HEALTH CHECK HANDBOOK

VIRTUAL CONTROL

VMWARE CLOUD FOUNDATION SOLUTIONS

NSX Health Check Handbook

Complete NSX Manager health check procedures covering cluster status, transport nodes, logical switching, routing, firewall rules, and certificate validation.

NSX MANAGER | TRANSPORT NODES | ROUTING | DFW | CERTIFICATES

VCF 9.0 VMware Cloud Foundation

PROPRIETARY & CONFIDENTIAL

NSX 4.x Health Check Handbook

Comprehensive Health Verification for VMware NSX in VCF 9

Author: Virtual Control LLC **Date:** March 2026 **Version:** 1.0 **Classification:** Internal Use **Platform:** VMware Cloud Foundation 9.0 / NSX 4.2.x

Table of Contents

1. Overview & Purpose

This handbook provides a **complete, step-by-step health check procedure** for VMware NSX 4.x deployed within a VCF 9.0 environment. It is designed for VMware administrators who need to verify NSX health during:

- **Routine maintenance windows** — Weekly or monthly proactive checks
- **Pre/post-upgrade validation** — Before and after NSX patches or upgrades
- **Incident troubleshooting** — When connectivity, firewall, or routing issues occur
- **Environment handover** — Documenting health state for audits or transfers

What This Document Covers

Area	Components Checked
Management Plane	NSX Manager cluster, services, certificates, backups
Control Plane	Controller connectivity, transport node preparation
Data Plane	TEP connectivity, tunnels, BFD sessions
Edge Services	Edge clusters, HA state, routing, BGP
Security	Distributed firewall, gateway firewall, rule realization
Networking	Segments, logical switches, MAC/ARP/VTEP tables

Health Check Methodology

Each check in this handbook follows a consistent format:

1. **What to check** — Description of the component and why it matters
2. **How to check** — Exact CLI command or API call (copy-paste ready)
3. **Expected output** — What a healthy result looks like
4. **Pass / Warn / Fail criteria** — Clear thresholds with visual indicators
5. **Remediation** — What to do if the check fails

Environment Variables: Throughout this document, replace the following placeholders with your actual values:
`$NSX_VIP` = NSX Manager VIP (e.g., nsx-vip.lab.local)
`$NSX_NODE1/2/3` = Individual NSX Manager nodes
`$NSX_USER` = admin
`$NSX_PASS` = NSX Manager admin password

2. Prerequisites

Required Access

Access Type	Target	Credentials Needed
HTTPS (443)	NSX Manager VIP	admin / password
SSH (22)	Each NSX Manager node	admin or root
SSH (22)	ESXi hosts (transport nodes)	root
SSH (22)	Edge nodes (bare metal)	admin or root
API	NSX Manager VIP:443	admin / password

Required Tools

Tool	Purpose	Install Location
`curl`	API calls to NSX Manager	Jumpbox / workstation
`ssh`	CLI access to NSX / ESXi / Edge	Jumpbox / workstation
`jq`	JSON parsing for API output	`apt install jq` or `brew install jq`
`openssl`	Certificate verification	Pre-installed on Linux/Mac
Web Browser	NSX Manager UI verification	Workstation

Environment Variables Setup

```
# NSX Manager
export NSX_VIP="nsx-vip.lab.local"
export NSX_NODE1="nsx-mgr-01.lab.local"
export NSX_NODE2="nsx-mgr-02.lab.local"
export NSX_NODE3="nsx-mgr-03.lab.local"
export NSX_USER="admin"
export NSX_PASS="YourPassword123!"

# Convenience function for NSX API calls
nsx_api() {
  curl -sk -u "$NSX_USER:$NSX_PASS" \
    -H "Content-Type: application/json" \
```

```
    "https://$NSX_VIP$1" 2>/dev/null
}

# Example usage:
# nsx_api /api/v1/cluster/status | jq .
```

Security Note: Never store passwords in plain text in production. Use a credential vault or enter passwords interactively. These examples use environment variables for convenience during health checks only.

3. Quick Reference — All Checks Summary

#	Check	Method	Pass Criteria	Warn Criteria	Fail Criteria
4.1	Manager Cluster Status	API	`STABLE`	`DEGRADED`	`UNSTABLE` / unreachable
4.2	Node Health	API	All nodes `CONNECTED`	1 node degraded	2+ nodes down
4.3	Service Status	CLI	All services `running`	Non-critical stopped	Critical service stopped
5.1	Manager CPU	CLI/API	< 70%	70-85%	> 85%
5.2	Manager Memory	CLI/API	< 75%	75-90%	> 90%
5.3	Manager Disk	CLI	< 70% used	70-85% used	> 85% used
6.1	Certificates	API	> 30 days to expiry	7-30 days	< 7 days or expired
7.1	Backup Config	API	Configured & recent	> 24h since last	No backup configured
8.1	Transport Nodes	API	All `SUCCESS`	Any `IN_PROGRESS`	Any `FAILED`
8.3	NSX Agents	CLI	`nsx-mpa` + `nsx-proxy` running	Agent restarting	Agent not running
9.1	TEP Connectivity	CLI	vmkping MTU 1600 succeeds	Intermittent loss	vmkping fails
9.2	BFD Sessions	CLI	All `UP`	Any `INIT`	Any `DOWN`
10.1	Edge Cluster	API	All members `UP`	1 member degraded	Multiple members down
10.2	Edge HA	API	`ACTIVE/STANDBY` correct	HA state mismatch	Both `STANDBY`
11.1	BGP Neighbors	CLI	All `Established`	Any `OpenConfirm`	Any `Idle` / `Active`
11.3	Gateway State	API	`REALIZED`	`IN_PROGRESS`	`ERROR`
12.1	DFW Realization	API	All rules `REALIZED`	Any `IN_PROGRESS`	Any `ERROR`
14.1	Segments	API	All `SUCCESS`	Any `IN_PROGRESS`	Any `ERROR`
15	Alarms	API	0 critical alarms	Warning alarms present	Critical alarms present

4. NSX Manager Cluster Health

4.1 Cluster Status (API + CLI)

What: Verify the NSX Manager cluster is stable with all 3 nodes participating.

Why: A degraded management cluster means reduced redundancy. An unstable cluster can cause configuration failures and data loss.

API Method

```
# Get cluster status
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/api/v1/cluster/status" | jq .
```

Expected Output (Healthy):


```
{
  "detailed_cluster_status": {
    "overall_status": "STABLE",
    "groups": [
      {
        "group_type": "CONTROLLER",
        "group_status": "STABLE",
        "members": [
          {
            "member_uuid": "abc123...",
            "member_status": "UP",
            "member_fqdn": "nsx-mgr-01.lab.local"
          },
          {
            "member_uuid": "def456...",
            "member_status": "UP",
            "member_fqdn": "nsx-mgr-02.lab.local"
          },
          {
            "member_uuid": "ghi789...",
            "member_status": "UP",
            "member_fqdn": "nsx-mgr-03.lab.local"
          }
```

```
        ]
      },
      {
        "group_type": "MANAGER",
        "group_status": "STABLE",
        "members": [...]
      },
      {
        "group_type": "POLICY",
        "group_status": "STABLE",
        "members": [...]
      }
    ]
  },
  "mgmt_cluster_status": {
    "status": "STABLE"
  },
  "control_cluster_status": {
    "status": "STABLE"
  }
}
```

CLI Method (SSH to any NSX Manager)

```
ssh admin@$NSX_NODE1
# Then run:
get cluster status
```

Expected Output:

```
Cluster Id: a1b2c3d4-e5f6-7890-abcd-ef1234567890
Overall Status: STABLE

Group Type: CONTROLLER
  Group Status: STABLE
  Leader: nsx-mgr-01.lab.local

Group Type: MANAGER
  Group Status: STABLE
  Leader: nsx-mgr-01.lab.local
```

```
Group Type: POLICY
  Group Status: STABLE
  Leader: nsx-mgr-02.lab.local
```

Pass / Warn / Fail

Result	Criteria	Indicator
PASS	`overall_status` = `STABLE`, all groups `STABLE`	All 3 nodes UP, all group leaders elected
WARN	`overall_status` = `DEGRADED`	1 node down but quorum maintained
FAIL	`overall_status` = `UNSTABLE` or unreachable	2+ nodes down, no quorum, API unresponsive

Remediation if FAIL:
1. SSH to each node individually and check: `get cluster status`
2. Check node reachability: `ping $NSX_NODE1`
3. Restart cluster service: `restart cluster boot-manager`
4. Check logs: `/var/log/cluster-manager/cluster-manager.log`
5. If split-brain: identify the minority partition and restart those nodes

4.2 Individual Node Health

What: Check the health status of each individual NSX Manager node.

Why: A node can be in the cluster but unhealthy (high load, service failures).

API Method

```
# List all cluster nodes with health
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/api/v1/cluster/nodes" | jq '.results[] | {
    fqdn: .fqdn,
    ip: .ip_address,
    status: .status,
    version: .node_version
  }'
```

Check Individual Node Appliance Health

```
# Per-node health via appliance API
for NODE in $NSX_NODE1 $NSX_NODE2 $NSX_NODE3; do
  echo "=== $NODE ==="
  curl -sk -u "$NSX_USER:$NSX_PASS" \
    "https://$NODE/api/v1/node" | jq '{
      hostname: .hostname,
      kernel_version: .kernel_version,
      product_version: .product_version,
      uptime: .system_time
    }'
done
```

Pass / Warn / Fail

Result	Criteria	Indicator
PASS	All 3 nodes report status `CONNECTED` and matching versions	Healthy cluster
WARN	Version mismatch between nodes	Possible mid-upgrade state
FAIL	Any node `DISCONNECTED` or unreachable	Node failure

4.3 Service Status Verification

What: Verify all critical NSX Manager services are running on each node.

Why: Individual service failures can cause specific feature outages even when the cluster appears healthy.

CLI Method (SSH to each NSX Manager)

```
ssh admin@$NSX_NODE1
get services
```

Expected Output (all running):

```
Service Name          Service Status
--------------------------------------
async_replicator      running
cluster-manager       running
controller            running
corfu                 running
```

```
corfu-nonconfig       running
datastore             running
http                  running
install-upgrade       running
liagent               running
manager               running
migration-coordinator running
monitoring            running
nsx-message-bus       running
nsx-ui-plugin         running
phonehome             running
platform-client       running
policy                running
proton                running
search                running
syslog-server         running
upgrade-coordinator   running
```

API Method

```
# Get node services status
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/api/v1/node/services" | jq '.results[] | {
    service: .service_name,
    status: .service_status.runtime_state
  }'
```

Critical Services (must be running)

Service	Function	Impact if Down
`controller`	Control plane	Transport node connectivity loss
`corfu`	Distributed datastore	Configuration data unavailable
`manager`	Management plane	API / UI unavailable
`policy`	Policy engine	Security policy changes fail
`http`	Web server / API	All API calls fail
`proton`	Message bus	Cluster communication failure
`datastore`	Data persistence	Config persistence failure

`search`	Search engine	UI search / inventory failure

Critical: If `controller`, `corfu`, or `manager` are not running, the NSX environment is in a critical state. Do not make configuration changes until these services are restored.

5. NSX Manager Resource Utilization

5.1 CPU & Memory

What: Check CPU and memory utilization on each NSX Manager node.

Why: NSX Manager nodes under resource pressure may respond slowly or fail to process API requests, causing cascading issues across the environment.

CLI Method

```
ssh admin@$NSX_NODE1
# CPU utilization
get node-stats
```

API Method

```
# Node status with resource utilization
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_NODE1/api/v1/node/status" | jq '{
    cpu_cores: .cpu_cores,
    load_average: .load_average,
    mem_total: .mem_total,
    mem_used: .mem_used,
    mem_cache: .mem_cache,
    swap_total: .swap_total,
    swap_used: .swap_used,
    uptime: .uptime
  }'
```

Expected Output (Healthy):

```
{
  "cpu_cores": 12,
  "load_average": [1.2, 1.5, 1.8],
  "mem_total": 49152,
  "mem_used": 32768,
```

```
  "mem_cache": 8192,
  "swap_total": 8192,
  "swap_used": 0,
  "uptime": 8640000
}
```

Pass / Warn / Fail

Metric	PASS	WARN	FAIL
CPU Load Average (per core)	< 0.7	0.7 - 0.85	> 0.85
Memory Used %	< 75%	75% - 90%	> 90%
Swap Used	0 MB	< 1 GB	> 1 GB

Remediation:
1. High CPU: Check for excessive API polling — `get service http connection-limit`
2. High Memory: Restart non-critical services, or increase VM memory (requires downtime per node)
3. Swap usage: Indicates memory pressure — investigate with `top -b -n 1 | head -20` (as root)

5.2 Disk & Filesystem

What: Check disk space utilization on all NSX Manager filesystem partitions.

Why: Full disks cause service crashes, log loss, and database corruption.

CLI Method

```
ssh admin@$NSX_NODE1
# As admin:
get filesystem-info
```

API Method

```
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_NODE1/api/v1/node/status" | jq '.filesystem'
```

Root SSH Method (more detail)

```
ssh root@$NSX_NODE1
df -h
```

Expected Output:

```
Filesystem      Size  Used  Avail  Use%  Mounted on
/dev/sda2        40G   15G    23G   40%  /
/dev/sda3        80G   28G    48G   37%  /nonconfig-datastore
/dev/sda5        40G   12G    26G   32%  /config-datastore
/dev/sda6        20G    4G    15G   21%  /image
tmpfs            24G   12M    24G    1%  /dev/shm
```

Critical Filesystem Thresholds

Filesystem	PASS	WARN	FAIL
`/` (root)	< 70%	70-85%	> 85%
`/nonconfig-datastore`	< 70%	70-85%	> 85%
`/config-datastore`	< 70%	70-85%	> 85%
`/image`	< 80%	80-90%	> 90%

Remediation:
1. Clean old log files: `ls -la /var/log/` — identify large files
2. Clean old upgrade bundles: `ls -la /image/`
3. Compress or rotate logs: `logrotate -f /etc/logrotate.conf`
4. Check corfu compaction: `get service corfu compaction-status`

5.3 Thresholds & Alarms

What: Check if any resource-related alarms are active on NSX Manager nodes.

API Method

```
# Check system alarms related to resources
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/api/v1/alarms?feature_name=system" | jq '.results[] | {
    id: .id,
    severity: .severity,
```

```
    description: .description,
    status: .status,
    entity_id: .entity_id
  }'
```

6. Certificate Health

6.1 List All Certificates

What: Retrieve all certificates managed by NSX and verify none are expired or expiring soon.

Why: Expired certificates cause communication failures between NSX components, transport nodes, and external integrations.

API Method

```
# List all certificates
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/api/v1/trust-management/certificates" | jq '.results[] | {
    id: .id,
    display_name: .display_name,
    not_before: .not_before,
    not_after: .not_after,
    subject: .pem_encoded | split("\n")[0]
  }'
```

Extract Expiry Dates for Quick Review

```
# Check certificate expiry with days remaining
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/api/v1/trust-management/certificates" | jq -r '
  .results[] |
  "\(.display_name)\t\(.not_after)\t\(
    ((.not_after / 1000) - (now | floor)) / 86400 | floor
  ) days remaining"'
```

Expected Output:

```
nsx-mgr-01-cert     1774003200000    365 days remaining
nsx-mgr-02-cert     1774003200000    365 days remaining
nsx-mgr-03-cert     1774003200000    365 days remaining
```

```
nsx-api-cert          1774003200000    365 days remaining
mp-cluster-cert       1774003200000    365 days remaining
```

6.2 Expiry Verification

Pass / Warn / Fail

Result	Criteria	Indicator
PASS	All certificates > 30 days from expiry	Healthy
WARN	Any certificate 7-30 days from expiry	Plan replacement
FAIL	Any certificate < 7 days or expired	Immediate action required

Verify Certificate via OpenSSL

```
# Verify the NSX Manager certificate externally
echo | openssl s_client -connect $NSX_VIP:443 2>/dev/null | \
  openssl x509 -noout -dates -subject -issuer
```

Expected Output:

```
notBefore=Jan 15 00:00:00 2026 GMT
notAfter=Jan 15 00:00:00 2028 GMT
subject=CN = nsx-vip.lab.local
issuer=CN = NSX CA
```

Critical: If the NSX Manager certificate is expired:
1. Transport nodes will lose control plane connectivity
2. API calls will fail with certificate errors
3. DFW rule updates will stop propagating
Action: Follow VMware KB 83054 to replace expired certificates.

7. Backup Verification

7.1 Backup Configuration

What: Verify that NSX Manager backups are configured and running on schedule.

Why: Without valid backups, any cluster failure could result in complete NSX configuration loss.

API Method

```
# Get backup configuration
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/api/v1/cluster/backups/config" | jq '{
    backup_enabled: .backup_enabled,
    backup_schedule: .backup_schedule,
    remote_file_server: .remote_file_server.server,
    remote_path: .remote_file_server.directory_path
  }'
```

Expected Output (Healthy):

```
{
  "backup_enabled": true,
  "backup_schedule": {
    "resource_type": "WeeklyBackupSchedule",
    "days_of_week": ["MONDAY", "WEDNESDAY", "FRIDAY"],
    "hour_of_day": 2,
    "minute_of_day": 0
  },
  "remote_file_server": "backup-server.lab.local",
  "remote_path": "/backups/nsx/"
}
```

7.2 Backup History

```
# Check backup history
curl -sk -u "$NSX_USER:$NSX_PASS" \
```

```
  "https://$NSX_VIP/api/v1/cluster/backups/history" | jq '.results[0:5] | .[] | {
    start_time: .start_time,
    end_time: .end_time,
    backup_status: .success,
    node: .node_id
  }'
```

Pass / Warn / Fail

Result	Criteria	Indicator
PASS	Backup configured, last successful < 24 hours ago	Healthy
WARN	Backup configured, last successful > 24 hours ago	Check schedule
FAIL	No backup configured, or all recent backups failed	Critical risk

Remediation:
1. Configure backup: NSX UI → System → Backup & Restore
2. Trigger manual backup: `POST /api/v1/cluster/backups?action=backup_to_remote`
3. Verify remote server connectivity: SSH to NSX Manager → `ping backup-server.lab.local`
4. Check SFTP permissions on backup target directory

8. Transport Node Health

8.1 Transport Node Status

What: Verify all ESXi hosts prepared as NSX transport nodes are in a healthy state.

Why: Transport node failures mean VMs on that host lose NSX networking (overlay, DFW, load balancing).

API Method

```
# Get all transport node states
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/api/v1/transport-nodes/state" | jq '.results[] | {
    transport_node_id: .transport_node_id,
    node_deployment_state: .node_deployment_state.state,
    state: .state,
    status: .status,
    host_node_deployment_status: .host_node_deployment_status
  }'
```

Expected Output (Healthy):

```
{
  "transport_node_id": "abc123",
  "node_deployment_state": "NODE_READY",
  "state": "success",
  "status": "UP"
}
```

Summary View

```
# Compact summary of all transport node states
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/api/v1/transport-nodes/state?status=UP" | jq '.result_count'

curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/api/v1/transport-nodes/state?status=DOWN" | jq
```

```
'.result_count'

curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/api/v1/transport-nodes/state?status=DEGRADED" | jq
'.result_count'
```

Pass / Warn / Fail

Result	Criteria	Indicator
PASS	All transport nodes `SUCCESS` / `UP`	Healthy
WARN	Any node `IN_PROGRESS` (e.g., during upgrade)	Monitor
FAIL	Any node `FAILED` or `DOWN`	Immediate investigation

8.2 ESXi nsxcli Commands

What: Verify NSX components on individual ESXi hosts via the `nsxcli` shell.

SSH to ESXi and Enter nsxcli

```
ssh root@<esxi-host>
nsxcli
```

Check Manager Connection

```
> get managers
```

Expected Output:

```
Manager IP      : 192.168.1.71
Manager IP      : 192.168.1.72
Manager IP      : 192.168.1.73
Connected to    : 192.168.1.71
```

Check Controller Connection

```
> get controllers
```

Expected Output:

```
Controller IP  : 192.168.1.71
Controller IP  : 192.168.1.72
Controller IP  : 192.168.1.73
Connected to   : 192.168.1.72
Status         : Connected
```

8.3 NSX Agent Status

What: Verify the NSX agent processes (nsx-mpa, nsx-proxy) are running on each ESXi host.

Why: These agents handle management plane communication and data plane programming.

CLI Method (on ESXi host)

```
# Check NSX services on ESXi
/etc/init.d/nsx-mpa status
/etc/init.d/nsx-proxy status
/etc/init.d/nsx-context-mux status
```

Expected Output:

```
nsx-mpa is running
nsx-proxy is running
nsx-context-mux is running
```

Alternative Check

```
esxcli software vib list | grep -i nsx
```

Expected Output (shows installed NSX VIBs):

```
nsx-esx-datapath       4.2.0.0.0-12345678   VMW  VMwareCertified  2026-01-15
nsx-mpa                4.2.0.0.0-12345678   VMW  VMwareCertified  2026-01-15
nsx-platform-client    4.2.0.0.0-12345678   VMW  VMwareCertified  2026-01-15
nsx-proxy              4.2.0.0.0-12345678   VMW  VMwareCertified  2026-01-15
```

Remediation if agents not running:
1. Restart MPA: `/etc/init.d/nsx-mpa restart`
2. Restart proxy: `/etc/init.d/nsx-proxy restart`
3. Check logs: `/var/log/nsx-mpa.log` and `/var/log/nsx-syslog.log`
4. If persistent: Re-prepare host via NSX Manager UI → Fabric → Host Transport Nodes → Resolve

8.4 Port Connectivity

What: Verify ESXi transport nodes can reach NSX Managers on required ports.

Port Requirements

Source	Destination	Port	Protocol	Purpose
ESXi	NSX Manager	1234	TCP	NSX Manager → Host MPA
ESXi	NSX Manager	1235	TCP	NSX Central CLI
ESXi	NSX Manager	5671	TCP	Message bus (RabbitMQ)
ESXi	NSX Manager	443	TCP	HTTPS API
ESXi	ESXi (TEP)	4789	UDP	Geneve overlay
ESXi	ESXi (TEP)	6081	UDP	Geneve (BFD)

Connectivity Test from ESXi

```
# Test connectivity to NSX Manager ports from ESXi
nc -zv $NSX_VIP 1234
nc -zv $NSX_VIP 1235
nc -zv $NSX_VIP 5671
nc -zv $NSX_VIP 443
```

Expected Output:

```
Connection to nsx-vip.lab.local 1234 port [tcp/*] succeeded!
Connection to nsx-vip.lab.local 1235 port [tcp/*] succeeded!
Connection to nsx-vip.lab.local 5671 port [tcp/*] succeeded!
Connection to nsx-vip.lab.local 443 port [tcp/https] succeeded!
```

9. TEP Connectivity

9.1 vmkping Tests with MTU Validation

What: Verify Tunnel Endpoint (TEP) connectivity between all ESXi hosts with proper MTU.

Why: TEP connectivity is the foundation of NSX overlay networking. Failed TEP = no overlay connectivity between VMs on different hosts.

Standard vmkping (Default MTU)

```
# From ESXi host, ping another host's TEP IP
vmkping -I vmk10 -d -s 1572 <remote-TEP-IP>
```

MTU Note: The `-s 1572` flag sets the payload size. With IP (20 bytes) and ICMP (8 bytes) headers, total packet size = 1600 bytes, which is the minimum MTU for Geneve overlay. The `-d` flag sets the Don't Fragment bit to ensure the full MTU path is tested.

Test All TEP Pairs

```
# Script to test TEP connectivity from current host
REMOTE_TEPS="192.168.12.75 192.168.12.76 192.168.12.82"
for TEP in $REMOTE_TEPS; do
  echo "Testing TEP: $TEP"
  vmkping -I vmk10 -d -s 1572 -c 3 $TEP
  echo "---"
done
```

Expected Output (Healthy):

```
PING 192.168.12.75 (192.168.12.75): 1572 data bytes
1580 bytes from 192.168.12.75: icmp_seq=0 ttl=64 time=0.543 ms
1580 bytes from 192.168.12.75: icmp_seq=1 ttl=64 time=0.421 ms
1580 bytes from 192.168.12.75: icmp_seq=2 ttl=64 time=0.389 ms
```

```
--- 192.168.12.75 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
```

Pass / Warn / Fail

Result	Criteria	Indicator
PASS	0% packet loss with MTU 1600	TEP healthy
WARN	Intermittent loss (< 5%) or high latency (> 5ms)	Network issue
FAIL	100% loss or MTU 1600 fails (but smaller succeeds)	TEP broken or MTU mismatch

Remediation for MTU failures:
1. Check physical switch MTU: Must be ≥ 1700 (Geneve adds 50-byte overhead)
2. Check vDS MTU: `esxcli network vswitch dvs vmware list` — MTU should be 9000 or at least 1600
3. Check TEP VLAN: Ensure TEP VLAN is trunked on all switch ports
4. Test with smaller MTU to isolate: `vmkping -I vmk10 -d -s 1400 <TEP>`

9.2 BFD Session Verification

What: Verify Bidirectional Forwarding Detection (BFD) sessions between transport nodes.

Why: BFD provides fast failure detection for TEP tunnels. Down BFD = NSX considers the tunnel failed.

CLI Method (nsxcli on ESXi)

```
ssh root@<esxi-host>
nsxcli
> get bfd-sessions
```

Expected Output:

```
BFD Session          Remote IP        State   Diagnostic
-----------------------------------------------------
192.168.12.75        192.168.12.75    UP      No Diagnostic
192.168.12.76        192.168.12.76    UP      No Diagnostic
192.168.12.82        192.168.12.82    UP      No Diagnostic
```

Pass / Warn / Fail

Result	Criteria	Indicator

PASS	All BFD sessions `UP`	Healthy
WARN	Any session in `INIT` state	Establishing
FAIL	Any session `DOWN` or missing	Tunnel failure

9.3 Tunnel Status

What: Verify overlay tunnel status between transport nodes.

nsxcli Method

```
> get logical-switches
```

Expected Output:

```
Logical Switch UUID                            VNI     Port Count
-------------------------------------------------------------------
abc123-def456-789012-abcdef-123456789012    71001   5
def456-789012-abcdef-123456-789012abcdef    71002   3
```

```
> get tunnel-ports
```

Expected Output:

```
Tunnel Port  Remote IP        Encap  Status
-----------------------------------------
vxlan_12     192.168.12.75    Geneve UP
vxlan_13     192.168.12.76    Geneve UP
vxlan_14     192.168.12.82    Geneve UP
```

10. Edge Node Health

10.1 Edge Cluster Status

What: Verify the Edge cluster and all member Edge nodes are operational.

Why: Edge nodes provide north-south routing, NAT, load balancing, VPN, and gateway firewall services. Edge failure = services outage.

API Method

```
# List edge clusters
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/api/v1/edge-clusters" | jq '.results[] | {
    id: .id,
    display_name: .display_name,
    member_count: (.members | length),
    deployment_type: .deployment_type
  }'
```

```
# Get edge cluster status
EDGE_CLUSTER_ID="<edge-cluster-id>"
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/api/v1/edge-clusters/$EDGE_CLUSTER_ID/status" | jq .
```

Expected Output:

```
{
  "edge_cluster_id": "abc123...",
  "member_status": [
    {
      "member_index": 0,
      "transport_node_id": "tn-edge-01",
      "status": "UP"
    },
    {
      "member_index": 1,
```

```
      "transport_node_id": "tn-edge-02",
      "status": "UP"
    }
  ]
}
```

10.2 HA State Verification

What: Verify Edge High Availability is correctly configured with proper ACTIVE/STANDBY states.

API Method — Tier-0 HA Status

```
# Get Tier-0 gateways
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/policy/api/v1/infra/tier-0s" | jq '.results[] | {
    id: .id,
    display_name: .display_name,
    ha_mode: .ha_mode,
    failover_mode: .failover_mode
  }'
```

CLI Method (SSH to Edge Node)

```
ssh admin@<edge-node>
get high-availability status
```

Expected Output:

```
Active-Standby Status
  Service Router    : ACTIVE
  Distributed Router: ACTIVE
  HA Channel State  : UP
  Peer Status       : STANDBY (edge-02)
```

Pass / Warn / Fail

Result	Criteria	Indicator

PASS	One edge `ACTIVE`, one `STANDBY`, HA channel `UP`	Healthy HA
WARN	HA channel `DOWN` but roles correct	HA monitoring degraded
FAIL	Both edges `STANDBY` or both `ACTIVE` (split-brain)	HA failure

Split-Brain Warning: If both edges are ACTIVE, this is a split-brain condition. Traffic may be blackholed. Immediate remediation: restart HA on the secondary edge: `restart service high-availability`

10.3 Service Channels

What: Verify datapath and routing services on edge nodes.

CLI Method (on Edge Node)

```
ssh admin@<edge-node>
get service status
```

Expected Output:

```
Service          Status
--------------------------
dataplane        running
edge-health-agg  running
nestdb           running
nsx-edge-mpa     running
nsx-proxy        running
syslog           running
```

```
get interfaces
```

Expected Output:

```
Interface    IP Address         MAC               MTU   Admin Status
----------------------------------------------------------------------
fp-eth0      192.168.10.1/24    00:50:56:xx:xx:01 1500  UP
fp-eth1      192.168.20.1/24    00:50:56:xx:xx:02 1500  UP
fp-eth2      169.254.0.1/28     02:50:56:xx:xx:03 1500  UP
```

11. Routing & BGP Health

11.1 BGP Neighbor Summary

What: Verify BGP sessions with upstream physical routers are established and routes are being exchanged.

Why: BGP peering is critical for north-south traffic flow. Failed BGP = no external connectivity.

CLI Method (on Edge Node)

```
ssh admin@<edge-node>
get bgp neighbor summary
```

Expected Output:

```
BGP Summary:
  Router ID: 192.168.10.1
  Local AS:  65001

Neighbor        AS      State        Up/Down   Prefixes Received
-----------------------------------------------------------------
192.168.10.254  65000   Established  5d12h     12
192.168.20.254  65000   Established  5d12h     12
```

API Method

```
# Get BGP neighbors via policy API
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/policy/api/v1/infra/tier-0s/<tier0-id>/locale-
services/<locale-service-id>/bgp/neighbors/status" | jq .
```

Pass / Warn / Fail

Result	Criteria	Indicator
PASS	All neighbors `Established`, prefixes received > 0	BGP healthy
WARN	Any neighbor `OpenConfirm` or `OpenSent`	BGP negotiating

FAIL	Any neighbor `Idle` or `Active`	BGP session down

BGP Troubleshooting:
1. Check BGP config: `get bgp config`
2. Check route advertisements: `get bgp neighbor <IP> advertised-routes`
3. Check received routes: `get bgp neighbor <IP> received-routes`
4. Verify physical router config (AS number, peer IP, password if MD5)
5. Check firewall: TCP port 179 must be open between peers

11.2 Route Table Verification

What: Verify the routing table contains expected routes for all network segments.

CLI Method (Edge Node)

```
get route-table
```

Expected Output (abbreviated):

```
Flags: c - connected, s - static, b - BGP, ns - nsx_static
  > - selected route, * - FIB route

b>*  0.0.0.0/0          via 192.168.10.254    fp-eth0   [20/0]
c>*  192.168.10.0/24    directly connected    fp-eth0
c>*  192.168.20.0/24    directly connected    fp-eth1
b>*  10.0.0.0/8         via 192.168.10.254    fp-eth0   [20/0]
ns>* 172.16.0.0/16      via 169.254.0.2       linked    [3/0]
ns>* 172.17.0.0/16      via 169.254.0.2       linked    [3/0]
```

Verify Expected Routes

Route	Source	Expected Next Hop
Default (0.0.0.0/0)	BGP	Physical router IP
Management network	Connected	Local interface
Overlay segments	NSX static	DR backplane
External networks	BGP	Physical router IP

11.3 Gateway Realized State

What: Verify Tier-0 and Tier-1 gateways are fully realized (configured intent matches actual state).

API Method — Tier-0

```
# Check Tier-0 realized state
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/policy/api/v1/infra/tier-0s/<tier0-id>/state" | jq '{
    state: .state,
    details: .details
  }'
```

API Method — Tier-1

```
# Check Tier-1 realized state
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/policy/api/v1/infra/tier-1s" | jq '.results[] | {
    display_name: .display_name,
    id: .id
  }'

# Then for each Tier-1:
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/policy/api/v1/infra/tier-1s/<tier1-id>/state" | jq '{
    state: .state,
    details: .details
  }'
```

Pass / Warn / Fail

Result	Criteria	Indicator
PASS	All gateways `state: REALIZED`	Configuration applied
WARN	Any gateway `state: IN_PROGRESS`	Being configured
FAIL	Any gateway `state: ERROR`	Configuration failure

12. Distributed Firewall Health

12.1 DFW Rule Realization Status

What: Verify all DFW security policies and rules have been successfully realized (pushed) to transport nodes.

Why: Unrealized rules mean security policies are not being enforced, creating security gaps.

API Method

```
# Check DFW realization status
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/policy/api/v1/infra/realized-state/status?
intent_path=/infra/domains/default/security-policies" | jq '{
    publish_status: .publish_status,
    consolidated_status: .consolidated_status.consolidated_status
  }'
```

Detailed Status Per Policy

```
# List all security policies and their realization state
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/policy/api/v1/infra/domains/default/security-policies" | jq
'.results[] | {
    display_name: .display_name,
    id: .id,
    rule_count: (.rules | length)
  }'
```

```
# Check realization for a specific policy
POLICY_ID="<policy-id>"
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/policy/api/v1/infra/realized-state/status?
intent_path=/infra/domains/default/security-policies/$POLICY_ID" | jq .
```

Pass / Warn / Fail

Result	Criteria	Indicator
PASS	All policies `REALIZED`	Rules enforced on all hosts
WARN	Any policy `IN_PROGRESS`	Rules being pushed
FAIL	Any policy `ERROR`	Rules NOT enforced — security gap

12.2 ESXi dvfilter Verification

What: Verify DFW filters are attached to VM vNICs on ESXi hosts.

Why: If dvfilter is not attached, DFW rules are not applied to that VM regardless of realization status.

CLI Method (on ESXi host)

```
# List all dvfilters
summarize-dvfilter
```

Expected Output (abbreviated):

```
world 12345678 vmm0:MyVM vcUuid:'50 12 ab cd ...'
 ++  port 50331651 myvm.eth0
    vNic slot 2
    name: nic-12345678-eth0-vmware-sfw.2
    agentName: vmware-sfw
    state: IOChain Attached
```

Key indicators:
- `agentName: vmware-sfw` — confirms DFW is active
- `state: IOChain Attached` — confirms filter is operational
- Each VM vNIC should have a corresponding dvfilter entry

12.3 vsipioctl Commands

What: Verify DFW rules are programmed on the ESXi datapath using vsipioctl.

Get Filter List

```
vsipioctl getfilters
```

Expected Output:

```
Filter Name                              : nic-12345678-eth0-vmware-sfw.2
Filter Rules:
  Rule Count  : 47
  Applied To  : myvm.eth0
```

View Rules for a Specific Filter

```
vsipioctl getrules -f nic-12345678-eth0-vmware-sfw.2
```

Expected Output (abbreviated):

```
ruleset domain-c8:1001 {
  # DFW Section: Application Rules
  rule 1001 at 1 inout protocol any from any to any accept with log;
  rule 1002 at 2 inout protocol tcp from addrset src-001 to addrset dst-001 port 443
accept;
  rule 1003 at 3 inout protocol any from any to any drop;
}
```

View Address Sets

```
vsipioctl getaddrsets -f nic-12345678-eth0-vmware-sfw.2
```

13. Gateway Firewall Health

What: Verify Gateway Firewall rules on Tier-0 and Tier-1 gateways are realized and synchronized between HA edge pairs.

Why: Gateway firewall protects north-south traffic. Desynchronized rules between active/standby edges create intermittent security gaps during failover.

API Method — Gateway Firewall Status

```
# Tier-0 Gateway Firewall policies
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/policy/api/v1/infra/tier-0s/<tier0-id>/gateway-firewall" | jq
.

# Tier-1 Gateway Firewall policies
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/policy/api/v1/infra/tier-1s/<tier1-id>/gateway-firewall" | jq
.
```

Check Rule Realization on Edge

```
# SSH to edge node
ssh admin@<edge-node>
get firewall rules
```

Expected Output:

```
Rule ID    Action   Direction  Source          Destination     Service    Logged
-------------------------------------------------------------------------------
1001       allow    in-out     10.0.0.0/8      any             HTTPS      yes
1002       allow    in-out     any             10.0.0.0/8      SSH        yes
2001       drop     in-out     any             any             any        yes
```

Verify Sync Between Active/Standby

```
# On ACTIVE edge:
get firewall rules | wc -l

# On STANDBY edge:
get firewall rules | wc -l
```

The rule count should be identical on both edges.

Pass / Warn / Fail

Result	Criteria	Indicator
PASS	Rules realized, rule count matches between HA pair	Healthy
WARN	Rule count differs by 1-2 (possible in-flight update)	Monitor
FAIL	Significant rule count mismatch or rules not realized	Security gap

14. Segments & Logical Switches

14.1 Segment Status

What: Verify all NSX segments are operational and correctly realized.

API Method

```
# List all segments with status
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/policy/api/v1/infra/segments" | jq '.results[] | {
    display_name: .display_name,
    id: .id,
    type: .type,
    vlan_ids: .vlan_ids,
    transport_zone_path: .transport_zone_path,
    admin_state: .admin_state
  }'
```

Check Segment Realized State

```
# For each segment, check realization
SEGMENT_ID="<segment-id>"
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/policy/api/v1/infra/segments/$SEGMENT_ID/state" | jq '{
    state: .state,
    details: .details
  }'
```

Pass / Warn / Fail

Result	Criteria	Indicator
PASS	All segments `admin_state: UP`, realized `SUCCESS`	Healthy
WARN	Any segment `IN_PROGRESS`	Being configured
FAIL	Any segment `ERROR` or `admin_state: DOWN`	Connectivity loss

14.2 MAC / ARP / VTEP Tables

What: Verify logical switch forwarding tables are populated correctly.

nsxcli on ESXi

```
nsxcli
> get logical-switch <VNI> mac-table
```

Expected Output:

```
Inner MAC          Outer MAC          Outer IP         Flags
---------------------------------------------------------------
00:50:56:aa:bb:01  00:50:56:cc:dd:01  192.168.12.75   L
00:50:56:aa:bb:02  00:50:56:cc:dd:02  192.168.12.76   R
```

- **L** = Local (VM is on this host)
- **R** = Remote (VM is on another host, reachable via VTEP)

```
> get logical-switch <VNI> arp-table
```

Expected Output:

```
IP Address       MAC Address        Flags
----------------------------------------
172.16.1.10      00:50:56:aa:bb:01  L
172.16.1.11      00:50:56:aa:bb:02  R
```

```
> get logical-switch <VNI> vtep-table
```

Expected Output:

```
VTEP IP           VTEP MAC             Segment ID  Is Local
-------------------------------------------------------------
192.168.12.74     00:50:56:66:xx:01    192.168.12.0  true
```

```
192.168.12.75     00:50:56:66:xx:02    192.168.12.0  false
192.168.12.76     00:50:56:66:xx:03    192.168.12.0  false
```

15. NSX Alarms

What: Check for active alarms on the NSX Manager that indicate current or impending issues.

Why: NSX generates alarms for certificate expiry, resource exhaustion, connectivity failures, and more. Unresolved alarms indicate health issues.

API Method — All Active Alarms

```
# Get all open alarms
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/api/v1/alarms?status=OPEN,ACKNOWLEDGED&sort_by=severity" | jq
'.results[] | {
    alarm_id: .id,
    severity: .severity,
    feature: .feature_name,
    event_type: .event_type,
    description: .description,
    entity: .entity_id,
    last_reported: .last_reported_time,
    status: .status
  }'
```

Filter Critical Alarms Only

```
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/api/v1/alarms?status=OPEN&severity=CRITICAL" | jq
'.result_count'
```

Key Alarm Categories

Category	Feature Name	Example Alarm
Cluster	`clustering`	Manager node connectivity lost
Certificate	`trust_management`	Certificate expiring soon
Transport	`transport_node`	Transport node connection down
Edge	`edge`	Edge node unhealthy

Routing	`routing`	BGP neighbor down
Firewall	`firewall`	DFW realization failure
Backup	`backup`	Backup failure
Resource	`system`	Disk space critical

Pass / Warn / Fail

Result	Criteria	Indicator
PASS	0 critical alarms, 0 high-severity alarms	Clean
WARN	Warning-severity alarms present	Review recommended
FAIL	Critical or high-severity alarms present	Immediate action

Acknowledge / Resolve Alarms via API

```
# Acknowledge an alarm
curl -sk -X POST -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/api/v1/alarms/<alarm-id>?
action=set_status&new_status=ACKNOWLEDGED"

# Resolve an alarm (after fixing the root cause)
curl -sk -X POST -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/api/v1/alarms/<alarm-id>?
action=set_status&new_status=RESOLVED"
```

16. Port Reference Table

NSX Manager Ports

Source	Destination	Port	Protocol	Purpose
Admin Workstation	NSX Manager	443	TCP	Web UI / API
Admin Workstation	NSX Manager	22	TCP	SSH CLI
NSX Manager	NSX Manager	443	TCP	Inter-node API
NSX Manager	NSX Manager	5671	TCP	Messaging (AMQP)
NSX Manager	NSX Manager	9000	TCP	Cluster bootstrap
NSX Manager	NSX Manager	9200	TCP	Corfu datastore
NSX Manager	NSX Manager	9300	TCP	Search engine
NSX Manager	vCenter	443	TCP	Compute Manager
NSX Manager	vCenter	902	TCP	VM console proxy
NSX Manager	DNS	53	TCP/UDP	Name resolution
NSX Manager	NTP	123	UDP	Time sync
NSX Manager	Syslog	514/6514	UDP/TCP	Log forwarding
NSX Manager	SFTP Backup	22	TCP	Backup file transfer
NSX Manager	LDAP/AD	389/636	TCP	Authentication

Transport Node (ESXi) Ports

Source	Destination	Port	Protocol	Purpose
ESXi	NSX Manager	1234	TCP	MPA communication
ESXi	NSX Manager	1235	TCP	Central CLI
ESXi	NSX Manager	5671	TCP	Message bus
ESXi	NSX Manager	443	TCP	API / certificate
ESXi	ESXi (TEP)	4789	UDP	Geneve overlay
ESXi	ESXi (TEP)	6081	UDP	Geneve BFD
NSX Manager	ESXi	443	TCP	Host preparation

Edge Node Ports

Source	Destination	Port	Protocol	Purpose
Edge	NSX Manager	1234	TCP	MPA
Edge	NSX Manager	1235	TCP	Central CLI
Edge	NSX Manager	5671	TCP	Message bus
Edge	NSX Manager	443	TCP	API
Edge	ESXi (TEP)	4789	UDP	Geneve overlay
Edge	Edge	4789	UDP	Inter-edge overlay
Edge	Physical Router	179	TCP	BGP peering
Edge	Physical Router	ICMP	—	BFD (if configured)

17. Common Issues & Remediation

17.1 Control Plane Connectivity Issues

Symptom	Likely Cause	Resolution
Transport node shows `DOWN`	NSX Manager unreachable from host	Check port 1234/5671 connectivity; restart `nsx-mpa`
`get controllers` shows `Disconnected`	Controller service issue	Restart controller: `restart service controller` on NSX Manager
Multiple nodes disconnecting	Network partition	Check upstream switch/VLAN; verify management network

17.2 Transport Node Preparation Failures

Symptom	Likely Cause	Resolution
Host stuck in `IN_PROGRESS`	VIB installation hung	Reboot ESXi host; re-trigger preparation
`FAILED` with "VIB conflict"	Incompatible third-party VIB	Remove conflicting VIB; check compatibility matrix
`FAILED` with certificate error	Clock skew	Verify NTP on ESXi and NSX Manager

17.3 Certificate Expiry Impacts

Certificate Type	Impact When Expired	Recovery
NSX Manager API	API calls fail, UI inaccessible	Replace via CLI: `set certificate api`
Cluster certificate	Inter-node communication fails	Replace per VMware KB 83054
Transport node	Host loses control plane	Replace and re-prepare host

17.4 Cluster Split-Brain

Symptoms:

- Two NSX Manager groups each claiming to be leader
- Conflicting configuration states
- Intermittent API responses with different data

Resolution:

1. Identify the minority partition (fewer nodes)

2. Power off minority nodes
3. Verify majority cluster stabilizes
4. Power on minority nodes one at a time
5. Monitor cluster rejoin via `get cluster status`

> **Warning:** Never force-delete a cluster node during split-brain. This can cause permanent data loss. Always follow the VMware documented procedure.

18. CLI Quick Reference Card

NSX Manager CLI (SSH as admin)

Command	Purpose
`get cluster status`	Cluster health overview
`get services`	All service status
`get node-stats`	CPU/memory/disk stats
`get filesystem-info`	Disk utilization
`get certificate api`	API certificate details
`get managers`	Connected manager nodes
`get controllers`	Connected controllers
`get interface eth0`	Network interface info
`get log-file syslog follow`	Tail syslog in real-time
`restart cluster boot-manager`	Restart cluster membership
`restart service <name>`	Restart a specific service
`set logging-level <service> level debug`	Enable debug logging
`clear logging-level <service>`	Reset to default logging

ESXi nsxcli Commands

Command	Purpose
`get managers`	NSX Manager connectivity
`get controllers`	Controller connectivity
`get logical-switches`	Overlay segments on host
`get logical-switch <VNI> mac-table`	MAC forwarding table
`get logical-switch <VNI> arp-table`	ARP table
`get logical-switch <VNI> vtep-table`	VTEP table
`get bfd-sessions`	BFD tunnel status

`get tunnel-ports`	Geneve tunnel endpoints
`get firewall rules`	DFW rules (on edge)

ESXi Host Commands (as root)

Command	Purpose
`/etc/init.d/nsx-mpa status`	MPA agent status
`/etc/init.d/nsx-proxy status`	Proxy agent status
`/etc/init.d/nsx-mpa restart`	Restart MPA
`summarize-dvfilter`	DFW filter summary
`vsipioctl getfilters`	DFW filter list
`vsipioctl getrules -f <filter>`	DFW rules for a filter
`vsipioctl getaddrsets -f <filter>`	Address sets
`esxcli software vib list | grep nsx`	Installed NSX VIBs
`vmkping -I vmk10 -d -s 1572 <IP>`	TEP MTU test

Edge Node CLI

Command	Purpose
`get high-availability status`	HA state
`get bgp neighbor summary`	BGP sessions
`get route-table`	Routing table
`get interfaces`	Network interfaces
`get service status`	Running services
`get firewall rules`	Gateway FW rules
`get arp-table`	ARP entries

19. API Quick Reference

Authentication

```
# Basic authentication (used throughout this document)
curl -sk -u "admin:password" "https://$NSX_VIP/api/v1/..."

# Session-based authentication
curl -sk -c cookies.txt -X POST \
  -d '{"j_username":"admin","j_password":"password"}' \
  "https://$NSX_VIP/api/session/create"

# Subsequent calls with session cookie
curl -sk -b cookies.txt "https://$NSX_VIP/api/v1/..."
```

Key API Endpoints

Endpoint	Method	Purpose
`/api/v1/cluster/status`	GET	Cluster health
`/api/v1/cluster/nodes`	GET	Node inventory
`/api/v1/node`	GET	Local node info
`/api/v1/node/status`	GET	Node resource usage
`/api/v1/node/services`	GET	Service status
`/api/v1/trust-management/certificates`	GET	All certificates
`/api/v1/cluster/backups/config`	GET	Backup configuration
`/api/v1/cluster/backups/history`	GET	Backup history
`/api/v1/transport-nodes/state`	GET	Transport node status
`/api/v1/edge-clusters`	GET	Edge clusters
`/api/v1/edge-clusters/<id>/status`	GET	Edge cluster status
`/api/v1/alarms`	GET	Active alarms
`/policy/api/v1/infra/tier-0s`	GET	Tier-0 gateways

`/policy/api/v1/infra/tier-1s`	GET	Tier-1 gateways
`/policy/api/v1/infra/segments`	GET	Network segments
`/policy/api/v1/infra/domains/default/security-policies`	GET	DFW policies
`/policy/api/v1/infra/realized-state/status`	GET	Realization status

Common API Query Parameters

Parameter	Example	Purpose
`cursor`	`?cursor=0`	Pagination start
`page_size`	`?page_size=100`	Results per page
`sort_by`	`?sort_by=display_name`	Sort field
`sort_ascending`	`?sort_ascending=true`	Sort order
`included_fields`	`?included_fields=display_name,id`	Limit response fields

Version Check

```
# Get NSX Manager version
curl -sk -u "$NSX_USER:$NSX_PASS" \
  "https://$NSX_VIP/api/v1/node/version" | jq .
```

Expected Output:

```
{
  "node_version": "4.2.0.0.0.12345678",
  "product_version": "4.2.0.0.0.12345678"
}
```