**Approvals**

|  |  |  |
| --- | --- | --- |
| Prepared by: |  | Reviewed by: |
| Klaus Seggelke  Sr. Solution Architect Manager |  | <Name>  <Title> |
|  |  |  |
| <Name>  <Title> |  | <Name>  <Title> |
|  |  |  |
| <Name>  <Title> |  | <Name>  <Title> |

# Table of Contents

[1 Table of Contents i](#_Toc477880799)

[2 Introduction 4](#_Toc477880800)

[2.1 Overview 4](#_Toc477880801)

[2.2 Objectives 6](#_Toc477880802)

[2.3 Test bed setup 7](#_Toc477880803)

[2.3.1 Single Transient VNF 7](#_Toc477880804)

[2.3.2 Multi Transient VNF 8](#_Toc477880805)

[2.3.3 Server based VNF 9](#_Toc477880806)

[2.3.4 VNF Lifecycle Verification Architecture 10](#_Toc477880807)

[2.3.5 MANO API Implementation Options 12](#_Toc477880808)

[2.3.6 MANO API Implementation Considerations 12](#_Toc477880809)

[3 VNF Lifecycle Management Overview 14](#_Toc477880810)

[4 VNF Packaging 18](#_Toc477880811)

[4.1 VNF Packaging Overview 18](#_Toc477880812)

[4.2 Levels of NFV Entities 18](#_Toc477880813)

[4.3 VNF Package Lifecycle 19](#_Toc477880814)

[4.3.1 VNF Package bundling for distribution by VNF Provider 19](#_Toc477880815)

[4.3.2 VNF Package testing by VNF Provider 19](#_Toc477880816)

[4.3.3 VNF pre procurement by Service Provider 19](#_Toc477880817)

[4.3.4 VNF Package validation and certification by Service Provider 20](#_Toc477880818)

[4.3.5 VNF Install or Deployment 20](#_Toc477880819)

[5 Software Image Management 21](#_Toc477880820)

[6 Life-Cycle Management Information Retrieval 24](#_Toc477880821)

[7 Snapshot Management 25](#_Toc477880822)

[8 VNF Descriptor 28](#_Toc477880823)

[8.1 VNFD Base Unit 31](#_Toc477880824)

[8.2 VNFD Virtual Deployment Unit (VDU) 33](#_Toc477880825)

[8.2.1 VNFD:VDU Base Unit 33](#_Toc477880826)

[8.2.2 VNFD:VDU VNFC Elements 35](#_Toc477880827)

[8.2.3 VNFD:VDU CPU 35](#_Toc477880828)

[8.2.4 VNFD:VDU Memory 37](#_Toc477880829)

[8.2.5 VNFD:VDU Security 38](#_Toc477880830)

[8.2.6 VNFD:VDU HyperVisor 38](#_Toc477880831)

[8.2.7 VNFD:VDU Network Elements 39](#_Toc477880832)

[8.2.8 VNFD:VDU Virtual Switches 40](#_Toc477880833)

[8.2.9 VNFD:VDU General Reliability and Availability 40](#_Toc477880834)

[8.2.10 VNFD:VDU Storage 41](#_Toc477880835)

[8.3 VNFD Virtual Link 41](#_Toc477880836)

[8.4 VNFD Connection Points 42](#_Toc477880837)

[8.5 VNFD Deployment Flavors 43](#_Toc477880838)

[9 VNF State Management 44](#_Toc477880839)

[9.1 VNF State Management: Instantiation 44](#_Toc477880840)

[9.2 VNF Instantiation Failure with missing required virtual resources 50](#_Toc477880887)

[9.3 VNF State Management: Start 54](#_Toc477880888)

[9.4 VNF State Management: Stop 57](#_Toc477880889)

[9.5 VNF State Management: Termination 60](#_Toc477880890)

[9.6 VNF State Management: Upgrade, Update or Rollback 63](#_Toc477880891)

[10 Virtualized Resource Management 66](#_Toc477880892)

[10.1 vResource Scaling Up/Down 67](#_Toc477880893)

[10.1.1 Scale-up VNF instance 67](#_Toc477880894)

[10.1.2 Max scale-up VNF instance 74](#_Toc477880895)

[10.1.3 Scale-down VNF instance 75](#_Toc477880896)

[10.1.4 Scale-Up upto Max vResource limit 83](#_Toc477880897)

[10.1.5 Scale-Up/Down for virtualized specialized hardware acceleration 85](#_Toc477880898)

[10.2 vResource Scaling Out/In of VNFC 86](#_Toc477880899)

[10.2.1 Scale-Out VNFC instance 86](#_Toc477880900)

[10.2.2 Max scale-out VNFC instance 93](#_Toc477880901)

[10.2.3 Scale-in VNFC instance 94](#_Toc477880902)

[10.2.4 Max vResource VNFC limit reached before max VNFD limit for scale-out 102](#_Toc477880903)

[10.2.5 Removal of virtualized specialized hardware acceleration for VNFC scale-in 104](#_Toc477880904)

[10.3 vResource Scaling Out/In of VNF 105](#_Toc477880905)

[10.3.1 Scale-Out VNF instance 105](#_Toc477880906)

[10.3.2 Max scale-out VNF instance 112](#_Toc477880907)

[10.3.3 Scale-in VNF instance 113](#_Toc477880908)

[10.3.4 Max vResource limit reached before max VNFD limit for VNF scale-out 121](#_Toc477880909)

[10.3.5 Removal of virtualized specialized hardware acceleration for VNF scale-in 123](#_Toc477880910)

[10.4 vResource Migration 124](#_Toc477880912)

[10.5 vResource Network 128](#_Toc477880913)

[10.6 vResource Storage 131](#_Toc477880914)

[11 VNF Fault and Recovery Management 133](#_Toc477880915)

[11.1 VNF Fault Management 133](#_Toc477880916)

[11.2 VNF Recovery 139](#_Toc477880917)

[12 Performance Management 140](#_Toc477880918)

[13 Complex Use-cases 147](#_Toc477880919)

[14 Open Use Cases 153](#_Toc477880920)

[15 API Design and Architecture 153](#_Toc477880921)

[15.1 VNF RECORD 154](#_Toc477880922)

[15.2 EM API module 154](#_Toc477880923)

[15.3 NFVO API Module 155](#_Toc477880924)

[15.4 VNF API Module 156](#_Toc477880925)

[15.5 VNFM API Module 157](#_Toc477880926)

[15.5.1 Software Image 158](#_Toc477880927)

[15.5.2 VNFD 159](#_Toc477880928)

[15.5.3 Snapshot 160](#_Toc477880929)

[15.5.4 VNF 164](#_Toc477880930)

[15.5.5 Migration 168](#_Toc477880931)

[15.5.6 vNetwork 169](#_Toc477880932)

[15.5.7 vStorage 169](#_Toc477880933)

[15.5.8 Fault 170](#_Toc477880934)

[15.5.9 Performance 173](#_Toc477880935)

[15.6 VNFD API Module 176](#_Toc477880936)

[15.7 VIM API Module 176](#_Toc477880937)

[15.8 Traffic API Module 182](#_Toc477880938)

[16 Test Case Pseudo Code 185](#_Toc477880939)

[16.1 TC.81 Stop a max scale-up/scaled-out VNF instance in state Active under max traffic load 185](#_Toc477880940)

[17 List of all Functions 190](#_Toc477880941)

[18 References 193](#_Toc477880942)

[19 DESIGN CONSIDERATIONS 193](#_Toc477880943)

[19.1 Troubleshooting Aids 193](#_Toc477880944)

[20 Revision history 194](#_Toc477880945)

TABLE OF Test Cases

[TC.VNF.PKG.001 VNF Package Content Validation 21](#_Toc477880946)

[TC.VNF.PKG.002 VNF Package Onboarding 22](#_Toc477880947)

[TC.VNF.IMAGE.001 Software Image Retrieval with single vendor release without filter 23](#_Toc477880948)

[TC.VNF.IMAGE.002 Software Image Retrieval with single vendor release with filters 23](#_Toc477880949)

[TC.VNF.IMAGE.003 Software Image Retrieval with multi vendor release 24](#_Toc477880950)

[TC.VNF.IMAGE.004 Life-Cycle management information 25](#_Toc477880951)

[TC.VNF.SNAPSHOT.001 Compute Snapshot Creation and Deletion without traffic load 26](#_Toc477880952)

[TC.VNF.SNAPSHOT.002 Compute Snapshot Creation and Deletion under normal traffic load 26](#_Toc477880953)

[TC.VNF.SNAPSHOT.003 Compute Snapshot Instantiation 27](#_Toc477880954)

[TC.VNF.SNAPSHOT.004 Storage Snapshot Creation and Deletion 28](#_Toc477880955)

[TC.VNF.SNAPSHOT.005 Storage Snapshot Instantiation 28](#_Toc477880956)

[TC.VNFD.Coding.001 VNFD XML Coding Validation 31](#_Toc477880957)

[TC.VNFD.Coding.002 VNFD YANG Coding Validation 31](#_Toc477880958)

[TC.VNFD.Base.001 VNFD Base unit validation 32](#_Toc477880959)

[TC.VNFD.Base.002 VNFD validation for VNF release deployment 33](#_Toc477880960)

[TC.VNFD.VDU.001 VNFD:VDU Base Unit validation 34](#_Toc477880961)

[TC.VNFD.VDU.002 VNFD:VDU Base Unit validation 36](#_Toc477880962)

[TC.VNFD.VDU.003 VNFD:VDU CPU Unit validation 36](#_Toc477880963)

[TC.VNFD.VDU.004 VNFD:VDU Memory Unit validation 38](#_Toc477880964)

[TC.VNFD.VDU.005 VNFD:VDU security Unit validation 39](#_Toc477880965)

[TC.VNFD.VDU.006 VNFD:VDU HyperVisor Unit validation 39](#_Toc477880966)

[TC.VNFD.VDU.007 VNFD:VDU Network Elements Unit validation 40](#_Toc477880967)

[TC.VNFD.VDU.008 VNFD:VDU Virtual Switches validation 41](#_Toc477880968)

[TC.VNFD.VDU.009 VNFD:VDU General Reliability and Availability Unit validation 41](#_Toc477880969)

[TC.VNFD.VDU.010 VNFD:VDU Storage Unit validation 42](#_Toc477880970)

[TC.VNFD.VLINK.001 VNFD:VDU Virtual Link validation 42](#_Toc477880971)

[TC.VNFD.ConPoint.001 VNFD Connection Points Unit validation 43](#_Toc477880972)

[TC.VNFD.DEPLOY.001 VNFD Deployment Flavors Unit validation 44](#_Toc477880973)

[TC.VNF.STATE.INST.001 VNF Instantiation with configuration file 45](#_Toc477880974)

[TC.VNF.STATE.INST.002 VNF Instantiation with configuration file under load 46](#_Toc477880975)

[TC.VNF.STATE.INST.003 VNF Instantiation with active Element Management 47](#_Toc477880976)

[TC.VNF.STATE.INST.004 VNF Instantiation with active Element Management under traffic load 48](#_Toc477880977)

[TC.VNF.STATE.INST.005 VNF Instantiation with active Element Management with failed EM configuration 49](#_Toc477880978)

[TC.VNF.STATE.INST.006 VNF Instantiation with inactive Element Management 50](#_Toc477880979)

[TC.VNF.STATE.INST.008 VNF Instantiation Failure with missing required virtual resources 51](#_Toc477880980)

[TC.VNF.STATE.INST.008.1 VNF Instantiation Failure with missing required virtual resources “Incompatible vCPU type” 51](#_Toc477880981)

[TC.VNF.STATE.INST.008.2 VNF Instantiation Failure with missing required virtual resources “Instructions sets” 52](#_Toc477880982)

[TC.VNF.STATE.INST.008.3 VNF Instantiation Failure with missing required virtual resources “Specialized vNIC” 52](#_Toc477880983)

[TC.VNF.STATE.INST.008.4 VNF Instantiation Failure with missing required virtual resources “Hardware Acceleration, e.g. DPDK, SRIOV” 53](#_Toc477880984)

[TC.VNF.STATE.INST.008.5 VNF Instantiation Failure with missing required virtual resources “vMemory size” 53](#_Toc477880985)

[TC.VNF.STATE.INST.008.6 VNF Instantiation Failure with missing required virtual resources “vStorage type” 54](#_Toc477880986)

[TC.VNF.STATE.INST.008.7 VNF Instantiation Failure with missing required virtual resources “vStorage size” 54](#_Toc477880987)

[TC.VNF.STATE.INST.008.8 VNF Instantiation Failure with missing required virtual resources “vCPU counts” 55](#_Toc477880988)

[TC.VNF.STATE.START.001 VNF Start without traffic load 56](#_Toc477880989)

[TC.VNF.STATE.START.002 VNF Start under low traffic load and measure start time 56](#_Toc477880990)

[TC.VNF.STATE.START.003 VNF Start with normal traffic load and measure start time 57](#_Toc477880991)

[TC.VNF.STATE.STOP.001 VNF Stop without traffic load and measure stop time 58](#_Toc477880992)

[TC.VNF.STATE.STOP.002 VNF Stop with low traffic load and measure stop time 59](#_Toc477880993)

[TC.VNF.STATE.STOP.003 VNF Stop with normal traffic load and measure stop time 60](#_Toc477880994)

[TC.VNF.STATE.TERM.001 VNF terminate from VNF state STARTED without traffic load 61](#_Toc477880995)

[TC.VNF.STATE.TERM.002 VNF terminate from VNF state STARTED with low traffic load 61](#_Toc477880996)

[TC.VNF.STATE.TERM.003 VNF terminate from VNF state STARTED with normal traffic load 62](#_Toc477880997)

[TC.VNF.STATE.TERM.004 VNF terminate from state STOPPED without load 63](#_Toc477880998)

[TC.VNF.STATE.TERM.005 VNF terminate from VNF state STOPPED with low traffic load 63](#_Toc477880999)

[TC.VNF.STATE.UPDATE.001 VNF upgrade, update or rollback in VNF state STARTED without traffic 64](#_Toc477881000)

[TC.VNF.STATE.UPDATE.002 VNF upgrade, update or rollback in VNF state STOPPED 65](#_Toc477881001)

[TC.VNF.STATE.UPDATE.004 VNF upgrade, update or rollback in VNF state STARTED with low traffic 66](#_Toc477881002)

[TC.VNF.STATE.UPDATE.005 VNF upgrade, update or rollback in state Active with normal traffic 67](#_Toc477881003)

[TC.VNF.SCALE.UP.001 Scale-up VNF instance 68](#_Toc477881004)

[TC.VNF.SCALE.UP.001.1 Scale-up VNF instance with On-demand Scaling event generated by MANO 69](#_Toc477881005)

[TC.VNF.SCALE.UP.001.2 Scale-up VNF instance with On-demand Scaling event generated by VNF 70](#_Toc477881006)

[TC.VNF.SCALE.UP.001.3 Scale-up VNF instance with On-demand Scaling event generated by VIM 71](#_Toc477881007)

[TC.VNF.SCALE.UP.001.4 Scale-up VNF instance with On-demand Scaling event generated by EM 72](#_Toc477881008)

[TC.VNF.SCALE.UP.001.5 Scale-up VNF instance with Manual Scaling event generated by MANO 72](#_Toc477881009)

[TC.VNF.SCALE.UP.001.6 Scale-up VNF instance with Manual Scaling event generated by VNF 73](#_Toc477881010)

[TC.VNF.SCALE.UP.001.7 Scale-up VNF instance with Manual Scaling event generated by EM 74](#_Toc477881011)

[TC.VNF.SCALE.UP.002 Max scale-up VNF instance 75](#_Toc477881012)

[TC.VNF.SCALE.UP.003 Scale-down VNF instance 76](#_Toc477881013)

[TC.VNF.SCALE.UP.003.1 Scale-down VNF instance with On-demand Scaling event generated by MANO 78](#_Toc477881014)

[TC.VNF.SCALE.UP.003.2 Scale-down VNF instance with On-demand Scaling event generated by VNF 79](#_Toc477881015)

[TC.VNF.SCALE.UP.003.3 Scale-down VNF instance with On-demand Scaling event generated by VIM 80](#_Toc477881016)

[TC.VNF.SCALE.UP.003.4 Scale-down VNF instance with On-demand Scaling event generated by EM 81](#_Toc477881017)

[TC.VNF.SCALE.UP.003.5 Scale-down VNF instance with Manual Scaling event generated by MANO 82](#_Toc477881018)

[TC.VNF.SCALE.UP.003.6 Scale-down VNF instance with Manual Scaling event generated by VNF 82](#_Toc477881019)

[TC.VNF.SCALE.UP.003.6 Scale-down VNF instance with Manual Scaling event generated by EM 83](#_Toc477881020)

[TC.VNF.SCALE.UP.004 Max vResource limit reached before max VNFD limit for Scale-Up 84](#_Toc477881021)

[TC.VNF.SCALE.UP.005 Removal of virtualized specialized hardware acceleration for Scale-Up 86](#_Toc477881022)

[TC.VNFC.SCALE.OUT.001 Scale-Out VNFC instance 87](#_Toc477881023)

[TC.VNFC.SCALE.OUT.001.1 Scale-Out VNFC instance with On-demand Scaling event generated by MANO 88](#_Toc477881024)

[TC.VNFC.SCALE.OUT.001.2 Scale-Out VNFC instance with On-demand Scaling event generated by VNF 89](#_Toc477881025)

[TC.VNFC.SCALE.OUT.001.3 Scale-Out VNFC instance with On-demand Scaling event generated by VIM 90](#_Toc477881026)

[TC.VNFC.SCALE.OUT.001.4 Scale-Out VNFC instance with On-demand Scaling event generated by EM 91](#_Toc477881027)

[TC.VNFC.SCALE.OUT.001.5 Scale-Out VNFC instance with Manual Scaling event generated by MANO 92](#_Toc477881028)

[TC.VNFC.SCALE.OUT.001.6 Scale-Out VNFC instance with Manual Scaling event generated by VNF 92](#_Toc477881029)

[TC.VNFC.SCALE.OUT.001.7 Scale-Out VNFC instance with Manual Scaling event generated by EM 93](#_Toc477881030)

[TC.VNFC.SCALE.OUT.002 Max scale-out VNFC instance 94](#_Toc477881031)

[TC.VNFC.SCALE.OUT.003 Scale-in VNFC instance 95](#_Toc477881032)

[TC.VNFC.SCALE.OUT.003.1 Scale-in VNFC instance with On-demand Scaling event generated by MANO 97](#_Toc477881033)

[TC.VNFC.SCALE.OUT.003.2 Scale-in VNFC instance with On-demand Scaling event generated by VNF 98](#_Toc477881034)

[TC.VNFC.SCALE.OUT.003.3 Scale-in VNFC instance with On-demand Scaling event generated by VIM 99](#_Toc477881035)

[TC.VNFC.SCALE.OUT.003.4 Scale-in VNFC instance with On-demand Scaling event generated by EM 100](#_Toc477881036)

[TC.VNFC.SCALE.OUT.003.5 Scale-in VNFC instance with Manual Scaling event generated by MANO 100](#_Toc477881037)

[TC.VNFC.SCALE.OUT.003.6 Scale-in VNFC instance with Manual Scaling event generated by VNF 101](#_Toc477881038)

[TC.VNFC.SCALE.OUT.003.7 Scale-in VNFC instance with Manual Scaling event generated by EM 102](#_Toc477881039)

[TC.VNFC.SCALE.OUT.004 Max vResource VNFC limit reached before max VNFD limit for scale-out 103](#_Toc477881040)

[TC.VNFC.SCALE.OUT.005 Removal of virtualized specialized hardware acceleration for VNFC scale-in 105](#_Toc477881041)

[TC.VNF.SCALE.OUT.001 Scale-Out VNF instance 106](#_Toc477881042)

[TC.VNF.SCALE.OUT.001.1 Scale-Out VNF instance with On-demand Scaling event generated by MANO 107](#_Toc477881043)

[TC.VNF.SCALE.OUT.001.2 Scale-Out VNF instance with On-demand Scaling event generated by VNF 108](#_Toc477881044)

[TC.VNF.SCALE.OUT.001.3 Scale-Out VNF instance with On-demand Scaling event generated by VIM 109](#_Toc477881045)

[TC.VNF.SCALE.OUT.001.4 Scale-Out VNF instance with On-demand Scaling event generated by EM 110](#_Toc477881046)

[TC.VNF.SCALE.OUT.001.5 Scale-Out VNF instance with Manual Scaling event generated by MANO 110](#_Toc477881047)

[TC.VNF.SCALE.OUT.001.6 Scale-Out VNF instance with Manual Scaling event generated by VNF 111](#_Toc477881048)

[TC.VNF.SCALE.OUT.001.7 Scale-Out VNF instance with Manual Scaling event generated by EM 112](#_Toc477881049)

[TC.VNF.SCALE.OUT.002 Max scale-out VNF instance 113](#_Toc477881050)

[TC.VNF.SCALE.OUT.003 Scale-in VNF instance 114](#_Toc477881051)

[TC.VNF.SCALE.OUT.003.1 Scale-in VNF instance with On-demand Scaling event generated by MANO 116](#_Toc477881052)

[TC.VNF.SCALE.OUT.003.2 Scale-in VNF instance with On-demand Scaling event generated by VNF 117](#_Toc477881053)

[TC.VNF.SCALE.OUT.003.3 Scale-in VNF instance with On-demand Scaling event generated by VIM 118](#_Toc477881054)

[TC.VNF.SCALE.OUT.003.4 Scale-in VNF instance with On-demand Scaling event generated by EM 118](#_Toc477881055)

[TC.VNF.SCALE.OUT.003.5 Scale-in VNF instance with Manual Scaling event generated by MANO 119](#_Toc477881056)

[TC.VNF.SCALE.OUT.003.6 Scale-in VNF instance with Manual Scaling event generated by VNF 120](#_Toc477881057)

[TC.VNF.SCALE.OUT.003.7 Scale-in VNF instance with Manual Scaling event generated by EM 121](#_Toc477881058)

[TC.VNF.SCALE.OUT.004 Max vResource limit reached before max VNFD limit for VNF scale-out 122](#_Toc477881059)

[TC.VNF.SCALE.OUT.005 Removal of virtualized specialized hardware acceleration for VNF scale-in 124](#_Toc477881060)

[TC.VNF.MIGRATE.001 Migration of virtual resource with live migration set without traffic load 125](#_Toc477881061)

[TC.VNF.MIGRATE.002 Migration of virtual resource with live migration set under traffic load 126](#_Toc477881062)

[TC.VNF.MIGRATE.003 Migration of virtual resource with offline migration set without traffic load 127](#_Toc477881063)

[TC.VNF.MIGRATE.004 Migration of virtual resource with offline migration set under traffic load 128](#_Toc477881064)

[TC.VNF.VNETWORK.001 vNetworks resources 129](#_Toc477881065)

[TC.VNF.VNETWORK.002 VNFM is able to modify the bandwidth of the vNetwork 130](#_Toc477881066)

[TC.VNF.VSTORAGE.001 Increase of vStorage 132](#_Toc477881067)

[TC.VNF.VSTORAGE.002 Decrease of vStorage 133](#_Toc477881068)

[TC.VNF.FAULT.001 vCPU resource fault notification subscription and notification 134](#_Toc477881069)

[TC.VNF.FAULT.002 Double vCPU resource fault notification subscription and notification 135](#_Toc477881070)

[TC.VNF.FAULT.003 vMemory resource fault notification subscription and notification 137](#_Toc477881071)

[TC.VNF.FAULT.004 vNetwork resource fault notification subscription and notification 138](#_Toc477881072)

[TC.VNF.FAULT.005 vStorage resource fault notification subscription and notification 139](#_Toc477881073)

[TC.VNF.RECOVERY.001 Manual VNF recover of a faulty VNF 140](#_Toc477881074)

[TC.VNF.PERF.001 vCPU resource performance data notification subscription and notification 141](#_Toc477881075)

[TC.VNF.PERF.002 vMemory resource performance data notification subscription and notification 142](#_Toc477881076)

[TC.VNF.PERF.003 vStorage resource performance data notification subscription and notification 143](#_Toc477881077)

[TC.VNF.PERF.004 vCPU resource performance threshold notification subscription and notification 145](#_Toc477881078)

[TC.VNF.PERF.005 vMemory resource performance threshold notification subscription and notification 146](#_Toc477881079)

[TC.VNF.PERF.006 vStorage resource performance threshold notification subscription and notification 147](#_Toc477881080)

[TC.VNF.COMPLEX.001 VNF Start and Scaling with max traffic load 148](#_Toc477881081)

[TC.VNF.COMPLEX.002 Stop a max scale-up/scaled-out VNF instance in state Active under max traffic load 149](#_Toc477881082)

[TC.VNF.COMPLEX.003 Terminate max scale-up/out VNF in state Active under max traffic load 151](#_Toc477881083)

[TC.VNF.COMPLEX.004 VNF upgrade, update or rollback in state Active with max traffic 152](#_Toc477881084)

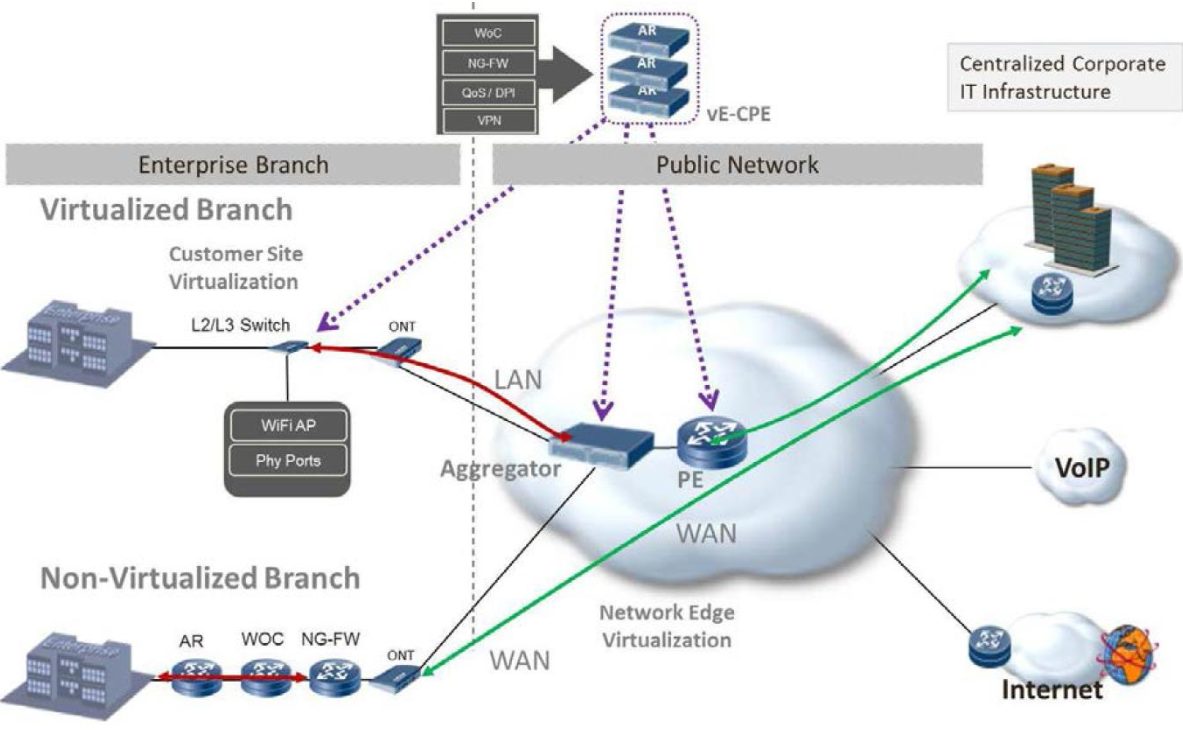
# Introduction

## Overview

Network provider are challenged by customer demands to activate new services quickly. Currently it takes a long time to configure dedicated devices to provide new services. The solution is to transition the multiple network devices from a dedicated hardware device to a software driven eco system by using off-the-self servers. The software solution runs on commonly available servers and provide so much performance that it comes close to the performance of dedicated hardware. A software based solution brings several advantages for the service providers:

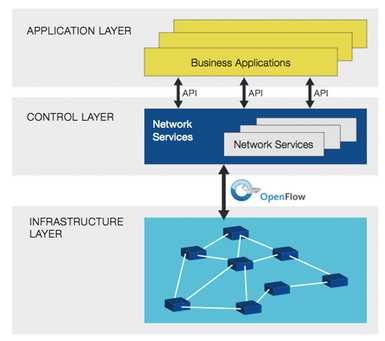
1. Quick setup and fast service delivery to customer
2. Performance and scalability can be dynamic and time bound
3. Fewer hardware devices to maintain and configure
4. Higher hardware usage optimization
5. Fewer service personal
6. Centralized configuration, maintenance and control

The following picture illustrates the NFV impact on service provider network by eliminating dedicated hardware and replacing it with NFV based virtualized functionality. The vE-CPE can be deployed at different locations pending on the required performance and throughput.



To allow competition, exchangeability and undependability of one single provider, a well-define software architecture is required where modules can be added and removed like a Lego system. NFV defines this overall software architecture of such system. It transfers the network functions of dedicated hardware devices to software based applications. This decouples the network functions from proprietary hardware without affecting the functionality.

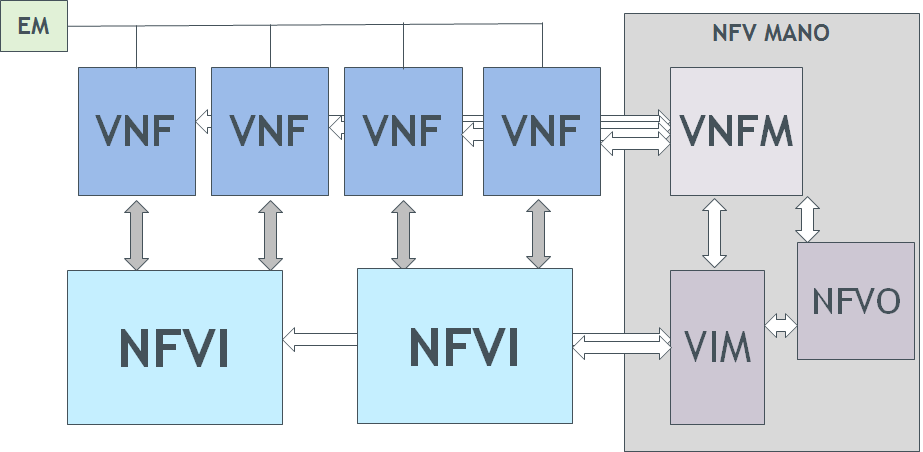
SDN (Service Defined Networks)



SDN (Service Defined Networks) is another architecture that is currently under heavy development and has major impacts already on the current eco system by forcing switch manufacture to expose well defined software interfaces. SDN takes on the actual network functionalities. Its architecture decouples the control plane from the data plane. This allows to create a centralized controller managing the network. The network devices allow programmability and exposing therefore flexibility.

SDN serves NFV by providing the programmable connectivity between NFVs Connections orchestrated by the SDN controller. NFV serves SDN by implementing its network functions in software, allowing SDN controller to be virtualized and run in the cloud, by adapting to needs of the moment. Both combine very well and complement each other.

NFV



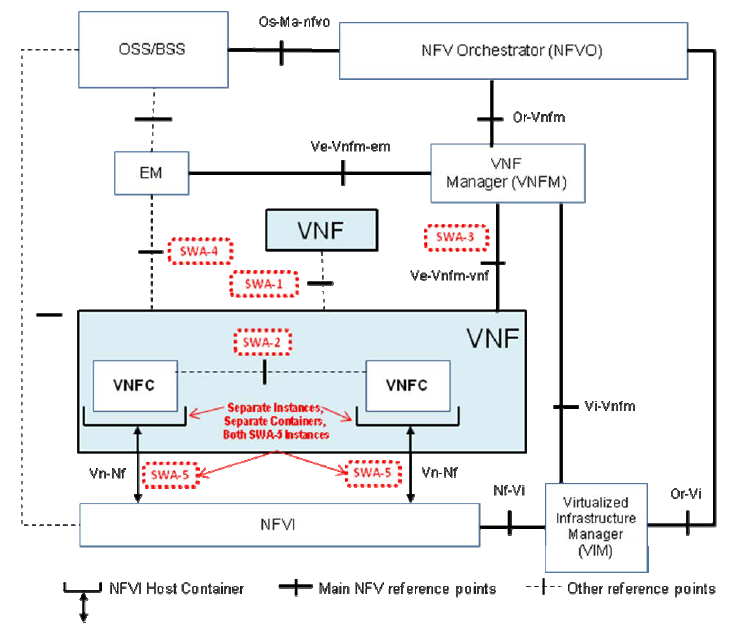
Virtualized Network Function (VNF) runs on NFV Infrastructure (NFVI). A VNF is been orchestrated by NFV Orchestrator (NFVO) and VNF Manager. It has well-defined reference points:

* + SWA-1 – Interface to other VNF
  + SWA-2 – Interface between two VNFC
  + SWA-3 – Interface to VNF manager
  + SWA-4 – Interface to Element Management (EM)
  + SWA-5 – Interface to NFVI

Once instantiated and activated, the VNF provides the required functionality by receiving packets, processing them, may manipulate the packets, may respond to a packet or forward the packet. Therefore, a VNF could be one of the two types:

1. Server VNF (e.g., HTTP server)
2. Transient VNF (e.g., switch, router, firewall)

The following picture illustrates the reference points and the location of the responsibility.



Internal interfaces don’t need to be exposed and can change from one release to another release of the VNF. The VNF provider can divide the software into one or more VNF Components (VNFC). Each VNFC instance is mapped to a NFVI Virtualized Container Interface. Functionality and Interfaces between VNFCs may change over time and considered as internal. Here the vendor has the flexibility to change its internal architecture. Any external interfaces should be kept and manipulated with care.

## Objectives

NFV validation can be divided into three different areas:

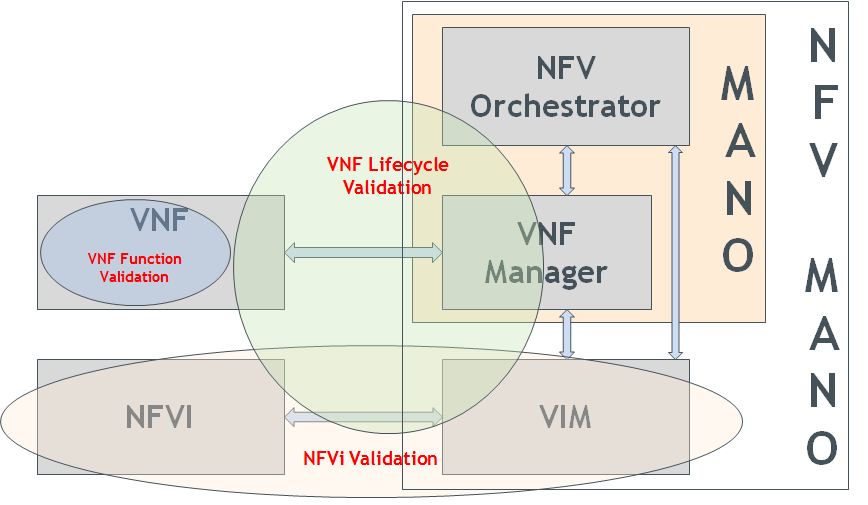
* NFVI validation
* VNF Lifecycle Validation
* VNF Function validation

The **NFVI validation** is focused on the validation of the NFVI layer. It concentrates on the virtual resource management, like vCPU, vMemory, vStorage, vNetwork and virtual image management and its performance.

**VNF Lifecycle Management** focuses on the life cycle management of the VNF. This includes instantiation, termination of a VNF, state management, scaling up/down or in/out.

The **VNF Function** validation main focus is to validate the actual functional operation of the VNF. The VNF functional validation can be divided into following parts:

1. VNF certification of the VNF functionality
2. Benchmark of the VNF functionality



In this document we focus on the VNF Lifecycle Management.

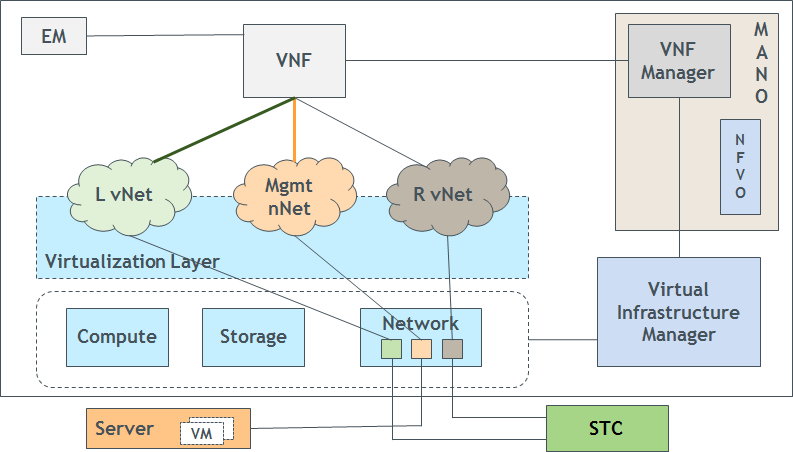
## Test bed setup

The architecture and the functionality of a VNF introduces a wide variety in the considerations of the test bed setup and also the test case execution. Following are examples of test bed setups.

### Single Transient VNF

A single VNF provides functionality by passing packets between two or more network interfaces and applying its manipulation on the packet data if required. Packets flow most likely bidirectional. Examples for this type are:

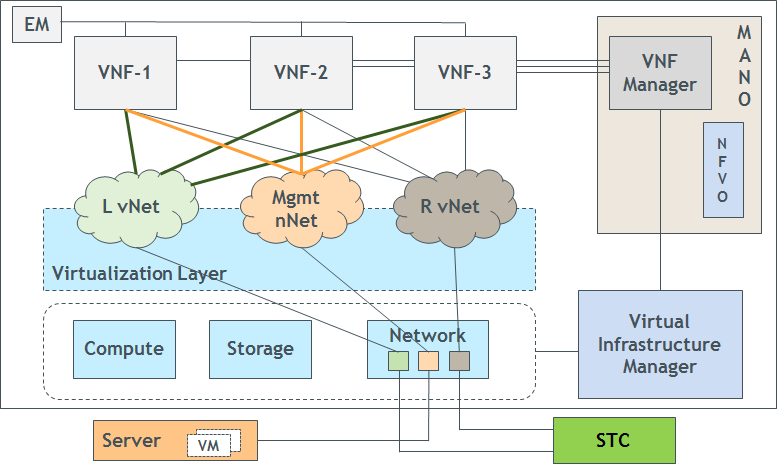
1. Firewall
2. Router
3. Switch
4. NAT/PAT



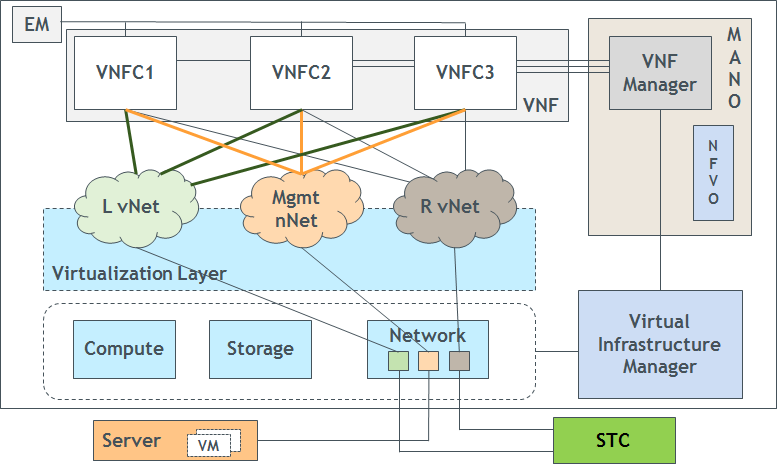
### Multi Transient VNF

A VNF could be composed of multiple VNFCs. All together deliver the functionality. The VNF vendor could also design the functionality as a multi VNF system where multiple VNFs need to be activated. It depends on the vendor’s decision how and why he slices the VNFs. A multi VNF configuration provides many challenges for our system as we need to check many VNFs. For now, we consider that the vendor would not go this path and rather create one VNF with many components. If we see a demand to cover the multi VNF configuration, we can enhance our test cases with the vendor’s input.

* Multi Transient VNF

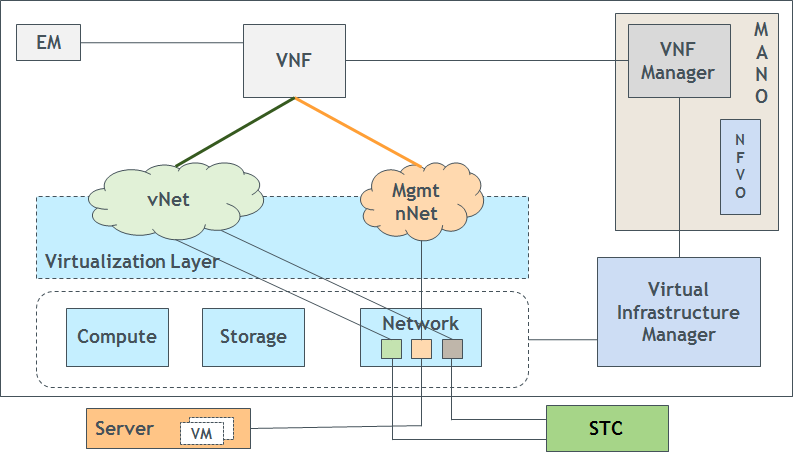


* Transient VNF with multiple VNFCs



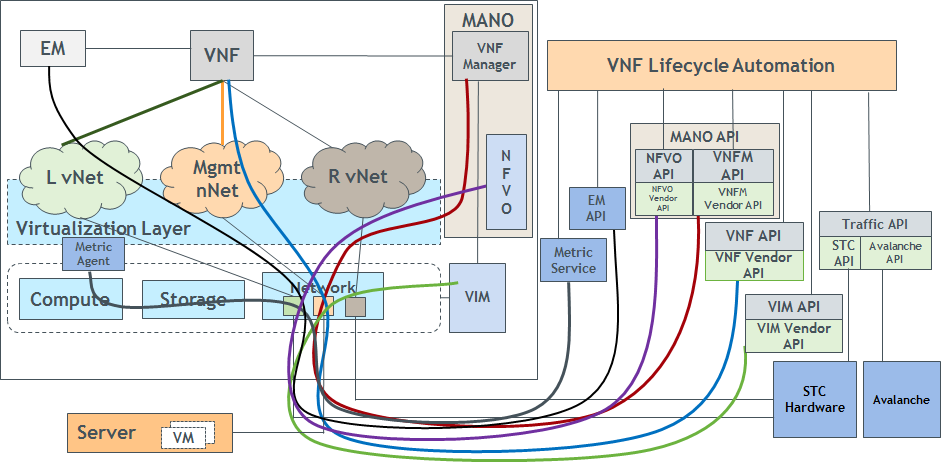
### Server based VNF

A server based VNF doesn’t forward packets. This type of VNF terminates the packet, extracts all required information out of the packet, processes the information and responds with one or multiple packets back to the requestor. To increase the overall network throughput of the server based VNF, two or more network interfaces could be assigned to the VNF. A server network IO is potentially limited by the overall network IO bandwidth. To reach higher bandwidth several physical interfaces may be used to increase the throughput.



### VNF Lifecycle Verification Architecture

The validation architecture of VNF Lifecycle Management is illustrated in the below picture.



The VNF Lifecycle Validation Automation is divided into following components:

1. VNF Lifecycle Automation Layer
2. MANO Automation Layer
   1. NFVO Automation Layer
      1. NFVO API
      2. NFVO Vendor API
   2. VNFM Automation Layer
      1. VNFM API
      2. VNFM Vendor API
3. VNF Automation Layer
   1. VNF API
   2. VNF Vendor API
4. VIM Automation Layer
   1. VIM API
   2. VIM Vendor API
5. Traffic Automation Layer
   1. STC API
   2. Avalanche API
6. EM API
7. NFVO API
8. Metric Service

#### VNF Lifecycle Automation Layer

Each VNF Lifecycle test case is implemented by a test sequence coded in the VNF Lifecycle Automation Layer. The VNF Lifecycle Automation Layer takes advantage of the following layers:

1. VNFM Automation Layer
2. VNF Automation Layer
3. VIM Automation Layer
4. Traffic Automation Layer

It calls each layers API to configure, instruct, control, manage and retrieve information or results.

#### MANO Automation Layer

The MANO Automation API layer represents the NFVO Automation Layer and VNFM Automation Layer. Pending on the customer used MANO architecture, the MANO Automation Layer adapts to required interface. The following chapters explain in more detail this module.

#### NFVO Automation Layer

The NFVO Layer is divided into two layers. The top layer NFVO API is the common layer and exposes a well-defined common interface to the upper layer. It abstracts the lower layer and resolves and encapsulates any vendor specific impacts. The vendor specific access to the NFVO is coded in the NFVO vendor API. This allows us to adapt to different vendor specific NFVO interfaces without impacting the overall architecture and test sequence.

#### VNFM Automation Layer

The VNFM Layer is divided into two layers. The top layer VNFM API is the common layer and exposes a well-defined common interface to the upper layer. It abstracts the lower layer and resolves and encapsulates any vendor specific impacts. The vendor specific access to the VNFM is coded in the VNFM vendor API. This allows us to adapt to different vendor specific VNFM interfaces without impacting the overall architecture and test sequence.

#### VNF Automation Layer

The VNF Layer is divided too into two layers. The top layer VNF API is the common layer and exposes a well-defined common interface to the upper layer. It abstracts the lower layer and resolves any vendor specific impacts. The vendor specific access to the VNF is coded in the VNF vendor API. This allows us to adapt to different vendor specific VNF interfaces without impacting the overall architecture and test sequence.

#### VIM Automation Layer

The VIM Layer is divided into two layers. The top layer VIM API is the common layer and exposes a well-defined common interface to the upper layer. It abstracts the lower layer and resolves any vendor specific impacts. The vendor specific access to the VIM is coded in the VIM vendor API. This allows us to adapt to different vendor specific VIM interfaces without impacting the overall architecture and test sequence.

#### Traffic Automation Layer

Test cases require to validate the correctness of the VNF functionality. In case of a transient VNF, the VNF Lifecycle verification can use STC to generate and analyze traffic. With the STC precision we are able to detect traffic packet loss and generate traffic up to line rate with ease.

A server VNF terminates the traffic and responses to requests with packets back. Majority of the data gets transported through a TCP connection and is therefore stateful traffic. If STC can’t provide this type of throughput, the traffic API must be able to adapt and use Avalanche product. Here we are able to generate line rate stateful traffic.

As the type of VNF has different demands on the type of traffic (stateful and stateless traffic), the Traffic API module gets separated into two layers. The common traffic API layer gets called by the VNF Lifecycle Automation layer. It provides a generic and common interface to the upper layer. The common traffic API layer takes advantage of the STC traffic layer or the Avalanche traffic layer. Each layer will have its own configuration file to drive its functionality. The traffic API layer determines based on the provided configuration to use either STC or Avalanche for the traffic generation.

### MANO API Implementation Options

At the time of the writing this document, many companies combine the functionality of the NFVO and VNFM into a single entity, named MANO. This is different from the NFVO MANO as it includes VIM, VNFM and NFVO. Therefore, the document uses the name MANO as an abbreviation for a module that combines the functionality of the NFVO and VNFM.

At the ETSI Plugtests following MANO provider were present:

Cisco - NFVO (command line interface)

Ensemble (ADVA)

Ericsson - Cloud Manager

Fraunhofer FOKUS - Open Baton

HPE - NFV Director

Huawei - Open-O

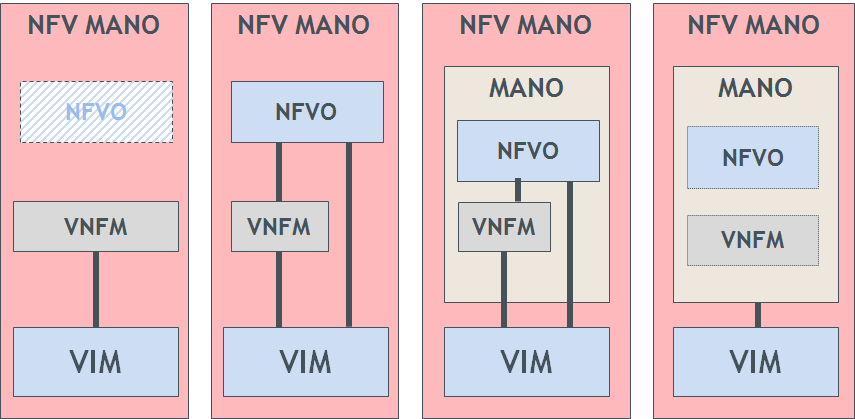
Openet - Weaver

OSM

Rift.io - Riftware

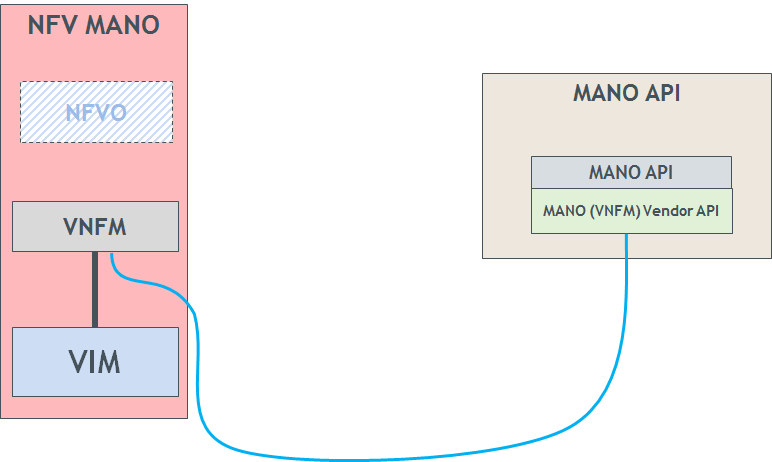
All of these MANO provider combined the NFVO and VNFM in their module. They did not expose a VNFM interface. Their UI interface was also very different. Many MANOs comply only partially to the current ETSI NFV standard. Therefore, our MANO API layer must be adaptable and ported to the customer selected MANO vendor. A service for porting our code is required until we either have implemented the interface of many MANO provider or provider agreed to a common interface.

Based on the current MANO implementation, our system must be able to be ported on the four possible implementations as outlined below.

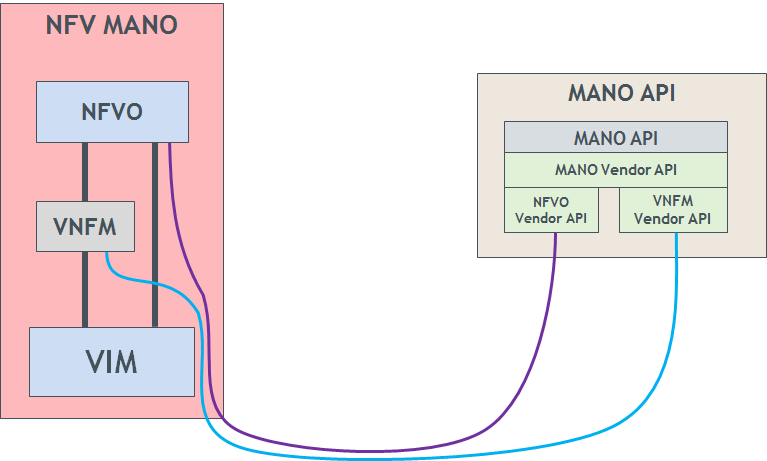


### MANO API Implementation Considerations

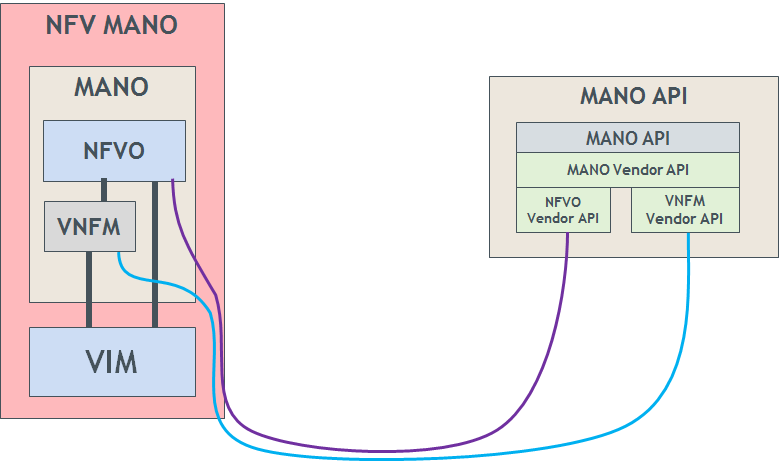
The MANO Automation API must adapt to the customer’s selection for the MANO functionality. As outlined in the previous section, four different options are currently considerable. In the first option, outlined in the next picture, the customer selected only a VNFM and explicitly neglected the NFVO from his setup. Here the MANO Automation interface has no NFVO component. All interactions are done through the VNFM. An VNF instantiation gets executed based on the VNFM exposed feature set. For example, the VNFM will not have a Network Service (NS) component. Therefore, NS related commands and controls are excluded and the instantiation is done on a lower level and not on the NS level.



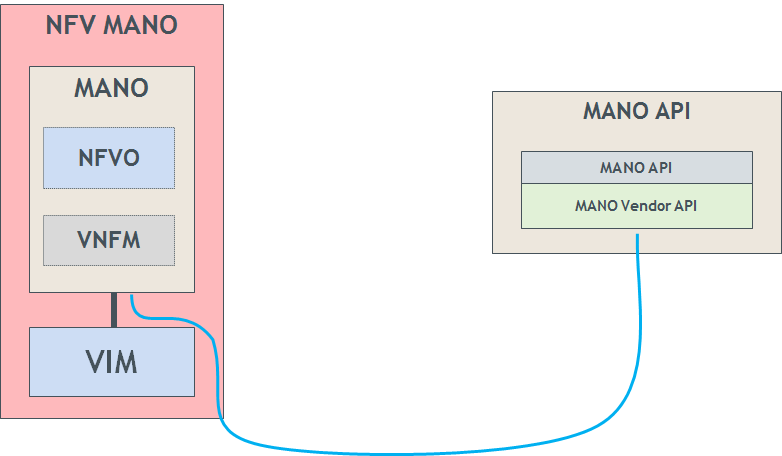
In the second option, the customer selected a system with separate NFVO and VNFM components. Here the MANO Automation API has NFVO and VNFM components. Any commands by default, like instantiation and termination is executed from the top layer, this means the commands get send to the NFVO. The VNFM API is used mostly for validation in addition to the VIM validation.



In the third option, the customer selected a single MANO provider, but this MANO implementation exposes a NFVO and VNFM interface. Also, here the MANO Automation API will have an internal API for the NFVO access and the VNFM access. As stated in the previous option, by default commands like instantiation, get initiated to the top layer, to the NFVO. The VNFM is more used for validation.



In the fourth and final option, the single MANO vendor exposed a single interface.



# VNF Lifecycle Management Overview

Test Cases for the VNF Lifecycle Management can be divided into different groups or areas. The following drawing illustrates them; they are

* VNF Instance Management
* Virtualized Resource Management
* Fault/Recovery
* VNF Package
* Performance Management

A more detailed explanation of these areas is given in the following table.

|  |  |
| --- | --- |
| Feature Group | Description |
| VNF Instance Management | Validation of VNF Instance Management includes VNF State Management and VNF installation and termination |
| Virtualized Resource Management | Validation of Virtualized Resource Management includes   * Scaling up/out the VNF resources 🡪 addition of virtualized resources * Scaling down/in the VNF resources 🡪 removal of virtualized resources * Scaling of virtualized specialized hardware acceleration * Scaling of network bandwidth * Scaling of Storage |
| Fault/Recovery | Validation of VNF Fault and Recovery Management includes   * VNF Recovery 🡪 healing * VNF Migration Management 🡪 resource migration |
| VNF Package | Validation of VNF Package Management   * VNF Package Content and Signature * VNFD validation * VNF snapshot management |
| Performance Management | Validation of Performance Management |

The Life-Cycle Management of a VNF includes following processes:

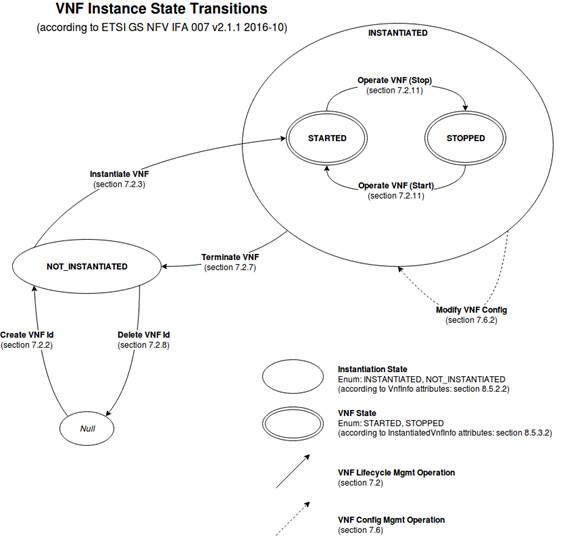
* VNF Instance Management
  + VNF State Management
    - VNF active and inactive
  + VNF installation and termination
  + VNF snapshot management
* Virtualized Resource Management
  + Scaling up/out the VNF resources 🡪 addition of virtualized resources
  + Scaling down/in the VNF resources 🡪 removal of virtualized resources
  + Scaling of virtualized specialized hardware acceleration
  + Scaling of network bandwidth
  + Scaling of Storage
* Performance Management
* VNF Fault and Recovery Management
  + VNF Recovery 🡪 healing
  + VNF Migration Management 🡪 resource migration
* Validation of VNF Package Management
  + VNF Package Content and Signature
  + VNFD validation

The lifecycle management of a VNF must ensure that the VNF follows the state transition diagram as illustrated below.

**NOTE**

The diagram represents the current description in the ETSI documents IFA-0007 and IFA-0008. The state diagram in document SWA-001 can’t be used as SWA documents are just informal and not-binding documents. IFA has acknowledged the differences. Once the result is available, this document may need to be corrected and the VNF Lifecycle Management test cases may need to be adjusted or enhanced.

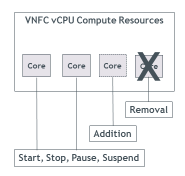
The major difference between the two state diagrams is that in IFA-007 the successful instantiation ends in the state STARTED. In SWA-001 the state INACTIVE is reached after the successful instantiation and configuration, a start command must be given here to get the VNF in the state ACTIVE.



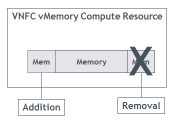
When a VNF gets instantiated, virtualized resource get allocated so that the VNF can operate. This virtualized resource management includes the allocation of

* vCPU
* vMemory
* vNIC
* vStorage
* vNICs (Video Hardware Acceleration NICs)

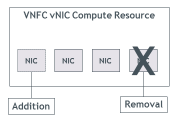
**Resource vCPU** can be added or removed to/from the VNFC. A vCPU resource can be started, stopped, paused and suspended.



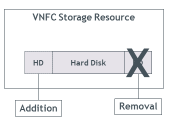
**Resource vMemory** can be added or removed to/from an VNFC.



**Resource nNIC** can be added or removed to/from a VNFC.



**Resource vStorage** can be added or removed to or from a VNFC.



# VNF Packaging

## VNF Packaging Overview

A VNF Package contains all required files and meta-data descriptors required to validate and instantiate a VNF. The VNF package is delivered to service providers by the VNF vendor.

The standardized meta-data descriptors describe

* NFV infrastructure resource requirements for a VNF in a service provider environment
* Design constraints and other dependencies in order for the VNF to successfully install, instantiate and terminate
* VNF operational behavior including VNF lifecycle events (e.g. scaling, upgrading)

The standardized packaging brings following advantages:

* Consistent, documented method for VNF providers to package VNFs
* Harmonize the service provider on-boarding process for VNFs coming from different VNF providers
* Integrity, trust and auditability of a VNF Package
* Flexible and extensible VNF packaging structure addressing wide variety of NFV infrastructure scenarios
* Packaged VNF-related meta-data are interpreted and packaged VNF are instantiated in a wide variety of orchestration systems irrespective of technology choice or infrastructure environment

## Levels of NFV Entities

For NFV management, there are four levels of entities, i.e.,

* Descriptors

General type definitions for entities such as VNFs and VLs, e.g. VNFD and VLD.

* Descriptor objects

Instance of a descriptor, e.g., an instance of a VNFD (not an instance of a VNF instantiated according to this VNFD). A descriptor object may provide (among other things) value ranges and default values for the attributes in the associated NFV entity class.

* NFV Entity Classes

Classes represent various NFV entities such as VNF and VL. There is one-to-one mapping between a descriptor object and an NFV entity class. An example of an NFV Entity Class is CDN Cache VNF.

* NFV Entity Instances

They are instances of a NFV entity class. An NFV entity instance is used to represent the current state and attribute values for a given NFV entity. Each NFV entity instance is bound by the associated descriptor object, e.g., value ranges and default values for attributes. An example of an NFV Entity Instance is a CDN Cache VNF instance.

## VNF Package Lifecycle

A VNF package goes through different steps to reach the state of deployment in a Service Provider system. Below are the different steps explained.

### VNF Package bundling for distribution by VNF Provider

The VNF providers use their own software development lifecycle, tools and procedures to archive a high quality standard for the VNF provided functionality. Once archived, all the software components associated with the version are retrieved for the packaging. This includes but not limited to own developed executables, libraries, scripts, configuration files as well as third party components, license agreements as well as build scripts.

Release notes get captured including clear description of the functionality of the release, any external dependencies, known bugs fixed relative to the prior releases as well as known issues in specific configurations.

The release gets bundled, the package gets signed and placed in a distribution repository once approved for distribution. From there the package gets delivered to the service provider by VNF vendor.

### VNF Package testing by VNF Provider

The VNF Package testing encompasses steps to guarantee that the package adheres to the standard structure and contains the mandatory metadata required in order to be considered compliant with the industry format.

It is the responsibility of the VNF provider to using parsing tools and perform a final test on the package in order to make sure that:

* VNF Package signature can be validated
* VNF Package can be unbundled
* VNF Package has the right structure (files, directories) as expected by onboarding tools

### VNF pre procurement by Service Provider

Via the VNF package the Service Provider can match the VNF against their own needs and allows to compare different offers from different suppliers. The service provider identifies and quantifies the VNF attributes against the service requirements by retrieving VNF metadata describing the scalability, reliability, manageability and security attributes out of the package.

### VNF Package validation and certification by Service Provider

A VNF Package is composed of several components like e.g. VNFD, software images, scripts, etc. During the on-boarding of the VNF Package, a validation of the package is performed. The validation is a procedure that verifies the integrity of the VNF Package.

A package is certified by performing acceptance testing and full functional testing against the VNF including configuration, management and service assurance.

The Service Provider can validate the package signature, origin, contents and structure. He can perform a full onboard, setup, install in a QA environment and certify the VNF for functionality as well as authenticity, integrity and packaging compliance.

### VNF Install or Deployment

Once the service provider selected the VNF provider, validated and confirmed the VNF package, the VNF package can be released for deployment and used to provide the VNF service.

NFV management and orchestration synchronization for VNF application software modification

VNF provider will provide newer versions of a VNF package to address problems or expose new features. When these new versions get deployed, the on-boarding of new versions requires the ability to keep track of multi version, multi environment multi instance and allow the service provider team to perform updates/upgrades with clear expectations of service continuity based on metadata information including component dependencies.

During the process of downgrading or upgrading an VNF, the system must update the information about a VNF instance stored in NFV management and orchestration as a result of a VNF application software modification performed through service provider’s management system, wherein such a process only comprises modifying the VNF’s application software without requiring a change of the VNF’s underlying virtualized resources or internal VNF component (VNFC) topology/composition. Examples of VNF application software modification are: update, upgrade, and downgrade. Such modification may be performed without requiring the termination of the VNF instance with the prior VNF application software version. Consequently, the relevant VNF Package is replaced by a different VNF Package which includes the VNF application software used in the modification. Complication arise when the new VNF package has new demands on the NFVI resources, like new vNIC card or obsoleting a previously supported vNIC card.

Provisioning of VNF configurable parameter

The VNFD is a static description file, not a dynamic configuration file. The metadata description in the VNFD is not changed during the whole VNF lifecycle. Some VNF parameters described in the VNFD can be declared to be configurable during the VNF design phase, and further be configured by the VNFM during or after VNF instantiation. The service provider should ensure that the VNF configurable parameters described in the VNFD are provisioned.

|  |  |
| --- | --- |
| TC.VNF.PKG.001 VNF Package Content Validation | |
| Purpose: | Verify vendor’s VNF Package file contains all mandatory files |
| Description | VNF Package includes different files. This test case validates that the mandatory files like VNFD and VNF image are present in the VNF package. |
| Test Setup | Ensure that the vendor provides the latest VNF Package file for inspection and later modifications to the VNF package file require to re-execute this test. |
| Parameters | The vendor’s VNF package file is the input to this test case. |
| Success Guarantee: | Vendor’s VNF package file must contain the mandatory content files. |
| Pre-conditions: |  |
| Procedure | 1. Validate the VNF Package signature 2. Validate VNF Package can be unbundled 3. Validate VNF Package has the right structure (files, directories) as expected by onboarding tools   Perform a validation that the VNF package contains the following:   * 1. VNFD   2. VNF image(s)   3. Scripts   4. … |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.PKG.002 VNF Package Onboarding | |
| Purpose: | Verify vendor’s VNF Package can be onboarded |
| Description | VNF Package Onboarding test case validates that the VNF package can be deployed by the MANO |
| Test Setup | Ensure that the vendor provides the latest VNF Package file and later modifications to the VNF package file require to re-execute this test. |
| Parameters | The vendor’s VNF package file is the input to this test case. |
| Success Guarantee: | Vendor’s VNF package is successful onboarded and listed as a new VNF source in the MANO and VIM |
| Pre-conditions: |  |
| Procedure | 1. On-board the VNF Package 2. Perform a validation that the VNF package has been onboarded successful via MANO and VIM interface |
| Extensions: |  |
| Notes: |  |

# Software Image Management

In an NFV environment, the management of image repository is designated to the NFV Orchestrator (NFVO). The VNFM can utilize the interface to the VIM (Virtualized Infrastructure Manager) to query about installed and available VNF images. This information can be used for other VNF management tasks.

The following test cases ensure the correctness of the VNFM’s capability to manage, retrieve and expose VNF image information.

|  |  |
| --- | --- |
| TC.VNF.IMAGE.001 Software Image Retrieval with single vendor release without filter | |
| Purpose: | Validate the MANO can query via Vi-Vnfm or Or-Vi reference point the installed/available software images and report the correct information |
| Description | The VNF images are installed and made available to the VIM. The VIM is able to gather all information and present the information per request on the Vi-Vnfm or Or-Vi reference point. Following information should be reported per software image:   * ID * Name * Provider * Version * checksum * Container Format * Disk Format * Create date * Update Date * Minimal disk size * Minimal Memory size * Size of the software * Status of the software * User defined meta Data |
| Test Setup | Minimal one vendor VNF image should be installed. It would be better to have multiple installed. This allows to validate handling of multiple image information and also filter capabilities. |
| Parameters | The test could include following:  Retrieve all images (no filter) 🡪 list of all installed images |
| Success Guarantee: | MANO must display all expect images |
| Pre-conditions: | Install images and ensure that VIM lists them |
| Procedure |  |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.IMAGE.002 Software Image Retrieval with single vendor release with filters | |
| Purpose: | Validate the MANO can query via Vi-Vnfm or Or-Vi reference point the installed/available software images and report the correct information |
| Description | The VNF images are installed and made available to the VIM. The VIM is able to gather all information and present the information per request on the Vi-Vnfm or Or-Vi reference point. Following information should be reported per software image:   * ID * Name * Provider * Version * checksum * Container Format * Disk Format * Create date * Update Date * Minimal disk size * Minimal Memory size * Size of the software * Status of the software * User defined meta Data |
| Test Setup | Minimal one vendor VNF image should be installed. It would be better to have multiple installed. This allows to validate handling of multiple image information and filter capabilities. |
| Parameters | The test could include following:  Retrieve image via exact filter 🡪 filter parameter narrows down to single image  Retrieve image via partial filter 🡪 filter parameter to narrow down the image list |
| Success Guarantee: | MANO must display all expect images |
| Pre-conditions: | Install images and ensure that VIM lists them |
| Procedure |  |
| Extensions: |  |
| Notes: | Sub test cases:   * Filter with exact match * Partial Filter |

The vendor needs to ensure that VNF upgrades/patches can generate correct VNFDs. This allows us also to test later the actual upgrade. Here we validate that the two VNFDs are truly different and not a copy without any changes.

|  |  |
| --- | --- |
| TC.VNF.IMAGE.003 Software Image Retrieval with multi vendor release | |
| Purpose: | Validate the MANO can query via Vi-Vnfm or Or-Vi reference point the installed/available software images and report the correct information while two different releases of the vendor’s software are installed |
| Description | The VNF images are installed and made available to the VIM. The VIM is able to gather all information and present the information per request on the Vi-Vnfm or Or-Vi reference point. Following information should be reported per software image:   * ID * Name * Provider * Version * checksum * Container Format * Disk Format * Create date * Update Date * Minimal disk size * Minimal Memory size * Size of the software * Status of the software * User defined meta Data |
| Test Setup | Minimal two vendor VNF images must be installed. This allows to validate handling of multiple image information and filter capabilities. |
| Parameters | The test could include following:  Retrieve all images (no filter) 🡪 list of all installed images  Retrieve image via exact filter 🡪 filter parameter narrows down to single image  Retrieve image via partial filter 🡪 filter parameter to narrow down the image list |
| Success Guarantee: | MANO must display all expect images and must differentiate between the different releases from the same vendor. |
| Pre-conditions: | Install images and ensure that VIM lists them |
| Procedure |  |
| Extensions: |  |
| Notes: | Validate that the VNFD information states the correct vendor's software image  Sub test cases:   * No filter image retrieval * Partial filter image retrieval * Exact match filter image retrieval |

# Life-Cycle Management Information Retrieval

|  |  |
| --- | --- |
| TC.VNF.IMAGE.004 Life-Cycle management information | |
| Purpose: | Validate the MANO can query the Life-Cycle Management information of a specific image and report the correct information |
| Description | The MANO should get all Life-Cycle Management Information from the VNFD. This information allows the VNFM to instantiate the right image with the required virtualized resources |
| Test Setup | Minimal two vendor VNF images must be installed. This allows to validate handling of multiple image information and filter capabilities. |
| Parameters | Retrieve Lifecycle Management information for a specific image via exact filter 🡪filter parameter narrows down to single image |
| Success Guarantee: | MANO must display expect life-cycle management information for the specified image and must differentiate between the different releases from the same vendor. |
| Pre-conditions: | Install images and ensure that VIM lists them |
| Procedure |  |
| Extensions: |  |
| Notes: | Sub test cases:   * No filter image retrieval * Partial filter image retrieval * Exact match filter image retrieval |

# Snapshot Management

The snapshot management allows the user to create a copy of the current virtual image or storage. Quick recovery instantiation, problem research or debugging of issues take advantage of this feature.

|  |  |
| --- | --- |
| TC.VNF.SNAPSHOT.001 Compute Snapshot Creation and Deletion without traffic load | |
| Purpose: | MANO should provide the capability to create a compute snapshot by issuing a command to the NFVI |
| Description | A compute Snapshot allows the user to build an image from the current running system and later to re-instantiate it without losing any data or configuration |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF by MANO 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Update VNF (--> time stamp) 4. Start the min traffic load 5. Validate traffic goes through (--> no dropped packets) 6. Stop traffic 7. Initiate the compute snapshot creation via the MANO interface 8. Validate VNF instructed NFVI to create a compute snapshot and a new image is found via the NFVI interface 9. Terminate VNF 10. Issue command to MANO to display the newly created VNF snapshot 11. Issue command deletion of compute snapshot to the MANO 12. Validate the snapshot has been deleted via MANO and NFVI interface |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.SNAPSHOT.002 Compute Snapshot Creation and Deletion under normal traffic load | |
| Purpose: | MANO should provide the capability to create a compute snapshot by issuing a command to the NFVI while normal traffic load |
| Description | A compute Snapshot allows the user to build an image from the current running system and later to re-instantiate it without losing any data or configuration |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF by MANO 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Start the normal traffic load 4. Validate traffic goes through (--> no dropped packets) 5. Initiate the compute snapshot creation via the VNFM interface 6. Validate MANO instructed NFVI to create a compute snapshot and a new image is found via the NFVI interface 7. Validate no traffic lost occurred 8. Stop traffic 9. Terminate VNF 10. Issue command to MANO to display the newly created VNF snapshot 11. Issue command deletion of compute snapshot to the MANO 12. Validate the snapshot has been deleted vi MANO and NFVI interface |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.SNAPSHOT.003 Compute Snapshot Instantiation | |
| Purpose: | MANO should provide the capability to re-instantiate a compute snapshot by issuing a command to the NFVI |
| Description | A compute Snapshot allows the user to build an image from the current running system and later to re-instantiate it without losing any data or configuration |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF by MANO 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Start the min traffic load 4. Validate traffic goes through (--> no dropped packets) 5. Stop traffic 6. Initiate the compute snapshot creation via the MANO interface 7. Validate MANO instructed NFVI to create a compute snapshot and a new image is found via the NFVI interface 8. Terminate VNF 9. Issue command to MANO to display the newly created VNF snapshot 10. Issue command to instantiate a compute snapshot to MANO 11. Validate the VNF has been instantiated from the snapshot 12. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 13. Start the min traffic load 14. Validate traffic goes through (--> no dropped packets) 15. Stop Traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.SNAPSHOT.004 Storage Snapshot Creation and Deletion | |
| Purpose: | MANO should provide the capability to create a storage snapshot by issuing a command to the NFVI |
| Description | A storage Snapshot allows the user to build an image from the current used storage and later to re-instantiate it without loss of any storage data |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Start the min traffic load 4. Validate traffic goes through (--> no dropped packets) 5. Stop traffic 6. Initiate the storage snapshot creation via the MANO interface 7. Validate MANO instructed NFVI to create a storage snapshot and a new image is found via the NFVI interface 8. Terminate VNF 9. Issue command to MANO to display the newly created VNF storage snapshot 10. Issue command deletion of storage snapshot to the MANO 11. Validate the storage snapshot has been deleted via MANO and NFVI interface |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.SNAPSHOT.005 Storage Snapshot Instantiation | |
| Purpose: | MANO should provide the capability to re-instantiate a storage snapshot by issuing a command to the NFVI |
| Description | A compute Snapshot allows the user to build an image from the current running system and later to re-instantiate it without loss of storage data |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Start the min traffic load 4. Validate traffic goes through (--> no dropped packets) 5. Stop traffic 6. Initiate the storage snapshot creation via the MANO interface 7. Validate MANO instructed NFVI to create a storage snapshot and a new image is found via the NFVI interface 8. Stop VNF 9. Validate VNF instantiation state is INSTANTIATED and VNF state is STOPPED (--> time stamp when correct state reached) 10. Issue command to MANO to display the newly created VNF storage snapshot 11. Issue command to instantiate a storage snapshot and mount it to the VNF 12. Validate the VNF has been assigned the new storage snapshot 13. Validate VNF instantiation state is INSTANTIATED and VNF state is STOPPED (--> time stamp when correct state reached) 14. Start the VNF 15. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 16. Start the min traffic load 17. Validate traffic goes through (--> no dropped packets) 18. Stop Traffic 19. Terminate VNF |
| Extensions: |  |
| Notes: |  |

# VNF Descriptor

Each VNF has exactly one associated VNF Descriptor (VNFD). A VNF Descriptor includes

* + Requirements for initial deployment state
  + Connections between the VNFCs
  + Exposed interfaces provided by the VNF
  + Resource requirements for each VNFC as each VNFC instance is hosted inside its own unique virtualization container

VNFD is a deployment template describing a VNF in term of deployment and operational behavior requirements

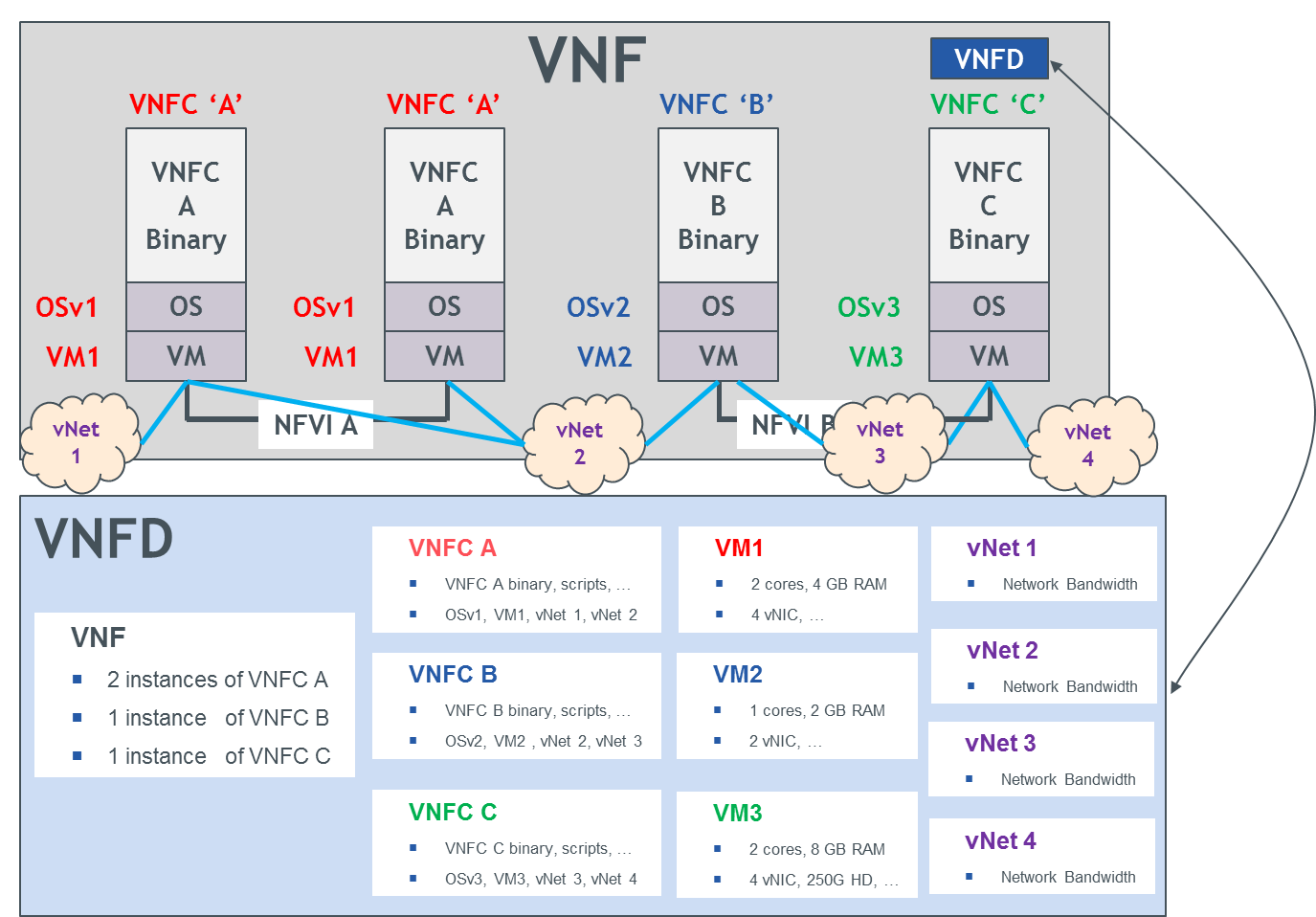
* + Virtualized Compute Resources (CPU/core, memory)
  + Virtualized Network Resources
  + Virtualized Storage Resources
  + Virtualized NIC cards

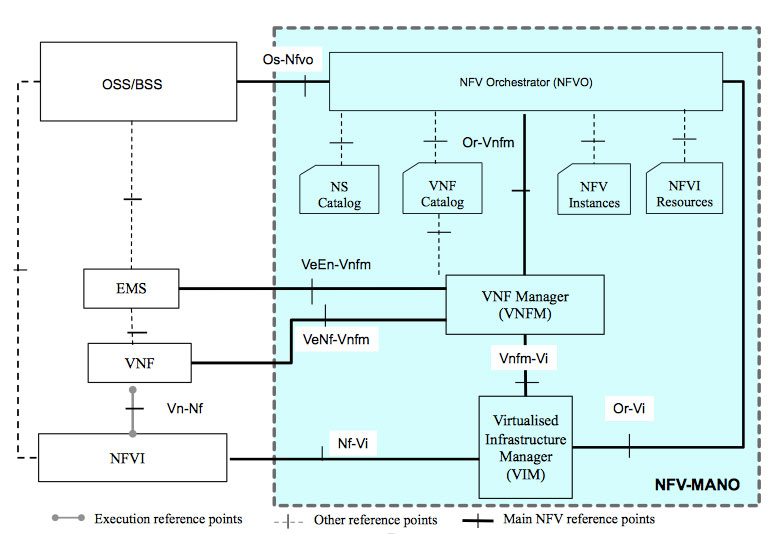
NFVO and VNFM query the VNF Catalogue for retrieving VNFD information.

The VNFD describes the vendor's terms of deployment and operational behavior requirements. It is divided into multiple sections or units:

1. VNFD Base Unit
2. VNFD Virtual Deployment Unit (VDU)
3. VNFD Virtual Link
4. VNFD Connection Points
5. VNFD Deployment Flavors

The following picture illustrates from a high level the VNFD elements.





The VNF Descriptor can be accessed only by the NFV Orchestrator or the VNF Manager. To ensure that a vendor VNF works correctly in a NFV environment, Spirent should validate the correctness of the vendors VNFD. There are different approaches to validate the vendor’s VNFD:

1. Visual Inspection
   1. Scan the vendor’s VNFD file, display the information and visual inspect the information for correctness
   2. Use the VNF Manager to access and display the VNFD information and visual inspect the information for correctness
   3. Use the NFV Orchestrator to access and display the VNFD information and visual inspect the information for correctness
2. Collect a Spirent VNFD text file
   1. Scan the vendor’s VNFD file, validate the VNFD information against the Spirent text file content for correctness or
   2. Use the VNF Manager to access the VNFD information and validate them against the Spirent text file content for correctness or
   3. Use the NFV Orchestrator to access the VNFD information and validate them against the Spirent text file content for correctness

The below mentioned test cases don’t spell out all these combinations.

|  |  |
| --- | --- |
| TC.VNFD.Coding.001 VNFD XML Coding Validation | |
| Purpose: | Verify vendor’s VNFD file is coded according to the XML schema |
| Description | VNFD can be coded based on XML schema or YANG schema. This test case is for XML schema. |
| Test Setup | Ensure that the vendor provides the latest VNFD file for inspection and later modifications to the VNFD file require to re-execute this test. |
| Parameters | The vendor’s VNFD file is the input to this test case. |
| Success Guarantee: | Vendor’s VNFD file must pass the XML schema |
| Pre-conditions: |  |
| Procedure | Perform a validation that the VNFD file is coded based on the XML schema. |
| Extensions: |  |
| Notes: | Either Visual inspection or automated validation |

|  |  |
| --- | --- |
| TC.VNFD.Coding.002 VNFD YANG Coding Validation | |
| Purpose: | Verify vendor’s VNFD file is coded according to the YANG schema |
| Description | VNFD can be coded based on XML schema or YANG schema. This test case is for YANG schema. |
| Test Setup | Ensure that the vendor provides the latest VNFD file for inspection and later modifications to the VNFD file require to re-execute this test. |
| Parameters | The vendor’s VNFD file is the input to this test case. |
| Success Guarantee: | Vendor’s VNFD file must pass the YANG schema |
| Pre-conditions: |  |
| Procedure | Perform a validation that the VNFD file is coded based on the YANG schema. |
| Extensions: |  |
| Notes: | Either Visual inspection or automated validation |

## VNFD Base Unit

The base information elements of the VNFD includes following parameters

- ID of the VNFD,

- vendor name,

- descriptor version

- version of the software

- lifecycle\_event

- dependency

- monitor parameter

- auto scale policy

- manifest file

- manifest file security.

|  |  |
| --- | --- |
| TC.VNFD.Base.001 VNFD Base unit validation | |
| Purpose: | Verify vendor’s VNFD base unit against integrity and correctness |
| Description | VNFD Base unit elements includes following parameters  - ID of the VNFD,  - vendor name,  - descriptor version  - version of the software  - lifecycle\_event  - dependency  - monitor parameter  - auto scale policy  - manifest file  - manifest file security.  They should be validated for correctness, existence and against vendor provided information. |
| Test Setup | Ensure that the vendor provides the latest VNFD file for inspection and later modifications to the VNFD file require to re-execute this test. |
| Parameters | The vendor’s VNFD file is the input to this test case. |
| Success Guarantee: | The vendor has correctly coded the VNFD Base unit elements. |
| Pre-conditions: |  |
| Procedure | Either Visual inspection or automated validation  Sequence:   1. Verify VNFD ID (e.g. name)   vnfd:Id   1. Verify vendor name   vnfd:vendor   1. Verify version of the VNF Descriptor   vnfd:descriptor\_version   1. Verify version of the VNF software   vnfd:version   1. Verify lifecycle events (optional)   vnfd:lifecycle\_event   1. Verify dependencies between VDUs (optional)   vnfd:dependency   1. Verify monitoring parameter, which can be tracked for this VNF (optional)   vnfd:monitoring\_parameter   1. Verify auto scale policy (optional)   vnfd:auto\_scale\_policy   1. Verify manifest file (optional)   vnfd:manifest\_file   1. Verify manifest file security (optional)   vnfd:manifest\_file\_security |
| Extensions: |  |
| Notes: |  |

The vendor will face the condition to upgrade/update his VNF for the purpose of patch or new release deployment. The new release of the VNF will have its own VNFD. The next test case is to validate that the vendor can correctly code and manipulate the VNFD for a future release. It is expected that the VNFD of the new release has passed all other VNFD validation.

|  |  |
| --- | --- |
| TC.VNFD.Base.002 VNFD validation for VNF release deployment | |
| Purpose: | Verify vendor’s VNFD of existing and new release differentiate and allow correct new release deployment |
| Description | To deploy a new release or patch of a VNFD, the vendor must code the VNFD of the new release correctly. |
| Test Setup | The vendor provides two VNFD files. One VNFD is for the current release, the second VNFD is for the future release. |
| Parameters | Two vendor’s VNFD files are the input to this test case. |
| Success Guarantee: | The vendor has correctly coded the VNFD of the future release so that there is a differentiation. |
| Pre-conditions: |  |
| Procedure | Either Visual inspection or automated validation  Sequence:   1. Verify VNFD ID (e.g. name) for each file 2. Verify vendor name is the same for both files 3. Verify versions of the VNF Descriptor are different 4. Verify versions of the VNF software are different 5. Verify lifecycle event attribute is properly coded for each VNFD version 6. Verify dependency attribute is properly coded for each VNFD version 7. Verify monitoring parameter is properly coded for each VNFD version 8. Verify auto scale policy attribute is properly coded for each VNFD version 9. Verify manifest file attribute is properly coded for each VNFD version 10. Verify manifest file security attribute is properly coded for each VNFD version |
| Extensions: |  |
| Notes: |  |

## VNFD Virtual Deployment Unit (VDU)

The VNFD Virtual Deployment Unit (VDU) exists of following units:

* 1. VNFD:VDU Base Unit
  2. VNFD:VDU VNFC Elements
  3. VNFD:VDU CPU
  4. VNFD:VDU Memory
  5. VNFD:VDU Security
  6. VNFD:VDU HyperVisor
  7. VNFD:VDU Network Elements
  8. VNFD:VDU Virtual Switches
  9. VNFD:VDU General Reliability and Availability
  10. VNFD:VDU Storage

### VNFD:VDU Base Unit

The VNFD Virtual Deployment Unit (VNFD:VDU) has following base elements:

- Unique ID

- VM image reference

- Computation Requirements

- Virtual Memory resource

- Virtual Network Bandwidth Resource

- LifeCycle Event

- High Availability

- Scale out/In

- Monitoring Parameter

These elements should be validated for correctness, existence, against integrity and information provided by the VNF vendor.

|  |  |
| --- | --- |
| TC.VNFD.VDU.001 VNFD:VDU Base Unit validation | |
| Purpose: | Verify vendor’s VNFD VDU base unit against integrity and correctness |
| Description | The VNFD Virtual Deployment Unit (VNFD:VDU) has following base elements:  - Unique ID  - VM image reference  - Computation Requirements  - Virtual Memory resource  - Virtual Network Bandwidth Resource  - LifeCycle Event  - High Availability  - Scale out/In  - Monitoring Parameter  They should be validated for correctness, existence and against vendor provided information. |
| Test Setup | Ensure that the vendor provides the latest VNFD file for inspection and later modifications to the VNFD file require to re-execute this test. |
| Parameters | The vendor’s VNFD file is the input to this test case. |
| Success Guarantee: | The vendor has correctly coded the VNFD Base unit elements. |
| Pre-conditions: |  |
| Procedure | Either Visual inspection or automated validation  Sequence:   1. Verify VDU's unique identifier for a specific VNF   vnfd:vdu:id   1. Verify VM image reference (optional)   vnfd:vdu:vm\_image   1. Verify required computation resources characteristics (e.g. processor power, number of virtual CPUs), including Key Quality Indicators (KQIs) for performance and reliability/availability vnfd:vdu:computation\_requirement 2. Verify virtual memory needed for the VDU vnfd:vdu:virtual\_memory\_resource\_element 3. Verify requirements in terms of the virtual network bandwidth needed for the VDU   vnfd:vdu:virtual\_network\_bandwidth\_resource   1. Verify LifeCycle Events (optional)   vnfd:vdu:lifecycle\_event   1. Verify constraints (optional)   vnfd:vdu:constraint   1. Verify redundancy model to ensure high availability (examples include ActiveActive and ActivePassive)   This is an optional attribute.  vnfd:vdu:high\_availability   1. Verify minimum and maximum number of instances which can be created to support scale out/in functionality (optional) vnfd:vdu:scale\_in\_out 2. Verify monitoring parameters trackable for this VNF like memory-consumption, CPU-utilization, bandwidth-consumption, VNFC downtime (optional)   vnfd:vdu:monitoring\_parameter |
| Extensions: |  |
| Notes: |  |

### VNFD:VDU VNFC Elements

The VNFD:VDU:VNFC defines the connection points for every VNFC. The information in the VNFD should be validated for correctness and against the vendor provided information. The virtual link reference should be confirmed to be listed in the VNF internal Virtual Link.

|  |  |
| --- | --- |
| TC.VNFD.VDU.002 VNFD:VDU Base Unit validation | |
| Purpose: | Verify vendor’s VNFD VDU base unit against integrity and correctness |
| Description | The VNFD:VDU:VNFC defines the connection points for every VNFC. The information in the VNFD should be validated for correctness and against the vendor provided information. The virtual link reference should be confirmed to be listed in the VNF internal Virtual Link. |
| Test Setup | Ensure that the vendor provides the latest VNFD file for inspection and later modifications to the VNFD file require to re-execute this test. |
| Parameters | The vendor’s VNFD file is the input to this test case. |
| Success Guarantee: | The vendor has correctly coded the VNFD:VDU:VNFC elements. The virtual link reference should be confirmed to be listed in the VNF internal Virtual Link. |
| Pre-conditions: |  |
| Procedure | Either Visual inspection or automated validation  Sequence:   1. Verify VNFC unique identifier within the namespace of a specific VNF vnfd:vdu:vnfc:id 2. Verify VNFC Connection Point ID vnfd:vdu:vnfc:connection\_point:id 3. Verify VNFC Connection Point type vnfd:vdu:vnfc:connection\_point:type 4. Verify VNFC Connection Point Virtual Link reference vnfd:vdu:vnfc:connection\_point:virtual\_link\_reference |
| Extensions: |  |
| Notes: | The virtual link reference must be listed in the VNF internal Virtual Link |

### VNFD:VDU CPU

The VNFD:VDU includes elements about the CPU's. The elements should be validated for correctness and against the vendor provided information.

|  |  |
| --- | --- |
| TC.VNFD.VDU.003 VNFD:VDU CPU Unit validation | |
| Purpose: | Verify vendor’s VNFD:VDU CPU unit against integrity and correctness |
| Description | The VNFD:VDU includes elements about the CPU's. The elements should be validated for correctness and against the vendor provided information. |
| Test Setup | Ensure that the vendor provides the latest VNFD file for inspection and later modifications to the VNFD file require to re-execute this test. |
| Parameters | The vendor’s VNFD file is the input to this test case. |
| Success Guarantee: | The vendor has correctly coded the VNFD:VDU CPU elements. |
| Pre-conditions: |  |
| Procedure | Either Visual inspection or automated validation  Sequence:   1. Verify CPU instruction set extensions for which VDU has been developed, optimized or tested with (optional)   vnfd:vdu:cpu\_instruction\_set\_extension   1. Verify CPU model (optional)   vnfd:vdu:cpu\_model   1. Verify CPU model specification binding (optional)   vnfd:vdu:cpu\_model\_specification\_binding   1. Verify minimum CPU clock speed (optional)   vnfd:vdu:cpu\_min\_clock\_speed     1. Verify number of CPU cores allocated to the VDU (optional)   vnfd:vdu:cpu\_core\_reservation   1. Verify Simultaneous Multi-Threading HW thread specification (optional)   vnfd:vdu:cpu\_simultaneous\_multi\_threading\_hw\_thread\_specification   1. Verify CPU core oversubscription policy in terms of virtual cores to physical cores/threads on the platform (optional)   vnfd:vdu:cpu\_core\_oversubscription\_policy   1. Verify CPU core and HW thread allocation topology policy (optional)   vnfd:vdu:cpu\_core\_and\_hw\_thread\_allocation\_topology\_policy   1. Verify size of the last level cache (optional)   vnfd:vdu:cpu\_last\_level\_cache\_size   1. Verify ability of an I/O device to have direct access to the CPU cache (optional)   vnfd:vdu:cpu\_direct\_io\_access\_to\_cache   1. Verify Translation Look-aside Buffer (TLB) parameters such as:    1. TLB Size    2. TLB Large Page Support    3. IOTLB Size    4. IOTLB Large Page Support   This is an optional attribute.  vnfd:vdu:cpu\_translation\_look\_aside\_buffer\_parameter   1. Verify Hot add CPU support (optional)   vnfd:vdu:cpu\_hot\_add   1. Verify data processing accelerator framework support (optional)   vnfd:vdu:cpu\_support\_accelerator |
| Extensions: |  |
| Notes: |  |

### VNFD:VDU Memory

The VNFD:VDU includes elements about the Memory. The elements should be validated for correctness and against the vendor provided information.

|  |  |
| --- | --- |
| TC.VNFD.VDU.004 VNFD:VDU Memory Unit validation | |
| Purpose: | Verify vendor’s VNFD:VDU Memory unit against integrity and correctness |
| Description | The VNFD:VDU includes elements about Memory. The elements should be validated for correctness and against the vendor provided information. |
| Test Setup | Ensure that the vendor provides the latest VNFD file for inspection and later modifications to the VNFD file require to re-execute this test. |
| Parameters | The vendor’s VNFD file is the input to this test case. |
| Success Guarantee: | The vendor has correctly coded the VNFD:VDU Memory elements. |
| Pre-conditions: |  |
| Procedure | Either Visual inspection or automated validation  Sequence:   1. Verify memory related parameters like   Memory Type, Memory Speed, Number of memory channels, Size of available memory, Reliability characteristics, Memory oversubscription policy, Memory bandwidth required per VDU, Number of large pages required per VDU, Non-Uniform Memory Architecture (NUMA) Allocation Policy  This is an optional attribute.  vnfd:vdu:memory\_parameter   1. Verify Hot add memory support (optional)   vnfd:vdu:memory\_hot\_add |
| Extensions: |  |
| Notes: |  |

### VNFD:VDU Security

The VNFD:VDU includes elements about security. The elements should be validated for correctness and against the vendor provided information.

|  |  |
| --- | --- |
| TC.VNFD.VDU.005 VNFD:VDU security Unit validation | |
| Purpose: | Verify vendor’s VNFD:VDU Security unit against integrity and correctness |
| Description | The VNFD:VDU includes elements about security. The elements should be validated for correctness and against the vendor provided information. |
| Test Setup | Ensure that the vendor provides the latest VNFD file for inspection and later modifications to the VNFD file require to re-execute this test. |
| Parameters | The vendor’s VNFD file is the input to this test case. |
| Success Guarantee: | The vendor has correctly coded the VNFD:VDU Security elements. |
| Pre-conditions: |  |
| Procedure | Either Visual inspection or automated validation  Sequence:   1. Verify platform security parameters which can include the availability of features such as:    * The ability to generate true random numbers    * The availability of a Measure Launch Environment (MLE)   This is an optional attribute.  vnfd:vdu:platform\_security\_parameter |
| Extensions: |  |
| Notes: |  |

### VNFD:VDU HyperVisor

The VNFD:VDU includes elements about the HyperVisor. The elements should be validated for correctness and against the vendor provided information.

|  |  |
| --- | --- |
| TC.VNFD.VDU.006 VNFD:VDU HyperVisor Unit validation | |
| Purpose: | Verify vendor’s VNFD:VDU HyperVisor unit against integrity and correctness |
| Description | The VNFD:VDU includes elements about the HyperVisor. The elements should be validated for correctness and against the vendor provided information. |
| Test Setup | Ensure that the vendor provides the latest VNFD file for inspection and later modifications to the VNFD file require to re-execute this test. |
| Parameters | The vendor’s VNFD file is the input to this test case. |
| Success Guarantee: | The vendor has correctly coded the VNFD:VDU HyperVisor elements. |
| Pre-conditions: |  |
| Procedure | Either Visual inspection or automated validation  Sequence:   1. Verify hypervisor related parameters like   Hypervisor type, Hypervisor version, Hypervisor Address Translation support parameters including:   * 1. Second Level Address Translation   2. Second Level Address Translation with Large page support   3. Second Level Address Translation for I/O   4. Second Level Address Translation for I/O with Large page support   5. Support for interrupt remapping   6. Support for data processing acceleration libraries in the hypervisor   vnfd:vdu:hypervisor\_parameter |
| Extensions: |  |
| Notes: |  |

### VNFD:VDU Network Elements

The VNFD:VDU includes elements about Network Elements. The elements should be validated for correctness and against the vendor provided information.

|  |  |
| --- | --- |
| TC.VNFD.VDU.007 VNFD:VDU Network Elements Unit validation | |
| Purpose: | Verify vendor’s VNFD:VDU Network Elements unit against integrity and correctness |
| Description | The VNFD:VDU includes elements about Network Elements. The elements should be validated for correctness and against the vendor provided information. |
| Test Setup | Ensure that the vendor provides the latest VNFD file for inspection and later modifications to the VNFD file require to re-execute this test. |
| Parameters | The vendor’s VNFD file is the input to this test case. |
| Success Guarantee: | The vendor has correctly coded the VNFD:VDU Network Elements. |
| Pre-conditions: |  |
| Procedure | Either Visual inspection or automated validation  Sequence:   1. Verify network interface card capabilities like   TCP Offload, Checksum Offload, Jumbo Frame, VLAN Tag stripping, RDMA support, SR-IOV support, DPDK support  This is an optional attribute.  vnfd:vdu:network\_interface\_card\_capability   1. Verify network speed/bandwidth to be guaranteed per requested NIC (optional)   vnfd:vdu:network\_interface\_bandwidth   1. Verify name and version of the data processing acceleration library used (optional)   vnfd:vdu:data\_processing\_acceleration\_library |
| Extensions: |  |
| Notes: |  |

### VNFD:VDU Virtual Switches

The VNFD:VDU includes elements about the Virtual Switches. The elements should be validated for correctness and against the vendor provided information.

|  |  |
| --- | --- |
| TC.VNFD.VDU.008 VNFD:VDU Virtual Switches validation | |
| Purpose: | Verify vendor’s VNFD:VDU Virtual Switches unit against integrity and correctness |
| Description | The VNFD:VDU includes elements about Virtual Switches. The elements should be validated for correctness and against the vendor provided information. |
| Test Setup | Ensure that the vendor provides the latest VNFD file for inspection and later modifications to the VNFD file require to re-execute this test. |
| Parameters | The vendor’s VNFD file is the input to this test case. |
| Success Guarantee: | The vendor has correctly coded the VNFD:VDU Virtual Switches elements. |
| Pre-conditions: |  |
| Procedure | Either Visual inspection or automated validation  Sequence:   1. Verify vSwitch type, version and key features such as overlay tunnel termination support (optional)   vnfd:vdu:vswitch\_capability |
| Extensions: |  |
| Notes: |  |

### VNFD:VDU General Reliability and Availability

The VNFD:VDU includes elements about General Reliability and Availability. The elements should be validated for correctness and against the vendor provided information.

|  |  |
| --- | --- |
| TC.VNFD.VDU.009 VNFD:VDU General Reliability and Availability Unit validation | |
| Purpose: | Verify vendor’s VNFD:VDU General Reliability and Availability unit against integrity and correctness |
| Description | The VNFD:VDU includes elements about the General Reliability and Availability. The elements should be validated for correctness and against the vendor provided information. |
| Test Setup | Ensure that the vendor provides the latest VNFD file for inspection and later modifications to the VNFD file require to re-execute this test. |
| Parameters | The vendor’s VNFD file is the input to this test case. |
| Success Guarantee: | The vendor has correctly coded the VNFD:VDU General Reliability and Availability elements. |
| Pre-conditions: |  |
| Procedure | Either Visual inspection or automated validation  Sequence:   1. Verify tracking system status for CORRECTABLE errors (optional)   vnfd:vdu:corrected\_error\_notification   1. Verify tracking system status for UNCORRECTABLE errors (optional) vnfd:vdu:uncorrected\_error\_notification |
| Extensions: |  |
| Notes: |  |

### VNFD:VDU Storage

The VNFD:VDU includes elements about the Storage. The elements should be validated for correctness and against the vendor provided information.

|  |  |
| --- | --- |
| TC.VNFD.VDU.010 VNFD:VDU Storage Unit validation | |
| Purpose: | Verify vendor’s VNFD:VDU Storage unit against integrity and correctness |
| Description | The VNFD:VDU includes elements about Storage. The elements should be validated for correctness and against the vendor provided information. |
| Test Setup | Ensure that the vendor provides the latest VNFD file for inspection and later modifications to the VNFD file require to re-execute this test. |
| Parameters | The vendor’s VNFD file is the input to this test case. |
| Success Guarantee: | The vendor has correctly coded the VNFD:VDU Storage elements. |
| Pre-conditions: |  |
| Procedure | Either Visual inspection or automated validation  Sequence:   1. Verify required storage characteristics (e.g. size), including Key Quality Indicators (KQIs) for performance and reliability/availability (optional)   vnfd:vdu:storage\_requirement   1. Verify bandwidth value for which VDU had been developed, optimized or tested from storage perspective (optional)   vnfd:vdu:rdma\_support\_bandwidth |
| Extensions: |  |
| Notes: |  |

## VNFD Virtual Link

The VNFD includes elements about internal virtual links. The elements should be validated for correctness and against the vendor provided information. The connection points references should be validated to be in the vnfd:vdu:vnfc:connection\_point:id or vnfd:connection\_point:id

|  |  |
| --- | --- |
| TC.VNFD.VLINK.001 VNFD:VDU Virtual Link validation | |
| Purpose: | Verify vendor’s VNFD:VDU HyperVisor unit against integrity and correctness |
| Description | The VNFD includes elements about internal virtual links. The elements should be validated for correctness and against the vendor provided information. The connection points references should be validated to be in the vnfd:vdu:vnfc:connection\_point:id or vnfd:connection\_point:id. |
| Test Setup | Ensure that the vendor provides the latest VNFD file for inspection and later modifications to the VNFD file require to re-execute this test. |
| Parameters | The vendor’s VNFD file is the input to this test case. |
| Success Guarantee: | The vendor has correctly coded the VNFD:VDU Virtual Link elements. |
| Pre-conditions: |  |
| Procedure | Either Visual inspection or automated validation  Sequence:   1. Verify internal Virtual Link unique identifier - vnfd:virtual\_link:id 2. Verify connectivity type (e.g. E-LINE, E-LAN or E-Tree)   vnfd:virtual\_link:connectivity\_type   1. Verify references to Connection Points (vnfd:vdu:vnfc:connection\_point:id, vnfd:connection\_point:id)   vnfd:virtual\_link:connection\_points\_references   1. Verify required throughput of the link (e.g. bandwidth of E-Line, root bandwidth of E-Tree and aggregate capacity of E\_LAN)   vnfd:virtual\_link:root\_requirements   1. Verify required throughput of leaf connections to the link (for E-Tree and E-LAN branches)   This is an optional attribute.  vnfd:virtual\_link:leaf\_requirements   1. Verify QoS options to be supported on the VL e.g. latency, jitter, etc. (optional) vnfd:virtual\_link:qos 2. Verify test access facilities to be supported on the VL (e.g. none, passive monitoring, or active (intrusive) loopbacks at endpoints)   This is an optional attribute  vnfd:virtual\_link:test\_access |
| Extensions: |  |
| Notes: |  |

## VNFD Connection Points

The VNFD includes elements about the connection points. The elements should be validated for correctness and against the vendor provided information. Validate that the virtual link reference is found in the VNF internal virtual Link.

|  |  |
| --- | --- |
| TC.VNFD.ConPoint.001 VNFD Connection Points Unit validation | |
| Purpose: | Verify vendor’s VNFD Connection Points unit against integrity and correctness |
| Description | The VNFD includes elements about the connection points. The elements should be validated for correctness and against the vendor provided information. Validate that the virtual link reference is found in the VNF internal virtual Link. |
| Test Setup | Ensure that the vendor provides the latest VNFD file for inspection and later modifications to the VNFD file require to re-execute this test. |
| Parameters | The vendor’s VNFD file is the input to this test case. |
| Success Guarantee: | The vendor has correctly coded the VNFD Connection Points elements. |
| Pre-conditions: |  |
| Procedure | Either Visual inspection or automated validation  Sequence:   1. Verify Connection Point identifier   vnfd:connection\_point:id   1. Verify Virtual Link reference (optional)   vnfd:connection\_point:virtual\_link\_reference   1. Verify Connection Point type   vnfd:connection\_point:type |
| Extensions: |  |
| Notes: |  |

## VNFD Deployment Flavors

The VNFD includes elements about the deployment flavor. The elements should be validated for correctness and against the vendor provided information.

|  |  |
| --- | --- |
| TC.VNFD.DEPLOY.001 VNFD Deployment Flavors Unit validation | |
| Purpose: | Verify vendor’s VNFD Deployment Flavors unit against integrity and correctness |
| Description | The VNFD includes elements about the deployment flavor. The elements should be validated for correctness and against the vendor provided information. |
| Test Setup | Ensure that the vendor provides the latest VNFD file for inspection and later modifications to the VNFD file require to re-execute this test. |
| Parameters | The vendor’s VNFD file is the input to this test case. |
| Success Guarantee: | The vendor has correctly coded the VNFD Deployment Flavors elements. |
| Pre-conditions: |  |
| Procedure | Either Visual inspection or automated validation   1. Verify VNF flavor identifier   vnfd:deployment\_flavour:id   1. Verify monitoring parameter and its value against which this flavor is being described   vnfd:deployment\_flavour:flavor\_key   1. Verify constraints in place for a specific deployment flavor (optional)   vnfd:deployment\_flavour:constraint   1. Verify references for a specific VDU which should be used for this deployment flavor   vnfd:deployment\_flavour:constituent\_vdu:vdu\_reference   1. Verify number of VDU instances required vnfd:deployment\_flavour:constituent\_vdu:number\_ofinstances 2. Verify references to the VNFCs which should be used for this deployment flavor   vnfd:deployment\_flavour:constituent\_vdu:constituent\_vnfc |
| Extensions: |  |
| Notes: |  |

# VNF State Management

In this section, we concentrate on the state management of a VNF instance. This includes the instantiation, activation (start), deactivation (stop) and termination.

## VNF State Management: Instantiation

The request to instantiate a new VNF could be initiated by

1. NFVO
2. EM
3. Manual

This section is not to go through all the different possible originator of the instantiation requests. We concentrate here on the action of the instantiation and their correct execution. VNF provides specific functionality. In many cases a VNF requires configuration parameters to fulfill its tasks and may require license related configuration too. This configuration could be provided by a configuration file, by the EM or none. Pending on the VNF requirements, the VNFM or NFVO may need to go through different execution path.

|  |  |
| --- | --- |
| TC.VNF.STATE.INST.001 VNF Instantiation with configuration file | |
| Purpose: | Verify the MANO has the capability to instantiate a VNF and transitions to the VNF state “STARTED” automatically as there is no EM involvement and all configuration is given via configuration file.  When completed successful, the VNF instantiation state is INSTANTIATED and VNF state is STARTED. Traffic gets generated to validate the operational functionality. |
| Description | The VNF instantiation can be initiated through the MANO, the NFVO or the VNFM. It involves the allocation of virtual and physical resources and applying the initial configuration. A license may get applied to the VNF to allow its correct operation.  By default, the NFVI should have allocated the minimal resources and other required hardware capabilities. This includes also needed vNIC cards.  If the VNF has specific resource requirements (CPU type, NIC type, …), the compatibility validation must pass if the resources are available. If they are not fulfilled, the VNF should not be instantiated. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: | If the VNF depends on the existence of any hardware capabilities, these required hardware capabilities should be pre-installed in the system and the NFVI should list them as available virtualized resources. |
| Procedure | Measure the time to instantiate a VNF. The test therefore gets executed multiple times in the loop to get an average.  Sequence:   1. Instantiate the VNF (--> time stamp) 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Update VNF (--> time stamp) 4. Validate configuration file has been applied to the VNF 5. Validate license has been applied to the VNF if applicable 6. Validate the right vResources have been allocated 7. Start low traffic load 8. Validate traffic flow as the VNF should be in VNF stats STARTED 9. Stop Traffic 10. Calculate the instantiation time 11. Terminate VNF |
| Extensions: | Measure the time it takes to instantiate a VNF and get it into the state “STARTED” 🡪 VNF Service instantiation time |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.STATE.INST.002 VNF Instantiation with configuration file under load | |
| Purpose: | Verify the MANO has the capability to instantiate a VNF and transitions to the VNF state “STARTED” automatically as there is no EM involvement and all configuration is given via configuration file.  When completed successful, the VNF instantiation state is INSTANTIATED and VNF state is STARTED. Traffic gets generated at first to validate the operational functionality at the end and the system can instantiate under load.  Because we apply load, we can determine if traffic causes an increase in the instantiation time. |
| Description | The VNF instantiation can be initiated through the MANO, the NFVO or the VNFM. It involves the allocation of virtual and physical resources and applying the initial configuration. A license may get applied to the VNF to allow its correct operation.  By default, the NFVI should have allocated the minimal resources and other required hardware capabilities. This includes also needed vNIC cards.  If the VNF has specific resource requirements (CPU type, NIC type, …), the compatibility validation must pass if the resources are available. If they are not fulfilled, the VNF should not be instantiated. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: | If the VNF depends on the existence of any hardware capabilities, these required hardware capabilities should be pre-installed in the system and the NFVI should list them as available virtualized resources. |
| Procedure | Measure the time to instantiate a VNF. The test therefore gets executed multiple times in the loop to get an average.  Sequence:   1. Start low Traffic 2. Instantiate the VNF (--> time stamp) 3. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 4. Update VNF (--> time stamp) 5. Validate configuration file has been applied to the VNF 6. Validate license has been applied to the VNF if applicable 7. Validate the right vResources have been allocated 8. Validate traffic flow as the VNF should be in VNF stats STARTED 9. Stop Traffic 10. Calculate the instantiation time 11. Terminate VNF |
| Extensions: | Measure the time it takes to instantiate a VNF and get it into the state “STARTED” 🡪 VNF Service instantiation time |
| Notes: |  |



|  |  |
| --- | --- |
| TC.VNF.STATE.INST.003 VNF Instantiation with active Element Management | |
| Purpose: | Verify the MANO has the capability to instantiate a VNF, informs the EM of the existence, waits for the EM configuration completion and transitions to the VNF state “STARTED” after EM completed the configuration successfully. If the configuration failed by the EM, the VNF state should stay in the VNF instantiation state “Instantiated” and VNF state “STARTED”. Traffic gets generated to validate the operational functionality. |
| Description | The VNF instantiation can be initiated through the MANO, the NFVO or the VNFM. It involves the allocation of virtual and physical resources and applying the initial configuration. A license may get applied to the VNF to allow it correct operation.  By default, the NFVI should have allocated the minimal resources and other required hardware capabilities. This includes also needed NIC cards.  The EM is responsible for FCAPS management functionality of a VNF:  F – Fault Management  C – Configuration Management  A – Accounting Management  P – Performance Management  S – Security Management |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: | If the VNF depends on the existence of any hardware capabilities, these required hardware capabilities should be pre-installed in the system and the NFVI should list them as available virtualized resources. |
| Procedure | Measure the time to instantiate a VNF and may do it multiple times to get an average.  Sequence:   1. Start the EM or ensure EM is up and can configure the VNF 2. Instantiate the VNF (--> time stamp) 3. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 4. Validate the right vResources have been allocated 5. Validate configuration has been applied by the EM to the VNF 6. Validate license has been applied to the VNF if applicable 7. Start low traffic load 8. Validate traffic flow as the VNF should be in VNF stats STARTED 9. Stop Traffic 10. Terminate VNF 11. Calculate the instantiation time |
| Extensions: | Measure the time it takes to instantiate a VNF and get it into the VNF state “STARTED”. 🡪 VNF Service instantiation time |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.STATE.INST.004 VNF Instantiation with active Element Management under traffic load | |
| Purpose: | Verify the MANO has the capability to instantiate a VNF, informs the EM of the existence, waits for the EM configuration completion and transitions to the VNF state “STARTED” after EM completed the configuration successfully. If the configuration failed by the EM, the VNF state should stay in the VNF instantiation state “Instantiated” and VNF state “STARTED”. Traffic gets generated at first to validate the instantiation under load and the operational functionality at the end.  Because we apply load, we can determine if traffic causes an increase in the instantiation time. |
| Description | The VNF instantiation can be initiated through the MANO, the NFVO or the VNFM. It involves the allocation of virtual and physical resources and applying the initial configuration. A license may get applied to the VNF to allow it correct operation.  By default, the NFVI should have allocated the minimal resources and other required hardware capabilities. This includes also needed NIC cards.  The EM is responsible for FCAPS management functionality of a VNF:  F – Fault Management  C – Configuration Management  A – Accounting Management  P – Performance Management  S – Security Management |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: | If the VNF depends on the existence of any hardware capabilities, these required hardware capabilities should be pre-installed in the system and the NFVI should list them as available virtualized resources. |
| Procedure | Measure the time to instantiate a VNF and may do it multiple times to get an average.  Sequence:   1. Start the EM or ensure EM is up and can configure the VNF 2. Start low traffic load 3. Instantiate the VNF (--> time stamp) 4. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 5. Validate the right vResources have been allocated 6. Validate configuration has been applied by the EM to the VNF 7. Validate license has been applied to the VNF if applicable 8. Validate traffic flow as the VNF should be in VNF stats STARTED 9. Stop Traffic 10. Terminate VNF 11. Calculate the instantiation time |
| Extensions: | Measure the time it takes to instantiate a VNF and get it into the VNF state “STARTED”. 🡪 VNF Service instantiation time |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.STATE.INST.005 VNF Instantiation with active Element Management with failed EM configuration | |
| Purpose: | Verify the MANO has the capability to instantiate a VNF, informs the EM of the existence, waits for the EM configuration completion. Because the EM doesn’t have any configuration, the VNF state should stay in the VNF state “STARTED” |
| Description | The VNF instantiation can be initiated through the MANO, the NFVO or the VNFM. It involves the allocation of virtual and physical resources and applying the initial configuration. A license may get applied to the VNF to allow it correct operation.  By default, the NFVI should have allocated the minimal resources and other required hardware capabilities. This includes also needed NIC cards.  The EM is responsible for FCAPS management functionality of a VNF:  F – Fault Management  C – Configuration Management  A – Accounting Management  P – Performance Management  S – Security Management  The EM should report to the MANO that the configuration failed . |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: | If the VNF depends on the existence of any hardware capabilities, these required hardware capabilities should be pre-installed in the system and the NFVI should list them as available virtualized resources. |
| Procedure | Measure the time to instantiate a VNF and may do it multiple times to get an average.  Sequence:   1. Start the EM or ensure EM is up and can’t configure the VNF 2. Instantiate the VNF (--> time stamp) 3. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED after EM failed the configuration (--> time stamp when reach the state) 4. Validate the right vResources have been allocated 5. Validate configuration has NOT been applied by the EM to the VNF if possible 6. Validate license has been applied to the VNF if applicable 7. Start low traffic 8. Validate no traffic flow as the VNF did not receive configuration 9. Stop Traffic 10. Terminate VNF 11. Calculate the instantiation time |
| Extensions: | Measure the time it takes to instantiate a VNF and get it into the state “Instantiated-Not Configured”. 🡪 VNF Service instantiation time |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.STATE.INST.006 VNF Instantiation with inactive Element Management | |
| Purpose: | Verify the MANO has the capability to instantiate a VNF, informs the EM of the existence. Because EM is inactive, the MANO should transition to the instantiation state “Instantiated”. The VNF state should stay in the state “STARTED” |
| Description | The VNF instantiation can be initiated through the MANO, the NFVO or the VNFM. It involves the allocation of virtual and physical resources and applying the initial configuration. A license may get applied to the VNF to allow it correct operation.  By default, the NFVI should have allocated the minimal resources and other required hardware capabilities. This includes also needed NIC cards. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: | If the VNF depends on the existence of any hardware capabilities, these required hardware capabilities should be pre-installed in the system and the NFVI should list them as available virtualized resources. |
| Procedure | Sequence:   1. Deactivate EM if needed 2. Instantiate VNF 3. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 4. Validate the right vResources have been allocated 5. Start low traffic 6. Validate no traffic flow as the VNF didn’t receive configuration 7. Stop Traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Because of the inactive EM, the VNFM should transition the VNF instantiation state is INSTANTIATED and VNF state is STARTED. |







## 

|  |  |
| --- | --- |
| TC.VNF.STATE.INST.008 VNF Instantiation Failure with missing required virtual resources | |
| Purpose: | Verify the VNF does not get instantiated when one or more specified resources are missing, e.g., specialized CPU type, specialized NIC type, vMemory, vStorage, … |
| Description | The VNF instantiation can be initiated through the MANO, the NFVO or the VNFM. It involves the allocation of virtual and physical resources and applying the initial configuration. The instantiation must fail when one or more required resources are not available. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: | The VNF depends on the existence of specific hardware capabilities. The test case is to create a negative condition. The MANO, NFVO or VNFM, should not instantiate the VNF because one or more virtualized resources are not available. |
| Procedure | Eliminate one or more required virtual resources are missing in the NFVI  Sequence:   1. Instantiate VNF 2. Validate MANO reports no VNF instance and the error 3. Validate VIM reports no VNF instance and the error |
| Extensions: |  |
| Notes: | Each VNF has its own resource requirements. Therefore, many different types of negative test cases could generate out of this single test case.  Here are examples of sub test cases:   * Incompatible vCPU type * Missing Instructions sets, e.g. MMX, SSE, SSE2, … * Missing specialized vNIC * Missing hardware acceleration, e.g. DPDK, SRIOV * Missing vMemory size * Missing vStorage type * Missing vStorage size * Missing vCPU counts |

|  |  |
| --- | --- |
| TC.VNF.STATE.INST.008.1 VNF Instantiation Failure with missing required virtual resources “Incompatible vCPU type” | |
| Purpose: | Verify the VNF does not get instantiated when one or more specified resources are missing of compatible vCPU type |
| Description | The VNF instantiation can be initiated through the MANO (NFVO or VNFM). It involves the allocation of virtual and physical resources and applying the initial configuration. The instantiation must fail when one or more required resources are not available. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: | The VNF depends on the existence of specific hardware capabilities. The test case is to create a negative condition. The NFVI or MANO should not instantiate the VNF because one or more virtualized resources are not available. |
| Procedure | Eliminate one or more required virtual resources are missing in the NFVI  Sequence:   1. Instantiate VNF 2. Validate MANO reports no VNF instance and the error 3. Validate VIM reports no VNF instance and the error |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.STATE.INST.008.2 VNF Instantiation Failure with missing required virtual resources “Instructions sets” | |
| Purpose: | Verify the VNF does not get instantiated when one or more specified resources are missing of type Instructions sets |
| Description | The VNF instantiation can be initiated through the MANO (NFVO or VNFM). It involves the allocation of virtual and physical resources and applying the initial configuration. The instantiation must fail when one or more required resources are not available. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: | The VNF depends on the existence of specific hardware capabilities. The test case is to create a negative condition. The NFVI or MANO should not instantiate the VNF because one or more virtualized resources are not available. |
| Procedure | Eliminate one or more required virtual resources are missing in the NFVI  Sequence:   1. Instantiate VNF 2. Validate MANO reports no VNF instance and the error 3. Validate VIM reports no VNF instance and the error |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.STATE.INST.008.3 VNF Instantiation Failure with missing required virtual resources “Specialized vNIC” | |
| Purpose: | Verify the VNF does not get instantiated when one or more specified resources are missing of specialized vNIC |
| Description | The VNF instantiation can be initiated through the MANO (NFVO or VNFM). It involves the allocation of virtual and physical resources and applying the initial configuration. The instantiation must fail when one or more required resources are not available. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: | The VNF depends on the existence of specific hardware capabilities. The test case is to create a negative condition. The NFVI or MANO should not instantiate the VNF because one or more virtualized resources are not available. |
| Procedure | Eliminate one or more required virtual resources are missing in the NFVI  Sequence:   1. Instantiate VNF 2. Validate MANO reports no VNF instance and the error 3. Validate VIM reports no VNF instance and the error |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.STATE.INST.008.4 VNF Instantiation Failure with missing required virtual resources “Hardware Acceleration, e.g. DPDK, SRIOV” | |
| Purpose: | Verify the VNF does not get instantiated when one or more specified resources are missing of hardware acceleration, e.g. DPDK, SRIOV |
| Description | The VNF instantiation can be initiated through the MANO (NFVO or VNFM). It involves the allocation of virtual and physical resources and applying the initial configuration. The instantiation must fail when one or more required resources are not available. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: | The VNF depends on the existence of specific hardware capabilities. The test case is to create a negative condition. The NFVI or MANO should not instantiate the VNF because one or more virtualized resources are not available. |
| Procedure | Eliminate one or more required virtual resources are missing in the NFVI  Sequence:   1. Instantiate VNF 2. Validate MANO reports no VNF instance and the error 3. Validate VIM reports no VNF instance and the error |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.STATE.INST.008.5 VNF Instantiation Failure with missing required virtual resources “vMemory size” | |
| Purpose: | Verify the VNF does not get instantiated when one or more specified resources are missing of vMemory size |
| Description | The VNF instantiation can be initiated through the MANO (NFVO or VNFM). It involves the allocation of virtual and physical resources and applying the initial configuration. The instantiation must fail when one or more required resources are not available. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: | The VNF depends on the existence of specific hardware capabilities. The test case is to create a negative condition. The NFVI or MANO should not instantiate the VNF because one or more virtualized resources are not available. |
| Procedure | Eliminate one or more required virtual resources are missing in the NFVI  Sequence:   1. Instantiate VNF 2. Validate MANO reports no VNF instance and the error 3. Validate VIM reports no VNF instance and the error |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.STATE.INST.008.6 VNF Instantiation Failure with missing required virtual resources “vStorage type” | |
| Purpose: | Verify the VNF does not get instantiated when one or more specified resources are missing of type vStorage type |
| Description | The VNF instantiation can be initiated through the MANO (NFVO or VNFM). It involves the allocation of virtual and physical resources and applying the initial configuration. The instantiation must fail when one or more required resources are not available. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: | The VNF depends on the existence of specific hardware capabilities. The test case is to create a negative condition. The NFVI or MANO should not instantiate the VNF because one or more virtualized resources are not available. |
| Procedure | Eliminate one or more required virtual resources are missing in the NFVI  Sequence:   1. Instantiate VNF 2. Validate MANO reports no VNF instance and the error 3. Validate VIM reports no VNF instance and the error |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.STATE.INST.008.7 VNF Instantiation Failure with missing required virtual resources “vStorage size” | |
| Purpose: | Verify the VNF does not get instantiated when one or more specified resources are missing of vStorage size |
| Description | The VNF instantiation can be initiated through the MANO (NFVO or VNFM). It involves the allocation of virtual and physical resources and applying the initial configuration. The instantiation must fail when one or more required resources are not available. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: | The VNF depends on the existence of specific hardware capabilities. The test case is to create a negative condition. The NFVI or MANO should not instantiate the VNF because one or more virtualized resources are not available. |
| Procedure | Eliminate one or more required virtual resources are missing in the NFVI  Sequence:   1. Instantiate VNF 2. Validate MANO reports no VNF instance and the error 3. Validate VIM reports no VNF instance and the error |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.STATE.INST.008.8 VNF Instantiation Failure with missing required virtual resources “vCPU counts” | |
| Purpose: | Verify the VNF does not get instantiated when one or more specified resources are missing of vCPU counts |
| Description | The VNF instantiation can be initiated through the MANO (NFVO or VNFM). It involves the allocation of virtual and physical resources and applying the initial configuration. The instantiation must fail when one or more required resources are not available. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: | The VNF depends on the existence of specific hardware capabilities. The test case is to create a negative condition. The NFVI or MANO should not instantiate the VNF because one or more virtualized resources are not available. |
| Procedure | Eliminate one or more required virtual resources are missing in the NFVI  Sequence:   1. Instantiate VNF 2. Validate MANO reports no VNF instance and the error 3. Validate VIM reports no VNF instance and the error |
| Extensions: |  |
| Notes: |  |

## VNF State Management: Start

A VNF starts processing packets only after reaching the VNF state “STARTED”. This state gets reached by issuing the command “Start” when the VNF is in the VNF state “STOPPED”.

|  |  |
| --- | --- |
| TC.VNF.STATE.START.001 VNF Start without traffic load | |
| Purpose: | Verify VNF can be activated and processes packets after the activation |
| Description | After the VNF is instantiated, the VNF state is “STARTED”. A command STOP needs to be issued to get the VNF state into STOPPED so that the test can be executed. To allow processing of its functionality, it must be activated. Via the MANO (VNFM or NFVO) the command “Start” is been issued to bring the VNF into an active state and validate the functionality by emitting packets and measuring its correctness. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate the VNF without load 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Start the traffic load 4. Validate the traffic flows with the VNF provided functionality 5. Stop Traffic 6. Stop VNF (--> time stamp) 7. Validate VNF instantiation state is INSTANTIATED and VNF state is STOPPED (--> time stamp when correct state reached) 8. Start the traffic load 9. Validate no traffic flows with the VNF provided functionality 10. Stop Traffic 11. Start VNF (--> time stamp) 12. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 13. Start the traffic load 14. Validate the traffic flows with the VNF provided functionality 15. Stop Traffic 16. Ensure that no traffic flows once stop is completed 17. Terminate VNF 18. Measure time it takes to start |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.STATE.START.002 VNF Start under low traffic load and measure start time | |
| Purpose: | Verify VNF can be activated, processes packets after the activation and measure the activation time |
| Description | After the VNF is instantiated, the VNF state is “STARTED”. A command STOP needs to be issued to get the VNF state into STOPPED so that the test can be executed. To allow processing of its functionality, it must be activated. Via the MANO (VNFM or NFVO) the command “Start” is been issued to bring the VNF into an active state and validate the functionality by emitting packets and measuring its correctness. The traffic gets emitted before the “START” command to allow the measure of the activation time and the quality (dropped packets after receiving the first packet) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Measure the time it takes for the activation. We may need to go through multiple iterations to get an average time.  Sequence:   1. Instantiate the VNF without load 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Start the low traffic load 4. Validate traffic goes through 5. Stop traffic 6. Stop VNF (--> time stamp) 7. Validate VNF instantiation state is INSTANTIATED and VNF state is STOPPED (--> time stamp when correct state reached) 8. Start the low traffic load 9. Validate no traffic goes through 10. Start VNF (--> time stamp) 11. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 12. Validate the provided functionality and calculate the time for the activation (first time stamp arrival) 13. Stop VNF 14. Ensure that no traffic flows once stop is completed 15. Stop Traffic Load 16. Terminate VNF |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.STATE.START.003 VNF Start with normal traffic load and measure start time | |
| Purpose: | Verify VNF can be activated under normal traffic load, processes packets after the activation and measure the activation time |
| Description | After the VNF is instantiated, the VNF state is “STARTED”. A command STOP needs to be issued to get the VNF state into STOPPED so that the test can be executed. To allow processing of its functionality, it must be activated. Via the MANO (VNFM or NFVO) the command “Start” is been issued to bring the VNF into an active state and validate the functionality by emitting packets and measuring its correctness. The normal traffic gets emitted before the START command to allow the measure of the activation time and the quality (dropped packets after receiving the first packet). The traffic load does not cause a scaling event in the system, but it is close to the limit to cause a scaling event. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Measure the time it takes for the activation and forwards the normal traffic. We may need to go through multiple iterations to get an average time.  Sequence:   1. Instantiate the VNF without load 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Start the low traffic load 4. Validate traffic goes through 5. Stop traffic 6. Stop VNF (--> time stamp) 7. Validate VNF instantiation state is INSTANTIATED and VNF state is STOPPED (--> time stamp when correct state reached) 8. Start the normal traffic load 9. Validate no traffic goes through 10. Start VNF (--> time stamp) 11. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 12. Validate the provided functionality and calculate the time for the activation (first time stamp arrival and no dropped packets) 13. Validate no scaling had occurred 14. Stop VNF 15. Ensure that no traffic flows once stop is completed 16. Stop Traffic Load 17. Terminate VNF |
| Extensions: |  |
| Notes: |  |

## VNF State Management: Stop

Once a VNF is in the state “Active”, it can be instructed to change the state to “Inactive” via the command Stop. Only active VNFs should respond to this command. Once deactivated, the VNF stops its normal operation and no traffic flow will occur.

|  |  |
| --- | --- |
| TC.VNF.STATE.STOP.001 VNF Stop without traffic load and measure stop time | |
| Purpose: | Verify VNF can be deactivated without traffic load, processes no packets after the deactivation and measure the deactivation time |
| Description | After the VNF is instantiated, the VNF state is “STARTED”. In this state the VNF can process packets. Via the MANO (VNFM or NFVO) the command “Stop” is been issued to bring the VNF into an inactive state and validate the functionality has been deactivated by emitting packets after the deactivation and not packets should be processed. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Measure the time it takes for the stop. We may need to go through multiple iterations to get an average time.  Sequence:   1. Instantiate the VNF without load 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Start the low traffic load 4. Validate the provided functionality (--> no dropped packets) 5. Stop Traffic 6. Validate no traffic flows 7. Stop VNF (--> time stamp) 8. Validate VNF instantiation state is INSTANTIATED and VNF state is STOPPED (--> time stamp when correct state reached) 9. Start the low traffic load 10. Ensure that no traffic flows once stop is completed 11. Stop Traffic Load 12. Calculate the time for the deactivation 13. Terminate VNF |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.STATE.STOP.002 VNF Stop with low traffic load and measure stop time | |
| Purpose: | Verify VNF can be deactivated under low traffic load, processes no packets after the deactivation and measure the deactivation time |
| Description | After the VNF is instantiated, the VNF state is “STARTED”. Via the MANO (VNFM or NFVO) the command “Stop” is been issued to bring the VNF into an inactive state and validate the functionality has been deactivated by emitting packets and no packets should be processed at the end. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Measure the time it takes for the stop. We may need to go through multiple iterations to get an average time.  Sequence:   1. Instantiate the VNF without load 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Start the low traffic load 4. Validate the provided functionality (--> no dropped packets) 5. Stop VNF (--> time stamp) 6. Validate VNF instantiation state is INSTANTIATED and VNF state is STOPPED (--> time stamp when correct state reached) 7. Ensure that no traffic flows once stop is completed (--> time stamp) 8. Stop Traffic Load 9. Calculate the time for the deactivation (last time stamp arrival) 10. Terminate VNF |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.STATE.STOP.003 VNF Stop with normal traffic load and measure stop time | |
| Purpose: | Verify VNF can be deactivated under max traffic load, processes no packets after the deactivation and measure the deactivation time |
| Description | After the VNF is instantiated, the VNF state is “STARTED”. Via the MANO (VNFM or NFVO) the command “Stop” is been issued to bring the VNF into an inactive state and validate the functionality has been deactivated by emitting packets and not packets should be processed at the end. The traffic load does not cause a scaling event in the system, but it is close to the limit to cause a scaling event. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Measure the time it takes for the stop. We may need to go through multiple iterations to get an average time.  Sequence:  Measure the time it takes for the stop. We may need to go through multiple iterations to get an average time.  Sequence:   1. Instantiate the VNF without load 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Start the normal traffic load 4. Validate the provided functionality (--> no dropped packets) 5. Stop VNF (--> time stamp) 6. Validate VNF instantiation state is INSTANTIATED and VNF state is STOPPED (--> time stamp when correct state reached) 7. Ensure that no traffic flows once stop is completed (--> time stamp) 8. Stop Traffic Load 9. Calculate the time for the deactivation (last time stamp arrival) 10. Terminate VNF |
| Extensions: |  |
| Notes: |  |

## VNF State Management: Termination

Once a VNF is instantiated is has allocated and assigned resources. To free up the resources for an unused VNF, it must be terminated. This causes that all resources get released.

|  |  |
| --- | --- |
| TC.VNF.STATE.TERM.001 VNF terminate from VNF state STARTED without traffic load | |
| Purpose: | Verify VNF can be terminated in VNF state “STARTED” without traffic load and all vResources have been released |
| Description | After the VNF is instantiated, the VNF state is “STARTED”. Via the MANO (VNFM or NFVO) the command “Terminate” is been issued to remove the VNF. After completion of the termination all reserved vResources of the VNF must be released. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Measure the time it takes for the termination. We may need to go through multiple iterations to get an average time.  Sequence:   1. Instantiate the VNF without load 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Start the low traffic load 4. Validate the provided functionality (--> no dropped packets) 5. Stop traffic load 6. Ensure that no traffic flows once stop is completed 7. Terminate VNF (--> time stamp) 8. Validate VNF is terminate and all resources have been released (🡪 time stamp) 9. Calculate the time for the termination (last time stamp arrival) |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.STATE.TERM.002 VNF terminate from VNF state STARTED with low traffic load | |
| Purpose: | Verify VNF can be terminated in VNF state “STARTED” under low traffic load and all vResources have been released |
| Description | After the VNF is instantiated, the VNF state is “STARTED”. Via the MANO (VNFM or NFVO) the command “Terminate” is been issued to remove the VNF while low traffic is generated. After completion of the termination all reserved vResources of the VNF must be released and no traffic should flow. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Measure the time it takes for the termination. We may need to go through multiple iterations to get an average time.  Sequence:   1. Instantiate VNF without load (--> time stamp) 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Start the low traffic load 4. Validate the provided functionality and all traffic goes through (🡪 no dropped packets) 5. Terminate VNF (🡪 time stamp) 6. Validate VNF is terminated and all resources have been released 7. Ensure that no traffic flows once stop is completed 8. Calculate the time for the termination (🡪 last time stamp arrival) |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.STATE.TERM.003 VNF terminate from VNF state STARTED with normal traffic load | |
| Purpose: | Verify VNF can be terminated in state “STARTED” under normal traffic load and all vResources have been released |
| Description | After the VNF is instantiated, the VNF state is “STARTED”. Via the MANO (VNFM or NFVO) the command “Terminate” is been issued to remove the VNF while normal traffic is generated. After completion of the termination all reserved vResources of the VNF must be released and no traffic should flow. The traffic load does not cause a scaling event in the system, but it is close to the limit to cause a scaling event. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Measure the time it takes for the termination. We may need to go through multiple iterations to get an average time.  Sequence:   1. Instantiate VNF without load (--> time stamp) 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Start the normal traffic load 4. Validate the provided functionality and all traffic goes through (🡪 no dropped packets) 5. Terminate VNF (🡪 time stamp) 6. Validate VNF is terminated and all resources have been released 7. Ensure that no traffic flows once stop is completed 8. Calculate the time for the termination (🡪 last time stamp arrival) |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.STATE.TERM.004 VNF terminate from state STOPPED without load | |
| Purpose: | Verify VNF can be terminated in state “STOPPED” without load and all vResources have been released |
| Description | After the VNF is instantiated and configured, the VNF state is “STARTED”. A command stop must be issued to get the VNF into the VNF state “STOPPED”. Via the MANO (VNFM or NFVO) the command “Terminate” is been issued to remove the VNF without traffic. After completion of the termination all reserved vResources of the VNF must be released. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Measure the time it takes for the termination. We may need to go through multiple iterations to get an average time.  Sequence:   1. Instantiate the VNF without load 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Start the low traffic load 4. Validate the provided functionality (--> no dropped packets) 5. Stop traffic 6. Validate no traffic goes through 7. Stop VNF (--> time stamp) 8. Validate VNF instantiation state is INSTANTIATED and VNF state is STOPPED (--> time stamp when correct state reached) 9. Validate VNF is terminate and all resources have been released (--> time stamp) 10. Calculate the time for the termination |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.STATE.TERM.005 VNF terminate from VNF state STOPPED with low traffic load | |
| Purpose: | Verify VNF can be terminated in VNF state “STOPPED” with low traffic load and all vResources have been released |
| Description | After the VNF is instantiated and configured, the VNF state is “STARTED”. A command stop must be issued to get the VNF into the VNF state “STOPPED”. Via the MANO (VNFM or NFVO) the command “Terminate” is been issued to remove the VNF. Traffic is generated at the same time to check if the VNF would temporarily go into a state “STARTED”. After completion of the termination all reserved vResources of the VNF must be released. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Measure the time it takes for the termination. We may need to go through multiple iterations to get an average time.  Sequence:   1. Instantiate the VNF without load 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Start the low traffic load 4. Validate the provided functionality (--> no dropped packets) 5. Stop VNF (--> time stamp) 6. Validate VNF instantiation state is INSTANTIATED and VNF state is STOPPED (--> time stamp when correct state reached) 7. Validate no traffic goes through 8. Terminate VNF (--> time stamp) 9. Validate VNF is terminate and all resources have been released (--> time stamp) 10. Validate no traffic has been forwarded while the termination 11. Stop traffic 12. Calculate the time for the termination |
| Extensions: |  |
| Notes: |  |

## VNF State Management: Upgrade, Update or Rollback

VNF provider will be faced to release new software versions of their products to expose new functionality, increase performance or fix bugs. The following test cases validate the process of

1. Upgrading a VNF
2. Updating a VNF
3. Rollback a VNF

|  |  |
| --- | --- |
| TC.VNF.STATE.UPDATE.001 VNF upgrade, update or rollback in VNF state STARTED without traffic | |
| Purpose: | Verify VNF can be upgraded, updated or rolled back in VNF state “STARTED” and returns into the same state back afterwards |
| Description | After the VNF is instantiated, the VNF state is “STARTED”. The command VNF upgrade, update or rollback gets initiated. After completion, the new VNF should be still in the same state. After termination, all reserved vResources of the VNF must be released. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Ensure VNF doesn't receive the required configuration from the EM or configuration file  Sequence:   1. Ensure EM is up or config file is present 2. Instantiate the VNF 3. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 4. Start the low traffic load 5. Validate the provided functionality (--> no dropped packets) 6. Initiate the upgrade, update or rollback process 7. Validate VNF new version is in place and VNF instantiation state is INSTANTIATED and VNF state is STARTED 8. Validate the provided functionality (--> no dropped packets) 9. Stop Traffic 10. Terminate VNF 11. Validate that all vResources have been released |
| Extensions: |  |
| Notes: | The test case includes three different sub test cases:   1. Upgrade a VNF 2. Update a VNF 3. Rollback a VNF |

|  |  |
| --- | --- |
| TC.VNF.STATE.UPDATE.002 VNF upgrade, update or rollback in VNF state STOPPED | |
| Purpose: | Verify VNF can be upgraded, updated or rolled back in VNF state “STOPPED” and returns into the same state back afterwards |
| Description | After the VNF is instantiated and configured, the VNF gets stopped and his state ends in “STOPPED”. The command VNF upgrade, update or rollback gets initiated. After completion, the new VNF should be still in the same state. The EM or the config file should be applied to the new VNF instance. After termination, all reserved vResources of the VNF must be released. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Ensure VNF receives the required configuration from the EM or configuration file  Sequence:   1. Ensure EM is up or config file is present 2. Instantiate VNF 3. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 4. Stop VNF 5. Validate VNF instantiation state is INSTANTIATED and VNF state is STOPPED (--> time stamp when correct state reached) 6. Start the traffic load 7. Validate no traffic goes through 8. Initiate the upgrade, update or rollback process 9. Validate VNF new version is in place and VNF state is STOPPED 10. Validate no traffic has been forwarded 11. Stop Traffic 12. Terminate VNF 13. Validate that all vResources have been released |
| Extensions: |  |
| Notes: | The test case includes three different sub test cases:   * Upgrade a VNF * Update a VNF * Rollback a VNF |

|  |  |
| --- | --- |
| TC.VNF.STATE.UPDATE.004 VNF upgrade, update or rollback in VNF state STARTED with low traffic | |
| Purpose: | Verify VNF can be upgraded, updated or rolled back in VNF state “STARTED” and returns into the same state back afterwards. The process is done under low traffic to determine the service disruption time. |
| Description | After the VNF is instantiated and configured, the VNF state is “STARTED”. The command VNF upgrade, update or rollback gets initiated. After completion, the new VNF should be still in the same state. The EM or the config file should be applied to the new VNF instance and return to the active state by processing packets. After termination, all reserved vResources of the VNF must be released. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Ensure VNF receive the required configuration from the EM or configuration file  Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Start the low traffic load 4. Validate traffic goes through (--> no dropped packets) 5. Initiate the upgrade, update or rollback process 6. Validate VNF new version is active and VNF state is STARTED 7. Validate traffic gets forwarded and measure the length of service disruption 8. Stop Traffic 9. Terminate VNF 10. Validate that all vResources have been released |
| Extensions: |  |
| Notes: | The test case includes three different sub test cases:   * Upgrade a VNF * Update a VNF * Rollback a VNF |

|  |  |
| --- | --- |
| TC.VNF.STATE.UPDATE.005 VNF upgrade, update or rollback in state Active with normal traffic | |
| Purpose: | Verify VNF can be upgraded, updated or rolled back in state “STARTED” and returns into the same state back afterwards. The process is done under normal traffic to determine the service disruption time. The traffic load does not cause a scaling event in the system, but it is close to the limit to cause a scaling event. |
| Description | After the VNF is instantiated and configured, the VNF state is “STARTED”. The command VNF upgrade, update or rollback gets initiated. After completion, the new VNF should be still in the same state. The EM or the config file should be applied to the new VNF instance and return to the active state by processing packets. After termination, all reserved vResources of the VNF must be released. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Ensure VNF receive the required configuration from the EM or configuration file  Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Start the normal traffic load 4. Validate traffic goes through (--> no dropped packets) 5. Initiate the upgrade, update or rollback process 6. Validate VNF new version is active and VNF state is STARTED 7. Validate traffic gets forwarded and measure the length of service disruption 8. Stop Traffic 9. Terminate VNF 10. Validate that all vResources have been released |
| Extensions: |  |
| Notes: | The test case includes three different sub test cases:   * Upgrade a VNF * Update a VNF * Rollback a VNF |

# Virtualized Resource Management

The management of virtualized resources is an integral component of the VNF Lifecycle Management. It encompasses

1. VNF scaling by adding and removing virtual resources

A MANO can scale a VNF in two different ways:

* 1. Adding and removing VNF
  2. Adding and removing of VNFC to/from an existing VNF

1. Virtual resource migration
2. Virtual network related management
3. Virtual storage related management

## vResource Scaling Up/Down

A VNF can adapt to the demands of the applied traffic. It could increase it performance by allocation of additional vResources and when the traffic lowers, it could release vResources. This process is called scaling. The resource increase/decrease without changes to VNFC and VNF count is scaling Up/Down. The following test cases address the validation of the VNF scaling for Up/Down.

### Scale-up VNF instance

|  |  |
| --- | --- |
| TC.VNF.SCALE.UP.001 Scale-up VNF instance | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO (VNFM or NFVO) should report the additional allocated virtualized resources. The MANO should have added more resources to the existing VNFCs. This process may require to reboot/restart the VNFC and losing the configuration. So the EM may require to reconfigure the VNF. Therefore, traffic should be generated to validate that the VNFCs can process packets correctly. The trigger for this event to occur could be by different options:   * VNFM/MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate VNF has resized, has higher capacity and determine the service disruption and no new addition VNFC have been added 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-up the VNF doesn’t add new VNFCs. It keeps the number of VNFC constant and just adds additional resources.  The scaling trigger could be one of the following:   * VNFM/MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling   Each of this trigger can be seen a sub test cases. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.UP.001.1 Scale-up VNF instance with On-demand Scaling event generated by MANO | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO (VNFM or NFVO) should report the additional allocated virtualized resources. The MANO should have added more resources to the existing VNFCs. This process may require to reboot/restart the VNFC and losing the configuration. So the EM may require to reconfigure the VNF. Therefore, traffic should be generated to validate that the VNFCs can process packets correctly. The trigger for this event to occur is:   * MANO generated events - Auto Scaling ( e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate VNF has resized, has higher capacity and determine the service disruption and no new addition VNFC have been added 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-up the VNF doesn’t add new VNFCs. It keeps the number of VNFC constant and just adds additional resources. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.UP.001.2 Scale-up VNF instance with On-demand Scaling event generated by VNF | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO (VNFM or NFVO) should report the additional allocated virtualized resources. The MANO should have added more resources to the existing VNFCs. This process may require to reboot/restart the VNFC and losing the configuration. So the EM may require to reconfigure the VNF. Therefore, traffic should be generated to validate that the VNFCs can process packets correctly. The trigger for this event to occur is:   * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate VNF has resized, has higher capacity and determine the service disruption and no new addition VNFC have been added 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-up the VNF doesn’t add new VNFCs. It keeps the number of VNFC constant and just adds additional resources. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.UP.001.3 Scale-up VNF instance with On-demand Scaling event generated by VIM | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO (VNFM or NFVO) should report the additional allocated virtualized resources. The MANO should have added more resources to the existing VNFCs. This process may require to reboot/restart the VNFC and losing the configuration. So the EM may require to reconfigure the VNF. Therefore, traffic should be generated to validate that the VNFCs can process packets correctly. The trigger for this event to occur is:   * VIM generated event – On-demand Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate VNF has resized, has higher capacity and determine the service disruption and no new addition VNFC have been added 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-up the VNF doesn’t add new VNFCs. It keeps the number of VNFC constant and just adds additional resources. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.UP.001.4 Scale-up VNF instance with On-demand Scaling event generated by EM | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO (VNFM or NFVO) should report the additional allocated virtualized resources. The MANO should have added more resources to the existing VNFCs. This process may require to reboot/restart the VNFC and losing the configuration. So the EM may require to reconfigure the VNF. Therefore, traffic should be generated to validate that the VNFCs can process packets correctly. The trigger for this event to occur is:   * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate VNF has resized, has higher capacity and determine the service disruption and no new addition VNFC have been added 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-up the VNF doesn’t add new VNFCs. It keeps the number of VNFC constant and just adds additional resources. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.UP.001.5 Scale-up VNF instance with Manual Scaling event generated by MANO | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO (VNFM or NFVO) should report the additional allocated virtualized resources. The MANO should have added more resources to the existing VNFCs. This process may require to reboot/restart the VNFC and losing the configuration. So the EM may require to reconfigure the VNF. Therefore, traffic should be generated to validate that the VNFCs can process packets correctly. The trigger for this event to occur is:   * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate VNF has resized, has higher capacity and determine the service disruption and no new addition VNFC have been added 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-up the VNF doesn’t add new VNFCs. It keeps the number of VNFC constant and just adds additional resources. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.UP.001.6 Scale-up VNF instance with Manual Scaling event generated by VNF | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO (VNFM or NFVO) should report the additional allocated virtualized resources. The MANO should have added more resources to the existing VNFCs. This process may require to reboot/restart the VNFC and losing the configuration. So the EM may require to reconfigure the VNF. Therefore, traffic should be generated to validate that the VNFCs can process packets correctly. The trigger for this event to occur is:   * VNF generated event - Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate VNF has resized, has higher capacity and determine the service disruption and no new addition VNFC have been added 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-up the VNF doesn’t add new VNFCs. It keeps the number of VNFC constant and just adds additional resources. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.UP.001.7 Scale-up VNF instance with Manual Scaling event generated by EM | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO (VNFM or NFVO) should report the additional allocated virtualized resources. The MANO should have added more resources to the existing VNFCs. This process may require to reboot/restart the VNFC and losing the configuration. So the EM may require to reconfigure the VNF. Therefore, traffic should be generated to validate that the VNFCs can process packets correctly. The trigger for this event to occur is:   * EM generated event – Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate VNF has resized, has higher capacity and determine the service disruption and no new addition VNFC have been added 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-up the VNF doesn’t add new VNFCs. It keeps the number of VNFC constant and just adds additional resources. |

### Max scale-up VNF instance

|  |  |
| --- | --- |
| TC.VNF.SCALE.UP.002 Max scale-up VNF instance | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling to the max higher performance is the objective of this test. |
| Description | Once the scaling increase is completed, the VNFM should report the additional allocated virtualized resources. The VNFM should have added more resources to one or more of the existing VNFCs. This process may require to reboot/restart the VNFC and losing the configuration. So the EM may require to reconfigure the VNF. Therefore, traffic should be generated to validate that the VNFCs can process packets correctly. The trigger for this event to occur could be by different options:   * VNFM/MANO generated events - Auto Scaling ( e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling ( e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the maximum 6. Validate VNF has resized to the max and has max capacity (e.g., generate higher traffic load) 7. Determine the service disruption during the resizing to the max and the time it takes to complete it 8. Stop traffic 9. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-up the VNF doesn’t add new VNFCs. It keeps the number of VNFC constant and just adds additional resources.  The scaling trigger could be one of the following:   * VNFM/MANO generated events - Auto Scaling ( e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |

### Scale-down VNF instance

|  |  |
| --- | --- |
| TC.VNF.SCALE.UP.003 Scale-down VNF instance | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling down to the lower performance is the objective of this test. |
| Description | Once the scaling decrease is completed, the VNFM should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the VNFM needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur could be by different options:   * VNFM/MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the maximum 6. Validate VNF has resized to the max 7. Determine if and length of service disruption 8. Generate max traffic load 9. Validate max capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate VNF has released the resources 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-down the VNF doesn’t remove VNFCs. It keeps the number of VNFC constant and just removes resources.  Each of this are sub test cases:  The scaling trigger could be one of the following:   * VNFM/MANO generated events - Auto Scaling ( e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling ( e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling   Each of this trigger can be seen a sub test cases. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.UP.003.1 Scale-down VNF instance with On-demand Scaling event generated by MANO | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling down to the lower performance is the objective of this test. |
| Description | Once the scaling decrease is completed, the VNFM should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the VNFM needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur could is the following:   * VNFM/MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the maximum 6. Validate VNF has resized to the max 7. Determine if and length of service disruption 8. Generate max traffic load 9. Validate max capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate VNF has released the resources 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-down the VNF doesn’t remove VNFCs. It keeps the number of VNFC constant and just removes resources. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.UP.003.2 Scale-down VNF instance with On-demand Scaling event generated by VNF | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling down to the lower performance is the objective of this test. |
| Description | Once the scaling decrease is completed, the VNFM should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the VNFM needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur could is the following:   * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the maximum 6. Validate VNF has resized to the max 7. Determine if and length of service disruption 8. Generate max traffic load 9. Validate max capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate VNF has released the resources 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-down the VNF doesn’t remove VNFCs. It keeps the number of VNFC constant and just removes resources. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.UP.003.3 Scale-down VNF instance with On-demand Scaling event generated by VIM | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling down to the lower performance is the objective of this test. |
| Description | Once the scaling decrease is completed, the VNFM should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the VNFM needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur could is the following:   * VIM generated event – On-demand Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the maximum 6. Validate VNF has resized to the max 7. Determine if and length of service disruption 8. Generate max traffic load 9. Validate max capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate VNF has released the resources 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-down the VNF doesn’t remove VNFCs. It keeps the number of VNFC constant and just removes resources. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.UP.003.4 Scale-down VNF instance with On-demand Scaling event generated by EM | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling down to the lower performance is the objective of this test. |
| Description | Once the scaling decrease is completed, the VNFM should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the VNFM needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur could is the following:   * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the maximum 6. Validate VNF has resized to the max 7. Determine if and length of service disruption 8. Generate max traffic load 9. Validate max capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate VNF has released the resources 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-down the VNF doesn’t remove VNFCs. It keeps the number of VNFC constant and just removes resources. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.UP.003.5 Scale-down VNF instance with Manual Scaling event generated by MANO | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling down to the lower performance is the objective of this test. |
| Description | Once the scaling decrease is completed, the VNFM should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the VNFM needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur could is the following:   * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the maximum 6. Validate VNF has resized to the max 7. Determine if and length of service disruption 8. Generate max traffic load 9. Validate max capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate VNF has released the resources 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-down the VNF doesn’t remove VNFCs. It keeps the number of VNFC constant and just removes resources. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.UP.003.6 Scale-down VNF instance with Manual Scaling event generated by VNF | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling down to the lower performance is the objective of this test. |
| Description | Once the scaling decrease is completed, the VNFM should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the VNFM needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur could is the following:   * VNF generated event - Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the maximum 6. Validate VNF has resized to the max 7. Determine if and length of service disruption 8. Generate max traffic load 9. Validate max capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate VNF has released the resources 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-down the VNF doesn’t remove VNFCs. It keeps the number of VNFC constant and just removes resources. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.UP.003.6 Scale-down VNF instance with Manual Scaling event generated by EM | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling down to the lower performance is the objective of this test. |
| Description | Once the scaling decrease is completed, the VNFM should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the VNFM needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur could is the following:   * EM generated event - Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the maximum 6. Validate VNF has resized to the max 7. Determine if and length of service disruption 8. Generate max traffic load 9. Validate max capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate VNF has released the resources 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-down the VNF doesn’t remove VNFCs. It keeps the number of VNFC constant and just removes resources. |

### Scale-Up upto Max vResource limit

|  |  |
| --- | --- |
| TC.VNF.SCALE.UP.004 Max vResource limit reached before max VNFD limit for Scale-Up | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling to the max higher performance is the objective of this test. But the vResource limit is reached earlier than the max VNFD limit. vMemory, vStorage, vNIC and vCPU are just examples how the VNF could be limited by. The VNFM must handle this corner case. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources to one or more VNFCs. This process may require to reboot/restart the VNF and losing the configuration. So the EM may require to reconfigure the VNFM. Therefore, traffic should be generated to validate that the VNF can process packets correctly. The trigger for this event to occur could be by different options:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Ensure NFVI has less vResources as the max VNFD indicates 2. Instantiate VNF 3. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 4. Generate low traffic load 5. Validate that traffic flows through without issues (--> no dropped packets) 6. Trigger a resize of the NFV resources to the maximum (limited by NFVI) 7. Validate VNF has resized to the max limit 8. Generate higher traffic load 9. Validate that the load has no loss 10. Determine the service disruption during the resizing to the max and the time it takes to complete it 11. Stop traffic 12. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could have reached its limit:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   The scaling trigger could be one of the following:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |

### Scale-Up/Down for virtualized specialized hardware acceleration

|  |  |
| --- | --- |
| TC.VNF.SCALE.UP.005 Removal of virtualized specialized hardware acceleration for Scale-Up | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling to the lower performance is the objective of this test by removing vHardware resources. The MANO must handle this case of vNIC resource management. |
| Description | If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the MANO needs to have the capability to remove these resources when decreasing the scaling. The MANO should report reduced resources after the completion. The EM may apply new configurations. The functionality of the VNF should be validated. Lower scalability of the VNF service should be the result. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to use more specialized hardware resources 6. Validate VNF has resized 7. Generate higher traffic load 8. Validate increased its capacity by no traffic loss 9. Validate that MANO has allocated more specialized hardware resources 10. Generate low traffic load 11. Trigger a resize of the NFV resources to use less specialized hardware resources 12. Validate VNF has resized and has decreased its capacity 13. Validate that MANO has allocated less specialized hardware resources and the previous specialized hardware resources have been freed up 14. Determine the service disruption during the resizing 15. Validate no dropped packets 16. Stop traffic 17. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be removed:   * vNIC * Other specialized hardware   The scaling trigger could be one of the following:   * MANO generated events - Auto Scaling ( e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling ( e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |

## vResource Scaling Out/In of VNFC

The MANO can adapt a VNF to the demands of the applied traffic by adding and removing VNFC(s). It could increase it performance by allocation of additional vResources and when the traffic lowers, it could release vResources. This process is called scaling. The addition and removal of vResources is considered **scaling Out/In by** **increasing or decreasing the number of VNFC of a VNF**. The following test cases address the validation of the VNFC scaling Out/In.

### Scale-Out VNFC instance

|  |  |
| --- | --- |
| TC.VNFC.SCALE.OUT.001 Scale-Out VNFC instance | |
| Purpose: | A VNF should have the capability to dynamically scale by adding additional VNFC(s) based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources by increasing number of VNFCs. The EM may require to reconfigure the VNFCs. Therefore, traffic should be generated to validate that the VNFCs can process packets correctly. The trigger for this event to occur could be by different options:   * MANO generated events - Auto Scaling ( e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling ( e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate VNF has resized by adding VNFCs, has higher capacity and determine the service disruption 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-out the VNF adds new VNFCs.  The scaling trigger could be one of the following:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling   Each of this trigger can be seen a sub test cases. |

|  |  |
| --- | --- |
| TC.VNFC.SCALE.OUT.001.1 Scale-Out VNFC instance with On-demand Scaling event generated by MANO | |
| Purpose: | A VNF should have the capability to dynamically scale by adding additional VNFC(s) based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources by increasing number of VNFCs. The EM may require to reconfigure the VNFCs. Therefore, traffic should be generated to validate that the VNFCs can process packets correctly. The trigger for this event to occur is the following:   * MANO generated events - Auto Scaling ( e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate VNF has resized by adding VNFCs, has higher capacity and determine the service disruption 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-out the VNF adds new VNFCs. |

|  |  |
| --- | --- |
| TC.VNFC.SCALE.OUT.001.2 Scale-Out VNFC instance with On-demand Scaling event generated by VNF | |
| Purpose: | A VNF should have the capability to dynamically scale by adding additional VNFC(s) based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources by increasing number of VNFCs. The EM may require to reconfigure the VNFCs. Therefore, traffic should be generated to validate that the VNFCs can process packets correctly. The trigger for this event to occur is the following:   * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate VNF has resized by adding VNFCs, has higher capacity and determine the service disruption 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-out the VNF adds new VNFCs. |

|  |  |
| --- | --- |
| TC.VNFC.SCALE.OUT.001.3 Scale-Out VNFC instance with On-demand Scaling event generated by VIM | |
| Purpose: | A VNF should have the capability to dynamically scale by adding additional VNFC(s) based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources by increasing number of VNFCs. The EM may require to reconfigure the VNFCs. Therefore, traffic should be generated to validate that the VNFCs can process packets correctly. The trigger for this event to occur is the following:   * VIM generated event – On-demand Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate VNF has resized by adding VNFCs, has higher capacity and determine the service disruption 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-out the VNF adds new VNFCs. |

|  |  |
| --- | --- |
| TC.VNFC.SCALE.OUT.001.4 Scale-Out VNFC instance with On-demand Scaling event generated by EM | |
| Purpose: | A VNF should have the capability to dynamically scale by adding additional VNFC(s) based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources by increasing number of VNFCs. The EM may require to reconfigure the VNFCs. Therefore, traffic should be generated to validate that the VNFCs can process packets correctly. The trigger for this event to occur is the following:   * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate VNF has resized by adding VNFCs, has higher capacity and determine the service disruption 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-out the VNF adds new VNFCs. |

|  |  |
| --- | --- |
| TC.VNFC.SCALE.OUT.001.5 Scale-Out VNFC instance with Manual Scaling event generated by MANO | |
| Purpose: | A VNF should have the capability to dynamically scale by adding additional VNFC(s) based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources by increasing number of VNFCs. The EM may require to reconfigure the VNFCs. Therefore, traffic should be generated to validate that the VNFCs can process packets correctly. The trigger for this event to occur is the following:   * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate VNF has resized by adding VNFCs, has higher capacity and determine the service disruption 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-out the VNF adds new VNFCs. |

|  |  |
| --- | --- |
| TC.VNFC.SCALE.OUT.001.6 Scale-Out VNFC instance with Manual Scaling event generated by VNF | |
| Purpose: | A VNF should have the capability to dynamically scale by adding additional VNFC(s) based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources by increasing number of VNFCs. The EM may require to reconfigure the VNFCs. Therefore, traffic should be generated to validate that the VNFCs can process packets correctly. The trigger for this event to occur is the following:   * VNF generated event - Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate VNF has resized by adding VNFCs, has higher capacity and determine the service disruption 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-out the VNF adds new VNFCs. |

|  |  |
| --- | --- |
| TC.VNFC.SCALE.OUT.001.7 Scale-Out VNFC instance with Manual Scaling event generated by EM | |
| Purpose: | A VNF should have the capability to dynamically scale by adding additional VNFC(s) based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources by increasing number of VNFCs. The EM may require to reconfigure the VNFCs. Therefore, traffic should be generated to validate that the VNFCs can process packets correctly. The trigger for this event to occur is the following:   * EM generated event – Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate VNF has resized by adding VNFCs, has higher capacity and determine the service disruption 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-out the VNF adds new VNFCs. |

### Max scale-out VNFC instance

|  |  |
| --- | --- |
| TC.VNFC.SCALE.OUT.002 Max scale-out VNFC instance | |
| Purpose: | A VNF should have the capability to dynamically scale by adding additional VNFC(s) based on the required load. The scaling to the max higher performance is the objective of this test. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources by adding more VNFCs. The EM may require to configure the VNFCs. Therefore, traffic should be generated to validate that the VNFCs can process packets correctly. The trigger for this event to occur could be one of options:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the maximum 6. Validate VNF has resized to the max and has max capacity (e.g., generate higher traffic load) and added new VNFCs 7. Determine the service disruption during the resizing to the max and the time it takes to complete it 8. Stop traffic 9. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-out the VNF adds new VNFCs.  The scaling trigger could be one of the following:   * VNFM/MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |

### Scale-in VNFC instance

|  |  |
| --- | --- |
| TC.VNFC.SCALE.OUT.003 Scale-in VNFC instance | |
| Purpose: | Once the VNFC scaling decrease is completed, the MANO should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the MANO needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur could be by different options:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |
| Description | A VNF should have the capability to dynamically scale based on the required load. The scaling in to the lower performance is the objective of this test. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the next level 6. Validate VNF has resized to the next level 7. Determine if and length of service disruption 8. Generate traffic load the next level 9. Validate capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate VNF has released the resources and decreased the VNFCs 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-in the VNF removes VNFCs.  Each of this are sub test cases.  The scaling trigger could be one of the following:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling   Each of this trigger can be seen a sub test cases. |

|  |  |
| --- | --- |
| TC.VNFC.SCALE.OUT.003.1 Scale-in VNFC instance with On-demand Scaling event generated by MANO | |
| Purpose: | Once the VNFC scaling decrease is completed, the MANO should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the MANO needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur is the following:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) |
| Description | A VNF should have the capability to dynamically scale based on the required load. The scaling in to the lower performance is the objective of this test. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the next level 6. Validate VNF has resized to the next level 7. Determine if and length of service disruption 8. Generate traffic load the next level 9. Validate capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate VNF has released the resources and decreased the VNFCs 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-in the VNF removes VNFCs. |

|  |  |
| --- | --- |
| TC.VNFC.SCALE.OUT.003.2 Scale-in VNFC instance with On-demand Scaling event generated by VNF | |
| Purpose: | Once the VNFC scaling decrease is completed, the MANO should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the MANO needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur is the following:   * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) |
| Description | A VNF should have the capability to dynamically scale based on the required load. The scaling in to the lower performance is the objective of this test. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the next level 6. Validate VNF has resized to the next level 7. Determine if and length of service disruption 8. Generate traffic load the next level 9. Validate capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate VNF has released the resources and decreased the VNFCs 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-in the VNF removes VNFCs. |

|  |  |
| --- | --- |
| TC.VNFC.SCALE.OUT.003.3 Scale-in VNFC instance with On-demand Scaling event generated by VIM | |
| Purpose: | Once the VNFC scaling decrease is completed, the MANO should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the MANO needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur is the following:   * VIM generated event – On-demand Scaling |
| Description | A VNF should have the capability to dynamically scale based on the required load. The scaling in to the lower performance is the objective of this test. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the next level 6. Validate VNF has resized to the next level 7. Determine if and length of service disruption 8. Generate traffic load the next level 9. Validate capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate VNF has released the resources and decreased the VNFCs 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-in the VNF removes VNFCs. |

|  |  |
| --- | --- |
| TC.VNFC.SCALE.OUT.003.4 Scale-in VNFC instance with On-demand Scaling event generated by EM | |
| Purpose: | Once the VNFC scaling decrease is completed, the MANO should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the MANO needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur is the following:   * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) |
| Description | A VNF should have the capability to dynamically scale based on the required load. The scaling in to the lower performance is the objective of this test. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the next level 6. Validate VNF has resized to the next level 7. Determine if and length of service disruption 8. Generate traffic load the next level 9. Validate capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate VNF has released the resources and decreased the VNFCs 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-in the VNF removes VNFCs. |

|  |  |
| --- | --- |
| TC.VNFC.SCALE.OUT.003.5 Scale-in VNFC instance with Manual Scaling event generated by MANO | |
| Purpose: | Once the VNFC scaling decrease is completed, the MANO should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the MANO needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur is the following:   * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) |
| Description | A VNF should have the capability to dynamically scale based on the required load. The scaling in to the lower performance is the objective of this test. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the next level 6. Validate VNF has resized to the next level 7. Determine if and length of service disruption 8. Generate traffic load the next level 9. Validate capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate VNF has released the resources and decreased the VNFCs 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-in the VNF removes VNFCs. |

|  |  |
| --- | --- |
| TC.VNFC.SCALE.OUT.003.6 Scale-in VNFC instance with Manual Scaling event generated by VNF | |
| Purpose: | Once the VNFC scaling decrease is completed, the MANO should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the MANO needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur is the following:   * VNF generated event - Manual Scaling |
| Description | A VNF should have the capability to dynamically scale based on the required load. The scaling in to the lower performance is the objective of this test. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the next level 6. Validate VNF has resized to the next level 7. Determine if and length of service disruption 8. Generate traffic load the next level 9. Validate capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate VNF has released the resources and decreased the VNFCs 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-in the VNF removes VNFCs. |

|  |  |
| --- | --- |
| TC.VNFC.SCALE.OUT.003.7 Scale-in VNFC instance with Manual Scaling event generated by EM | |
| Purpose: | Once the VNFC scaling decrease is completed, the MANO should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the MANO needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur is the following:   * EM generated event – Manual Scaling |
| Description | A VNF should have the capability to dynamically scale based on the required load. The scaling in to the lower performance is the objective of this test. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the next level 6. Validate VNF has resized to the next level 7. Determine if and length of service disruption 8. Generate traffic load the next level 9. Validate capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate VNF has released the resources and decreased the VNFCs 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-in the VNF removes VNFCs. |

### Max vResource VNFC limit reached before max VNFD limit for scale-out

|  |  |
| --- | --- |
| TC.VNFC.SCALE.OUT.004 Max vResource VNFC limit reached before max VNFD limit for scale-out | |
| Purpose: | A VNF should have the capability to dynamically scale by adding VNFC(s) based on the required load. The scaling to the max higher performance is the objective of this test. But the vResource limit is reached earlier than the max VNFD limit. vMemory, vStorage, vNIC and vCPU are just examples how the VNF could be limited by. The MANO must handle this corner case. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources and VNFCs. The EM may require to reconfigure the VNFCs. Therefore, traffic should be generated to validate that the VNF can process packets correctly. The trigger for this event to occur could be one of the options:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Ensure NFVI has less vResources as the max VNFD indicates 2. Instantiate VNF 3. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 4. Generate low traffic load 5. Validate that traffic flows through without issues (--> no dropped packets) 6. Trigger a resize of the NFV resources to the maximum (limited by NFVI) 7. Validate VNF has resized to the max limit and added new VNFCs 8. Generate higher traffic load 9. Validate that the load 10. Determine the service disruption during the resizing to the max and the time it takes to complete it 11. Stop traffic 12. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could have reached its limit:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   The scaling trigger could be one of the following:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |

### Removal of virtualized specialized hardware acceleration for VNFC scale-in

|  |  |
| --- | --- |
| TC.VNFC.SCALE.OUT.005 Removal of virtualized specialized hardware acceleration for VNFC scale-in | |
| Purpose: | A VNF should have the capability to dynamically scale VNFC(s) based on the required load. The scaling to the lower performance is the objective of this test by removing vHardware resources. The MANO must handle this case of vNIC resource management. |
| Description | If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the MANO needs to have the capability to remove these resources when decreasing the scaling. The MANO should report reduced resources after the completion. The EM may apply new configurations. The functionality of the VNF should be validated. Lower scalability of the VNF service should be the result. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to use more specialized hardware resources 6. Validate VNF has resized 7. Generate higher traffic load 8. Validate increased its capacity by no traffic loss 9. Validate that MANO has allocated more specialized hardware resources and added new VNFCs 10. Generate low traffic load 11. Trigger a resize of the NFV resources to use less specialized hardware resources 12. Validate VNF has resized and has decreased its capacity and removed VNFCs 13. Validate that MANO has allocated less specialized hardware resources and the previous specialized hardware resources have been freed up 14. Determine the service disruption during the resizing 15. Validate no dropped packets 16. Stop traffic 17. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be removed:   * vNIC * Other specialized hardware   The scaling trigger could be one of the following:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |

## vResource Scaling Out/In of VNF

The MANO can adapt the VNF to the demands of the applied traffic by adding additional VNF. It could increase it performance by allocation of additional vResources and when the traffic lowers, it could release vResources. This process is called scaling. The addition and removal of vResources is considered **scaling Out/In by** **increasing or decreasing the number of VNF instances, not VNFC(s)**. The following test cases address the validation of the VNF scaling Out/In.

### Scale-Out VNF instance

|  |  |
| --- | --- |
| TC.VNF.SCALE.OUT.001 Scale-Out VNF instance | |
| Purpose: | The MANO should have the capability to dynamically scale by adding VNF(s) based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources by increasing number of VNFs. The EM may require to reconfigure the VNFs. Therefore, traffic should be generated to validate that the VNFs can process packets correctly. The trigger for this event to occur could be by different options:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate MANO has resized by adding VNFs, has higher capacity and determine the service disruption 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-out the VNF adds new VNFs.  The scaling trigger could be one of the following:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling   Each of this trigger can be seen a sub test cases. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.OUT.001.1 Scale-Out VNF instance with On-demand Scaling event generated by MANO | |
| Purpose: | The MANO should have the capability to dynamically scale by adding VNF(s) based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources by increasing number of VNFs. The EM may require to reconfigure the VNFs. Therefore, traffic should be generated to validate that the VNFs can process packets correctly. The trigger for this event to occur is the following:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate MANO has resized by adding VNFs, has higher capacity and determine the service disruption 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-out the VNF adds new VNFs. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.OUT.001.2 Scale-Out VNF instance with On-demand Scaling event generated by VNF | |
| Purpose: | The MANO should have the capability to dynamically scale by adding VNF(s) based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources by increasing number of VNFs. The EM may require to reconfigure the VNFs. Therefore, traffic should be generated to validate that the VNFs can process packets correctly. The trigger for this event to occur is the following:   * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate MANO has resized by adding VNFs, has higher capacity and determine the service disruption 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-out the VNF adds new VNFs. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.OUT.001.3 Scale-Out VNF instance with On-demand Scaling event generated by VIM | |
| Purpose: | The MANO should have the capability to dynamically scale by adding VNF(s) based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources by increasing number of VNFs. The EM may require to reconfigure the VNFs. Therefore, traffic should be generated to validate that the VNFs can process packets correctly. The trigger for this event to occur is the following:   * VIM generated event – On-demand Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate MANO has resized by adding VNFs, has higher capacity and determine the service disruption 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-out the VNF adds new VNFs. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.OUT.001.4 Scale-Out VNF instance with On-demand Scaling event generated by EM | |
| Purpose: | The MANO should have the capability to dynamically scale by adding VNF(s) based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources by increasing number of VNFs. The EM may require to reconfigure the VNFs. Therefore, traffic should be generated to validate that the VNFs can process packets correctly. The trigger for this event to occur is the following:   * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate MANO has resized by adding VNFs, has higher capacity and determine the service disruption 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-out the VNF adds new VNFs. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.OUT.001.5 Scale-Out VNF instance with Manual Scaling event generated by MANO | |
| Purpose: | The MANO should have the capability to dynamically scale by adding VNF(s) based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources by increasing number of VNFs. The EM may require to reconfigure the VNFs. Therefore, traffic should be generated to validate that the VNFs can process packets correctly. The trigger for this event to occur is the following:   * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate MANO has resized by adding VNFs, has higher capacity and determine the service disruption 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-out the VNF adds new VNFs. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.OUT.001.6 Scale-Out VNF instance with Manual Scaling event generated by VNF | |
| Purpose: | The MANO should have the capability to dynamically scale by adding VNF(s) based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources by increasing number of VNFs. The EM may require to reconfigure the VNFs. Therefore, traffic should be generated to validate that the VNFs can process packets correctly. The trigger for this event to occur is the following:   * VNF generated event - Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate MANO has resized by adding VNFs, has higher capacity and determine the service disruption 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-out the VNF adds new VNFs. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.OUT.001.7 Scale-Out VNF instance with Manual Scaling event generated by EM | |
| Purpose: | The MANO should have the capability to dynamically scale by adding VNF(s) based on the required load. The scaling to higher performance is the objective of this test. The VNF may has multiple levels of performance. Here we validate only one step, from the basic (lowest level) to the next level up. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources by increasing number of VNFs. The EM may require to reconfigure the VNFs. Therefore, traffic should be generated to validate that the VNFs can process packets correctly. The trigger for this event to occur is the following:   * EM generated event – Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources and generate more traffic to validate the higher capacity or cause the trigger 6. Validate MANO has resized by adding VNFs, has higher capacity and determine the service disruption 7. Stop traffic 8. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-out the MANO adds new VNFs. |

### Max scale-out VNF instance

|  |  |
| --- | --- |
| TC.VNF.SCALE.OUT.002 Max scale-out VNF instance | |
| Purpose: | MANO should have the capability to dynamically scale based on the required load. The scaling to the max higher performance is the objective of this test by adding VNF(s). |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources by adding more VNFs. The EM may require to configure the VNFs. Therefore, traffic should be generated to validate that the VNFs can process packets correctly. The trigger for this event to occur could be one of options:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the maximum 6. Validate MANO has resized the VNF(s) to the max and has max capacity (e.g., generate higher traffic load) and added new VNFs 7. Determine the service disruption during the resizing to the max and the time it takes to complete it 8. Stop traffic 9. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-out the MANO adds new VNFs.  The scaling trigger could be one of the following:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |

### Scale-in VNF instance

|  |  |
| --- | --- |
| TC.VNF.SCALE.OUT.003 Scale-in VNF instance | |
| Purpose: | A MANO should have the capability to dynamically scale by adding VNF(s) based on the required load. The scaling-in to the lower performance is the objective of this test. |
| Description | Once the VNF scaling decrease is completed, the MANO should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the MANO needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur could be by different options:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the next level 6. Validate VNF has resized to the next level 7. Determine if and length of service disruption 8. Generate traffic load the next level 9. Validate capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate MANO has released the resources and decreased the VNFs 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-in the MANO removes VNFs.  Each of this are sub test cases.  The scaling trigger could be one of the following:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling   Each of this trigger can be seen a sub test cases. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.OUT.003.1 Scale-in VNF instance with On-demand Scaling event generated by MANO | |
| Purpose: | A MANO should have the capability to dynamically scale by adding or removing VNF(s) based on the required load. The scaling-in to the lower performance is the objective of this test. |
| Description | Once the scaling decrease is completed, the MANO should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the MANO needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur is the following:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the next level 6. Validate VNF has resized to the next level 7. Determine if and length of service disruption 8. Generate traffic load the next level 9. Validate capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate MANO has released the resources and decreased the VNFs 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-in the VNF removes VNFCs. |

|  |  |
| --- | --- |
| TC.VNF.SCALE.OUT.003.2 Scale-in VNF instance with On-demand Scaling event generated by VNF | |
| Purpose: | A MANO should have the capability to dynamically scale by adding or removing VNF(s) based on the required load. The scaling-in to the lower performance is the objective of this test. |
| Description | Once the scaling decrease is completed, the MANO should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the MANO needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur is the following:   * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the next level 6. Validate VNF has resized to the next level 7. Determine if and length of service disruption 8. Generate traffic load the next level 9. Validate capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate MANO has released the resources and decreased the VNFs 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF(s) should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-in the MANO removes VNF(s). |

|  |  |
| --- | --- |
| TC.VNF.SCALE.OUT.003.3 Scale-in VNF instance with On-demand Scaling event generated by VIM | |
| Purpose: | A MANO should have the capability to dynamically scale by adding or removing VNF(s) based on the required load. The scaling-in to the lower performance is the objective of this test. |
| Description | Once the scaling decrease is completed, the MANO should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the MANO needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur is the following:   * VIM generated event – On-demand Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the next level 6. Validate VNF has resized to the next level 7. Determine if and length of service disruption 8. Generate traffic load the next level 9. Validate capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate VNF has released the resources and decreased the VNF(s) 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF(s) should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-in the MANO removes VNF(s). |

|  |  |
| --- | --- |
| TC.VNF.SCALE.OUT.003.4 Scale-in VNF instance with On-demand Scaling event generated by EM | |
| Purpose: | A MANO should have the capability to dynamically scale by adding or removing VNF(s) based on the required load. The scaling-in to the lower performance is the objective of this test. |
| Description | Once the scaling decrease is completed, the MANO should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the MANO needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur is the following:   * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the next level 6. Validate VNF has resized to the next level 7. Determine if and length of service disruption 8. Generate traffic load the next level 9. Validate capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate MANO has released the resources and decreased the VNF(s) 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-in the MANO removes VNF(s). |

|  |  |
| --- | --- |
| TC.VNF.SCALE.OUT.003.5 Scale-in VNF instance with Manual Scaling event generated by MANO | |
| Purpose: | A MANO should have the capability to dynamically scale by adding or removing VNF(s) based on the required load. The scaling-in to the lower performance is the objective of this test. |
| Description | Once the scaling decrease is completed, the MANO should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the MANO needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur is the following:   * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the next level 6. Validate VNF has resized to the next level 7. Determine if and length of service disruption 8. Generate traffic load the next level 9. Validate capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate MANO has released the resources and decreased the VNF(s) 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-in the MANO removes VNF(s). |

|  |  |
| --- | --- |
| TC.VNF.SCALE.OUT.003.6 Scale-in VNF instance with Manual Scaling event generated by VNF | |
| Purpose: | A MANO should have the capability to dynamically scale by adding or removing VNF(s) based on the required load. The scaling-in to the lower performance is the objective of this test. |
| Description | Once the scaling decrease is completed, the MANO should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the MANO needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur is the following:   * VNF generated event - Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the next level 6. Validate VNF has resized to the next level 7. Determine if and length of service disruption 8. Generate traffic load the next level 9. Validate capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate MANO has released the resources and decreased the VNF(s) 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-in the MANO removes VNF(s). |

|  |  |
| --- | --- |
| TC.VNF.SCALE.OUT.003.7 Scale-in VNF instance with Manual Scaling event generated by EM | |
| Purpose: | A MANO should have the capability to dynamically scale by adding or removing VNF(s) based on the required load. The scaling-in to the lower performance is the objective of this test. |
| Description | Once the scaling decrease is completed, the MANO should report reduced virtualized resources. The service should not be impacted. Therefore, traffic load should be processed without drops if the load is not above the limit. If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the MANO needs to have the capability to remove these resources when decreasing the scaling. The trigger for this event to occur is the following:   * EM generated event – Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the next level 6. Validate VNF has resized to the next level 7. Determine if and length of service disruption 8. Generate traffic load the next level 9. Validate capacity without traffic loss 10. Trigger the downsize of the VNF 11. Validate MANO has released the resources and decreased the VNF(s) 12. Validate traffic drop occurs 13. Reduce traffic load to level that the downsized VNF should process 14. Validate traffic flows through without issues (🡪 no dropped packets) 15. Stop traffic 16. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled down:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-in the MANO removes VNF(s). |

### Max vResource limit reached before max VNFD limit for VNF scale-out

|  |  |
| --- | --- |
| TC.VNF.SCALE.OUT.004 Max vResource limit reached before max VNFD limit for VNF scale-out | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling to the max higher performance is the objective of this test. But the vResource limit is reached earlier than the max VNFD limit. vMemory, vStorage, vNIC and vCPU are just examples how the VNF could be limited by. The MANO must handle this corner case. |
| Description | Once the scaling increase is completed, the MANO should report the additional allocated virtualized resources. The MANO should have added more resources and VNF(s). The EM may require to reconfigure the VNF(s). Therefore, traffic should be generated to validate that the VNF(s) can process packets correctly. The trigger for this event to occur could be one of the options:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Ensure NFVI has less vResources as the max VNFD indicates 2. Instantiate VNF 3. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 4. Generate low traffic load 5. Validate that traffic flows through without issues (--> no dropped packets) 6. Trigger a resize of the NFV resources to the maximum (limited by NFVI) 7. Validate MANO has resized the VNF to the max limit and added new VNF(s) 8. Generate higher traffic load 9. Validate that the load 10. Determine the service disruption during the resizing to the max and the time it takes to complete it 11. Stop traffic 12. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could have reached its limit:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   The scaling trigger could be one of the following:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |

### Removal of virtualized specialized hardware acceleration for VNF scale-in

|  |  |
| --- | --- |
| TC.VNF.SCALE.OUT.005 Removal of virtualized specialized hardware acceleration for VNF scale-in | |
| Purpose: | A VNF should have the capability to dynamically scale based on the required load. The scaling to the lower performance is the objective of this test by removing vHardware resources. The MANO must handle this case of vNIC resource management. |
| Description | If a VNF functionality depends on specialized hardware acceleration devices/NIC cards, the MANO needs to have the capability to remove these resources when decreasing the scaling. The MANO should report reduced resources after the completion. The EM may apply new configurations. The functionality of the VNF should be validated. Lower scalability of the VNF service should be the result. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to use more specialized hardware resources 6. Validate VNF has resized 7. Generate higher traffic load 8. Validate increased its capacity by no traffic loss 9. Validate that MANO has allocated more specialized hardware resources and added new VNF(s) 10. Generate low traffic load 11. Trigger a resize of the NFV resources to use less specialized hardware resources 12. Validate VNF has resized and has decreased its capacity and removed VNF(s) 13. Validate that MANO has allocated less specialized hardware resources and the previous specialized hardware resources have been freed up 14. Determine the service disruption during the resizing 15. Validate no dropped packets 16. Stop traffic 17. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be removed:   * vNIC * Other specialized hardware   The scaling trigger could be one of the following:   * MANO generated events - Auto Scaling (e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling (e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling |

## vResource Migration

There are conditions where VNFs need to me migrated from one server to another as they are in service. There are many reasons for this event. Here are some examples:

1. Servers require hardware maintenance
2. Hardware replacement
3. vNIC replacements or additions
4. Upgrade to newer NFVI software version

Overall the VNF gets moved from one entity to another and at the end the service should be provided without any issues. Pending on the VNF capabilities the service may get disrupted.

|  |  |
| --- | --- |
| TC.VNF.MIGRATE.001 Migration of virtual resource with live migration set without traffic load | |
| Purpose: | Once the VNFM is instructed to migrate resources, it should result in a VNF that can service the functionality. Therefore, service availability should be validated before and after the migration. The duration should be measured for the migration process. Following vResources could be migrated: network, memory, compute, storage, hardware acceleration devices/NIC cards. |
| Description | The VNFD indicates that the VNF supports live migration. Therefore, once instructed, the VNFM should build a new VNF to migrate an existing VNF from one set of vResources to another new set of vResources with minimal service disruption |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Verify VNF supports live migration 4. Generate low traffic load 5. Validate that traffic flows through without issues (--> no dropped packets) 6. Stop Traffic 7. Trigger a migration of the vResource by issuing a command to the VNFM (--> time stamp) 8. Validate VNF has migrated, the new vResource is been used and the older vResource is released (--> time stamp) 9. Generate low traffic load 10. Validate that traffic flows through without issues (--> no dropped packets) 11. Stop Traffic 12. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be migrated:   1. vMemory 2. vCPU cores 3. vStorage 4. vNIC or specialized hardware   Each of this are sub test cases. |

|  |  |
| --- | --- |
| TC.VNF.MIGRATE.002 Migration of virtual resource with live migration set under traffic load | |
| Purpose: | Once the VNFM is instructed to migrate resources, it should result in a VNF that can service the functionality. Therefore, service availability should be validated before and after the migration. The duration should be measured for the migration process. Following vResources could be migrated: network, memory, compute, storage, hardware acceleration devices/NIC cards. During the migration the service may get disrupted. The service disruption should be measured. |
| Description | The VNFD indicates that the VNF supports live migration. Therefore, once instructed, the VNFM should build a new VNF to migrate an existing VNF from one set of vResources to another new set of vResources with minimal service disruption |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Verify VNF supports live migration 4. Generate low traffic load 5. Validate that traffic flows through without issues (--> no dropped packets) 6. Trigger a migration of the vResource by issuing a command to the VNFM (--> time stamp) 7. Validate VNF has migrated, the new vResource is been used and the older vResource is released (--> time stamp) 8. Validate that traffic flows through without issues (--> no dropped packets) 9. Determine the service disruption by packet drop calculation 10. Stop Traffic 11. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be migrated:   1. vMemory 2. vCPU cores 3. vStorage 4. vNIC or specialized hardware   Each of this are sub test cases. |

|  |  |
| --- | --- |
| TC.VNF.MIGRATE.003 Migration of virtual resource with offline migration set without traffic load | |
| Purpose: | Once the VNFM is instructed to migrate resources, it should result in a VNF that can service the functionality. Therefore, service availability should be validated before and after the migration. The duration should be measured for the migration process. Following vResources could be migrated: network, memory, compute, storage, hardware acceleration devices/NIC cards. For offline migration set the VNFM deactivates the VNF for the migration process. A service disruption will therefore occur. |
| Description | The VNFD indicates that the VNF support offline migration. Therefore, once instructed, the VNFM should rebuild a new VNF to migrate an existing VNF from one set of vResources to another new set of vResources by taking the VNF offline and the new VNF online at a later time. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Verify VNF supports offline migration 4. Generate low traffic load 5. Validate that traffic flows through without issues (--> no dropped packets) 6. Stop Traffic 7. Trigger a migration of the vResource by issuing a command to the VNFM (--> time stamp) 8. Validate VNF has migrated, the new vResource is been used and the older vResource is released (--> time stamp) 9. Generate low traffic load 10. Validate that traffic flows through without issues (--> no dropped packets) 11. Stop Traffic 12. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be migrated:   1. vMemory 2. vCPU cores 3. vStorage 4. vNIC or specialized hardware   Each of this are sub test cases. |

|  |  |
| --- | --- |
| TC.VNF.MIGRATE.004 Migration of virtual resource with offline migration set under traffic load | |
| Purpose: | Once the VNFM is instructed to migrate resources, it should result in a VNF that can service the functionality. Therefore, service availability should be validated before and after the migration. The duration should be measured for the migration process. Following vResources could be migrated: network, memory, compute, storage, hardware acceleration devices/NIC cards. For offline migration set the VNFM deactivates the VNF for the migration process. A service disruption will therefore occur. The service disruption must be measured. |
| Description | The VNFD indicates that the VNF support offline migration. Therefore, once instructed, the VNFM should rebuild a new VNF to migrate an existing VNF from one set of vResources to another new set of vResources by taking the VNF offline and the new VNF online. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Verify VNF supports offline migration 4. Generate low traffic load 5. Validate that traffic flows through without issues (--> no dropped packets) 6. Trigger a migration of the vResource by issuing a command to the VNFM (--> time stamp) 7. Validate VNF has migrated, the new vResource is been used and the older vResource is released (--> time stamp) 8. Validate that traffic flows through without issues (--> no dropped packets) 9. Stop Traffic 10. Determine the service disruption by packet drop calculation 11. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be migrated:   1. vMemory 2. vCPU cores 3. vStorage 4. vNIC or specialized hardware   Each of this are sub test cases. |

## vResource Network

Virtualized networks encompass different types of network:

- local

- vlan

- vxlan

-gre

- l3-vpn

The VNFD describes the network demands of the VNF.

A network allows to define the bandwidth that can flow through it. In this way it is possible to either protect a VNF for extreme traffic flows or restrict it because of service contract.

An IP interface could be IPv4, IPv6 or dual stack (IPv4 and IPv6). The IP address could be configured as static IP (with netmask and gateway) or DHCP. An IP address could be a private or public IP address allowing functionalities like NAT. Multicast IP address and anycast IP address and their usage are other types of traffic and may require vNetwork and VNF validation.

An vNetwork has an operation state: enabled or disabled.

An vNetwork has a sharing criteria of not-shared and shared (could include a list of shared consumers).

An vNetwork exposes the feature to define the MAC address of the interface or run with an unspecified MAC address. If the VNF depends on the MAC address, test cases could be added to address this.

An vNetwork specifies the type of vNIC: normal vNIC, direct PCIE pass-through, … . Test cases could be added to validate that the VNF can work the different types of network access types.

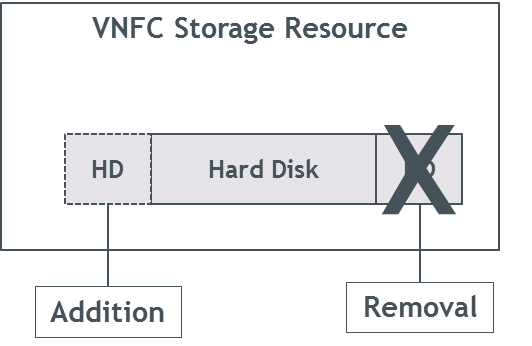
An vNetwork specifies any acceleration capabilities, like RDMA, packet dispatch, TCP Chimney. Here the VNF may take advantage of them and additional test cases should be added.

|  |  |
| --- | --- |
| TC.VNF.VNETWORK.001 vNetworks resources | |
| Purpose: | After instantiation of a VNF, the VNFM should deliver the correct allocated virtualized networks as given in the VNFD. There are different types of network available:   * local * vlan * vxlan * gre * l3-vpn   Another important parameter is the bandwidth. A network interface can be configured as static IP (provided) with netmask and gateway or DHCP including the IP version, IPv4 and IPv6, private and public IP address, multicast IP address and anycast IP address.  An vNetwork has an operation state: enabled or disabled.  An vNetwork has a sharing criteria of not-shared and shared (could include a list of shared consumers).  An vNetwork exposes the feature to define the MAC address of the interface or run with an unspecified MAC address.  An vNetwork specifies the type of vNIC: normal vNIC, direct PCIE pass-through, … .  An vNetwork specifies any acceleration capabilities, like RDMA, packet dispatch, TCP Chimney.  A VNF shall have at least one SWA-1 interface and one SWA-5 interface. |
| Description | The VNFD can describe a complex vNetwork used by the VNF. The test case is to validate all vNetwork related resources.. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Stop Traffic 6. Validate the vNetwork resources and the correctness of instantiation 7. Terminate VNF 8. Validate all vNetwork resources has been released |
| Extensions: |  |
| Notes: | The test case could be divided into further subcategories validating  a. Types of network:  local  vlan  vxlan  gre  l3-vpn  b. IP address assignment  static IP (netmask and gateway) or DHCP  private and public IP address  c. IP protocol  IPv4  IPv6  d. vNetwork sharing  not-shared and shared  e. MAC address  defined MAC address or unspecified MAC address  f. Number of network interfaces  VNF could be capable of handling a dynamic number of network interfaces |

|  |  |
| --- | --- |
| TC.VNF.VNETWORK.002 VNFM is able to modify the bandwidth of the vNetwork | |
| Purpose: | After instantiation of a VNF, the VNFM should deliver the correct allocated virtualized networks as given in the VNFD. There are different types of network available:   * local * vlan * vxlan * gre * l3-vpn   Via the VNFM interface the vNetwork bandwidth should be reduced or increased. By emitting traffic before and after and exceeding the new bandwidth traffic must be dropped. We could measure the time it takes to process the command. |
| Description | The VNFD can describe a complex vNetwork used by the VNF. The test case is to validate vNetwork bandwidth control. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate normal traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Issue VNFM command to lower the bandwidth of the vNetwork 6. Validate that traffic gets dropped as we reach over the limit (--> dropped packets) 7. Issue VNFM command to increase the bandwidth of the vNetwork 8. Validate no traffic gets dropped as traffic load is under the limit (--> no dropped packets) 9. Stop traffic 10. Terminate VNF |
| Extensions: |  |
| Notes: | The test case could be divided into further subcategories validating  a. Types of network:  local  vlan  vxlan  gre  l3-vpn  b. IP address assignment  static IP (netmask and gateway) or DHCP  private and public IP address  c. IP protocol  IPv4  IPv6  d. vNetwork sharing  not-shared and shared  e. MAC address  defined MAC address or unspecified MAC address  f. Number of network interfaces  VNF could be capable of handling a dynamic number of network interfaces |

## vResource Storage

The virtualized Storage resource can be increased and decreased in size. The following test cases validate the functionality of the VNF to scope with this changes.



|  |  |
| --- | --- |
| TC.VNF.VSTORAGE.001 Increase of vStorage | |
| Purpose: | If the VNF functionality is pure storage bound, the VNFM increases the scalability by adding storage resources. At the end of this process, the VNFM should report more virtualized storage resources.  There are different types of storages:   * volume * object   Following source could be used as a trigger to initiate the expansion of storage:   * VNFM/MANO generated events - Auto Scaling ( e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling ( e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling   Because the EM may apply new configurations, at the end of the process the functionality of the VNF should be validated. |
| Description | A VNF may rely on vStorage. The test case is to validate the capability to increase the vStorage for a VNF. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate normal traffic load 4. Validate that traffic flows through without issues (🡪 no dropped packets) 5. Issue command to VIM, EM or VNFM to cause an increase request for storage 6. Validate VNFM added successfully more storage to the VNF 7. Validate that traffic flows without drop of packets (🡪 no dropped packets) 8. Stop Traffic 9. Terminate VNF |
| Extensions: |  |
| Notes: | The test case has multiple sub cases where the vStorage increase request could be generated by   * VIM * EM * VNFM * VNF * Increased traffic load |

|  |  |
| --- | --- |
| TC.VNF.VSTORAGE.002 Decrease of vStorage | |
| Purpose: | If the VNF functionality is pure storage bound, the VNFM increases the scalability by adding storage resources. At the end of this process, the VNFM should report more virtualized storage resources.  There are different types of storages:   * volume * object   Following source could be used as a trigger to initiate the expansion of storage:   * VNFM/MANO generated events - Auto Scaling ( e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling ( e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling   Because the EM may apply new configurations, at the end of the process the functionality of the VNF should be validated. |
| Description | A VNF may rely on vStorage. The test case is to validate the capability to decrease the vStorage for a VNF. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate normal traffic load 4. Validate that traffic flows through without issues (🡪 no dropped packets) 5. Issue command to VIM, EM, VNFM to cause a decrease request for storage 6. Validate VNFM removed successfully less storage to the VNF 7. Validate that traffic flows without drop of packets (🡪 no dropped packets) 8. Stop Traffic 9. Terminate VNF |
| Extensions: |  |
| Notes: | The test case has multiple sub cases where the vStorage decrease request could be generated by   * VIM * EM * VNFM * VNF * Reduced traffic load |

# VNF Fault and Recovery Management

The NFV Infrastructure may detect faults inside the hardware. A VNFM can register for notifications of this type at the VIM. Once the VIM detects hardware faults or clearing of faults, it will notify the registered party of the change. Here are examples of this type:

* + Virtualized hardware errors of CPU or memory
  + Virtualized container crashes
  + Virtual network ports errors
  + Virtual container’s to storage disconnections

Once the fault is detected and propagated to the VNFM, the VNFM can react on it and recover the VNF from the fault.

## VNF Fault Management

The VNF Fault Management testing encompasses the validation that the VNFM can register, deregister and process fault notifications for the types of

* vCPU
* vMemory
* vStorage
* vNetwork

|  |  |
| --- | --- |
| TC.VNF.FAULT.001 vCPU resource fault notification subscription and notification | |
| Purpose: | Verify the VNFM can register and deregister for vCPU faults and process the notification correctly |
| Description | VNFM should register successfully register and deregister for virtualized resource fault notification of vCPU and can process a notification of this type correctly. The VIM should indicate that the VNFM is registered for the notification.  Initiate the virtualized resource fault notification through the VIM. The VNFM should successfully indicate the impact and may take actions when configured to do so. The state of the alarm could be:   * Fired * updated * cleared   Following severity is available:   * Critical * Major * Minor * Warning * Indeterminate * Cleared   vCompute could have a fault of CPU failure. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Issue command to the VNFM to register for vCPU fault notification 4. Validate via VNFM and VIM interface that registration was successful 5. Issue a command to VIM that causes a vCPU fault for the VNFM 6. Validate that VNFM processes the vCPU fault notification 7. Issue a command to VIM that clears the vCPU fault for the VNFM 8. Validate that VNFM processes the vCPU fault removal notification 9. Issue command to the VNFM to deregister for vCPU fault notification 10. Validate via VNFM and VIM interface that deregistration was successful 11. Terminate VNF |
| Extensions: |  |
| Notes: | The test case could have multiple subcases addressing the different severities:   * Critical * Major * Minor * Warning * Indeterminate   The state of the alarm could be another set of sub test cases:   * Fired * Updated * Cleared |

|  |  |
| --- | --- |
| TC.VNF.FAULT.002 Double vCPU resource fault notification subscription and notification | |
| Purpose: | Verify the VNFM can register and deregister for vCPU faults and process two notifications correctly |
| Description | VNFM should register successfully register and deregister for virtualized resource fault notification of vCPU and can process two notification of this type correctly. The VIM should indicate that the VNFM is registered for the notification. The VNF could be implemented with multiple VNFCs and therefore generate multiple notifications.  Initiate the virtualized resource fault notification through the VIM. The VNFM should successfully indicate the impact and may take actions when configured to do so. The state of the alarm could be:   * Fired * updated * cleared   Following severity is available:   * Critical * Major * Minor * Warning * Indeterminate * Cleared   vCompute could have a fault of CPU failure. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Issue command to the VNFM to register for vCPU fault notification 4. Validate via VNFM and VIM interface that registration was successful 5. Issue a command to VIM that causes a vCPU fault for the VNFM 6. Validate that VNFM processes the vCPU fault notification 7. Issue a command to VIM that causes another vCPU fault for the VNFM 8. Validate that VNFM processes the vCPU fault update notification 9. Issue a command to VIM that clears the vCPU fault for the VNFM 10. Validate that VNFM processes the vCPU fault removal notification 11. Issue command to the VNFM to deregister for vCPU fault notification 12. Validate via VNFM and VIM interface that deregistration was successful 13. Terminate VNF |
| Extensions: |  |
| Notes: | The test case could have multiple subcases addressing the different severities:   * Critical * Major * Minor * Warning * Indeterminate   The state of the alarm could be another set of sub test cases:   * Fired * Updated * Cleared |

|  |  |
| --- | --- |
| TC.VNF.FAULT.003 vMemory resource fault notification subscription and notification | |
| Purpose: | Verify the VNFM can register and deregister for vMemory faults and process the notification correctly |
| Description | VNFM should register successfully register and deregister for virtualized resource fault notification of vMemory and can process a notification of this type correctly. The VIM should indicate that the VNFM is registered for the notification.  Initiate the virtualized resource fault notification through the VIM. The VNFM should successfully indicate the impact and may take actions when configured to do so. The state of the alarm could be:   * Fired * Updated * Cleared   Following severity is available:   * Critical * Major * Minor * Warning * Indeterminate * Cleared   vMemory could have a fault of memory failure. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Issue command to the VNFM to register for vMemory fault notification 4. Validate via VNFM and VIM interface that registration was successful 5. Issue a command to VIM that causes a vMemory fault for the VNFM 6. Validate that VNFM processes the vMemory fault notification 7. Issue a command to VIM that clears the vMemory fault 8. Validate that VNFM processes the vMemory fault clear notification 9. Issue command to the VNFM to deregister for vMemory fault notification 10. Validate via VNFM and VIM interface that deregistration was successful 11. Terminate VNF |
| Extensions: |  |
| Notes: | The test case could have multiple subcases addressing the different severities:   * Critical * Major * Minor * Warning * Indeterminate   The state of the alarm could be another set of sub test cases:   * Fired * Updated * Cleared |

|  |  |
| --- | --- |
| TC.VNF.FAULT.004 vNetwork resource fault notification subscription and notification | |
| Purpose: | Verify the VNFM can register and deregister for vNetwork faults and process the notification correctly |
| Description | VNFM should register successfully register and deregister for virtualized resource fault notification of vMemory and can process a notification of this type correctly. The VIM should indicate that the VNFM is registered for the notification.  Initiate the virtualized resource fault notification through the VIM. The VNFM should successfully indicate the impact and may take actions when configured to do so. The state of the alarm could be:   * Fired * Updated * Cleared   Following severity is available:   * Critical * Major * Minor * Warning * Indeterminate * Cleared   vNetwork could have a fault of memory failure. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Issue command to the VNFM to register for vNetwork fault notification 4. Validate via VNFM and VIM interface that registration was successful 5. Issue a command to VIM that causes a vNetwork fault for the VNFM 6. Validate that VNFM processes the vNetwork fault notification 7. Issue command to the VNFM to deregister for vNetwork fault notification 8. Validate via VNFM and VIM interface that deregistration was successful 9. Terminate VNF |
| Extensions: |  |
| Notes: | The test case could have multiple subcases addressing the different severities:   * Critical * Major * Minor * Warning * Indeterminate   The state of the alarm could be another set of sub test cases:   * Fired * Updated * Cleared |

|  |  |
| --- | --- |
| TC.VNF.FAULT.005 vStorage resource fault notification subscription and notification | |
| Purpose: | Verify the VNFM can register and deregister for vStorage faults and process the notification correctly |
| Description | VNFM should register successfully register for virtualized resource fault notification of vStorage and can process a notification of this type correctly. The VIM should indicate that the VNFM is registered for the notification.  Initiate the virtualized resource fault notification through the VIM. The VNFM should successfully indicate the impact and may take actions when configured to do so. The state of the alarm could be:   * Fired * updated * cleared   Following severity is available:   * Critical * Major * Minor * Warning * Indeterminate * Cleared   vStorage could have a fault of hardware access failure. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Issue command to the VNFM to register for vStorage fault notification 4. Validate via VNFM and VIM interface that registration was successful 5. Issue a command to VIM that causes a vStorage fault for the VNFM 6. Validate that VNFM processes the vStorage fault notification 7. Issue a command to VIM that clears vStorage fault for the VNFM 8. Validate that VNFM processes the vStorage fault clear notification 9. Issue command to the VNFM to deregister for vStorage fault notification 10. Validate via VNFM and VIM interface that deregistration was successful 11. Terminate VNF |
| Extensions: |  |
| Notes: | The test case could have multiple subcases addressing the different severities:   * Critical * Major * Minor * Warning * Indeterminate   The state of the alarm could be another set of sub test cases:   * Fired * Updated * Cleared |

## VNF Recovery

Once a VNF is a fault state, the VNFM should recover the VNF from this state and bring it back to normal operation.

|  |  |
| --- | --- |
| TC.VNF.RECOVERY.001 Manual VNF recover of a faulty VNF | |
| Purpose: | Validate VNFM recovers a faulty VNF |
| Description | The VNFM must be able to recover a faulty VNF. The recovery instruction could come from a command issued to the VNFM, from the NFVO or EM. Traffic should be send before the instruction gets issued. We should measure how long it takes for the VNFM to recover the VNF. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate normal traffic load 4. Validate that traffic flows through without issues (🡪 no dropped packets) 5. Issue command to inject a fault 6. Validate that traffic gets dropped as VNF is now in fault condition (🡪 dropped packets) 7. Manually issue VNFM command healing trigger (🡪 time stamp) 8. Validate fault has been removed (🡪 time stamp) 9. Validate traffic flows through without issues (🡪 no dropped packets, 🡪 time stamp) 10. Stop Traffic 11. Terminate VNF |
| Extensions: |  |
| Notes: | Sub test cases could be created based on the source of the recovery instruction   * Command to VNFM directly * Command from NFVO * Command from EM   Another set of sub test cases is the type of fault injection. Here are samples:   * VIM command to stop the VNF * VIM command to stop one of the compute notes of the VNF * VIM command to cause vStorage fault * VIM command to cause vMemory fault * VIM command to cause vNetwork fault * VIM command to cause EM communication fault * VIM command to cause VNF and VNFM communication fault |

# Performance Management

Performance management is concentrated on measurements collection and their notification. It exposes the access to collect statistical information of resource consumption levels like

* vCPU power consumption
* vCPU load
* VM memory usage oversubscription
* VM disk latency

Following operations are available:

* Create and Delete Performance Management Job operation
* Query Performance Management Job operation
* Subscribe notification for performance information availability and threshold crossed notification
* Create and Delete Threshold operation
* Query existing Threshold operations

|  |  |
| --- | --- |
| TC.VNF.PERF.001 vCPU resource performance data notification subscription and notification | |
| Purpose: | Validate VNFM can subscribe to and process notifications of the type vCPU resource performance data |
| Description | VNFM creates a PM job for a virtualized resource of type vCPU. Via periodic update notification the VNFM gets the data. At the end the PM job gets canceled and the VIM should be clean of jobs.  The PM job should have following inputs:   * Resource selector is CPU * performance metric type * collection period * reporting period * reporting boundary to define the duration   The VIM should report the PM job with the same parameters.  The notification can report the availability of performance information and/or threshold crossed notifications. This test is for performance notification. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate traffic 4. Validate traffic flows through 5. Issue command to the VNFM to collect vCPU performance data 6. Validate via VNFM and VIM interface that collection command was successful 7. Issue command to the VNFM to register for vCPU performance data notification 8. Validate via VNFM and VIM interface that registration was successful 9. Validate that VNFM gets the notification and updates the data 10. Generate either higher traffic 11. Validate traffic flows through 12. Validate that VNFM gets the notification and updates the data with higher values 13. Generate either lower traffic 14. Validate traffic flows through 15. Validate that VNFM gets the notification and updates the data with lower values 16. Issue command to the VNFM to deregister for vCPU performance notification 17. Validate via VNFM and VIM interface that deregistration was successful 18. Issue command to the VNFM to remove vCPU performance collection 19. Validate via VNFM and VIM interface that removal was successful 20. Terminate VNF |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.PERF.002 vMemory resource performance data notification subscription and notification | |
| Purpose: | Validate VNFM can subscribe to and process notifications of the type vMemory resource performance data |
| Description | VNFM creates a PM job for a virtualized resource of type vMemory. Via periodic update notification the VNFM gets the data. At the end the PM job gets canceled and the VIM should be clean of jobs.  The PM job should have following inputs:   * Resource selector is vMemory * performance metric type * collection period * reporting period * reporting boundary to define the duration   The VIM should report the PM job with the same parameters.  The notification can report the availability of performance information and/or threshold crossed notifications. This test is for performance data notification. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate traffic 4. Validate traffic flows through 5. Issue command to the VNFM to collect vMemory performance data 6. Validate via VNFM and VIM interface that collection command was successful 7. Issue command to the VNFM to register for vMemory performance data notification 8. Validate via VNFM and VIM interface that registration was successful 9. Validate that VNFM gets the notification and updates the data 10. Generate either higher traffic 11. Validate traffic flows through 12. Validate that VNFM gets the notification and updates the data with higher values 13. Generate either lower traffic 14. Validate traffic flows through 15. Validate that VNFM gets the notification and updates the data with lower values 16. Issue command to the VNFM to deregister for vMemory performance notification 17. Validate via VNFM and VIM interface that deregistration was successful 18. Issue command to the VNFM to remove vMemory performance collection 19. Validate via VNFM and VIM interface that removal was successful 20. Terminate VNF |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.PERF.003 vStorage resource performance data notification subscription and notification | |
| Purpose: | Validate VNFM can subscribe to and process notifications of the type vStorage resource performance data |
| Description | VNFM creates a PM job for a virtualized resource of type vStorage. Via periodic update notification the VNFM gets the data. At the end the PM job gets canceled and the VIM should be clean of jobs.  The PM job should have following inputs:   * Resource selector is vStorage * performance metric type * collection period * reporting period * reporting boundary to define the duration   The VIM should report the PM job with the same parameters.  The notification can report the availability of performance information and/or threshold crossed notifications. This test is for performance data notification. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate traffic 4. Validate traffic flows through 5. Issue command to the VNFM to collect vStorage performance data 6. Validate via VNFM and VIM interface that collection command was successful 7. Issue command to the VNFM to register for vStorage performance data notification 8. Validate via VNFM and VIM interface that registration was successful 9. Validate that VNFM gets the notification and updates the data 10. Generate either higher traffic 11. Validate traffic flows through 12. Validate that VNFM gets the notification and updates the data with higher values 13. Generate either lower traffic 14. Validate traffic flows through 15. Validate that VNFM gets the notification and updates the data with lower values 16. Issue command to the VNFM to deregister for vStorage performance notification 17. Validate via VNFM and VIM interface that deregistration was successful 18. Issue command to the VNFM to remove vStorage performance collection 19. Validate via VNFM and VIM interface that removal was successful 20. Terminate VNF |
| Extensions: |  |
| Notes: |  |

|  |  |
| --- | --- |
| TC.VNF.PERF.004 vCPU resource performance threshold notification subscription and notification | |
| Purpose: | Validate VNFM can subscribe to and process resource performance threshold notifications of the type vCPU resource |
| Description | VNFM creates a PM job for a virtualized resource of type vCPU for data collection and a threshold notification. In order a threshold to be active, there must be a PM job collecting the needed metric for the selected entries.  By increasing the network load, the vCPU threshold could be triggered. VNFM should process the notification and indicate the event.  The PM job should have following inputs:   * Resource selector is vCPU * performance metric type * collection period * reporting period * reporting boundary to define the duration   The VIM should report the PM job with the same parameters.  The notification can report the availability of performance information and/or threshold crossed notifications. This test is for performance threshold notification. For the threshold crossed notification to occur, the emission must stay on to generate cpu load over threshold and longer than excessive time.  Invocation method could be   * vCPU instance with threshold level of 1% * vNetwork bandwidth with threshold level of 1% * vMemory usage with threshold level of 80% * vStorage usage with threshold level of 70% |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate traffic 4. Validate traffic flows through 5. Issue command to the VNFM to register for vCPU performance data collection 6. Validate via VNFM and VIM interface that registration was successful 7. Issue command to the VNFM to register for vCPU performance threshold notification 8. Validate via VNFM and VIM interface that registration was successful 9. Increase the load to cause the threshold notification event 10. Validate that VNFM gets the notification when the threshold is reached 11. Stop traffic 12. Issue command to the VNFM to deregister for vCPU performance threshold notification 13. Validate via VNFM and VIM interface that deregistration was successful 14. Issue command to the VNFM to deregister for vCPU performance data collection 15. Validate via VNFM and VIM interface that deregistration was successful 16. Terminate VNF |
| Extensions: |  |
| Notes: | There are many sub cases where the threshold could be set on different statistic values and levels. But the focus is here not on the VIM functionality. We concentrate on the VNFM. So it is more important that VNFM take the different command parameter and apply them correctly to the VIM. |

|  |  |
| --- | --- |
| TC.VNF.PERF.005 vMemory resource performance threshold notification subscription and notification | |
| Purpose: | Validate VNFM can subscribe to and process resource performance threshold notifications of the type vMemory resource |
| Description | VNFM creates a PM job for a virtualized resource of type vMemory for data collection and a threshold notification. In order a threshold to be active, there must be a PM job collecting the needed metric for the selected entries.  By increasing the network load, the vMemory threshold could be triggered. VNFM should process the notification and indicate the event.  The PM job should have following inputs:   * Resource selector is vMemory * performance metric type * collection period * reporting period * reporting boundary to define the duration   The VIM should report the PM job with the same parameters.  The notification can report the availability of performance information and/or threshold crossed notifications. This test is for performance threshold notification. For the threshold crossed notification to occur, the emission must stay on to generate cpu load over threshold and longer than excessive time.  Invocation method could be   * vCPU instance with threshold level of 1% * vNetwork bandwidth with threshold level of 1% * vMemory usage with threshold level of 80% * vStorage usage with threshold level of 70% |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate traffic 4. Validate traffic flows through 5. Issue command to the VNFM to register for vMemory performance data collection 6. Validate via VNFM and VIM interface that registration was successful 7. Issue command to the VNFM to register for vMemory performance threshold notification 8. Validate via VNFM and VIM interface that registration was successful 9. Increase the load to cause the event 10. Validate that VNFM gets the notification when the threshold is reached 11. Issue command to the VNFM to deregister for vMemory performance threshold notification 12. Validate via VNFM and VIM interface that deregistration was successful 13. Issue command to the VNFM to deregister for vMemory performance data collection 14. Validate via VNFM and VIM interface that deregistration was successful 15. Terminate VNF |
| Extensions: |  |
| Notes: | There are many sub cases where the threshold could be set on different statistic values and levels. But the focus is here not on the VIM functionality. We concentrate on the VNFM. So it is more important that VNFM take the different command parameter and apply them correctly to the VIM. |

|  |  |
| --- | --- |
| TC.VNF.PERF.006 vStorage resource performance threshold notification subscription and notification | |
| Purpose: | Validate VNFM can subscribe to and process resource performance threshold notifications of the type vStorage resource |
| Description | VNFM creates a PM job for a virtualized resource of type vStorage for data collection and a threshold notification. In order a threshold to be active, there must be a PM job collecting the needed metric for the selected entries.  By increasing the network load, the vStorage threshold could be triggered. VNFM should process the notification and indicate the event.  The PM job should have following inputs:   * Resource selector is vStorage * performance metric type * collection period   The VIM should report the PM job with the same parameters.  The notification can report the availability of performance information and/or threshold crossed notifications. This test is for performance threshold notification. For the threshold crossed notification to occur, the emission must stay on to generate cpu load over threshold and longer than excessive time.  Invocation method could be   * vCPU instance with threshold level of 1% * vNetwork bandwidth with threshold level of 1% * vMemory usage with threshold level of 80% * vStorage usage with threshold level of 70% |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate traffic 4. Validate traffic flows through 5. Issue command to the VNFM to register for vStorage performance data collection 6. Validate via VNFM and VIM interface that registration was successful 7. Issue command to the VNFM to register for vStorage performance threshold notification 8. Validate via VNFM and VIM interface that registration was successful 9. Increase the load to cause the event 10. Validate that VNFM gets the notification when the threshold is reached 11. Issue command to the VNFM to deregister for vStorage performance threshold notification 12. Validate via VNFM and VIM interface that deregistration was successful 13. Issue command to the VNFM to deregister for vStorage performance data collection 14. Validate via VNFM and VIM interface that deregistration was successful 15. Terminate VNF |
| Extensions: |  |
| Notes: | There are many sub cases where the threshold could be set on different statistic values and levels. But the focus is here not on the VIM functionality. We concentrate on the VNFM. So it is more important that VNFM take the different command parameter and apply them correctly to the VIM. |

# Complex Use-cases

|  |  |
| --- | --- |
| TC.VNF.COMPLEX.001 VNF Start and Scaling with max traffic load | |
| Purpose: | Verify VNF can be activated under max traffic load, processes packets after the activation and measure the activation/fully serviceability time when the scaling is completed and no packets get dropped |
| Description | After the VNF is instantiated, the VNF state is “Inactive. To allow processing of its functionality, it must be activated. Via the VNFM the command “Start” is been issued to bring the VNF into an active state and validate the functionality by emitting packets and measuring its correctness. The max traffic gets emitted before the emission to allow the measure of the activation time and also the quality (dropped packets after receiving the first packet) and how fast the VM can scale up to process max traffic without drops. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Measure the time it takes for the activation and forwards the max traffic. We may need to go through multiple iterations to get an average time.  Sequence:   1. Start the max traffic load 2. Instantiate VNF 3. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 4. Wait for the VNF to scale up to the maximum (--> time stamp) 5. Validate traffic flows with no dropped packets (--> time stamp) 6. Validate the allocated vResources are on max level 7. Calculate the time for the activation (first time stamp arrival and no dropped packets) 8. Stop VNF 9. Ensure that no traffic flows once stop is completed 10. Stop Traffic Load 11. Terminate VNF |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-up the VNF doesn’t add new VNFCs. It keeps the number of VNFC constant and just adds additional resources. These are sub test cases.  The scaling trigger could be one of the following:   * VNFM/MANO generated events - Auto Scaling ( e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling ( e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling   Each of this trigger can be seen a sub test cases. |

|  |  |
| --- | --- |
| TC.VNF.COMPLEX.002 Stop a max scale-up/scaled-out VNF instance in state Active under max traffic load | |
| Purpose: | A VNF can be scaled out or up to increase it throughput performance. A VNFM should allow and have the capability to stop a fully scaled VNF under a full load. The stopping of the VNF may also cause to free up unused vResources. |
| Description | The VNFM should add additional allocated virtualized compute resources up to the limit defined by the VNFD. Any additional request to scale-up should be ignored. Traffic should be generated to validate that the VNFCs can process packets correctly. The triggers for this event to occur could be by different options but should be ignored when reached the limit:   * VNFM/MANO generated events - Auto Scaling ( e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling ( e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling   Once the VNF is in the max scaled-up/scaled-out state, the VNFM gets requested to terminate the VNF under load. |
| Test Setup |  |
| Parameters | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (🡪 no dropped packets) 5. Trigger a resize of the NFV resources to reach the maximum 6. Validate VNF has resized to the max and has max capacity 7. Generate max traffic load to load all VNF instances 8. Validate all traffic flows through and has reached max capacity 9. Clear counters 10. Stop the VNF (--> time stamp) 11. Validate VNF has been stopped (--> time stamp) 12. Validate no traffic flows through (--> last arrival time stamp) 13. Stop traffic 14. Calculate the time to stop a max scaled VNF under load (🡪 last arrival time stamp) |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure |  |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-up the VNF doesn’t add new VNFCs. It keeps the number of VNFC constant and just adds additional resources.  The scaling trigger could be one of the following:   * VNFM/MANO generated events - Auto Scaling ( e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling ( e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling   Each of this trigger can be seen a sub test cases. |

|  |  |
| --- | --- |
| TC.VNF.COMPLEX.003 Terminate max scale-up/out VNF in state Active under max traffic load | |
| Purpose: | A VNF can be scaled out or up to increase it throughput performance. A VNFM should allow and have the capability to terminate a fully scaled VNF under a full load. The termination of the VNF results in the release of all vResources. |
| Description | The VNFM should add additional allocated virtualized compute resources up to the limit defined by the VNFD. Any additional request to scale-up should be ignored. Traffic should be generated to validate that the VNFCs can process packets correctly. The triggers for this event to occur could be by different options but should be ignored when reached the limit:   * VNFM/MANO generated events - Auto Scaling ( e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling ( e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling   Once the VNF is in the max scaled-up/scaled-out state, the VNFM gets requested to terminate the VNF under max load. |
| Test Setup |  |
| Parameters | Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the maximum 6. Generate max traffic load to load all VNF instances 7. Validate VNF has resized to the max and has max capacity 8. Terminate the VNF (--> time stamp) 9. Validate VNF has been terminated and all resources have been release (--> time stamp) 10. Validate no traffic flows through 11. Stop traffic 12. Calculate the time to terminate the max scaled VNF under load |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure |  |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-up the VNF doesn’t add new VNFCs. It keeps the number of VNFC constant and just adds additional resources.  The scaling trigger could be one of the following:   * VNFM/MANO generated events - Auto Scaling ( e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling ( e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling   Each of this trigger can be seen a sub test cases. |

|  |  |
| --- | --- |
| TC.VNF.COMPLEX.004 VNF upgrade, update or rollback in state Active with max traffic | |
| Purpose: | Verify VNF can be upgraded, updated or rolled back in state “Active” and returns into the same state back. The process is done under max traffic to determine the service disruption time and the new VNF can scale right away up to the max limit. |
| Description | After the VNF is instantiated, configured and activated, the VNF state is “Active”. The command VNF upgrade, update or rollback gets initiated. After completion the new VNF should be still in the same state. The EM or the config file should be applied to the new VNF instance and return to the active state by processing packets. After termination all reserved vResources of the VNF must be released. |
| Test Setup |  |
| Parameters |  |
| Success Guarantee: |  |
| Pre-conditions: |  |
| Procedure | Ensure VNF receive the required configuration from the EM or configuration file  Sequence:   1. Instantiate VNF 2. Validate VNF instantiation state is INSTANTIATED and VNF state is STARTED (--> time stamp when correct state reached) 3. Generate low traffic load 4. Validate that traffic flows through without issues (--> no dropped packets) 5. Trigger a resize of the NFV resources to the maximum 6. Start the max traffic load 7. Validate traffic goes through (--> no dropped packets) 8. Initiate the upgrade, update or rollback process 9. Validate VNF new version is active and state is Active 10. Validate VNF has scaled up to the max level, traffic gets forwarded and measure the length of service disruption 11. Stop Traffic 12. Terminate VNF 13. Validate that all vResources have been released |
| Extensions: |  |
| Notes: | Following resources could be scaled up:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware   In case scale-up the VNF doesn’t add new VNFCs. It keeps the number of VNFC constant and just adds additional resources.  The scaling trigger could be one of the following:   * VNFM/MANO generated events - Auto Scaling ( e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling ( e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling   Each of this trigger can be seen a sub test cases.  The test case includes three different sub test cases:   * Upgrade a VNF * Update a VNF * Rollback a VNF |

# Open Use Cases

* Capacity utilization for calendar time

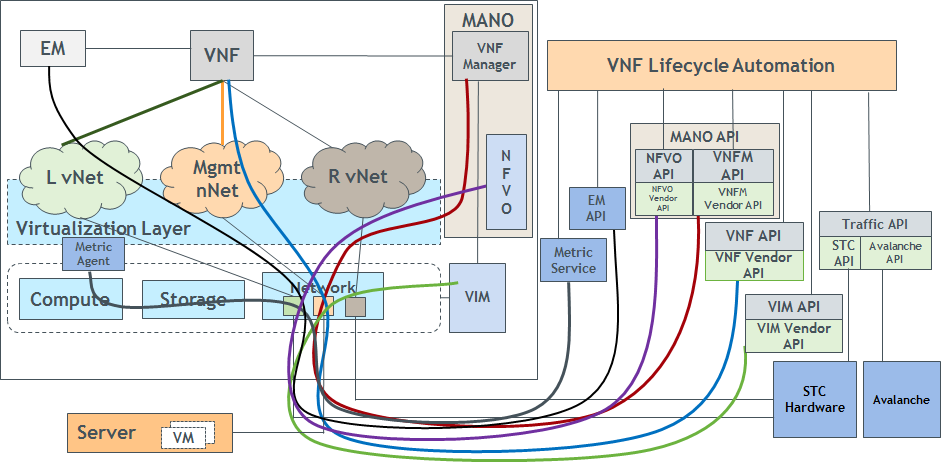
NFVI allocates and deallocates virtual resources for a given time period based on calendar time

* Capacity query for a time period

Virtual resource allocation that includes a time period for what the capacity is queried

# API Design and Architecture

The overall design of the VNF Lifecycle Management validation is illustrated in the below diagram. The VNF Lifecycle Automation is the heart of the system and implements the sequence of the different test cases.



The VNF Lifecycle Automation takes advantage of the following modules:

1. EM API module
2. NFVO API Module
3. VNF API Module
4. VNFM API Module
5. VNFD API Module
6. VIM API Module
7. Traffic API Module

The VNF Lifecycle Automation needs to detect and handle the following conditions:

1. Presence or unavailability of NFVO
2. EM or VNF config file
3. License file needed or not
4. …

## VNF RECORD

The VNF Record context allows us to reference a single VNF and to target manipulations to a single VNF when supporting multiple VNFs.

|  |  |
| --- | --- |
| 1. Function: VNFRECORD.Create() | |
| Input: | None |
| Return | vnfRecord SUCCESS  FALSE Error |
| Description | The command creates a VNF record where all VNF specific commands will receive this vnfRecord as a parameter. In Pythin this command could be implemented as a creation of a class. |

|  |  |
| --- | --- |
| 1. Function: VNFRECORD.Delete() | |
| Input: | vnfRecord |
| Return | TRUE SUCCESS  FALSE Error |
| Description | The command erases a VNF record. In Pythin this command could be implemented as a command del vnfRecord. |

## EM API module

The EM API module supports the access to the EM. A VNF may depend on the existence of a EM and its capability to configure the VNF. Once the VNF is configured by the EM, the VNF can perform its tasks and process packets.

class ElementManagement:

def GetSate()

def StartEM ()

def StopEM()

def SuspendEM()

def SetEMConfiguration()

def RemoveEMConfiguration()

def RecoveryCMDtoEM()

|  |  |
| --- | --- |
| 1. Function: ElementManagement.GetState() | |
| Input: | vnfRecord |
| Return | RUN EM runs and active  SUSPENDED EM suspended  STOPPED EM stopped |
| Description | The command returns the current state of the EM. |

|  |  |
| --- | --- |
| 1. Function: ElementManagement.Start() | |
| Input: | vnfRecord |
| Return | TRUE EM runs  FALSE EM stopped |
| Description | The command StartEM starts the EM and returns TRUE on success. If the EM has been already started, this command will either return TRUE or restart the EM. If the EM can’t be instantiated, the command returns FALSE. |

|  |  |
| --- | --- |
| 1. Function: ElementManagement.Stop() | |
| Input: | vnfRecord |
| Return | TRUE EM stopped  FALSE EM runs |
| Description | The command StopEM stops the EM and returns TRUE on success. If the EM has been already stopped, this command will return TRUE. If the EM can’t be stopped, the command returns FALSE. |

|  |  |
| --- | --- |
| 1. Function: ElementManagement.SuspendEM() | |
| Input: | vnfRecord |
| Return | TRUE EM suspended  FALSE EM state not change |
| Description | The command suspended the EM so that it does not process any information. It basically drops all messages from the VNFM. |

|  |  |
| --- | --- |
| 1. Function: ElementManagement.SetEMConfiguration() | |
| Input: | vnfRecord  Link to the EM configuration File |
| Return | TRUE EM configuration successful applied  FALSE EM configuration apply error |
| Description | The command applies EM configuration so that the EM can work correctly and service the VNF with required configurations. |

|  |  |
| --- | --- |
| 1. Function: ElementManagement.RemoveEMConfiguration() | |
| Input: | vnfRecord |
| Return | TRUE EM configuration removed  FALSE EM configuration could not be removed |
| Description | The command removes all configuration from the EM. The EM can therefore not configure and manage an VNF. |

|  |  |
| --- | --- |
| 1. Function: EM.RecoveryCMDtoEM() | |
| Input: | vnfRecord |
| Return | TRUE Recovery command initiated  FALSE Failed to initiate the recovery |
| Description | The command instructs the EM to initiate the recovery process of the VNF. The EM will instruct the VNFM to actual do the recovery process. |

## NFVO API Module

If the NFV environment would have a NFVO in place, the following module would come available for use.

class NFVO:

def RecoveryCMDtoNFVO()

|  |  |
| --- | --- |
| 1. Function: NFVO.RecoveryCMDtoNFVO() | |
| Input: | vnfRecord |
| Return | TRUE Recovery command initiated  FALSE Failed to initiate the recovery |
| Description | The command instructs the NFVO to initiate the recovery process of the VNF. The NFVO will instruct the VNFM to actual do the recovery process. |

## VNF API Module

The VNF module provides access to the configuration and license file for the VNF.

class VNF:

def SetVNFConfigFile()

def RemoveVNFConfigFile()

def SetVNFLicense()

def RemoveVNFLicense()

|  |  |
| --- | --- |
| 1. Function: VNF.SetVNFConfigFile() | |
| Input: | vnfRecord  Link to the VNF configuration file |
| Return | TRUE VNF configuration successful applied  FALSE VNF configuration apply error |
| Description | The command applies static VNF configuration so that the VNF can work correctly and service the traffic. |

|  |  |
| --- | --- |
| 1. Function: VNF.RemoveVNFConfigFile() | |
| Input: | vnfRecord |
| Return | TRUE VNF configuration successful removed  FALSE VNF configuration file removal error |
| Description | The command removes the VNF configuration so that the VNF loses its configuration. |

|  |  |
| --- | --- |
| 1. Function: VNF.SetVNFLicense() | |
| Input: | vnfRecord  Link to VNF License File |
| Return | TRUE VNF license successful applied  FALSE VNF license apply error |
| Description | The command applies VNF license so that the VNF can be configured and go into service. |

|  |  |
| --- | --- |
| 1. Function: VNF.RemoveVNFLicense() | |
| Input: | vnfRecord |
| Return | TRUE VNF license successful removed  FALSE VNF license removal error |
| Description | The command removes the VNF license. |

## VNFM API Module

The VNFM module communicates with the VNF Manager. It provides following groups of functionality:

1. Software Image
2. VNFD
3. VNF
4. Migration
5. vNetwork
6. vStorage
7. Fault Management
8. Performance Management

class VNFM:

def swImage.ListSwImage()

def VNFD.GetLifecyleManagementInfo()

def VNFD.GetvResourceList()

def VNFD.GetvStorageSize()

VNFM.Snapshot.CreateVnfSnapshotImage()

def Snapshot.CreateComputeImage()

def Snapshot.DeleteComputeImage

def Snapshot.GetComputeImageList()

def Snapshot.InstantiateComputeImage()

def Snapshot.CreateStorageImage()

def Snapshot.DeleteStorageImage()

def Snapshot.GetStorageImageList

def Snapshot.InstantiateStorageImage()

VNFM.Snapshot.EraseStorageID()

def Snapshot.MountStorageImage()

def Snapshot.ValidateStorageIsMounted()

def Snapshot.ValidateComputeIsInstantiated()

VNFM.VNF.GetComputeID()

VNFM.VNF.GetStorageID()

def VNF.Instantiate()

def VNF.Terminate()

def VNF.Start()

def VNF.Stop()

def VNF.Upgrade()

def VNF.Update()

def VNF.Rollback()

def VNF.GetVNFstate()

def VNF.GetvResourceList()

def VNF.ValidateAllocatedvResources()

def VNF.AllvResourcesReleased()

def VNF.GetScalingLevel()

def VNF.ScaleUpOrDown()

def VNF.ScaleInOrOut()

def VNF.ValidateResized()

def VNF.TriggerResizevRresources()

def Migration.TriggerMigration()

def Migration.ValidateMigrationSuccess()

def RecoveryCMDtoVNFM()

def VNF.RecoveryCMDtoVNF()

def VNetwork.ValidatevNetworkResourceAllocation()

def VNetwork.ValidatevNetworkResourceRelease()

def VNetwork.SetBandwidth()

def VStorage.GetvStorageSize()

def VStorage.TriggerIncreasevStorage()

def VStorage.DecreasevStorage()

def Fault.RegisterForNotification()

def Fault.DeregisterForNotification()

def Fault.ListFaultNotificationRegistration()

def Fault.ListFaultNotificationInformation()

def Fault.VNFFaultInjection()

def Fault.VNFFaultInjectionRemoval()

def Performance.ListPerformanceNotificationRegistration()

def Performance.RegisterPerformanceNotification()

def Performance.DeregisterPerformanceNotification()

def Performance.ListPerformanceDataInformation()

### Software Image

The group of Software Image includes all commands related to the software Images.

|  |  |
| --- | --- |
| 1. Function: VNFM.swImage.ListSwImage() | |
| Input: | Filter Parameter:   * ComputeImageID * Name * Provider * Version * checksum * Container Format * Disk Format * Create date * Update Date * Size of the software * Status of the software * User defined meta Data   If none is given, no filter should be applied |
| Return | List of Software Images with the following information per image:   * vnfPackageID * Name * Provider * Version * checksum * Container Format * Disk Format * Create date * Update Date * Size of the software * Status of the software * User defined meta Data   If none is returned, no software image found. |
| Description | The command returns the installed software images by applying the filter and further reducing the list. |

### VNFD

The VNFD commands give access to the VNFD provided by the VNFM.

|  |  |
| --- | --- |
| 1. Function: VNFM.VNFD.GetLifecyleManagementInfo() | |
| Input: | vnfPackageID |
| Return | Lifecycle Management Information |
| Description | The command retrieves the lifecycle management information from the VNFD. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNFD.GetvResourceList() | |
| Input: | 1. Software Image ID 2. Traffic Load:    1. Initial    2. Low traffic load    3. Normal traffic    4. Max traffic load |
| Return | vResource List of the Software Image for the specified traffic load |
| Description | The command returns the vResource list for the specified traffic load. The VNF would have allocated the vResources when the specified traffic load would be applied. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNFD.GetvStorageSize() | |
| Input: | 1. Software Image ID 2. Traffic Load:    1. Initial    2. Low traffic load    3. Normal traffic    4. Max traffic load |
| Return | vStorage size for the Software Image for the specified traffic load |
| Description | The command returns the vStorage size for the specified traffic load. The VNF would have allocated the vStorage size when the specified traffic load would be applied. |

### Snapshot

Snapshot related commands are listed in this command group.

|  |  |
| --- | --- |
| 1. Function: VNFM.Snapshot.CreateVnfSnapshotImage() | |
| Input: | vnfRecord  snapshotType – SNAPSHOT\_TYPE\_HOT or SNAPSHOT\_TYPE\_COLD |
| Return | vnfSnapshotID Success  Empty List Failure, e.g. VNF is not instantiated |
| Description | The command initiates the snapshot image creation of the specified VNF. Snapshots are been created for the compute nodes and the storage nodes. If the VNF would have multiple compute nodes, they would be bundled under a single ComputeImageID. The same applies to storage. This simplifies the later instantiation.  There are two types of snapshots, cold snapshot when only the disks of the VNFC’s are captured and warm snapshots when not only the disks, but also the memory and device states of the VNFC’s are captured. A cold snapshot is useful to create backups of the VNF/VNFC while the warm snapshot can be used to debug the VNF/VNFC state after a critical error or can be used for backup purposes if the operating system and application of the VNF/VNFC is able to handle the restoration of past state. This results into following :   1. snapshotType is set to SNAPSHOT\_TYPE\_COLD: VNFM.Snapshot.CreateVnfSnapshotImage() calls only VNFM.Snapshot.CreateStorageImage() 2. snapshotType is set to SNAPSHOT\_TYPE\_WARM: VNFM.Snapshot.CreateVnfSnapshotImage() calls VNFM.Snapshot.CreateStorageImage() and VNFM.Snapshot.CreateComputeImage()   As a result of this difference of snapshots, each snapshot image must have information of what type it is and the content of it. If snapshot includes only storage, it can not be used to instantiate a whole VNF as the compute (memory) part is missing. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Snapshot.DeleteVnfSnapshotImage() | |
| Input: | vnfSnapshotID  snapshotType – SNAPSHOT\_TYPE\_HOT or SNAPSHOT\_TYPE\_COLD |
| Return | TRUE Success  FALSE Failure |
| Description | The command erases the specified snapshot image based on the provided snapshot Type:  snapshotType is set to SNAPSHOT\_TYPE\_COLD:  VNFM.Snapshot.DeleteVnfSnapshotImage() calls only VNFM.Snapshot.DeleteStorageImage()  snapshotType is set to SNAPSHOT\_TYPE\_WARM:  VNFM.Snapshot.DeleteVnfSnapshotImage() calls VNFM.Snapshot.DeleteStorageImage() and VNFM.Snapshot.DeleteComputeImage() |
| 1. Function: VNFM.Snapshot.GetVnfSnapshotImageList() | |
| Input: | Filter:   * None 🡪 no filter * vnfSnapshotID * vnfsnapshotType * Date * Name |
| Return | List of vCompute Image information with information per image:   * vnfSnapshotID * vnfsnapshotType * Name * Date * Size * Owner |
| Description | The command lists all the vnfsnapshot images on the system by applying the specified filter. If the parameter vnfsnapshotType  snapshotType is set to SNAPSHOT\_TYPE\_COLD: VNFM.Snapshot.GetVnfSnapshotImageList() calls only VNFM.Snapshot.GetStorageImageList()  2.         snapshotType is set to SNAPSHOT\_TYPE\_WARM: VNFM.Snapshot.GetVnfSnapshotImageList() calls VNFM.Snapshot.GetStorageImageList() and VNFM.Snapshot.GetComputeImageList() - should return a snapshotID if both compute and storage snapshot exist. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Snapshot.InstantiateVnfSnapshotImage() | |
| Input: | vnfRecord  vnfSnapshotID |
| Return | TRUE Success  FALSE Failure |
| Description | The command instantiates the stored vnfSnapshot image which may include Compute image and Storage Image to form a VNF. The snapshotID should include all required images to instantiate the VNF. The instantiation is driven by the VNF snapshot type:  snapshotType is set to SNAPSHOT\_TYPE\_COLD  VNFM.Snapshot.InstantiateVnfSnapshotImage() calls only VNFM.Snapshot.InstantiateStorageImage()  snapshotType is set to SNAPSHOT\_TYPE\_WARM  VNFM.Snapshot.InstantiateVnfSnapshotImage() calls VNFM.Snapshot.InstantiateStorageImage() and VNFM.Snapshot.InstantiateComputeImage() |

|  |  |
| --- | --- |
| 1. Function: VNFM.Snapshot.CreateComputeImage() | |
| Input: | vnfRecord |
| Return | vnfSnapshotID Success  FALSE Failure |
| Description | The command initiates the snapshot image creation of the specified VNF. If the VNF would have multiple compute nodes or VMs, each compute node image gets bundles under a single compute image ID. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Snapshot.DeleteComputeImage() | |
| Input: | vnfSnapshotID |
| Return | TRUE Success  FALSE Failure |
| Description | The command erases the specified vCompute node image. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Snapshot.GetComputeImageList() | |
| Input: | Filter:   * None 🡪 no filter * vnfSnapshotIDDate * Name |
| Return | List of vCompute Image information with information per image:   * vnfSnapshotID * Name * Date * Size * Owner |
| Description | The command lists all the vCompute images on the system by applying the specified filter. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Snapshot.InstantiateComputeImage() | |
| Input: | vnfRecord  vnfSnapshotID |
| Return | TRUE Success  FALSE Failure |
| Description | The command instantiates the stored Compute node image and Storage Image to form a VNF. The snapshotID should include all required images to instantiate the VNF |

|  |  |
| --- | --- |
| 1. Function: VNFM.Snapshot.CreateStorageImage() | |
| Input: | vnfRecord |
| Return | vnfSnapshotID Success  FALSE Failure |
| Description | The command initiates the snapshot image creation of the specified VNF. A VNF may have multiple vStorage nodes in use. This call would bundle all storage nodes of the VNF as a single storage image for simpler handling. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Snapshot.DeleteStorageImage() | |
| Input: | vnfRecord  vnfSnapshotID |
| Return | TRUE Success  FALSE Failure |
| Description | The command erases the specified vStorage image. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Snapshot.GetStorageImageList() | |
| Input: | Filter:   * None 🡪 no filter * vnfSnapshotIDDate * Name |
| Return | List of vStorage Image information with information per image:   * vnfSnapshotIDName * Date * Size * Owner |
| Description | The command lists all the vStorage images on the system by applying the specified filter. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Snapshot.InstantiateStorageImage() | |
| Input: | vnfSnapshotID |
| Return | StorageID Success  None Failure |
| Description | The command instantiates the stored vStorage image of the snapshot. If the snapshot would include additional images like compute node images, they will be ignored. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Snapshot.EraseStorageID() | |
| Input: | StorageID |
| Return | TRUE Success  FALSE Failure |
| Description | The command releases the instantiated vStorage image. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Snapshot.MountStorageImage() | |
| Input: | vnfRecord  StorageID |
| Return | TRUE Success  FALSE Failure |
| Description | The command mounts the vStorage to the VNF for use. The current active storage gets replaced by the provided storage ID and replaced storage gets released. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Snapshot.ValidateStorageIsMounted() | |
| Input: | vnfRecord  StorageID |
| Return | TRUE Success  FALSE Failure |
| Description | The command validates that the given vStorage ID is mounted to the VNF for use. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Snapshot.ValidateComputeIsInstantiated() | |
| Input: | vnfRecord  vnfSnapshotID |
| Return | TRUE Success  FALSE Failure |
| Description | The command validates that the given ComputeImage ID is instantiated as a VNF. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNF.GetComputeID() | |
| Input: | vnfRecord |
| Return | ComputeID Success  FALSE Failure |
| Description | The command returns the Compute ID for the provided VNF. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNF.GetStorageID() | |
| Input: | vnfRecord |
| Return | StorageID Success  FALSE Failure |
| Description | The command returns Storage ID for the provided VNF. |

### VNF

VNF related commands like instantiation, start, stop and termination are in the group VNF.

|  |  |
| --- | --- |
| 1. Function: VNFM.VNF.Instantiate() | |
| Input: | vnfRecord  vnfPackageID |
| Return | TRUE Success  FALSE Failure |
| Description | The command instantiates the software image given by the vnfPackageID. At the end a new VNF has been instantiated. It allocates all the required resources so that the VNF is functional working. EM, license file and static configuration file must be setup prior to this call. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNF.Terminate() | |
| Input: | vnfRecord |
| Return | TRUE Success  FALSE Failure |
| Description | The command terminates the VNF referenced by the ComputeID. All vResources has been released when the command returns. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNF.Start() | |
| Input: | vnfRecord |
| Return | TRUE Success  FALSE Failure |
| Description | The command transitions the VNF from the state “Inactive” to the state “Active”. If a problem occurred, it returns FALSE. Otherwise the VNF will be able to forward and process packets. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNF.Stop() | |
| Input: | vnfRecord |
| Return | TRUE Success  FALSE Failure |
| Description | The command transitions the VNF from the state “Active” to the state “Inactive”. If a problem occurred, it returns FALSE. Otherwise the VNF will lose its capability to forward and process packets. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNF.Upgrade() | |
| Input: | vnfRecord to be upgraded existing VNF instance  vnfPackageID the new sw image to what the VNF should be upgraded |
| Return | TRUE Success, the new compute ID of the upgraded entity  FALSE Failure |
| Description | A VNF may need to be upgraded to a new version of the software. The command will replace the existing VNF with the new software image. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNF.Update() | |
| Input: | vnfRecord to be updated existing VNF instance  vnfPackageID the new sw image to what the VNF should be updated |
| Return | TRUE Success, the new compute ID of the updated entity  FALSE Failure |
| Description | A VNF may need to be updated with a bug fix of the software. The command will replace the existing VNF with the new software image. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNF.Rollback() | |
| Input: | vnfRecord to be rolledback existing VNF instance  vnfPackageID the new sw image to what the VNF should be rolledback |
| Return | TRUE Success, the new compute ID of the rolledback entity  FALSE Failure |
| Description | A VNF may need to be rolledback to an older version of the software. The command will replace the existing VNF with the older software image. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNF.GetVNFstate() | |
| Input: | vnfRecord |
| Return | State of the VNF:   * NotConfigured * Inactive * Active * FALSE |
| Description | The command returns the current state of the VNF. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNF.GetvResourceList() | |
| Input: | vnfRecord |
| Return | Returns the list of all allocated vResources of the instantiated VNF |
| Description | The command returns a list of all allocated vResources of the instantiated VNF |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNF.ValidateAllocatedvResources() | |
| Input: | vnfRecord  vnfPackageID  VNF Resize Level:   * Initial (INITIAL\_VRESOURCE\_LEVEL) * Normal resize vResource Level (NORMAL\_VRESOURCE\_LEVEL) * Elevated resize vResource Level (ELEVATED\_VRESOURCE\_LEVEL) * Max resize vResource Level (MAX\_RESIZE\_VRESOURCE\_LEVEL) * Max resize vResource Level but reduced by VIM vResources (MAX\_REDUCED\_RESIZE\_VRESOURCE\_LEVEL)   Type of vResource to be validated:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware * vResource = any |
| Return | TRUE vResources are correctly allocated  FALSE Failure, vResource mismatch |
| Description | The command validates that the VNF has been assigned the expected vResources in the current state. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNF.AllvResourcesReleased() | |
| Input: | vnfRecord |
| Return | TRUE Success  FALSE Failure |
| Description | The command validates that all vResources of the VNF have been released after the termination command. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNF.GetScalingLevel() | |
| Input: | vnfRecord |
| Return | VNF Resize Level:   * Initial (INITIAL\_VRESOURCE\_LEVEL) * Normal resize vResource Level (NORMAL\_VRESOURCE\_LEVEL) * Elevated resize vResource Level (ELEVATED\_VRESOURCE\_LEVEL) * Max resize vResource Level (MAX\_RESIZE\_VRESOURCE\_LEVEL) * Max resize vResource Level but reduced by VIM vResources (MAX\_REDUCED\_RESIZE\_VRESOURCE\_LEVEL) |
| Description | The command delivers the current vResource level of the instantiated VNF. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNF.ScaleUpOrDown() | |
| Input: | vnfRecord  Previous VNF Resize Level:   * Initial (INITIAL\_VRESOURCE\_LEVEL) * Normal resize vResource Level (NORMAL\_VRESOURCE\_LEVEL) * Elevated resize vResource Level (ELEVATED\_VRESOURCE\_LEVEL) * Max resize vResource Level (MAX\_RESIZE\_VRESOURCE\_LEVEL) * Max resize vResource Level but reduced by VIM vResources (MAX\_REDUCED\_RESIZE\_VRESOURCE\_LEVEL) |
| Return | DOWN VNF resize down  UP VNF resized up  EQUAL VNF didn’t resize |
| Description | The command compares the provided resize level to the current active VNF resize level. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNF.ScaleInOrOut() | |
| Input: | vnfRecord  Previous VNF Resize Level:   * Initial (INITIAL\_VRESOURCE\_LEVEL) * Normal resize vResource Level (NORMAL\_VRESOURCE\_LEVEL) * Elevated resize vResource Level (ELEVATED\_VRESOURCE\_LEVEL) * Max resize vResource Level (MAX\_RESIZE\_VRESOURCE\_LEVEL) * Max resize vResource Level but reduced by VIM vResources (MAX\_REDUCED\_RESIZE\_VRESOURCE\_LEVEL) |
| Return | IN VNF resize down  OUT VNF resized up  EQUAL VNF didn’t resize |
| Description | The command compares the provided resize level to the current active VNF resize level. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNF.TriggerResizevResources() | |
| Input: | vnfRecord  Trigger type:   * VNFM/MANO generated events - Auto Scaling ( e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM) * VNF generated event – On-demand Scaling ( e.g. based on traffic, VNF generates the request internally) * VIM generated event – On-demand Scaling * EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale) * MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API) * VNF generated event - Manual Scaling * EM generated event – Manual Scaling   Resize Level:   * Initial (INITIAL\_VRESOURCE\_LEVEL) * Normal resize vResource Level (NORMAL\_VRESOURCE\_LEVEL) * Elevated resize vResource Level (ELEVATED\_VRESOURCE\_LEVEL) * Max resize vResource Level (MAX\_RESIZE\_VRESOURCE\_LEVEL) * Max resize vResource Level but reduced by VIM vResources (MAX\_REDUCED\_RESIZE\_VRESOURCE\_LEVEL)   Type of vResource to be resized:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware * vResource = any |
| Return | TRUE Success  FALSE Failure |
| Description | The command’s task is to initiate the resizing of the VNF. To trigger a resize of a VNF, depends on the type of the VNF implementation and the used functionality. |

### Migration

Migration related commands can be found here.

|  |  |
| --- | --- |
| 1. Function: VNFM.Migration.GetMigrationType () | |
| Input: | vnfRecord |
| Return | LIVE\_MIGRATION VNF supports live migration  OFFLINE\_MIGRATION VNF supports offline migration  FALSE No migration supported by VNF |
| Description | The command returns migration type supported by VNF. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Migration.TriggerMigration() | |
| Input: | vnfRecord  Resource Type for the migration:   * vMemory * vCPU cores * vStorage * vNIC or specialized hardware * any vResource |
| Return | TRUE Success, migration has been started  FALSE Failure |
| Description | The command will cause a migration of the specified vResource type. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Migration.ValidateMigrationSuccess() | |
| Input: | vnfRecord |
| Return | TRUE Success, migration completed successfully  IN\_PROGRESS Migration is still in progress  FALSE Failure, migration completed unsuccessfully |
| Description | The command returns the current status of the migration process. Therefore the command VNFM.Migration.TriggerMigration() should have been giver before calling this command. If the migration is not completed, the command returns IN\_PROGRESS. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNF.RecoveryCMDtoVNF() | |
| Input: | vnfRecord |
| Return | TRUE Success, recovery has been started  FALSE Failure |
| Description | The command instructs the VNF to start the process of a VNF recovery. This command should be given only after a VNF fault injection has been applied to the VNF so that the recovery command can heal the VNF from a problem. |

|  |  |
| --- | --- |
| 1. Function: VNFM.RecoveryCMDtoVNFM() | |
| Input: | vnfRecord |
| Return | TRUE Success, recovery has been started  FALSE Failure |
| Description | The command instructs the VNFM to start the process of a VNF recovery. This command should be given only after a VNF fault injection has been applied to the VNF so that the recovery command can heal the VNF from a problem. |

### vNetwork

vNetwork related commands are listed in this group.

|  |  |
| --- | --- |
| 1. Function: VNFM.VNetwork.ValidatevNetworkResourceAllocation() | |
| Input: | vnfRecord |
| Return | TRUE Success, vNetwork allocation and setup is correct  FALSE Failure |
| Description | The command validates the vNetwork allocation and setup against the VNFD and the current VNF Resize Level. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNetwork.ValidatevNetworkResourceRelease() | |
| Input: | vnfRecord |
| Return | TRUE Success, all vNetwork resources released  FALSE Failure |
| Description | The command validates the freeing of all vNetwork resources. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VNetwork.SetBandwidth() | |
| Input: | vnfRecord  Percentage or  Low, Medium, High |
| Return | TRUE Success, bandwidth of the vNetwork is set to the requested level  FALSE Failure |
| Description | The command modifies the default settings of the vNetwork bandwidth. |

### vStorage

vStorage commands are listed in this group.

|  |  |
| --- | --- |
| 1. Function: VNFM.VStorage.GetvStorageSize() | |
| Input: | StorageID or vnfRecord  Scale   * Gbytes * Mbytes * Bytes |
| Return | SIZE Size of the storage reported in the requested scale  FALSE Failure |
| Description | The command determines the size of the storage. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VStorage.TriggerIncreasevStorage() | |
| Input: | StorageID or vnfRecord  Trigger by   * VIM * EM * VNFM * VNF * Increased traffic load |
| Return | TRUE Success, trigger of storage increase issued  FALSE Failure |
| Description | The command issues a request to trigger a storage increase. |

|  |  |
| --- | --- |
| 1. Function: VNFM.VStorage.DecreasevStorage() | |
| Input: | StorageID or vnfRecord  Trigger by   * VIM * EM * VNFM * VNF * Increased traffic load |
| Return | TRUE Success, trigger of storage decrease issued  FALSE Failure |
| Description | The command issues a request to trigger a storage decrease. |

### Fault

Fault Management require their own group and have been listed here.

|  |  |
| --- | --- |
| 1. Function: VNFM.Fault.RegisterForNotification() | |
| Input: | vnfRecord  Type:   * vCPU * vMemory * vNetwork * vStorage * ANY   Severity:   * Critical * Major * Minor * Warning * Indeterminate * ANY |
| Return | FaultRegisterID Success  FALSE Registration failed |
| Description | The command registers a Fault Notification of the specified type and severity at the VIM through the VNFM. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Fault.DeregisterForNotification() | |
| Input: | vnfRecord  FaultRegisterID |
| Return | TRUE Successful deregistration  FALSE Failure |
| Description | The command deregisters a previous registered Fault Notification from the VIM through the VNFM. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Fault.ListFaultNotificationRegistration() | |
| Input: | vnfRecord or ANY  Type:   * vCPU * vMemory * vNetwork * vStorage * ANY   Severity:   * Critical * Major * Minor * Warning * Indeterminate * ANY |
| Return | List of Fault Notifications passing the filter with following information:   * FaultRegisterID * Type * Severity |
| Description | The command retrieves all Fault Notification registrations at the VIM passing the filter. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Fault.ListFaultNotificationInformation() | |
| Input: | FaultRegisterID or ANY  vnfRecord or ANY  Type   * vCPU * vMemory * vNetwork * vStorage |
| Return | List of  FaultRegisterID  vnfRecord  Type   * vCPU * vMemory * vNetwork   vStorage Severity   * Critical * Major * Minor * Warning * Indeterminate   Alarm State   * Fired * Updated * Cleared |
| Description | Retrieval of Fault Notification Registration Information |

|  |  |
| --- | --- |
| 1. Function: VNFM.Fault.VNFFaultInjection() | |
| Input: | vnfRecord  Command   * VIM command to stop the VNF * VIM command to stop one of the compute notes of the VNF * VIM command to cause vStorage fault * VIM command to cause vMemory fault * VIM command to cause vNetwork fault * VIM command to cause EM communication fault * VIM command to cause VNF and VNFM communication fault   Fault Type   * vCPU * vMemory * vNetwork * vStorage   Severity   * Critical * Major * Minor * Warning * Indeterminate |
| Return | faultInjectionID Success of fault injection  FALSE Failure |
| Description | The command injects a vResource fault for the VNF. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Fault.VNFFaultInjectionRemoval() | |
| Input: | vnfRecord  faultInjectionID |
| Return | TRUE Success of fault injection removal  FALSE Failure |
| Description | The command clears or removes the previously injected vResource fault for the VNF. |

### Performance

Another group is the group Performance and provides commands for Performance Management.

|  |  |
| --- | --- |
| 1. Function: VNFM.Performance.ListPMJobs() | |
| Input: | vnfRecord or ANY  pmJobID or ANY  Type   * vCPU * vMemory * vNetwork * vStorage * ALL |
| Return | List of performance management jobs   * vnfRecord * pmJobID * Type * Collection period * Reporting period * Reporting boundary to define the duration |
| Description | The command lists all performance management jobs matching the filter. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Performance.CreatePMJob() | |
| Input: | vnfRecord  Type   * vCPU * vMemory * vNetwork * vStorage * ALL   Collection period  Reporting period  Reporting boundary to define the duration |
| Return | pmJobID Successful create  FALSE Error |
| Description | The command creates a performance management job which controls the collection and reporting of performance information. The user needs to provide information to specify the type of collection, the collection period and reporting style. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Performance.DeletePMJob() | |
| Input: | vnfRecord  pmJobID |
| Return | TRUE Successful delete  FALSE Error |
| Description | The command deletes the performance management job with the given ID. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Performance.ListThresholds() | |
| Input: | vnfRecord or ANY  thresholdID or ANY  metricType   * vCPU * vMemory * vNetwork * vStorage * ALL   thresholdType   * single/multi value threshold * static/dynamic/template base threshold |
| Return | List of thresholds   * vnfRecord * thresholdID * metricType * thresholdType * single/multi value threshold * static/dynamic/template base threshold * thresholdDetails * value to be crossed * direction in which the value is crossed * details on the notification to be generated |
| Description | The command lists all thresholds matching the filter. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Performance.CreateThreshold() | |
| Input: | vnfRecord  metricType   * vCPU * vMemory * vNetwork * vStorage * ALL   thresholdType   * single/multi value threshold * static/dynamic/template base threshold   thresholdDetails   * value to be crossed * direction in which the value is crossed * details on the notification to be generated |
| Return | thresholdID Successful create  FALSE Error |
| Description | The command creates a threshold to specify threshold levels on specified metric type and VNF for which notifications will be generated when crossed. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Performance.DeleteThreshold() | |
| Input: | vnfRecord  thresholdID |
| Return | TRUE Successful delete  FALSE Error |
| Description | The command deletes the threshold with the given ID. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Performance.ListPerformanceNotificationSubscription() | |
| Input: | vnfInstanceID or ANY  perfSubscriptionID or ANY  perfNotificationType or ANY |
| Return | List of performance notifications subscriptions   * vnfInstanceID * perfSubscriptionID * perfNotificationType |
| Description | The command lists all subscriptions for performance notifications matching the filter. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Performance.SubscribePerformanceNotification() | |
| Input: | vnfInstanceID  perfNotificationType   * performance data notification * performance threshold notification |
| Return | perfSubscriptionID Successfully registered  FALSE Error |
| Description | The command subscribes for performance data notification. The user needs to provide information to specify the type of notification. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Performance.UnsubscribePerformanceNotification() | |
| Input: | vnfInstanceID  perfSubscriptionID |
| Return | TRUE Successful unsubscribe  FALSE Error |
| Description | The command unsubscribes the performance notification with the given ID. |

|  |  |
| --- | --- |
| 1. Function: VNFM.Performance.ListPerformanceDataInformation() | |
| Input: | vnfRecord  pmJobID |
| Return | List of performance reports:   * vnfType * vnfID * metric name * list of performance values:   + time stamp   + value   + unit |
| Description | The command retrieves the collected performance data information. |

## VNFD API Module

The VNFD API module is responsible to provide commands for the VNFD validation. This section requires additional work.

|  |  |
| --- | --- |
| 1. Function: () | |
| Input: |  |
| Return |  |
| Description |  |

## VIM API Module

The VIM API module gives access to the VIM. Information, statistic information, notification registrations and states can be gathered and validated against the VNFM information.

Class VIM:

swImage.ListSwImage()

swImage.Installation()

swImage.Deletion()

AddvResourceCapability()

RemovevResourceCapability()

Snapshot.GetComputeImageList()

Snapshot.DeleteComputeImage()

Snapshot.GetStorageImageList()

Snapshot.DeleteStorageImage()

ReduceNFVIvResource()

Fault.ListFaultNotificationRegistration()

Fault.ListFaultNotificationInformation()

Fault.VNFFaultInjection()

Performance.ListPerformanceNotificationRegistration()

Performance.ListPerformanceDataInformation()

The words “Cross Reference” mean that a similar VNF command exists and the same command information have been copied here at this location.

|  |
| --- |
| 1. Function: VIM.swImage.ListSwImage() |

Cross Reference:

|  |  |
| --- | --- |
| Input: | Filter Parameter:   * ComputeImageID * Name * Provider * Version * checksum * Container Format * Disk Format * Create date * Update Date * Size of the software * Status of the software * User defined meta Data   If none is given, no filter should be applied |
| Return | List of Software Images with the following information per image:   * vnfPackageID * Name * Provider * Version * checksum * Container Format * Disk Format * Create date * Update Date * Size of the software * Status of the software * User defined meta Data   If none is returned, no software image found. |
| Description | The command returns the installed software images by applying the filter and further reducing the list. |



|  |  |
| --- | --- |
| 1. Function: VIM.swImage.Installation() | |
| Input: | Link to the software image package |
| Return | vnfPackageID Installation successful  FALSE Installation Failure |
| Description | The command installs the software image in the VIM and makes it available for use. |

|  |  |
| --- | --- |
| 1. Function: VIM.swImage.Deletion() | |
| Input: | vnfPackageID |
| Return | TRUE software image has been removed  FALSE Error |
| Description | The command deletes the previous installed software image. |

|  |  |
| --- | --- |
| 1. Function: VIM.AddvResourceCapability() | |
| Input: | vResourceRemovalID |
| Return | TRUE Success  FALSE Failure |
| Description | The command adds the previous removed vResources. |

|  |  |
| --- | --- |
| 1. Function: VIM.RemovevResourceCapability() | |
| Input: | List of vResource types and counts to be removed  Type   * vCPU * vMemory * vNetwork * vStorage * vNIC   Number of vResources to be removed |
| Return | vResourceRemovalID Success  FALSE Failure |
| Description | The command removes the specified number of vResources so that the VNF scaling will not be able to take advantage of them and reaches a vResource exhaustion when scaling up/out. |

|  |
| --- |
| 1. Function: VIM.Snapshot.GetComputeImageList() |

Cross Reference

|  |  |
| --- | --- |
| Input: | Filter:   * None 🡪 no filter * vnfSnapshotIDDate * Name |
| Return | List of vCompute Image information with information per image:   * vnfSnapshotID * Name * Date * Size * Owner |
| Description | The command lists all the vCompute images on the system by applying the specified filter. |



|  |
| --- |
| 1. Function: VIM.Snapshot.DeleteComputeImage() |

Cross Reference:

|  |  |
| --- | --- |
| Input: | vnfSnapshotID |
| Return | TRUE Success  FALSE Failure |
| Description | The command erases the specified vCompute node image. |



|  |
| --- |
| 1. Function: VIM.Snapshot.GetStorageImageList() |

Cross Reference:

|  |  |
| --- | --- |
| Input: | Filter:   * None 🡪 no filter * vnfSnapshotIDDate * Name |
| Return | List of vStorage Image information with information per image:   * vnfSnapshotIDName * Date * Size * Owner |
| Description | The command lists all the vStorage images on the system by applying the specified filter. |



|  |
| --- |
| 1. Function: VIM.Snapshot.DeleteStorageImage() |

Cross Reference

|  |  |
| --- | --- |
| Input: | vnfRecord  vnfSnapshotID |
| Return | TRUE Success  FALSE Failure |
| Description | The command erases the specified vStorage image. |



|  |
| --- |
| 1. Function: VIM.Fault.ListFaultNotificationRegistration() |

Cross Reference

|  |  |
| --- | --- |
| Input: | vnfRecord or ANY  Type:   * vCPU * vMemory * vNetwork * vStorage * ANY   Severity:   * Critical * Major * Minor * Warning * Indeterminate * ANY |
| Return | List of Fault Notifications passing the filter with following information:   * FaultRegisterID * Type * Severity |
| Description | The command retrieves all Fault Notification registrations at the VIM passing the filter. |



|  |
| --- |
| 1. Function: VIM.Fault.ListFaultNotificationInformation() |

Cross Reference

|  |  |
| --- | --- |
| Input: | FaultRegisterID or ANY  vnfRecord or ANY  Type   * vCPU * vMemory * vNetwork * vStorage |
| Return | List of  FaultRegisterID  vnfRecord  Type   * vCPU * vMemory * vNetwork   vStorage Severity   * Critical * Major * Minor * Warning * Indeterminate   Alarm State   * Fired * Updated * Cleared |
| Description | Retrieval of Fault Notification Registration Information |



|  |
| --- |
| 1. Function: VIM.Fault.VNFFaultInjection() |

Cross Reference

|  |  |
| --- | --- |
| Input: | vnfRecord  Command   * VIM command to stop the VNF * VIM command to stop one of the compute notes of the VNF * VIM command to cause vStorage fault * VIM command to cause vMemory fault * VIM command to cause vNetwork fault * VIM command to cause EM communication fault * VIM command to cause VNF and VNFM communication fault   Fault Type   * vCPU * vMemory * vNetwork * vStorage   Severity   * Critical * Major * Minor * Warning * Indeterminate |
| Return | faultInjectionID Success of fault injection  FALSE Failure |
| Description | The command injects a vResource fault for the VNF. |



1. Function: VIM.Fault.VNFFaultInjectionRemoval()

Cross Reference

|  |  |
| --- | --- |
| Input: | vnfRecord  faultInjectionID |
| Return | TRUE Success of fault injection removal  FALSE Failure |
| Description | The command clears or removes the previously injected vResource fault for the VNF. |



|  |
| --- |
| 1. Function: VIM.Performance.ListPMJobs() |

Cross Reference

|  |  |
| --- | --- |
| Input: | pmJobID or ANY  Type   * vCPU * vMemory * vNetwork * vStorage * ALL |
| Return | List of performance management jobs   * pmJobID * Type * Collection period * Reporting period * Reporting boundary to define the duration |
| Description | The command lists all performance management jobs matching the filter. |

|  |
| --- |
| 1. Function: VIM.Performance.ListThresholds() |

Cross Reference

|  |  |
| --- | --- |
| Input: | thresholdID or ANY  metricType   * vCPU * vMemory * vNetwork * vStorage * ALL   thresholdType   * single/multi value threshold * static/dynamic/template base threshold |
| Return | List of thresholds   * thresholdID * metricType * thresholdType * single/multi value threshold * static/dynamic/template base threshold * thresholdDetails * value to be crossed * direction in which the value is crossed * details on the notification to be generated |
| Description | The command lists all thresholds matching the filter. |

|  |
| --- |
| 1. Function: VIM.Performance.ListPerformanceNotificationSubscription() |

Cross Reference:

|  |  |
| --- | --- |
| Input: | vnfInstanceID or ANY  perfSubscriptionID or ANY  perfNotificationType or ANY |
| Return | List of performance notifications subscriptions   * vnfInstanceID * perfSubscriptionID * perfNotificationType |
| Description | The command lists all subscriptions for performance notifications matching the filter. |



|  |
| --- |
| 1. Function: VIM.Performance.ListPerformanceDataInformation() |

Cross Reference

|  |  |
| --- | --- |
| Input: | vnfRecord  pmJobID |
| Return | List of performance reports:   * vnfType * vnfID * metric name * list of performance values:   + time stamp   + value   + unit |
| Description | The command retrieves the collected performance data information. |



## Traffic API Module

The traffic module exposes the functionality to control the test traffic.

class traffic:

StartTraffic()

StopTraffic()

ConfigTrafficLoad()

DoesTrafficFlow()

AnyTrafficLoss()

ClearCounters()

CalculateActivationTime()

CalculateDeactivationTime()

CalculateTerminationTime()

CalculateServiceDisruptionLength()

CalculateResizeServiceDisruption()

|  |  |
| --- | --- |
| 1. Function: Traffic.StartTraffic() | |
| Input: | Delay Time  Return when emission starts: True or False |
| Return | TRUE Emission started  FALSE Emission not setup |
| Description | The command setups the emission. The emission can be delayed by the Delay Time (in seconds). The flag “Return when emission starts” influences if the command should return only when the emission actual started or immediately should be returned back. |

|  |  |
| --- | --- |
| 1. Function: Traffic.StopTraffic() | |
| Input: | Delay Time  Return when emission stopped: True or False |
| Return | TRUE Emission stopped  FALSE Emission stop not setup |
| Description | The command ends the emission. The emission stop can be delayed by the Delay Time (in seconds). The flag “Return when emission starts” influences if the command should return only when the emission actual stopped or immediately should be returned back. |

|  |  |
| --- | --- |
| 1. Function: Traffic.ConfigTrafficLoad() | |
| Input: | Traffic Load:   * Low traffic load * Normal traffic * Max traffic load * Initial * Percentage   Traffic Configuration Parameter |
| Return | TRUE Traffic load and configuration parameter applied  FALSE Nothing applied, error |
| Description | The command applies the new traffic load and traffic configurations. If traffic already flows, the parameters get applied at run time. The Traffic Configuration Parameter include any specific information that are required to run the traffic. |

|  |  |
| --- | --- |
| 1. Function: Traffic.DoesTrafficFlow() | |
| Input: | None |
| Return | TRUE Traffic flow detected, VNF forwards packets  FALSE No traffic gets processed by VNF |
| Description | The command checks if the VNF processes any packets. Dropped packets may occur but are not considered as a negative outcome of the command. If all packets get dropped, the command fails. |

|  |  |
| --- | --- |
| 1. Function: Traffic.AnyTrafficLoss() | |
| Input: | None |
| Return | TRUE Traffic flows with dropped packets  FALSE Traffic flows without packet drop |
| Description | The command checks if any packets get dropped. |

|  |  |
| --- | --- |
| 1. Function: Traffic.ClearCounters() | |
| Input: | None |
| Return | TRUE All counters have been cleared  FALSE Counters could not be cleared, error |
| Description | The command gets issued to clear all counters. |

|  |  |
| --- | --- |
| 1. Function: Traffic.CalculateActivationTime() | |
| Input: | None |
| Return | Time msec  FALSE No activation time available |
| Description | The command calculates the activation time. It is assumed that a sequence of actions has been issued that allows the calculation to occur. |

|  |  |
| --- | --- |
| 1. Function: Traffic.CalculateDeactivationTime() | |
| Input: | None |
| Return | Time msec  FALSE No deactivation time available |
| Description | The command calculates the deactivation time. It is assumed that a sequence of actions has been issued that allows the calculation to occur. |

|  |  |
| --- | --- |
| 1. Function: Traffic.CalculateTerminationTime() | |
| Input: | None |
| Return | Time msec  FALSE No termination time available |
| Description | The command calculates the termination time. It is assumed that a sequence of actions has been issued that allows the calculation to occur. |

|  |  |
| --- | --- |
| 1. Function: Traffic.CalculateServiceDisruptionLength() | |
| Input: | None |
| Return | Time msec  FALSE No service disruption length (time) available |
| Description | The command calculates the service disruption length. It is assumed that a sequence of actions has been issued that allows the calculation to occur. |

|  |  |
| --- | --- |
| 1. Function: Traffic.CalculateResizeServiceDisruption() | |
| Input: | None |
| Return | Time msec  FALSE No resize service disruption time available |
| Description | The command calculates the resize service disruption time. It is assumed that a sequence of actions has been issued that allows the calculation to occur. |

# Test Case Pseudo Code

## TC.81 Stop a max scale-up/scaled-out VNF instance in state Active under max traffic load

Sequence:

1. Instantiate VNF
2. Validate VNF state is Inactive
3. Start VNF
4. Validate VNF state is Active
5. Generate low traffic load
6. Validate that traffic flows through without issues (🡪 no dropped packets)
7. Trigger a resize of the NFV resources to reach the maximum
8. Validate VNF has resized to the max and has max capacity
9. Generate max traffic load to load all VNF instances
10. Validate all traffic flows through and has reached max capacity
11. Clear counters
12. Stop the VNF (--> time stamp)
13. Validate VNF has been stopped (--> time stamp)
14. Validate no traffic flows through (--> last arrival time stamp)
15. Stop traffic
16. Calculate the time to stop a max scaled VNF under load (🡪 last arrival time stamp)

Pseudo Code:

Input:

# All required input information are stored in the test case input variable tcInput and includes following information:

vnfPackageID

Trigger type --> triggerType

* + VNFM/MANO generated events - Auto Scaling ( e.g. based on monitoring of resource utilization of the VNF's VMs, upon events received from VNF, EM, VIM)
  + VNF generated event – On-demand Scaling ( e.g. based on traffic, VNF generates the request internally)
  + VIM generated event – On-demand Scaling
  + EM generated event – On-demand Scaling (e.g. EM receives information from the VNF and determines the need to auto scale)
  + MANO generated event - Manual Scaling (e.g. by operating the MANO interface/API)
  + VNF generated event - Manual Scaling
  + EM generated event – Manual Scaling

Type of vResource to be resized --> vResourceType

* + vMemory
  + vCPU cores
  + vStorage
  + vNIC or specialized hardware
  + vResource = any

EM Config File Link 🡪 emConfigFile

VNF Config File Link 🡪 vnfConfigFile

VNF License File Link 🡪 vnfLicenseFile

Traffic Configuration Parameter 🡪 trafficConfigParameter

# Test Results are stored in the test case result variable tcResult

Comment: To make the pseudo code simpler to read, in a Python implementation the following code

tcResult.overallStatus = status

tcInput.emConfigFile

should be translated to dictionaries

tcResult[ ‘overallStatus’ ] = status

tcInput[ ‘emConfigFile’ ]

1. Instantiate VNF

vnfRecord = VNFRECORD.Create()

tcResult.overallStatus = SUCCESS

# Check Element Manger or configuration file

if EM\_VNF\_SYSTEM == TRUE :

# Instantiate Element Management

# Set the required EM configuration so that VNF can be configured by EM

if ( status = ElementManagement.SetEMConfiguration( vnfRecord, tcInput.emConfigFile )) <> TRUE :

tcResult.overallStatus = status

return ERROR

# Start the EM so that it can service the VNF

if ( status = ElementManagement.StartEM( vnfRecord) ) <> TRUE :

tcResult.overallStatus = status

return ERROR

else:

# No Element Manager required, set VNF config and license

if ( status = VNF.SetVNFConfigFile( vnfRecord, tcInput.vnfConfigFile ) ) <> TRUE :

tcResult.overallStatus = status

return ERROR

if ( status = VNF.SetVNFLicense( vnfRecord, tcInput.vnfLicenseFile ) ) <> TRUE :

tcResult.overallStatus = status

return ERROR

tcResult.timeRecord.instantiationStart = time.clock()

status = VNFM.VNF.Instantiate( vnfRecord, tcInput.vnfPackageID )

# positive computeID is success, negative computeID is unique failure code

if status <> TRUE :

tcResult.overallStatus = status

return ERROR

2. Validate VNF state is Inactive

#allow the VNF to get into the inactive state

wait\_time = MAX\_VNF\_INSTATIATION\_TIME

while ( vnfState = VNFM.VNF.GetVNFState(vnfRecord) ) <> VNF\_STATE\_INACTIVE :

if vnfState == VNF\_STATE\_ERROR :

tcResult.overallStatus = vnfState

return ERROR

if wait\_time-- == 0 :

tcResult.overallStatus = VNF\_STATE\_ERROR

return ERROR

time.sleep( 1 sec )

tcResult.timeRecord.instantiationEnd = time.clock()

tcResult.timeRecord.instantiationTime = tcResult.timeRecord.instantiationEnd – tcResult.timeRecord.instantiationStart

tcResult.resourceList.InitialVNFD = VNFM.VNFD.GetvResourceList( vnfRecord, vnfPackageID, INITIAL )

3. Start VNF

tcResult.timeRecord.startVNFStart = time.clock()

status = VNFM.VNF.Start(vnfRecord)

if status <> TRUE :

VNFM.VNF.Terminate(vnfRecord)

tcResult.overallStatus = status

return ERROR

4. Validate VNF state is Active

wait\_time = MAX\_VNF\_ACTIVATE\_TIME

while ( vnfState = VNFM.VNF.GetVNFState(vnfRecord) ) <> VNF\_STATE\_ACTIVE :

if vnfState == VNF\_STATE\_ERROR :

VNFM.VNF.Terminate(vnfRecord)

tcResult.overallStatus = vnfState

return ERROR

if wait\_time-- == 0 :

tcResult.overallStatus = VNF\_STATE\_ERROR

return ERROR

time.sleep( 1 sec )

tcResult.timeRecord.startVNFEnd = time.clock()

tcResult.timeRecord.activateTime = tcResult.timeRecord.startVNFStart – tcResult.timeRecord.startVNFEnd

tcResult.resourceList.activeResource = VNFM.VNF.GetvResourceList( vnfRecord )

if (status = VNFM.VNF.ValidateAllocatedvResources( vnfRecord, vnfPackageID, INITIAL\_VRESOURCE\_LEVEL )) <> TRUE :

VNFM.VNF.Terminate(vnfRecord)

tcResult.overallStatus = status

return ERROR

5. Generate low traffic load

if (status = traffic.ConfigTrafficLoad( LOW\_TRAFFIC\_LOAD, trafficConfigParameter )) <> TRUE :

VNFM.VNF.Terminate(vnfRecord)

tcResult.overallStatus = status

return ERROR

if (status = traffic.StartTraffic()) <> TRUE :

VNFM.VNF.Terminate(vnfRecord)

tcResult.overallStatus = status

return ERROR

6. Validate that traffic flows through without issues ( no dropped packets)

if (status = traffic.DoesTrafficFlow()) <> TRUE :

VNFM.VNF.Terminate(vnfRecord)

tcResult.overallStatus = status

return ERROR

if (status = traffic.AnyTrafficLoss()) == TRUE :

VNFM.VNF.Terminate(vnfRecord)

tcResult.overallStatus = status

return ERROR

if (status = VNFM.VNF.ValidateAllocatedvResources( vnfRecord, vnfPackageID, NORMAL\_VRESOURCE\_LEVEL )) <> TRUE :

VNFM.VNF.Terminate(vnfRecord)

tcResult.overallStatus = status

return ERROR

7. Trigger a resize of the NFV resources to reach the maximum

if (status = VNFM.VNF.TriggerResizevRresources(vnfRecord, triggerType, MAX\_RESIZE\_LEVEL, vResourceType)) <> TRUE :

traffic.stopTraffic()

VNFM.VNF.Terminate(vnfRecord)

tcResult.overallStatus = status

return ERROR

8. Validate VNF has resized to the max and has max capacity

tcResult.resourceList.maxResource = VNFM.VNF.GetvResourceList( vnfRecord)

if (status = VNFM.VNF.ValidateAllocatedvResources( vnfRecord, vnfPackageID, MAX\_RESIZE\_VRESOURCE\_LEVEL )) <> TRUE :

VNFM.VNF.Terminate(vnfRecord)

tcResult.overallStatus = status

return ERROR

9. Generate max traffic load to load all VNF instances

if (status = traffic.ConfigTrafficLoad( MAX\_TRAFFIC\_LOAD, trafficConfigParameter )) <> TRUE :

VNFM.VNF.Terminate(vnfRecord)

tcResult.overallStatus = status

return ERROR

10. Validate all traffic flows through and has reached max capacity

if (status = traffic.DoesTrafficFlow()) <> TRUE :

VNFM.VNF.Terminate(vnfRecord)

tcResult.overallStatus = status

return ERROR

if (status = traffic.AnyTrafficLoss()) == TRUE :

tcResult.errorInfo += "Traffic loss when apply max traffic"

tcResult.overallStatus |= ERROR\_TRAFFIC\_LOSS

if (status = VNFM.VNF.ValidateAllocatedvResources( vnfRecord, vnfPackageID, MAX\_VRESOURCE\_LEVEL )) <> TRUE :

tcResult.errorInfo += "vResource change when apply max traffic"

tcResult.overallStatus |= ERROR\_VRESOURCE\_CHANGE

11. Clear counters

traffic.ClearCounters()

12. Stop the VNF (--> time stamp)

tcResult.timeRecord.stopVNFStart = time.clock()

status = VNFM.VNF.Stop(vnfRecord)

if status <> SUCCESS :

VNFM.VNF.Terminate(vnfRecord)

tcResult.overallStatus = status

return ERROR

13. Validate VNF has been stopped (--> time stamp)

wait\_time = MAX\_VNF\_INACTIVATE\_TIME

while ( vnfState = VNFM.VNF.GetVNFState(vnfRecord) ) <> VNF\_STATE\_INACTIVE :

if vnfState == VNF\_STATE\_ERROR :

VNFM.VNF.Terminate(vnfRecord)

tcResult.overallStatus = vnfState

return ERROR

if wait\_time-- == 0 :

tcResult.overallStatus = VNF\_STATE\_ERROR

return ERROR

time.sleep( 1 sec )

tcResult.timeRecord.stopVNFEnd = time.clock()

14. Validate no traffic flows through (--> last arrival time stamp)

if (status = traffic.DoesTrafficFlow()) == TRUE :

VNFM.VNF.Terminate(vnfRecord)

tcResult.overallStatus = status

return ERROR

15. Stop traffic

if (status = traffic.Stop()) <> TRUE :

VNFM.VNF.Terminate(vnfRecord)

tcResult.overallStatus = status

return ERROR

16. Calculate the time to stop a max scaled VNF under load ( last arrival time stamp)

tcResult.timeRecord.deactivationTime = traffic.CalculateDeactivationTime()

tcResult.timeRecord.StopCompletionTime = tcRresult.timeRecord.stopVNFStart – tcResult.timeRecord.stopVNFEnd

vnfRecord.Delete()

return tcResult.overallStatus

# List of all Functions

[1 Function: VNFRECORD.Create() 117](#_Toc477275689)

[2 Function: VNFRECORD.Delete() 117](#_Toc477275690)

[3 Function: ElementManagement.GetState() 117](#_Toc477275691)

[4 Function: ElementManagement.Start() 118](#_Toc477275692)

[5 Function: ElementManagement.Stop() 118](#_Toc477275693)

[6 Function: ElementManagement.SuspendEM() 118](#_Toc477275694)

[7 Function: ElementManagement.SetEMConfiguration() 118](#_Toc477275695)

[8 Function: ElementManagement.RemoveEMConfiguration() 118](#_Toc477275696)

[9 Function: EM.RecoveryCMDtoEM() 118](#_Toc477275697)

[10 Function: NFVO.RecoveryCMDtoNFVO() 119](#_Toc477275698)

[11 Function: VNF.SetVNFConfigFile() 119](#_Toc477275699)

[12 Function: VNF.RemoveVNFConfigFile() 119](#_Toc477275700)

[13 Function: VNF.SetVNFLicense() 120](#_Toc477275701)

[14 Function: VNF.RemoveVNFLicense() 120](#_Toc477275702)

[15 Function: VNFM.swImage.ListSwImage() 121](#_Toc477275703)

[16 Function: VNFM.VNFD.GetLifecyleManagementInfo() 122](#_Toc477275704)

[17 Function: VNFM.VNFD.GetvResourceList() 122](#_Toc477275705)

[18 Function: VNFM.VNFD.GetvStorageSize() 122](#_Toc477275706)

[19 Function: VNFM.Snapshot.CreateVnfSnapshotImage() 123](#_Toc477275707)

[20 Function: VNFM.Snapshot.DeleteVnfSnapshotImage() 123](#_Toc477275708)

[21 Function: VNFM.Snapshot.GetVnfSnapshotImageList() 124](#_Toc477275709)

[22 Function: VNFM.Snapshot.InstantiateVnfSnapshotImage() 124](#_Toc477275710)

[23 Function: VNFM.Snapshot.CreateComputeImage() 124](#_Toc477275711)

[24 Function: VNFM.Snapshot.DeleteComputeImage() 125](#_Toc477275712)

[25 Function: VNFM.Snapshot.GetComputeImageList() 125](#_Toc477275713)

[26 Function: VNFM.Snapshot.InstantiateComputeImage() 125](#_Toc477275714)

[27 Function: VNFM.Snapshot.CreateStorageImage() 125](#_Toc477275715)

[28 Function: VNFM.Snapshot.DeleteStorageImage() 125](#_Toc477275716)

[29 Function: VNFM.Snapshot.GetStorageImageList() 126](#_Toc477275717)

[30 Function: VNFM.Snapshot.InstantiateStorageImage() 126](#_Toc477275718)

[31 Function: VNFM.Snapshot.EraseStorageID() 126](#_Toc477275719)

[32 Function: VNFM.Snapshot.MountStorageImage() 126](#_Toc477275720)

[33 Function: VNFM.Snapshot.ValidateStorageIsMounted() 126](#_Toc477275721)

[34 Function: VNFM.Snapshot.ValidateComputeIsInstantiated() 127](#_Toc477275722)

[35 Function: VNFM.VNF.GetComputeID() 127](#_Toc477275723)

[36 Function: VNFM.VNF.GetStorageID() 127](#_Toc477275724)

[37 Function: VNFM.VNF.Instantiate() 127](#_Toc477275725)

[38 Function: VNFM.VNF.Terminate() 127](#_Toc477275726)

[39 Function: VNFM.VNF.Start() 127](#_Toc477275727)

[40 Function: VNFM.VNF.Stop() 128](#_Toc477275728)

[41 Function: VNFM.VNF.Upgrade() 128](#_Toc477275729)

[42 Function: VNFM.VNF.Update() 128](#_Toc477275730)

[43 Function: VNFM.VNF.Rollback() 128](#_Toc477275731)

[44 Function: VNFM.VNF.GetVNFstate() 128](#_Toc477275732)

[45 Function: VNFM.VNF.GetvResourceList() 129](#_Toc477275733)

[46 Function: VNFM.VNF.ValidateAllocatedvResources() 129](#_Toc477275734)

[47 Function: VNFM.VNF.AllvResourcesReleased() 129](#_Toc477275735)

[48 Function: VNFM.VNF.GetScalingLevel() 129](#_Toc477275736)

[49 Function: VNFM.VNF.ScaleUpOrDown() 130](#_Toc477275737)

[50 Function: VNFM.VNF.ScaleInOrOut() 130](#_Toc477275738)

[51 Function: VNFM.VNF.TriggerResizevResources() 130](#_Toc477275739)

[52 Function: VNFM.Migration.GetMigrationType () 131](#_Toc477275740)

[53 Function: VNFM.Migration.TriggerMigration() 131](#_Toc477275741)

[54 Function: VNFM.Migration.ValidateMigrationSuccess() 131](#_Toc477275742)

[55 Function: VNFM.VNF.RecoveryCMDtoVNF() 132](#_Toc477275743)

[56 Function: VNFM.RecoveryCMDtoVNFM() 132](#_Toc477275744)

[57 Function: VNFM.VNetwork.ValidatevNetworkResourceAllocation() 132](#_Toc477275745)

[58 Function: VNFM.VNetwork.ValidatevNetworkResourceRelease() 132](#_Toc477275746)

[59 Function: VNFM.VNetwork.SetBandwidth() 132](#_Toc477275747)

[60 Function: VNFM.VStorage.GetvStorageSize() 133](#_Toc477275748)

[61 Function: VNFM.VStorage.TriggerIncreasevStorage() 133](#_Toc477275749)

[62 Function: VNFM.VStorage.DecreasevStorage() 133](#_Toc477275750)

[63 Function: VNFM.Fault.RegisterForNotification() 133](#_Toc477275751)

[64 Function: VNFM.Fault.DeregisterForNotification() 134](#_Toc477275752)

[65 Function: VNFM.Fault.ListFaultNotificationRegistration() 134](#_Toc477275753)

[66 Function: VNFM.Fault.ListFaultNotificationInformation() 135](#_Toc477275754)

[67 Function: VNFM.Fault.VNFFaultInjection() 135](#_Toc477275755)

[68 Function: VNFM.Fault.VNFFaultInjectionRemoval() 136](#_Toc477275756)

[69 Function: VNFM.Performance.ListPMJobs() 136](#_Toc477275757)

[70 Function: VNFM.Performance.CreatePMJob() 136](#_Toc477275758)

[71 Function: VNFM.Performance.DeletePMJob() 137](#_Toc477275759)

[72 Function: VNFM.Performance.ListThresholds() 137](#_Toc477275760)

[73 Function: VNFM.Performance.CreateThreshold() 137](#_Toc477275761)

[74 Function: VNFM.Performance.DeleteThreshold() 138](#_Toc477275762)

[75 Function: VNFM.Performance.ListPerformanceNotificationSubscription() 138](#_Toc477275763)

[76 Function: VNFM.Performance.SubscribePerformanceNotification() 138](#_Toc477275764)

[77 Function: VNFM.Performance.UnsubscribePerformanceNotification() 138](#_Toc477275765)

[78 Function: VNFM.Performance.ListPerformanceDataInformation() 139](#_Toc477275766)

[79 Function: () 139](#_Toc477275767)

[80 Function: VIM.swImage.ListSwImage() 140](#_Toc477275768)

[81 Function: VIM.swImage.Installation() 140](#_Toc477275769)

[82 Function: VIM.swImage.Deletion() 141](#_Toc477275770)

[83 Function: VIM.AddvResourceCapability() 141](#_Toc477275771)

[84 Function: VIM.RemovevResourceCapability() 141](#_Toc477275772)

[85 Function: VIM.Snapshot.GetComputeImageList() 141](#_Toc477275773)

[86 Function: VIM.Snapshot.DeleteComputeImage() 141](#_Toc477275774)

[87 Function: VIM.Snapshot.GetStorageImageList() 142](#_Toc477275775)

[88 Function: VIM.Snapshot.DeleteStorageImage() 142](#_Toc477275776)

[89 Function: VIM.Fault.ListFaultNotificationRegistration() 142](#_Toc477275777)

[90 Function: VIM.Fault.ListFaultNotificationInformation() 143](#_Toc477275778)

[91 Function: VIM.Fault.VNFFaultInjection() 143](#_Toc477275779)

[92 Function: VIM.Fault.VNFFaultInjectionRemoval() 144](#_Toc477275780)

[93 Function: VIM.Performance.ListPMJobs() 144](#_Toc477275781)

[94 Function: VIM.Performance.ListThresholds() 144](#_Toc477275782)

[95 Function: VIM.Performance.ListPerformanceNotificationSubscription() 145](#_Toc477275783)

[96 Function: VIM.Performance.ListPerformanceDataInformation() 145](#_Toc477275784)

[97 Function: Traffic.StartTraffic() 146](#_Toc477275785)

[98 Function: Traffic.StopTraffic() 146](#_Toc477275786)

[99 Function: Traffic.ConfigTrafficLoad() 146](#_Toc477275787)

[100 Function: Traffic.DoesTrafficFlow() 147](#_Toc477275788)

[101 Function: Traffic.AnyTrafficLoss() 147](#_Toc477275789)

[102 Function: Traffic.ClearCounters() 147](#_Toc477275790)

[103 Function: Traffic.CalculateActivationTime() 147](#_Toc477275791)

[104 Function: Traffic.CalculateDeactivationTime() 147](#_Toc477275792)

[105 Function: Traffic.CalculateTerminationTime() 147](#_Toc477275793)

[106 Function: Traffic.CalculateServiceDisruptionLength() 147](#_Toc477275794)

[107 Function: Traffic.CalculateResizeServiceDisruption() 148](#_Toc477275795)

# References

Table 18‑1. Applicable Documents

| Reference | Title |
| --- | --- |
| None |  |
|  |  |

# DESIGN CONSIDERATIONS

## Troubleshooting Aids

# Revision history

| **Rev** | **Date** | **Rev by / Req by** | **Purpose for Each Revision**  E.g. First Draft, Working draft and reason, Released, etc.. |
| --- | --- | --- | --- |
| 1 | 06/23/2016 | Klaus Seggelke | Initial Draft |
| 2 | 08/24/2016 | Klaus Seggelke | Working draft 0.1 |
| 3 | 10/20/16 | Klaus Seggelke | New ID hierarchy: vnfID is the top followed by computeID and storageID  Packaging chapter added |
| 4 | 10/20/16 |  | Introduction of vnfRecord |
| 5 | 10/20/16 |  | VNF Snapshot modification including the adding of vnfSnapshotID |
| 6 | 10/25/16 |  | Updated Pseudo code to reflect the previous changes  Add new snapshot functions with type Delete/Get/Instantiate VnfSnapshotImage |
| 7 | 10/28/2016 |  | VIM.Fault.VNFFaultInjectionRemoval() and VNFM.Fault.VNFFaultInjectionRemoval() added |
| 8 | mm/dd/yy |  | Sequence change for “VNF Start and Scaling with max traffic load” and “vCPU resource fault notification subscription and notification”  New Test Case added “Double vCPU resource fault notification subscription and notification” |
| 9 | mm/dd/yy |  |  |
| 10 | mm/dd/yy |  |  |
| 11 | mm/dd/yy |  |  |
| 12 | mm/dd/yy |  |  |
| 13 | 02/21/2017 |  | Updated scaling triggers.  Updated steps for test cases TC\_VNF\_STATE\_INST\_001, TC\_VNF\_STATE\_INST\_002 and TC\_VNF\_STATE\_TERM\_002 to match the Python OpenStack implementation. |