzy+1Qidujp5GkDgqt1IHFkRwRYRxDBFFyQg
S4EkutQut+hT7aFVphm50iPzUIZG76paobC1bzQMoxlbnrXTY4445RZgdSISAPlp
jD7lyBElrsICprDov/3qFv62URD+8Odebyrp1HECxis15lgImUr0oJj6uM94nXZS
WyFInHdIKs/DhIJnCpLIMPUxHm87iOqVo2x2plZtb6dTy0oGyxc=
-----END CERTIFICATE-----
```

3.4.2 Replacing a TN certificate manually

```
TN_UUID='f5465f1d-2144-4121-aacf-28b597dfe19d_manual'

#Generate Self-signed Certitificate with correct extensions
openssl req -newkey rsa:2048 -keyout $TN_UUID'.key' -out
$TN_UUID'.csr' -subj '/CN='$TN_UUID -nodes
openssl x509 -req -in $TN_UUID'.csr' -signkey $TN_UUID'.key' -out
$TN_UUID'.crt' -days 825 -extfile <(printf
"extendedKeyUsage=clientAuth")

#Manually copy certificate and key to the host
scp f5465f1d-2144-4121-aacf-28b597dfe19d_manual.key
root@$TN_IP:/etc/vmware/nsx/host-privkey.pem
scp f5465f1d-2144-4121-aacf-28b597dfe19d_manual.crt
root@$TN_IP:/etc/vmware/nsx/host-cert.pem


#SSH to the host and use the NSXCLI to push the new certificate to the
control plane, we need NSX Manager credentials and certificate
thumbprint
[root@esx-02a:~] nsxcli
esx-02a.corp.vmbeans.com>push host-certificate 192.168.110.17 username
admin thumbprint
75906a5167bad9772e2b0672dfca9d2d4aea6e61d129ef5edff2d8ae7c1a3a9f
Password for API user:
Host certificate was pushed to management plane successfully
```

3.4.3 Retrieve validity of the TN certificates currently in use

```
NSX_MANAGER='192.168.110.17'
NSX_USER='admin'
```

```
NSX_PASSWORD='VMware1!VMware1!'
URI="/api/v1/messaging/clients"
METHOD='GET'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json")


results=$(echo $response |  jq  ' .results')

element_count=$(echo "$results" | jq 'length')

for i in $(seq $element_count)
do
    element=$(echo "$results" | jq --argjson index "$i" '.[$index]')
    if [[ $(echo "$element" | jq '.certificate') != "null" ]]; then
      # Extract the certificate value from the JSON
      certificate=$(echo "$element" | jq -r '.certificate')
      # Extract the TN ID value from the JSON
      tn_id=$(echo "$element" | jq -r '.client_id')
      # Save the certificate to a temporary file
      echo "$certificate" > certificate.crt
      echo "Certificate validity for node $tn_id is:"
      openssl x509 -noout -dates -in certificate.crt
    fi
done

rm certificate.crt
```

3.5 Principal Identities Certificates

3.5.1 Create PI with corresponding self-signed certificate

```
NSX_MANAGER='192.168.110.17'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI="/api/v1/trust-management/certificates?action=import"
METHOD='POST'
PI_NAME="neteng"
PI_ROLE="network_engineer"
NSX_PI_CERT_FILE="pi.crt"
NSX_PI_KEY_FILE="pi.key"
```

```
#Generate self-signed certificate
openssl req -newkey rsa:2048 -x509 -nodes -keyout "$NSX_PI_KEY_FILE" -
new -out "$NSX_PI_CERT_FILE" -subj /CN="$PI_NAME" -extensions
client_server_ssl -config <(cat /etc/ssl/openssl.cnf <(printf
'[client_server_ssl]\nextendedKeyUsage = clientAuth\n')) -sha256 -days
825

#Upload certificate to NSX, private key is not included
BODY=$(cat <<END
  {
    "display_name": "$PI_NAME",
    "pem_encoded": "$(awk '{printf "%s\\n", $0}' $NSX_PI_CERT_FILE)"
  }
END
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")

#retrieve UUID of the uploaded cert. It is needed to associate the PI
to the certificate
CERT_ID=$(echo $response | jq -r '.results[0] | .id')

#Generate a random UUID, required for PI creation
NODE_ID=$(cat /proc/sys/kernel/random/uuid)

#Create new PI and associate it with the cert previously uploaded
URI="/api/v1/trust-management/principal-identities"

BODY=$(cat <<END
    {
         "display_name": "$PI_NAME",
         "name": "$PI_NAME",
         "role": "$PI_ROLE",
         "certificate_id": "$CERT_ID",
         "node_id": "$NODE_ID"
    }
END
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
```

```
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")



#Test PI credentials
URI="/policy/api/v1/infra/segments"
METHOD='GET'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    --key $NSX_PI_KEY_FILE --cert $NSX_PI_CERT_FILE \
    -H "content-type: application/json")

echo $response

#Visualize certificate properties, verify it is a self-signed
certificate
openssl x509 -in pi.crt  -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:

21:ae:04:ba:83:2d:9b:35:25:e3:9d:ce:a2:e9:18:68:50:a7:0f:38
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN = neteng
        Validity
            Not Before: Oct  8 15:19:00 2023 GMT
            Not After : Jan 10 15:19:00 2026 GMT
        Subject: CN = neteng
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:b6:0f:25:00:2c:0c:84:19:ce:88:08:68:15:f9:
                    07:62:59:a8:4c:b9:b4:35:6e:56:c1:04:2c:78:e0:
                    e0:a5:e6:03:2f:1b:88:99:de:da:59:22:c1:78:95:
                    41:23:b2:69:8e:bc:25:a2:3a:8c:95:97:de:c4:08:
                    76:b4:0c:70:0a:2c:6e:a5:ca:ba:8e:e3:0f:19:20:
                    f1:3b:b0:81:61:62:49:5c:a4:17:46:bf:2a:62:d2:
                    7b:2c:09:30:31:f7:a9:40:2a:ac:11:bc:60:69:d2:
                    06:13:12:50:d9:87:6e:29:7b:e4:ce:69:bb:60:44:
                    b3:5b:a9:4d:46:f3:b8:4f:c4:f6:7a:81:b1:94:c5:
```

```
                    70:2e:47:29:b7:3a:3e:fe:d9:aa:e4:31:90:ef:6b:
                    c1:15:54:3d:c1:a4:27:cb:19:5d:d2:f5:7f:b9:0c:
                    6c:fc:1f:de:fa:88:d2:91:95:fb:5f:ee:45:f9:4b:
                    51:33:17:83:b6:93:6c:f8:1d:84:fe:b2:36:78:62:
                    57:d2:6f:10:42:2c:38:43:be:8c:94:80:8b:03:32:
                    46:b1:ab:be:78:9c:b3:63:1f:56:2f:7d:ef:86:25:
                    87:dc:4f:34:fb:96:d8:5a:45:9f:35:7d:93:65:38:
                    1c:0e:3a:f8:66:ac:dc:6b:e7:d9:59:38:fa:5f:8c:
                    25:91
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Extended Key Usage:
                TLS Web Client Authentication
            X509v3 Subject Key Identifier:

82:48:6A:3E:AA:CB:DE:0B:6D:F1:E1:43:40:A1:47:C4:81:45:FB:08
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        2e:4a:31:df:cd:1a:7a:ea:de:76:ae:7a:99:c6:0e:c2:9a:a9:
        30:5a:f8:46:c3:00:f5:62:d4:f2:3c:2f:66:f5:f7:e3:44:69:
        5d:05:c7:f6:58:8f:fd:3f:ad:59:71:25:1d:01:15:26:32:52:
        77:95:d2:a1:50:7e:4c:72:b9:a1:f3:be:73:08:76:8b:e5:76:
        eb:de:49:cf:44:c8:4e:b0:fb:6a:a2:88:36:bd:4d:c8:4a:9e:
        71:f5:7e:ee:d6:bd:51:f1:be:5b:ba:1d:54:56:66:ea:40:f6:
        76:56:10:34:b1:75:73:40:15:78:b8:66:80:ea:6c:5c:8a:fd:
        aa:8b:72:e3:8e:a5:d7:36:bc:40:36:0a:20:ac:8a:1d:f2:eb:
        00:ae:6b:33:95:b9:5d:db:bc:6a:ef:f3:c9:22:3b:9e:1f:91:
        ec:81:5b:5e:9d:b5:d5:1a:c6:b0:df:3e:7b:6b:b8:e3:f3:4e:
        b9:6a:6e:91:47:bf:f5:72:76:4b:2c:26:70:11:0e:4c:f4:78:
        7c:c9:ac:94:ae:38:3b:5e:43:b1:ba:8a:76:59:0d:96:a3:38:
        9e:75:e0:6e:28:de:a8:ab:c5:79:8c:a2:37:bf:56:30:fa:5e:
        d2:67:d3:c2:fc:9b:d5:3e:34:49:e8:6e:77:2b:de:59:42:f1:
        d1:31:13:08
-----BEGIN CERTIFICATE-----
MIIC5jCCAc6gAwIBAgIUIa4EuoMtmzUl453OoukYaFCnDzgwDQYJKoZIhvcNAQEL
BQAwETEPMA0GA1UEAwwGbmV0ZW5nMB4XDTIzMTAwODE1MTkwMFoXDTI2MDExMDE1
MTkwMFowETEPMA0GA1UEAwwGbmV0ZW5nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8A
MIIBCgKCAQEAtg8lACwMhBnOiAhoFfkHYlmoTLm0NW5WwQQseODgpeYDLxuImd7a
WSLBeJVBI7JpjrwlojqMlZfexAh2tAxwCixupcq6juMPGSDxO7CBYWJJXKQXRr8q
YtJ7LAkwMfepQCqsEbxgadIGExJQ2YduKXvkzmm7YESzW6lNRvO4T8T2eoGxlMVw
Lkcptzo+/tmq5DGQ72vBFVQ9waQnyxld0vV/uQxs/B/e+ojSkZX7X+5F+UtRMxeD
tpNs+B2E/rI2eGJX0m8QQiw4Q76MlICLAzJGsau+eJyzYx9WL33vhiWH3E80+5bY
WkWfNX2TZTgcDjr4Zqzca+fZWTj6X4wlkQIDAQABozYwNDATBgNVHSUEDDAKBggr
BgEFBQcDAjAdBgNVHQ4EFgQUgkhqPqrL3gtt8eFDQKFHxIFF+wgwDQYJKoZIhvcN
AQELBQADggEBAC5KMd/NGnrq3nauepnGDsKaqTBa+EbDAPVi1PI8L2b19+NEaV0F
x/ZYj/0/rVlxJR0BFSYyUneV0qFQfkxyuaHzvnMIdovlduveSc9EyE6w+2qiiDa9
```

```
TchKnnH1fu7WvVHxvlu6HVRWZupA9nZWEDSxdXNAFXi4ZoDqbFyK/aqLcuOOpdc2
vEA2CiCsih3y6wCuazOVuV3bvGrv88kiO54fkeyBW16dtdUaxrDfPntruOPzTrlq
bpFHv/VydkssJnARDkz0eHzJrJSuODteQ7G6inZZDZajOJ514G4o3qirxXmMoje/
VjD6XtJn08L8m9U+NEnobncr3llC8dExEwg=
-----END CERTIFICATE-----
```

3.5.2 Update certificate of existing PI

```
NSX_MANAGER='192.168.110.17'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI="/api/v1/trust-management/principal-identities"
METHOD='GET'


curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" | jq '.results[] | {name,id}'


#Sample output ->
{
  "name": "neteng",
  "id": "36350762-bc41-493f-8e2b-7f06fe1d4b73"
}
{
  "name": "LocalManagerIdentity-ab26ee5f-696e-4826-b413-10106cbd8a67",
  "id": "566c838c-8808-4725-b575-7bb220bd4938"
}
{
  "name": "GlobalManagerIdentity-d38e8535-4a72-45fe-939f-
475c07b75aa7",
  "id": "d5afccaa-20f4-49a7-821f-addffb664295"
}
{
  "name": "GlobalManagerIdentity-e308a77a-7828-4256-93b9-
a3f9512dca4a",
  "id": "1ddb6d1f-98c6-464c-8907-65130b281b80"
}
#####
#Assign the id of the PI for which you want to perform the certificate
replcement to the PI_UUID variable
```

```
PI_UUID="36350762-bc41-493f-8e2b-7f06fe1d4b73"



#Generate new self-signed certificate
URI="/api/v1/trust-management/certificates?action=import"
METHOD='POST'
PI_NAME="neteng_updated"
PI_ROLE="network_engineer"
NSX_PI_CERT_FILE="pi.crt"
NSX_PI_KEY_FILE="pi.key"

openssl req -newkey rsa:2048 -x509 -nodes -keyout "$NSX_PI_KEY_FILE" -
new -out "$NSX_PI_CERT_FILE" -subj /CN="$PI_NAME" -extensions
client_server_ssl -config <(cat /etc/ssl/openssl.cnf <(printf
'[client_server_ssl]\nextendedKeyUsage = clientAuth\n')) -sha256 -days
825

#Upload the new certificate to NSX, private key is not included
BODY=$(cat <<END
  {
    "display_name": "$PI_NAME",
    "pem_encoded": "$(awk '{printf "%s\\n", $0}' $NSX_PI_CERT_FILE)"
  }
END
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")

#retrieve UUID of the uploaded cert. It is needed to associate the PI
to the certificate
CERT_ID=$(echo $response | jq -r '.results[0] | .id')

#Associate the new certificate with the existing PI
URI="/api/v1/trust-management/principal-
identities?action=update_certificate"
METHOD='POST'
BODY=$(cat <<END
  {
    "principal_identity_id": "$PI_UUID",
    "certificate_id" : "$CERT_ID"  }
```

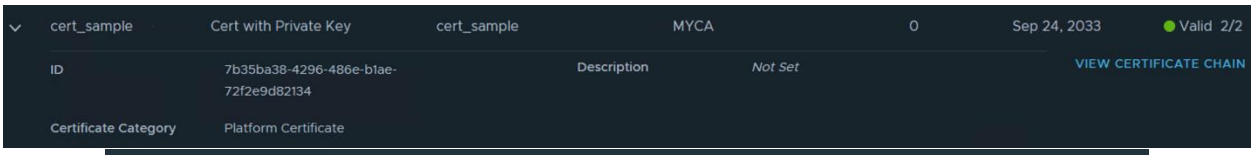
```
END
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")
```

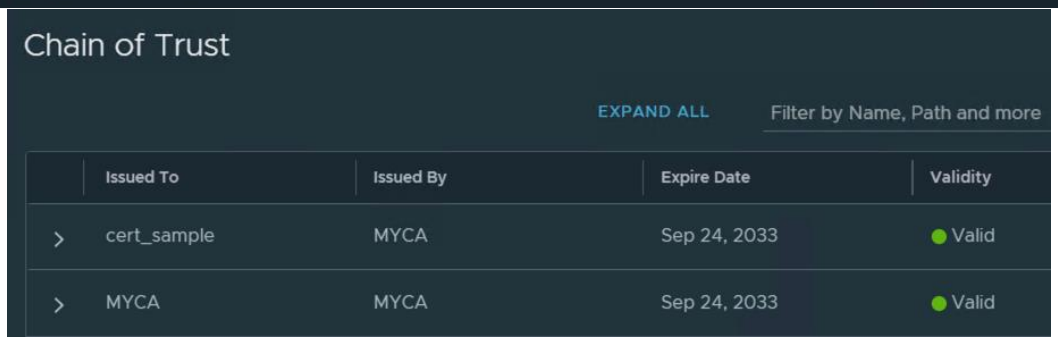
3.6 Additional Recipes

3.6.1 Import a CA signed certificate via the API (CSR Generated outside of NSX)

```
#Generate CA private key and certificate
openssl genrsa -out ca.key  2048
openssl req -x509 -new -nodes -key ca.key -sha256 -days 3650 -out
ca.crt -subj "/CN=MYCA" -extensions v3_ca

#Generate private key for sample certificate
openssl genrsa -out cert_sample.key 2048

#Generate CSR for sample certificate, we will include both SERVER and
CLIENT EKUs so that the certificate can be used for any service (NSX
generated CSR have the same EKU settings)
openssl req -new -key cert_sample.key -out  cert_sample.csr -subj
/CN=cert_sample -extensions client_server_ssl -config <(cat
/etc/ssl/openssl.cnf <(printf '[client_server_ssl]\nextendedKeyUsage =
serverAuth,clientAuth\n'))

#Sign CSR via the CA cert and key
openssl x509 -req -in cert_sample.csr -CA ca.crt -CAkey ca.key -
CAcreateserial -out cert_sample.crt -days 3650 -sha256 -extfile
<(printf "extendedKeyUsage=serverAuth,clientAuth")


#Generate JSON to upload certificate
BODY=$(cat <<END
  {
    "display_name": "cert_sample",
    "pem_encoded": "$(awk '{printf "%s\\n", $0}' cert_sample.crt
ca.crt )",
    "private_key": "$(awk '{printf "%s\\n", $0}' cert_sample.key )"
  }
END
```

```
)


NSX_MANAGER='192.168.110.17'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI='/api/v1/trust-management/certificates?action=import'
METHOD='POST'

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")
```





3.6.2 Import a CA signed certificate via the API (CSR Generated in NSX)

```
NSX_MANAGER='192.168.110.17'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI='/api/v1/trust-management/csrs'
METHOD='POST'
BODY=$(cat <<END
{
  "display_name": "sample_cert",
  "subject":
  {
    "attributes":
    [
      {"key":"CN","value":"sample_cert"}
```

```
    ]
  },
  "key_size": "2048",
  "algorithm": "RSA"
}
END
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")


#Generate CA private key and certificate
openssl genrsa -out ca.key  2048
openssl req -x509 -new -nodes -key ca.key -sha256 -days 3650 -out
ca.crt -subj "/CN=MYCA" -extensions v3_ca


#Sign CSR via the CA cert and key
#This step is or completeness only and can be used in a lab, in a real
environment the CSR will be signed by the Enterprise CA
#You can get the content of the PEM file to be summitted to the CA via
this command:
# echo $response | jq -r '.pem_encoded'
openssl x509 -req -in <(echo $response | jq -r '.pem_encoded') -CA
ca.crt -CAkey ca.key -CAcreateserial -out cert_sample.crt -days 3650 -
sha256 -extfile <(printf "extendedKeyUsage=serverAuth,clientAuth")


#Generate JSON to upload certificate
BODY=$(cat <<END
  {
    "display_name": "cert_sample",
    "pem_encoded": "$(awk '{printf "%s\\n", $0}' cert_sample.crt
ca.crt )"
  }
END
)

URI='/api/v1/trust-management/certificates?action=import'
METHOD='POST'

response=$(curl -k -X $METHOD \
```

```
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")
```

3.6.3 Generate and import a self-signed certificate generated outside of NSX

```
NSX_MANAGER='192.168.110.17'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI="/api/v1/trust-management/certificates?action=import"
METHOD='POST'
PI_NAME="neteng"

#Generate self-signed certificate
openssl req -newkey rsa:2048 -x509 -nodes -keyout sample.key -new -out
sample.crt -subj /CN="sample_cert" -extensions client_server_ssl -
config <(cat /etc/ssl/openssl.cnf <(printf
'[client_server_ssl]\nextendedKeyUsage = clientAuth\n')) -sha256 -days
3650

#Upload certificate to NSX, private key is not included
BODY=$(cat <<END
  {
    "display_name": "sample_cert",
    "pem_encoded": "$(awk '{printf "%s\\n", $0}' sample.crt)",
    "private_key": "$(awk '{printf "%s\\n", $0}' sample.key)"
  }
END
)

response=$(curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json" \
    -d "$BODY")

#retrieve UUID of the uploaded cert. It may be needed if using this
block in larger scripts
CERT_ID=$(echo $response | jq -r '.results[0] | .id')
```

3.6.4 Verify certificate replacement via opensll on a specific port

```
openssl s_client -showcerts -connect 192.168.110.17:1236

#output:
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 CN = APH-AR for 1788601c-3b10-4664-8674-845ce07e1fc1
verify error:num=18:self-signed certificate
verify return:1
depth=0 CN = APH-AR for 1788601c-3b10-4664-8674-845ce07e1fc1
verify return:1
80BB4CFBCF7F0000:error:0A000126:SSL routines:ssl3_read_n:unexpected
eof while reading:../ssl/record/rec_layer_s3.c:308:
---
Certificate chain
 0 s:CN = APH-AR for 1788601c-3b10-4664-8674-845ce07e1fc1
   i:CN = APH-AR for 1788601c-3b10-4664-8674-845ce07e1fc1
   a:PKEY: rsaEncryption, 2048 (bit); sigalg: RSA-SHA256
   v:NotBefore: Sep 21 17:45:41 2023 GMT; NotAfter: Dec 24 17:45:41
2025 GMT
-----BEGIN CERTIFICATE-----
MIIDZTCCAk2gAwIBAgIGAYq42PcYMA0GCSqGSIb3DQEBCwUAMDoxODA2BgNVBAMM
L0FQSC1BUiBmb3IgMTc4ODYwMWMtM2IxMC00NjY0LTg2NzQtODQ1Y2UwN2UxZmMx
MB4XDTIzMDkyMTE3NDU0MVoXDTI1MTIyNDE3NDU0MVowOjE4MDYGA1UEAwwvQVBI
LUFSIGZvciAxNzg4NjAxYy0zYjEwLTQ2NjQtODY3NC04NDVjZTA3ZTFmYzEwggEi
MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDDV/p4eeQ5hW6ZLYQMEZojktzn
8lT3XMEzayURe4Jlt3S0tjBlIOk56VQYN4m8VuQYWpXy+S/0GWhlUruuk2nUl/jo
FCa7JCoPdfUstb0J46m2/7dHnM0hUEdyU+sTJ+zyKtW3PqhhdrI8XPVbzdKuaIs/
DafaKYLsA4xj/AxQEJHH7FWb5+hvJxMemnednVLud0IL1LakFoc6a/FBeJJCuVtw
4F9+rXysUZNvclYQp6d8jWwfQyHfTmobgbHo5KR/Ywf8QTD2zbwb82yG7EZDCuKm
CaeUO2d68x6Ny+VX2NT95onuhfvCXJpfYF9sk6MQswDlokhwPztXTn8cj67rAgMB
AAGjcTBvMCAGA1UdJQEB/wQWMBQGCCsGAQUFBwMCBggrBgEFBQcDATAMBgNVHRMB
Af8EAjAAMD0GA1UdEQEB/wQzMDGCL0FQSC1BUiBmb3IgMTc4ODYwMWMtM2IxMC00
NjY0LTg2NzQtODQ1Y2UwN2UxZmMxMA0GCSqGSIb3DQEBCwUAA4IBAQCJfjg7Sido
PxvoLrCR2UnWs4xYvJ+e+8ijRDmwshZ/kq4oV/ixefalwbhKzCLePMzUpDLJcG2M
0XSPgP4PTkK/6v2s83rSVVMAr1I5OIkg99YzcjInVLQLlg3eQkgHmMTp0WgSUSAH
V/jmO8Wo3YEehnUAdZ8SqBWTYe9k+WmNKPQXLAKMfizoYnbc4Pc2K29wwcNEU1pE
jGaEtT24tq0B52+yJrDKROd5DwBrSQpZG3wuLufFwKVKQl38Q2kQwLAbtYAzCiH8
g/xKVo6ZhESALqW9YPvXXpLTI2LYZiyPPY+72X0RjepLoq+sh8h0SCiy4BSRZheu
tHYIkly4j507
-----END CERTIFICATE-----
---
```

```
Server certificate
subject=CN = APH-AR for 1788601c-3b10-4664-8674-845ce07e1fc1
issuer=CN = APH-AR for 1788601c-3b10-4664-8674-845ce07e1fc1
```

3.6.5 Manage Corfu certificate expiry check

This setting is a node level configuration. In a 3-node cluster it must be applied, or checked on the three nodes individually pointing to the three individual node IPs.

```
#Get Corfu Certificate Expiry Check Status

NSX_MANAGER='192.168.110.17'
NSX_USER='admin'
NSX_PASSWORD='VMware1!VMware1!'
URI='/api/v1/node/services/datastore/corfu_cert_expiry_check'
METHOD='GET'

curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json"

#Disable Corfu Certificate Expiry Check

URI='/api/v1/node/services/datastore/corfu_cert_expiry_check?action=di
sable'
METHOD='POST'

curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json"

#Enable Corfu Certificate Expiry Check

URI='/api/v1/node/services/datastore/corfu_cert_expiry_check?action=en
able'
METHOD='POST'

curl -k -X $METHOD \
    "https://$NSX_MANAGER$URI" \
    -u "$NSX_USER:$NSX_PASSWORD" \
    -H "content-type: application/json"
```