



## **Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; NFV descriptors based on TOSCA specification**

### *Disclaimer*

---

The present document has been produced and approved by the Network Functions Virtualisation (NFV) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.  
It does not necessarily represent the views of the entire ETSI membership.

---

**Reference**

RGS/NFV-SOL001ed261

---

**Keywords**

data, information model, model, NFV

**ETSI**

---

650 Route des Lucioles  
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C  
Association à but non lucratif enregistrée à la  
Sous-Préfecture de Grasse (06) N° 7803/88

---

**Important notice**

The present document can be downloaded from:

<http://www.etsi.org/standards-search>

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the prevailing version of an ETSI deliverable is the one made publicly available in PDF format at [www.etsi.org/deliver](http://www.etsi.org/deliver).

Users of the present document should be aware that the document may be subject to revision or change of status.

Information on the current status of this and other ETSI documents is available at

<https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx>

If you find errors in the present document, please send your comment to one of the following services:

<https://portal.etsi.org/People/CommiteeSupportStaff.aspx>

---

**Copyright Notification**

No part may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorized by written permission of ETSI.

The content of the PDF version shall not be modified without the written authorization of ETSI.

The copyright and the foregoing restriction extend to reproduction in all media.

© ETSI 2019.

All rights reserved.

**DECT™**, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members.

**3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners.

**oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners.

**GSM®** and the GSM logo are trademarks registered and owned by the GSM Association.

# Contents

Intellectual Property Rights .....	16
Foreword.....	16
Modal verbs terminology.....	16
1 Scope .....	17
2 References .....	17
2.1 Normative references .....	17
2.2 Informative references.....	18
3 Definition of terms, symbols and abbreviations.....	18
3.1 Terms.....	18
3.2 Symbols.....	19
3.3 Abbreviations .....	19
4 Overview of TOSCA model.....	19
5 General concept of using TOSCA to model NFV descriptors .....	19
5.1 Introduction .....	19
5.2 Network Service Descriptor .....	19
5.3 Virtualised Network Function Descriptor .....	20
5.4 Physical Network Function Descriptor.....	20
5.5 toasca_definitions_version and Namespace prefix .....	20
5.6 Imports statement .....	21
5.6.1 VNFD TOSCA service template .....	21
5.6.2 NSD TOSCA service template .....	21
5.6.3 PNFD TOSCA service template .....	22
5.7 Type extension .....	23
5.7.1 Introduction.....	23
5.7.2 Rules .....	23
5.7.3 VNFD Types.....	23
5.7.4 NSD types.....	24
6 VNFD TOSCA model.....	24
6.1 Introduction .....	24
6.2 Data Types.....	25
6.2.1 toasca.datatypes.nfv.L2AddressData .....	25
6.2.1.1 Description .....	25
6.2.1.2 Properties .....	25
6.2.1.3 Definition .....	25
6.2.1.4 Examples .....	26
6.2.1.5 Additional Requirements .....	26
6.2.2 toasca.datatypes.nfv.L3AddressData .....	26
6.2.2.1 Description .....	26
6.2.2.2 Properties .....	26
6.2.2.3 Definition .....	27
6.2.2.4 Examples.....	27
6.2.2.5 Additional Requirements .....	27
6.2.3 toasca.datatypes.nfv.AddressData .....	27
6.2.3.1 Description.....	27
6.2.3.2 Properties .....	28
6.2.3.3 Definition .....	28
6.2.3.4 Examples.....	29
6.2.3.5 Additional Requirements .....	29
6.2.4 toasca.datatypes.nfv.VirtualNetworkInterfaceRequirements .....	29
6.2.4.1 Description .....	29
6.2.4.2 Properties .....	29
6.2.4.3 Definition .....	30
6.2.4.4 Examples.....	30

6.2.4.5	Additional Requirements .....	30
6.2.5	tosca.datatypes.nfv.ConnectivityType .....	31
6.2.5.1	Description .....	31
6.2.5.2	Properties .....	31
6.2.5.3	Definition .....	31
6.2.5.4	Examples .....	32
6.2.5.5	Additional Requirements .....	32
6.2.6	tosca.datatypes.nfv.RequestedAdditionalCapability .....	32
6.2.6.1	Description .....	32
6.2.6.2	Properties .....	32
6.2.6.3	Definition .....	32
6.2.6.4	Examples .....	33
6.2.6.5	Additional Requirements .....	33
6.2.7	tosca.datatypes.nfv.VirtualMemory .....	33
6.2.7.1	Description .....	33
6.2.7.2	Properties .....	33
6.2.7.3	Definitions .....	34
6.2.7.4	Examples .....	34
6.2.7.5	Additional Requirements .....	34
6.2.8	tosca.datatypes.nfv.VirtualCpu .....	35
6.2.8.1	Description .....	35
6.2.8.2	Properties .....	35
6.2.8.3	Definition .....	35
6.2.8.4	Examples .....	36
6.2.8.5	Additional Requirements .....	36
6.2.9	tosca.datatypes.nfv.VirtualCpuPinning .....	36
6.2.9.1	Description .....	36
6.2.9.2	Properties .....	36
6.2.9.3	Definition .....	37
6.2.9.4	Examples .....	37
6.2.9.5	Additional Requirements .....	37
6.2.10	tosca.datatypes.nfv.VnfcConfigurableProperties .....	37
6.2.10.1	Description .....	37
6.2.10.2	Properties .....	38
6.2.10.3	Definition .....	38
6.2.10.4	Examples .....	38
6.2.10.5	Additional Requirements .....	39
6.2.11	tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties .....	39
6.2.11.1	Description .....	39
6.2.11.2	Properties .....	39
6.2.11.3	Definition .....	39
6.2.12	tosca.datatypes.nfv.VduProfile .....	40
6.2.12.1	Description .....	40
6.2.12.2	Properties .....	40
6.2.12.3	Definition .....	40
6.2.12.4	Examples .....	41
6.2.12.5	Additional requirements .....	42
6.2.13	tosca.datatypes.nfv.VIProfile .....	42
6.2.13.1	Description .....	42
6.2.13.2	Properties .....	42
6.2.13.3	Definition .....	43
6.2.13.4	Examples .....	43
6.2.13.5	Additional Requirements .....	43
6.2.14	tosca.datatypes.nfv.VirtualLinkProtocolData .....	43
6.2.14.1	Description .....	43
6.2.14.2	Properties .....	44
6.2.14.3	Definition .....	44
6.2.14.4	Examples .....	44
6.2.14.5	Additional Requirements .....	44
6.2.15	tosca.datatypes.nfv.L2ProtocolData .....	45
6.2.15.1	Description .....	45
6.2.15.2	Properties .....	45

6.2.15.3	Definition .....	45
6.2.15.4	Examples .....	46
6.2.15.5	Additional Requirements .....	46
6.2.16	tosca.datatypes.nfv.L3ProtocolData .....	46
6.2.16.1	Description .....	46
6.2.16.2	Properties .....	46
6.2.16.3	Definition .....	47
6.2.16.4	Examples .....	48
6.2.16.5	Additional Requirements .....	48
6.2.17	tosca.datatypes.nfv.IpAllocationPool .....	48
6.2.17.1	Description .....	48
6.2.17.2	Properties .....	48
6.2.17.3	Definition .....	48
6.2.17.4	Examples .....	48
6.2.17.5	Additional Requirements .....	49
6.2.18	tosca.datatypes.nfv.InstantiationLevel .....	49
6.2.18.1	Description .....	49
6.2.18.2	Properties .....	49
6.2.18.3	Definition .....	49
6.2.18.4	Examples .....	49
6.2.18.5	Additional Requirements .....	50
6.2.19	tosca.datatypes.nfv.VduLevel .....	50
6.2.19.1	Description .....	50
6.2.19.2	Properties .....	50
6.2.19.3	Definition .....	50
6.2.19.4	Examples .....	50
6.2.19.5	Additional Requirements .....	50
6.2.20	tosca.datatypes.nfv.VnfLcmOperationsConfiguration .....	51
6.2.20.1	Description .....	51
6.2.20.2	Properties .....	51
6.2.20.3	Definition .....	52
6.2.20.4	Examples .....	52
6.2.20.5	Additional Requirements .....	52
6.2.21	tosca.datatypes.nfv.VnfInstantiateOperationConfiguration .....	52
6.2.21.1	Description .....	52
6.2.21.2	Properties .....	53
6.2.21.3	Definition .....	53
6.2.21.4	Examples .....	53
6.2.21.5	Additional Requirements .....	53
6.2.22	tosca.datatypes.nfv.VnfScaleOperationConfiguration .....	53
6.2.22.1	Description .....	53
6.2.22.2	Properties .....	53
6.2.22.3	Definition .....	54
6.2.22.4	Examples .....	54
6.2.22.5	Additional Requirements .....	54
6.2.23	tosca.datatypes.nfv.VnfScaleToLevelOperationConfiguration .....	54
6.2.23.1	Description .....	54
6.2.23.2	Properties .....	54
6.2.23.3	Definition .....	54
6.2.23.4	Examples .....	55
6.2.23.5	Additional Requirements .....	55
6.2.24	tosca.datatypes.nfv.VnfHealOperationConfiguration .....	55
6.2.24.1	Description .....	55
6.2.24.2	Properties .....	55
6.2.24.3	Definition .....	55
6.2.24.4	Examples .....	56
6.2.24.5	Additional Requirements .....	56
6.2.25	tosca.datatypes.nfv.VnfTerminateOperationConfiguration .....	56
6.2.25.1	Description .....	56
6.2.25.2	Properties .....	56
6.2.25.3	Definition .....	56
6.2.25.4	Examples .....	57

6.2.25.5	Additional Requirements .....	57
6.2.26	tosca.datatypes.nfv.VnfOperateOperationConfiguration .....	57
6.2.26.1	Description .....	57
6.2.26.2	Properties .....	57
6.2.26.3	Definition .....	57
6.2.26.4	Examples .....	58
6.2.26.5	Additional Requirements .....	58
6.2.27	tosca.datatypes.nfv.ScaleInfo .....	58
6.2.27.1	Description .....	58
6.2.27.2	Properties .....	58
6.2.27.3	Definition .....	58
6.2.27.4	Examples .....	59
6.2.27.5	Additional Requirements .....	59
6.2.28	tosca.datatypes.nfv.ScalingAspect .....	59
6.2.28.1	Description .....	59
6.2.28.2	Properties .....	59
6.2.28.3	Definition .....	59
6.2.28.4	Examples .....	60
6.2.28.5	Additional Requirements .....	60
6.2.29	tosca.datatypes.nfv.LinkBitrateRequirements .....	60
6.2.29.1	Description .....	60
6.2.29.2	Properties .....	60
6.2.29.3	Definition .....	61
6.2.29.4	Examples .....	61
6.2.29.5	Additional Requirements .....	61
6.2.30	tosca.datatypes.nfv.Qos .....	61
6.2.30.1	Description .....	61
6.2.30.2	Properties .....	61
6.2.30.3	Definition .....	62
6.2.30.4	Examples .....	62
6.2.30.5	Additional Requirements .....	62
6.2.31	tosca.datatypes.nfv.VnfConfigurableProperties .....	62
6.2.31.1	Description .....	62
6.2.31.2	Properties .....	63
6.2.31.3	Definition .....	63
6.2.31.4	Examples .....	64
6.2.31.5	Additional Requirements .....	64
6.2.32	tosca.datatypes.nfv.VnfAdditionalConfigurableProperties .....	64
6.2.32.1	Description .....	64
6.2.32.2	Properties .....	64
6.2.32.3	Definition .....	65
6.2.32.4	Examples .....	65
6.2.32.5	Additional Requirements .....	65
6.2.33	tosca.datatypes.nfv.VnfInfoModifiableAttributes .....	65
6.2.33.1	Description .....	65
6.2.33.2	Properties .....	65
6.2.33.3	Definition .....	65
6.2.33.4	Examples .....	66
6.2.33.5	Additional Requirements .....	66
6.2.34	tosca.datatypes.nfv.VnfInfoModifiableAttributesExtensions .....	66
6.2.34.1	Description .....	66
6.2.34.2	Properties .....	66
6.2.34.3	Definition .....	66
6.2.34.4	Examples .....	66
6.2.34.5	Additional Requirements .....	66
6.2.35	tosca.datatypes.nfv.VnfInfoModifiableAttributesMetadata .....	67
6.2.35.1	Description .....	67
6.2.35.2	Properties .....	67
6.2.35.3	Definition .....	67
6.2.35.4	Examples .....	67
6.2.35.5	Additional Requirements .....	68
6.2.36	tosca.datatypes.nfv.CpProtocolData .....	68

6.2.36.1	Description .....	68
6.2.36.2	Properties .....	68
6.2.36.3	Definition .....	68
6.2.36.4	Examples .....	69
6.2.36.5	Additional Requirements .....	69
6.2.37	tosca.datatypes.nfv.LogicalNodeData .....	69
6.2.37.1	Description .....	69
6.2.37.2	Properties .....	69
6.2.37.3	Definition .....	69
6.2.37.4	Examples .....	70
6.2.37.5	Additional Requirements .....	70
6.2.38	tosca.datatypes.nfv.SwImageData .....	70
6.2.38.1	Description .....	70
6.2.38.2	Properties .....	70
6.2.38.3	Definition .....	71
6.2.38.4	Examples .....	72
6.2.38.5	Additional Requirements .....	72
6.2.39	tosca.datatypes.nfv.VirtualBlockStorageData .....	72
6.2.39.1	Description .....	72
6.2.39.2	Properties .....	73
6.2.39.3	Definition .....	73
6.2.39.4	Examples .....	73
6.2.39.5	Additional Requirements .....	73
6.2.40	tosca.datatypes.nfv.VirtualObjectStorageData .....	74
6.2.40.1	Description .....	74
6.2.40.2	Properties .....	74
6.2.40.3	Definition .....	74
6.2.40.4	Examples .....	74
6.2.40.5	Additional Requirements .....	74
6.2.41	tosca.datatypes.nfv.VirtualFileStorageData .....	74
6.2.41.1	Description .....	74
6.2.41.2	Properties .....	75
6.2.41.3	Definition .....	75
6.2.41.4	Examples .....	75
6.2.41.5	Additional Requirements .....	75
6.2.42	tosca.datatypes.nfv.VirtualLinkBitrateLevel .....	75
6.2.42.1	Description .....	75
6.2.42.2	Properties .....	76
6.2.42.3	Definition .....	76
6.2.42.4	Examples .....	76
6.2.42.5	Additional Requirements .....	76
6.2.43	tosca.datatypes.nfv.VnfOperationAdditionalParameters .....	76
6.2.43.1	Description .....	76
6.2.43.2	Properties .....	77
6.2.43.3	Definition .....	77
6.2.43.4	Examples .....	77
6.2.43.5	Additional Requirements .....	77
6.2.44	tosca.datatypes.nfv.VnfChangeFlavourOperationConfiguration .....	77
6.2.44.1	Description .....	77
6.2.44.2	Properties .....	78
6.2.44.3	Definition .....	78
6.2.44.4	Examples .....	78
6.2.44.5	Additional Requirements .....	78
6.2.45	tosca.datatypes.nfv.VnfChangeExtConnectivityOperation Configuration .....	78
6.2.45.1	Description .....	78
6.2.45.2	Properties .....	78
6.2.45.3	Definition .....	78
6.2.45.4	Examples .....	79
6.2.45.5	Additional Requirements .....	79
6.2.46	tosca.datatypes.nfv.VnfMonitoringParameter .....	79
6.2.46.1	Description .....	79
6.2.46.2	Properties .....	79

6.2.46.3	Definition .....	80
6.2.46.4	Examples .....	80
6.2.46.5	Additional Requirements .....	80
6.2.47	tosca.datatypes.nfv.VnfcMonitoringParameter .....	80
6.2.47.1	Description .....	80
6.2.47.2	Properties .....	80
6.2.47.3	Definition .....	81
6.2.47.4	Examples .....	82
6.2.47.5	Additional Requirements .....	82
6.2.48	tosca.datatypes.nfv.VirtualLinkMonitoringParameter .....	82
6.2.48.1	Description .....	82
6.2.48.2	Properties .....	82
6.2.48.3	Definition .....	82
6.2.48.4	Examples .....	83
6.2.48.5	Additional Requirements .....	83
6.2.49	tosca.datatypes.nfv.InterfaceDetails .....	83
6.2.49.1	Description .....	83
6.2.49.2	Properties .....	83
6.2.49.3	Definition .....	84
6.2.49.4	Examples .....	84
6.2.49.5	Additional Requirements .....	84
6.2.50	tosca.datatypes.nfv.UriComponents .....	84
6.2.50.1	Description .....	84
6.2.50.2	Properties .....	84
6.2.50.3	Definition .....	85
6.2.50.4	Examples .....	85
6.2.50.5	Additional Requirements .....	85
6.2.51	tosca.datatypes.nfv.UriAuthority .....	86
6.2.51.1	Description .....	86
6.2.51.2	Properties .....	86
6.2.51.3	Definition .....	86
6.2.51.4	Examples .....	87
6.2.51.5	Additional Requirements .....	87
6.2.52	tosca.datatypes.nfv.VnfProfile .....	87
6.2.52.1	Description .....	87
6.2.52.2	Properties .....	87
6.2.52.3	Definition .....	88
6.2.52.4	Example .....	88
6.2.52.5	Additional Requirements .....	88
6.2.53	tosca.datatypes.nfv.ChecksumData .....	88
6.2.53.1	Description .....	88
6.2.53.2	Properties .....	89
6.2.53.3	Definition .....	89
6.2.53.4	Examples .....	89
6.2.53.5	Additional Requirements .....	89
6.3	Artifact Types .....	89
6.3.1	tosca.artifacts.nfv.SwImage .....	89
6.3.1.1	Description .....	89
6.3.1.2	Description .....	90
6.3.2	tosca.artifacts.Implementation.nfv.Mistral .....	90
6.3.2.1	Description .....	90
6.3.2.2	Definition .....	90
6.4	Capability Types .....	90
6.4.1	tosca.capabilities.nfv.VirtualBindable .....	90
6.4.1.1	Description .....	90
6.4.1.2	Properties .....	91
6.4.1.3	Definition .....	91
6.4.2	tosca.capabilities.nfv.VirtualLinkable .....	91
6.4.2.1	Description .....	91
6.4.2.2	Properties .....	91
6.4.2.3	Definition .....	91
6.4.3	tosca.capabilities.nfv.VirtualCompute .....	91



6.4.3.1	Description .....	91
6.4.3.2	Properties .....	92
6.4.3.3	Definition .....	92
6.4.4	tosca.capabilities.nfv.VirtualStorage .....	93
6.4.4.1	Description .....	93
6.4.4.2	Definition .....	93
6.5	Requirements Types .....	93
6.6	Relationship Types .....	93
6.6.1	tosca.relationships.nfv.VirtualBindsTo .....	93
6.6.1.1	Description .....	93
6.6.1.2	Properties .....	94
6.6.1.3	Definition .....	94
6.6.2	tosca.relationships.nfv.VirtualLinksTo .....	94
6.6.2.1	Description .....	94
6.6.2.2	Properties .....	94
6.6.2.3	Definition .....	94
6.6.3	tosca.relationships.nfv.AttachesTo .....	94
6.6.3.1	Description .....	94
6.6.3.2	Properties .....	95
6.6.3.3	Definition .....	95
6.7	Interface Types .....	95
6.7.1	tosca.interfaces.nfv.Vnflcm .....	95
6.7.1.1	Description .....	95
6.7.1.2	Definition .....	96
6.7.1.3	Additional Requirements .....	98
6.7.1.4	Support of LCM scripts .....	98
6.7.1.5	Examples .....	99
6.8	Node Types .....	101
6.8.1	tosca.nodes.nfv.VNF .....	101
6.8.1.1	Description .....	101
6.8.1.2	Properties .....	101
6.8.1.3	Attributes .....	103
6.8.1.4	Requirements .....	103
6.8.1.5	Capabilities .....	103
6.8.1.6	Definition .....	104
6.8.1.7	Artifact .....	105
6.8.1.8	Additional Requirements .....	105
6.8.1.9	Example .....	106
6.8.2	tosca.nodes.nfv.VnfExtCp .....	108
6.8.2.1	Description .....	108
6.8.2.2	Properties .....	108
6.8.2.3	Attributes .....	108
6.8.2.4	Requirements .....	109
6.8.2.5	Definition .....	109
6.8.2.6	Additional Requirements .....	109
6.8.2.7	Example .....	109
6.8.3	tosca.nodes.nfv.Vdu.Compute .....	110
6.8.3.1	Description .....	110
6.8.3.2	Properties .....	110
6.8.3.3	Attributes .....	111
6.8.3.4	Requirements .....	111
6.8.3.5	Capabilities .....	111
6.8.3.6	Definition .....	112
6.8.3.7	Additional requirements .....	113
6.8.3.8	Example .....	113
6.8.4	tosca.nodes.nfv.Vdu.VirtualBlockStorage .....	116
6.8.4.1	Description .....	116
6.8.4.2	Properties .....	116
6.8.4.3	Attributes .....	116
6.8.4.4	Requirements .....	116
6.8.4.5	Capabilities .....	116
6.8.4.6	Definition .....	116

6.8.4.7	Additional requirements .....	117
6.8.5	tosca.nodes.nfv.Vdu.VirtualObjectStorage .....	117
6.8.5.1	Description .....	117
6.8.5.2	Properties .....	117
6.8.5.3	Attributes .....	117
6.8.5.4	Requirements .....	117
6.8.5.5	Capabilities .....	117
6.8.5.6	Definition .....	118
6.8.5.7	Additional requirements .....	118
6.8.6	tosca.nodes.nfv.Vdu.VirtualFileStorage .....	118
6.8.6.1	Description .....	118
6.8.6.2	Properties .....	118
6.8.6.3	Attributes .....	118
6.8.6.4	Requirements .....	119
6.8.6.5	Capabilities .....	119
6.8.6.6	Definition .....	119
6.8.6.7	Additional requirements .....	119
6.8.7	tosca.nodes.nfv.Cp .....	119
6.8.7.1	Description .....	119
6.8.7.2	Properties .....	120
6.8.7.3	Attributes .....	120
6.8.7.4	Requirements .....	120
6.8.7.5	Capabilities .....	120
6.8.7.6	Definition .....	121
6.8.7.7	Additional requirements .....	121
6.8.8	tosca.nodes.nfv.VduCp .....	121
6.8.8.1	Description .....	121
6.8.8.2	Properties .....	122
6.8.8.3	Attributes .....	122
6.8.8.4	Requirements .....	122
6.8.8.5	Definition .....	123
6.8.9	tosca.nodes.nfv.VnfVirtualLink .....	123
6.8.9.1	Description .....	123
6.8.9.2	Properties .....	124
6.8.9.3	Requirements .....	124
6.8.9.4	Capabilities .....	124
6.8.9.5	Definition .....	124
6.9	Group Types .....	125
6.9.1	tosca.groups.nfv.PlacementGroup .....	125
6.9.1.1	Description .....	125
6.9.1.2	Properties .....	125
6.9.1.3	Definition .....	125
6.9.1.4	Additional Requirements .....	126
6.9.1.5	Examples .....	126
6.10	Policy Types .....	126
6.10.1	tosca.policies.nfv.InstantiationLevels .....	126
6.10.1.1	Description .....	126
6.10.1.2	Properties .....	126
6.10.1.3	Definition .....	127
6.10.2	tosca.policies.nfv.VduInstantiationLevels .....	127
6.10.2.1	Description .....	127
6.10.2.2	Properties .....	127
6.10.2.3	Definition .....	128
6.10.2.4	Additional Requirements .....	128
6.10.3	tosca.policies.nfv.VirtualLinkInstantiationLevels .....	128
6.10.3.1	Description .....	128
6.10.3.2	Properties .....	128
6.10.3.3	Definition .....	129
6.10.3.4	Additional Requirements .....	129
6.10.4	Void .....	129
6.10.5	tosca.policies.nfv.ScopingAspects .....	129
6.10.5.1	Description .....	129

6.10.5.2	Properties .....	129
6.10.5.3	Definition .....	130
6.10.5.4	Examples .....	130
6.10.6	tosca.policies.nfv.VduScalingAspectDeltas .....	130
6.10.6.1	Description .....	130
6.10.6.2	Properties .....	130
6.10.6.3	Definition .....	131
6.10.6.4	Examples .....	131
6.10.7	tosca.policies.nfv.VirtualLinkBitrateScalingAspectDeltas .....	131
6.10.7.1	Description .....	131
6.10.7.2	Properties .....	131
6.10.7.3	Definition .....	132
6.10.7.4	Examples .....	132
6.10.8	tosca.policies.nfv.VduInitialDelta .....	132
6.10.8.1	Description .....	132
6.10.8.2	Properties .....	132
6.10.8.3	Definition .....	133
6.10.8.4	Examples .....	133
6.10.9	tosca.policies.nfv.VirtualLinkBitrateInitialDelta .....	133
6.10.9.1	Description .....	133
6.10.9.2	Properties .....	133
6.10.9.3	Definition .....	134
6.10.9.4	Examples .....	134
6.10.10	AffinityRule, AntiAffinityRule .....	134
6.10.10.1	Description .....	134
6.10.10.2	Properties .....	134
6.10.10.3	targets .....	135
6.10.10.4	Definition .....	135
6.10.10.5	Examples .....	136
6.10.11	tosca.policies.nfv.SecurityGroupRule .....	136
6.10.11.1	Description .....	136
6.10.11.2	Properties .....	137
6.10.11.3	Definition .....	137
6.10.11.4	Additional Requirements .....	139
6.10.12	tosca.policies.nfv.SupportedVnfInterface .....	139
6.10.12.1	Description .....	139
6.10.12.2	Properties .....	139
6.10.12.3	Definition .....	139
6.10.12.4	Additional requirements .....	140
6.10.12.5	Example .....	140
6.11	VNFD TOSCA service template design .....	140
6.11.1	General .....	140
6.11.2	Single or multiple deployment flavour design with TOSCA-Simple-Profile-YAML-v1.2 .....	140
6.11.3	Single deployment flavour design with TOSCA-Simple-Profile-YAML-v1.1 .....	141
7	NSD TOSCA model .....	142
7.1	Introduction .....	142
7.2	Data Types .....	143
7.2.1	tosca.datatype.nfv.ServiceAvailability .....	143
7.2.1.1	Description .....	143
7.2.1.2	Properties .....	143
7.2.1.3	Definition .....	143
7.2.1.4	Examples .....	143
7.2.2	tosca.datatypes.nfv.VnfProfile .....	143
7.2.2.1	Description .....	143
7.2.3	tosca.datatype.nfv.NsVIPProfile .....	144
7.2.3.1	Description .....	144
7.2.3.2	Properties .....	144
7.2.3.3	Definition .....	144
7.2.3.4	Examples .....	145
7.2.3.5	Additional Requirements .....	145
7.2.4	tosca.datatypes.nfv.ConnectivityType .....	145

7.2.4.1	Description .....	145
7.2.5	tosca.datatypes.nfv.NsVirtualLinkQos .....	145
7.2.5.1	Description .....	145
7.2.5.2	Properties .....	145
7.2.5.3	Definition .....	145
7.2.5.4	Examples .....	146
7.2.5.5	Additional Requirements .....	146
7.2.6	tosca.datatypes.nfv.LinkBitrateRequirements .....	146
7.2.6.1	Description .....	146
7.2.7	Void .....	146
7.2.8	Void .....	146
7.2.9	Void .....	146
7.2.10	Void .....	146
7.2.11	tosca.datatypes.nfv.CpProtocolData .....	146
7.2.11.1	Description .....	146
7.2.12	tosca.datatypes.nfv.AddressData .....	146
7.2.12.1	Description .....	146
7.2.13	tosca.datatypes.nfv.L2AddressData .....	146
7.2.13.1	Description .....	146
7.2.14	tosca.datatypes.nfv.L3AddressData .....	146
7.2.14.1	Description .....	146
7.2.15	tosca.datatypes.nfv.Qos .....	147
7.2.15.1	Description .....	147
7.3	Artifact Types .....	148
7.4	Capabilities Types .....	148
7.4.1	tosca.capabilities.nfv.VirtualLinkable .....	148
7.4.1.1	Description .....	148
7.5	Requirements Types .....	148
7.6	Relationship Types .....	148
7.6.1	tosca.relationships.nfv.VirtualLinksTo .....	148
7.6.1.1	Description .....	148
7.7	Interface Types .....	149
7.7.1	tosca.interfaces.nfv.Nslcm .....	149
7.7.1.1	Description .....	149
7.7.1.2	Definition .....	149
7.7.1.3	Additional Requirements .....	150
7.7.1.4	Support of LCM scripts .....	150
7.7.1.5	Examples .....	151
7.8	Node Types .....	151
7.8.1	tosca.nodes.nfv.NS .....	151
7.8.1.1	Description .....	151
7.8.1.2	Properties .....	152
7.8.1.3	Attributes .....	152
7.8.1.4	Requirements .....	152
7.8.1.5	Capabilities .....	152
7.8.1.6	Definition .....	153
7.8.1.7	Artifact .....	153
7.8.1.8	Additional requirements .....	153
7.8.2	tosca.nodes.nfv.Sap .....	154
7.8.2.1	Description .....	154
7.8.2.2	Properties .....	154
7.8.2.3	Attributes .....	154
7.8.2.4	Requirements .....	154
7.8.2.5	Capabilities .....	154
7.8.2.6	Definition .....	154
7.8.2.7	Additional requirements .....	155
7.8.2.8	Example .....	155
7.8.3	tosca.nodes.nfv.NsVirtualLink .....	155
7.8.3.1	Description .....	155
7.8.3.2	Properties .....	155
7.8.3.3	Attributes .....	156
7.8.3.4	Requirements .....	156

7.8.3.5	Capabilities.....	156
7.8.3.6	Definition .....	156
7.8.3.7	Artifact .....	157
7.8.3.8	Additional Requirements .....	157
7.8.3.9	Example .....	157
7.8.4	tosca.nodes.nfv.Cp.....	157
7.8.4.1	Description .....	157
7.9	Group Types.....	157
7.9.1	tosca.groups.nfv.PlacementGroup .....	157
7.9.1.1	Description .....	157
7.10	Policy Types.....	157
7.10.1	AffinityRule, AntiAffinityRule.....	157
7.10.1.1	Description .....	157
7.10.1.2	Targets.....	158
7.10.1.3	Examples.....	158
7.10.2	tosca.policies.nfv.SecurityGroupRule.....	159
7.10.2.1	Description .....	159
7.10.2.2	Additional Requirements .....	159
7.11	NSD TOSCA service template design.....	159
7.11.1	General.....	159
7.11.2	Single or multiple deployment flavour design with TOSCA-Simple-Profile-YAML-v1.2.....	159
7.11.3	Single deployment flavour design with TOSCA-Simple-Profile-YAML-v1.1 .....	160
8	PNFD TOSCA model.....	161
8.1	Introduction .....	161
8.2	Data Types.....	162
8.2.1	tosca.datatypes.nfv.CpProtocolData .....	162
8.2.1.1	Description .....	162
8.2.2	tosca.datatypes.nfv.AddressData .....	162
8.2.2.1	Description .....	162
8.2.3	tosca.datatypes.nfv.L2AddressData.....	162
8.2.3.1	Description .....	162
8.2.4	tosca.datatypes.nfv.L3AddressData.....	162
8.2.4.1	Description .....	162
8.2.5	tosca.datatypes.nfv.LocationInfo .....	162
8.2.5.1	Description .....	162
8.2.5.2	Properties .....	162
8.2.5.3	Definition .....	163
8.2.5.4	Examples.....	163
8.2.5.5	Additional Requirements .....	163
8.2.6	tosca.datatypes.nfv.CivicAddressElement .....	163
8.2.6.1	Description .....	163
8.2.6.2	Properties .....	163
8.2.6.3	Definition .....	164
8.2.6.4	Examples.....	164
8.2.6.5	Additional Requirements .....	164
8.3	Artifact Types.....	164
8.4	Capabilities Types .....	164
8.4.1	tosca.capabilities.nfv.VirtualLinkable .....	164
8.4.1.1	Description .....	164
8.5	Requirements Types .....	164
8.6	Relationship Types .....	165
8.6.1	tosca.relationships.nfv.VirtualLinksTo.....	165
8.6.1.1	Description .....	165
8.7	Interface Types .....	165
8.8	Node Types .....	165
8.8.1	tosca.nodes.nfv.PNF .....	165
8.8.1.1	Description .....	165
8.8.1.2	Properties .....	165
8.8.1.3	Attributes.....	166
8.8.1.4	Requirements .....	166
8.8.1.5	Capabilities.....	166

8.8.1.6	Definition .....	166
8.8.1.7	Artifact .....	166
8.8.1.8	Additional Requirements .....	167
8.8.1.9	Example .....	167
8.8.2	tosca.nodes.nfv.PnfExtCp.....	167
8.8.2.1	Description .....	167
8.8.2.2	Properties .....	167
8.8.2.3	Attributes.....	167
8.8.2.4	Requirements .....	167
8.8.2.5	Capabilities.....	167
8.8.2.6	Definition .....	168
8.8.3	tosca.nodes.nfv.Cp.....	168
8.8.3.1	Description .....	168
8.9	Group Types.....	168
8.10	Policy Types.....	168
8.10.1	tosca.policies.nfv.SecurityGroupRule.....	168
8.10.1.1	Description .....	168
8.10.1.2	Additional Requirements .....	168
8.11	PNFD TOSCA service template design .....	168
8.11.1	General.....	168
<b>Annex A (informative): Examples.....</b>		<b>170</b>
A.1	Deployment flavour design mapping .....	170
A.1.1	Introduction .....	170
A.1.2	Design principle .....	170
A.2	VNFD with deployment flavour modelling design example.....	171
A.3	VNF external connection point .....	179
A.3.1	General .....	179
A.3.2	External connection point re-exposing an internal connection point.....	180
A.3.3	External connection point connected to an internal virtual link .....	181
A.4	VNFD modelling design example by using TOSCA composition.....	182
A.5	VNFD with Single deployment flavour modelling design example .....	185
A.6	Scaling and Instantiation Level examples .....	189
A.6.1	ScalingAspect and InstantiationLevels policies with uniform delta.....	189
A.6.2	ScalingAspect and InstantiationLevels policies with non-uniform deltas .....	194
A.7	Service Access Point .....	198
A.7.1	General .....	198
A.7.2	VNF External connection point exposing as a SAP .....	198
A.7.3	SAP connected to an NS virtual link .....	200
A.8	NSD with Single deployment flavour modelling design example .....	201
A.9	Mapping between NFV IM and TOSCA concepts.....	203
A.9.1	Introduction .....	203
A.9.2	Mapping between ETSI GS NFV-IFA 011 IM and TOSCA concepts.....	203
A.9.3	Mapping between ETSI GS NFV-IFA 014 IM and TOSCA concepts.....	204
A.10	PNFD modelling design example.....	205
A.11	NSD with Multiple deployment flavour modelling design example.....	207
A.12	NSD TOSCA service template design .....	213
<b>Annex B (normative): etsi_nfv_sol001_type definitions .....</b>		<b>218</b>
B.1	Purpose.....	218
B.2	VNFD type definitions file.....	218
B.3	NSD type definitions file.....	218

B.4	PNFD type definitions file .....	219
<b>Annex C (normative):</b>	<b>Conformance .....</b>	<b>220</b>
C.1	Purpose .....	220
C.2	NFV TOSCA YAML service template .....	220
C.3	NFV TOSCA processor.....	221
<b>Annex D (informative):</b>	<b>Mapping between properties of TOSCA types and API attributes.....</b>	<b>222</b>
D.1	Introduction .....	222
D.2	VNFD-related constructs.....	222
<b>Annex E (informative):</b>	<b>Authors &amp; contributors.....</b>	<b>229</b>
<b>Annex F (informative):</b>	<b>Change History .....</b>	<b>230</b>
History .....		232

---

## Intellectual Property Rights

### Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (<https://ipr.etsi.org>).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

### Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

---

## Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification Group (ISG) Network Functions Virtualisation (NFV).

---

## Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the [ETSI Drafting Rules](#) (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.



---

# 1 Scope

The present document specifies a data model for NFV descriptors, using the TOSCA Simple Profile in YAML [3], fulfilling the requirements specified in ETSI GS NFV-IFA 011 [1] and ETSI GS NFV-IFA 014 [2] for a Virtualised Network Function Descriptor (VNFD), a Network Service Descriptor (NSD) and a Physical Network Function Descriptor (PNFD). The present document also specifies requirements on the VNFM and NFVO specific to the handling of NFV descriptors based on the TOSCA Simple Profile in YAML specification [3].

---

## 2 References

### 2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at <https://docbox.etsi.org/Reference/>.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

- [1] ETSI GS NFV-IFA 011: "Network Functions Virtualisation (NFV) Release 2; Management and Orchestration; VNF Descriptor and Packaging Specification".
- [2] ETSI GS NFV-IFA 014: "Network Functions Virtualisation (NFV) Release 2; Management and Orchestration; Network Service Templates Specification".
- [3] TOSCA-Simple-Profile-yaml-v1.2: "TOSCA Simple Profile in YAML Version 1.2".
- [4] TOSCA-Simple-Profile-yaml-v1.1: "TOSCA Simple Profile in YAML Version 1.1".
- [5] Private Enterprise Numbers registry at IANA.

NOTE: Available at <https://www.iana.org/assignments/enterprise-numbers/enterprise-numbers>.

- [6] IETF RFC 5234: "Augmented BNF for Syntax Specifications: ABNF".
- [7] ETSI GS NFV-IFA 027: "Network Functions Virtualisation (NFV) Release 2; Management and Orchestration; Performance Measurements Specification".
- [8] IETF RFC 3986: "Uniform Resource Identifier (URI): Generic Syntax".
- [9] IETF RFC 4122: "A Universally Unique Identifier (UUID) URN Namespace".
- [10] ISO 3166 (all parts): "Codes for the representation of names of countries and their subdivisions".
- [11] IETF RFC 4776: "Dynamic Host Configuration Protocol (DHCPv4 and DHCPv6) Option for Civic Addresses Configuration Information".

NOTE: Available at <https://tools.ietf.org/html/rfc4776>.

- [12] IETF RFC 7230: "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing".
- [13] IETF RFC 5646: "Tags for Identifying Languages".
- [14] Hash Function Textual Names registry at IANA.

NOTE: Available at <https://www.iana.org/assignments/hash-function-text-names>.

- [15] The Open Group Base Specifications Issue 7, 2018 edition IEEE™ Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008).

NOTE: Available at [http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1\\_chap09.html](http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap09.html).

## 2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

- [i.1] ETSI GS NFV-IFA 007: "Network Functions Virtualisation (NFV); Management and Orchestration; Or-Vnfm reference point - Interface and Information Model Specification".
- [i.2] ETSI GS NFV 003: "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV".
- [i.3] ETSI GS NFV-REL 001: "Network Functions Virtualisation (NFV); Resiliency Requirements".
- [i.4] ETSI GS NFV-SOL 002: "Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; RESTful protocols specification for the Ve-Vnfm Reference Point".
- [i.5] Assigned Internet Protocol Numbers.

NOTE: Available at <https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>.

- [i.6] ETSI GS NFV-SOL 004: "Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; VNF Package specification".
- [i.7] Mistral Workflow Language v2 specification.

NOTE: Available at [https://docs.openstack.org/mistral/latest/user/wf\\_lang\\_v2.html](https://docs.openstack.org/mistral/latest/user/wf_lang_v2.html).

- [i.8] ETSI GS NFV-IFA 013: "Network Functions Virtualisation (NFV); Management and Orchestration; Os-Ma-Nfvo reference point - Interface and Information Model Specification".
- [i.9] ETSI GS NFV-SOL 003: "Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; RESTful protocols specification for the Or-Vnfm Reference Point".
- [i.10] ETSI GS NFV-SOL 005: "Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; RESTful protocols specification for the Os-Ma-nfvo Reference Point".
- [i.11] ETSI GS NFV-SOL 007: "Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; Network Service Descriptor File Structure Specification".

---

## 3 Definition of terms, symbols and abbreviations

### 3.1 Terms

For the purposes of the present document, the terms given in ETSI GS NFV 003 [i.2] and the following apply:

**TOSCA interface type:** reusable entity that describes a set of TOSCA operations that can be included as part of a Node type or Relationship Type definition

NOTE: See TOSCA-Simple-Profile-YAML-v1.2 [3].

**TOSCA operation:** behavioural lifecycle procedure in a TOSCA node or relationship definition that can be invoked by an orchestration engine, whose implementation definition can be provided in the service template as part of a node template definition or a relationship template definition, or rely on an implementation of the operation built in the orchestration engine

## 3.2 Symbols

Void.

## 3.3 Abbreviations

For the purposes of the present document, the abbreviations given in ETSI GS NFV 003 [i.2] apply.

---

# 4 Overview of TOSCA model

TOSCA (Topology and Orchestration Specification for Cloud Applications) is a modelling language for describing the components of a cloud application and their relationships. TOSCA uses the concept of service templates to describe cloud workloads. TOSCA further provides means of associating standard or user-defined lifecycle operations to a cloud application component or to a relationship between components. The present document is based on TOSCA-Simple-Profile-yaml v1.2 [3], which describes a YAML rendering for TOSCA.

---

# 5 General concept of using TOSCA to model NFV descriptors

## 5.1 Introduction

An NFV deployment template is modelled by using one or more TOSCA service template as defined in TOSCA-Simple-Profile-yaml-v1.2 [3].

Three main deployment templates are identified in the present document:

- The Virtualised Network Function Descriptor (VNFD).
- The Network Service Descriptor (NSD).
- The Physical Network Function Descriptor (PNFD).

## 5.2 Network Service Descriptor

The Network Service Descriptor (NSD) is a deployment template which consists of information used by the NFVO for lifecycle management of an NS as defined in ETSI GS NFV-IFA 014 [2]. The NSD:

- References zero, one or more Virtualised Network Function Descriptors (VNFD).
- References zero, one or more Physical Network Functions Descriptors (PNFD).
- References zero, one or more nested NSD.
- Includes zero, one or more Virtual Link Descriptors (VLD).
- Includes zero, one or more VNF Forwarding Graph Descriptors (VNFFGD).

A VNFFGD describes a topology of the Network Service or a portion of the Network Service.

A VLD describes the resource for deploying and managing the lifecycle of virtual links between the constituents of an NS.

A PNFD describes the connectivity requirements to integrate PNFs in an NS.

A nested NSD is an NSD from which a nested NS can be instantiated within a parent NS instance.

## 5.3 Virtualised Network Function Descriptor

The VNFD is a component of a VNF package. It is used by both the NFVO and the VNFM.

A VNFD is a deployment template which describes a VNF in terms of deployment and operational behaviour requirements. It also contains Virtualised Deployment Units (VDUs), internal virtual link descriptors, external connection point descriptors, software image descriptors, and deployment flavour descriptors, as defined in ETSI GS NFV-IFA 011 [1].

A VNFD contains the following main pieces of information, as shown in figure 5.3-1:

- Virtualisation Deployment Unit (VDU) is a construct supporting the description of the deployment and operational behaviour of a VNF Component (VNFC). A VNFC instance created based on the VDU maps to a single virtualisation container (e.g. a VM). A VDU describes the resources needed to deploy and manage the lifecycle of a VNFC. A VDU includes internal Connection Point Descriptors (CPDs) that describe internal connection points that can either be used to connect a VNFC to an internal virtual link or be re-exposed outside the VNF as external connection points.
- External CPD: describes an external connection point of a VNF, where either an internal connection point of a VDU is exposed as external connection point or the external connection point is directly connected to an internal virtual link.
- Internal VLD: describes the resource requirements for deploying and managing the lifecycle of virtual links between one or more VNFC instances created based on one or more VDUs.

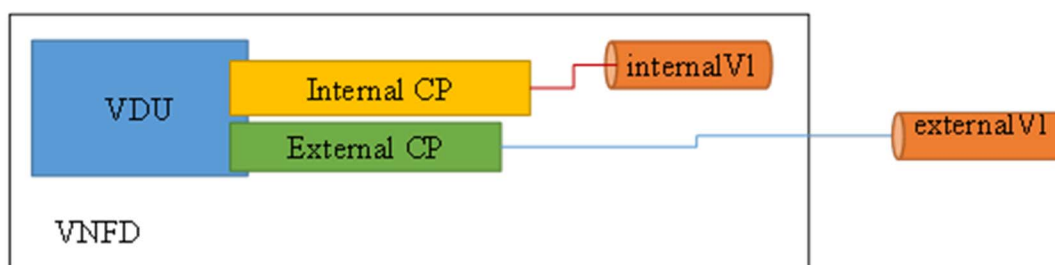


Figure 5.3-1: Overview of VNF descriptor

The information within a VNFD is structured according to one or more VNF deployment flavours (VnfDf) that specify different deployment configuration of a VNF, in terms of its internal topology and resource needs.

## 5.4 Physical Network Function Descriptor

The Physical Network Function Descriptor (PNFD) information element is a deployment template enabling on-boarding PNFs and referencing them from an NSD. It focuses on connectivity aspects only.

## 5.5 `tosca_definitions_version` and Namespace prefix

The "tosca\_definitions\_version" keyword when used in the present document shall comply with the definition as specified in section 3.1.2 of TOSCA-Simple-Profile-YAML-v1.2 [3] with the associated Namespace Alias value defined in the TOSCA-Simple-Profile-YAML-v1.2 specification [3].

Table 5.5-1 defines the TOSCA Namespace prefix that shall be used to declare the namespace of all the TOSCA types as specified in the present document.

**Table 5.5-1**

Namespace Prefix	Specification Description
toscanfv	The TOSCA namespace prefix of all the TOSCA types as specified in the present document.

## 5.6 Imports statement

### 5.6.1 VNFD TOSCA service template

A VNFD TOSCA service template as specified in clause 6.11 shall include a TOSCA import definition referencing the following files:

- The file defined in clause B.2 that includes all the type definitions from clause 6 of the present document.
- Others, as described in clause 6.11.

As specified in [3], the import statement can use a single-line or multi-line grammar.

The single-line grammar supports the import of single or multiple uniquely named VNFD types and other template definition files.

EXAMPLE 1:

```
imports:
- https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi_nfv_sol001_vnfd_types.yaml
- any_other_files.yaml
- custom_vnfd_datatypes_extension.yaml
```

The multi-line grammar also supports the import of single or multiple uniquely named VNFD types and other template definition files. The "file" keyword is a mandatory parameter in this grammar.

EXAMPLE 2:

```
imports:
- file:
https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi_nfv_sol001_vnfd_types.yaml
- file: any_other_files.yaml
- file: custom_vnfd_datatypes_extension.yaml
```

### 5.6.2 NSD TOSCA service template

A NSD TOSCA service template as specified in clause 7.11 shall include a TOSCA import definition referencing the following files:

- The file defined in clause B.3 that includes all the type definitions from clause 7 of the present document.
- Others, as described in clause 7.11.

As specified in [3], the import statement can use a single-line or multi-line grammar.

The single-line grammar supports the import of single or multiple uniquely named NSD types and other template definition files.

EXAMPLE 1:

```
imports:
- https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi_nfv_sol001_nsd_types.yaml
- any_other_files.yaml
- custom_nsd_node_types_extension.yaml
```

The multi-line grammar also supports the import of single or multiple uniquely named NSD types and other template definition files. The "file" keyword is a mandatory parameter in this grammar.

EXAMPLE 2:

```
imports:
- file:
https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi_nfv_sol001_nsd_types.yaml
- file: any_other_files.yaml
- file: custom_nsd_node_types_extension.yaml
```

### 5.6.3 PNFD TOSCA service template

A PNFD TOSCA service template as specified in clause 8.11 shall include a TOSCA import definition referencing the following files:

- The file defined in clause B.4 that includes all the type definitions from clause 8 of the present document.
- Others, as described in clause 8.11.

As specified in [3], the import statement can use a single-line or multi-line grammar.

The single-line grammar supports the import of single or multiple uniquely named PNFD types and other template definition files.

EXAMPLE 1:

```
imports:
- https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi_nfv_sol001_pnfd_types.yaml
- any_other_files.yaml
- custom_pnfd_node_types_extension.yaml
```

The multi-line grammar also supports the import of single or multiple uniquely named PNFD types and other template definition files. The "file" keyword is a mandatory parameter in this grammar.

EXAMPLE 2:

```
imports:
- file:
https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi_nfv_sol001_pnfd_types.yaml
- file: any_other_files.yaml
- file: custom_pnfd_node_types_extension.yaml
```

## 5.7 Type extension

### 5.7.1 Introduction

Type extension is used when VNF-specific type information is introduced in the VNFD (e.g. modifiable attributes, configurable properties and additional parameters to LCM operations) or NSD.

### 5.7.2 Rules

Type extension may be applied to NFV types defined in the present document within the limits specified table 5.7.3-1, adhering to the following rule.

A derived type shall extend the base type in such a way that it remains substitutable for the base type with the following requirements.

New properties may be introduced with no restriction within the limits specified table 5.7.3-1.

Existing properties may be extended according to the following rules:

- a) A scalar property may not be extended to another type (e.g. a string property may not be replaced with an integer property or with a complex property of the same name).
- b) A complex property of data type "X" may only be extended to a property of type "Y" where "Y" is derived from "X" according to the present rules (recursive rule: present rules applied to each property of the derived data type).
- c) A property of type list with entry schema "X" may only be extended to a list with entry schema "Y" where "Y" is an extension of "X" according to the present rules (recursive rule: present rules applied to the elements of the list).
- d) A property of type map with entry schema "X" may only be extended to a map with entry schema "Y" where "Y" is an extension of "X" according to the present rules (recursive rule: present rules applied to the values of the map).

In general, the same rules (1 and 2) apply to introducing/extending other elements beyond properties such as capabilities, requirements, interfaces, operations, inputs, etc. as well; in the current version of the specification, however, there are only examples of extending properties and the `tosca.nodes.nfv.VNF` node type.

### 5.7.3 VNFD Types

Table 5.7.3-1 specifies the extension point where VNFD author may extend the pre-defined types.

**Table 5.7.3-1: VNFD type extension points**

Type	Keyname	Property name
tosca.nodes.nfv.VNF	properties	modifiable_attributes (as a new property) configurable_properties (as a new property)
	interfaces	Vnflcm.{operation_name}.inputs.additional_parameters (as a new property) Vnflcm.{operation_name}.inputs (as new properties)
tosca.nodes.nfv.Vdu.Compute	properties	configurable_properties (as a new property)
tosca.datatypes.nfv.VnfInfoModifiableAttributes	properties	extensions (as a new property) metadata (as a new property)
tosca.datatypes.nfv.VnfConfigurableProperties	properties	additional_configurable_properties (as a new property)
tosca.datatypes.nfv.VnfcConfigurableProperties	properties	additional_vnfc_configurable_properties (as a new property)
tosca.datatypes.nfv.VnfInfoModifiableAttributesExtensions	properties	(new properties)
tosca.datatypes.nfv.VnfInfoModifiableAttributesMetadata	properties	(new properties)
tosca.datatypes.nfv.VnfAdditionalConfigurableProperties	properties	(new properties)
tosca.datatypes.nfv.VnfAdditionalVnfcConfigurableProperties	properties	(new properties)
tosca.datatypes.nfv.VnfOperationAdditionalParameters	properties	(new properties)

## 5.7.4 NSD types

None.

# 6 VNFD TOSCA model

## 6.1 Introduction

The VNFD information model specified by ETSI GS NFV-IFA 011 [1] is mapped to the TOSCA concepts. The VNFD is represented as one or more TOSCA service templates to be used by the VNFM for deploying and managing the lifecycle of a VNF instance.

Table 6.1-1 describes the mapping of the main information elements defined in ETSI GS NFV-IFA 011 [1] applicable to a VNFD and the corresponding NFV-specific TOSCA Types, as well the basic TOSCA types defined in TOSCA Simple YAML Profile [3] from which they are derived from. The full definition of all types can be found in the following clauses.

NOTE: There is only part of VNFD features as specified in ETSI GS NFV-IFA 011 [1] is included in this version of the present document, the following are features not included:

- VnfIndicators
- autoScale

**Table 6.1-1: Mapping of ETSI GS NFV-IFA 011 [1] information elements with TOSCA types**

ETSI GS NFV-IFA 011 [1] Elements	VNFD TOSCA types	Derived from
VNFD	tosca.nodes.nfv.VNF	tosca.nodes.Root
Vdu	n/a (see note 1)	n/a
Cpd (Connection Point)	tosca.nodes.nfv.Cp	tosca.nodes.Root
VduCpd (internal connection point)	tosca.nodes.nfv.VduCp	tosca.nodes.nfv.Cp
VnfVirtualLinkDesc (Virtual Link)	tosca.nodes.nfv.VnfVirtualLink	tosca.nodes.Root
VnfExtCpd (External Connection Point)	tosca.nodes.nfv.VnfExtCp tosca.nodes.nfv.VduCp	tosca.nodes.nfv.Cp
VirtualStorageDesc	tosca.nodes.nfv.Vdu.VirtualBlockStorage tosca.nodes.nfv.Vdu.VirtualObjectStorage tosca.nodes.nfv.Vdu.VirtualFileStorage	tosca.nodes.Root
VirtualComputeDesc	tosca.nodes.nfv.Vdu.Compute	tosca.nodes.Root
SwImageDesc	tosca.artifacts.nfv.SwImage	tosca.artifacts.Deployment .Image
VnfDf	n/a (see note 2)	n/a
SecurityGroupRule	tosca.policies.nfv.SecurityGroupRule	tosca.policies.Root
VnfConfigurableProperties	tosca.datatypes.nfv.VnfConfigurableProperties	tosca.datatypes.Root
VnfInfoModifiableAttributes	tosca.datatypes.nfv.VnfInfoModifiableAttributes	tosca.datatypes.Root
NOTE 1: The Vdu information element is represented as a collection of tosca.nodes.nfv.VduCp, tosca.nodes.nfv.Vdu.Compute, tosca.nodes.nfv.Vdu.VirtualBlockStorage, tosca.nodes.nfv.Vdu.VirtualObjectStorage and tosca.nodes.nfv.Vdu.VirtualFileStorage types.		
NOTE 2: The VnfDf information element is represented as a TOSCA service template.		

Figure 6.1-1 provides an overview of the TOSCA node types used to build a service template representing a VNFD for a specific deployment flavour, and of the relationship between them. The figure shows one of the three types of virtual storage. A detailed description is provided in clause 6.11.



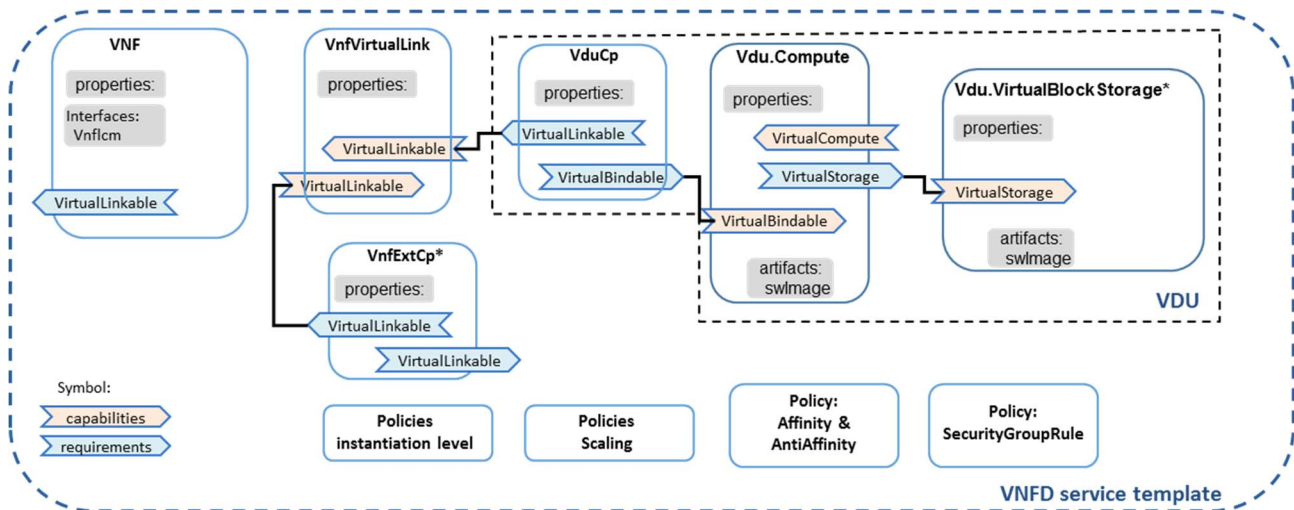


Figure 6.1-1: Service template VNFD overview

## 6.2 Data Types

### 6.2.1 toasca.datatypes.nfv.L2AddressData

#### 6.2.1.1 Description

The L2AddressData data type describes the information on the MAC addresses to be assigned to a connection point as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.1.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

Table 6.2.1.1-1: Type name, shorthand, and URI

<b>Shorthand Name</b>	L2AddressData
<b>Type Qualified Name</b>	toscanfv:L2AddressData
<b>Type URI</b>	tosca.datatypes.nfv.L2AddressData

#### 6.2.1.2 Properties

The properties of the L2AddressData data type shall comply with the provisions set out in table 6.2.1.2-1.

Table 6.2.1.2-1: Properties

Name	Required	Type	Constraints	Description
mac_address_assignment	yes	boolean		Specify if the address assignment is the responsibility of management and orchestration function or not. If it is set to True, it is the management and orchestration function responsibility.

#### 6.2.1.3 Definition

The syntax of the L2AddressData data type shall comply with the following definition:

```
tosca.datatypes.nfv.L2AddressData:
  derived_from: toasca.datatypes.Root
  description: Describes the information on the MAC addresses to be assigned to a
  connection point.
  properties:
```

```

mac_address_assignment:
  type: boolean
  description: Specifies if the address assignment is the responsibility of
management and orchestration function or not. If it is set to True, it is the
management and orchestration function responsibility
  required: true

```

#### 6.2.1.4 Examples

```

<some_tosca_entity>:
  properties:
    l2_address_data:
      mac_address_assignment: true

```

#### 6.2.1.5 Additional Requirements

None.

### 6.2.2 tosca.datatypes.nfv.L3AddressData

#### 6.2.2.1 Description

The L3AddressData data type supports providing information about Layer 3 level addressing scheme and parameters applicable to a CP, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.2.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.2.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	L3AddressData
<b>Type Qualified Name</b>	toscanfv:L3AddressData
<b>Type URI</b>	tosca.datatypes.nfv.L3AddressData

#### 6.2.2.2 Properties

The properties of the L3AddressData data type shall comply with the provisions set out in table 6.2.2.2-1.

**Table 6.2.2.2-1: Properties**

Name	Required	Type	Constraints	Description
ip_address_assignment	yes	boolean		Specify if the address assignment is the responsibility of management and orchestration function or not. If it is set to True, it is the management and orchestration function responsibility.
floating_ip_activated	yes	boolean		Specify if the floating IP scheme is activated on the Connection Point or not.
ip_address_type	no	string	Valid values: ipv4, ipv6	Define address type. The address type should be aligned with the address type supported by the layer_protocols properties of the parent VnfExtCp.
number_of_ip_address	no	Integer	greater_than: 0	Minimum number of IP addresses to be assigned.

### 6.2.2.3 Definition

The syntax of the L3AddressData data type shall comply with the following definition:

```

tosca.datatypes.nfv.L3AddressData:
  derived_from: toska.datatypes.Root
  description: Provides information about Layer 3 level addressing scheme and
parameters applicable to a CP
  properties:
    ip_address_assignment:
      type: boolean
      description: Specifies if the address assignment is the responsibility of
management and orchestration function or not. If it is set to True, it is the
management and orchestration function responsibility
      required: true
    floating_ip_activated:
      type: boolean
      description: Specifies if the floating IP scheme is activated on the
Connection Point or not
      required: true
    ip_address_type:
      type: string
      description: Defines address type. The address type should be aligned with the
address type supported by the layer_protocols properties of the parent VnfExtCp
      required: false
      constraints:
        - valid_values: [ ipv4, ipv6 ]
    number_of_ip_address:
      type: integer
      description: Minimum number of IP addresses to be assigned
      required: false
      constraints:
        - greater_than: 0

```

### 6.2.2.4 Examples

```

<some_tosca_entity>:
  properties:
    l3_address_data:
      ip_address_assignment: true
      floating_ip_activated: true
      ip_address_type: ipv4
      number_of_ip_address: 4

```

### 6.2.2.5 Additional Requirements

None.

## 6.2.3 toska.datatypes.nfv.AddressData

### 6.2.3.1 Description

The AddressData data type describes information about the addressing scheme and parameters applicable to a CP, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.3.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.3.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	AddressData
<b>Type Qualified Name</b>	toscanfv:AddressData
<b>Type URI</b>	tosca.datatypes.nfv.AddressData

### 6.2.3.2 Properties

The properties of the AddressData data type shall comply with the provisions set out in table 6.2.3.2-1.

**Table 6.2.3.2-1: Properties**

Name	Required	Type	Constraints	Description
address_type	yes	string	Valid values: mac_address ip_address.	Describes the type of the address to be assigned to a connection point The content type shall be aligned with the address type supported by the layerProtocol property of the connection point.
l2_address_data	no	tosca.datatypes.nfv.L2AddressData	Shall be present when the address_type is mac_address.	Provides the information on the MAC addresses to be assigned to a connection point.
l3_address_data	no	tosca.datatypes.nfv.L3AddressData	Shall be present when the address_type is ip_address.	Provides the information on the IP addresses to be assigned to a connection point.

### 6.2.3.3 Definition

The syntax of the AddressData data type shall comply with the following definition:

```

tosca.datatypes.nfv.AddressData:
  derived_from: toscanfv:Root
  description: Describes information about the addressing scheme and parameters
  applicable to a CP
  properties:
    address_type:
      type: string
      description: Describes the type of the address to be assigned to a connection
      point. The content type shall be aligned with the address type supported by the
      layerProtocol property of the connection point
      required: true
      constraints:
        - valid_values: [ mac_address, ip_address ]
    l2_address_data:
      type: toscanfv.L2AddressData
      description: Provides the information on the MAC addresses to be assigned to a
      connection point.
      required: false
    l3_address_data:
      type: toscanfv.L3AddressData
      description: Provides the information on the IP addresses to be assigned to a
      connection point
      required: false

```

### 6.2.3.4 Examples

```
<some_tosca_entity>:
properties:
  address_data:
    address_type: ip_address
  l3_address_data:
    ip_address_assignment: true
    floating_ip_activated: true
    ip_address_type: ipv4
    number_of_ip_address: 4
```

### 6.2.3.5 Additional Requirements

None.

## 6.2.4 tosca.datatypes.nfv.VirtualNetworkInterfaceRequirements

### 6.2.4.1 Description

The VirtualNetworkInterfaceRequirements data type describes requirements on a virtual network interface, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.4.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.4.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VirtualNetworkInterfaceRequirements
<b>Type Qualified Name</b>	toscanfv:VirtualNetworkInterfaceRequirements
<b>Type URI</b>	tosca.datatypes.nfv.VirtualNetworkInterfaceRequirements

### 6.2.4.2 Properties

The properties of the VirtualNetworkInterfaceRequirements data type shall comply with the provisions set out in table 6.2.4.2-1.

**Table 6.2.4.2-1: Properties**

Name	Required	Type	Constraints	Description
name	no	string		Provides a human readable name for the requirement.
description	no	string		Provides a human readable description of the requirement.
support_mandatory	yes	boolean		Indicates whether fulfilling the constraint is mandatory (TRUE) for successful operation or desirable (FALSE).
network_interface_requirements	yes	map of string		The network interface requirements. A map of strings that contain a set of key-value pairs that describes the hardware platform specific network interface deployment requirements.  More information regarding the usage of this property is available at: <a href="https://register.etsi.org">https://register.etsi.org</a>

Name	Required	Type	Constraints	Description
nic_io_requirements	no	tosca.datatypes.nfv.LogicalNodeData		This references (couples) the CP with any logical node I/O requirements (for network devices) that may have been created. Linking these attributes is necessary so that so that I/O requirements that need to be articulated at the logical node level can be associated with the network interface requirements associated with the CP.

### 6.2.4.3 Definition

The syntax of the VirtualNetworkInterfaceRequirements data type shall comply with the following definition:

```

tosca.datatypes.nfv.VirtualNetworkInterfaceRequirements:
  derived_from: toska.datatypes.Root
  description: Describes requirements on a virtual network interface
  properties:
    name:
      type: string
      description: Provides a human readable name for the requirement.
      required: false
    description:
      type: string
      description: Provides a human readable description of the requirement.
      required: false
    support_mandatory:
      type: boolean
      description: Indicates whether fulfilling the constraint is mandatory (TRUE)
for successful operation or desirable (FALSE).
      required: true
    network_interface_requirements:
      type: map
      description: The network interface requirements. A map of strings that contain
a set of key-value pairs that describes the hardware platform specific network
interface deployment requirements.
      required: true
      entry_schema:
        type: string
    nic_io_requirements:
      type: toska.datatypes.nfv.LogicalNodeData
      description: references (couples) the CP with any logical node I/O
requirements (for network devices) that may have been created. Linking these
attributes is necessary so that so that I/O requirements that need to be articulated
at the logical node level can be associated with the network interface requirements
associated with the CP.
      required: false

```

### 6.2.4.4 Examples

None.

### 6.2.4.5 Additional Requirements

None.

## 6.2.5 tosca.datatypes.nfv.ConnectivityType

### 6.2.5.1 Description

The ConnectivityType data type describes additional connectivity information of a virtualLink, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.5.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.5.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	ConnectivityType
<b>Type Qualified Name</b>	toscanfv:ConnectivityType
<b>Type URI</b>	tosca.datatypes.nfv.ConnectivityType

### 6.2.5.2 Properties

The properties of the ConnectivityType shall comply with the provisions set out in table 6.2.5.2-1.

**Table 6.2.5.2-1: Properties**

Name	Required	Type	Constraints	Description
layer_protocols	yes	list of string	Valid values: ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire	Identifies the protocol a virtualLink gives access to (ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire). The top layer protocol of the virtualLink protocol stack shall always be provided. The lower layer protocols may be included when there are specific requirements on these layers.
flow_pattern	no	string	Valid values: line, tree, mesh	Identifies the flow pattern of the connectivity.

### 6.2.5.3 Definition

The syntax of the ConnectivityType data type shall comply with the following definition:

```
tosca.datatypes.nfv.ConnectivityType:
  derived_from: tosca.datatypes.Root
  description: describes additional connectivity information of a virtualLink
  properties:
    layer_protocols:
      type: list
      description: Identifies the protocol a virtualLink gives access to (ethernet,
  mpls, odu2, ipv4, ipv6, pseudo-wire). The top layer protocol of the virtualLink
  protocol stack shall always be provided. The lower layer protocols may be included
  when there are specific requirements on these layers.
      required: true
      entry_schema:
        type: string
      constraints:
        - valid_values: [ ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire ]
    flow_pattern:
      type: string
      description: Identifies the flow pattern of the connectivity
      required: false
      constraints:
        - valid_values: [ line, tree, mesh ]
```

### 6.2.5.4 Examples

```
<some_tosca_entity>:
  properties:
    connectivity_type:
      layer_protocol:
        - ipv4
      flow_pattern: mesh
```

### 6.2.5.5 Additional Requirements

None.

## 6.2.6 toska.datatypes.nfv.RequestedAdditionalCapability

### 6.2.6.1 Description

The RequestedAdditionalCapability data type describes requested additional capability for a particular VDU, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.6.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.6.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	RequestedAdditionalCapability
<b>Type Qualified Name</b>	toscanfv:RequestedAdditionalCapability
<b>Type URI</b>	tosca.datatypes.nfv.RequestedAdditionalCapability

### 6.2.6.2 Properties

The properties of the RequestedAdditionalCapability data type shall comply with the provisions set out in table 6.2.6.2-1.

**Table 6.2.6.2-1: Properties**

Name	Required	Type	Constraints	Description
requested_additional_capability_name	yes	string		Identifies a requested additional capability for the VDU.
support_mandatory	yes	boolean		Indicates whether the requested additional capability is mandatory for successful operation.
min_requested_additional_capability_version	no	string		Identifies the minimum version of the requested additional capability.
preferred_requested_additional_capability_version	no	string		Identifies the preferred version of the requested additional capability.
target_performance_parameters	yes	map of string		Identifies specific attributes, dependent on the requested additional capability type.

### 6.2.6.3 Definition

The syntax of the RequestedAdditionalCapability data type shall comply with the following definition:

```
tosca.datatypes.nfv.RequestedAdditionalCapability:
  derived_from: toska.datatypes.Root
  description: describes requested additional capability for a particular VDU
  properties:
    requested_additional_capability_name:
      type: string
      description: Identifies a requested additional capability for the VDU.
```



```

    required: true
  support_mandatory:
    type: boolean
    description: Indicates whether the requested additional capability is
mandatory for successful operation.
    required: true
  min_requested_additional_capability_version:
    type: string
    description: Identifies the minimum version of the requested additional
capability.
    required: false
  preferred_requested_additional_capability_version:
    type: string
    description: Identifies the preferred version of the requested additional
capability.
    required: false
  target_performance_parameters:
    type: map
    description: Identifies specific attributes, dependent on the requested
additional capability type.
    required: true
  entry_schema:
    type: string

```

#### 6.2.6.4 Examples

None.

#### 6.2.6.5 Additional Requirements

None.

### 6.2.7 `tosca.datatypes.nfv.VirtualMemory`

#### 6.2.7.1 Description

The `VirtualMemory` data type supports the specification of requirements related to virtual memory of a virtual compute resource, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.7.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.7.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>VirtualMemory</code>
<b>Type Qualified Name</b>	<code>toscanfv:VirtualMemory</code>
<b>Type URI</b>	<code>tosca.datatypes.nfv.VirtualMemory</code>

#### 6.2.7.2 Properties

The properties of the `VirtualMemory` data type shall comply with the provisions set out in table 6.2.7.2-1.

Table 6.2.7.2-1: Properties

Name	Required	Type	Constraints	Description
virtual_mem_size	yes	scalar-unit.size		Amount of virtual memory.
virtual_mem_oversubscription_policy	no	string		The memory core oversubscription policy in terms of virtual memory to physical memory on the platform.
vdu_mem_requirements	no	map of string		The hardware platform specific VDU memory requirements. A map of strings that contains a set of key-value pairs that describes hardware platform specific VDU memory requirements.  More information regarding the usage of this property is available at: <a href="https://register.etsi.org">https://register.etsi.org</a>
numa_enabled	no	boolean	default: false	It specifies the memory allocation to be cognisant of the relevant process/core allocation.

### 6.2.7.3 Definitions

The syntax of the VirtualMemory data type shall comply with the following definition:

```

tosca.datatypes.nfv.VirtualMemory:
  derived_from: tosca.datatypes.Root
  description: supports the specification of requirements related to virtual
memory of a virtual compute resource
  properties:
    virtual_mem_size:
      type: scalar-unit.size
      description: Amount of virtual memory.
      required: true
    virtual_mem_oversubscription_policy:
      type: string
      description: The memory core oversubscription policy in terms of virtual
memory to physical memory on the platform.
      required: false
    vdu_mem_requirements:
      type: map
      description: The hardware platform specific VDU memory requirements. A map
of strings that contains a set of key-value pairs that describes hardware platform
specific VDU memory requirements.
      required: false
      entry_schema:
        type: string
    numa_enabled:
      type: boolean
      description: It specifies the memory allocation to be cognisant of the
relevant process/core allocation.
      required: false
      default: false

```

### 6.2.7.4 Examples

None.

### 6.2.7.5 Additional Requirements

None.

## 6.2.8 toska.datatypes.nfv.VirtualCpu

### 6.2.8.1 Description

The VirtualCpu data type supports the specification of requirements related to virtual CPU(s) of a virtual compute resource, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.8.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.8.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VirtualCpu
<b>Type Qualified Name</b>	toscanfv:VirtualCpu
<b>Type URI</b>	tosca.datatypes.nfv.VirtualCpu

### 6.2.8.2 Properties

The properties of the VirtualCpu data type shall comply with the provisions set out in table 6.2.8.2-1.

**Table 6.2.8.2-1: Properties**

Name	Required	Type	Constraints	Description
cpu_architecture	no	string		CPU architecture type. Examples are x86, ARM.
num_virtual_cpu	yes	integer	greater_than: 0	Number of virtual CPUs.
virtual_cpu_clock	no	scalar-unit.frequency		Minimum virtual CPU clock rate.
virtual_cpu_oversubscription_policy	no	string		CPU core oversubscription policy e.g. the relation of virtual CPU cores to physical CPU cores/threads.
vdu_cpu_requirements	no	map of string		The hardware platform specific VDU CPU requirements. A map of strings that contains a set of key-value pairs describing VDU CPU specific hardware platform requirements.  More information regarding the usage of this property is available at: <a href="https://register.etsi.org">https://register.etsi.org</a> .
virtual_cpu_pinning	no	tosca.datatypes.nfv.VirtualCpuPinning		The virtual CPU pinning configuration for the virtualised compute resource.

### 6.2.8.3 Definition

The syntax of the VirtualCpu data type shall comply with the following definition:

```
tosca.datatypes.nfv.VirtualCpu:
  derived_from: toska.datatypes.Root
  description: Supports the specification of requirements related to virtual CPU(s)
of a virtual compute resource
  properties:
    cpu_architecture:
      type: string
      description: CPU architecture type. Examples are x86, ARM
      required: false
    num_virtual_cpu:
      type: integer
      description: Number of virtual CPUs
      required: true
```

```

constraints:
  - greater_than: 0
virtual_cpu_clock:
  type: scalar-unit.frequency
  description: Minimum virtual CPU clock rate
  required: false
virtual_cpu_oversubscription_policy:
  type: string
  description: CPU core oversubscription policy e.g. the relation of virtual CPU
cores to physical CPU cores/threads.
  required: false
vdu_cpu_requirements:
  type: map
  description: The hardware platform specific VDU CPU requirements. A map of
strings that contains a set of key-value pairs describing VDU CPU specific hardware
platform requirements.
  required: false
  entry_schema:
    type: string
virtual_cpu_pinning:
  type: toska.datatypes.nfv.VirtualCpuPinning
  description: The virtual CPU pinning configuration for the virtualised compute
resource.
  required: false

```

#### 6.2.8.4 Examples

None.

#### 6.2.8.5 Additional Requirements

None.

### 6.2.9 toska.datatypes.nfv.VirtualCpuPinning

#### 6.2.9.1 Description

The VirtualCpuPinning data type supports the specification of requirements related to the virtual CPU pinning configuration of a virtual compute resource, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.9.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.9.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VirtualCpuPinning
<b>Type Qualified Name</b>	toscanfv:VirtualCpuPinning
<b>Type URI</b>	tosca.datatypes.nfv.VirtualCpuPinning

#### 6.2.9.2 Properties

The properties of the VirtualCpuPinning data type shall comply with the provisions set out in table 6.2.9.2-1.

Table 6.2.9.2-1: Properties

Name	Required	Type	Constraints	Description
virtual_cpu_pinning_policy	no	string	Valid values: static, dynamic	Indicates the policy for CPU pinning. The policy can take values of "static" or "dynamic". In case of "dynamic" the allocation of virtual CPU cores to logical CPU cores is decided by the VIM. (e.g. SMT (Simultaneous Multi-Threading) requirements). In case of "static" the allocation is requested to be according to the virtual_cpu_pinning_rule.
virtual_cpu_pinning_rule	no	list of string		Provides the list of rules for allocating virtual CPU cores to logical CPU cores/threads.

### 6.2.9.3 Definition

The syntax of the VirtualCpuPinning data type shall comply with the following definition:

```

tosca.datatypes.nfv.VirtualCpuPinning:
  derived_from: toasca.datatypes.Root
  description: Supports the specification of requirements related to the virtual CPU
pinning configuration of a virtual compute resource
  properties:
    virtual_cpu_pinning_policy:
      type: string
      description: Indicates the policy for CPU pinning. The policy can take values
of "static" or "dynamic". In case of "dynamic" the allocation of virtual CPU cores to
logical CPU cores is decided by the VIM. (e.g. SMT (Simultaneous Multi-Threading)
requirements). In case of "static" the allocation is requested to be according to the
virtual_cpu_pinning_rule.
      required: false
      constraints:
        - valid_values: [ static, dynamic ]
    virtual_cpu_pinning_rule:
      type: list
      description: Provides the list of rules for allocating virtual CPU cores to
logical CPU cores/threads
      required: false
      entry_schema:
        type: string

```

### 6.2.9.4 Examples

None.

### 6.2.9.5 Additional Requirements

The virtual\_cpu\_pinning\_rule shall be included if the virtual\_cpu\_pinning\_policy property is set to "static" and shall be absent otherwise.

## 6.2.10 toasca.datatypes.nfv.VnfcConfigurableProperties

### 6.2.10.1 Description

The VnfcConfigurableProperties data type defines the configurable properties of a VNFC, as defined in ETSI GS NFV-IFA 011 [1].

Table 6.2.10.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.10.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VnfcConfigurableProperties
<b>Type Qualified Name</b>	toscanfv:VnfcConfigurableProperties
<b>Type URI</b>	tosca.datatypes.nfv.VnfcConfigurableProperties

## 6.2.10.2 Properties

The properties of the VnfcconfigurableProperties shall comply with the provisions set out in table 6.2.10.2-1.

**Table 6.2.10.2-1: Properties**

Name	Required	Type	Constraints	Description
additional_vnfc_configurable_properties	no	tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties		Describes additional configuration for VNFC that can be modified using the ModifyVnfInfo operation.

## 6.2.10.3 Definition

The syntax of the VnfcConfigurableProperties data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfcConfigurableProperties:
  derived_from: tosca.datatypes.Root
  description: Defines the configurable properties of a VNFC
  #properties:
    # additional_vnfc_configurable_properties:
    #   type: tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties
    #   description: Describes additional configuration for VNFC that can be
modified using the ModifyVnfInfo operation
    #   required: false
    # derived types are expected to introduce
    # additional_vnfc_configurable_properties with its type derived from
    # tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties
```

## 6.2.10.4 Examples

```
tosca_definitions_version: tosca_simple_yaml_1_2

node_types:
  MyCompany.nodes.nfv.Vdu.Aux:
    derived_from: tosca.nodes.nfv.Vdu.Compute
    properties:
      configurable_properties:
        type: MyCompany.datatypes.nfv.AuxVnfcConfigurableProperties

data_types:
  MyCompany.datatypes.nfv.AuxVnfcConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfcConfigurableProperties
    properties:
      additional_vnfc_configurable_properties:
        type: MyCompany.datatypes.nfv.AuxVnfcAdditionalConfigurableProperties
        required: true
```

```

MyCompany.datatypes.nfv.AuxVnfcAdditionalConfigurableProperties:
  derived_from: toasca.datatypes.nfv.VnfcAdditionalConfigurableProperties
  properties:
    name_prefix_in_vim:
      type: string
      required: false
    dns_server:
      type: string
      required: true

topology_template:
  ..

node_templates:
  aux:
    type: MyCompany.nodes.nfv.Vdu.Aux
    properties:
      ..

```

### 6.2.10.5 Additional Requirements

None.

## 6.2.11 toasca.datatypes.nfv.VnfcAdditionalConfigurableProperties

### 6.2.11.1 Description

The VnfcAdditionalConfigurableProperties type is an empty base type for deriving data types for describing additional configurable properties for a given VNFC. Table 6.2.11.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.11.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VnfcAdditionalConfigurableProperties
<b>Type Qualified Name</b>	toscanfv:VnfcAdditionalConfigurableProperties
<b>Type URI</b>	toasca.datatypes.nfv.VnfcAdditionalConfigurableProperties

### 6.2.11.2 Properties

None.

### 6.2.11.3 Definition

The syntax of the VnfcAdditionalConfigurableProperties data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties:
  derived_from: toasca.datatypes.Root
  description: VnfcAdditionalConfigurableProperties type is an empty base type
for deriving data types for describing additional configurable properties for a
given VNFC.

```

## 6.2.12 tosca.datatypes.nfv.VduProfile

### 6.2.12.1 Description

The VduProfile data type describes additional instantiation data for a given Vdu.Compute used in a specific deployment flavour. Table 6.2.12.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.12.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VduProfile
<b>Type Qualified Name</b>	toscanfv:VduProfile
<b>Type URI</b>	tosca.datatypes.nfv.VduProfile

### 6.2.12.2 Properties

The properties of the VduProfile data type shall comply with the provisions set out in table 6.2.12.2-1.

**Table 6.2.12.2-1: Properties**

Name	Required	Type	Constraints	Description
min_number_of_instances	yes	integer	greater_or_equal: 0	Minimum number of instances of the VNFC based on this Vdu.Compute that is permitted to exist for a particular VNF deployment flavour.
max_number_of_instances	yes	integer	greater_or_equal: 0	Maximum number of instances of the VNFC based on this Vdu.Compute that is permitted to exist for a particular VNF deployment flavour.
NOTE: A vduuld property, which exists in ETSI GS NFV-IFA 011 [1] is not needed, as the VduProfile is contained in the Vdu.Compute node.				

### 6.2.12.3 Definition

The syntax of the VduProfile data type shall comply with the following definition:

```
tosca.datatypes.nfv.VduProfile:
  derived_from: tosca.datatypes.Root
  description: describes additional instantiation data for a given Vdu.Compute used
in a specific deployment flavour.
  properties:
    min_number_of_instances:
      type: integer
      description: Minimum number of instances of the VNFC based on this Vdu.Compute
that is permitted to exist for a particular VNF deployment flavour.
      required: true
      constraints:
        - greater_or_equal: 0
    max_number_of_instances:
      type: integer
      description: Maximum number of instances of the VNFC based on this Vdu.Compute
that is permitted to exist for a particular VNF deployment flavour.
      required: true
      constraints:
        - greater_or_equal: 0
```



### 6.2.12.4 Examples

```

tosca_definitions_version: tosca_simple_yaml_1_2

topology template:
...

node_templates:
  VDU_A:
    type: tosca.nodes.nfv.Vdu.Compute
    properties:
      vdu_profile:
        min_number_of_instances: 2
        max_number_of_instances: 6
    # other properties omitted for brevity
    requirements:
      - virtual_storage: VirtualStorage_A1
      - virtual_storage: VirtualStorage_A2
    capabilities:
      virtual_binding:

  VirtualStorage_A1:
    type: tosca.nodes.nfv.Vdu.VirtualBlockStorage
    properties:
      # omitted for brevity
    capabilities:
      virtual_storage

  VirtualStorage_A2:
    type: tosca.nodes.nfv.Vdu.VirtualBlockStorage
    properties:
      # omitted for brevity
    capabilities:
      virtual_storage

  VduCp_A1:
    type: tosca.nodes.nfv.VduCp
    properties:
      # omitted for brevity
    requirements:
      - virtual_binding: VDU_A
      - virtual_link

  VduCp_A2:
    type: tosca.nodes.nfv.VduCp
    properties:
      # omitted for brevity
    requirements:
      - virtual_binding: VDU_A
      - virtual_link

```

Above snippet shows part of a topology template. The VDU\_A node template is a Vdu.Compute node that is connected to two VirtualBlockStorage nodes: VirtualStorage\_A1 and VirtualStorage\_A2. It also has two VduCps: VduCp\_A1 and Vdu\_CpA2.

The minimum number of instances of VDU\_A that are permitted to exist is 2. Likewise, the minimum number of instances of VirtualStorage\_A1, VirtualStorage\_A2, VduCp\_A1 and VduCp\_A2 that are permitted to exist is 2.

The maximum number of instances of VDU\_A that are permitted to exist is 6. Likewise, the maximum number of instances of VirtualStorage\_A1, VirtualStorage\_A2, VduCp\_A1 and VduCp\_A2 that are permitted to exist is 6.

### 6.2.12.5 Additional requirements

The properties of the vdu\_profile indicate the maximum and minimum number of Vdu.Compute instances that are permitted to exist, created from a given Vdu.Compute node template during its lifecycle, as well as:

- The maximum and minimum number of instances of each VirtualBlockStorage, VirtualObjectStorage and VirtualFileStorage nodes connected to the Vdu.Compute via one particular occurrence of the virtual\_storage requirement.
- The maximum and minimum number instances of each VduCp node connected to the Vdu.Compute via one particular occurrence of the virtual\_binding capability.

## 6.2.13 tosca.datatypes.nfv.VIProfile

### 6.2.13.1 Description

The VIProfile data type describes additional instantiation data for a given VL used in a specific VNF deployment flavour. Table 6.2.13.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.13.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VIProfile
<b>Type Qualified Name</b>	toscanfv:VIProfile
<b>Type URI</b>	tosca.datatypes.nfv.VIProfile

### 6.2.13.2 Properties

The properties of the VIProfile data type shall comply with the provisions set out in table 6.2.13.2-1.

**Table 6.2.13.2-1: Properties**

Name	Required	Type	Constraints	Description
max_bitrate_requirements	yes	tosca.datatypes.nfv.LinkBitrateRequirements		Specifies the maximum bitrate requirements for a VL instantiated according to this profile.
min_bitrate_requirements	yes	tosca.datatypes.nfv.LinkBitrateRequirements		Specifies the minimum bitrate requirements for a VL instantiated according to this profile.
qos	no	tosca.datatypes.nfv.Qos		Specifies the QoS requirements of a VL instantiated according to this profile.
virtual_link_protocol_data	no	list of tosca.datatypes.nfv.VirtualLinkProtocolData		Specifies the protocol data for a virtual link.
NOTE: A vnfVirtualLinkDescId property, which exists in ETSI GS NFV-IFA 011 [1] is not needed, as the VLProfile is contained in the VL node.				

### 6.2.13.3 Definition

The syntax of the VIProfile data type shall comply with the following definition:

```

tosca.datatypes.nfv.VIProfile:
  derived_from: tosca.datatypes.Root
  description: Describes additional instantiation data for a given VL used in a
  specific VNF deployment flavour.
  properties:
    max_bitrate_requirements:
      type: tosca.datatypes.nfv.LinkBitrateRequirements
      description: Specifies the maximum bitrate requirements for a VL instantiated
  according to this profile.
      required: true
    min_bitrate_requirements:
      type: tosca.datatypes.nfv.LinkBitrateRequirements
      description: Specifies the minimum bitrate requirements for a VL instantiated
  according to this profile.
      required: true
    qos:
      type: tosca.datatypes.nfv.Qos
      description: Specifies the QoS requirements of a VL instantiated according to
  this profile.
      required: false
    virtual_link_protocol_data:
      type: list
      description: Specifies the protocol data for a virtual link.
      required: false
    entry_schema:
      type: tosca.datatypes.nfv.VirtualLinkProtocolData

```

### 6.2.13.4 Examples

None.

### 6.2.13.5 Additional Requirements

None.

## 6.2.14 tosca.datatypes.nfv.VirtualLinkProtocolData

### 6.2.14.1 Description

The VirtualLinkProtocolData data type describes one protocol layer and associated protocol data for a given virtual link used in a specific VNF deployment flavour. Table 6.2.14.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.14.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VirtualLinkProtocolData
<b>Type Qualified Name</b>	toscanfv:VirtualLinkProtocolData
<b>Type URI</b>	tosca.datatypes.nfv.VirtualLinkProtocolData

### 6.2.14.2 Properties

The properties of the VirtualLinkProtocolData data type shall comply with the provisions set out in table 6.2.14.2-1.

**Table 6.2.14.2-1: Properties**

Name	Required	Type	Constraints	Description
associated_layer_protocol	yes	string	Valid values: ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire	Identifies one of the protocols a virtualLink gives access to (ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire) as specified by the connectivity_type property.
l2_protocol_data	no	tosca.datatype s.nfv.L2ProtocolData		Specifies the L2 protocol data for a virtual link. Shall be present when the associatedLayerProtocol attribute indicates a L2 protocol and shall be absent otherwise.
l3_protocol_data	no	tosca.datatype s.nfv.L3ProtocolData		Specifies the L3 protocol data for this virtual link. Shall be present when the associatedLayerProtocol attribute indicates a L3 protocol and shall be absent otherwise.

### 6.2.14.3 Definition

The syntax of the VirtualLinkProtocolData data type shall comply with the following definition:

```

tosca.datatypes.nfv.VirtualLinkProtocolData:
  derived_from: tosca.datatypes.Root
  description: describes one protocol layer and associated protocol data for a
  given virtual link used in a specific VNF deployment flavour
  properties:
    associated_layer_protocol:
      type: string
      description: Identifies one of the protocols a virtualLink gives access
      to (ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire) as specified by the
      connectivity_type property.
      required: true
      constraints:
        - valid_values: [ ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire ]
    l2_protocol_data:
      type: tosca.datatypes.nfv.L2ProtocolData
      description: Specifies the L2 protocol data for a virtual link. Shall be
      present when the associatedLayerProtocol attribute indicates a L2 protocol and
      shall be absent otherwise.
      required: false
    l3_protocol_data:
      type: tosca.datatypes.nfv.L3ProtocolData
      description: Specifies the L3 protocol data for this virtual link. Shall
      be present when the associatedLayerProtocol attribute indicates a L3 protocol and
      shall be absent otherwise.
      required: false

```

### 6.2.14.4 Examples

None.

### 6.2.14.5 Additional Requirements

None.

## 6.2.15 tosca.datatypes.nfv.L2ProtocolData

### 6.2.15.1 Description

The L2ProtocolData data type describes L2 protocol data for a given virtual link used in a specific VNF deployment flavour. Table 6.2.15.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.15.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	L2ProtocolData
<b>Type Qualified Name</b>	toscanfv:L2ProtocolData
<b>Type URI</b>	tosca.datatypes.nfv.L2ProtocolData

### 6.2.15.2 Properties

The properties of the L2ProtocolData data type shall comply with the provisions set out in table 6.2.15.2-1.

**Table 6.2.15.2-1: Properties**

Name	Required	Type	Constraints	Description
name	no	string		Identifies the network name associated with this L2 protocol.
network_type	no	string	Valid values: flat, vlan, vxlan, gre	Specifies the network type for this L2 protocol. The value may be overridden at run-time.
vlan_transparent	no	boolean	default: false	Specifies whether to support VLAN transparency for this L2 protocol or not.
mtu	no	integer	greater_than : 0	Specifies the maximum transmission unit (MTU) value for this L2 protocol.

### 6.2.15.3 Definition

The syntax of the L2ProtocolData data type shall comply with the following definition:

```
tosca.datatypes.nfv.L2ProtocolData:
  derived_from: tosca.datatypes.Root
  description: describes L2 protocol data for a given virtual link used in a
specific VNF deployment flavour.
  properties:
    name:
      type: string
      description: Identifies the network name associated with this L2 protocol.
      required: false
    network_type:
      type: string
      description: Specifies the network type for this L2 protocol. The value
may be overridden at run-time.
      required: false
      constraints:
        - valid_values: [ flat, vlan, vxlan, gre ]
    vlan_transparent:
      type: boolean
      description: Specifies whether to support VLAN transparency for this L2
protocol or not.
      required: false
      default: false
    mtu:
```

```

type: integer
description: Specifies the maximum transmission unit (MTU) value for this
L2 protocol.
required: false
constraints:
  - greater_than: 0

```

#### 6.2.15.4 Examples

None.

#### 6.2.15.5 Additional Requirements

None.

### 6.2.16 tosca.datatypes.nfv.L3ProtocolData

#### 6.2.16.1 Description

The L3ProtocolData data type describes L3 protocol data for a given virtual link used in a specific VNF deployment flavour. Table 6.2.16.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.16.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	L3ProtocolData
<b>Type Qualified Name</b>	toscanfv:L3ProtocolData
<b>Type URI</b>	tosca.datatypes.nfv.L3ProtocolData

#### 6.2.16.2 Properties

The properties of the L3ProtocolData data type shall comply with the provisions set out in table 6.2.16.2-1.

**Table 6.2.16.2-1: Properties**

Name	Required	Type	Constraints	Description
name	no	string		Identifies the network name associated with this L3 protocol.
ip_version	yes	string	Valid values: ipv4 ipv6	Specifies IP version of this L3 protocol.  The value of the ip_version property shall be consistent with the value of the layer_protocol in the connectivity_type property of the virtual link node.
cidr	yes	string		Specifies the CIDR (Classless Inter-Domain Routing) of this L3 protocol. The value may be overridden at run-time.
ip_allocation_pools	no	list of tosca.datatypes.nfv.IpAllocationPool		Specifies the allocation pools with start and end IP addresses for this L3 protocol. The value may be overridden at run-time.
gateway_ip	no	string		Specifies the gateway IP address for this L3 protocol. The value may be overridden at run-time.
dhcp_enabled	no	boolean		Indicates whether DHCP (Dynamic Host Configuration Protocol) is enabled or disabled for this L3 protocol. The value may be overridden at run-time.

Name	Required	Type	Constraints	Description
ipv6_address_mode	no	string	Valid values: slaac  dhcpv6-stateful  dhcpv6-stateless	Specifies IPv6 address mode.  May be present when the value of the ipVersion attribute is "ipv6" and shall be absent otherwise. The value may be overridden at run-time.

### 6.2.16.3 Definition

The syntax of the L3ProtocolData data type shall comply with the following definition:

```

tosca.datatypes.nfv.L3ProtocolData:
  derived_from: toasca.datatypes.Root
  description: describes L3 protocol data for a given virtual link used in a
specific VNF deployment flavour.
  properties:
    name:
      type: string
      description: Identifies the network name associated with this L3 protocol.
      required: false
    ip_version:
      type: string
      description: Specifies IP version of this L3 protocol. The value of the
ip_version property shall be consistent with the value of the layer_protocol in
the connectivity_type property of the virtual link node.
      required: true
      constraints:
        - valid_values: [ ipv4, ipv6 ]
    cidr:
      type: string
      description: Specifies the CIDR (Classless Inter-Domain Routing) of this
L3 protocol. The value may be overridden at run-time.
      required: true
    ip_allocation_pools:
      type: list
      description: Specifies the allocation pools with start and end IP
addresses for this L3 protocol. The value may be overridden at run-time.
      required: false
      entry_schema:
        type: toasca.datatypes.nfv.IpAllocationPool
    gateway_ip:
      type: string
      description: Specifies the gateway IP address for this L3 protocol. The
value may be overridden at run-time.
      required: false
    dhcp_enabled:
      type: boolean
      description: Indicates whether DHCP (Dynamic Host Configuration Protocol)
is enabled or disabled for this L3 protocol. The value may be overridden at run-
time.
      required: false
    ipv6_address_mode:
      type: string
      description: Specifies IPv6 address mode. May be present when the value of
the ipVersion attribute is "ipv6" and shall be absent otherwise. The value may be
overridden at run-time.
      required: false
      constraints:
        - valid_values: [ slaac, dhcpv6-stateful, dhcpv6-stateless ]

```

### 6.2.16.4 Examples

None.

### 6.2.16.5 Additional Requirements

None.

## 6.2.17 tosca.datatypes.nfv.IpAllocationPool

### 6.2.17.1 Description

The IpAllocationPool data type specifies a range of IP addresses. Table 6.2.17.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.17.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	IpAllocationPool
<b>Type Qualified Name</b>	toscanfv:IpAllocationPool
<b>Type URI</b>	tosca.datatypes.nfv.IpAllocationPool

### 6.2.17.2 Properties

The properties of the IpAllocationPool data type shall comply with the provisions set out in table 6.2.17.2-1.

**Table 6.2.17.2-1: Properties**

Name	Required	Type	Constraints	Description
start_ip_address	yes	string		The IP address to be used as the first one in a pool of addresses derived from the cidr block full IP range
end_ip_address	yes	string		The IP address to be used as the last one in a pool of addresses derived from the cidr block full IP range

### 6.2.17.3 Definition

The syntax of the IpAllocationPool data type shall comply with the following definition:

```
tosca.datatypes.nfv.IpAllocationPool:
  derived_from: tosca.datatypes.Root
  description: Specifies a range of IP addresses
  properties:
    start_ip_address:
      type: string
      description: The IP address to be used as the first one in a pool of
addresses derived from the cidr block full IP range
      required: true
    end_ip_address:
      type: string
      description: The IP address to be used as the last one in a pool of
addresses derived from the cidr block full IP range
      required: true
```

### 6.2.17.4 Examples

None.



### 6.2.17.5 Additional Requirements

None.

## 6.2.18 `tosca.datatypes.nfv.InstantiationLevel`

### 6.2.18.1 Description

The `InstantiationLevel` data type describes the scale level for each aspect that corresponds to a given level of resources to be instantiated within a deployment flavour in term of the number VNFC instances . Table 6.2.18.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.18.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>InstantiationLevel</code>
<b>Type Qualified Name</b>	<code>toscanfv:InstantiationLevel</code>
<b>Type URI</b>	<code>tosca.datatypes.nfv.InstantiationLevel</code>

### 6.2.18.2 Properties

The properties of the `InstantiationLevel` data type shall comply with the provisions set out in table 6.2.18.2-1.

**Table 6.2.18.2-1: Properties**

Name	Required	Type	Constraints	Description
<code>description</code>	yes	string		Human readable description of the level.
<code>scale_info</code>	no	map of <code>tosca.datatypes.nfv.ScaleInfo</code>		Represents for each aspect the scale level that corresponds to this instantiation level. <code>scale_info</code> shall be present if the VNF supports scaling.

### 6.2.18.3 Definition

The syntax of the `InstantiationLevel` data type shall comply with the following definition:

```
tosca.datatypes.nfv.InstantiationLevel:
  derived_from: toska.datatypes.Root
  description: Describes the scale level for each aspect that corresponds to a given
level of resources to be instantiated within a deployment flavour in term of the
number VNFC instances
  properties:
    description:
      type: string
      description: Human readable description of the level
      required: true
    scale_info:
      type: map # key: aspectId
      description: Represents for each aspect the scale level that corresponds to
this instantiation level. scale_info shall be present if the VNF supports scaling.
      required: false
    entry_schema:
      type: toska.datatypes.nfv.ScaleInfo
```

### 6.2.18.4 Examples

See clause A.6.

### 6.2.18.5 Additional Requirements

None.

## 6.2.19 toska.datatypes.nfv.VduLevel

### 6.2.19.1 Description

The VduLevel data type indicates for a given Vdu.Compute in a given level the number of instances to deploy. Table 6.2.19.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.19.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VduLevel
<b>Type Qualified Name</b>	toscanfv:VduLevel
<b>Type URI</b>	tosca.datatypes.nfv.VduLevel

### 6.2.19.2 Properties

The properties of the VduLevel data type shall comply with the provisions set out in table 6.2.19.2-1.

**Table 6.2.19.2-1: Properties**

Name	Required	Type	Constraints	Description
number_of_instances	yes	integer	greater_or_equal : 0	Number of instances of VNFC based on this VDU to deploy for this level.

### 6.2.19.3 Definition

The syntax of the VduLevel data type shall comply with the following definition:

```
tosca.datatypes.nfv.VduLevel:
  derived_from: toska.datatypes.Root
  description: Indicates for a given Vdu.Compute in a given level the number of
instances to deploy
  properties:
    number_of_instances:
      type: integer
      description: Number of instances of VNFC based on this VDU to deploy for this
level.
      required: true
      constraints:
        - greater_or_equal: 0
```

### 6.2.19.4 Examples

See clause A.6.

### 6.2.19.5 Additional Requirements

None.

## 6.2.20 toska.datatypes.nfv.VnfLcmOperationsConfiguration

### 6.2.20.1 Description

The VnfLcmOperationsConfiguration data type represents information to configure lifecycle management operations as specified in ETSI GS NFV-IFA 007 [i.1]. Each VNF LCM operations configuration property represents a container for all attributes that affect the invocation of the corresponding VNF Lifecycle Management operation. Table 6.2.20.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.20.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VnfLcmOperationsConfiguration
<b>Type Qualified Name</b>	toscanfv:VnfLcmOperationsConfiguration
<b>Type URI</b>	tosca.datatypes.nfv.VnfLcmOperationsConfiguration

### 6.2.20.2 Properties

The properties of the VnfLcmOperationsConfiguration data type shall comply with the provisions set out in table 6.2.20.2-1.

**Table 6.2.20.2-1: Properties**

Name	Required	Type	Constraints	Description
instantiate	no	tosca.datatypes.nfv.VnfInstantiateOperationConfiguration		Configuration parameters for the InstantiateVnf operation.
scale	no	tosca.datatypes.nfv.VnfScaleOperationConfiguration		Configuration parameters for the ScaleVnf operation.
scale_to_level	no	tosca.datatypes.nfv.VnfScaleToLevelOperationConfiguration		Configuration parameters for the ScaleVnfToLevel operation.
change_flavour	no	tosca.datatypes.nfv.VnfChangeFlavourOperationConfiguration		Configuration parameters for the changeVnfFlavourOpConfig operation.
heal	no	tosca.datatypes.nfv.VnfHealOperationConfiguration		Configuration parameters for the HealVnf operation.
terminate	no	tosca.datatypes.nfv.VnfTerminateOperationConfiguration		Configuration parameters for the TerminateVnf operation.
operate	no	tosca.datatypes.nfv.VnfOperateOperationConfiguration		Configuration parameters for the OperateVnf operation.
change_ext_connectivity	no	tosca.datatypes.nfv.VnfChangeExtConnectivityOperationConfiguration		Configuration parameters for the changeExtVnfConnectivityOpConfig operation.

### 6.2.20.3 Definition

The syntax of the VnfLcmOperationsConfiguration data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfLcmOperationsConfiguration:
  derived_from: toska.datatypes.Root
  description: Represents information to configure lifecycle management operations
  properties:
    instantiate:
      type: toska.datatypes.nfv.VnfInstantiateOperationConfiguration
      description: Configuration parameters for the InstantiateVnf operation
      required: false
    scale:
      type: toska.datatypes.nfv.VnfScaleOperationConfiguration
      description: Configuration parameters for the ScaleVnf operation
      required: false
    scale_to_level:
      type: toska.datatypes.nfv.VnfScaleToLevelOperationConfiguration
      description: Configuration parameters for the ScaleVnfToLevel operation
      required: false
    change_flavour:
      type: toska.datatypes.nfv.VnfChangeFlavourOperationConfiguration
      description: Configuration parameters for the changeVnfFlavourOpConfig
operation
      required: false
    heal:
      type: toska.datatypes.nfv.VnfHealOperationConfiguration
      description: Configuration parameters for the HealVnf operation
      required: false
    terminate:
      type: toska.datatypes.nfv.VnfTerminateOperationConfiguration
      description: Configuration parameters for the TerminateVnf operation
      required: false
    operate:
      type: toska.datatypes.nfv.VnfOperateOperationConfiguration
      description: Configuration parameters for the OperateVnf operation
      required: false
    change_ext_connectivity:
      type: toska.datatypes.nfv.VnfChangeExtConnectivityOperationConfiguration
      description: Configuration parameters for the changeExtVnfConnectivityOpConfig
operation
      required: false

```

### 6.2.20.4 Examples

None.

### 6.2.20.5 Additional Requirements

None.

## 6.2.21 toska.datatypes.nfv.VnfInstantiateOperationConfiguration

### 6.2.21.1 Description

The VnfInstantiateOperationConfiguration data type represents information that affect the invocation of the InstantiateVnf operation, as specified in ETSI GS NFV-IFA 011 [1]. Table 6.2.21.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.21.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VnfInstantiateOperationConfiguration
<b>Type Qualified Name</b>	toscanfv:VnfInstantiateOperationConfiguration
<b>Type URI</b>	tosca.datatypes.nfv.VnfInstantiateOperationConfiguration

### 6.2.21.2 Properties

None.

### 6.2.21.3 Definition

The syntax of the VnfInstantiateOperationConfiguration data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfInstantiateOperationConfiguration:
  derived_from: toasca.datatypes.Root
  description: represents information that affect the invocation of the
InstantiateVnf operation.
  # properties:
```

### 6.2.21.4 Examples

See clause 6.8.1.9.

### 6.2.21.5 Additional Requirements

None.

## 6.2.22 toasca.datatypes.nfv.VnfScaleOperationConfiguration

### 6.2.22.1 Description

VnfScaleOperationConfiguration represents information that affect the invocation of the ScaleVnf operation, as specified in ETSI GS NFV-IFA 011 [1]. Table 6.2.22.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.22.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VnfScaleOperationConfiguration
<b>Type Qualified Name</b>	toscanfv:VnfScaleOperationConfiguration
<b>Type URI</b>	tosca.datatypes.nfv.VnfScaleOperationConfiguration

### 6.2.22.2 Properties

The properties of the VnfScaleOperationConfiguration data type shall comply with the provisions set out in table 6.2.22.2-1.

**Table 6.2.22.2-1: Properties**

Name	Required	Type	Constraints	Description
scaling_by_more_than_one_step_supported	no	boolean		Signals whether passing a value larger than one in the numScalingSteps parameter of the ScaleVnf operation is supported by this VNF. Default is FALSE, i.e. "not supported".

### 6.2.22.3 Definition

The syntax of the VnfScaleOperationConfiguration data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfScaleOperationConfiguration:
  derived_from: toska.datatypes.Root
  description: Represents information that affect the invocation of the ScaleVnf
operation
  properties:
    scaling_by_more_than_one_step_supported:
      type: boolean
      description: Signals whether passing a value larger than one in the
numScalingSteps parameter of the ScaleVnf operation is supported by this VNF.
      required: false
      default: false

```

### 6.2.22.4 Examples

See clause 6.8.1.9.

### 6.2.22.5 Additional Requirements

None.

## 6.2.23 toska.datatypes.nfv.VnfScaleToLevelOperationConfiguration

### 6.2.23.1 Description

The VnfScaleToLevelOperationConfiguration data type represents information that affect the invocation of the ScaleVnfToLevel operation, as specified in ETSI GS NFV-IFA 011 [1]. Table 6.2.23.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.23.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VnfScaleToLevelOperationConfiguration
<b>Type Qualified Name</b>	toscanfv:VnfScaleToLevelOperationConfiguration
<b>Type URI</b>	tosca.datatypes.nfv.VnfScaleToLevelOperationConfiguration

### 6.2.23.2 Properties

The properties of the VnfScaleToLevelOperationConfiguration data type shall comply with the provisions set out in table 6.2.23.2-1.

**Table 6.2.23.2-1: Properties**

Name	Required	Type	Constraints	Description
arbitrary_target_levels_supported	yes	boolean		Signals whether scaling according to the parameter "scaleInfo" is supported by this VNF.

### 6.2.23.3 Definition

The syntax of the VnfScaleToLevelOperationConfiguration data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfScaleToLevelOperationConfiguration:
  derived_from: toska.datatypes.Root
  description: represents information that affect the invocation of the
ScaleVnfToLevel operation

```

```

properties:
  arbitrary_target_levels_supported:
    type: boolean
    description: Signals whether scaling according to the parameter "scaleInfo" is
supported by this VNF
    required: true

```

#### 6.2.23.4 Examples

See clause 6.8.1.9.

#### 6.2.23.5 Additional Requirements

None.

### 6.2.24 `tosca.datatypes.nfv.VnfHealOperationConfiguration`

#### 6.2.24.1 Description

The `VnfHealOperationConfiguration` data type represents information that affect the invocation of the `HealVnf` operation, as specified in ETSI GS NFV-IFA 011 [1]. Table 6.2.24.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.24.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>VnfHealOperationConfiguration</code>
<b>Type Qualified Name</b>	<code>toscanfv:VnfHealOperationConfiguration</code>
<b>Type URI</b>	<code>tosca.datatypes.nfv.VnfHealOperationConfiguration</code>

#### 6.2.24.2 Properties

The properties of the `VnfHealOperationConfiguration` data type shall comply with the provisions set out in table 6.2.24.2-1.

**Table 6.2.24.2-1: Properties**

Name	Required	Type	Constraints	Description
<code>causes</code>	no	list of string		Supported "cause" parameter values.

#### 6.2.24.3 Definition

The syntax of the `VnfHealOperationConfiguration` data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfHealOperationConfiguration:
  derived_from: toasca.datatypes.Root
  description: represents information that affect the invocation of the HealVnf
operation
  properties:
    causes:
      type: list
      description: Supported "cause" parameter values
      required: false
    entry_schema:
      type: string

```

#### 6.2.24.4 Examples

See clause 6.8.1.9.

#### 6.2.24.5 Additional Requirements

None.

### 6.2.25 `tosca.datatypes.nfv.VnfTerminateOperationConfiguration`

#### 6.2.25.1 Description

The `VnfTerminateOperationConfiguration` data type represents information that affect the invocation of the `TerminateVnf`, as specified in ETSI GS NFV-IFA 011 [1]. Table 6.2.25.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.25.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>VnfTerminateOperationConfiguration</code>
<b>Type Qualified Name</b>	<code>toscanfv:VnfTerminateOperationConfiguration</code>
<b>Type URI</b>	<code>tosca.datatypes.nfv.VnfTerminateOperationConfiguration</code>

#### 6.2.25.2 Properties

The properties of the `VnfTerminateOperationConfiguration` data type shall comply with the provisions set out in table 6.2.25.2-1.

**Table 6.2.25.2-1: Properties**

Name	Required	Type	Constraints	Description
<code>min_graceful_termination_timeout</code>	yes	scalar-unit.time		Minimum timeout value for graceful termination of a VNF instance.
<code>max_recommended_graceful_termination_timeout</code>	no	scalar-unit.time		Maximum recommended timeout value that can be needed to gracefully terminate a VNF instance of a particular type under certain conditions, such as maximum load condition. This is provided by VNF provider as information for the operator facilitating the selection of optimal timeout value. This value is not used as constraint.

#### 6.2.25.3 Definition

The syntax of the `VnfTerminateOperationConfiguration` data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfTerminateOperationConfiguration:
  derived_from: toasca.datatypes.Root
  description: represents information that affect the invocation of the TerminateVnf
  properties:
    min_graceful_termination_timeout:
      type: scalar-unit.time
      description: Minimum timeout value for graceful termination of a VNF instance
      required: true
    max_recommended_graceful_termination_timeout:
      type: scalar-unit.time
```



```

description: Maximum recommended timeout value that can be needed to
gracefully terminate a VNF instance of a particular type under certain conditions,
such as maximum load condition. This is provided by VNF provider as information for
the operator facilitating the selection of optimal timeout value. This value is not
used as constraint
required: false

```

#### 6.2.25.4 Examples

None.

#### 6.2.25.5 Additional Requirements

None.

### 6.2.26 `tosca.datatypes.nfv.VnfOperateOperationConfiguration`

#### 6.2.26.1 Description

The `VnfOperateOperationConfiguration` data type represents information that affect the invocation of the `OperateVnf` operation, as specified in ETSI GS NFV-IFA 011 [1]. Table 6.2.26.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.26.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>VnfOperateOperationConfiguration</code>
<b>Type Qualified Name</b>	<code>toscanfv:VnfOperateOperationConfiguration</code>
<b>Type URI</b>	<code>tosca.datatypes.nfv.VnfOperateOperationConfiguration</code>

#### 6.2.26.2 Properties

The properties of the `VnfOperateOperationConfiguration` data type shall comply with the provisions set out in table 6.2.26.2-1.

**Table 6.2.26.2-1: Properties**

Name	Required	Type	Constraints	Description
<code>min_graceful_stop_timeout</code>	yes	scalar-unit.time		Minimum timeout value for graceful stop of a VNF instance.
<code>max_recommended_graceful_stop_timeout</code>	no	scalar-unit.time		Maximum recommended timeout value that can be needed to gracefully stop a VNF instance of a particular type under certain conditions, such as maximum load condition. This is provided by VNF provider as information for the operator facilitating the selection of optimal timeout value. This value is not used as constraint.

#### 6.2.26.3 Definition

The syntax of the `VnfOperateOperationConfiguration` data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfOperateOperationConfiguration:
  derived_from: toska.datatypes.Root
  description: represents information that affect the invocation of the OperateVnf
operation
  properties:
    min_graceful_stop_timeout:
      type: scalar-unit.time

```

```

description: Minimum timeout value for graceful stop of a VNF instance
required: true
max_recommended_graceful_stop_timeout:
  type: scalar-unit.time
  description: Maximum recommended timeout value that can be needed to
  gracefully stop a VNF instance of a particular type under certain conditions, such as
  maximum load condition. This is provided by VNF provider as information for the
  operator facilitating the selection of optimal timeout value. This value is not used
  as constraint
  required: false

```

#### 6.2.26.4 Examples

None.

#### 6.2.26.5 Additional Requirements

None.

### 6.2.27 `tosca.datatypes.nfv.ScaleInfo`

#### 6.2.27.1 Description

The `scaleInfo` data type indicates for a given `scaleAspect` the corresponding `scaleLevel`. Table 6.2.27.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.27.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>ScaleInfo</code>
<b>Type Qualified Name</b>	<code>toscanfv:ScaleInfo</code>
<b>Type URI</b>	<code>tosca.datatypes.nfv.ScaleInfo</code>

#### 6.2.27.2 Properties

The properties of the `ScaleInfo` data type shall comply with the provisions set out in table 6.2.27.2-1.

**Table 6.2.27.2-1: Properties**

Name	Required	Type	Constraints	Description
<code>scale_level</code>	yes	integer	<code>greater_or_equal : 0</code>	The scale level for a particular aspect.

#### 6.2.27.3 Definition

The syntax of the `ScaleInfo` data type shall comply with the following definition:

```

tosca.datatypes.nfv.ScaleInfo:
  derived_from: toska.datatypes.Root
  description: Indicates for a given scaleAspect the corresponding scaleLevel
  properties:
    scale_level:
      type: integer
      description: The scale level for a particular aspect
      required: true
      constraints:
        - greater_or_equal: 0

```

### 6.2.27.4 Examples

See clause 6.8.1.9.

### 6.2.27.5 Additional Requirements

None.

## 6.2.28 `tosca.datatypes.nfv.ScalingAspect`

### 6.2.28.1 Description

The `ScalingAspect` data type describes the details of an aspect used for horizontal scaling. Table 6.2.28.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.28.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>ScalingAspect</code>
<b>Type Qualified Name</b>	<code>toscanfv:ScalingAspect</code>
<b>Type URI</b>	<code>tosca.datatypes.nfv.ScalingAspect</code>

### 6.2.28.2 Properties

The properties of the `ScalingAspect` data type shall comply with the provisions set out in table 6.2.28.2-1.

**Table 6.2.28.2-1: Properties**

Name	Required	Type	Constraints	Description
<code>name</code>	yes	string		Human readable name of the aspect.
<code>description</code>	yes	string		Human readable description of the aspect.
<code>max_scale_level</code>	yes	integer	<code>positiveInteger</code>	Total number of scaling steps that can be applied w.r.t. this aspect. The value of this property corresponds to the number of scaling steps can be applied to this aspect when scaling it from the minimum scale level (i.e. 0) to the maximum scale level defined by this property.
<code>step_deltas</code>	no	list of string		List of scaling deltas to be applied for the different subsequent scaling steps of this aspect. The first entry in the array shall correspond to the first scaling step (between scale levels 0 to 1) and the last entry in the array shall correspond to the last scaling step (between <code>maxScaleLevel-1</code> and <code>maxScaleLevel</code> ).

### 6.2.28.3 Definition

The syntax of the `ScalingAspect` data type shall comply with the following definition:

```
tosca.datatypes.nfv.ScalingAspect:
  derived_from: toasca.datatypes.Root
  properties:
    name:
      type: string
      required: true
    description:
      type: string
      required: true
```

```

max_scale_level:
  type: integer # positiveInteger
  required: true
  constraints:
    - greater_or_equal: 0
step_deltas:
  type: list
  required: false
entry_schema:
  type: string # Identifier

```

#### 6.2.28.4 Examples

See clause A.6.

#### 6.2.28.5 Additional Requirements

None.

### 6.2.29 `tosca.datatypes.nfv.LinkBitrateRequirements`

#### 6.2.29.1 Description

The `LinkBitrateRequirements` data type describes the requirements in terms of bitrate for a virtual link. Table 6.2.29.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.29.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>LinkBitrateRequirements</code>
<b>Type Qualified Name</b>	<code>toscanfv:LinkBitrateRequirements</code>
<b>Type URI</b>	<code>tosca.datatypes.nfv.LinkBitrateRequirements</code>

#### 6.2.29.2 Properties

The properties of the `LinkBitrateRequirements` data type shall comply with the provisions set out in table 6.2.29.2-1.

**Table 6.2.29.2-1: Properties**

Name	Required	Type	Constraints	Description
root	yes	integer	greater_or_equal: 0	Specifies the throughput requirement in bits per second of the link (e.g. bitrate of E-Line, root bitrate of E-Tree, aggregate capacity of E-LAN).
leaf	no	integer	greater_or_equal: 0	Specifies the throughput requirement in bits per second of leaf connections to the link when applicable to the connectivity type (e.g. for E-Tree and E-LAN branches).

### 6.2.29.3 Definition

The syntax of the LinkBitrateRequirements data type shall comply with the following definition:

```

tosca.datatypes.nfv.LinkBitrateRequirements:
  derived_from: toasca.datatypes.Root
  description: describes the requirements in terms of bitrate for a virtual link
  properties:
    root:
      type: integer # in bits per second
      description: Specifies the throughput requirement in bits per second of the
link (e.g. bitrate of E-Line, root bitrate of E-Tree, aggregate capacity of E-LAN).
      required: true
      constraints:
        - greater_or_equal: 0
    leaf:
      type: integer # in bits per second
      description: Specifies the throughput requirement in bits per second of leaf
connections to the link when applicable to the connectivity type (e.g. for E-Tree and
E LAN branches).
      required: false
      constraints:
        - greater_or_equal: 0

```

### 6.2.29.4 Examples

None.

### 6.2.29.5 Additional Requirements

None.

## 6.2.30 toasca.datatypes.nfv.Qos

### 6.2.30.1 Description

The QoS describes QoS data type a given VL used in a VNF deployment flavour. Table 6.2.30.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.30.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	Qos
<b>Type Qualified Name</b>	toscanfv:Qos
<b>Type URI</b>	tosca.datatypes.nfv.Qos

### 6.2.30.2 Properties

The properties of the Qos data type shall comply with the provisions set out in table 6.2.30.2-1.

**Table 6.2.30.2-1: Properties**

Name	Required	Type	Constraints	Description
latency	yes	scalar-unit.time	greater_than: 0 s	Specifies the maximum latency.
packet_delay_variation	yes	scalar-unit.time		Specifies the maximum jitter.
packet_loss_ratio	no	float	in_range: [0,1]	Specifies the maximum packet loss ratio.

### 6.2.30.3 Definition

The syntax of the Qos data type shall comply with the following definition:

```

tosca.datatypes.nfv.Qos:
  derived_from: toasca.datatypes.Root
  description: describes QoS data for a given VL used in a VNF deployment flavour
  properties:
    latency:
      type: scalar-unit.time #Number
      description: Specifies the maximum latency
      required: true
      constraints:
        - greater_than: 0 s
    packet_delay_variation:
      type: scalar-unit.time #Number
      description: Specifies the maximum jitter
      required: true
      constraints:
        - greater_or_equal: 0 s
    packet_loss_ratio:
      type: float
      description: Specifies the maximum packet loss ratio
      required: false
      constraints:
        - in_range: [ 0.0, 1.0 ]

```

### 6.2.30.4 Examples

None.

### 6.2.30.5 Additional Requirements

None.

## 6.2.31 toasca.datatypes.nfv.VnfConfigurableProperties

### 6.2.31.1 Description

The VnfConfigurableProperties data type describes configurable properties for a given VNF (e.g. related to auto scaling and auto healing).

Table 6.2.31.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.31.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VnfConfigurableProperties
<b>Type Qualified Name</b>	toscanfv:VnfConfigurableProperties
<b>Type URI</b>	tosca.datatypes.nfv.VnfConfigurableProperties

### 6.2.31.2 Properties

The properties of the VnfConfigurableProperties data type shall comply with the provisions set out in table 6.2.31.2-1.

**Table 6.2.31.2-1: Properties**

Name	Required	Type	Constraints	Description
is_autoscale_enabled	no	boolean		It permits to enable (TRUE)/disable (FALSE) the auto-scaling functionality. If the properties is not present for configuring, then VNF property is not supported
is_autoheal_enabled	no	boolean		It permits to enable (TRUE)/disable (FALSE) the auto-healing functionality. If the properties is not present for configuring, then VNF property is not supported
additional_configurable_property	no	tosca.datatypes.nfv.VnfAdditionalConfigurableProperties		It provides VNF specific configurable properties that can be modified using the ModifyVnfInfo operation

### 6.2.31.3 Definition

The syntax of the VnfConfigurableProperties data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfConfigurableProperties:
  derived_from: toska.datatypes.Root
  description: indicates configuration properties for a given VNF (e.g. related to
  auto scaling and auto healing).
  properties:
    is_autoscale_enabled:
      type: boolean
      description: It permits to enable (TRUE)/disable (FALSE) the auto-scaling
      functionality. If the properties is not present for configuring, then VNF property is
      not supported
      required: false
    is_autoheal_enabled:
      type: boolean
      description: It permits to enable (TRUE)/disable (FALSE) the auto-healing
      functionality. If the properties is not present for configuring, then VNF property is
      not supported
      required: false
    # additional_configurable_properties:
    # description: It provides VNF specific configurable properties that can be
    # modified using the ModifyVnfInfo operation
    # required: false
    # type: toska.datatypes.nfv.VnfAdditionalConfigurableProperties
  # derived types are expected to introduce
  # additional_configurable_properties with its type derived from
  # toska.datatypes.nfv.VnfAdditionalConfigurableProperties

```

### 6.2.31.4 Examples

Example definition of configurable properties.

```
tosca_definitions_version: tosca_simple_yaml_1_2

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      flavour_id:
        constraints:
          - valid_values: [ simple, complex ]
      configurable_properties:
        type: MyCompany.datatypes.nfv.VnfConfigurableProperties

data_types:
  Mycompany.datatypes.nfv.VnfConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfConfigurableProperties
    properties:
      additional_configurable_properties:
        type: MyCompany.datatypes.nfv.VnfAdditionalConfigurableProperties

  MyCompany.datatypes.nfv.VnfAdditionalConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfAdditionalConfigurableProperties
    properties:
      name_prefix_in_vim:
        type: string
        required: false
      dns_server:
        type: string
        required: true
```

### 6.2.31.5 Additional Requirements

None.

## 6.2.32 tosca.datatypes.nfv.VnfAdditionalConfigurableProperties

### 6.2.32.1 Description

The VnfAdditionalConfigurableProperties data type is an empty base type for deriving data types for describing additional configurable properties for a given VNF. Table 6.2.32.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.32.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VnfAdditionalConfigurableProperties
<b>Type Qualified Name</b>	toscanfv:VnfAdditionalConfigurableProperties
<b>Type URI</b>	tosca.datatypes.nfv.VnfAdditionalConfigurableProperties

### 6.2.32.2 Properties

None.



### 6.2.32.3 Definition

The syntax of the `VnfAdditionalConfigurableProperties` data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfAdditionalConfigurableProperties:
  derived_from: toska.datatypes.Root
  description: is an empty base type for deriving data types for describing
  additional configurable properties for a given VNF
```

### 6.2.32.4 Examples

See clause 6.2.31.4.

### 6.2.32.5 Additional Requirements

None.

## 6.2.33 toska.datatypes.nfv.VnfInfoModifiableAttributes

### 6.2.33.1 Description

The `VnfInfoModifiableAttributes` data type describes VNF-specific extension and metadata for a given VNF. Table 6.2.33.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.33.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>VnfInfoModifiableAttributes</code>
<b>Type Qualified Name</b>	<code>toscanfv:VnfInfoModifiableAttributes</code>
<b>Type URI</b>	<code>tosca.datatypes.nfv.VnfInfoModifiableAttributes</code>

### 6.2.33.2 Properties

The properties of the `VnfInfoModifiableAttributes` data type shall comply with the provisions set out in table 6.2.33.2-1.

**Table 6.2.33.2-1: Properties**

Name	Required	Type	Constraints	Description
extensions	no	<code>tosca.datatypes.nfv.VnfInfoModifiableAttributesExtensions</code>		"Extension" properties of <code>VnfInfo</code> that are writeable.
metadata	no	<code>tosca.datatypes.nfv.VnfInfoModifiableAttributesMetadata</code>		"Metadata" properties of <code>VnfInfo</code> that are writeable.

### 6.2.33.3 Definition

The syntax of the `VnfInfoModifiableAttributes` data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfInfoModifiableAttributes:
  derived_from: toska.datatypes.Root
  description: Describes VNF-specific extension and metadata for a given VNF
  #properties:
    #extensions:
      #type: toska.datatypes.nfv.VnfInfoModifiableAttributesExtensions
      #description: "Extension" properties of VnfInfo that are writeable
      #required: false
```

```

# derived types are expected to introduce
# extensions with its type derived from
# toska.datatypes.nfv.VnfInfoModifiableAttributesExtensions
#metadata:
#type: toska.datatypes.nfv.VnfInfoModifiableAttributesMetadata
#description: "Metadata" properties of VnfInfo that are writeable
#required: false
# derived types are expected to introduce
# metadata with its type derived from
# toska.datatypes.nfv.VnfInfoModifiableAttributesMetadata

```

#### 6.2.33.4 Examples

See clause 6.8.1.9.

#### 6.2.33.5 Additional Requirements

None.

### 6.2.34 toska.datatypes.nfv.VnfInfoModifiableAttributesExtensions

#### 6.2.34.1 Description

The VnfInfoModifiableAttributesExtensions data type is an empty base type for deriving data types for describing VNF-specific extension. Table 6.2.34.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.34.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VnfInfoModifiableAttributesExtensions
<b>Type Qualified Name</b>	toscanfv:VnfInfoModifiableAttributesExtensions
<b>Type URI</b>	toska.datatypes.nfv.VnfInfoModifiableAttributesExtensions

#### 6.2.34.2 Properties

None.

#### 6.2.34.3 Definition

The syntax of the VnfInfoModifiableAttributesExtensions data type shall comply with the following definition:

```

toska.datatypes.nfv.VnfInfoModifiableAttributesExtensions:
  derived_from: toska.datatypes.Root
  description: is an empty base type for deriving data types for describing VNF-
specific extension

```

#### 6.2.34.4 Examples

See clause 6.8.1.9.

#### 6.2.34.5 Additional Requirements

None.

## 6.2.35 tosca.datatypes.nfv.VnfInfoModifiableAttributesMetadata

### 6.2.35.1 Description

The VnfInfoModifiableAttributesMetadata data type is an empty base type for deriving data types for describing VNF-specific metadata. Table 6.2.35.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.35.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VnfInfoModifiableAttributesMetadata
<b>Type Qualified Name</b>	toscanfv:VnfInfoModifiableAttributesMetadata
<b>Type URI</b>	tosca.datatypes.nfv.VnfInfoModifiableAttributesMetadata

### 6.2.35.2 Properties

None.

### 6.2.35.3 Definition

The syntax of the VnfInfoModifiableAttributesMetadata data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfInfoModifiableAttributesMetadata:
  derived_from: tosca.datatypes.Root
  description: is an empty base type for deriving data types for describing VNF-
specific metadata
```

### 6.2.35.4 Examples

Example metadata definition:

```
tosca_definitions_version: tosca_simple_yaml_1_2

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      flavour_id:
        constraints:
          - valid_values: [ simple, complex ]
      modifiable_attributes:
        type: mycompany.datatypes.nfv.VnfInfoModifiableAttributes

data_types:
  mycompany.datatypes.nfv.VnfInfoModifiableAttributes:
    derived_from: tosca.datatypes.nfv.VnfInfoModifiableAttributes
    properties:
      metadata:
        type: mycompany.datatypes.nfv.VnfInfoModifiableAttributesMetadata

  mycompany.datatypes.nfv.VnfInfoModifiableAttributesMetadata:
    derived_from: tosca.datatypes.nfv.VnfInfoModifiableAttributesMetadata
    properties:
      metadata_key_1:
        type: string
        required: false
      metadata_key_2:
```

```

type: string
required: false

```

### 6.2.35.5 Additional Requirements

None.

## 6.2.36 tosca.datatypes.nfv.CpProtocolData

### 6.2.36.1 Description

The CpProtocolData data type describes and associates the protocol layer that a CP uses together with other protocol and connection point information. Table 6.2.36.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.36.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	CpProtocolData
<b>Type Qualified Name</b>	toscanfv:CpProtocolData
<b>Type URI</b>	tosca.datatypes.nfv.CpProtocolData

### 6.2.36.2 Properties

The properties of the CpProtocolData data type shall comply with the provisions set out in table 6.2.36.2-1.

**Table 6.2.36.2-1: Properties**

Name	Required	Type	Constraints	Description
associated_layer_protocol	yes	string	Valid values: ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire	One of the values of the property layer_protocols of the CP.
address_data	no	list of tosca.datatypes.nfv.AddressData		Provides information on the addresses to be assigned to the CP.

### 6.2.36.3 Definition

The syntax of the CpProtocolData data type shall comply with the following definition:

```

tosca.datatypes.nfv.CpProtocolData:
  derived_from: tosca.datatypes.Root
  description: Describes and associates the protocol layer that a CP uses together
with other protocol and connection point information
.
  properties:
    associated_layer_protocol:
      type: string
      required: true
      description: One of the values of the property layer_protocols of the CP
      constraints:
        - valid_values: [ ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire ]
    address_data:
      type: list
      description: Provides information on the addresses to be assigned to the CP
  entry_schema:
    type: tosca.datatypes.nfv.AddressData
    required: false

```

### 6.2.36.4 Examples

None.

### 6.2.36.5 Additional Requirements

None.

## 6.2.37 `tosca.datatypes.nfv.LogicalNodeData`

### 6.2.37.1 Description

The `LogicalNodeData` data type describes compute, memory and I/O requirements associated with a particular VDU. Table 6.2.37.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.37.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>LogicalNodeData</code>
<b>Type Qualified Name</b>	<code>toscanfv:LogicalNodeData</code>
<b>Type URI</b>	<code>tosca.datatypes.nfv.LogicalNodeData</code>

### 6.2.37.2 Properties

The properties of the `LogicalNodeData` data type shall comply with the provisions set out in table 6.2.37.2-1.

**Table 6.2.37.2-1: Properties**

Name	Required	Type	Constraints	Description
<code>logical_node_requirements</code>	no	map of string		<p>The logical node-level compute, memory and I/O requirements. A map of strings that contains a set of key-value pairs that describes hardware platform specific deployment requirements, including the number of CPU cores on this logical node, a memory configuration specific to a logical node or a requirement related to the association of an I/O device with the logical node.</p> <p>More information regarding the usage of this property is available at: <a href="https://register.etsi.org">https://register.etsi.org</a></p>

### 6.2.37.3 Definition

The syntax of the `LogicalNodeData` data type shall comply with the following definition:

```
tosca.datatypes.nfv.LogicalNodeData:
  derived_from: toska.datatypes.Root
  description: Describes compute, memory and I/O requirements associated with a
  particular VDU.
  properties:
    logical_node_requirements:
      type: map
      description: The logical node-level compute, memory and I/O requirements. A
  map of strings that contains a set of key-value pairs that describes hardware
  platform specific deployment requirements, including the number of CPU cores on this
  logical node, a memory configuration specific to a logical node or a requirement
  related to the association of an I/O device with the logical node.
```

```

required: false
entry_schema:
  type: string

```

#### 6.2.37.4 Examples

None.

#### 6.2.37.5 Additional Requirements

None.

### 6.2.38 `tosca.datatypes.nfv.SwImageData`

#### 6.2.38.1 Description

The `SwImageData` data type describes information related to a software image artifact. Table 6.2.38.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.38.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>SwImageData</code>
<b>Type Qualified Name</b>	<code>tosca.nfv:SwImageData</code>
<b>Type URI</b>	<code>tosca.datatypes.nfv.SwImageData</code>

#### 6.2.38.2 Properties

The properties of the `SwImageData` data type shall comply with the provisions set out in table 6.2.38.2-1.

**Table 6.2.38.2-1: Properties**

Name	Required	Type	Constraints	Description
<code>name</code>	yes	string		Name of this software image.
<code>version</code>	yes	string		Version of this software image.
<code>checksum</code>	yes	<code>tosca.datatypes.nfv.ChecksumData</code>		Checksum of the software image file.
<code>container_format</code>	yes	string	Valid values: [aki, ami, ari, bare, docker, ova, ovf]	The container format describes the container file format in which software image is provided. Description of valid values: aki: a kernel image ami: a machine image ari: a ramdisk image bare: the image does not have a container or metadata envelope docker: docker container format ova: OVF package in a tarfile ovf: OVF container format  Future versions of the present document may extend the list of possible values.

Name	Required	Type	Constraints	Description
disk_format	yes	string	Valid values: [aki, ami, ari, iso, qcow2, raw, vdi, vhd, vhdx, vmdk]	The disk format of a software image is the format of the underlying disk image. Description of valid values: aki: a kernel image ami: a machine image ari: a ramdisk image iso: an archive format for the data contents of an optical disc, such as CD-ROM qcow2: a common disk image format, which can expand dynamically and supports copy on write raw: an unstructured disk image format vdi: a common disk image format vhd: a common disk image format vhdx: enhanced version of VHD format vmdk: a common disk image format Future versions of the present document may extend the list of possible values.
min_disk	yes	scalar-unit.size	greater_or_equal: 0 B	The minimal disk size requirement for this software image.
min_ram	no	scalar-unit.size	greater_or_equal: 0 B	The minimal RAM requirement for this software image.
size	yes	scalar-unit.size		The size of this software image.
operating_system	no	string		Identifies the operating system used in the software image.
supported_virtualisation_environments	no	list of string		Identifies the virtualisation environments (e.g. hypervisor) compatible with this software image.

### 6.2.38.3 Definition

The syntax of the SwImageData data type shall comply with the following definition:

```

tosca.datatypes.nfv.SwImageData:
  derived_from: tosca.datatypes.Root
  description: describes information related to a software image artifact
  properties: # in SOL001 v0.8.0: "properties or metadata:"
    name:
      type: string
      description: Name of this software image
      required: true
    version:
      type: string
      description: Version of this software image
      required: true
    checksum:
      type: tosca.datatypes.nfv.ChecksumData
      description: Checksum of the software image file
      required: true
    container_format:
      type: string
      description: The container format describes the container file format in which
software image is provided
      required: true
      constraints:
        - valid_values: [ aki, ami, ari, bare, docker, ova, ovf ]
    disk_format:
      type: string
      description: The disk format of a software image is the format of the
underlying disk image

```

```

    required: true
    constraints:
      - valid_values: [ aki, ami, ari, iso, qcow2, raw, vdi, vhd, vhdx, vmdk ]
  min_disk:
    type: scalar-unit.size # Number
    description: The minimal disk size requirement for this software image
    required: true
    constraints:
      - greater_or_equal: 0 B
  min_ram:
    type: scalar-unit.size # Number
    description: The minimal RAM requirement for this software image
    required: false
    constraints:
      - greater_or_equal: 0 B
  size:
    type: scalar-unit.size # Number
    description: The size of this software image
    required: true
  operating_system:
    type: string
    description: Identifies the operating system used in the software image
    required: false
  supported_virtualisation_environments:
    type: list
    description: Identifies the virtualisation environments (e.g. hypervisor)
    compatible with this software image
    required: false
  entry_schema:
    type: string

```

#### 6.2.38.4 Examples

None.

#### 6.2.38.5 Additional Requirements

None.

### 6.2.39 `tosca.datatypes.nfv.VirtualBlockStorageData`

#### 6.2.39.1 Description

The `VirtualBlockStorageData` data type describes block storage requirements associated with compute resources in a particular VDU, either as a local disk or as virtual attached storage. Table 6.2.39.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.39.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>VirtualBlockStorageData</code>
<b>Type Qualified Name</b>	<code>toscanfv:VirtualBlockStorageData</code>
<b>Type URI</b>	<code>tosca.datatypes.nfv.VirtualBlockStorageData</code>



### 6.2.39.2 Properties

The properties of the VirtualBlockStorageData data type shall comply with the provisions set out in table 6.2.39.2-1.

**Table 6.2.39.2-1: Properties**

Name	Required	Type	Constraints	Description
size_of_storage	yes	scalar-unit.size	greater_or_equal:0 B	Size of virtualised storage resource
vdu_storage_requirements	no	map of string		The hardware platform specific storage requirements. A map of strings that contains a set of key-value pairs that represents the hardware platform specific storage deployment requirements  More information regarding the usage of this property is available at: <a href="https://register.etsi.org">https://register.etsi.org</a>
rdma_enabled	no	boolean	default: false	Indicate if the storage support RDMA

### 6.2.39.3 Definition

The syntax of the VirtualBlockStorageData data type shall comply with the following definition:

```

tosca.datatypes.nfv.VirtualBlockStorageData:
  derived_from: toska.datatypes.Root
  description: VirtualBlockStorageData describes block storage requirements
  associated with compute resources in a particular VDU, either as a local disk or as
  virtual attached storage
  properties:
    size_of_storage:
      type: scalar-unit.size
      description: Size of virtualised storage resource
      required: true
      constraints:
        - greater_or_equal: 0 B
    vdu_storage_requirements:
      type: map
      description: The hardware platform specific storage requirements. A map of
      strings that contains a set of key-value pairs that represents the hardware platform
      specific storage deployment requirements
      required: false
      entry_schema:
        type: string
    rdma_enabled:
      type: boolean
      description: Indicates if the storage support RDMA
      required: false
      default: false

```

### 6.2.39.4 Examples

None.

### 6.2.39.5 Additional Requirements

None.

## 6.2.40 tosca.datatypes.nfv.VirtualObjectStorageData

### 6.2.40.1 Description

The VirtualObjectStorageData data type describes object storage requirements associated with compute resources in a particular VDU. Table 6.2.40.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.40.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VirtualObjectStorageData
<b>Type Qualified Name</b>	toscanfv:VirtualObjectStorageData
<b>Type URI</b>	tosca.datatypes.nfv.VirtualObjectStorageData

### 6.2.40.2 Properties

The properties of the VirtualObjectStorageData data type shall comply with the provisions set out in table 6.2.40.2-1.

**Table 6.2.40.2-1: Properties**

Name	Required	Type	Constraints	Description
max_size_of_storage	no	scalar-unit.size	greater_or_equal:0 B	Maximum size of virtualised storage resource

### 6.2.40.3 Definition

The syntax of the VirtualObjectStorageData data type shall comply with the following definition:

```
tosca.datatypes.nfv.VirtualObjectStorageData:
  derived_from: tosca.datatypes.Root
  description: VirtualObjectStorageData describes object storage requirements
  associated with compute resources in a particular VDU
  properties:
    max_size_of_storage:
      type: scalar-unit.size
      description: Maximum size of virtualized storage resource
      required: false
      constraints:
        - greater_or_equal: 0 B
```

### 6.2.40.4 Examples

None.

### 6.2.40.5 Additional Requirements

None.

## 6.2.41 tosca.datatypes.nfv.VirtualFileStorageData

### 6.2.41.1 Description

The VirtualObjectFileData data type describes file storage requirements associated with compute resources in a particular VDU. Table 6.2.41.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.41.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VirtualObjectFileData
<b>Type Qualified Name</b>	toscanfv:VirtualFileStorageData
<b>Type URI</b>	tosca.datatypes.nfv.VirtualFileStorageData

### 6.2.41.2 Properties

The properties of the VirtualFileStorageData data type shall comply with the provisions set out in table 6.2.41.2-1.

**Table 6.2.41.2-1: Properties**

Name	Required	Type	Constraints	Description
size_of_storage	yes	scalar-unit.size	greater_or_equal:0 B	Size of virtualised storage resource
file_system_protocol	yes	string		The shared file system protocol (e.g. NFS, CIFS).

### 6.2.41.3 Definition

The syntax of the VirtualFileStorageData data type shall comply with the following definition:

```

tosca.datatypes.nfv.VirtualFileStorageData:
  derived_from: toscanfv:VirtualFileStorageData
  description: VirtualFileStorageData describes file storage requirements
  associated with compute resources in a particular VDU
  properties:
    size_of_storage:
      type: scalar-unit.size
      description: Size of virtualized storage resource
      required: true
      constraints:
        - greater_or_equal: 0 B
    file_system_protocol:
      type: string
      description: The shared file system protocol (e.g. NFS, CIFS)
      required: true

```

### 6.2.41.4 Examples

None.

### 6.2.41.5 Additional Requirements

None.

## 6.2.42 toscanfv:VirtualLinkBitrateLevel

### 6.2.42.1 Description

The VirtualLinkBitrateLevel data type describes bitrate requirements applicable to the virtual link instantiated from a particular VnfVirtualLink. Table 6.2.42.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.42.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VirtualLinkBitrateLevel
<b>Type Qualified Name</b>	toscanfv:VirtualLinkBitrateLevel
<b>Type URI</b>	tosca.datatypes.nfv.VirtualLinkBitrateLevel

### 6.2.42.2 Properties

The properties of the VirtualLinkBitrateLevel data type shall comply with the provisions set out in table 6.2.42.2-1.

**Table 6.2.42.2-1: Properties**

Name	Required	Type	Constraints	Description
bitrate_requirements	yes	tosca.datatypes.nfv.LinkBitrateRequirements		Virtual link bitrate requirements for an instantiation level or bitrate delta for a scaling step.

### 6.2.42.3 Definition

The syntax of the VirtualLinkBitrateLevel data type shall comply with the following definition:

```
tosca.datatypes.nfv.VirtualLinkBitrateLevel:
  derived_from: toscanfv:VirtualLinkBitrateLevel
  description: Describes bitrate requirements applicable to the virtual link
  instantiated from a particular VnfVirtualLink
  properties:
    bitrate_requirements:
      type: toscanfv.LinkBitrateRequirements
      description: Virtual link bitrate requirements for an instantiation level or
      bitrate delta for a scaling step
      required: true
```

### 6.2.42.4 Examples

See clause A.6.

### 6.2.42.5 Additional Requirements

None.

## 6.2.43 toscanfv.VnfOperationAdditionalParameters

### 6.2.43.1 Description

The VnfOperationAdditionalParameters data type is an empty base type for deriving data type for describing VNF-specific parameters to be passed when invoking lifecycle management operations as specified in ETSI GS NFV-IFA 011 [1]. Table 6.2.43.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.43.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VnfOperationAdditionalParameters
<b>Type Qualified Name</b>	toscanfv:VnfOperationAdditionalParameters
<b>Type URI</b>	tosca.datatypes.nfv.VnfOperationAdditionalParameters

### 6.2.43.2 Properties

None.

### 6.2.43.3 Definition

The syntax of the VnfOperationAdditionalParameters data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfOperationAdditionalParameters:
  derived_from: toska.datatypes.Root
  description: Is an empty base type for deriving data type for describing VNF-
specific parameters to be passed when invoking lifecycle management operations
  # properties:
```

### 6.2.43.4 Examples

```
tosca_definitions_version: toska_simple_yaml_1_2

node_types:
  MyCompany.nodes.nfv.SunshineDB.1_0.1_0:
    derived_from: toska.nodes.nfv.VNF
    properties:
      ..
    interfaces:
      Vnflcm:
        instantiate:
          inputs:
            additional_parameters:
              type: MyCompany.datatypes.nfv.VnfInstantiateAdditionalParameters

data_types:
  MyCompany.datatypes.nfv.VnfInstantiateAdditionalParameters:
    derived_from: toska.datatypes.nfv.VnfOperationAdditionalParameters
    properties:
      parameter_1:
        type: string
        required: true
      parameter_2:
        type: string
        required: false
        default: value_2
```

### 6.2.43.5 Additional Requirements

None.

## 6.2.44 toska.datatypes.nfv.VnfChangeFlavourOperationConfiguration

### 6.2.44.1 Description

The VnfChangeFlavourOperationConfiguration data type represents information that affect the invocation of the ChangeVnfFlavour operation, as specified in ETSI GS NFV-IFA 011 [1]. Table 6.2.44.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.44.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VnfChangeFlavourOperationConfiguration
<b>Type Qualified Name</b>	toscanfv:VnfChangeFlavourOperationConfiguration
<b>Type URI</b>	tosca.datatypes.nfv.VnfChangeFlavourOperationConfiguration

#### 6.2.44.2 Properties

None.

#### 6.2.44.3 Definition

The syntax of the VnfChangeFlavourOperationConfiguration data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfChangeFlavourOperationConfiguration:
  derived_from: toasca.datatypes.Root
  description: represents information that affect the invocation of the
ChangeVnfFlavour operation
  # properties:
```

#### 6.2.44.4 Examples

None.

#### 6.2.44.5 Additional Requirements

None.

### 6.2.45 toasca.datatypes.nfv.VnfChangeExtConnectivityOperation Configuration

#### 6.2.45.1 Description

The VnfChangeExtConnectivityOperationConfiguration data type represents information that affect the invocation of the ChangeExtVnfConnectivity operation, as specified in ETSI GS NFV-IFA 011 [1]. Table 6.2.45.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.45.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VnfChangeExtConnectivityOperationConfiguration
<b>Type Qualified Name</b>	toscanfv:VnfChangeExtConnectivityOperationConfiguration
<b>Type URI</b>	tosca.datatypes.nfv.VnfChangeExtConnectivityOperationConfiguration

#### 6.2.45.2 Properties

None.

#### 6.2.45.3 Definition

The syntax of the VnfChangeExtConnectivityOperationConfiguration data type shall comply with the following definition:

```
tosca.datatypes.nfv.VnfChangeExtConnectivityOperationConfiguration:
  derived_from: toasca.datatypes.Root
  description: represents information that affect the invocation of the
ChangeExtVnfConnectivity operation
```

# properties:

#### 6.2.45.4 Examples

None.

#### 6.2.45.5 Additional Requirements

None.

### 6.2.46 `tosca.datatypes.nfv.VnfMonitoringParameter`

#### 6.2.46.1 Description

This data type provides information on virtualised resource related performance metrics applicable to VNF. Table 6.2.46.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.46.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VnfMonitoringParameter
<b>Type Qualified Name</b>	toscanfv:VnfMonitoringParameter
<b>Type URI</b>	tosca.datatypes.nfv.VnfMonitoringParameter

#### 6.2.46.2 Properties

The properties of the VnfMonitoringParameter data type shall comply with the provisions set out in table 6.2.46.2-1.

**Table 6.2.46.2-1: Properties**

Name	Required	Type	Constraints	Description
name	no	string		Human readable name of the monitoring parameter.
performance_metric	yes	string	valid values: v_cpu_usage_mean_vnf v_cpu_usage_peak_vnf v_memory_usage_mean_vnf v_memory_usage_peak_vnf v_disk_usage_mean_vnf v_disk_usage_peak_vnf byte_incoming_vnf_ext_cp byte_outgoing_vnf_ext_cp packet_incoming_vnf_ext_cp packet_outgoing_vnf_ext_cp	Identifies the performance metric, according to ETSI GS NFV-IFA 027 [7].  Performance metric values shall be set to the corresponding measurement name defined in clause 7.2 of ETSI GS NFV-IFA 027 [7], without appending a sub-counter.
collection_period	no	scalar-unit.time		Describes the periodicity at which to collect the performance information.

### 6.2.46.3 Definition

The syntax of the VnfMonitoringParameter data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfMonitoringParameter:
  derived_from: toska.datatypes.Root
  description: Represents information on virtualised resource related performance
  metrics applicable to the VNF.
  properties:
    name:
      type: string
      description: Human readable name of the monitoring parameter
      required: true
    performance_metric:
      type: string
      description: Identifies the performance metric, according to ETSI GS NFV-IFA 027.
      required: true
      constraints:
        - valid_values: [ v_cpu_usage_mean_vnf, v_cpu_usage_peak_vnf,
v_memory_usage_mean_vnf, v_memory_usage_peak_vnf, v_disk_usage_mean_vnf,
v_disk_usage_peak_vnf, byte_incoming_vnf_ext_cp, byte_outgoing_vnf_ext_cp,
packet_incoming_vnf_ext_cp, packet_outgoing_vnf_ext_cp ]
    collection_period:
      type: scalar-unit.time
      description: Describes the periodicity at which to collect the performance
  information.
      required: false
      constraints:
        - greater_than: 0 s

```

### 6.2.46.4 Examples

None.

### 6.2.46.5 Additional Requirements

None.

## 6.2.47 toska.datatypes.nfv.VnfcMonitoringParameter

### 6.2.47.1 Description

This data type provides information on virtualised resource related performance metrics applicable to a VNFC. Table 6.2.47.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.47.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VnfcMonitoringParameter
<b>Type Qualified Name</b>	toscanfv:VnfcMonitoringParameter
<b>Type URI</b>	tosca.datatypes.nfv.VnfcMonitoringParameter

### 6.2.47.2 Properties

The properties of the VnfcMonitoringParameter data type shall comply with the provisions set out in table 6.2.47.2-1.



Table 6.2.47.2-1: Properties

Name	Required	Type	Constraints	Description
name	no	string		Human readable name of the monitoring parameter.
performance_metric	yes	string	valid values: v_cpu_usage_mean_vnf v_cpu_usage_peak_vnf v_memory_usage_mean_vnf v_memory_usage_peak_vnf v_disk_usage_mean_vnf v_disk_usage_peak_vnf byte_incoming_vnf_int_cp byte_outgoing_vnf_int_cp packet_incoming_vnf_int_cp packet_outgoing_vnf_int_cp	Identifies the performance metric, according to ETSI GS NFV-IFA 027 [7].  Performance metric values shall be set to the corresponding measurement name defined in clause 7.2 of ETSI GS NFV-IFA 027 [7], without appending a sub-counter.
collection_period	no	scalar-unit.time		Describes the periodicity at which to collect the performance information.

### 6.2.47.3 Definition

The syntax of the VnfcMonitoringParameter data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfcMonitoringParameter:
  derived_from: toasca.datatypes.Root
  description: Represents information on virtualised resource related performance
metrics applicable to the VNF.
  properties:
    name:
      type: string
      description: Human readable name of the monitoring parameter
      required: true
    performance_metric:
      type: string
      description: Identifies the performance metric, according to ETSI GS NFV-IFA 027.
      required: true
    constraints:
      - valid_values: [ v_cpu_usage_mean_vnf, v_cpu_usage_peak_vnf,
v_memory_usage_mean_vnf, v_memory_usage_peak_vnf, v_disk_usage_mean_vnf,
v_disk_usage_peak_vnf, byte_incoming_vnf_int_cp, byte_outgoing_vnf_int_cp,
packet_incoming_vnf_int_cp, packet_outgoing_vnf_int_cp ]
    collection_period:
      type: scalar-unit.time
      description: Describes the periodicity at which to collect the performance
information.
      required: false
      constraints:
        - greater_than: 0 s

```

### 6.2.47.4 Examples

None.

### 6.2.47.5 Additional Requirements

None.

## 6.2.48 `tosca.datatypes.nfv.VirtualLinkMonitoringParameter`

### 6.2.48.1 Description

This data type provides information on virtualised resource related performance metrics. Table 6.2.48.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.48.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>VirtualLinkMonitoringParameter</code>
<b>Type Qualified Name</b>	<code>toscanfv: VirtualLinkMonitoringParameter</code>
<b>Type URI</b>	<code>tosca.datatypes.nfv.VirtualLinkMonitoringParameter</code>

### 6.2.48.2 Properties

The properties of the `VirtualLinkMonitoringParameter` data type shall comply with the provisions set out in table 6.2.48.2-1.

**Table 6.2.48.2-1: Properties**

Name	Required	Type	Constraints	Description
<code>name</code>	no	string		Human readable name of the monitoring parameter.
<code>performance_metric</code>	yes	string	valid values: <code>packet_outgoing_virtual_link</code> <code>packet_incoming_virtual_link</code>	Identifies a performance metric derived from those defined in ETSI GS NFV-IFA 027 [7].  The <code>packet_outgoing_virtual_link</code> and <code>packet_incoming_virtual_link</code> metrics shall be obtained by aggregation the <code>PacketOutgoing</code> and <code>PacketIncoming</code> measurements defined in clause 7.1 of ETSI GS NFV-IFA 027 [7] of all virtual link ports attached to the virtual link to which the metrics apply.
<code>collection_period</code>	no	scalar-unit.time		Describes the periodicity at which to collect the performance information.

### 6.2.48.3 Definition

The syntax of the `VirtualLinkMonitoringParameter` data type shall comply with the following definition:

```
tosca.datatypes.nfv.VirtualLinkMonitoringParameter:
  derived_from: toska.datatypes.Root
  description: Represents information on virtualised resource related performance
  metrics applicable to the VNF.
  properties:
    name:
      type: string
      description: Human readable name of the monitoring parameter
      required: true
    performance_metric:
      type: string
```

```

description: Identifies a performance metric derived from those defined in ETSI GS
NFV-IFA 027.The packetOutgoingVirtualLink and packetIncomingVirtualLink metrics shall
be obtained by aggregation the PacketOutgoing and PacketIncoming measurements defined
in clause 7.1 of ETSI GS NFV-IFA 027 of all virtual link ports attached to the virtual
link to which the metrics apply.
required: true
constraints:
  - valid_values: [ packet_outgoing_virtual_link, packet_incoming_virtual_link ]
collection_period:
type: scalar-unit.time
description: Describes the periodicity at which to collect the performance
information.
required: false
constraints:
  - greater_than: 0 s

```

#### 6.2.48.4 Examples

None.

#### 6.2.48.5 Additional Requirements

None.

### 6.2.49 `tosca.datatypes.nfv.InterfaceDetails`

#### 6.2.49.1 Description

The `InterfaceDetails` data type describes information used to access an interface exposed by a VNF. It corresponds to the `interfaceDetails` attribute of the `VnfInterfaceDetails` information element defined in ETSI GS NFV-IFA 011 [1]. Table 6.2.49.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.49.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>InterfaceDetails</code>
<b>Type Qualified Name</b>	<code>toscanfv:InterfaceDetails</code>
<b>Type URI</b>	<code>tosca.datatypes.nfv.InterfaceDetails</code>

#### 6.2.49.2 Properties

The properties of the `InterfaceDetails` data type shall comply with the provisions set out in table 6.2.49.2-1.

**Table 6.2.49.2-1: Properties**

Name	Required	Type	Constraints	Description
<code>uri_components</code>	no	<code>tosca.datatypes.nfv.UriComponents</code>		Provides components to build a Uniform Resource Identifier (URI) where to access the interface end point.
<code>interface_specific_data</code>	no	map of string		Provides additional details that are specific to the type of interface considered.

### 6.2.49.3 Definition

The syntax of the InterfaceDetails data type shall comply with the following definition:

```

tosca.datatypes.nfv.InterfaceDetails:
  derived_from: toska.datatypes.Root
  description: information used to access an interface exposed by a VNF
  properties:
    uri_components:
      type: toska.datatypes.nfv.UriComponents
      description: Provides components to build a Uniform Ressource Identifier (URI)
where to access the interface end point.
      required: false
    interface_specific_data:
      type: map
      description: Provides additional details that are specific to the type of
interface considered.
      required: false
    entry_schema:
      type: string

```

### 6.2.49.4 Examples

See clause 6.10.12.

### 6.2.49.5 Additional Requirements

None.

## 6.2.50 toska.datatypes.nfv.UriComponents

### 6.2.50.1 Description

The UriComponents data type describes information used to build a URI that complies with IETF RFC 3986 [8]. Table 6.2.50.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.50.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	UriComponents
<b>Type Qualified Name</b>	toscanfv:UriComponents
<b>Type URI</b>	tosca.datatypes.nfv.UriComponents

### 6.2.50.2 Properties

The properties of the UriComponents data type shall comply with the provisions set out in table 6.2.50.2-1.

Table 6.2.50.2-1: Properties

Name	Required	Type	Constraints	Description
scheme	yes	string	String values shall comply with IETF RFC 3986 [8]	Corresponds to the scheme component of a URI, as per IETF RFC 3986 [8]
authority	no	tosca.datatypes.nfv.UriAuthority		Corresponds to the authority component of a URI, as per IETF RFC 3986 [8] See note
path	no	string	String values shall comply with IETF RFC 3986 [8]	Corresponds to the path component of a URI, as per IETF RFC 3986 [8]
query	no	string	String values shall comply with IETF RFC 3986 [8]	Corresponds to the query component of a URI, as per IETF RFC 3986 [8]
fragment	no	string	String values shall comply with IETF RFC 3986 [8]	Corresponds to the fragment component of a URI, as per IETF RFC 3986 [8]
NOTE: If this property is not included while the URI scheme requires it, the VNFM is expected to generate it, based on knowledge of the network configuration of the external CP instance that provides the connectivity for this interface.				

### 6.2.50.3 Definition

The syntax of the UriComponents data type shall comply with the following definition:

```

tosca.datatypes.nfv.UriComponents:
  derived_from: toska.datatypes.Root
  description: information used to build a URI that complies with IETF RFC 3986 [8].
  properties:
    scheme:
      type: string # shall comply with IETF RFC 3986
      description: scheme component of a URI.
      required: true
    authority:
      type: toska.datatypes.nfv.UriAuthority
      description: Authority component of a URI
      required: false
    path:
      type: string # shall comply with IETF RFC 3986
      description: path component of a URI.
      required: false
    query:
      type: string # shall comply with IETF RFC 3986
      description: query component of a URI.
      required: false
    fragment:
      type: string # shall comply with IETF RFC 3986
      description: fragment component of a URI.
      required: false

```

### 6.2.50.4 Examples

See clause 6.10.12.

### 6.2.50.5 Additional Requirements

When this datatype is used to provide information for accessing APIs defined in ETSI GS NFV-SOL 002 [i.4], the path property may be included and the query and fragment properties shall be absent. The values of the scheme, authority and path properties form the {apiRoot} of the URI prefix.

## 6.2.51 toasca.datatypes.nfv.UriAuthority

### 6.2.51.1 Description

The UriAuthority data type corresponds to the authority component of a URI as specified in IETF RFC 3986 [8]. Table 6.2.51.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.51.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	UriAuthority
<b>Type Qualified Name</b>	toscanfv:UriAuthority
<b>Type URI</b>	tosca.datatypes.nfv.UriAuthority

### 6.2.51.2 Properties

The properties of the UriAuthority data type shall comply with the provisions set out in table 6.2.51.2-1.

**Table 6.2.51.2-1: Properties**

Name	Required	Type	Constraints	Description
user_info	no	string	String values shall comply with IETF RFC 3986 [8]	Corresponds to the user_info field of the authority component of a URI, as per IETF RFC 3986 [8].  For HTTP and HTTPS URIs, the provisions in sections 2.7.1 and 2.7.2 of IETF RFC 7230 [12] apply, respectively.
host	no	string	String values shall comply with IETF RFC 3986 [8]	Corresponds to the host field of the authority component of a URI, as per IETF RFC 3986 [8]. See note 1.
port	no	string	String values shall comply with IETF RFC 3986 [8]	Corresponds to the port field of the authority component of a URI, as per IETF RFC 3986 [8]. See note 2.
NOTE 1: If this property is not included the VNFM is expected to generate it, based on knowledge of the network configuration of the external CP instance that provides the connectivity for this interface.				
NOTE 2: If this property is not included the default port for the protocol declared by the scheme property of the parent UriComponents structure shall be used unless there are configuration mechanisms applied that are outside the scope of the present document. If no default port exists for the URI scheme, the port property shall be included unless there are configuration mechanisms applied that are outside the scope of the present document.				

### 6.2.51.3 Definition

The syntax of the UriAuthority data type shall comply with the following definition:

```
tosca.datatypes.nfv.UriAuthority:
  derived_from: toasca.datatypes.Root
  description: information that corresponds to the authority component of a URI as
  specified in IETF RFC 3986 [8]
  properties:
    user_info:
      type: string # shall comply with IETF RFC 3986
      description: user_info field of the authority component of a URI
      required: false
    host:
      type: string # shall comply with IETF RFC 3986
      description: host field of the authority component of a URI
      required: false
```

```

port:
  type: string # shall comply with IETF RFC 3986
  description: port field of the authority component of a URI
  required: false

```

#### 6.2.51.4 Examples

See clause 6.10.12.

#### 6.2.51.5 Additional Requirements

When this datatype is used to provide information for accessing APIs defined in ETSI GS NFV-SOL 002 [i.4], the host property and port properties may be included and the user\_info property shall not be included. If the host property is included and the value is a registered name, it is assumed that means are in place to resolve the host name to the correct IP address. If the host property is not included, it is assumed that the VNFM will use the IP address associated to one of the connection point instances created from the VnfExpCp and VduCp node types declared as a target of the SupportedVnfInterface policy.

**NOTE:** This means that if multiple CP instances exist that were created from a particular VnfExtCp or VduCp node template, the VNFM may use any of them to attempt accessing the interface. If no reply is received because the selected CP instance is out of service or is not reachable, the VNFM is expected to try reaching the interface through another CP instance.

### 6.2.52 tosca.datatypes.nfv.VnfProfile

#### 6.2.52.1 Description

The VnfProfile data type describes a profile for instantiating VNFs of a particular NS DF according to a specific VNFD and VNF DF as defined in ETSI GS NFV-IFA 014 [2]. Table 6.2.52.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.52.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VnfProfile
<b>Type Qualified Name</b>	toscanfv:VnfProfile
<b>Type URI</b>	tosca.datatypes.nfv.VnfProfile

#### 6.2.52.2 Properties

The properties of the VnfProfile data type shall comply with the provisions set out in table 6.2.52.2-1.

**Table 6.2.52.2-1: Properties**

Name	Required	Type	Constraints	Description
instantiation_Level	no	string		Identifier of the instantiation level of the VNF DF to be used for instantiation. If not present, the default instantiation level as declared in the VNFD shall be used.
min_number_of_instances	yes	integer	greater_or_equal : 0	Minimum number of instances of the VNF based on this VNFD that is permitted to exist for this VnfProfile.
max_number_of_instances	yes	integer	greater_or_equal : 0	Maximum number of instances of the VNF based on this VNFD that is permitted to exist for this VnfProfile.

### 6.2.52.3 Definition

The syntax of the VnfProfile data type shall comply with the following definition:

```

tosca.datatypes.nfv.VnfProfile:
  derived_from: toska.datatypes.Root
  description: describes a profile for instantiating VNFs of a particular NS DF
  according to a specific VNFD and VNF DF.
  properties:
    instantiation_level:
      type: string
      description: Identifier of the instantiation level of the VNF DF to be used for
  instantiation. If not present, the default instantiation level as declared in the VNFD
  shall be used.
      required: false
    min_number_of_instances:
      type: integer
      description: Minimum number of instances of the VNF based on this VNFD that is
  permitted to exist for this VnfProfile.
      required: true
      constraints:
        - greater_or_equal: 0
    max_number_of_instances:
      type: integer
      description: Maximum number of instances of the VNF based on this VNFD that is
  permitted to exist for this VnfProfile.
      required: true
      constraints:
        - greater_or_equal: 0

```

### 6.2.52.4 Example

None.

### 6.2.52.5 Additional Requirements

None.

## 6.2.53 toska.datatypes.nfv.ChecksumData

### 6.2.53.1 Description

The ChecksumData data type describes information about the result of performing a checksum operation over some arbitrary data. Table 6.2.53.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.2.53.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	ChecksumData
<b>Type Qualified Name</b>	toscanfv:ChecksumData
<b>Type URI</b>	tosca.datatypes.nfv.ChecksumData



### 6.2.53.2 Properties

The properties of the ChecksumData data type shall comply with the provisions set out in table 6.2.53.2-1.

**Table 6.2.53.2-1: Properties**

Name	Required	Type	Constraints	Description
algorithm	yes	string	Valid values: sha-224, sha-256, sha-384, sha-512	Describes the algorithm used to obtain the checksum value, as described in [14].
hash	yes	string		Contains the result of applying the algorithm indicated by the algorithm property to the data to which this ChecksumData refers.

### 6.2.53.3 Definition

The syntax of the ChecksumData data type shall comply with the following definition:

```

tosca.datatypes.nfv.ChecksumData:
  derived_from: toska.datatypes.Root
  description: Describes information about the result of performing a checksum
operation over some arbitrary data
  properties:
    algorithm:
      type: string
      description: Describes the algorithm used to obtain the checksum value
      required: true
      constraints:
        - valid_values: [sha-224, sha-256, sha-384, sha-512 ]
    hash:
      type: string
      description: Contains the result of applying the algorithm indicated by the
algorithm property to the data to which this ChecksumData refers
      required: true

```

### 6.2.53.4 Examples

```

<some_tosca_entity>:
  properties:
    checksum:
      algorithm: sha-256
      hash: b9c3036539fd7a5f87a1bf38eb05fdde8b556a1a7e664dbeda90ed3cd74b4f9d

```

### 6.2.53.5 Additional Requirements

None.

## 6.3 Artifact Types

### 6.3.1 toska.artifacts.nfv.SwImage

#### 6.3.1.1 Description

The SwImage artifact describes the software image which is directly loaded on the virtualisation container realizing of the VDU or is to be loaded on a virtual storage resource, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.3.1.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.3.1.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	SwImage
<b>Type Qualified Name</b>	toscanfv:SwImage
<b>Type URI</b>	tosca.artifacts.nfv.SwImage

### 6.3.1.2 Description

```
tosca.artifacts.nfv.SwImage:
  derived_from: toasca.artifacts.Deployment.Image
  description: describes the software image which is directly loaded on the
virtualisation container realizing of the VDU or is to be loaded on a virtual storage
resource
```

## 6.3.2 toasca.artifacts.Implementation.nfv.Mistral

### 6.3.2.1 Description

This artifact type represents a Mistral file that contains Mistral language [i.7] constructs that can be executed within a Mistral workbook. Support of this type is optional.

**Table 6.3.2.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	Mistral
<b>Type Qualified Name</b>	toscanfv:Mistral
<b>Type URI</b>	tosca.artifacts.Implementation.nfv.Mistral
<b>derived_from</b>	tosca.artifacts.Implementation

### 6.3.2.2 Definition

The syntax of the Mistral artifact type shall comply with the following definition:

```
tosca.artifacts.Implementation.nfv.Mistral:
  derived_from: toasca.artifacts.Implementation
  description: artifacts for Mistral workflows
  mime_type: application/x-yaml
  file_ext: [ yaml ]
```

## 6.4 Capability Types

### 6.4.1 toasca.capabilities.nfv.VirtualBindable

#### 6.4.1.1 Description

The VirtualBindable capability indicates that the node that includes it can be pointed by a toasca.relationships.nfv.VirtualBindsTo relationship type which is used to model the VduHasCpd association illustrated in ETSI GS NFV-IFA 011 [1]. Table 6.4.1.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.4.1.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VirtualBindable
<b>Type Qualified Name</b>	toscanfv:VirtualBindable
<b>Type URI</b>	tosca.capabilities.nfv.VirtualBindable

### 6.4.1.2 Properties

None.

### 6.4.1.3 Definition

The syntax of the VirtualBindable capability type shall comply with the following definition:

```
tosca.capabilities.nfv.VirtualBindable:
  derived_from: toska.capabilities.Node
  description: Indicates that the node that includes it can be pointed by a
tosca.relationships.nfv.VirtualBindsTo relationship type which is used to model the
VduHasCpd association
```

## 6.4.2 toska.capabilities.nfv.VirtualLinkable

### 6.4.2.1 Description

A node type that includes the VirtualLinkable capability indicates that it can be pointed by toska.relationships.nfv.VirtualLinksTo relationship type, which is used to model the VduCpdReferencesVirtualLinkDesc and the VnfExtCpdMapsToVirtualLinkDesc associations illustrated in ETSI GS NFV-IFA 011 [1]. Table 6.4.2.1-1 specifies the declared names for this capability type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.4.2.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VirtualLinkable
<b>Type Qualified Name</b>	toscanfv:VirtualLinkable
<b>Type URI</b>	tosca.capabilities.nfv.VirtualLinkable

### 6.4.2.2 Properties

None.

### 6.4.2.3 Definition

The syntax of the VirtualLinkable capability type shall comply with the following definition:

```
tosca.capabilities.nfv.VirtualLinkable:
  derived_from: toska.capabilities.Node
  description: A node type that includes the VirtualLinkable capability indicates that
it can be pointed by toska.relationships.nfv.VirtualLinksTo relationship type
```

## 6.4.3 toska.capabilities.nfv.VirtualCompute

### 6.4.3.1 Description

The VirtualCompute capability type describes the capabilities related to virtual compute resources, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.4.3.1-1 specifies the declared names for this capability type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.4.3.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VirtualCompute
<b>Type Qualified Name</b>	toscanfv:VirtualCompute
<b>Type URI</b>	tosca.capabilities.nfv.VirtualCompute

### 6.4.3.2 Properties

The properties of the VirtualCompute capability type shall comply with the provisions set out in table 6.4.3.2-1.

**Table 6.4.3.2-1: Properties**

Name	Required	Type	Constraints	Description
logical_node	no	map of tosca.datatypes.nfv.LogicalNodeData		The Logical Node requirements.
request_additional_capabilities	no	map of tosca.datatypes.nfv.RequestedAdditionalCapability		Describes additional capability for a particular VDU.
compute_requirements	no	map of string		Describes compute requirements.
virtual_memory	yes	tosca.datatypes.nfv.VirtualMemory		Describes virtual memory of the virtualised compute.
virtual_cpu	yes	tosca.datatypes.nfv.VirtualCpu		Describes virtual CPU(s) of the virtualised compute.
virtual_local_storage	no	list of tosca.datatypes.nfv.VirtualBlockStorageData		A list of virtual system disks created and destroyed as part of the VM lifecycle.

### 6.4.3.3 Definition

The syntax of the VirtualCompute capability type shall comply with the following definition:

```
tosca.capabilities.nfv.VirtualCompute:
  derived_from: toasca.capabilities.Node
  description: Describes the capabilities related to virtual compute resources
  properties:
    logical_node:
      type: map
      description: Describes the Logical Node requirements
      required: false
      entry_schema:
        type: toasca.datatypes.nfv.LogicalNodeData
    requested_additional_capabilities:
      type: map
      description: Describes additional capability for a particular VDU
      required: false
      entry_schema:
        type: toasca.datatypes.nfv.RequestedAdditionalCapability
    compute_requirements:
      type: map
      required: false
      entry_schema:
        type: string
    virtual_memory:
```

```

type: toska.datatypes.nfv.VirtualMemory
description: Describes virtual memory of the virtualized compute
required: true
virtual_cpu:
type: toska.datatypes.nfv.VirtualCpu
description: Describes virtual CPU(s) of the virtualized compute
required: true
virtual_local_storage:
type: list
description: A list of virtual system disks created and destroyed as part of the
VM lifecycle
required: false
entry_schema:
type: toska.datatypes.nfv.VirtualBlockStorageData
description: virtual system disk definition

```

## 6.4.4 toska.capabilities.nfv.VirtualStorage

### 6.4.4.1 Description

The VirtualStorage capability indicates that the node that includes it can be pointed by a toska.relationships.nfv.AttachesTo relationship type which is used to model the VduHasVirtualStorageDesc association illustrated in ETSI GS NFV-IFA 011 [1]. Table 6.4.4.1-1 specifies the declared names for this capability type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.4.4.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VirtualStorage
<b>Type Qualified Name</b>	toscanfv:VirtualStorage
<b>Type URI</b>	tosca.capabilities.nfv.VirtualStorage

### 6.4.4.2 Definition

The syntax of the VirtualStorage capability type shall comply with the following definition:

```

tosca.capabilities.nfv.VirtualStorage:
derived_from: toska.capabilities.Root
description: Describes the attachment capabilities related to Vdu.Storage

```

## 6.5 Requirements Types

None.

## 6.6 Relationship Types

### 6.6.1 toska.relationships.nfv.VirtualBindsTo

#### 6.6.1.1 Description

This relationship type represents an association between Vdu.Compute and VduCp node types. Table 6.6.1.1-1 specifies the declared names for this relationship type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.6.1.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VirtualBindsTo
<b>Type Qualified Name</b>	toscanfv:VirtualBindsTo
<b>Type URI</b>	tosca.relationships.nfv.VirtualBindsTo

### 6.6.1.2 Properties

None.

### 6.6.1.3 Definition

The syntax of the VirtualBindsTo relationship type shall comply with the following definition:

```
tosca.relationships.nfv.VirtualBindsTo:
  derived_from: toasca.relationships.DependsOn
  description: Represents an association relationship between Vdu.Compute and VduCp
node types
  valid_target_types: [ toasca.capabilities.nfv.VirtualBindable ]
```

## 6.6.2 toasca.relationships.nfv.VirtualLinksTo

### 6.6.2.1 Description

This relationship type represents an association between the VduCp and VnfVirtualLink node types. Table 6.6.2.1-1 specifies the declared names for this relationship type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.6.2.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VirtualLinksTo
<b>Type Qualified Name</b>	toscanfv:VirtualLinksTo
<b>Type URI</b>	tosca.relationship.nfv.VirtualLinksTo

### 6.6.2.2 Properties

None.

### 6.6.2.3 Definition

The syntax of the VnfVirtualLinksTo relationship type shall comply with the following definition:

```
tosca.relationships.nfv.VirtualLinksTo:
  derived_from: toasca.relationships.DependsOn
  description: Represents an association relationship between the VduCp and
VnfVirtualLink node types
  valid_target_types: [ toasca.capabilities.nfv.VirtualLinkable ]
```

## 6.6.3 toasca.relationships.nfv.AttachesTo

### 6.6.3.1 Description

This relationship type represents an association between the Vdu.Compute and one of the following node types: Vdu.VirtualBlockStorage, Vdu.VirtualObjectStorage or Vdu.VirtualFileStorage. Table 6.6.3.1-1 specifies the declared names for this relationship type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.6.3.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	AttachesTo
<b>Type Qualified Name</b>	toscanfv:AttachesTo
<b>Type URI</b>	tosca.relationships.nfv.AttachesTo

### 6.6.3.2 Properties

None.

### 6.6.3.3 Definition

The syntax of the AttachesTo relationship type shall comply with the following definition:

```
tosca.relationships.nfv.AttachesTo:
  derived_from: toasca.relationships.Root
  description: Represents an association relationship between the Vdu.Compute and one
of the node types, Vdu.VirtualBlockStorage, Vdu.VirtualObjectStorage or
Vdu.VirtualFileStorage
  valid_target_types: [ toasca.capabilities.nfv.VirtualStorage ]
```

## 6.7 Interface Types

### 6.7.1 toasca.interfaces.nfv.Vnflcm

#### 6.7.1.1 Description

The toasca.interfaces.nfv.Vnflcm contains a set of TOSCA operations corresponding to the following VNF LCM operations defined in ETSI GS NFV-IFA 007 [i.1]:

- Instantiate VNF
- Terminate VNF
- Modify VNF information
- Change VNF Flavour
- Change External VNF Connectivity
- Operate VNF
- Heal VNF
- Scale VNF
- Scale VNF To Level

In addition, the VNF shall also support TOSCA operations corresponding to preamble and postamble to the execution of the base operation. The name of these operations is constructed according to the following pattern:

- <base\_operation\_name>\_start for a preamble
- <base\_operation\_name>\_end for a postamble

The designations ("\_start", "\_end") in the name of TOSCA operations are postfixes so that related operations are adjacent in an alphabetical listing.

Table 6.7.1.1-1 specifies the declared names for this interface type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

Table 6.7.1.1-1: Type name, shorthand, and URI

<b>Shorthand Name</b>	Vnflcm
<b>Type Qualified Name</b>	toscanfv:Vnflcm
<b>Type URI</b>	tosca.interfaces.nfv.Vnflcm

### 6.7.1.2 Definition

The syntax of the Vnflcm interface type shall comply with the following definition:

```

tosca.interfaces.nfv.Vnflcm:
  derived_from: toasca.interfaces.Root
  description: This interface encompasses a set of TOSCA operations corresponding to
the VNF LCM operations defined in ETSI GS NFV-IFA 007 as well as to preamble and
postamble procedures to the execution of the VNF LCM operations.
  instantiate:
    description: Invoked upon receipt of an Instantiate VNF request
    # inputs:
    # additional_parameters:
    # type: toasca.datatypes.nfv.VnfOperationAdditionalParameters
    # required: false
    # derived types are expected to introduce additional_parameters with its
    # type derived from toasca.datatypes.nfv.VnfOperationAdditionalParameters
  instantiate_start:
    description: Invoked before instantiate
  instantiate_end:
    description: Invoked after instantiate
  terminate:
    description: Invoked upon receipt Terminate VNF request
    # inputs:
    # additional_parameters:
    # type: toasca.datatypes.nfv.VnfOperationAdditionalParameters
    # required: false
    # derived types are expected to introduce additional_parameters with its
    # type derived from toasca.datatypes.nfv.VnfOperationAdditionalParameters
  terminate_start:
    description: Invoked before terminate
  terminate_end:
    description: Invoked after terminate
  modify_information:
    description: Invoked upon receipt of a Modify VNF Information request
  modify_information_start:
    description: Invoked before modify_information
  modify_information_end:
    description: Invoked after modify_information
  change_flavour:
    description: Invoked upon receipt of a Change VNF Flavour request
    # inputs:
    # additional_parameters:
    # type: toasca.datatypes.nfv.VnfOperationAdditionalParameters
    # required: false
    # derived types are expected to introduce additional_parameters with its
    # type derived from toasca.datatypes.nfv.VnfOperationAdditionalParameters
  change_flavour_start:
    description: Invoked before change_flavour
  change_flavour_end:
    description: Invoked after change_flavour
  change_external_connectivity:
    description: Invoked upon receipt of a Change External VNF Connectivity request
    # inputs:

```



```
# additional_parameters:
# type: toasca.datatypes.nfv.VnfOperationAdditionalParameters
# required: false
# derived types are expected to introduce additional_parameters with its
# type derived from toasca.datatypes.nfv.VnfOperationAdditionalParameters
change_external_connectivity_start:
description: Invoked before change_external_connectivity
change_external_connectivity_end:
description: Invoked after change_external_connectivity
operate:
description: Invoked upon receipt of an Operate VNF request
# inputs:
# additional_parameters:
# type: toasca.datatypes.nfv.VnfOperationAdditionalParameters
# required: false
# derived types are expected to introduce additional_parameters with its
# type derived from toasca.datatypes.nfv.VnfOperationAdditionalParameters
operate_start:
description: Invoked before operate
operate_end:
description: Invoked after operate
heal:
description: Invoked upon receipt of a Heal VNF request
# inputs:
# additional_parameters:
# type: toasca.datatypes.nfv.VnfOperationAdditionalParameters
# required: false
# derived types are expected to introduce additional_parameters with its
# type derived from toasca.datatypes.nfv.VnfOperationAdditionalParameters
heal_start:
description: Invoked before heal
heal_end:
description: Invoked after heal
scale:
description: Invoked upon receipt of a Scale VNF request
# inputs:
# additional_parameters:
# type: toasca.datatypes.nfv.VnfOperationAdditionalParameters
# required: false
# derived types are expected to introduce additional_parameters with its
# type derived from toasca.datatypes.nfv.VnfOperationAdditionalParameters
scale_start:
description: Invoked before scale
scale_end:
description: Invoked after scale
scale_to_level:
description: Invoked upon receipt of a Scale VNF to Level request
# inputs:
# additional_parameters:
# type: toasca.datatypes.nfv.VnfOperationAdditionalParameters
# required: false
# derived types are expected to introduce additional_parameters with its
# type derived from toasca.datatypes.nfv.VnfOperationAdditionalParameters
scale_to_level_start:
description: Invoked before scale_to_level
scale_to_level_end:
description: Invoked after scale_to_level
```

### 6.7.1.3 Additional Requirements

All VNF supported LCM operations shall be listed in the service template, except "instantiate" and "terminate" that may be omitted, as specified in ETSI GS NFV-IFA 011 [1] for the supportedOperation attribute of a deployment flavour.

The implementation and inputs keynames specified in TOSCA-Simple-Profile-YAML-v1.2 [3] for an operation definition may be included for each operation listed in the Vnflcm interface definition.

If a TOSCA operation representing a VNF LCM operation is listed in the service template without an associated implementation, then it means that:

- the VNF LCM operation is supported (i.e. this is the manifestation of the supportedOperation attribute as per ETSI GS NFV-IFA 011 [1]); and
- the processing logic associated with the LCM operation is the default implementation provided by the VNFM.

If an implementation is associated to a TOSCA operation that represents a preamble or a postamble to a VNF LCM operation, the implementation logic is executed before or after the execution of the VNF LCM operation implementation, respectively.

The VNFM shall make available all parameters from the message invoking the VNF LCM operation as inputs to the corresponding TOSCA interface operations.

In the operation definitions on the Vnflcm interface, the additional\_parameters (VNF-specific extension of the `tosca.datatypes.nfv.VnfOperationAdditionalParameters`) of the inputs section describes the name and type of the additional parameters (additionalParams) that can be submitted in the VNF LCM operation request. See an example in clause 6.2.43 (`tosca.datatypes.nfv.VnfOperationAdditionalParameters`).

The inputs keyname can also be used to specify additional input parameters for executing the TOSCA operation, beyond those received in the VNF LCM operation request. To distinguish them from the latter ones, such input parameters shall not be named "additional\_parameters".

The implementation of the operation corresponding to preamble and postamble TOSCA operations (`instantiate_start`, `instantiate_end`, `scale_start`, `scale_end`, etc.), if present, shall be invoked with the same parameters as the corresponding base operations ones (`instantiate`, `scale`, etc.). The inputs of the operations corresponding to the postamble and preamble operations shall not be defined in the VNFD.

### 6.7.1.4 Support of LCM scripts

In ETSI GS NFV-IFA 011 [1], the definition of the "LifeCycleManagementScript" information element of the VNFD associates scripts with events, where an event can be an external or an internal stimulus. These events are mapped to TOSCA operations of the VNF node type in the following way:

- external stimuli are mapped to TOSCA operations corresponding to the VNF LCM operations defined in ETSI GS NFV-IFA 007 [i.1];
- internal stimuli are mapped to preamble and postamble of these TOSCA operations;
- events that cannot be mapped to these TOSCA operations (`lcmTransitionEvent` as described in ETSI GS NFV-IFA 011 [1], clause 7.1.13) can be mapped to further TOSCA operations by extending the TOSCA interface.

The LCM scripts can be regarded as artifacts that provide a VNF-specific implementation of the TOSCA operation corresponding to the stimulus. The script input parameters shall be provided to the script according to the declaration in the inputs field of the operation definition. The artifact type definition shall enable identifying the DSL used by the script. The artifact type definition for Python is provided in section 5.4.4.1 of TOSCA-Simple-Profile-YAML-v1.2 [3]. The artifact definition for Mistral is provided in clause 6.3.2 of the present document.

**NOTE:** As all input parameters needed for operations corresponding to external and internal stimuli are defined in the "input parameters of the external stimuli operations", the VNF Designer is expected to make the list of parameters as complete as needed to handle not only the external stimuli but also the internal stimuli.

### 6.7.1.5 Examples

The following example template fragments illustrate the concepts.

```
tosca_definitions_version: tosca_simple_yaml_1_2

imports:
- ..

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    ..

topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineDB.1_0.1_0
    ..

node_templates:
  SunshineDB:
    type: MyCompany.SunshineDB.1_0.1_0
    ..
  interfaces:
    Vnflcm:
      instantiate:
        ..
```

In the above example, as there is no implementation and inputs specified to the operations, the built-in implementation of the operation is invoked when the Instantiate VNF request is received on the LCM interface of the Or-Vnfm reference point. The received parameters (flavourId, instantiationLevelId, etc.) are passed to the built-in implementation (as flavour\_id, instantiation\_level\_id).

```
tosca_definitions_version: tosca_simple_yaml_1_2

imports:
- ..

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    ..

topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineDB.1_0.1_0
    ..

node_templates:
  SunshineDB:
    type: MyCompany.SunshineDB.1_0.1_0
    ..
  interfaces:
    Vnflcm:
      instantiate:
        implementation: instantiate-script
        ..
```

In the above example, the `instantiate-script` is invoked when the `Instantiate VNF` request is received, passing the received parameters to it similarly to the previous example. This example does not imply a one-to-one mapping between operations and script names.

```
tosca_definitions_version: tosca_simple_yaml_1_2

imports:
- ..

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    ..

topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineDB.1_0.1_0
    ..

node_templates:
  SunshineDB:
    type: MyCompany.SunshineDB.1_0.1_0
    ..
  interfaces:
    Vnflcm:
      ..
      scale_start: pre-scale-script
      scale_end: post-scale-script
      ..
```

In the above example, LCM scripts are associated with the "scale start" and "scale end" internal stimuli. As no script is associated to the `scale` operation, its default implementation runs (after running the `pre-scale-script`, and before running the `post-scale-script`).

```
tosca_definitions_version: tosca_simple_yaml_1_2

imports:
- ..

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    ..

topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineDB.1_0.1_0
    ..

node_templates:
  SunshineDB:
    type: MyCompany.SunshineDB.1_0.1_0
    ..
  interfaces:
    Vnflcm:
      instantiate:
        implementation: instantiate-script
        inputs:
          script_input_1: value_1
          script_input_2: value_2
```

```
artifacts:
  instantiate-script:
    description: Instantiate workflow script
    type: toska.artifacts.Implementation.Python
    file: instantiate.py
    #repository: ..
    #deploy_path: ..
    ..
```

In the above example:

- The inputs section provides additional input values to the instantiate-script (i.e. the manifestation of the scriptInput attribute of LifeCycleManagementScript as defined in ETSI GS NFV-IFA 011 [1]).

NOTE: There is another kind of input called `additional_parameters` dedicated to the additional parameters (additionalParams) received in the message invoking the VNF LCM operation; this input is not illustrated by the above examples; see clause 6.2.43.4 on how to declare `additional_parameters` in the derived VNF node type.

- TOSCA artifacts definition is used to convey the type of DSL used as a scripting language that is associated with an operation (i.e. the manifestation of the scriptDsl attribute of LifeCycleManagementScript as per ETSI GS NFV-IFA 011 [1]).

## 6.8 Node Types

### 6.8.1 toska.nodes.nfv.VNF

#### 6.8.1.1 Description

The VNF node type is the generic abstract type from which all VNF specific abstract node types shall be derived to form, together with other node types, the TOSCA service template(s) representing the VNFD information element as defined in ETSI GS NFV-IFA 011 [1]. Table 6.8.1.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.8.1.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VNF
<b>Type Qualified Name</b>	toscanfv:VNF
<b>Type URI</b>	tosca.nodes.nfv.VNF

#### 6.8.1.2 Properties

The properties of the VNF node type shall comply with the provisions set out in table 6.8.1.2-1.

**Table 6.8.1.2-1: Properties**

Name	Required	Type	Constraints	Description
descriptor_id	yes	string		Identifier of this VNFD information element. This attribute shall be globally unique.  The VNFD Identifier shall be used as the unique identifier of the VNF Package that contains this VNFD. Any modification of the content of the VNFD or the VNF Package shall result in a new VNFD Identifier.
descriptor_version	yes	string		Identifies the version of the VNFD.

Name	Required	Type	Constraints	Description
provider	yes	string		Provider of the VNF and of the VNFD.
product_name	yes	string		Name to identify the VNF Product. Invariant for the VNF Product lifetime.
software_version	yes	string		Software version of the VNF. This is changed when there is any change to the software that is included in the VNF Package.
product_info_name	no	string		Human readable name for the VNF Product. Can change during the VNF Product lifetime.
product_info_description	no	string		Human readable description of the VNF Product. Can change during the VNF Product lifetime.
vnfm_info	yes	list of string		Identifies VNFM(s) compatible with the VNF described in this version of the VNFD. To indicate that a VNF can be managed by any ETSI NFV compliant VNFM, the string value shall be the concatenation of the string "etsivnfm" and the minimum version of ETSI GS NFV-SOL 002 [i.4] to be supported by this VNFM (e.g. etsivnfm:v2.3.1). If the VNF is compatible with multiple versions, multiple values may be included. See note 1.  To indicate a specific VNFM product, the string value shall be the concatenation of the IANA enterprise number of the VNFM provider [5], followed by a product-specific string.
localization_languages	no	list of string	Valid values: string values that comply with IETF RFC 5646 [13]	Information about localization languages of the VNF (includes e.g. strings in the VNFD).  This allows to provide one or more localization languages to support selecting a specific localization language at VNF instantiation time.
default_localization_language	no	string	Valid values: string values that comply with IETF RFC 5646 [13]	Default localization language that is instantiated if no information about selected localization language is available. Shall be present if "localizationLanguage" is present and shall be absent otherwise.
configurable_properties	no	tosca.datatypes.nfv.VnfConfigurableProperties		Describes the configurable properties of the VNF (e.g. related to auto scaling and auto healing).
modifiable_attributes	no	tosca.datatypes.nfv.VnfInfoModifiableAttributes		Describes the modifiable attributes of the VNF.
lcm_operations_configuration	no	tosca.datatypes.nfv.VnfLcmOperationsConfiguration		Describes the configuration parameters for the VNF LCM operations.

Name	Required	Type	Constraints	Description
monitoring_parameters	no	list of toscadataypes.nfv.VnfMonitoringParameter		Describes monitoring parameters applicable to the VNF.
flavour_id	yes	string		Identifier of this DF within the VNFD.
flavour_description	yes	string		Human readable description of the DF.
vnf_profile	no	tosca.dataypes.nfv.VnfProfile		Describes a profile for instantiating VNFs of a particular NS DF according to a specific VNFD and VNF DF. See note 2.
NOTE 1: When LCM scripts are used, the support of this minimum version might not be sufficient to ensure that the VNF can be managed by a VNFM. The support of the domain specific language(s) used by these LCM scripts is another criterion for determining the compatibility of the VNF with a VNFM.				
NOTE 2: This property is only used in a NSD service template when describing a VNF node template with the corresponding VnfProfile information.				

The value of the descriptor\_id string shall comply with an UUID format as specified in section 3 of [9].

The syntax of the vnfm\_info string values shall comply with the following ABNF [6] snippet:

```
value = any_etsi_nfv_compliant_product | product_specific
any_etsi_nfv_compliant_product = "etsivnfm" SEP "version"
version = "v" version_identifier
version_identifier = 1*2DIGIT DOT 1*2DIGIT DOT 1*2DIGIT
; the version identifier is encoded as a sequence of items of 1 or 2 digits separated by dots
representing the 3 fields (major, technical and editorial) of the version of an ETSI deliverable.
product_specific = enterprise_number SEP product_specific_string
enterprise_number = 1*DIGIT
product_specific_string = *(ALPHA / DIGIT / "-" / ".")
SEP = ":"
DOT = "."
```

This implies that vnfm\_info string values shall also comply with the pattern defined by the following regular expression [15]:  $(^etsivnfm:v[0-9]?[0-9]\.[0-9]?[0-9]\.[0-9]?[0-9]$(^([0-9]+:[a-zA-Z0-9.-]+)$)$

### 6.8.1.3 Attributes

None.

### 6.8.1.4 Requirements

The requirements of the VNF node type shall comply with the provisions set out in table 6.8.1.4-1.

**Table 6.8.1.4-1: Requirements**

Name	Required	Type	Constraints	Description
virtual_link	yes	tosca.capabilities.nfv.VirtualLinkable		Describes the requirements for linking to virtual link

### 6.8.1.5 Capabilities

None.

### 6.8.1.6 Definition

The syntax of the VNF node type shall comply with the following definition:

```

tosca.nodes.nfv.VNF:
  derived_from: tosca.nodes.Root
  description: The generic abstract type from which all VNF specific abstract node
types shall be derived to form, together with other node types, the TOSCA service
template(s) representing the VNFD
  properties:
    descriptor_id: # instead of vnfd_id
      type: string # GUID
      description: Globally unique identifier of the VNFD
      required: true
    descriptor_version: # instead of vnfd_version
      type: string
      description: Identifies the version of the VNFD
      required: true
    provider: # instead of vnf_provider
      type: string
      description: Provider of the VNF and of the VNFD
      required: true
    product_name: # instead of vnf_product_name
      type: string
      description: Human readable name for the VNF Product
      required: true
    software_version: # instead of vnf_software_version
      type: string
      description: Software version of the VNF
      required: true
    product_info_name: # instead of vnf_product_info_name
      type: string
      description: Human readable name for the VNF Product
      required: false
    product_info_description: # instead of vnf_product_info_description
      type: string
      description: Human readable description of the VNF Product
      required: false
    vnfm_info:
      type: list
      required: true
      description: Identifies VNFM(s) compatible with the VNF
      entry_schema:
        type: string
        constraints:
          - pattern: (^etsivnfm:v[0-9]?[0-9]\.[0-9]?[0-9]\.[0-9]?[0-9]$)|(^[0-9]+:[a-zA-
Z0-9.-]+)$)
    localization_languages:
      type: list
      description: Information about localization languages of the VNF
      required: false
      entry_schema:
        type: string #IETF RFC 5646 string
    default_localization_language:
      type: string #IETF RFC 5646 string
      description: Default localization language that is instantiated if no information
about selected localization language is available
      required: false
    #configurable_properties:
      #type: tosca.datatypes.nfv.VnfConfigurableProperties
      #description: Describes the configurable properties of the VNF

```



```

#required: false
# derived types are expected to introduce configurable_properties
# with its type derived from toska.datatypes.nfv.VnfConfigurableProperties
#modifiable_attributes:
#type: toska.datatypes.nfv.VnfInfoModifiableAttributes
#description: Describes the modifiable attributes of the VNF
#required: false
# derived types are expected to introduce modifiable_attributes
# with its type derived from
# toska.datatypes.nfv.VnfInfoModifiableAttributes
lcm_operations_configuration:
type: toska.datatypes.nfv.VnfLcmOperationsConfiguration
description: Describes the configuration parameters for the VNF LCM operations
required: false
monitoring_parameters:
type: list
entry_schema:
type: toska.datatypes.nfv.VnfMonitoringParameter
description: Describes monitoring parameters applicable to the VNF.
required: false
flavour_id:
type: string
description: Identifier of the Deployment Flavour within the VNFD
required: true
flavour_description:
type: string
description: Human readable description of the DF
required: true
vnf_profile:
type: toska.datatypes.nfv.VnfProfile
description: Describes a profile for instantiating VNFs of a particular NS DF
according to a specific VNFD and VNF DF
required: false
requirements:
- virtual_link:
capability: toska.capabilities.nfv.VirtualLinkable
relationship: toska.relationships.nfv.VirtualLinksTo
occurrences: [ 0, 1 ]
# Additional requirements shall be defined in the VNF specific node type (deriving
from toska.nodes.nfv.VNF) corresponding to NS virtual links that need to connect to
VnfExtCps
interfaces:
Vnflcm:
type: toska.interfaces.nfv.Vnflcm

```

### 6.8.1.7 Artifact

None.

### 6.8.1.8 Additional Requirements

For a given VNFD, a new VNF node type shall be defined following the below requirements:

- a) The node type shall be derived from: toska.nodes.nfv.VNF.
- b) The following properties listed in toska.nodes.nfv.VNF where the "required:" field is set to "true" shall be included with their values indicated as constraints and as default values:
  - a. descriptor\_id

- b. descriptor\_version
- c. provider
- d. product\_name
- e. software\_version
- f. vnfm\_info
- g. flavour\_id
- h. flavour\_description

NOTE 1: Indicating their values as default allows not to include them in property assignments in node templates, e.g. in the NSD.

- c) An empty string shall be indicated as the default value of the flavour\_description property.
- d) The capabilities, requirements, interfaces of toasca.nodes.nfv.VNF shall be preserved.
- e) Depending on the number of external connection points of the VNF that need to connect to NS virtual links, additional requirements for VirtualLinkable capability shall be defined. In this case, it is the VNFD author's choice to use the requirement for VirtualLinkable capability defined in the toasca.nodes.nfv.VNF node type or use only the additional requirements defined in the derived VNF specific node type. In the latter case, the virtual\_link requirement should be included in the node type definition with occurrences [ 0, 0 ].
- f) The rule for naming this node type in the service template should be:
  - provider.product\_name.software\_version.descriptor\_version, by concatenating the values of the corresponding properties of the created VNF node type.

NOTE 2: If the software\_version value or descriptor\_version value contains a dot (i.e. "."), this character should be replaced with an underscore (i.e. "\_").

VNF Providers shall use the following types to derive the VNF specific modifiable attributes and additional configurable properties:

- toasca.datatypes.nfv.VnfInfoModifiableAttributesExtensions.
- toasca.datatypes.nfv.VnfInfoModifiableAttributesMetadata.
- toasca.datatypes.nfv.VnfAdditionalConfigurableProperties.
- toasca.datatypes.nfv.VnfInfoModifiableAttributes.
- toasca.datatypes.nfv.VnfConfigurableProperties.

See illustrative examples in clauses 6.8.1.9, 6.2.35.4 and 6.2.31.4.

Node templates of the VNF specific node type shall not include the vnf\_profile property when they are part of a VNFD service template.

For a given NSD, when describing a referenced VNFD as a node templates, the vnf\_profile property shall be included with a valid value.

### 6.8.1.9 Example

Example usage of modifiable\_attributes:

```
tosca_definitions_version: toasca_simple_yaml_1_2

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: toasca.nodes.nfv.VNF
```

```

properties:
  flavour_id:
    constraints:
      - valid_values: [ simple, complex ]
    modifiable_attributes:
      type: mycompany.datatypes.nfv.VnfInfoModifiableAttributes

data_types:
  mycompany.datatypes.nfv.VnfInfoModifiableAttributes:
    derived_from: tosca.datatypes.nfv.VnfInfoModifiableAttributes
    properties:
      extensions:
        type: mycompany.datatypes.nfv.VnfInfoModifiableAttributesExtensions

  mycompany.datatypes.nfv.VnfInfoModifiableAttributesExtensions:
    derived_from: tosca.datatypes.nfv.VnfInfoModifiableAttributesExtensions
    properties:
      http_proxy:
        type: string
        required: true
      https_proxy:
        type: string
        required: false

```

Example usage of lcm\_operations\_configuration:

Top level service template:

```

tosca_definitions_version: tosca_simple_yaml_1_2

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      flavour_id:
        type: string
        constraints:
          - valid_values: [ simple, complex ]
      lcm_operations_configuration:
        scale:
          scaling_by_more_than_one_step_supported: true
        scale_to_level:
          arbitrary_target_levels_supported: true
      heal:
        causes:
          - service_unavailable
          - performance_degraded
      terminate:
        min_graceful_termination_timeout: 60 s
        max_recommended_graceful_termination_timeout: 600 s
      operate:
        min_graceful_stop_timeout: 60 s
        max_recommended_graceful_stop_timeout: 600 s

```

Example usage of describing a VNF node template with vnf\_profile in a NSD TOSCA service template:

```
tosca_definitions_version: toska_simple_yaml_1_2

description: an example of NSD TOSCA service template

topology template:
...

node_templates:
  VNF_1:
    type: toska.nodes.nfv.exampleVNF
    properties:
      flavour_id: small
      descriptor_id: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      provider: MyCompany
      product_name: SunshineDB
      software_version: 1.0
      descriptor_version: 1.0
      vnf_profile:
        instantiation_level: level_1
        min_number_of_instances: 2
        max_number_of_instances: 6
      # other properties omitted for brevity
    requirements:
      - virtual_link: NsVirtualLink
```

## 6.8.2 toska.nodes.nfv.VnfExtCp

### 6.8.2.1 Description

The VnfExtCp node type represents the VnfExtCpd information element as defined in ETSI GS NFV-IFA 011 [1], which describes a logical external connection point, exposed by this VNF enabling connecting with an external Virtual Link. Table 6.8.2.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.8.2.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VnfExtCp
<b>Type Qualified Name</b>	toscanfv:VnfExtCp
<b>Type URI</b>	tosca.nodes.nfv.VnfExtCp

### 6.8.2.2 Properties

The properties of the VnfExtCp node type shall comply with the provisions set out in table 6.8.2.2-1.

**Table 6.8.2.2-1: Properties**

Name	Required	Type	Constraints	Description
virtual_network_interface_requirements	no	list of toska.datatypes.nfv.VirtualNetworkInterfaceRequirements		The actual virtual NIC requirements that is been assigned when instantiating the connection point.

### 6.8.2.3 Attributes

None.

### 6.8.2.4 Requirements

The requirements of the VnfExtCp node type shall comply with the provisions set out in table 6.8.2.4-1.

**Table 6.8.2.4-1: Requirements**

Name	Required	Type	Constraints	Description
external_virtual_link	no	tosca.capabilities.nfv.VirtualLinkable		Specifies that CP instances require to be connected to a node that has a VirtualLinkable capability
internal_virtual_link	yes	tosca.capabilities.nfv.VirtualLinkable		Specifies that CP instances require to be connected to a node that has a VirtualLinkable capability

### 6.8.2.5 Definition

The syntax of the VnfExtCp node type shall comply with the following definition:

```
tosca.nodes.nfv.VnfExtCp:
  derived_from: tosca.nodes.nfv.Cp
  description: Describes a logical external connection point, exposed by the VNF
  enabling connection with an external Virtual Link
  properties:
    virtual_network_interface_requirements:
      type: list
      description: The actual virtual NIC requirements that is been assigned when
  instantiating the connection point
      required: false
      entry_schema:
        type: tosca.datatypes.nfv.VirtualNetworkInterfaceRequirements
  requirements:
    - external_virtual_link:
      capability: tosca.capabilities.nfv.VirtualLinkable
      relationship: tosca.relationships.nfv.VirtualLinksTo
    - internal_virtual_link: #name in ETSI NFV IFA 011 v0.7.3: intVirtualLinkDesc
      capability: tosca.capabilities.nfv.VirtualLinkable
      relationship: tosca.relationships.nfv.VirtualLinksTo
```

### 6.8.2.6 Additional Requirements

A node template of this type is used to represent a VNF external connection point only in the case the VnfExtCp is connected to an internal virtual link. The node template has the following requirements:

- internal\_virtual\_link requirement to allow to connect it to an internal virtual link;
- external\_virtual\_link requirement to allow to connect it to an external virtual link.

In the case where a VNF external connection point is re-exposing a VduCp (internal connection point), the VduCp node type shall be used in the service template, instead of the VnfExtCp node type.

### 6.8.2.7 Example

In a typical scenario, the VnfExtCp node template will be part of a service template representing a certain VNF deployment flavour. The service template substitutes for an abstract VNF node type. In this substitution, the virtual\_link requirement of the abstract VNF node type is mapped to the external\_virtual\_link requirement of the VnfExtCp node. This example is illustrated in clause A.3.2.

When a VNF external connection point re-exposes a Vdu connection point, the service template does not require an explicit node template of type VnfExtCp in a typical scenario where an abstract VNF node type is substituted by a service template representing a certain VNF deployment flavour. In this substitution, the virtual\_link requirement of the abstract VNF node type is mapped to the virtual\_link requirement of the VduCp node. This example is illustrated in clause A.3.1.

## 6.8.3 toska.nodes.nfv.Vdu.Compute

### 6.8.3.1 Description

The Vdu.Compute node type describes the virtual compute part of a VDU which is a construct supporting the description of the deployment and operational behaviour of a VNFC, as defined in ETSI GS NFV-IFA 011 [1].

Table 6.8.3.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.8.3.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	Vdu.Compute
<b>Type Qualified Name</b>	toscanfv:Vdu.Compute
<b>Type URI</b>	tosca.nodes.nfv.Vdu.Compute

### 6.8.3.2 Properties

The properties of the Vdu.Compute node type shall comply with the provisions set out in table 6.8.3.2-1.

**Table 6.8.3.2-1: Properties**

Name	Required	Type	Constraints	Description
name	yes	string		Human readable name of the Vdu.
description	yes	string		Human readable description of the Vdu.
boot_order	no	list of string		The key indicates the boot index (lowest index defines highest boot priority). The Value references a node template name from which a valid boot device is created e.g. VirtualBlockStorage node template from which a VirtualBlockStorage instance is created.  If no boot order is defined the default boot order defined in the VIM or NFVI shall be used.
nfvi_constraints	no	list of string		Describes constraints on the NFVI for the VNFC instance(s) created from this Vdu. For example, aspects of a secure hosting environment for the VNFC instance that involve additional entities or processes.
monitoring_parameters	no	list of toska.dataty pes.nfv.Vnfc MonitoringP arameter		Describes monitoring parameters applicable to a VNFC based on this VDU.
configurable_properties	no	map of toska.dataty pes.nfv.Vnfc Configurabl eProperties		Describes the configurable properties of all VNFC instances based on this VDU.

Name	Required	Type	Constraints	Description
boot_data	no	string		Contains a string or a URL to a file contained in the VNF package used to customize a virtualised compute resource at boot time. The bootData may contain variable parts that are replaced by deployment specific values before being sent to the VIM. For the parameters with "volatile" values, the parameters of each variable part shall be declared in the <code>tosca.datatypes.nfv.VnfOperationAdditionalParameters</code> ; for parameters with "persistent" values during the lifetime of the VNF instance, the parameters shall be declared in the <code>tosca.datatypes.nfv.VnfInfoModifiableAttributesExtensions</code> .
vdu_profile	yes	<code>tosca.datatypes.nfv.VduProfile</code>		Defines additional instantiation data for the <code>VDU.Compute</code> node.
sw_image_data	no	<code>tosca.datatypes.nfv.SwImageData</code>		Defines information related to a <code>SwImage</code> artifact used by this <code>Vdu.Compute</code> node.

### 6.8.3.3 Attributes

None.

### 6.8.3.4 Requirements

The requirements of the `Vdu.Compute` node type shall comply with the provisions set out in table 6.8.3.4-1.

**Table 6.8.3.4-1: Requirements**

Name	Required	Required capability type	Constraints	Description
virtual_storage	no	<code>tosca.capabilities.nfv.VirtualStorage</code>		Describes storage requirements for a <code>virtual_storage</code> instance attached to the virtualisation container created from <code>virtual_compute</code> defined for this <code>vdu</code>

### 6.8.3.5 Capabilities

The capabilities of the `Vdu.Compute` node type shall comply with the provisions set out in table 6.8.3.5-1.

**Table 6.8.3.5-1: Capabilities**

Name	Type	Constraints	Description
virtual_compute	<code>tosca.capabilities.nfv.VirtualCompute</code>		Describes virtual compute resources capabilities.
virtual_binding	<code>tosca.capabilities.nfv.VirtualBindable</code>		Defines ability of <code>VirtualBindable</code> . See note.
NOTE:	Due to current grammar limitations in TOSCA-Simple-Profile-YAML-v1.2 [3], the <code>Vdu.Compute</code> node template cannot limit the number of the <code>virtual_binding</code> capability occurrences. It is the VNFD author's responsibility to make sure that the number of <code>VduCps</code> connected to a particular <code>Vdu.Compute</code> node is compatible with the virtualised resources assigned to this node.		

### 6.8.3.6 Definition

The syntax of the Vdu.Compute node type shall comply with the following definition:

```

tosca.nodes.nfv.Vdu.Compute:
  derived_from: toska.nodes.Root
  description: Describes the virtual compute part of a VDU which is a construct
  supporting the description of the deployment and operational behavior of a VNFC
  properties:
    name:
      type: string
      description: Human readable name of the VDU
      required: true
    description:
      type: string
      description: Human readable description of the VDU
      required: true
    boot_order:
      type: list # explicit index (boot index) not necessary, contrary to IFA011
      description: References a node template name from which a valid boot device is
  created
    required: false
    entry_schema:
      type: string
    nfvi_constraints:
      type: list
      description: Describes constraints on the NFVI for the VNFC instance(s) created
  from this VDU
    required: false
    entry_schema:
      type: string
    monitoring_parameters:
      type: list
      description: Describes monitoring parameters applicable to a VNFC instantiated
  from this VDU
    required: false
    entry_schema:
      type: toska.datatypes.nfv.VnfcMonitoringParameter
  #configurable_properties:
    #type: toska.datatypes.nfv.VnfcConfigurableProperties
    #required: false
    # derived types are expected to introduce
    # configurable_properties with its type derived from
    # toska.datatypes.nfv.VnfcConfigurableProperties
  vdu_profile:
    type: toska.datatypes.nfv.VduProfile
    description: Defines additional instantiation data for the VDU.Compute node
    required: true
  sw_image_data:
    type: toska.datatypes.nfv.SwImageData
    description: Defines information related to a SwImage artifact used by this
  Vdu.Compute node
    required: false # property is required when the node template has an associated
  artifact of type toska.artifacts.nfv.SwImage and not required otherwise
  boot_data:
    type: string
    description: Contains a string or a URL to a file contained in the VNF package
  used to customize a virtualised compute resource at boot time. The bootData may
  contain variable parts that are replaced by deployment specific values before being
  sent to the VIM.
    required: false

```



```

capabilities:
  virtual_compute:
    type: toska.capabilities.nfv.VirtualCompute
    occurrences: [ 1, 1 ]
  virtual_binding:
    type: toska.capabilities.nfv.VirtualBindable
    occurrences: [ 1, UNBOUNDED ]
requirements:
  - virtual_storage:
    capability: toska.capabilities.nfv.VirtualStorage
    relationship: toska.relationships.nfv.AttachesTo
    occurrences: [ 0, UNBOUNDED ]

```

### 6.8.3.7 Additional requirements

Node templates of type `tosca.nodes.nfv.Vdu.Compute` may contain an artifact definition of type `tosca.artifacts.nfv.SwImage`. There shall be a maximum number of one such artifact definition in a `tosca.nodes.nfv.Vdu.Compute` node template. The node template name of type `tosca.nodes.nfv.Vdu.Compute` fulfills the purpose of the "SwImageDescId" attribute of the `SwImageDesc` information element in ETSI GS NFV-IFA 011 [1] and hence it will be used in APIs to identify the software image id from the VNFD perspective. See example in clause 6.8.3.8.

VNF Providers shall use the datatype `tosca.datatypes.nfv.VnfcConfigurableProperties` to derive the type of `configurable_properties` when introducing this property with VDU-specific elements in a `Vdu.Compute` specific node type.

The node template name of type `tosca.nodes.nfv.Vdu.Compute` fulfills the purpose of the 'virtualComputeDescId' attribute of the `virtualComputeDesc` information element in ETSI GS NFV-IFA 011 [1] and hence it will be used in APIs to identify the virtual compute id (`vnfdVirtualComputeDescId`).

**NOTE:** The use of the node template name of type `tosca.nodes.nfv.Vdu.Compute` for the 'virtualComputeDescId' attribute of the `virtualComputeDesc` information element in ETSI GS NFV-IFA 011 [1] implies in the present document version a one-to-one mapping of `virtualComputeDesc` with VDU. This deviates from the ETSI GS NFV-IFA 011 [1] modelling that defines a mapping where a `virtualComputeDesc` can be reused by one or more VDU, i.e. it implies a one-to-many mapping of `virtualComputeDesc` with VDU. This can have an impact in the determination of the number of compute flavours needed to be created with the VIM.

### 6.8.3.8 Example

This example illustrates boot data values as string.

```

tosca_definitions_version: toska_simple_yaml_1_2
..
node_types:
  mycompany.nodes.nfv.SunshineDB.1_0.1_0:
    derived_from: toska.nodes.nfv.VNF
    properties:
      ..
      modifiable_attributes:
        type: mycompany.datatypes.nfv.VnfInfoModifiableAttributes
      ..
data_types:
  mycompany.datatypes.nfv.VnfInfoModifiableAttributes:
    derived_from: toska.datatypes.nfv.VnfInfoModifiableAttributes
    properties:
      extensions:
        type: mycompany.datatypes.nfv.VnfInfoModifiableAttributesExtensions
    required: false

```

```

mycompany.datatypes.nfv.VnfInfoModifiableAttributesExtensions:
  derived_from: toska.datatypes.nfv.VnfInfoModifiableAttributesExtensions
  properties:
    http_proxy:
      type: string
      required: true
    https_proxy:
      type: string
      required: false

topology_template:
  inputs:
    modifiable_attributes:
      type: mycompany.datatypes.nfv.VnfInfoModifiableAttributes

substitution_mappings:
  node_type: mycompany.nodes.nfv.SunshineDB.1_0.1_0
  ..

node_templates:
  vnf:
    type: mycompany.nodes.nfv.SunshineDB.1_0.1_0
    properties:
      ..
      modifiable_attributes: { get_input: modifiable_attributes }

dbBackend:
  type: toska.nodes.nfv.Vdu.Compute
  properties:
    ..
    boot_data: { concat: [
      "#!/bin/bash\n",
      "echo setting HTTP proxy to: ", { get_property: [vnf, modifiable_attributes,
extensions, http_proxy ] }, "\n",
      "...
    ] }
    ..

```

This example illustrates fetching the boot data value from a file.

```

tosca_definitions_version: toska_simple_yaml_1_2
..
topology_template:
  ..

node_templates:
  dbBackend:
    type: toska.nodes.nfv.Vdu.Compute
    properties:
      ..
      boot_data: { get_artifact: [ SELF, user_data ] }
      ..
    artifacts:
      user_data:
        file: db-backend-init.yaml
        type: toska.artifacts.File
      ..

```

This example illustrates the association of a software image artifact to a Vdu.Compute node. The name of the artifact "dbBackend" will be used in external APIs to identify the image.

```

tosca_definitions_version: tosca_simple_yaml_1_2
..
topology_template:
  ..
  node_templates:
    dbBackend:
      type: tosca.nodes.nfv.Vdu.Compute
      properties:
        ..
        sw_image_data:
          ..
      artifacts:
        sw_image:
          type: tosca.artifacts.nfv.SwImage
          file: images/dbBackend.v1.0.1.qcow2
        ..

```

This example illustrates the association of a software image artifact to more than Vdu.Compute nodes. The name of the artifact "dbBackend and oamService" will be used in external APIs to identify the image of each Vdu.Compute node.

```

tosca_definitions_version: tosca_simple_yaml_1_2
..
topology_template:
  ..
  node_templates:
    dbBackend:
      type: tosca.nodes.nfv.Vdu.Compute
      properties:
        ..
        sw_image_data:
          ..
      artifacts:
        sw_image:
          type: tosca.artifacts.nfv.SwImage
          file: images/dbBackend.v1.0.1.qcow2

    oamService:
      type: tosca.nodes.nfv.Vdu.Compute
      properties:
        ..
        sw_image_data:
          ..
      artifacts:
        sw_image:
          type: tosca.artifacts.nfv.SwImage
          file: images/dbBackend.v1.0.1.qcow2
        ..

```

## 6.8.4 tosca.nodes.nfv.Vdu.VirtualBlockStorage

### 6.8.4.1 Description

The VirtualBlockStorage node type describes the specifications of requirements related to virtual block storage resources, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.8.4.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.8.4.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VirtualBlockStorage
<b>Type Qualified Name</b>	toscanfv:VirtualBlockStorage
<b>Type URI</b>	tosca.nodes.nfv.Vdu.VirtualBlockStorage

### 6.8.4.2 Properties

The properties of the VirtualBlockStorage node type shall comply with the provisions set out in table 6.8.4.2-1.

**Table 6.8.4.2-1: Properties**

Name	Required	Type	Constraints	Description
virtual_block_storage_data	yes	tosca.datatypes.nfv.VirtualBlockStorage		Describes the block storage characteristics.
sw_image_data	no	tosca.datatypes.nfv.SwImageData		Defines information related to a SwImage artifact used by this Vdu.Compute node.

### 6.8.4.3 Attributes

None

### 6.8.4.4 Requirements

None.

### 6.8.4.5 Capabilities

The capabilities of the VirtualBlockStorage node type shall comply with the provisions set out in table 6.8.4.5-1.

**Table 6.8.4.5-1: Capabilities**

Name	Type	Constraints	Description
virtual_storage	tosca.capabilities.nfv.VirtualStorage		Defines the capabilities of virtual_storage.

### 6.8.4.6 Definition

The syntax of the VirtualBlockStorage node type shall comply with the following definition:

```
tosca.nodes.nfv.Vdu.VirtualBlockStorage:
  derived_from: tosca.nodes.Root
  description: This node type describes the specifications of requirements related to
  virtual block storage resources
  properties:
    virtual_block_storage_data:
      type: tosca.datatypes.nfv.VirtualBlockStorageData
      description: Describes the block storage characteristics.
```

```

required: true
sw_image_data:
  type: toska.datatypes.nfv.SwImageData
  description: Defines information related to a SwImage artifact used by this
Vdu.Compute node.
  required: false # property is required when the node template has an associated
artifact of type toska.artifacts.nfv.SwImage and not required otherwise
capabilities:
  virtual_storage:
    type: toska.capabilities.nfv.VirtualStorage
    description: Defines the capabilities of virtual_storage.

```

#### 6.8.4.7 Additional requirements

Node templates of type `tosca.nodes.nfv.Vdu.VirtualBlockStorage` may contain an artifact definition of type `tosca.artifacts.nfv.SwImage`. There shall be a maximum number of one such artifact definition in a `tosca.nodes.nfv.Vdu.VirtualBlockStorage` node template. The node template name of type `tosca.nodes.nfv.Vdu.VirtualBlockStorage` fulfills the purpose of the "SwImageDescId" attribute of the `SwImageDesc` information element in ETSI GS NFV-IFA 011 [1] and hence it will be used in APIs to identify the software image id from the VNFD descriptor. See example in clause 6.8.3.8.

### 6.8.5 `tosca.nodes.nfv.Vdu.VirtualObjectStorage`

#### 6.8.5.1 Description

The `VirtualObjectStorage` node type describes the specifications of requirements related to virtual object storage resources, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.8.5.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.8.5.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>VirtualObjectStorage</code>
<b>Type Qualified Name</b>	<code>toscanfv:VirtualObjectStorage</code>
<b>Type URI</b>	<code>tosca.nodes.nfv.Vdu.VirtualObjectStorage</code>

#### 6.8.5.2 Properties

The properties of the `VirtualObjectStorage` node type shall comply with the provisions set out in table 6.8.5.2-1.

**Table 6.8.5.2-1: Properties**

Name	Required	Type	Constraints	Description
<code>virtual_object_storage_data</code>	yes	<code>tosca.datatypes.nfv.VirtualObjectStorageData</code>		Describes the object storage characteristics.

#### 6.8.5.3 Attributes

None.

#### 6.8.5.4 Requirements

None.

#### 6.8.5.5 Capabilities

The capabilities of the `VirtualObjectStorage` node type shall comply with the provisions set out in table 6.8.5.5-1.

**Table 6.8.5.5-1: Capabilities**

Name	Type	Constraints	Description
virtual_storage	tosca.capabilities.nfv.VirtualStorage		Defines the capabilities of virtual_storage.

### 6.8.5.6 Definition

The syntax of the VirtualObjectStorage node type shall comply with the following definition:

```
tosca.nodes.nfv.Vdu.VirtualObjectStorage:
  derived_from: toska.nodes.Root
  description: This node type describes the specifications of requirements related
to virtual object storage resources
  properties:
    virtual_object_storage_data:
      type: toska.datatypes.nfv.VirtualObjectStorageData
      description: Describes the object storage characteristics.
      required: true
  capabilities:
    virtual_storage:
      type: toska.capabilities.nfv.VirtualStorage
      description: Defines the capabilities of virtual_storage.
```

### 6.8.5.7 Additional requirements

None.

## 6.8.6 toska.nodes.nfv.Vdu.VirtualFileStorage

### 6.8.6.1 Description

The VirtualFileStorage node type describes the specifications of requirements related to virtual file storage resources, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.8.6.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.8.6.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VirtualFileStorage
<b>Type Qualified Name</b>	toscanfv:VirtualFileStorage
<b>Type URI</b>	tosca.nodes.nfv.Vdu.VirtualFileStorage

### 6.8.6.2 Properties

The properties of the VirtualFileStorage node type shall comply with the provisions set out in table 6.8.6.2-1.

**Table 6.8.6.2-1: Properties**

Name	Required	Type	Constraints	Description
virtual_file_storage_data	yes	tosca.datatypes.nfv.VirtualFileStorageData		Describes the file storage characteristics.

### 6.8.6.3 Attributes

None.

#### 6.8.6.4 Requirements

The requirements of the VirtualFileStorage node type shall comply with the provisions set out in table 6.8.6.4-1.

**Table 6.8.6.4-1: Requirements**

Name	Required	Type	Constraints	Description
virtual_link	yes	tosca.capabilities.nfv.VirtualLinkable		Describes the requirements for linking to virtual link

#### 6.8.6.5 Capabilities

The capabilities of the VirtualFileStorage node type shall comply with the provisions set out in table 6.8.6.5-1.

**Table 6.8.6.5-1: Capabilities**

Name	Type	Constraints	Description
virtual_storage	tosca.capabilities.nfv.VirtualStorage		Defines the capabilities of virtual_storage.

#### 6.8.6.6 Definition

The syntax of the VirtualFileStorage node type shall comply with the following definition:

```

tosca.nodes.nfv.Vdu.VirtualFileStorage:
  derived_from: tosca.nodes.Root
  description: This node type describes the specifications of requirements
related to virtual file storage resources
  properties:
    virtual_file_storage_data:
      type: tosca.datatypes.nfv.VirtualFileStorageData
      description: Describes the file storage characteristics.
      required: true
  capabilities:
    virtual_storage:
      type: tosca.capabilities.nfv.VirtualStorage
      #description: Defines the capabilities of virtual_storage.
  requirements:
    - virtual_link:
      capability: tosca.capabilities.nfv.VirtualLinkable
      relationship: tosca.relationships.nfv.VirtualLinksTo
      # description: Describes the requirements for linking to virtual link

```

#### 6.8.6.7 Additional requirements

None.

### 6.8.7 tosca.nodes.nfv.Cp

#### 6.8.7.1 Description

A Cp node type represents the Cpd information element as defined in ETSI GS NFV-IFA 011 [1], which describes network connectivity to a compute resource or a VL. This is an abstract type used as parent for the various Cp node types. Table 6.8.7.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.8.7.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	Cp
<b>Type Qualified Name</b>	toscanfv:Cp
<b>Type URI</b>	tosca.nodes.nfv.Cp

### 6.8.7.2 Properties

The properties of the Cp node type shall comply with the provisions set out in table 6.8.7.2-1.

**Table 6.8.7.2-1: Properties**

Name	Required	Type	Constraints	Description
layer_protocols	yes	list of string	Valid values: ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire	Identifies which protocol the connection point uses for connectivity purposes.
role	no	string	valid values: root, leaf	Identifies the role of the port in the context of the traffic flow patterns in the VNF or parent NS. For example a VNF with a tree flow pattern within the VNF will have legal cpRoles of ROOT and LEAF.
description	no	string		Provides human-readable information on the purpose of the connection point (e.g. connection point for control plane traffic).
protocol	no	list of tosca.datatypes.nfv.CpProtocolData		Provides information on the addresses to be assigned to the connection point(s) instantiated from this Connection Point Descriptor.
trunk_mode	no	boolean		Information about whether the CP instantiated from this Cp is in Trunk mode (802.1Q or other). When operating in "trunk mode", the Cp is capable of carrying traffic for several VLANs. Absence of this property implies that trunkMode is not configured for the Cp i.e. It is equivalent to boolean value "false".

### 6.8.7.3 Attributes

None.

### 6.8.7.4 Requirements

None.

### 6.8.7.5 Capabilities

None.



### 6.8.7.6 Definition

The syntax of the Cp node type shall comply with the following definition:

```

tosca.nodes.nfv.Cp:
  derived_from: toska.nodes.Root
  description: Provides information regarding the purpose of the connection point
  properties:
    layer_protocols:
      type: list
      description: Identifies which protocol the connection point uses for connectivity
    purposes:
      required: true
      entry_schema:
        type: string
        constraints:
          - valid_values: [ ethernet, mpls, odu2, ipv4, ipv6, pseudo-wire ]
      role: #Name in ETSI GS NFV-IFA 011 v0.7.3: cpRole
      type: string
      description: Identifies the role of the port in the context of the traffic flow
    patterns in the VNF or parent NS
      required: false
      constraints:
        - valid_values: [ root, leaf ]
      description:
        type: string
        description: Provides human-readable information on the purpose of the connection
    point
      required: false
    protocol:
      type: list
      description: Provides information on the addresses to be assigned to the
    connection point(s) instantiated from this Connection Point Descriptor
      required: false
      entry_schema:
        type: toska.datatypes.nfv.CpProtocolData
    trunk_mode:
      type: boolean
      description: Provides information about whether the CP instantiated from this Cp
    is in Trunk mode (802.1Q or other). When operating in "trunk mode", the Cp is capable
    of carrying traffic for several VLANs. Absence of this property implies that trunkMode
    is not configured for the Cp i.e. It is equivalent to boolean value "false".
      required: false

```

### 6.8.7.7 Additional requirements

The 'protocol' property shall not be included in a derived PnfExtCp node and shall be included in all other cases.

## 6.8.8 toska.nodes.nfv.VduCp

### 6.8.8.1 Description

A VduCp node type represents the VduCpd information element as defined in ETSI GS NFV-IFA 011 [1], which describes network connectivity between a VNFC instance (based on VDU) and an internal VL. Table 6.8.8.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.8.8.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VduCp
<b>Type Qualified Name</b>	toscanfv:VduCp
<b>Type URI</b>	tosca.nodes.nfv.VduCp

### 6.8.8.2 Properties

The properties of the VduCp node type shall comply with the provisions set out in table 6.8.8.2-1.

**Table 6.8.8.2-1: Properties**

Name	Required	Type	Constraints	Description
bitrate_requirement	no	integer	greater_or_equal: 0	Bitrate requirement in bit per second on this connection point.
virtual_network_interface_requirements	no	list of tosca.data types.nfv. VirtualNetworkInterfaceRequirements		Specifies requirements on a virtual network interface realizing the CPs instantiated from this CPD.
order	no	integer	greater_or_equal: 0	The order of the NIC on the compute instance (e.g. eth2).  See note.  compute (aka multi vNICs) and ordering is desired, it is <i>*mandatory*</i> that all ports will be set with an order value and. The order values shall represent a positive, arithmetic progression that starts with 0 (e.g. 0, 1, 2, ..., n).  If the property is not present, it shall be left to the VIM to assign the value when creating the instance.
vnic_type	no	string	Allowed values: normal, virtio, direct-physical	Describes the type of the virtual network interface realizing the CPs instantiated from this CPD. This is used to determine which mechanism driver(s) to be used to bind the port.

NOTE: When binding more than one port to a single.

### 6.8.8.3 Attributes

None.

### 6.8.8.4 Requirements

The requirements of the VduCp node type shall comply with the provisions set out in table 6.8.8.4-1.

**Table 6.8.8.4-1: Requirements**

Name	Required	Type	Constraints	Description
virtual_binding	yes	tosca.capabilities.nfv.VirtualBindable		Describe the requirement for binding with VDU
virtual_link	no	tosca.capabilities.nfv.VirtualLinkable		Describes the requirements for linking to virtual link

### 6.8.8.5 Definition

The syntax of the VduCp node type shall comply with the following definition:

```

tosca.nodes.nfv.VduCp:
  derived_from: toska.nodes.nfv.Cp
  description: describes network connectivity between a VNFC instance based on this
VDU and an internal VL
  properties:
    bitrate_requirement:
      type: integer # in bits per second
      description: Bitrate requirement in bit per second on this connection point
      required: false
      constraints:
        - greater_or_equal: 0
    virtual_network_interface_requirements:
      type: list
      description: Specifies requirements on a virtual network interface realising
the CPs instantiated from this CPD
      required: false
      entry_schema:
        type: toska.datatypes.nfv.VirtualNetworkInterfaceRequirements
    order:
      type: integer
      description: The order of the NIC on the compute instance (e.g.eth2)
      required: false
      constraints:
        - greater_or_equal: 0
    vnic_type:
      type: string
      description: Describes the type of the virtual network interface realizing the
CPs instantiated from this CPD
      required: false
      constraints:
        - valid_values: [ normal, virtio, direct-physical ]
    requirements:
      - virtual_link:
          capability: toska.capabilities.nfv.VirtualLinkable
          relationship: toska.relationships.nfv.VirtualLinksTo
      - virtual_binding:
          capability: toska.capabilities.nfv.VirtualBindable
          relationship: toska.relationships.nfv.VirtualBindsTo
          node: toska.nodes.nfv.Vdu.Compute

```

## 6.8.9 toska.nodes.nfv.VnfVirtualLink

### 6.8.9.1 Description

The VnfVirtualLink node type represents the VnfVirtualLinkDesc information element as defined in ETSI GS NFV-IFA 011 [1], which describes the information about an internal VNF VL. Table 6.8.9.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.8.9.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VnfVirtualLink
<b>Type Qualified Name</b>	toscanfv:VnfVirtualLink
<b>Type URI</b>	tosca.nodes.nfv.VnfVirtualLink

### 6.8.9.2 Properties

The properties of the VnfVirtualLink node type shall comply with the provisions set out in table 6.8.9.2-1.

**Table 6.8.9.2-1: Properties**

Name	Required	Type	Constraints	Description
connectivity_type	yes	ConnectivityType		Specifies the protocol exposed by the VL and the flow pattern supported by the VL.
description	no	string		Provides human-readable information on the purpose of the VL (e.g. control plane traffic).
test_access	no	list of string	Valid values: passive_monitoring, active_loopback	Test access facilities available on the VL.
vl_profile	yes	tosca.datatypes.nfv.VIProfile		Defines additional data for the VL: maximum and minimum bit rate requirements and QoS.
monitoring_parameters	no	list of tosca.datatypes.nfv.VirtualLinkMonitoringParameter		Describe monitoring parameters applicable to a VL instantiated from this node type.

### 6.8.9.3 Requirements

None.

### 6.8.9.4 Capabilities

The capabilities of the VnfVirtualLink node type shall comply with the provisions set out in table 6.8.9.4-1.

**Table 6.8.9.4-1: Capabilities**

Name	Type	Constraints	Description
virtual_linkable	tosca.capabilities.nfv.VirtualLinkable		Defines ability of VirtualLinkable.

### 6.8.9.5 Definition

The syntax of the VnfVirtualLink node type shall comply with the following definition:

```

tosca.nodes.nfv.VnfVirtualLink:
  derived_from: tosca.nodes.Root
  description: Describes the information about an internal VNF VL
  properties:
    connectivity_type:
      type: tosca.datatypes.nfv.ConnectivityType
      description: Specifies the protocol exposed by the VL and the flow pattern
supported by the VL
      required: true
    description:
      type: string
      description: Provides human-readable information on the purpose of the VL
      required: false
    test_access:
      type: list
      description: Test access facilities available on the VL
      required: false

```

```

entry_schema:
  type: string
  constraints:
    - valid_values: [ passive_monitoring, active_loopback ]
vl_profile:
  type: toska.datatypes.nfv.VlProfile
  description: Defines additional data for the VL
  required: true
monitoring_parameters:
  type: list
  entry_schema:
    type: toska.datatypes.nfv.VirtualLinkMonitoringParameter
    description: Describes monitoring parameters applicable to the VL
    required: false
capabilities:
  virtual_linkable:
    type: toska.capabilities.nfv.VirtualLinkable

```

## 6.9 Group Types

### 6.9.1 toska.groups.nfv.PlacementGroup

#### 6.9.1.1 Description

PlacementGroup is used for describing the affinity or anti-affinity relationship applicable between the virtualisation containers to be created based on different VDUs, or between internal VLs to be created based on different VnfVirtualLinkDesc(s). Table 6.9.1.1-1 specifies the declared names for this group type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.9.1.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	PlacementGroup
<b>Type Qualified Name</b>	toscanfv:PlacementGroup
<b>Type URI</b>	tosca.groups.nfv.PlacementGroup

#### 6.9.1.2 Properties

The properties of the PlacementGroup group type shall comply with the provisions set out in table 6.9.1.2-1.

**Table 6.9.1.2-1: Properties**

Name	Required	Type	Constraints	Description
description	yes	string		Human readable description of the group

#### 6.9.1.3 Definition

The syntax of the PlacementGroup group type shall comply with the following definition:

```

tosca.groups.nfv.PlacementGroup:
  derived_from: toska.groups.Root
  description: PlacementGroup is used for describing the affinity or anti-affinity
relationship applicable between the virtualization containers to be created based on
different VDUs, or between internal VLs to be created based on different
VnfVirtualLinkDesc(s)

```

```

properties:
  description:
    type: string
    description: Human readable description of the group
    required: true
  members: [ tosca.nodes.nfv.Vdu.Compute, tosca.nodes.nfv.VnfVirtualLink ]

```

#### 6.9.1.4 Additional Requirements

A group with type `tosca.groups.nfv.PlacementGroup` shall contain more than one member when used as the target of an `AffinityRule` or `AntiAffinityRule` policy.

#### 6.9.1.5 Examples

See clause 6.10.10.5.

## 6.10 Policy Types

### 6.10.1 `tosca.policies.nfv.InstantiationLevels`

#### 6.10.1.1 Description

The `InstantiationLevels` type is a policy type representing all the instantiation levels of resources to be instantiated within a deployment flavour and including default instantiation level in term of the number of VNFC instances to be created as defined in ETSI GS NFV-IFA 011 [1]. This policy concerns the whole VNF (deployment flavour) represented by the `topology_template` and thus has no explicit target list. Table 6.10.1.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.10.1.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>InstantiationLevels</code>
<b>Type Qualified Name</b>	<code>toscanfv:InstantiationLevels</code>
<b>Type URI</b>	<code>tosca.policies.nfv.InstantiationLevels</code>

#### 6.10.1.2 Properties

The properties of the `InstantiationLevels` policy type shall comply with the provisions set out in table 6.10.1.2-1.

**Table 6.10.1.2-1: Properties**

Name	Required	Type	Constraints	Description
<code>levels</code>	yes	map of <code>tosca.datatypes.nfv.InstantiationLevel</code>		Describes the various levels of resources that can be used to instantiate the VNF using this flavour.
<code>default_level</code>	no	string		The default instantiation level for this flavour.

### 6.10.1.3 Definition

The syntax of the InstantiationLevels policy type shall comply with the following definition:

```

tosca.policies.nfv.InstantiationLevels:
  derived_from: tosca.policies.Root
  description: The InstantiationLevels type is a policy type representing all the
instantiation levels of resources to be instantiated within a deployment flavour and
including default instantiation level in term of the number of VNFC instances to be
created as defined in ETSI GS NFV-IFA 011 [1].
  properties:
    levels:
      type: map # key: levelId
      description: Describes the various levels of resources that can be used to
instantiate the VNF using this flavour.
      required: true
      entry_schema:
        type: tosca.datatypes.nfv.InstantiationLevel
      constraints:
        - min_length: 1
    default_level:
      type: string # levelId
      description: The default instantiation level for this flavour.
      required: false # required if multiple entries in levels

```

## 6.10.2 tosca.policies.nfv.VduInstantiationLevels

### 6.10.2.1 Description

The VduInstantiationLevels type is a policy type representing all the instantiation levels of resources to be instantiated within a deployment flavour in term of the number of VNFC instances to be created from each vdu.Compute. as defined in ETSI GS NFV-IFA 011 [1].

Table 6.10.2.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.10.2.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VduInstantiationLevels
<b>Type Qualified Name</b>	toscanfv:VduInstantiationLevels
<b>Type URI</b>	tosca.policies.nfv.VduInstantiationLevels

### 6.10.2.2 Properties

The properties of the VduInstantiationLevels policy type shall comply with the provisions set out in table 6.10.2.2-1.

**Table 6.10.2.2-1: Properties**

Name	Required	Type	Constraints	Description
levels	yes	map of tosca.d atatype s.nfv.Vd uLevel		Describes the Vdu.Compute levels of resources that can be used to instantiate the VNF using this flavour.

### 6.10.2.3 Definition

The syntax of the VduInstantiationLevels policy type shall comply with the following definition:

```

tosca.policies.nfv.VduInstantiationLevels:
  derived_from: tosca.policies.Root
  description: The VduInstantiationLevels type is a policy type representing all the
instantiation levels of resources to be instantiated within a deployment flavour in
term of the number of VNFC instances to be created from each vdu.Compute. as defined
in ETSI GS NFV-IFA 011 [1]
  properties:
    levels:
      type: map # key: levelId
      description: Describes the Vdu.Compute levels of resources that can be used to
instantiate the VNF using this flavour
      required: true
      entry_schema:
        type: tosca.datatypes.nfv.VduLevel
      constraints:
        - min_length: 1
    targets: [ tosca.nodes.nfv.Vdu.Compute ]

```

### 6.10.2.4 Additional Requirements

A VduInstantiationLevels policy shall contain an entry for each instantiation level (and only for them) defined in the InstantiationLevels policy.

## 6.10.3 tosca.policies.nfv.VirtualLinkInstantiationLevels

### 6.10.3.1 Description

The VirtualLinkInstantiationLevels type is a policy type representing all the instantiation levels of virtual link resources to be instantiated within a deployment flavour as defined in ETSI GS NFV-IFA 011 [1]. Table 6.10.3.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.10.3.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VirtualLinkInstantiationLevels
<b>Type Qualified Name</b>	toscanfv:VirtualLinkInstantiationLevels
<b>Type URI</b>	tosca.policies.nfv.VirtualLinkInstantiationLevels

### 6.10.3.2 Properties

The properties of the VirtualLinkInstantiationLevels policy type shall comply with the provisions set out in table 6.10.3.2-1.

**Table 6.10.3.2-1: Properties**

Name	Required	Type	Constraints	Description
levels	yes	map of tosca.dataty pes.nfv.Virtu alLinkBitrate Level		Describes the virtual link levels of resources that can be used to instantiate the VNF using this flavour.



### 6.10.3.3 Definition

The syntax of the VirtualLinkInstantiationLevels policy type shall comply with the following definition:

```

tosca.policies.nfv.VirtualLinkInstantiationLevels:
  derived_from: tosca.policies.Root
  description: The VirtualLinkInstantiationLevels type is a policy type representing
all the instantiation levels of virtual link resources to be instantiated within a
deployment flavour as defined in ETSI GS NFV-IFA 011 [1].
  properties:
    levels:
      type: map # key: levelId
      description: Describes the virtual link levels of resources that can be used to
instantiate the VNF using this flavour.
      required: true
      entry_schema:
        type: tosca.datatypes.nfv.VirtualLinkBitrateLevel
      constraints:
        - min_length: 1
  targets: [ tosca.nodes.nfv.VnfVirtualLink ]

```

### 6.10.3.4 Additional Requirements

A VirtualLinkInstantiationLevels policy shall contain an entry for each instantiation level (and only for them) defined in the InstantiationLevels policy.

## 6.10.4 Void

## 6.10.5 tosca.policies.nfv.ScalingAspects

### 6.10.5.1 Description

The ScalingAspects type is a policy type representing the scaling aspects used for horizontal scaling as defined in ETSI GS NFV-IFA 011 [1]. This policy concerns the whole VNF (deployment flavour) represented by the topology\_template and thus has no explicit target list. Table 6.10.5.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.10.5.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	ScalingAspects
<b>Type Qualified Name</b>	toscanfv:ScalingAspects
<b>Type URI</b>	tosca.policies.nfv.ScalingAspects

### 6.10.5.2 Properties

The properties of the ScalingAspects policy type shall comply with the provisions set out in table 6.10.5.2-1.

**Table 6.10.5.2-1: Properties**

Name	Required	Type	Constraints	Description
aspects	yes	Map of tosca.datatypes.nfv. ScalingAspect		Describe maximum scale level for total number of scaling steps that can be applied to a particular aspect.

### 6.10.5.3 Definition

The syntax of the ScalingAspects policy type shall comply with the following definition:

```

tosca.policies.nfv.ScalingAspects:
  derived_from: toska.policies.Root
  description: The ScalingAspects type is a policy type representing the scaling
  aspects used for horizontal scaling as defined in ETSI GS NFV-IFA 011 [1]
  properties:
    aspects:
      type: map # key: aspectId
      description: Describe maximum scale level for total number of scaling steps that
      can be applied to a particular aspect
      required: true
      entry_schema:
        type: toska.datatypes.nfv.ScalingAspect
      constraints:
        - min_length: 1

```

### 6.10.5.4 Examples

See clause A.6.

## 6.10.6 toska.policies.nfv.VduScalingAspectDeltas

### 6.10.6.1 Description

The VduScalingAspectDeltas type is a policy type representing the Vdu.Compute detail of an aspect deltas used for horizontal scaling, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.10.6.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.10.6.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VduScalingAspectDeltas
<b>Type Qualified Name</b>	toscanfv.VduScalingAspectDeltas
<b>Type URI</b>	tosca.policies.nfv.VduScalingAspectDeltas

### 6.10.6.2 Properties

The properties of the VduScalingAspectDeltas policy type shall comply with the provisions set out in table 6.10.6.2-1.

**Table 6.10.6.2-1: Properties**

Name	Required	Type	Constraints	Description
aspect:	yes	string		Represents the scaling aspect to which this policy applies.
deltas	yes	map of toska.datatypes.nfv.VduLevel		Describes the Vdu.Compute scaling deltas to be applied for every scaling steps of a particular aspect.

### 6.10.6.3 Definition

The syntax of the VduScalingAspectDeltas policy type shall comply with the following definition:

```

tosca.policies.nfv.VduScalingAspectDeltas:
  derived_from: toska.policies.Root
  description: The VduScalingAspectDeltas type is a policy type representing the
  Vdu.Compute detail of an aspect deltas used for horizontal scaling, as defined in ETSI
  GS NFV-IFA 011 [1]
  properties:
    aspect:
      type: string
      description: Represents the scaling aspect to which this policy applies
      required: true
    deltas:
      type: map # key: scalingDeltaId
      description: Describes the Vdu.Compute scaling deltas to be applied for every
      scaling steps of a particular aspect.
      required: true
      entry_schema:
        type: toska.datatypes.nfv.VduLevel
      constraints:
        - min_length: 1
  targets: [ toska.nodes.nfv.Vdu.Compute ]

```

### 6.10.6.4 Examples

See clause A.6.

## 6.10.7 toska.policies.nfv.VirtualLinkBitrateScalingAspectDeltas

### 6.10.7.1 Description

The VirtualLinkBitrateScalingAspectDeltas type is a policy type representing the VnfVirtualLink detail of an aspect deltas used for horizontal scaling, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.10.7.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.10.7.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VirtualLinkBitrateScalingAspectDeltas
<b>Type Qualified Name</b>	toscanfv:VirtualLinkBitrateScalingAspectDeltas
<b>Type URI</b>	tosca.policies.nfv.VirtualLinkBitrateScalingAspectDeltas

### 6.10.7.2 Properties

The properties of the VirtualLinkBitrateScalingAspectDeltas policy type shall comply with the provisions set out in table 6.10.7.2-1.

**Table 6.10.7.2-1: Properties**

Name	Required	Type	Constraints	Description
aspect	yes	string		Represents the scaling aspect to which this policy applies.
aspect_deltas	yes	map of toska.datatypes.nfv.VirtualLinkBitrateLevel		Describes the VnfVirtualLink scaling deltas to be applied for every scaling steps of a particular aspect.

### 6.10.7.3 Definition

The syntax of the VirtualLinkBitrateScalingAspectDeltas policy type shall comply with the following definition:

```

tosca.policies.nfv.VirtualLinkBitrateScalingAspectDeltas:
  derived_from: tosca.policies.Root
  description: The VirtualLinkBitrateScalingAspectDeltas type is a policy type
  representing the VnfVirtualLink detail of an aspect deltas used for horizontal
  scaling, as defined in ETSI GS NFV-IFA 011 [1].
  properties:
    aspect:
      type: string
      description: Represents the scaling aspect to which this policy applies.
      required: true
    deltas:
      type: map # key: scalingDeltaId
      description: Describes the VnfVirtualLink scaling deltas to be applied for every
      scaling steps of a particular aspect.
      required: true
      entry_schema:
        type: tosca.datatypes.nfv.VirtualLinkBitrateLevel
      constraints:
        - min_length: 1
  targets: [ tosca.nodes.nfv.VnfVirtualLink ]

```

### 6.10.7.4 Examples

See clause A.6.

## 6.10.8 tosca.policies.nfv.VduInitialDelta

### 6.10.8.1 Description

The VduInitialDelta type is a policy type representing the Vdu.Compute detail of an initial delta used for horizontal scaling, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.10.8.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.10.8.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VduInitialDelta
<b>Type Qualified Name</b>	toscanfv.VduInitialDelta
<b>Type URI</b>	tosca.policies.nfv.VduInitialDelta

### 6.10.8.2 Properties

The properties of the VduInitialDelta policy type shall comply with the provisions set out in table 6.10.8.2-1.

**Table 6.10.8.2-1: Properties**

Name	Required	Type	Constraints	Description
initial_delta:	yes	tosca.datatypes.nfv.VduLevel		Represents the initial minimum size of the VNF.

### 6.10.8.3 Definition

The syntax of the VduInitialDelta policy type shall comply with the following definition:

```
tosca.policies.nfv.VduInitialDelta:
  derived_from: tosca.policies.Root
  description: The VduInitialDelta type is a policy type representing the
  Vdu.Compute detail of an initial delta used for horizontal scaling, as defined in
  ETSI GS NFV-IFA 011 [1].
  properties:
    initial_delta:
      type: tosca.datatypes.nfv.VduLevel
      description: Represents the initial minimum size of the VNF.
      required: true
  targets: [ tosca.nodes.nfv.Vdu.Compute ]
```

### 6.10.8.4 Examples

See clause A.6.

## 6.10.9 tosca.policies.nfv.VirtualLinkBitrateInitialDelta

### 6.10.9.1 Description

The VirtualLinkBitrateInitialDelta type is a policy type representing the VnfVirtualLink detail of an initial deltas used for horizontal scaling, as defined in ETSI GS NFV-IFA 011 [1]. Table 6.10.9.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.10.9.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	VirtualLinkBitrateInitialDelta
<b>Type Qualified Name</b>	toscanfv:VirtualLinkBitrateInitialDelta
<b>Type URI</b>	tosca.policies.nfv.VirtualLinkBitrateInitialDelta

### 6.10.9.2 Properties

The properties of the VirtualLinkBitrateInitialDelta policy type shall comply with the provisions set out in table 6.10.9.2-1.

**Table 6.10.9.2-1: Properties**

Name	Required	Type	Constraints	Description
initial_delta:	yes	tosca.datatypes.nfv.VirtualLinkBitrateLevel		Represents the initial minimum size of the VNF.

### 6.10.9.3 Definition

The syntax of the VirtualLinkBitrateInitialDelta policy type shall comply with the following definition:

```

tosca.policies.nfv.VirtualLinkBitrateInitialDelta:
  derived_from: tosca.policies.Root
  description: The VirtualLinkBitrateInitialDelta type is a policy type
  representing the VnfVirtualLink detail of an initial deltas used for horizontal
  scaling, as defined in ETSI GS NFV-IFA 011 [1].
  properties:
    initial_delta:
      type: tosca.datatypes.nfv.VirtualLinkBitrateLevel
      description: Represents the initial minimum size of the VNF.
      required: true
  targets: [ tosca.nodes.nfv.VnfVirtualLink ]

```

### 6.10.9.4 Examples

See clause A.6.

## 6.10.10 AffinityRule, AntiAffinityRule

### 6.10.10.1 Description

The AffinityRule or AntiAffinityRule describes the affinity or anti-affinity rules applicable for the defined targets:

- If there is only one node template with node type `tosca.nodes.nfv.Vdu.Compute` or `tosca.nodes.nfv.VnfVirtualLink` set as the targets, the AffinityRule or AntiAffinityRule applies between the virtualisation containers to be created based on a particular VDU, or between internal VLs to be created based on a particular `VnfVirtualLinkDesc` as described in ETSI GS NFV-IFA 011 [1].
- If there are more than one node templates with node type `tosca.nodes.nfv.Vdu.Compute` or `tosca.nodes.nfv.VnfVirtualLink` set as the targets, or a group with type `tosca.groups.nfv.PlacementGroup` which contains more than one members set as targets, the AffinityRule or AntiAffinityRule applies between the virtualisation containers to be created based on different VDUs, or between internal VLs to be created based on different `VnfVirtualLinkDesc(s)` as described in ETSI GS NFV-IFA 011 [1].

Tables 6.10.10.1-1 and 6.10.10.1-2 specify the declared names for the policy types. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.10.10.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	AffinityRule
<b>Type Qualified Name</b>	toscanfv:AffinityRule
<b>Type URI</b>	tosca.policies.nfv.AffinityRule

**Table 6.10.10.1-2: Type name, shorthand, and URI**

<b>Shorthand Name</b>	AntiAffinityRule
<b>Type Qualified Name</b>	toscanfv:AntiAffinityRule
<b>Type URI</b>	tosca.policies.nfv.AntiAffinityRule

### 6.10.10.2 Properties

The properties of the AffinityRule and AntiAffinityRule types shall comply with the provisions set out in table 6.10.10.2-1.

**Table 6.10.10.2-1: Properties**

Name	Required	Type	Constraints	Description
scope	Yes	String	Possible values are "nfvi_pop", "zone", "zone_group", "nfvi_node".	Specifies the scope of the local affinity rule

### 6.10.10.3 targets

The targets of the AffinityRule and AntiAffinityRule policy types shall comply with the provisions set out in table 6.10.10.3-1.

**Table 6.10.10.3-1: Targets**

Name	Required	Type	Constraints	Description
targets	Yes	string[]	Valid types: toasca.nodes.nfv.Vdu.Compute, toasca.nodes.nfv.VnfVirtualLink, toasca.groups.nfv.PlacementGroup	In case of LocalAffinityOrAntiAffinityRule as defined in ETSI GS NFV-IFA 011 [1], the valid type of the targets is toasca.nodes.nfv.Vdu.Compute or toasca.nodes.nfv.VnfVirtualLink. In case of affinityOrAntiAffinityGroup as defined in ETSI GS NFV-IFA 011 [1], the valid types of the targets are: toasca.nodes.nfv.Vdu.Compute and toasca.nodes.nfv.VnfVirtualLink or a toasca.groups.nfv.PlacementGroup.

### 6.10.10.4 Definition

The syntax of the AffinityRule policy type shall comply with the following definition:

```
tosca.policies.nfv.AffinityRule:
  derived_from: toasca.policies.Placement
  description: The AffinityRule describes the affinity rules applicable for the
defined targets
  properties:
    scope:
      type: string
      description: scope of the rule is an NFVI_node, an NFVI_PoP, etc.
      required: true
      constraints:
        - valid_values: [ nfvi_node, zone, zone_group, nfvi_pop ]
    targets: [ toasca.nodes.nfv.Vdu.Compute, toasca.nodes.nfv.VnfVirtualLink,
tosca.groups.nfv.PlacementGroup ]
```

The syntax of the AntiAffinityRule policy type shall comply with the following definition:

```
tosca.policies.nfv.AntiAffinityRule:
  derived_from: toasca.policies.Placement
  description: The AntiAffinityRule describes the anti-affinity rules applicable for
the defined targets
  properties:
    scope:
      type: string
      description: scope of the rule is an NFVI_node, an NFVI_PoP, etc.
      required: true
      constraints:
        - valid_values: [ nfvi_node, zone, zone_group, nfvi_pop ]
```

```
targets: [ tosca.nodes.nfv.Vdu.Compute, tosca.nodes.nfv.VnfVirtualLink,
tosca.groups.nfv.PlacementGroup ]
```

### 6.10.10.5 Examples

The following example template fragments illustrate the concepts:

```
node_templates:
  VDU_1:
    type: tosca.nodes.nfv.Vdu.Compute

policies:
  policy_affinity_local_VDU_1:
    type: tosca.policies.nfv.AffinityRule
    targets: [ VDU_1 ]
    properties:
      scope: nfvi_node
```

The above example illustrates a local affinity rule for VDU\_1.

```
node_template:
  VDU_1:
    type: tosca.nodes.nfv.Vdu.Compute

  VDU_2:
    type: tosca.nodes.nfv.Vdu.Compute

groups:
  affinityOrAntiAffinityGroup_1:
    type: tosca.groups.nfv.PlacementGroup
    members: [ VDU_1, VDU_2 ]

policies:
  policy_antiaffinity_group_1:
    type: tosca.policies.nfv.AntiAffinityRule
    targets: [ affinityOrAntiAffinityGroup_1 ]
    properties:
      scope: nfvi_node
```

The above example illustrates an anti-affinity policy among a group which contains VDU\_1 and VDU\_2 as members.

## 6.10.11 tosca.policies.nfv.SecurityGroupRule

### 6.10.11.1 Description

The SecurityGroupRule type is a policy type specifying the matching criteria for the ingress and/or egress traffic to and from visited connection points as defined in ETSI GS NFV-IFA 011 [1]. Table 6.10.11.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.10.11.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	SecurityGroupRule
<b>Type Qualified Name</b>	toscanfv:SecurityGroupRule
<b>Type URI</b>	tosca.policies.nfv.SecurityGroupRule



### 6.10.11.2 Properties

The properties of the SecurityGroupRule policy type shall comply with the provisions set out in table 6.10.11.2-1.

**Table 6.10.11.2-1: Properties**

Name	Required	Type	Constraints	Description
description	no	string		Human readable description of the security group rule.
direction	no	string	ingress, egress	The direction in which the security group rule is applied. The direction of 'ingress' or 'egress' is specified against the associated CP. I.e. 'ingress' means the packets entering a CP, while 'egress' means the packets sent out of a CP.
ether_type	no	string	ipv4, ipv6	Indicates the protocol carried over the Ethernet layer.
protocol	no	string	see note	Indicates the protocol carried over the IP layer. Permitted values: any protocol defined in the IANA protocol registry [i.5], e.g. TCP, UDP, ICMP, etc.
port_range_min	no	integer	0 - 65535	Indicates minimum port number in the range that is matched by the security group rule. If a value is provided at design-time, this value may be overridden at run-time based on other deployment requirements or constraints.
port_range_max	no	integer	0 – 65535	Indicates maximum port number in the range that is matched by the security group rule. If a value is provided at design-time, this value may be overridden at run-time based on other deployment requirements or constraints.
NOTE:	"protocol" constraints values: hopopt, icmp, igmp, ggp, ipv4, st, tcp, cbt, egp, igp, bbn_rcc_mon, nvp_ii, pup, argus, emcon, xnet, chaos, udp, mux, dcn_meas, hmp, prm, xns_idp, trunk_1, trunk_2, leaf_1, leaf_2, rdp, irtp, iso_tp4, netblt, mfe_nsp, merit_inp, dccp, 3pc, idpr, xtp, ddp, idpr_cmt, tp++, il, ipv6, sdrp, ipv6_route, ipv6_frag, idrp, rsvp, gre, dsr, bna, esp, ah, i_nls, swipe, narp, mobile, tlsp, skip, ipv6_icmp, ipv6_no_nxt, ipv6_opts, cftp, sat_expak, kryptolan, rvd, ippc, sat_mon, visa, ipc, cpnx, cphb, wsn, pvp, br_sat_mon, sun_nd, wb_mon, wb_expak, iso_ip, vmtp, secure_vmtp, vines, ttp, iptm, nsfnet_igp, dgp, tcf, eigrp, ospfigp, sprite_rpc, larp, mtp, ax.25, ipip, micp, scc_sp, etherip, encap, gmtp, ifmp, pnni, pim, aris, scps, qnx, a/n, ip_comp, snp, compaq_peer, ipx_in_ip, vrrp, pgm, l2tp, ddx, iatp, stp, srp, uti, smp, sm, ptp, isis, fire, crtp, crudp, sscopmce, iplt, sps, pipe, sctp, fc, rsvp_e2e_ignore, mobility, udp_lite, mpls_in_ip, manet, hip, shim6, wesp, rohc.			

### 6.10.11.3 Definition

The syntax of the SecurityGroupRule policy type shall comply with the following definition:

```

tosca.policies.nfv.SecurityGroupRule:
  derived_from: toasca.policies.Root
  description: The SecurityGroupRule type is a policy type specified the matching
  criteria for the ingress and/or egress traffic to/from visited connection points
  as defined in ETSI GS NFV-IFA 011 [1].
  properties:
    description:
      type: string
      description: Human readable description of the security group rule.
      required: false
    direction:
      type: string

```

```

description: The direction in which the security group rule is applied. The
direction of 'ingress' or 'egress' is specified against the associated CP. I.e.,
'ingress' means the packets entering a CP, while 'egress' means the packets sent
out of a CP.
required: false
constraints:
  - valid_values: [ ingress, egress ]
default: ingress
ether_type:
  type: string
  description: Indicates the protocol carried over the Ethernet layer.
  required: false
  constraints:
    - valid_values: [ ipv4, ipv6 ]
  default: ipv4
protocol:
  type: string
  description: Indicates the protocol carried over the IP layer. Permitted
values include any protocol defined in the IANA protocol registry, e.g. TCP, UDP,
ICMP, etc.
  required: false
  constraints:
    - valid_values: [ hopopt, icmp, igmp, ggp, ipv4, st, tcp, cbt, egp, igp,
bbn_rcc_mon, nvp_ii, pup, argus, emcon, xnet, chaos, udp, mux, dcn_meas, hmp, prm,
xns_idp, trunk_1, trunk_2, leaf_1, leaf_2, rdp, irtp, iso_tp4, netblt, mfe_nsp,
merit_inp, dccp, 3pc, idpr, xtp, ddp, idpr_cmt, tp++, il, ipv6, sdrp, ipv6_route,
ipv6_frag, idrp, rsvp, gre, dsr, bna, esp, ah, i_nlsp, swipe, nar, mobile, tlsp,
skip, ipv6_icmp, ipv6_no_nxt, ipv6_opts, cftp, sat_expak, kryptolan, rvd, ippc,
sat_mon, visa, ipc, cpnx, cphb, wsn, pvp, br_sat_mon, sun_nd, wb_mon, wb_expak,
iso_ip, vmtp, secure_vmtp, vines, ttp, iptm, nsfnet_igp, dgp, tcf, eigrp, ospfigp,
sprite_rpc, larp, mtp, ax.25, ipip, micp, scc_sp, etherip, encap, gmt, ifmp,
pnni, pim, aris, scps, qnx, a/n, ip_comp, snp, compaq_peer, ipx_in_ip, vrrp, pgm,
l2tp, ddx, iatp, stp, srp, uti, smp, sm, ptp, isis, fire, crtp, crudp, sscopmce,
iplt, sps, pipe, sctp, fc, rsvp_e2e_ignore, mobility, udp_lite, mpls_in_ip, manet,
hip, shim6, wesp, rohc ]
  default: tcp
port_range_min:
  type: integer
  description: Indicates minimum port number in the range that is matched by the
security group rule. If a value is provided at design-time, this value may be
overridden at run-time based on other deployment requirements or constraints.
  required: false
  constraints:
    - greater_or_equal: 0
    - less_or_equal: 65535
  default: 0
port_range_max:
  type: integer
  description: Indicates maximum port number in the range that is matched by the
security group rule. If a value is provided at design-time, this value may be
overridden at run-time based on other deployment requirements or constraints.
  required: false
  constraints:
    - greater_or_equal: 0
    - less_or_equal: 65535
  default: 65535
targets: [ tosca.nodes.nfv.VduCp, tosca.nodes.nfv.VnfExtCp ]

```

### 6.10.11.4 Additional Requirements

If an ingress traffic is accepted, the corresponding egress response shall be allowed, regardless of the security group rules on the egress direction. If an egress traffic is allowed, the corresponding ingress response shall be accepted, regardless of the security group rules on the ingress direction. The design of security group rule follows a permissive model where all security group rules applied to a connection point are dealt with in an "OR" logic fashion, i.e. the traffic is allowed if it matches any security group rule applied to this connection point.

## 6.10.12 `tosca.policies.nfv.SupportedVnfInterface`

### 6.10.12.1 Description

The `SupportedVnfInterface` policy type represents interfaces produced by a VNF, the details to access them and the applicable connection points to use to access these interfaces. It corresponds to the `VnfInterfaceDetails` information element defined in ETSI GS NFV-IFA 011 [1]. Table 6.10.12.1-1 specifies the declared names for this policy type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 6.10.12.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>SupportedVnfInterface</code>
<b>Type Qualified Name</b>	<code>toscanfv:SupportedVnfInterface</code>
<b>Type URI</b>	<code>tosca.policies.nfv.SupportedVnfInterface</code>

### 6.10.12.2 Properties

The properties of the `SupportedVnfInterface` policy type shall comply with the provisions set out in table 6.10.12.2-1.

**Table 6.10.12.2-1: Properties**

Name	Required	Type	Constraints	Description
<code>interface_name</code>	yes	string	Valid values <code>vnf_indicator</code> <code>vnf_configuration</code>	Identifies an interface produced by the VNF.
<code>details</code>	no	<code>tosca.datatypes.nfv.InterfaceDetails</code>		Provide additional data to access the interface endpoint (e.g. API URI prefix).

### 6.10.12.3 Definition

The syntax of the `SupportedVnfInterface` policy type shall comply with the following definition:

```
tosca.policies.nfv.SupportedVnfInterface:
  derived_from: toasca.policies.Root
  description: this policy type represents interfaces produced by a VNF, the details
to access them and the applicable connection points to use to access these interfaces
  properties:
    interface_name:
      type: string
      description: Identifies an interface produced by the VNF.
      required: true
      constraints:
        - valid_values: [ vnf_indicator, vnf_configuration ]
    details:
      type: toasca.datatypes.nfv.InterfaceDetails
      description: Provide additional data to access the interface endpoint
      required: false
  targets: [ toasca.nodes.nfv.VnfExtCp, toasca.nodes.nfv.VduCp ]
```

#### 6.10.12.4 Additional requirements

The valid targets for this policy type shall be the node templates representing the connection point descriptors from which to instantiate the connection point instances through which the interfaces can be accessed. This may be a VnfExtCp node template or a VduCp node template when an internal connection point is re-exposed externally.

#### 6.10.12.5 Example

```

policies:
  policy_interface_1:
    type: toasca.policies.nfv.SupportedVnfInterface
    targets: [ MyVnfmFacingExtCp ]
    properties:
      interface_name: vnf_indicator
    details:
      uri_components:
        scheme: https
        authority:
          host: myvnf.example.com

```

## 6.11 VNFD TOSCA service template design

### 6.11.1 General

The TOSCA service template design for a VNFD in the general case uses two levels of service templates as described in clause 6.11.2. In this design, the top level contains an abstract node template, i.e. without an implementation of the creation operation and is therefore substituted by one of the lower level service templates. This design is applicable regardless of whether the VNF has one or multiple deployment flavours.

In the particular case of a VNF with only one deployment flavour there is an alternative design which is described in clause 6.11.3 and which uses only one service template.

### 6.11.2 Single or multiple deployment flavour design with TOSCA-Simple-Profile-YAML-v1.2

VNFD shall be implemented as one top-level service template and one or multiple lower level service templates, where each lower level service template represents a deployment flavour. In addition, an abstract node type representing the VNF with multiple deployment flavours is also required. The top level service template shall be the main entry point of the VNF package as specified in ETSI GS NFV-SOL 004 [i.6], i.e. the Entry-definitions file. The node type definition is included in a separate YAML file imported from the top and from the lower level files.

See clause A.2 for an example of VNFD design with multiple deployment flavours.

The top level service template shall comply with TOSCA-Simple-Profile-YAML-v1.2 [3] and shall include:

- a) an import statement referencing the TOSCA types definition file as defined in clause B.2;
- b) an import statement referencing the definitions of all low level service template(s);
- c) an import statement referencing a yaml file which contains a VNF specific node type definition which shall be derived from toasca.nodes.nfv.VNF node type as defined in clause 6.8.1;
- d) Optionally, import statements referencing additional VNF-specific files containing type definitions used by this TOSCA service template; and
- e) a topology template with a node template of the VNF specific node type, which shall include the flavour\_id and other properties and the requirements as defined in clause 6.8.1.

The deployment and lifecycle management of instances of this VNF node type is done by means of substitution by any of the lower level service templates.

The lower level service template is an implementable TOSCA service template for the deployment of a specific deployment flavour. The lower level service template shall comply with TOSCA-Simple-Profile-YAML-v1.2 [3] and shall include:

- a) an import statement referencing the TOSCA types definition file as defined in clause B.2;
- b) an import statement referencing a yaml file which contains a VNF specific node type definition which shall be derived from `tosca.nodes.nfv.VNF` node type as defined in clause 6.8.1;
- c) Optionally, import statements referencing additional VNF-specific files containing type definitions used by this TOSCA service template; and
- d) a topology template describing the internal topology of the VNF with:
  - `substitution_mappings` indicating:
    - the same node type as defined in the top level service template;
    - a `flavour_id` property value which identifies the DF corresponding to this low level template within the VNFD; and
    - the mapping of the `virtual_link` requirements on external connection points;
  - a node template referencing the VNF specific node type, implementations of the operations of the LCM interface to be executed by the VNFM, if applicable; and
  - additional node templates of type `Vdu.Compute`, `Vdu.VirtualBlockStorage`, `Vdu.VirtualObjectStorage`, `Vdu.VirtualFileStorage` `VduCp`, etc. that define the topology and composition of the VNF flavour.

The yaml file with the VNF specific node type definition shall be included in the VNF package.

NOTE: The format and structure of a VNF package is defined in ETSI GS NFV-SOL 004 [i.6].

When the `flavour_id` of a VNF has been chosen (e.g. through an input parameter of a VNF instantiation request received by a VNFM) among the values included in the VNF node type imported into the top level service template, it is then used as the property constraint for selecting a particular lower level TOSCA service template inside the VNF package by using abstract node template matching with substitution as described in TOSCA-Simple-Profile-YAML-v1.2 [3].

### 6.11.3 Single deployment flavour design with TOSCA-Simple-Profile-YAML-v1.1

In case of single deployment flavour scenario, TOSCA-Simple-Profile-YAML-v1.1 [4] may be used to describe the VNFD with a single TOSCA service template.

NOTE 1: The present document does not preclude this pattern using TOSCA-Simple-Profile-YAML-v1.2 [3]. In this case one single service template is used as specified in TOSCA-Simple-Profile-YAML-v1.1 [4] and includes:

- a) an import statement referencing the TOSCA types definition file as defined in clause B.2;
- b) Optionally, import statements referencing additional VNF-specific files containing type definitions used by this TOSCA service template;
- c) an abstract VNF node type definition derived from the `tosca.nodes.nfv.VNF` node type, as defined in clause 6.8.1; and
- d) a topology template describing the internal topology of the VNF with:
  - `substitution_mappings` indicating the same abstract VNF node type and the mapping of the `virtual_link` requirements on external connection points;
  - a node template of this abstract VNF node type with the `flavour_id` and other properties and, if applicable, implementations of the operations of the LCM interface to be executed by the VNFM; and

- additional node templates of type Vdu.Compute, Vdu.VirtualBlockStorage, Vdu.VirtualObjectStorage, Vdu.VirtualFileStorage, VduCp, etc. that define the topology and composition of the VNF flavour.

See clause A.5 for an example of VNFD design with single deployment flavour.

NOTE 2: The service template is deployed stand-alone, i.e. without performing a substitution. However including the substitution\_mappings rule indicate its ability to substitute a node template of the abstract VNF node type, which may appear in an NSD.

## 7 NSD TOSCA model

### 7.1 Introduction

The NSD information model specified by ETSI GS NFV-IFA 014 [2] is mapped to the TOSCA concepts. NSD occurrences are represented as TOSCA service templates, as defined in the TOSCA Simple YAML Profile v1.2 [3], to be used by the NFVO for managing the lifecycle of NS instances.

Table 7.1-1 shows an overview of the mapping between the main NSD information elements defined in ETSI GS NFV-IFA 014 [2] and TOSCA types defined in the present document. The definition of all TOSCA types for representing all information elements is described in the following clauses.

NOTE: There is only part of NSD features as specified in ETSI GS NFV-IFA 014 [2] included in this version of the present document, the following are features not included:

- Multiple NS deployment flavours
- NS scaling Aspect
- Multiple Ns instantiation levels
- Nested NS design
- Vnffgd
- NS Monitoring
- VnfIndicators
- Dependencies

**Table 7.1-1: Mapping of ETSI GS NFV-IFA 014 [2] information elements with TOSCA types**

ETSI NFV Information Element ETSI GS NFV-IFA 014 [2]	TOSCA type	Derived from
NSD	tosca.nodes.nfv.NS	tosca.nodes.Root
Sapd	tosca.nodes.nfv.Sap	tosca.nodes.Root
NsVirtualLinkDesc	tosca.nodes.nfv.NsVirtualLink	tosca.nodes.Root
Pnfd	tosca.nodes.nfv.PNF	tosca.nodes.Root
Vnfd	tosca.nodes.nfv.VNF	tosca.nodes.Root

## 7.2 Data Types

### 7.2.1 `tosca.datatype.nfv.ServiceAvailability`

#### 7.2.1.1 Description

`ServiceAvailability` describes the information on the Service Availability Level for a particular virtual link flavour, as defined in ETSI GS NFV-IFA 014 [2] and ETSI GS NFV-REL 001 [i.3]. Table 7.2.1.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 7.2.1.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>ServiceAvailability</code>
<b>Type Qualified Name</b>	<code>toscanfv:ServiceAvailability</code>
<b>Type URI</b>	<code>tosca.datatypes.nfv.ServiceAvailability</code>

#### 7.2.1.2 Properties

The properties of the `ServiceAvailability` data type shall comply with the provisions set out in table 7.2.1.2-1.

**Table 7.2.1.2-1: Properties**

Name	Required	Type	Constraints	Description
<code>level</code>	yes	string	<code>valid_values:</code> [ <code>level1</code> , <code>level2</code> , <code>level3</code> ]	Service available level as defined in [i.3]

#### 7.2.1.3 Definition

The syntax of the `ServiceAvailability` data type shall comply with the following definition:

```
tosca.datatypes.nfv.ServiceAvailability:
  derived_from: toasca.datatypes.Root
  description: service availability
  properties:
    level:
      type: string
      description: service availability levels
      required: true
      constraints:
        - valid_values: [ level1, level2, level3 ]
```

#### 7.2.1.4 Examples

None.

### 7.2.2 `tosca.datatypes.nfv.VnfProfile`

#### 7.2.2.1 Description

The `VnfProfile` data type is defined in clause 6.2.52 of the present document.

## 7.2.3 tosca.datatype.nfv.NsVIPProfile

### 7.2.3.1 Description

The NsVIPProfile data type describes additional instantiation data for a given NsVirtualLink used in a specific NS deployment flavour. Table 7.2.3.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 7.2.3.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	NsVIPProfile
<b>Type Qualified Name</b>	toscanfv:NsVIPProfile
<b>Type URI</b>	tosca.datatypes.nfv.NsVIPProfile

### 7.2.3.2 Properties

The properties of the NsVIPProfile data type shall comply with the provisions set out in table 7.2.3.2-1.

**Table 7.2.3.2-1: Properties**

Name	Required	Type	Constraints	Description
max_bitrate_requirements	yes	tosca.datatypes.nfv.LinkBitrateRequirements		Specifies the maximum bitrate requirements for a VL instantiated according to this profile.
min_bitrate_requirements	yes	tosca.datatypes.nfv.LinkBitrateRequirements		Specifies the minimum bitrate requirements for a VL instantiated according to this profile.
qos	no	tosca.datatypes.nfv.NsVirtualLinkQos		Specifies the QoS requirements of a VL instantiated according to this profile.
service_availability	no	tosca.datatypes.nfv.ServiceAvailability		Network service virtual link service availability levels, as described in ETSI GS NFV-REL 001 [i.3].
NOTE: A virtualLinkDescId property, which exists in ETSI GS NFV-IFA 014 [2] is not needed, as the NsVIPProfile is contained in the NsVirtualLink node.				

### 7.2.3.3 Definition

The syntax of the NsVIPProfile data type shall comply with the following definition:

```
tosca.datatypes.nfv.NsVIPProfile:
  derived_from: tosca.datatypes.Root
  description: Describes additional instantiation data for a given NsVirtualLink used
in a specific NS deployment flavour.
  properties:
    max_bitrate_requirements:
      type: tosca.datatypes.nfv.LinkBitrateRequirements
      description: Specifies the maximum bitrate requirements for a VL instantiated
according to this profile.
      required: true
    min_bitrate_requirements:
      type: tosca.datatypes.nfv.LinkBitrateRequirements
      description: Specifies the minimum bitrate requirements for a VL instantiated
according to this profile.
      required: true
    qos:
      type: tosca.datatypes.nfv.NsVirtualLinkQos
      description: Specifies the QoS requirements of a VL instantiated according to this
profile.
      required: false
```



```

service_availability:
  type: toska.datatypes.nfv.ServiceAvailability
  description: Network service virtual link service availability levels, as
described in ETSI GS NFV-REL 001
  required: false

```

#### 7.2.3.4 Examples

None.

#### 7.2.3.5 Additional Requirements

None.

### 7.2.4 toska.datatypes.nfv.ConnectivityType

#### 7.2.4.1 Description

The ConnectivityType data type is defined in clause 6.2.5 of the present document.

### 7.2.5 toska.datatypes.nfv.NsVirtualLinkQos

#### 7.2.5.1 Description

The NsVirtualLinkQoS describes QoS data type a given NsVirtualLink used in an NS deployment flavour. Table 7.2.5.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 7.2.5.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	NsVirtualLinkQos
<b>Type Qualified Name</b>	toscanfv:NsVirtualLinkQos
<b>Type URI</b>	tosca.datatypes.nfv.NsVirtualLinkQos

#### 7.2.5.2 Properties

The properties of the NsVirtualLinkQos data type shall comply with the provisions set out in table 7.2.5.2-1.

**Table 7.2.5.2-1: Properties**

Name	Required	Type	Constraints	Description
priority	no	integer	greater_or_equal: 0	Specifies the priority level in case of congestion on the underlying physical links.

#### 7.2.5.3 Definition

The syntax of the NsVirtualLinkQos data type shall comply with the following definition:

```

tosca.datatypes.nfv.NsVirtualLinkQos:
  derived_from: toska.datatypes.nfv.Qos
  description: describes QoS data for a given VL used in a VNF deployment flavour
  properties:
    priority:
      type: integer
      constraints:
        - greater_or_equal: 0

```

```
description: Specifies the priority level in case of congestion on the underlying
physical links
required: false
```

#### 7.2.5.4 Examples

None.

#### 7.2.5.5 Additional Requirements

None.

### 7.2.6 `tosca.datatypes.nfv.LinkBitrateRequirements`

#### 7.2.6.1 Description

The `LinkBitrateRequirements` data type is defined in clause 6.2.29 of the present document.

#### 7.2.7 Void

#### 7.2.8 Void

#### 7.2.9 Void

#### 7.2.10 Void

### 7.2.11 `tosca.datatypes.nfv.CpProtocolData`

#### 7.2.11.1 Description

The `CpProtocolData` data type is defined in clause 6.2.36 of the present document.

### 7.2.12 `tosca.datatypes.nfv.AddressData`

#### 7.2.12.1 Description

The `AddressData` data type is defined in clause 6.2.3 of the present document.

### 7.2.13 `tosca.datatypes.nfv.L2AddressData`

#### 7.2.13.1 Description

The `L2AddressData` data type is defined in clause 6.2.1 of the present document.

### 7.2.14 `tosca.datatypes.nfv.L3AddressData`

#### 7.2.14.1 Description

The `L3AddressData` data type is defined in clause 6.2.2 of the present document.

## 7.2.15 `tosca.datatypes.nfv.Qos`

### 7.2.15.1 Description

The Qos data type is defined in clause 6.2.30 of the present document.

## 7.2.16 `tosca.datatypes.nfv.NsProfile`

### 7.2.16.1 Description

The NsProfile data type describes a profile for instantiating nested NSs which are constituents of a NS with a particular NS DF as defined in ETSI GS NFV-IFA 014 [2]. Table 7.2.16.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 7.2.16.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	NsProfile
<b>Type Qualified Name</b>	toscanfv:NsProfile
<b>Type URI</b>	tosca.datatypes.nfv.NsProfile

### 7.2.16.2 Properties

The properties of the NsProfile data type shall comply with the provisions set out in table 7.2.16.2-1.

**Table 7.2.16.2-1: Properties**

Name	Required	Type	Constraints	Description
<code>ns_instantiation_level</code>	no	string		Identifier of the instantiation level of the NS DF to be used for instantiation. If not present, the default instantiation level as declared in the NSD shall be used.
<code>min_number_of_instances</code>	yes	integer	<code>greater_or_equal : 0</code>	Minimum number of instances of the NS based on this NSD that is permitted to exist for this NsProfile.
<code>max_number_of_instances</code>	yes	integer	<code>greater_or_equal : 0</code>	Maximum number of instances of the NS based on this NSD that is permitted to exist for this NsProfile.
<code>flavour_id</code>	yes	string		Identifies the applicable network service DF within the scope of the NSD.

### 7.2.16.3 Definition

The syntax of the NsProfile data type shall comply with the following definition:

```
tosca.datatypes.nfv.NsProfile:
  derived_from: toasca.datatypes.Root
  description: describes a profile for instantiating NSs of a particular NS DF
  according to a specific NSD and NS DF.
  properties:
    ns_instantiation_level:
      type: string
      description: Identifier of the instantiation level of the NS DF to be used for
      instantiation. If not present, the default instantiation level as declared in the NSD
      shall be used.
      required: false
    min_number_of_instances:
      type: integer
      description: Minimum number of instances of the NS based on this NSD that is
      permitted to exist for this NsProfile.
```

```

    required: true
    constraints:
      - greater_or_equal: 0
  max_number_of_instances:
    type: integer
    description: Maximum number of instances of the NS based on this NSD that is
    permitted to exist for this NsProfile.
    required: true
    constraints:
      - greater_or_equal: 0
  flavour_id:
    type: string
    description: Identifies the applicable network service DF within the scope of
    the NSD.
    required: true

```

#### 7.2.16.4 Example

None.

#### 7.2.16.5 Additional Requirements

None.

### 7.3 Artifact Types

None.

### 7.4 Capabilities Types

#### 7.4.1 `tosca.capabilities.nfv.VirtualLinkable`

##### 7.4.1.1 Description

The `VirtualLinkable` capability type is defined in clause 6.4.2 of the present document.

### 7.5 Requirements Types

None.

### 7.6 Relationship Types

#### 7.6.1 `tosca.relationships.nfv.VirtualLinksTo`

##### 7.6.1.1 Description

The `VirtualLinksTo` relationship type is defined in clause 6.6.2 of the present document representing an association relationship between the VNF or PNF or Sap of a Nested NS and `NsVirtualLink` node types when used in a NSD.

## 7.7 Interface Types

### 7.7.1 `tosca.interfaces.nfv.Nslcm`

#### 7.7.1.1 Description

The `tosca.interfaces.nfv.Nslcm` interface type contains a set of TOSCA operations corresponding to the following NS LCM operations defined in ETSI GS NFV-IFA 013 [i.8]:

- Instantiate NS
- Scale NS
- Update NS
- Heal NS
- Terminate NS

The interface also contains TOSCA operations corresponding to preamble and postamble to the execution of the aforementioned base operations. The name of these operations is constructed according to the following pattern:

`<base_operation_name>_start` for a preamble

`<base_operation_name>_end` for a postamble

The designations ("`_start`", "`_end`") in the name of TOSCA operations are postfixes so that related operations are adjacent in an alphabetical listing.

Table 7.7.1.1-1 specifies the declared names for this interface type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 7.7.1.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	Nslcm
<b>Type Qualified Name</b>	toscanfv:Nslcm
<b>Type URI</b>	tosca.interfaces.nfv.Nslcm

#### 7.7.1.2 Definition

The syntax of the `Nslcm` interface type shall comply with the following definition:

```
tosca.interfaces.nfv.Nslcm:
  derived_from: toska.interfaces.Root
  description: This interface encompasses a set of TOSCA operations corresponding to
  NS LCM operations defined in ETSI GS NFV-IFA 013. as well as to preamble and postamble
  procedures to the execution of the NS LCM operations.
  instantiate_start:
    description: Preamble to execution of the instantiate operation
  instantiate:
    description: Base procedure for instantiating an NS, corresponding to the
  Instantiate NS operation defined in ETSI GS NFV-IFA 013.
  instantiate_end:
    description: Postamble to the execution of the instantiate operation
  terminate_start:
    description: Preamble to execution of the terminate operation
  terminate:
    description: Base procedure for terminating an NS, corresponding to the Terminate
  NS operation defined in ETSI GS NFV-IFA 013.
  terminate_end:
    description: Postamble to the execution of the terminate operation
```

```

update_start:
  description: Preamble to execution of the update operation
update:
  description: Base procedure for updating an NS, corresponding to the Update NS
operation defined in ETSI GS NFV-IFA 013.
update_end:
  description: Postamble to the execution of the update operation
scale_start:
  description: Preamble to execution of the scale operation
scale:
  description: Base procedure for scaling an NS, corresponding to the Scale NS
operation defined in ETSI GS NFV-IFA 013.
scale_end:
  description: Postamble to the execution of the scale operation
heal_start:
  description: Preamble to execution of the heal operation
heal:
  description: Base procedure for healing an NS, corresponding to the Heal NS
operation defined in ETSI GS NFV-IFA 013.
heal_end:
  description: Postamble to the execution of the heal operation

```

### 7.7.1.3 Additional Requirements

The implementation and inputs keynames specified in TOSCA-Simple-Profile-YAML-v1.2 [3] for an operation definition may be included for each operation listed in the Nslcm interface definition.

When a TOSCA operation representing an NS LCM operation does not have an associated implementation keyname, the default implementation provided by the NFVO for this NS LCM operation applies.

The NSD consumer shall make available all parameters from the message invoking the NS LCM operation as inputs to the corresponding TOSCA interface operations. The `inputs` keyname can be used to specify additional input parameters for executing the operation.

The implementation of preamble and postamble TOSCA operations (e.g. `instantiate_start`), if present, is invoked with the same parameters as the corresponding base TOSCA operation (e.g. `instantiate`). The inputs of the preamble and postamble operations shall not be defined in the NSD.

If an implementation is associated to a TOSCA operation that represents a preamble or a postamble to a VNF LCM operation, the implementation logic is executed before or after the execution of the VNF LCM operation implementation, respectively.

### 7.7.1.4 Support of LCM scripts

In ETSI GS NFV-IFA 014 [2], the definition of the "LifeCycleManagementScript" information element of the NSD associates LCM scripts with events, where an event can be an external or an internal stimulus. These events are mapped to TOSCA operations of the NS node type in the following way:

- external stimuli are mapped to TOSCA operations corresponding to the NS LCM operations defined in ETSI GS NFV-IFA 013 [i.8];
- internal stimuli are mapped to preamble and postamble of these TOSCA operations.

LCM scripts can be regarded as artifacts that provide an NS-specific implementation of the TOSCA operation corresponding to the stimulus.

The script input parameters shall be provided to the script according to the declaration in the `inputs` field of the operation definition. The artifact type definition shall enable identifying the DSL used by the script. The artifact type definition for Python is provided in section 5.4.4.1 of TOSCA-Simple-Profile-YAML-v1.2 [3]. The artifact definition for Mistral is provided in clause A.7.2 of the present document.

### 7.7.1.5 Examples

The following example template fragments illustrate the concept. An LCM script is associated with the `instantiate_end` operation. As no LCM script is associated to the `instantiate` operation, its default implementation runs and before running the `post-instantiate-script`. The inputs section of the `instantiate_end` operation definition provides additional input values to the `post-instantiate-script`, and the TOSCA artifacts definition conveys the type of DSL used as a scripting language.

```
tosca_definitions_version: toska_simple_yaml_1_2

imports:
- ..

node_types:
  MyCompany.SunshineVPN.1_0.1_0:
    derived_from: toska.nodes.nfv.NS
    ..

topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineVPN.1_0.1_0
    ..

node_templates:
  SunshineVPN:
    type: MyCompany.SunshineVPN.1_0.1_0
    ..
  interfaces:
    Nslcm:
      instantiate_end:
        implementation: post-instantiate-script
        inputs:
          script_input_1: value_1
          script_input_2: value_2

artifacts:
  post-instantiate-script:
    description: Instantiate workflow script
    type: toska.artifacts.Implementation.Python
    file: instantiate.py
    #repository: ..
    #deploy_path: ..
    ..
```

## 7.8 Node Types

### 7.8.1 toska.nodes.nfv.NS

#### 7.8.1.1 Description

The NFV Network Service (NS) node type describes a NS in terms of deployment, operational behaviour, and requirements, as defined by ETSI GS NFV-IFA 014 [2]. Table 7.8.1.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 7.8.1.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	NS
<b>Type Qualified Name</b>	tosca.nfv:NS
<b>Type URI</b>	tosca.nodes.nfv.NS

### 7.8.1.2 Properties

The properties of the NS node type shall comply with the provisions set out in table 7.8.1.2-1.

**Table 7.8.1.2-1: Properties**

Name	Required	Type	Constraints	Description
descriptor_id	yes	string		Identifier of this NS descriptor.
designer	yes	string		Identifies the designer of the NSD.
version	yes	string		Identifies the version of the NSD.
name	yes	string		Provides the human readable name of the NSD.
invariant_id	yes	string		Identifies an NSD in a version independent manner. This attribute is invariant across versions of NSD.
flavour_id	yes	string		Identifier of this NS DF within the NSD.
ns_profile	no	tosca.datatype.s.nfv.NsProfile		Specifies a profile of a NS, when this NS is used as nested NS within another NS. See note.
NOTE : This property is only used in an NS node template, when it is representing a nested NS within another NS.				

The value of the descriptor\_id string shall comply with an UUID format as specified in section 3 of [9].

### 7.8.1.3 Attributes

None.

### 7.8.1.4 Requirements

The requirements of the NS node type shall comply with the provisions set out in table 7.8.1.4-1.

**Table 7.8.1.4-1: Requirements**

Name	Required	Type	Constraints	Description
virtual_link	yes	tosca.nodes.nfv.NsVirtualLink		Describes the requirements for linking to virtual link

### 7.8.1.5 Capabilities

None.



### 7.8.1.6 Definition

The syntax of the NS node type shall comply with the following definition:

```

tosca.nodes.nfv.NS:
  derived_from: toasca.nodes.Root
  properties:
    descriptor_id:
      type: string # GUID
      description: Identifier of this NS descriptor
      required: true
    designer:
      type: string
      description: Identifies the designer of the NSD.
      required: true
    version:
      type: string
      description: Identifies the version of the NSD.
      required: true
    name:
      type: string
      description: Provides the human readable name of the NSD.
      required: true
    invariant_id:
      type: string
      description: Identifies an NSD in a version independent manner. This
attribute is invariant across versions of NSD
      required: true
    flavour_id:
      type: string
      description: Identifier of the NS Deployment Flavour within the NSD
      required: true
    ns_profile:
      type: toasca.datatypes.nfv.NsProfile
      description: Specifies a profile of a NS, when this NS is used as nested
NS within another NS.
      required: false
  requirements:
    - virtual_link:
      capability: toasca.capabilities.nfv.VirtualLinkable
      relationship: toasca.relationships.nfv.VirtualLinksTo
      node: toasca.nodes.nfv.NsVirtualLink
      occurrences: [ 0, 1 ]
  interfaces:
    Nslcm:
      type: toasca.interfaces.nfv.Nslcm

```

### 7.8.1.7 Artifact

None.

### 7.8.1.8 Additional requirements

For a given NSD, a new VNF node type shall be defined following the below requirements:

- a) The node type shall be derived from: toasca.nodes.nfv.NS.
- b) All properties listed in toasca.nodes.nfv.NS where the "required:" field is set to "true" shall be included with their values indicated as constraints.

- c) Properties listed in in `tosca.nodes.nfv.NS` where the "required:" field is set to "false" may be included.
- d) The capabilities, requirements, interfaces of `tosca.nodes.nfv.VNF` shall be preserved.
- e) Depending on the number of SAPs of the NS, additional requirements for VirtualLinkable capability shall be defined. In this case, it is the NSD author's choice to use the requirement for VirtualLinkable capability defined in the `tosca.nodes.nfv.NS` node type or use only the additional requirements defined in the derived NS specific node type. In the latter case, the `virtual_link` requirement should be included in the node type definition with occurrences [ 0, 0 ].

## 7.8.2 `tosca.nodes.nfv.Sap`

### 7.8.2.1 Description

The Service Access Point (SAP) node type describes a connection point where a NS can be accessed, as defined by ETSI GS NFV-IFA 014 [2]. Table 7.8.2.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 7.8.2.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	Sap
<b>Type Qualified Name</b>	toscanfv:Sap
<b>Type URI</b>	tosca.nodes.nfv.Sap

### 7.8.2.2 Properties

The properties applied to Sap node are derived from Cp node type as defined in clause 6.8.7 of the present document.

### 7.8.2.3 Attributes

None.

### 7.8.2.4 Requirements

The requirements of the Sap node type shall comply with the provisions set out in table 7.8.2.4-1.

**Table 7.8.2.4-1: Requirements**

Name	Required	Type	Constraints	Description
external_virtual_link	no	tosca.capabilities.nfv.VirtualLinkable		Specifies that CP instances require to be connected to a node that has a VirtualLinkable capability
internal_virtual_link	yes	tosca.capabilities.nfv.VirtualLinkable		Specifies that CP instances require to be connected to a node that has a VirtualLinkable capability

### 7.8.2.5 Capabilities

None.

### 7.8.2.6 Definition

The syntax of the Sap node type shall comply with the following definition:

```
tosca.nodes.nfv.Sap:
  derived_from: toasca.nodes.nfv.Cp
  description: node definition of SAP.
```

```

requirements:
- external_virtual_link:
  capability: tosca.capabilities.nfv.VirtualLinkable
  relationship: tosca.relationships.nfv.VirtualLinksTo
- internal_virtual_link:
  capability: tosca.capabilities.nfv.VirtualLinkable
  relationship: tosca.relationships.nfv.VirtualLinksTo

```

### 7.8.2.7 Additional requirements

A node template of this type is used to represent a SAP only in the case the Sap is connected to a NsVirtualLink inside a NSD. The node template has the following requirements:

- internal\_virtual\_link requirement to allow to connect it to a NsVirtualLink inside a NSD;
- external\_virtual\_link requirement to allow to connect it to a NsVirtualLink outside a NSD.

In the case where a Sap is exposed by a VNF external connection point, a PNF external connection point or a Sap of the nested NS, the Sap node type does not apply.

### 7.8.2.8 Example

In a typical scenario, the Sap node template will be part of a service template representing a certain NS deployment flavour. The service template substitutes for an abstract NS node type. In this substitution, the virtual\_link requirement of the abstract NS node type is mapped to the external\_virtual\_link requirement of the Sap node. This example is illustrated in clause A.7.3.

When a Sap re-exposes a VNF external connection point, the service template does not require an explicit node template of type Sap in a typical scenario where an abstract NS node type is substituted by a service template representing a certain NS deployment flavour. In this substitution, the virtual\_link requirement of the abstract NS node type is mapped to the virtual\_link requirement of the VNF node. This example is illustrated in clause A.7.2.

## 7.8.3 tosca.nodes.nfv.NsVirtualLink

### 7.8.3.1 Description

The NsVirtualLink node type represents the NsVirtualLinkDesc information element as defined in ETSI GS NFV-IFA 014 [2], which describes the requirements for a virtual link of a network service. Table 7.8.3.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 7.8.3.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	NsVirtualLink
<b>Type Qualified Name</b>	toscanfv: NsVirtualLink
<b>Type URI</b>	tosca.nodes.nfv.NsVirtualLink

### 7.8.3.2 Properties

The properties of the NsVirtualLink node type shall comply with the provisions set out in table 7.8.3.2-1.

**Table 7.8.3.2-1: Properties**

Name	Required	Type	Constraints	Description
vl_profile	yes	tosca.datatype.nfv.NsVIProfile		Specifies instantiation parameters for a virtual link of a particular NS deployment flavour.
connectivity_type	yes	tosca.datatypes.nfv.ConnectivityType		Network service virtual link connectivity type.
test_access	no	list of string	Valid values: passive_monitoring, active_loopback	Test access facilities available on the VL.
description	no	string		Human readable information on the purpose of the virtual link (e.g. VL for control plane traffic).

### 7.8.3.3 Attributes

None.

### 7.8.3.4 Requirements

None.

### 7.8.3.5 Capabilities

The capabilities of the NsVirtualLink node type shall comply with the provisions set out in table 7.8.3.5-1.

**Table 7.8.3.5-1: Capabilities**

Name	Type	Constraints	Description
virtual_linkable	tosca.capabilities.nfv.VirtualLinkable		VirtualLinkable capability

### 7.8.3.6 Definition

The syntax of the NsVirtualLink node type shall comply with the following definition:

```

tosca.nodes.nfv.NsVirtualLink:
  derived_from: toska.nodes.Root
  description: node definition of Virtual Links
  properties:
    type: toska.datatypes.nfv.NsVIProfile # only covers min/max bitrate
  requirements
    description: Specifies instantiation parameters for a virtual link of a
  particular NS deployment flavour.
    required: true
  connectivity_type:
    type: toska.datatypes.nfv.ConnectivityType
    required: true
  test_access:
    type: list
    description: Test access facilities available on the VL
    required: false
  entry_schema:
    type: string
    constraints:
      - valid_values: [ passive_monitoring, active_loopback ]
  description:
    type: string

```

```

    required: false
    description: Human readable information on the purpose of the virtual link
    (e.g. VL for control plane traffic).
    capabilities:
      virtual_linkable:
        type: toasca.capabilities.nfv.VirtualLinkable

```

### 7.8.3.7 Artifact

None.

### 7.8.3.8 Additional Requirements

None.

### 7.8.3.9 Example

None.

## 7.8.4 toasca.nodes.nfv.Cp

### 7.8.4.1 Description

The Cp node type is defined in clause 6.8.7 of the present document.

## 7.9 Group Types

### 7.9.1 toasca.groups.nfv.PlacementGroup

#### 7.9.1.1 Description

The PlacementGroup group type is defined in clause 6.9.1 of the present document describing the affinity or anti-affinity relationship applicable between VNF instances created using different VNFDs, the Virtual Link instances created using different VLDs or the nested NS instances created using different NSDs when used in a NSD.

## 7.10 Policy Types

### 7.10.1 AffinityRule, AntiAffinityRule

#### 7.10.1.1 Description

The AffinityRule and AntiAffinityRule policy types as defined in clause 6.10.10 when used in a NSD TOSCA service template specify the affinity or anti-affinity rules applicable:

- If there is only one node template with node type toasca.nodes.nfv.VNF or toasca.nodes.nfv.NsVirtualLink set as the targets, the AffinityRule or AntiAffinityRule applies between the instances to be created based on the same VNFD, or between VLs to be created based on the same VLD as described in ETSI GS NFV-IFA 014 [2].
- If there are more than one node templates with node type toasca.nodes.nfv.VNF or toasca.nodes.nfv.NsVirtualLink or toasca.nodes.nfv.NS set as the targets, or a group with type toasca.groups.nfv.PlacementGroup which contains more than one members set as targets, the AffinityRule or AntiAffinityRule applies between VNF instances created using different VNFD, the Virtual Link instances created using different VLD or the nested NS instances created using different NSD. as described in ETSI GS NFV-IFA 014 [2].

### 7.10.1.2 Targets

The targets of the AffinityRule and AntiAffinityRule policy types shall comply with the provisions set out in table 7.10.1.2-1 when used in a NSD.

**Table 7.10.1.2-1: Targets**

Name	Required	Type	Constraints	Description
targets	Yes	string[]	Valid types: tosca.nodes.nfv.VNF, tosca.nodes.nfv.NsVirtualLink, tosca.nodes.nfv.NS, tosca.groups.nfv.PlacementGroup	In case of LocalAffinityOrAntiAffinityRule as defined in ETSI GS NFV-IFA 014 [2], the valid type of the targets is tosca.nodes.nfv.VNF or tosca.nodes.nfv.NsVirtualLink. In case of affinityOrAntiAffinityGroup as defined in ETSI GS NFV-IFA 014 [2], the valid types of the targets are: tosca.nodes.nfv.VNF, tosca.nodes.nfv.NsVirtualLink and tosca.nodes.nfv.NS or a tosca.groups.nfv.PlacementGroup.

### 7.10.1.3 Examples

The following example template fragments illustrate the concepts:

```
node_templates:
  VNF_1:
    type: tosca.nodes.nfv.exampleVNF

policies:
  policy_affinity_local_VNF_1:
    type: tosca.policies.nfv.AffinityRule
    targets: [ VNF_1 ]
    properties:
      scope: nfvi_node
```

The above example illustrates a local affinity rule for all the instances of VNF\_1.

```
node_template:
  VNF_1:
    type: tosca.nodes.nfv.Vdu.exampleVNF_1

  VNF_2:
    type: tosca.nodes.nfv.Vdu.exampleVNF_2

groups:
  affinityOrAntiAffinityGroup_1:
    type: tosca.groups.nfv.PlacementGroup
    members: [ VNF_1, VNF_2 ]

policies:
  policy_antiaffinity_group_1:
    type: tosca.policies.nfv.AntiAffinityRule
    targets: [ affinityOrAntiAffinityGroup_1 ]
    properties:
      scope: nfvi_node
```

The above example illustrates an anti-affinity policy among a group which contains VNF\_1 and VNF\_2 as members.

## 7.10.2 `tosca.policies.nfv.SecurityGroupRule`

### 7.10.2.1 Description

The `SecurityGroupRule` policy type as defined in clause 6.10.11 of the present document when used in a NSD specifies the matching criteria for the ingress and/or egress traffic to and from visited SAPs.

### 7.10.2.2 Additional Requirements

The valid type of the targets of the `SecurityGroupRule` policy when used in a NSD shall set to `tosca.nodes.nfv.Sap`.

## 7.11 NSD TOSCA service template design

### 7.11.1 General

The TOSCA service template design for a NSD in the general case uses two levels of service templates as described in clause 7.11.2. In this design, the top level contains an abstract node template, i.e. without an implementation of the creation operation and is therefore substituted by one of the lower level service templates. This design is applicable regardless of whether the NS has one or multiple deployment flavours.

In the particular case of a NS with only one deployment flavour there is an alternative design which is described in clause 7.11.3 and which uses only one service template.

### 7.11.2 Single or multiple deployment flavour design with TOSCA-Simple-Profile-YAML-v1.2

NSD shall be implemented as one top-level service template and one or multiple lower level service templates, where each lower level service template represents a deployment flavour. In addition, an abstract node type representing the NS with multiple deployment flavours is also required. The top level service template shall be the main entry point of the NSD file structure as specified in ETSI GS NFV-SOL 007 [i.11], i.e. the Entry-definitions file. The node type definition is included in a separate YAML file imported from the top and from the lower level files.

See clause A.11 for an example of NSD design with multiple deployment flavours.

The top level service template shall comply with TOSCA-Simple-Profile-YAML-v1.2 [3] and shall include:

- a) an import statement referencing the TOSCA types definition file as defined in clause B.3;
- b) an import statement referencing the definitions of all low level service template(s);
- c) an import statement referencing a yaml file which contains a NS specific node type definition which shall be derived from `tosca.nodes.nfv.NS` node type as defined in clause 7.8.1;
- d) a topology template with a node template of the NS-specific node type, which shall include the `flavour_id` and other properties and the requirements as defined in clause 7.8.1; and
- e) Optionally, import statements referencing additional NS-specific files containing type definitions used by this TOSCA service template.

The deployment and lifecycle management of instances of this NS node type is done by means of substitution by any of the lower level service templates.

The lower level service template is an implementable TOSCA service template for the deployment of a specific deployment flavour.

The lower level service template shall comply with TOSCA-Simple-Profile-YAML-v1.2 [3] and shall include:

- a) an import statement referencing the TOSCA types definition file as defined in clause B.3;
- b) an import statement referencing a yaml file which contains a NS specific node type definition which shall be derived from `tosca.nodes.nfv.NS` node type as defined in clause 7.8.1;

- c) one of more import statements respectively referencing the yaml file which contains the included VNF specific node type definition if any or the included PNF specific node type definition if any or the included NS specific node type definition if any;
- d) Optionally, import statements referencing additional NS-specific files containing type definitions used by this TOSCA service template; and
- e) a topology template describing the internal topology of the NS with:
  - substitution\_mappings indicating:
    - the same node type as defined in the top level service template;
    - a flavour\_id property value which identifies the DF corresponding to this low level template within the NSD;
    - the mapping of the virtual\_link requirements on SAPs;
  - a node template referencing the NS specific node type, implementations of the operations of the LCM interface to be executed by the NFVO, if applicable; and
  - additional node templates of type VNF, PNF, NS, NsVirtualLink, Sap, etc. that define the topology and composition of the NS flavour, the dependency requirements as defined in TOSCA-Simple-Profile-YAML-v1.2 [3] may be used between different VNF node templates, or between a VNF node template and a nested NS node template, or between different nested NS node templates to specify the order in which instances of the VNFs and/or nested NSs have to be created.

The yaml file with the NS-specific node type definition shall be included in the NSD file structure.

NOTE: The format and structure of a NSD file structure is defined in ETSI GS NFV-SOL 007 [i.11].

When the flavour\_id of a NS has been chosen (e.g. through an input parameter of a NS instantiation request received by a NFVO) among the values included in the NS node type imported into the top level service template, it is then used as the property constraint for selecting a particular lower level TOSCA service template inside the NSD file structure by using abstract node template matching with substitution as described in TOSCA-Simple-Profile-YAML-v1.2 [3].

### 7.11.3 Single deployment flavour design with TOSCA-Simple-Profile-YAML-v1.1

In case of single deployment flavour scenario, TOSCA-Simple-Profile-YAML-v1.1 [4] may be used to describe the NSD with a single TOSCA service template.

NOTE 1: The present document does not preclude this pattern using TOSCA-Simple-Profile-YAML-v1.2 [3].

In this case one single service template is used as specified in TOSCA-Simple-Profile-YAML-v1.1 [4] and includes:

- a) an import statement referencing the TOSCA types definition file as defined in clause B.3;
- b) one of more import statements respectively referencing the yaml file which contains the included VNF specific node type definition if any or the included PNF specific node type definition if any or the included NS specific node type definition if any;
- c) optionally, import statements referencing additional NS-specific files containing type definitions used by this TOSCA service template;
- d) an abstract NS node type definition derived from the tosca.nodes.nfv.NS node type, as defined in clause 7.8.1; and
- e) a topology template describing the internal topology of the NS with:
  - substitution\_mappings indicating the same abstract NS node type and the mapping of the virtual\_link requirements on SAPs;
  - a node template of this abstract NS node type with the flavour\_id and other properties and, if applicable, implementations of the operations of the LCM interface to be executed by the NFVO; and



- additional node templates of type VNF, PNF, NS, NsVirtualLink, Sap, etc. that define the topology and composition of the NS flavour, the dependency requirements as defined in TOSCA-Simple-Profile-YAML-v1.2 [3] may be used between different VNF node templates, or between a VNF node template and a nested NS node template, or between different nested NS node templates to specify the order in which instances of the VNFs and/or nested NSs have to be created.

See clause A.8 for an example of NSD design with single deployment flavour.

NOTE 2: The service template is deployed stand-alone, i.e. without performing a substitution. However including the substitution\_mappings rule indicate its ability to substitute a node template of the abstract NS node type, which may appear in an NSD.

## 8 PNF TOSCA model

### 8.1 Introduction

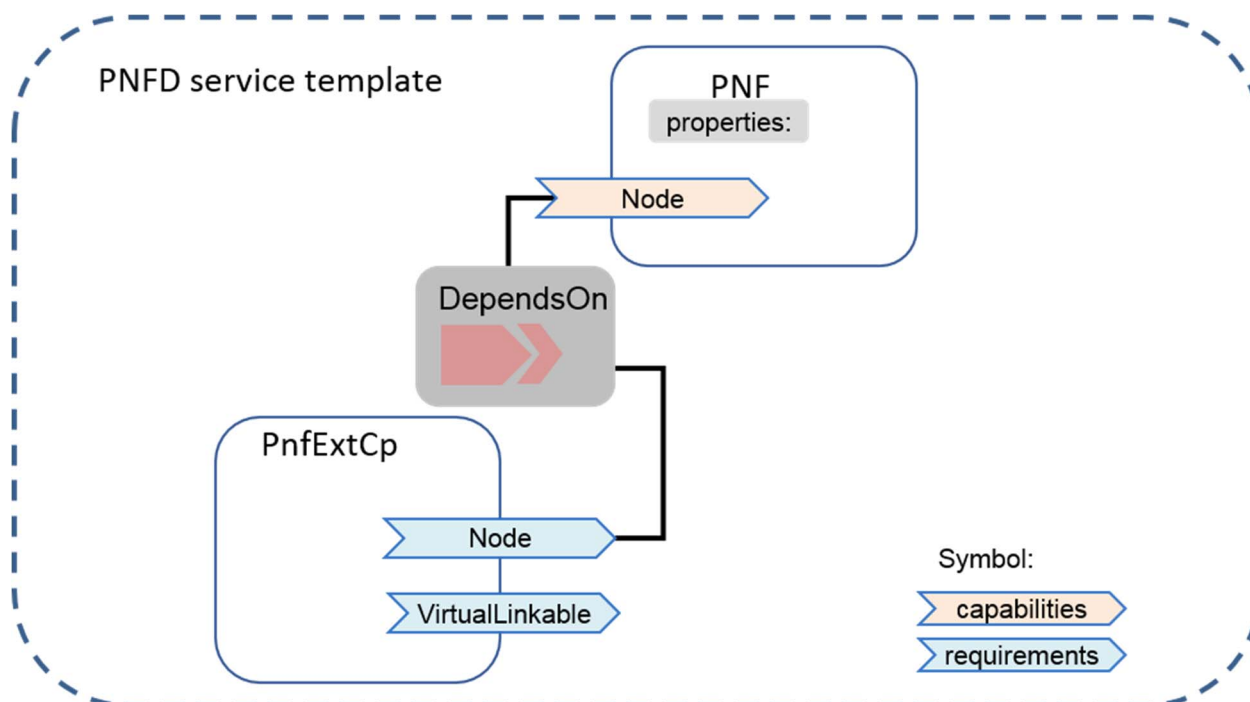
The PNF information model specified by ETSI GS NFV-IFA 014 [2] is mapped to the TOSCA concepts. It represents as TOSCA topology template to be used by NFVO for preparing network connection.

Table 8.1-1 shows the TOSCA Type "derived from" values used when applying the TOSCA Simple YAML Profile [3] to the PNF.

**Table 8.1-1: Mapping of ETSI GS NFV-IFA 014 [2] information elements with TOSCA types**

ETSI NFV Information Element ETSI GS NFV-IFA 014 [2]	TOSCA type	Derived from
Pnf	tosca.nodes.nfv.PNF	tosca.nodes.Root
PnfExtCpd (PNF External Connection Point)	tosca.nodes.nfv.PnfExtCp	tosca.nodes.Root

Figure 8.1-1 provides an overview of the TOSCA node types used to build a service template for a PNF.



**Figure 8.1-1: Service template PNF overview**

## 8.2 Data Types

### 8.2.1 `tosca.datatypes.nfv.CpProtocolData`

#### 8.2.1.1 Description

The `CpProtocolData` data type is defined in clause 6.2.36 of the present document.

### 8.2.2 `tosca.datatypes.nfv.AddressData`

#### 8.2.2.1 Description

The `AddressData` data type is defined in clause 6.2.3 of the present document.

### 8.2.3 `tosca.datatypes.nfv.L2AddressData`

#### 8.2.3.1 Description

The `L2AddressData` data type is defined in clause 6.2.1 of the present document.

### 8.2.4 `tosca.datatypes.nfv.L3AddressData`

#### 8.2.4.1 Description

The `L3AddressData` data type is defined in clause 6.2.2 of the present document.

### 8.2.5 `tosca.datatypes.nfv.LocationInfo`

#### 8.2.5.1 Description

The `LocationInfo` data type represents geographical information on the location where a PNF is deployed as specified in ETSI GS NFV-IFA 011 [1]. Table 8.2.5.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 8.2.5.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>LocationInfo</code>
<b>Type Qualified Name</b>	<code>toscanfv:LocationInfo</code>
<b>Type URI</b>	<code>tosca.datatypes.nfv. LocationInfo</code>

#### 8.2.5.2 Properties

The properties of the `LocationInfo` data type shall comply with the provisions set out in table 8.2.5.2-1.

**Table 8.2.5.2-1: Properties**

Name	Required	Type	Constraints	Description
<code>country_code</code>	yes	string		Shall be a two-letter ISO 3166 [10] country code in capital letters.
<code>civic_address_element</code>	no	list of <code>tosca.datatypes.nfv.CivicAddressElement</code>		Elements composing the civic address where the PNF is deployed.

### 8.2.5.3 Definition

The syntax of the LocationInfo data type shall comply with the following definition:

```

tosca.datatypes.nfv.LocationInfo:
  derived_from: tosca.datatypes.Root
  description: Represents geographical information on the location where a PNF is
  deployed.
  properties:
    country_code:
      type: string #two-letter ISO 3166 country code
      description: Country code
      required: true
    civic_address_element:
      type: list
      entry_schema
      type: tosca.datatypes.nfv.CivicAddressElement
      description: Elements composing the civic address where the PNF is deployed.
      required: false

```

### 8.2.5.4 Examples

```

<some_tosca_entity>:
  properties:
    geographical_location_info:
      country_code: FR
      civic_address_element:
        - element_1
          ca_type: 3
          ca_value: Paris

```

### 8.2.5.5 Additional Requirements

None.

## 8.2.6 tosca.datatypes.nfv.CivicAddressElement

### 8.2.6.1 Description

The CivicAddressElement data type represents an element of a civic location as specified in IETF RFC 4776 [11]. Table 8.2.6.1-1 specifies the declared names for this data type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 8.2.6.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	CivicAddressElement
<b>Type Qualified Name</b>	toscanfv: CivicAddressElement
<b>Type URI</b>	tosca.datatypes.nfv. CivicAddressElement

### 8.2.6.2 Properties

The properties of the CivicAddressElement data type shall comply with the provisions set out in table 8.2.6.2-1.

Table 8.2.6.2-1

Name	Required	Type	Constraints	Description
ca_type	yes	string		Describe the content type of caValue. The value of caType shall comply with section 3.4 of IETF RFC 4776 [11].
ca_value	yes	string		Content of civic address element corresponding to the caType. The format caValue shall comply with section 3.4 of IETF RFC 4776 [11].

### 8.2.6.3 Definition

The syntax of the CivicAddressElement data type shall comply with the following definition:

```

tosca.datatypes.nfv.CivicAddressElement:
  derived_from: toska.datatypes.Root
  description: Represents an element of a civic location as specified in IETF
RFC 4776 [11].
  properties:
    ca_type:
      type: string # RFC4776
      description: caType as per RFC4776
      required: true
    ca_value:
      type: string # RFC4776
      description: caValue as per RFC4776.
      required: true

```

### 8.2.6.4 Examples

See clause 8.2.5.4.

### 8.2.6.5 Additional Requirements

None.

## 8.3 Artifact Types

None.

## 8.4 Capabilities Types

### 8.4.1 toska.capabilities.nfv.VirtualLinkable

#### 8.4.1.1 Description

The VirtualLinkable capability type is defined in clause 6.4.2 of the present document.

## 8.5 Requirements Types

None.

## 8.6 Relationship Types

### 8.6.1 `tosca.relationships.nfv.VirtualLinksTo`

#### 8.6.1.1 Description

The `VirtualLinksTo` relationship type is defined in clause 6.6.2 of the present document representing an association relationship between a PNF external connection point and an `NsVirtualLink` node type.

## 8.7 Interface Types

None.

## 8.8 Node Types

### 8.8.1 `tosca.nodes.nfv.PNF`

#### 8.8.1.1 Description

The Physical Network Function (PNF) node type describes a PNF in terms of deployment behaviour requirements, which it contains PNFD identifier, version and functional description and so on as defined by ETSI GS NFV-IFA 014 [2]. Table 8.8.1.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 8.8.1.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	PNF
<b>Type Qualified Name</b>	<code>tosca:PNF</code>
<b>Type URI</b>	<code>tosca.nodes.nfv.PNF</code>

#### 8.8.1.2 Properties

The properties of the PNF node type shall comply with the provisions set out in table 8.8.1.2-1.

**Table 8.8.1.2-1: Properties**

Name	Required	Type	Constraints	Description
<code>descriptor_id</code>	yes	string		Identifier of this PNFD information element. It uniquely identifies the PNFD.
<code>function_description</code>	yes	string		Describes the PNF function.
<code>provider</code>	yes	string		Identifies the provider of the PNFD.
<code>version</code>	yes	string		Identifies the version of the PNFD.
<code>descriptor_invariant_id</code>	yes	string		Identifier of this PNFD in a version independent manner. This attribute is invariant across versions of PNFD.
<code>name</code>	yes	string		Name to identify the PNFD.
<code>geographical_location_info</code>	no	<code>tosca.datatype.nfv.LocationInfo</code>		Provides information about the geographical location (e.g. geographic coordinates or address of the building, etc.) of the PNF.

The value of the `descriptor_id` string and the value of the `descriptor_invariant_id` string shall comply with a UUID format as specified in section 3 of [8].

### 8.8.1.3 Attributes

None.

### 8.8.1.4 Requirements

The requirements of the VNF node type shall comply with the provisions set out in table 8.8.1.4-1.

**Table 8.8.1.4-1: Requirements**

Name	Required	Type	Constraints	Description
virtual_link	yes	tosca.nodes.nfv.Vnf VirtualLink		Describes the requirements for linking to virtual link

### 8.8.1.5 Capabilities

None.

### 8.8.1.6 Definition

```

tosca.nodes.nfv.PNF:
  derived_from: tosca.nodes.Root
  properties:
    descriptor_id: # instead of pnf_id
      type: string # GUID
      required: true
    function_description:
      type: string
      required: true
    provider:
      type: string
      required: true
    version:
      type: string
      required: true
    descriptor_invariant_id: # instead of pnf_id-invariant-id
      type: string # GUID
      required: true
    name:
      type: string
      required: true
    geographical_location_info:
      type: tosca.datatypes.nfv.LocationInfo
      required: false
    requirements:
      - virtual_link:
          capability: tosca.capabilities.nfv.VirtualLinkable
          relationship: tosca.relationships.nfv.VirtualLinksTo
          occurrences: [ 0, 1 ]
  # Additional requirements shall be defined in the PNF specific node type
  # (deriving from tosca.nodes.nfv.PNF) corresponding to NS virtual links that need to
  # connect to PnfExtCps

```

### 8.8.1.7 Artifact

None.

### 8.8.1.8 Additional Requirements

For a given PNFD, a new PNF node type shall be defined following the below requirements:

- a) The node type shall be derived from: `tosca.nodes.nfv.PNF`.
- b) All properties listed in `tosca.nodes.nfv.PNF` where the "required:" field is set to "true" shall be included with their values indicates as constraints.
- c) The requirements of `tosca.nodes.nfv.PNF` shall be preserved.
- d) Depending on the number of external connection points of the PNF that need to connect to NS virtual links, additional requirements for `VirtualLinkable` capability shall be defined. In this case, it is the PNFD author's choice to use the requirement for `VirtualLinkable` capability defined in the `tosca.nodes.nfv.PNF` node type or use only the additional requirements defined in the derived PNF specific node type. In the latter case, the `virtual_link` requirement should be included in the node type definition with occurrences [ 0, 0 ].

### 8.8.1.9 Example

See clause A.10.

## 8.8.2 `tosca.nodes.nfv.PnfExtCp`

### 8.8.2.1 Description

The `PnfExtCp` node type describes the characteristics of an external interface, a.k.a. an external CP, where to connect the PNF to a VL, as defined by ETSI GS NFV-IFA 014 [2]. Table 8.8.2.1-1 specifies the declared names for this node type. These names shall be used as specified in TOSCA-Simple-Profile-YAML-v1.2 [3].

**Table 8.8.2.1-1: Type name, shorthand, and URI**

<b>Shorthand Name</b>	<code>PnfExtCp</code>
<b>Type Qualified Name</b>	<code>toscanfv:PnfExtCp</code>
<b>Type URI</b>	<code>tosca.nodes.nfv.PnfExtCp</code>

### 8.8.2.2 Properties

The properties applied to `PnfExtCp` node are derived from `Cp` node type.

### 8.8.2.3 Attributes

None.

### 8.8.2.4 Requirements

The requirements of the `PnfExtCp` node type shall comply with the provisions set out in table 8.8.2.4-1.

**Table 8.8.2.4-1: Requirements**

Name	Required	Type	Constraints	Description
<code>external_virtual_link</code>	no	<code>tosca.capabilities.nfv.VirtualLinkable</code>		Specifies that CP instances require to be connected to a node that has a <code>VirtualLinkable</code> capability

### 8.8.2.5 Capabilities

None.

### 8.8.2.6 Definition

The syntax of the PnfExtCp node type shall comply with the following definition:

```
tosca.nodes.nfv.PnfExtCp:
  derived_from: toska.nodes.nfv.Cp
  description: node definition of PnfExtCp.
  requirements:
    - external_virtual_link:
      capability: toska.capabilities.nfv.VirtualLinkable
      relationship: toska.relationships.nfv.VirtualLinksTo
```

## 8.8.3 toska.nodes.nfv.Cp

### 8.8.3.1 Description

The Cp node type is defined in clause 6.8.7 of the present document.

## 8.9 Group Types

None.

## 8.10 Policy Types

### 8.10.1 toska.policies.nfv.SecurityGroupRule

#### 8.10.1.1 Description

The SecurityGroupRule policy type as defined in clause 6.10.11 of the present document when used in a PNFD specifies the matching criteria for the ingress and/or egress traffic to and from visited PNF external connection points.

#### 8.10.1.2 Additional Requirements

The valid type of the targets of the SecurityGroupRule policy when used in a PNFD shall set to toska.nodes.nfv.PnfExtCp.

## 8.11 PNFD TOSCA service template design

### 8.11.1 General

One single TOSCA service template is used to design a PNFD which shall comply with TOSCA-Simple-Profile-YAML-v1.2 [3] and includes:

NOTE 1: The present document does not preclude this pattern using TOSCA-Simple-Profile-YAML-v1.1 [4].

- a) an import statement referencing the TOSCA types definition file as defined in clause B.4;
- b) an abstract PNF node type definition derived from the toska.nodes.nfv.PNF node type, as defined in clause 8.8.1; and
- c) a topology template describing the internal topology of the PNF with:
  - substitution\_mappings indicating the same abstract PNF node type and the mapping of the virtual\_link requirements on PNF external connection point;



- a node template of this abstract PNF node type with the properties as defined in `tosca.nodes.nfv.PNF`;  
and
- additional node templates of type `PnfExtCp` that define the connection information of the PNF.

See clause A.10 for an example of PNFD design.

NOTE 2: The service template is deployed stand-alone, i.e. without performing a substitution. However including the `substitution_mappings` rule indicate its ability to substitute a node template of the abstract PNF node type, which may appear in an NSD.

## Annex A (informative): Examples

### A.1 Deployment flavour design mapping

#### A.1.1 Introduction

This annex describes the main design principle for VNF deployment flavour and the mapping between VNF deployment flavour elements to TOSCA types.

#### A.1.2 Design principle

Each Deployment flavour as specified in ETSI GS NFV-IFA 011 [1] describes a given deployment configuration of a VNF in terms of its internal topology and resource needs. Different deployment flavours can define different topologies of the same VNF, with different scalingAspect, different VDUs and different internal connectivity requirements. The idea of VNF deployment flavour as specified in [1] is that each deployment flavour describes the required vduProfiles and virtualLinkProfiles which are the additional instantiation data for the given VDUs and virtualLinks, once a specific deployment flavour has been chosen at the instantiation time, in order to successfully deploy the given VDUs and virtualLinks, the bindings between the VDU with the corresponding VduProfile and the virtualLinkDesc with the corresponding virtualLinkProfile are required.

To achieve the concept of deployment flavour by using TOSCA, the main design principle is to describe each deployment flavour as a standalone implementable TOSCA service template, and binding VDU and virtualLink with the corresponding VduProfile and virtualLinkProfile, respectively, together at the design time. Once a specific deployment flavour has been chosen at the instantiation time, the corresponding TOSCA service template will be used for deploying the VNF with the given deployment flavour.

Figure A.1.2-1 shows the general principle for a VNF deployment flavour design.

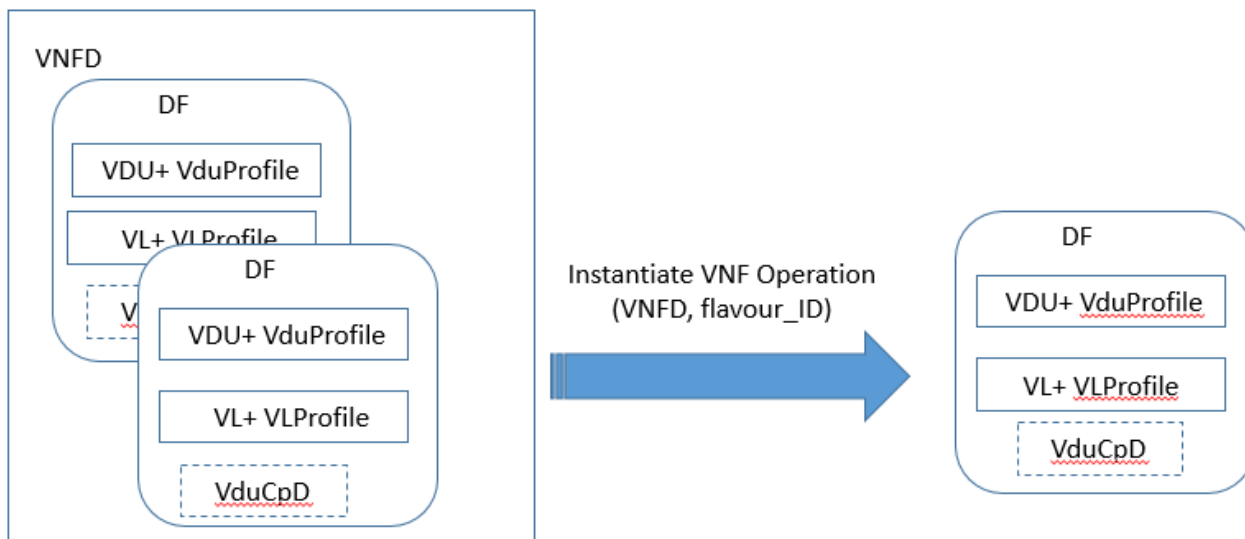


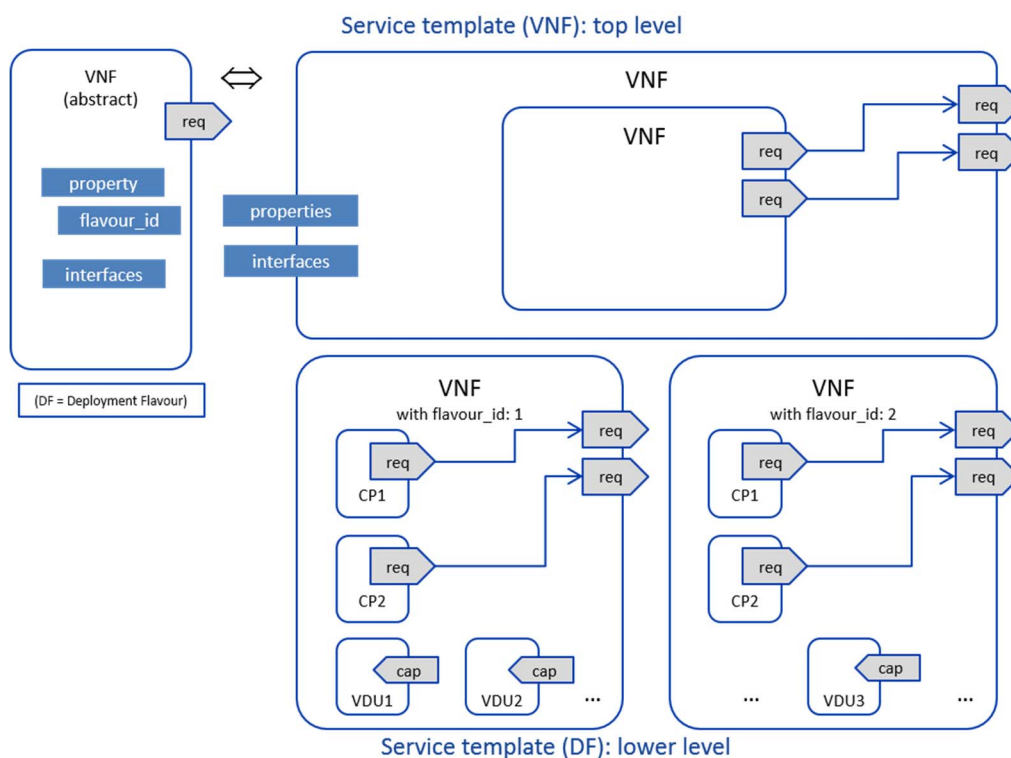
Figure A.1.2-1: General principle for a VNF deployment flavour design

## A.2 VNFD with deployment flavour modelling design example

Deployment flavours are represented as deployable TOSCA topology templates. This way one VNF service template represents one deployment flavour, and different deployment flavours are described by different VNF service templates. This is in line with the idea that different deployment flavours can define different topologies of the same VNF, with different scaling aspects, different VDUs and different internal connectivity.

In order to represent a VNF a top-level service template is used. This top-level service template contains a topology template with only an abstract VNF node which defines the common parts of the different deployment flavours (such as product information, modifiable attributes and parts of the lifecycle management interface definition). It also sets a constraint on the deployment flavour property (the required value of the flavour\_id property); this constraint comes from the VNF instantiation request which contains a flavour\_id selected among those available in the VNFD.

As a result, the VNFM will look into the available further service templates representing the different VNF deployment flavours of the VNF and use the one that has the matching flavour\_id property value to substitute for the abstract VNF. These are the low-level service templates.



**Figure A.2-1: VNFD overview with deployment flavour**

A VNFD contains a TOSCA top-level Service Template as entry point in the VNF package and one or more TOSCA low-level Service templates representing the different deployment flavours (see figure A.2-1). The VNFD is interpreted by an NFVO or VNF manager. In this example, the templates describe two variants of the VNF each corresponding to a deployment flavour: a simple one and a complex one. The simple VNF consists of one server: a DB backend whereas, the complex VNF variant consists of minimum three DB backend servers and one serviceNode, which may be scaled out in one-size increments.

SunshineDB: VNFD-top level

### **sunshine.vnfd.tosca.yaml**

```
tosca_definitions_version: tosca_simple_yaml_1_2

description: Relational database, non-scalable
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in NFV SOL
001 GS
  - sunshineVNF.yaml # contains the VNF node type definition
  - sunshinedbsimple.vnfd.tosca.yaml
  - sunshinedbcomplex.vnfd.tosca.yaml

topology_template:
  inputs:
    selected_flavour:
      type: string
      description: VNF deployment flavour selected by the consumer. It is provided in
the API

  node_templates:
    SunshineDB:
      type: MyCompany.SunshineDB.1_0.1_0
      properties:
        flavour_id: { get_input: selected_flavour }
        descriptor_id: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
        provider: MyCompany
        product_name: SunshineDB
        software_version: '1.0'
        descriptor_version: '1.0'
        vnfm_info:
          - '0:MyCompany-1.0.0'
      requirements:
        #- virtual_link_backend # mapped in lower-level templates
        #- virtual_link_service # mapped in lower-level templates
        # get_input function would be used by a TOSCA orchestrator at run time to
access the selected flavour id. If the deployment is not done by a TOSCA orchestrator
above function may not be needed
```

The sunshineVNF.yaml file has the following content:

```
tosca_definitions_version: tosca_simple_yaml_1_2

description: Relational database, non-scalable

imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in NFV SOL
001 GS

data_types:
  MyCompany.datatypes.nfv.VnfInstantiateAdditionalParameters:
    derived_from: tosca.datatypes.nfv.VnfOperationAdditionalParameters
    properties:
      parameter_1:
        type: string
        required: true
        default: value_1
      parameter_2:
        type: string
```

```

    required: false
    default: value_2
node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      provider:
        type: string
        constraints: equal: MyCompany ]
        default: MyCompany
      product_name:
        type: string
        constraints: [ equal: SunshineDB ]
        default: SunshineDB
      software_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      descriptor_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      flavour_id:
        type: string
        constraints: [ valid_values: [ simple, complex ] ]
        default: simple
      flavour_description:
        type: string
        default: "" #empty string
      vnfm_info:
        type: list
        entry_schema:
          type: string
          constraints: [ equal: '0:MyCompany-1.0.0' ]
          default: [ '0:MyCompany-1.0.0' ]
    requirements:
      - virtual_link:
          capability: tosca.capabilities.nfv.VirtualLinkable
          occurrences: [ 0, 0 ]
      - virtual_link_backend:
          capability: tosca.capabilities.nfv.VirtualLinkable
      - virtual_link_service:
          capability: tosca.capabilities.nfv.VirtualLinkable
    interfaces:
      Vnflcm:
        type: tosca.interfaces.nfv.Vnflcm
        instantiate:
          inputs:
            additional_parameters:
              type: MyCompany.datatypes.nfv.VnfInstantiateAdditionalParameters

```

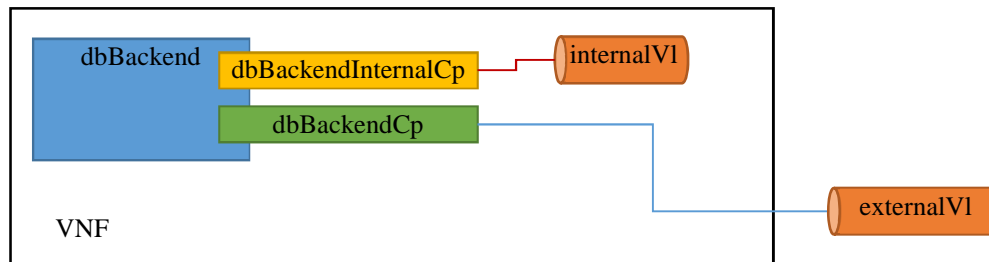
The vnf node template in the sunshine.vnfd.tosca.yaml file is abstract and is subject to substitution; the lower-level templates in the subsequent sections provide these substitutions. The actual lower-level template is selected based on the node type and a value constraint on the flavour\_id property.

Each low level service template contains a node template of type MyCompany.SunshineDB.1\_0.1\_0 with implementation of the LCM interfaces.

SunshineDB (simple): Lower level

**sunshinedbsimple.vnfd.tosca.yaml**

This example illustrates one Vdu.Compute nodes (dbBackend) with two connection points and two virtual links (see figure A.2-2). The flavour\_id is "Simple".



**Figure A.2-2: SunshineDB (simple): Lower level**

```

tosca_definitions_version: toska_simple_yaml_1_2

description: Relational database, simple
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in NFV SOL
    001 GS
  - sunshineVNF.yaml # contains the VNF node type definition

topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineDB.1_0.1_0
  properties:
    ..
    flavour_id: simple
  requirements:
    virtual_link_backend: [ dbBackendCp, virtual_link ] # IPv4 for SQL

node_templates:
  SunshineDB:
    type: MyCompany.SunshineDB.1_0.1_0
    properties:
      flavour_description: A simple flavour
    interfaces:
      Vnflcm:
        instantiate:
          implementation: instantiate.workbook.mistral.yaml
        terminate:
          implementation: terminate.workbook.mistral.yaml

  dbBackend:
    type: toska.nodes.nfv.Vdu.Compute
    properties:
      name: dbBackend
      description: dbBackend compute node
      boot_order: []
      nfvi_constraints: []
      configurable_properties: {}

```

```

vdu_profile:
  min_number_of_instances: 1
  max_number_of_instances: 1
capabilities:
  virtual_compute:
    properties:
      virtual_memory:
        virtual_mem_size: 8192 MiB
      virtual_cpu:
        cpu_architecture: x86
        num_virtual_cpu: 2
        virtual_cpu_clock: 1800 MHz
requirements:
  - virtual_storage: mariaDbStorage

mariaDbStorage:
  type: toasca.nodes.nfv.Vdu.VirtualBlockStorage
  properties:
    virtual_block_storage_data:
      size_of_storage: 100 GB
      rdma_enabled: true
    sw_image_data:
      name: Software of Maria Db
      version: '1.0'
      checksum:
        algorithm: sha-256
        hash: b9c3036539fd7a5f87a1bf38eb05fdde8b556a1a7e664dbeda90ed3cd74b4f9d
      container_format: bare
      disk_format: qcow2
      min_disk: 2 GB
      min_ram: 8192 MiB
      size: 2 GB
      operating_system: Linux
      supported_virtualisation_environments:
        - KVM
    ..
artifacts:
  sw_image:
    type: toasca.artifacts.nfv.SwImage
    file: maria.db.image.v1.0.qcow2

dbBackendCp:
  type: toasca.nodes.nfv.VduCp
  properties:
    layer_protocols: [ ipv4 ]
    role: leaf
    description: External connection point to access the DB on IPv4
    protocol: [ associated_layer_protocol: ipv4 ]
    trunk_mode: false
  requirements:
    - virtual_binding: dbBackend
    # - virtual_link: # the target node is determined in the NSD

dbBackendInternalCp:
  type: toasca.nodes.nfv.VduCp
  properties:
    layer_protocols: [ ipv4 ]
    role: leaf
    description: Internal connection point on an VL
    protocol: [ associated_layer_protocol: ipv4 ]
    trunk_mode: false

```

```

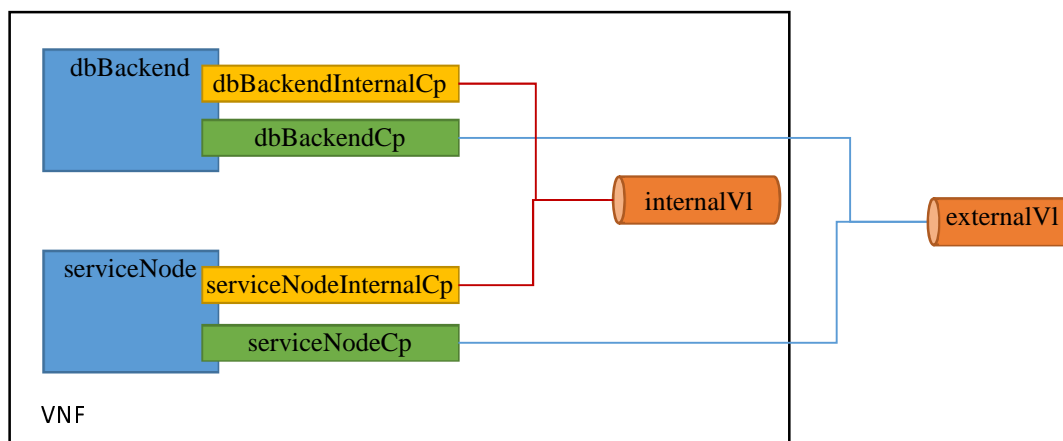
requirements:
  - virtual_binding: dbBackend
  - virtual_link: internalVl

internalVl:
  type: toasca.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ipv4 ]
      flow_pattern: mesh
    description: Internal Virtual link in the VNF
    vl_profile:
      max_bitrate_requirements:
        root: 100000
        leaf: 20000
      min_bitrate_requirements:
        root: 10000
        leaf: 10000

```

### SunshineDB (complex): Lower level

This example illustrates two Vdu.Compute nodes (dbBackend, and serviceNode) with two connection points and two virtual links (see figure A.2-3). The flavour\_id is "complex".



NOTE: The single external VL above illustrates that both serviceNodeCp and dbBackendCp are connected to the same external VL. Alternatively, external connection points can be connected to separate external VLs.

**Figure A.2-3: SunshineDB (complex): lower level**

#### sunshinedbcomplex.vnfd.tosca.yaml

```

tosca_definitions_version: toasca_simple_yaml_1_2

description: Relational database, complex
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in NFV SOL
    001 GS
  - sunshineVNF.yaml # contains the VNF node type definition

topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineDB.1_0.1_0
  properties:
    ..

```



```

flavour_id: complex
requirements:
  virtual_link_backend: [ dbBackendCp, virtual_link ] # IPv4 for SQL
  virtual_link_service: [ serviceNodeCp, virtual_link ] # IPv4 for SSH

node_templates:
  SunshineDB:
    type: MyCompany.SunshineDB.1_0.1_0
    properties:
      flavour_description: A complex flavour
    interfaces:
      Vnflcm:
        instantiate:
          implementation: instantiate.workbook.mistral.yaml
        terminate:
          implementation: terminate.workbook.mistral.yaml
        heal:
          implementation: heal.workbook.mistral.yaml

  dbBackend:
    type: toasca.nodes.nfv.Vdu.Compute
    properties:
      name: dbBackend
      description: dbBackend compute node
      boot_order: []
      nfvi_constraints: []
      configurable_properties: {}
      vdu_profile:
        min_number_of_instances: 3
        max_number_of_instances: 4
    capabilities:
      virtual_compute:
        properties:
          virtual_memory:
            virtual_mem_size: 8192 MiB
          virtual_cpu:
            cpu_architecture: x86
            num_virtual_cpu: 2
            virtual_cpu_clock: 1800 MHz
    requirements:
      - virtual_storage: mariaDbStorage

  serviceNode:
    type: toasca.nodes.nfv.Vdu.Compute
    properties:
      name: serviceNode
      description: brief description about serviceNode
      boot_order: []
      nfvi_constraints: []
      configurable_properties: {}
      vdu_profile:
        min_number_of_instances: 1
        max_number_of_instances: 1
    sw_image_data:
      name: Software of Maria Db
      version: '1.0'
      checksum:
        algorithm: sha-256
        hash: b9c3036539fd7a5f87a1bf38eb05fdde8b556a1a7e664dbeda90ed3cd74b4f9d
      container_format: bare
      disk_format: qcow2

```

```

    min_disk: 2 GB
    min_ram: 8192 MiB
    size: 2 GB
    operating_system: Linux
    supported_virtualisation_environments:
      - KVM
capabilities:
  virtual_compute:
    properties:
      virtual_memory:
        virtual_mem_size: 8192 MiB
      virtual_cpu:
        cpu_architecture: x86
        num_virtual_cpu: 2
        virtual_cpu_clock: 1800 MHz
requirements:
  - virtual_storage: mariaDbStorage
artifacts:
  sw_image:
    type: toska.artifacts.nfv.SwImage
    file: maria.db.image.v1.0.qcow2

mariaDbStorage:
  type: toska.nodes.nfv.Vdu.VirtualBlockStorage
  properties:
    virtual_block_storage_data:
      size_of_storage: 100 GB
      rdma_enabled: true
    sw_image_data:
      name: Software of Maria Db
      version: '1.0'
      checksum:
        algorithm: sha-256
        hash: b9c3036539fd7a5f87a1bf38eb05fdde8b556a1a7e664dbeda90ed3cd74b4f9d
      container_format: bare
      disk_format: qcow2
      min_disk: 2 GB
      min_ram: 8192 MiB
      size: 2 GB
      operating_system: Linux
      supported_virtualisation_environments:
        - KVM
  artifacts:
    sw_image:
      type: toska.artifacts.nfv.SwImage
      file: maria.db.image.v1.0.qcow2

dbBackendCp:
  type: toska.nodes.nfv.VduCp
  properties:
    layer_protocols: [ ipv4 ]
    role: leaf
    description: External connection point to access the DB on IPv4
    protocol: [ associated_layer_protocol: ipv4 ]
    trunk_mode: false
  requirements:
    - virtual_binding: dbBackend
    #- virtual_link: # the target node is determined in the NSD

dbBackendInternalCp:
  type: toska.nodes.nfv.VduCp

```

```

properties:
  layer_protocols: [ ipv4 ]
  role: leaf
  description: Internal connection point on an VL
  protocol: [ associated_layer_protocol: ipv4 ]
  trunk_mode: false
requirements:
  - virtual_binding: dbBackend
  - virtual_link: internalVl

serviceNodeCp:
  type: toasca.nodes.nfv.VduCp
  properties:
    layer_protocols: [ ipv4 ]
    role: leaf
    description: External connection point to access the DB on IPv4
    protocol: [ associated_layer_protocol: ipv4 ]
    trunk_mode: false
  requirements:
    - virtual_binding: serviceNode
    #- virtual_link: # the target node is determined in the NSD

serviceNodeInternalCp:
  type: toasca.nodes.nfv.VduCp
  properties:
    layer_protocols: [ ipv4 ]
    role: leaf
    description: Internal connection point on VL
    protocol: [ associated_layer_protocol: ipv4 ]
    trunk_mode: false
  requirements:
    - virtual_binding: serviceNode
    - virtual_link: internalVl

internalVl:
  type: toasca.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ipv4 ]
      flow_pattern: mesh
    description: Internal VL
    vl_profile:
      max_bitrate_requirements:
        root: 100000
        leaf: 20000
      min_bitrate_requirements:
        root: 10000
        leaf: 10000

```

---

## A.3 VNF external connection point

### A.3.1 General

A VNF external connection point may either be an internal connection point (a VDU connection point) which is re-exposed externally, i.e. it may be connected to an external virtual link, or a virtual link port. In the latter case the external connection point node has an association relationship of type `VirtualLinksTo` to the internal virtual link node.

The following clauses illustrate the use of both models of VNF external connection point, re-exposure of an internal connection point and connected to an internal virtual link.

### A.3.2 External connection point re-exposing an internal connection point

In this case there is no need for a VnfExtCp node template. When substituting the VNFD low level service template for the abstract VNF node, the virtual\_link requirement of the abstract VNF node is mapped to the VDU connection point's virtual\_link requirement. This is shown in figure A.3.2-1. In this case the VNF Service Template does not include a node template of type `tosca.nodes.nfv.VnfExtCp`, but the functionality of it is provided by a VDU connection point.

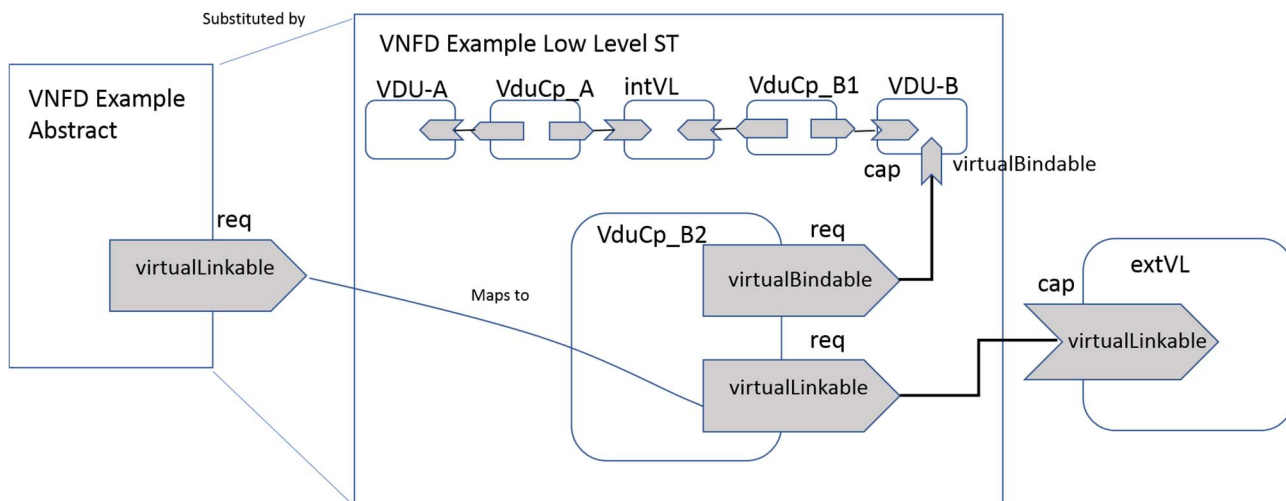


Figure A.3.2-1: VNFD with a VDU connection point acting as VnfExtCp

The following snippet shows the relevant part of the service template:

```
tosca_definitions_version: tosa_simple_yaml_1_2
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in NFV SOL
    001 GS

node_types:
  tosa.nodes.nfv.exampleVNF:
    derived from: tosa.nodes.nfv.VNF
    properties:
      # omitted for brevity
    requirements:
      - virtual_link:
          capability: tosa.capabilities.nfv.VirtualLinkable
    ...

topology_template:
  ...

substitution_mappings:
  node_type: tosa.nodes.nfv.exampleVNF
  requirements:
    virtual_link: [vduCp_B2, virtual_link]
```

```

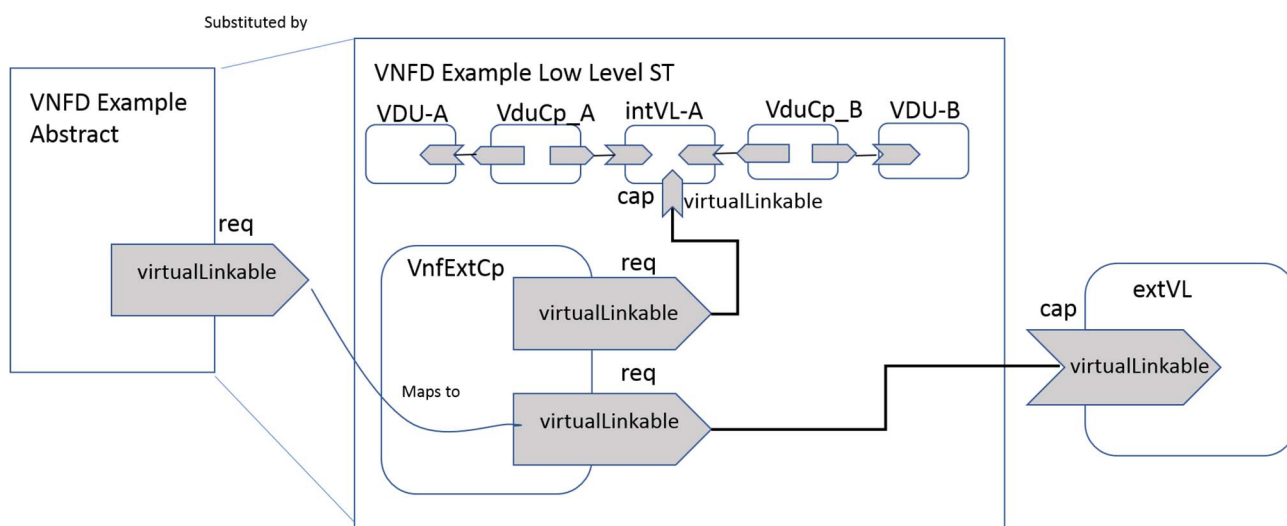
node_templates:
  vduCp_B2:
    type: toasca.nodes.nfv.VduCp
    # properties omitted for brevity
    requirements:
      - virtual_binding: VDU-B
      - virtual_link: # mapped to virtual_link requirement of VNF node

# other node template definitions (VDU-A, VDU-B, intVL, etc.):
...

```

### A.3.3 External connection point connected to an internal virtual link

In this case a VnfExtCp node template is needed. When substituting the VNFD low level service template for the abstract VNF node type, the `virtual_link` requirement of the abstract VNF node maps to the `external_virtual_link` requirement of the VnfExtCp node. The `internal_virtual_link` requirement of the VnfExtCp node is fulfilled with the corresponding capability of the internal VirtualLink node. This is shown in figure A.3.3-1.



**Figure A.3.3-1: VNFD with a VnfExtCp connected to an internal virtual link**

The following snippet shows the corresponding node template definition:

```

tosca_definitions_version: toasca_simple_yaml_1_2
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in NFV SOL
    001 GS

node_types:
  toasca.nodes.nfv.exampleVNF:
    derived from: toasca.nodes.nfv.VNF
    properties:
      # omitted for brevity
    requirements:
      - virtual_link:
          capability: toasca.capabilities.nfv.VirtualLinkable

topology_template:
...

```

```

substitution_mappings:
  node_type: toasca.nodes.nfv.exampleVNF
  requirements:
    virtual_link: [myMRFExtCp, external_virtual_link]

node_templates:
  myMRFExtCp:
    type: toasca.nodes.nfv.VnfExtCp
    # properties omitted for brevity
    ...
    requirements:
      - external_virtual_link: # mapped to virtual_link requirement of VNF node
      - internal_virtual_link: intVL-A

intVL-A:
  type: toasca.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ipv4 ]
      flow_pattern: mesh
    description: Internal VL
    vl_profile:
      max_bitrate_requirements:
        root: 1000000
        leaf: 100000
      min_bitrate_requirements:
        root: 100000
        leaf: 10000
      qos:
        latency: 100 ms
        packet_delay_variation: 80 ms
        packet_loss_ratio: 0.00001
    capabilities:
      virtual_linkable:

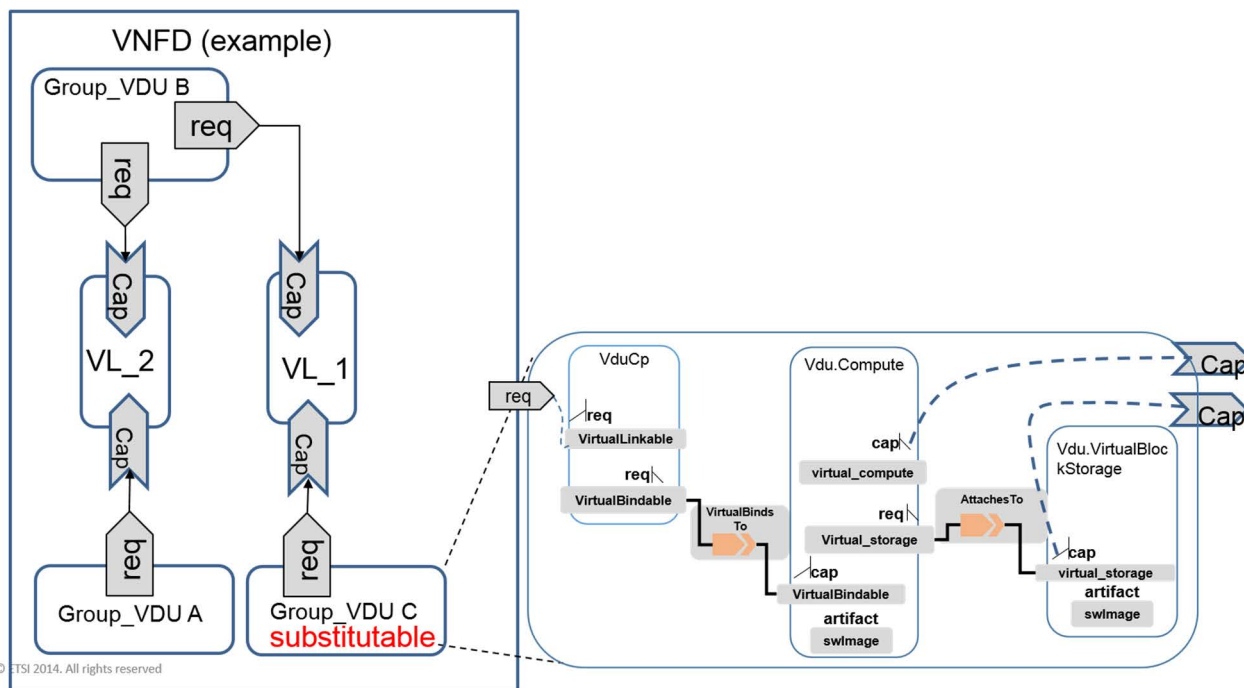
# Other node templates e.g. VDU-A, VDU-B, VduCP_A, etc.:
...

```

---

## A.4 VNFD modelling design example by using TOSCA composition

The following example in figure A.4-1 shows a VNFD descriptor contains three VDUs, which are interconnected by two virtualLinks.



**Figure A.4-1: Example of using substitution mapping for the VNFD design**

In this example, a separate service template is used to describe the composition of Vdu.Compute, Vdu.VirtualBlockStorage and VduCp, and then substituted as a node template in a VNFD service template. `tosca.nodes.nfv.groupVDU_A`, `tosca.nodes.nfv.groupVDU_B`, `tosca.nodes.nfv.groupVDU_C` types are used for substitution\_mapping.

```

tosca_definitions_version: tosca_simple_yaml_1_2
imports:
  - etsi_nfv_sol001_vnfd_2_6_1_types.yaml # all of TOSCA VNFD types as defined in
    NFV SOL 001 GS

topology_template:
  description: Template of a VNFD example

  node_templates:
    Group_VDU_A:
      type: tosca.nodes.nfv.groupVDU_A
      requirements:
        - virtual_link: VL_2

    Group_VDU_B:
      type: tosca.nodes.nfv.groupVDU_B
      requirements:
        - virtual_link: VL_2
        - virtual_link1: VL_1

    Group_VDU_C:
      type: tosca.nodes.nfv.groupVDU_C # the description of this type is described
                                       # in another service template.
      requirements:
        - virtual_link: VL_1

    VL_1:
      type: tosca.nodes.nfv.VnfVirtualLink
      properties:
        # omitted here for brevity

```

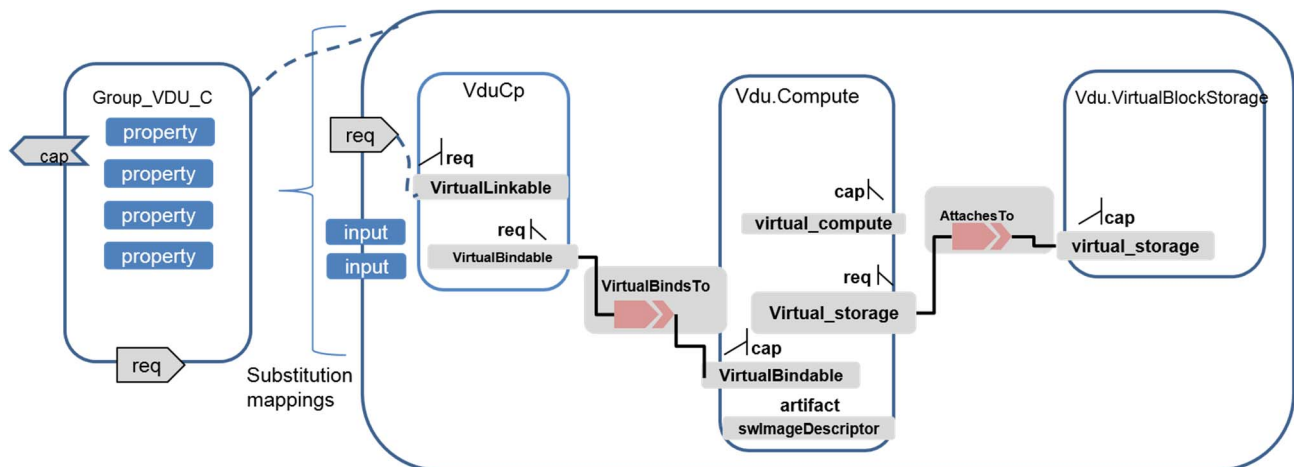
```

capabilities:
  virtual_linkable:

VL_2:
  type: toasca.nodes.nfv.VnfVirtualLink
  properties:
# omitted here for brevity
  capabilities:
    virtual_linkable:

```

The TOSCA service template example of Group\_VDU C is showing in figure A.4-2.



© ETSI 2014. All rights reserved

**Figure A.4-2: Example of composition Vdu.Compute, Vdu.VirtualBlockStorage and VduCp together using TOSCA substitution mapping**

```

tosca_definitions_version: toasca_simple_yaml_1_2

description: composition of Vdu.Compute, Vdu.VirtualBlockStorage and VduCp.

topology_template:
  substitution_mappings:
    node_type: toasca.nodes.nfv.groupVDU_C # substituted as a node type
  requirements:
    virtual_link: [ internalCpd, virtual_link ]
  capabilities:
    virtual_compute: [ vduC_compute, virtual_compute ]
    virtual_storage: [ vduC_storage, virtual_storage ]

  node_templates:
    vduC_compute:
      type: toasca.nodes.nfv.Vdu.Compute
      properties:
        # omitted here for brevity
      capabilities:
        virtual_compute:
      artifacts:
        sw_image:
      requirements:
        - virtual_storage: vduC_storage

```



```

vduC_storage:
  type: toska.nodes.nfv.Vdu.VirtualBlockStorage
  properties:
    # omitted here for brevity
  capabilities:
    virtual_storage:

internalCpd:
  type: toska.nodes.nfv.VduCp
  properties:
# omitted here for brevity
  requirements:
    - virtual_link:
    - virtual_binding: vduC_compute

```

## A.5 VNFD with Single deployment flavour modelling design example

In this example, there is one deployment flavour applied to this VNFD, and TOSCA-Simple-Profile-yaml-v1.1 is used for designing and processing this VNFD TOSCA model.

The one service template design illustrated by this example is only applicable when the VNF has only one deployment flavour.

The service template is the main entry point in the VNF Package, i.e. the Entry-definitions file, and is deployed as a stand-alone service template, i.e. without substituting for a node template. However, the service template still contains substitution\_mappings to indicate its ability to substitute for a node template of the specific node type.

The service template contains a node template of type MyCompany.SunshineDB.1\_0.1\_0. The node template contains the properties defined in the node type definition and implementations for the LCM interfaces.

### sunshinesimple.vnfd.tosca.yaml

This example illustrates a VNF with one VDU.Compute nodes (dbBackend) with two VDU connection points and one VNF virtual link (see figure A.5-1). The VNF is also connected to an external virtual link. The flavour\_id is "simple".

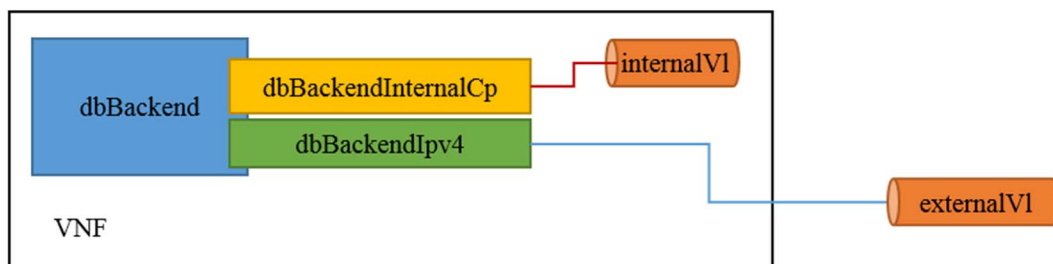


Figure A.5-1: SunshineDB (simple)

```

tosca_definitions_version: toska_simple_yaml_1_1

description: Relational database, simple

imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in NFV SOL
001 GS for a VNFD

```

```

data_types:
  MyCompany.datatypes.nfv.VnfInstantiateAdditionalParameters:
    derived_from: tosca.datatypes.nfv.VnfOperationAdditionalParameters
    properties:
      parameter_1:
        type: string
        required: true
        default: value_1
      parameter_2:
        type: string
        required: false
        default: value_2

node_types:
  MyCompany.SunshineDB.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      provider:
        type: string
        constraints: [ equal: MyCompany ]
        default: MyCompany
      product_name:
        type: string
        constraints: [ equal: SunshineDB ]
        default: SunshineDB
      software_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      descriptor_version:
        type: string
        constraints: [ equal: '1.0' ]
        default: '1.0'
      flavour_id:
        type: string
        constraints: [ equal: simple ]
        default: simple
      flavour_description:
        type: string
        default: ""
      vnfm_info:
        type: list
        entry_schema:
          type: string
          constraints: [ equal: '0:MyCompany-1.0.0' ]
          default: [ '0:MyCompany-1.0.0' ]
    interfaces:
      Vnflcm:
        type: tosca.interfaces.nfv.Vnflcm
        instantiate:
          inputs:
            additional_parameters:
              type: MyCompany.datatypes.nfv.VnfInstantiateAdditionalParameters
        #terminate:

```

```

topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineDB.1_0.1_0
  requirements:
    virtual_link: [ dbBackendIpv4, virtual_link ] # IPv4 for SQL

node_templates:
  SunshineDB:
    type: MyCompany.SunshineDB.1_0.1_0
    properties:
      descriptor_id: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      provider: MyCompany
      product_name: SunshineDB
      software_version: '1.0'
      descriptor_version: '1.0'
      vnf_info:
        - '0:MyCompany-1.0.0'
      flavour_id: simple
      flavour_description: 'vnf simple flavour description'
    interfaces:
      Vnflcm:
        instantiate:
          implementation: instantiate.workbook.mistral.yaml
        terminate:
          implementation: terminate.workbook.mistral.yaml
        heal:
          implementation: heal.workbook.mistral.yaml

  dbBackend:
    type: toasca.nodes.nfv.Vdu.Compute
    properties:
      name: dbbackend
      description: dbBackend
      boot_order: []
      nfvi_constraints: []
      configurable_properties: {}
      vdu_profile:
        min_number_of_instances: 1
        max_number_of_instances: 4
    capabilities:
      virtual_compute:
        properties:
          virtual_memory:
            virtual_mem_size: 8192 MiB
          virtual_cpu:
            cpu_architecture: x86
            num_virtual_cpu: 2
            virtual_cpu_clock: 1800 MHz
    requirements:
      - virtual_storage: mariaDbStorage

  mariaDbStorage:
    type: toasca.nodes.nfv.Vdu.VirtualBlockStorage
    properties:
      virtual_block_storage_data:
        size_of_storage: '200 GB'
        rdma_enabled: true
      sw_image_data:
        name: Software of Maria Db
        version: '1.0'

```

```

checksum:
  algorithm: sha-256
  hash: b9c3036539fd7a5f87a1bf38eb05fdde8b556a1a7e664dbeda90ed3cd74b4f9d
container_format: bare
disk_format: qcow2
min_disk: 2 GB
min_ram: 8192 MiB
size: 2 GB
operating_system: Linux
supported_virtualisation_environments:
  - KVM
artifacts:
  sw_image:
    type: tosca.artifacts.nfv.SwImage
    file: maria.db.image.v1.0.qcow2

dbBackendInternalCp:
  type: tosca.nodes.nfv.VduCp
  properties:
    protocol: [ associated_layer_protocol: ipv4 ]
    trunk_mode: false
    layer_protocols: [ ipv4 ]
    role: leaf
    description: Internal connection point on an VL
  requirements:
    - virtual_binding: dbBackend
    - virtual_link: internalVl

internalVl:
  type: tosca.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ipv4 ]
      flow_pattern: mesh
    test_access: [ ]
    description: Internal Virtual link in the VNF
  vl_profile:
    max_bitrate_requirements:
      root: 100000
      leaf: 20000
    min_bitrate_requirements:
      root: 10000
      leaf: 10000

dbBackendIpv4:
  type: tosca.nodes.nfv.VduCp
  properties:
    protocol: [ associated_layer_protocol: ipv4 ]
    trunk_mode: false
    layer_protocols: [ ipv4 ]
    role: leaf
    description: External connection point to access the DB on IPv4
  requirements:
    #- virtual_link: # the target node is determined in the NSD

    - virtual_binding: dbBackend

```

## A.6 Scaling and Instantiation Level examples

### A.6.1 ScalingAspect and InstantiationLevels policies with uniform delta

This example shows instantiationLevels, and ScalingAspect policies types in the complex scaling scenario where scaling aspect delta is based on "uniform delta" values.

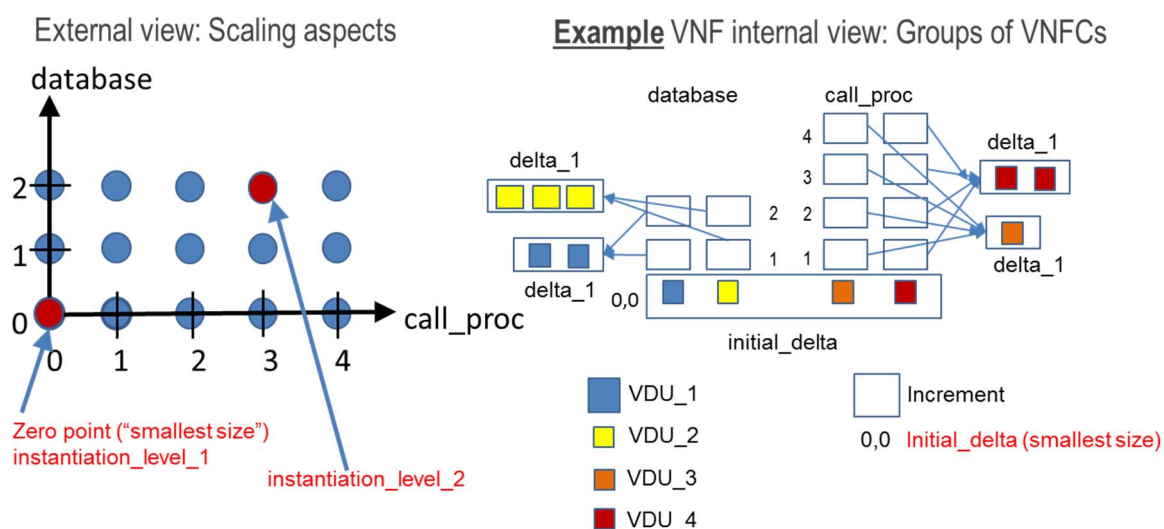


Figure A.6.1-1: Complex scaling example with uniform delta

```
tosca_definitions_version: tosca_simple_yaml_1_2
description: Complex scaling example (uniform delta value) described with policies
imports:
  - etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in NFV SOL
    001 GS

topology_template:
  node_templates:
    vdu_1:
      type: tosca.nodes.nfv.Vdu.Compute
      properties:
        name: ..
        description: ..
        boot_order:
          - ..
      vdu_profile:
        min_number_of_instances: 1
        max_number_of_instances: 5
      nfvi_constraints:
        - ..
      configurable_properties:
        additional_vnfc_configurable_properties: {}

    vdu_2:
      type: tosca.nodes.nfv.Vdu.Compute
      properties:
        name: ..
        description: ..
        boot_order:
          - ..
```

```
vdu_profile:
  min_number_of_instances: 1
  max_number_of_instances: 7
  nfvi_constraints:
    - ..
  configurable_properties:
    additional_vnfc_configurable_properties: {}

vdu_3:
  type: toasca.nodes.nfv.Vdu.Compute
  properties:
    name: ..
    description: ..
    boot_order:
      - ..
  vdu_profile:
    min_number_of_instances: 1
    max_number_of_instances: 5
  nfvi_constraints:
    - ..
  configurable_properties:
    additional_vnfc_configurable_properties: {}

vdu_4:
  type: toasca.nodes.nfv.Vdu.Compute
  properties:
    name: ..
    description: ..
    boot_order:
      - ..
  vdu_profile:
    min_number_of_instances: 1
    max_number_of_instances: 9
  nfvi_constraints:
    - ..
  configurable_properties:
    additional_vnfc_configurable_properties: {}

vl_1:
  type: toasca.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ipv4 ]
  vl_profile:
    min_bitrate_requirements:
      root: 1000000
    max_bitrate_requirements:
      root: 2000000

vl_2:
  type: toasca.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ipv4 ]
  vl_profile:
    min_bitrate_requirements:
      root: 1000000
    max_bitrate_requirements:
      root: 4000000
```

```
policies:
- scaling_aspects:
  type: toasca.policies.nfv.ScalingAspects
  properties:
    aspects:
      database:
        name: ..
        description: ..
        max_scale_level: 2
        step_deltas:
          - delta_1
      call_proc:
        name: ..
        description: ..
        max_scale_level: 4
        step_deltas:
          - delta_1

- vdu_1_initial_delta:
  type: toasca.policies.nfv.VduInitialDelta
  properties:
    initial_delta:
      number_of_instances: 1
  targets: [ vdu_1 ]

- vdu_1_scaling_aspect_deltas:
  type: toasca.policies.nfv.VduScalingAspectDeltas
  properties:
    aspect: database
    deltas:
      delta_1:
        number_of_instances: 2
  targets: [ vdu_1 ]

- vdu_2_initial_delta:
  type: toasca.policies.nfv.VduInitialDelta
  properties:
    initial_delta:
      number_of_instances: 1
  targets: [ vdu_2 ]

- vdu_2_scaling_aspect_deltas:
  type: toasca.policies.nfv.VduScalingDeltas
  properties:
    aspect: database
    deltas:
      delta_1:
        number_of_instances: 3
  targets: [ vdu_2 ]

- vl_1_bitrate_initial_delta:
  type: toasca.policies.nfv.VirtualLinkBitrateInitialDelta
  properties:
    initial_delta:
      bitrate_requirements:
        root: 1000000
  targets: [ vl_1 ]
```

```
- vl_1_bitrate_scaling_aspect_deltas:
  type: tosca.policies.nfv.VirtualLinkBitrateScalingAspectDeltas
  properties:
    aspect: database
    deltas:
      delta_1:
        bitrate_requirements:
          root: 1000000
    targets: [ vl_1 ]

- vdu_3_initial_delta:
  type: tosca.policies.nfv.VduInitialDelta
  properties:
    initial_delta:
      number_of_instances: 1
    targets: [ vdu_3 ]

- vdu_3_scaling_aspect_deltas:
  type: tosca.policies.nfv.VduScalingAspectDeltas
  properties:
    aspect: call_proc
    deltas:
      delta_1:
        number_of_instances: 1
    targets: [ vdu_3 ]

- vdu_4_initial_delta:
  type: tosca.policies.nfv.VduInitialDelta
  properties:
    initial_delta:
      number_of_instances: 1
    targets: [ vdu_4 ]

- vdu_4_scaling_aspect_deltas:
  type: tosca.policies.nfv.VduScalingAspectDeltas
  properties:
    aspect: call_proc
    deltas:
      delta_1:
        number_of_instances: 2
    targets: [ vdu_4 ]

- vl_2_bitrate_initial_delta:
  type: tosca.policies.nfv.VirtualLinkBitrateInitialDelta
  properties:
    initial_delta:
      bitrate_requirements:
        root: 1000000
    targets: [ vl_2 ]

- vl_2_bitrate_scaling_aspect_deltas:
  type: tosca.policies.nfv.VirtualLinkBitrateScalingAspectDeltas
  properties:
    aspect: call_proc
    deltas:
      delta_1:
        bitrate_requirements:
          root: 1000000
    targets: [ vl_2 ]
```



```
- instantiation_levels:
  type: tosca.policies.nfv.InstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        description: ..
        scale_info:
          database:
            scale_level: 0
          call_proc:
            scale_level: 0
      instantiation_level_2:
        description: ..
        scale_info:
          database:
            scale_level: 2
          call_proc:
            scale_level: 3
      default_level: instantiation_level_1

- vdu_1_instantiation_levels:
  type: tosca.policies.nfv.VduInstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        number_of_instances: 1
      instantiation_level_2:
        number_of_instances: 5
    targets: [ vdu_1 ]

- vdu_2_instantiation_levels:
  type: tosca.policies.nfv.VduInstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        number_of_instances: 1
      instantiation_level_2:
        number_of_instances: 7
    targets: [ vdu_2 ]

- vl_1_instantiation_levels:
  type: tosca.policies.nfv.VirtualLinkInstantiationLevels
  ..
  targets: [ vl_1 ]

- vdu_3_instantiation_levels:
  type: tosca.policies.nfv.VduInstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        number_of_instances: 1
      instantiation_level_2:
        number_of_instances: 4
    targets: [ vdu_3 ]

- vdu_4_instantiation_levels:
  type: tosca.policies.nfv.VduInstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        number_of_instances: 1
```

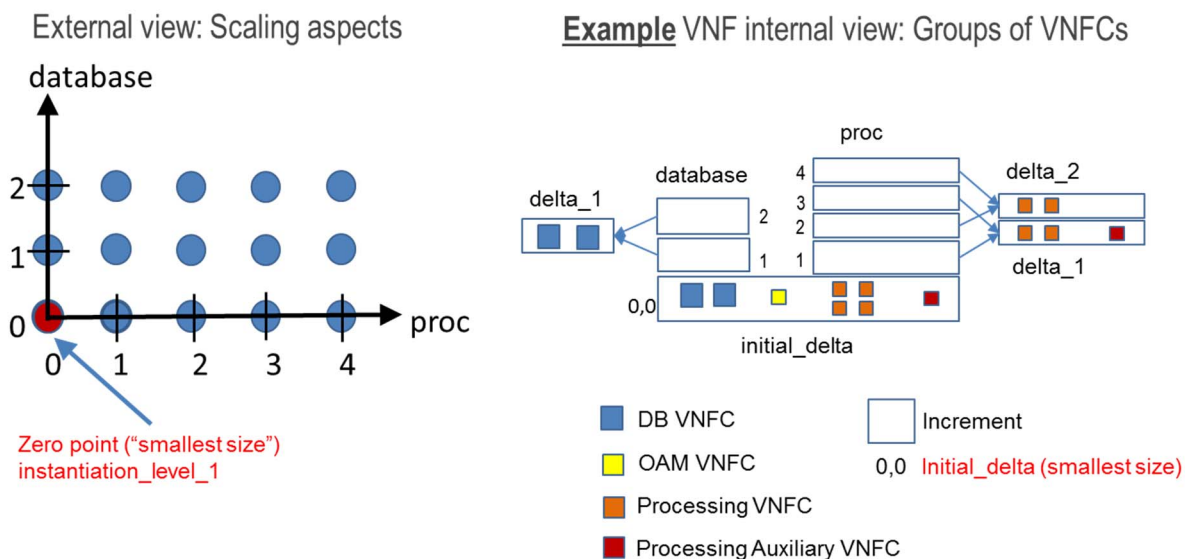
```

instantiation_level_2:
  number_of_instances: 7
  targets: [ vdu_4 ]

- v1_2_instantiation_levels:
  type: toasca.policies.nfv.VirtualLinkInstantiationLevels
  ..
  targets: [ v1_2 ]
    
```

## A.6.2 ScalingAspect and InstantationLevels policies with non-uniform deltas

This example shows instantiationLevel, ScalingAspect policies and group types in the complex scaling scenario where scaling aspect delta is based on "delta" (non-uniform) values for Processing Auxiliary VNFC.



**Figure A.6.2-1: Complex scaling example with non-uniform delta**

For simplicity, virtual link is not illustrated and instantiation level is one level (default) in this example, as these properties are already illustrated in clause A.6.1.

The "Proc" aspect contains a uniform delta part ("processing" and a non-uniform part ("processing\_auxiliary").

The "DB" aspect contains a non-uniform part ("db"), and no delta information for "oam".

```

tosca_definitions_version: toasca_simple_yaml_1_2
description: Complex example (uniform and non-uniform delta value) described with
policies
imports:
- etsi_nfv_sol001_vnfd_types.yaml # all of TOSCA VNFD types as defined in NFV SOL
001 GS

topology_template:
  node_templates:
    db:
      type: toasca.nodes.nfv.Vdu.Compute
      properties:
        name: ..
        description: ..
        boot_order:
          - ..
    
```

```
vdu_profile:
  min_number_of_instances: 2
  max_number_of_instances: 6
  nfvi_constraints:
    - ..
  configurable_properties:
    additional_vnfc_configurable_properties: {}

oam:
  type: toasca.nodes.nfv.Vdu.Compute
  properties:
    name: ..
    description: ..
    boot_order:
      - ..
    vdu_profile:
      min_number_of_instances: 1
      max_number_of_instances: 1
      nfvi_constraints:
        - ..
    configurable_properties:
      additional_vnfc_configurable_properties: {}

processing:
  type: toasca.nodes.nfv.Vdu.Compute
  properties:
    name: ..
    description: ..
    boot_order:
      - ..
    vdu_profile:
      min_number_of_instances: 4
      max_number_of_instances: 12
      nfvi_constraints:
        - ..
    configurable_properties:
      additional_vnfc_configurable_properties: {}

processing_auxiliary:
  type: toasca.nodes.nfv.Vdu.Compute
  properties:
    name: ..
    description: ..
    boot_order:
      - ..
    vdu_profile:
      min_number_of_instances: 1
      max_number_of_instances: 3
      nfvi_constraints:
        - ..
    configurable_properties:
      additional_vnfc_configurable_properties: {}

policies:
  - scaling_aspects:
      type: toasca.policies.nfv.ScalingAspects
      properties:
        aspects:
          database:
            name: ..
            description: ..
```

```

        max_scale_level: 2
        step_deltas:
          - delta_1
          - delta_1
      proc:
        name: ..
        description: ..
        max_scale_level: 4
        step_deltas:
          - delta_1
          - delta_2
          - delta_1
          - delta_2

- db_initial_delta:
  type: toasca.policies.nfv.VduInitialDelta
  properties:
    initial_delta:
      number_of_instances: 2
  targets: [ db ]

- db_scaling_aspect_deltas:
  type: toasca.policies.nfv.VduScalingAspectDeltas
  properties:
    aspect: database
    deltas:
      delta_1:
        number_of_instances: 2
  targets: [ db ]

- oam_initial_delta:
  type: toasca.policies.nfv.VduInitialDelta
  properties:
    initial_delta:
      number_of_instances: 1
  targets: [ oam ]

- processing_initial_delta:
  type: toasca.policies.nfv.VduInitialDelta
  properties:
    initial_delta:
      number_of_instances: 4
  targets: [ processing ]

- processing_scaling_aspect_deltas:
  type: toasca.policies.nfv.VduScalingAspectDeltas
  properties:
    aspect: proc
    deltas:
      delta_1:
        number_of_instances: 2
      delta_2:
        number_of_instances: 2
  targets: [ processing ]

- processing_auxiliary_initial_delta:
  type: toasca.policies.nfv.VduInitialDelta
  properties:
    initial_delta:
      number_of_instances: 1
  targets: [ processing_auxiliary ]

```

```
- processing_auxiliary_scaling_aspect_deltas:
  type: toasca.policies.nfv.VduScalingAspectDeltas
  properties:
    aspect: proc
    deltas:
      delta_1:
        number_of_instances: 1
      delta_2:
        number_of_instances: 0
    targets: [ processing_auxiliary ]

- instantiation_levels:
  type: toasca.policies.nfv.InstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        description: ..
        scale_info:
          database:
            scale_level: 0
        proc:
          scale_level: 0

- db_instantiation_levels:
  type: toasca.policies.nfv.VduInstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        number_of_instances: 2
    targets: [ db ]

- oam_instantiation_levels:
  type: toasca.policies.nfv.VduInstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        number_of_instances: 1
    targets: [ oam ]

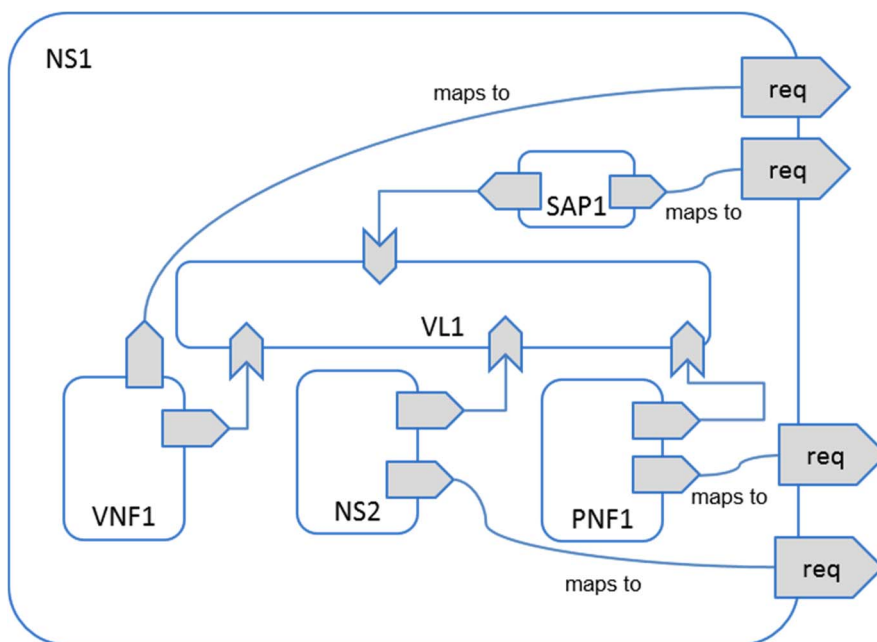
- processing_instantiation_levels:
  type: toasca.policies.nfv.VduInstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        number_of_instances: 4
    targets: [ processing ]

- processing_auxiliary_instantiation_levels:
  type: toasca.policies.nfv.VduInstantiationLevels
  properties:
    levels:
      instantiation_level_1:
        number_of_instances: 1
    targets: [ processing_auxiliary ]
```

## A.7 Service Access Point

### A.7.1 General

A SAP may either be a VNF external connection point (or a PNF external connection points or a Sap of a nested NS of this NS) which is exposed as CP of one of the NS constituents (VnfExtCp, PnfExtCp, nested NS Sap externally, or be exposed as a new CP on NS virtual link. This is shown in figure A.7.1-1. In the latter case the Sap node has an association relationship of type VirtualLinksTo to the NsVirtualLink node.

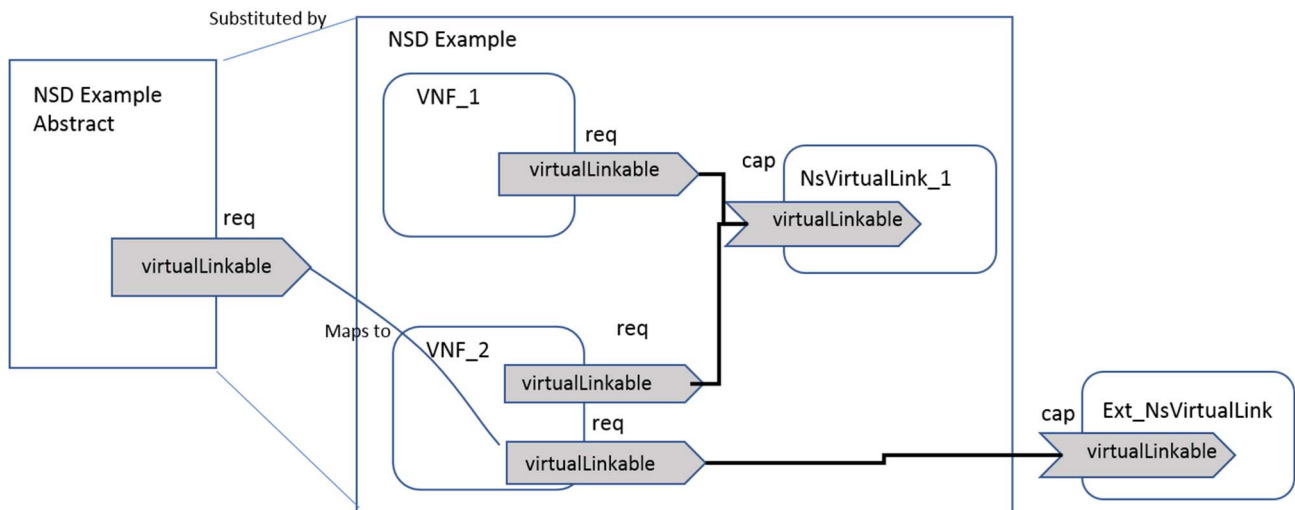


**Figure A.7.1-1: Overview of SAP options**

The following clauses illustrate the use of both models of SAP, re-exposure of a VNF external connection point and connected to an NS virtual link.

### A.7.2 VNF External connection point exposing as a SAP

In this case, Sap node template is not required. When substituting the NSD service template for the abstract NS node, the virtual\_link requirement of the abstract NS node is mapped to the VNF external connection point's virtual\_link requirement. This is shown in figure A.7.2-1. In this case the NS Service Template does not include a node template of type `tosca.nodes.nfv.Sap`, but the functionality of it is provided by a VNF external connection point.



**Figure A.7.2-1: VNF\_2 External connection point (VnfExtCp) exposed as a Sap**

The following snippet shows the relevant part of the NS service template:

```

tosca_definitions_version: tosca_simple_yaml_1_2
imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of TOSCA NSD types as defined in NFV SOL 001
  GS

node_types:
  tosca.nodes.nfv.exampleNS
  derived from: tosca.nodes.nfv.NS
  properties:
    # omitted for brevity
  requirements:
    - virtual_link:
      capability:
        tosca.capabilitites.nfv.VirtualLinkable
    ...

topology_template:
  ...

substitution_mappings:
  node_type: tosca.nodes.nfv.exampleNS
  requirements:
    virtual_link: [VNF_2, virtual_link_2] # the External connection point of VNF_2
                                           # will be used as the Sap of this NS

node_templates:
  VNF_2:
    type: tosca.nodes.nfv.exampleVNF2
    # properties omitted for brevity
    requirements:
      - virtual_link_1: NsVirtualLink_1 # connects to the External connection
                                        # point which maps to the
                                        # virtual_link_1 requirement of VNF_2
      - virtual_link_2: # mapped to virtual_link requirement of NS node

```

```
# other node template definitions (VNF_1, NsVirtualLink_1 etc.):
...
```

### A.7.3 SAP connected to an NS virtual link

In this case, a Sap node template is required. When substituting the NSD service template for the abstract NS node type, the virtual\_link requirement of the abstract NS node maps to the virtual\_link requirement of the Sap node. This is shown in figure A.7.3-1.

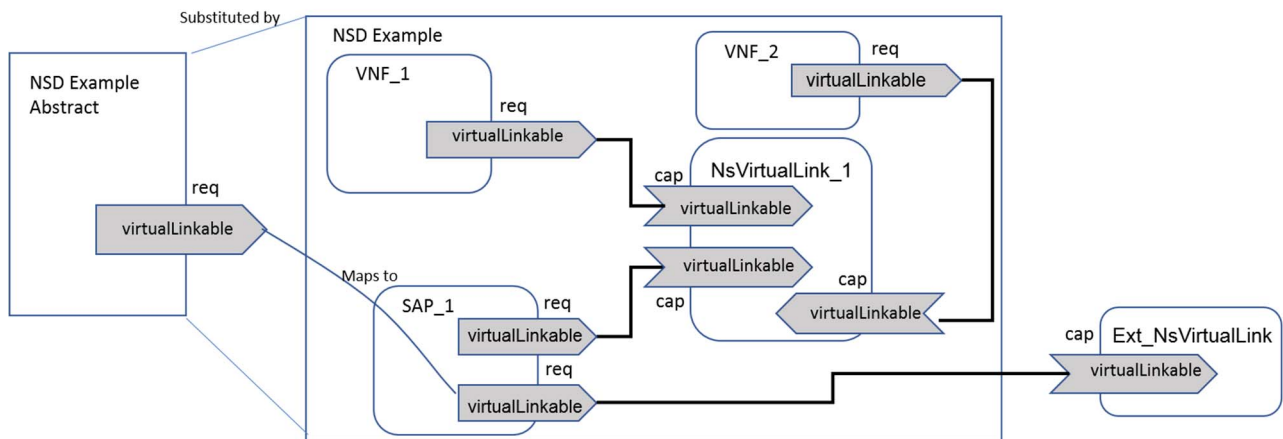


Figure A.7.3-1: SAP\_1 exposed as new CP on an internal virtual link

The following snippet shows the corresponding node template definition:

```
tosca_definitions_version: tosca_simple_yaml_1_2
imports:
- etsi_nfv_sol001_nsd_types.yaml # all of TOSCA NSD types as defined in NFV SOL 001
GS

node_types:
  tosca.nodes.nfv.exampleNS
  derived from: tosca.nodes.nfv.NS
  properties:
  # omitted for brevity
  requirements:
  - virtual_link:
    capability:
      tosca.capabilitites.nfv.VirtualLinkable

topology_template:
...

substitution_mappings:
  node_type: tosca.nodes.nfv.exampleNS
  requirements:
    virtual_link: [SAP_1, external_virtual_link] #SAP_1 is the Sap of the NS

node_templates:
  SAP_1:
    type: tosca.nodes.nfv.Sap
    # properties omitted for brevity
    ...
    requirements:
      internal_virtual_link: NsVirtualLink_1
```



```

external_virtual_link: # map to virtual_link requirement of the NS node

# Other node templates e.g. VNF_1, VNF_2, NsVirtualLink_1, etc.:
...

```

## A.8 NSD with Single deployment flavour modelling design example

In this example, there is one deployment flavour applied to this NSD, and TOSCA-Simple-Profile-yaml-v1.1 is used for designing and processing this NSD TOSCA model.

The one service template design illustrated by this example is only applicable when the NS has only one deployment flavour.

The service template is the main entry point in the NSD file structure, i.e. the Entry-definitions file, and is deployed as a stand-alone service template, i.e. without substituting for a node template. However, the service template still contains substitution\_mappings to indicate its ability to substitute for a node template of the specific node type.

### example\_NS.yaml

This example illustrates a NS contains two VNF: VNF\_1 and VNF\_2, they connect through NsVirtualLink\_1. One of the VnfExtCp of VNF\_2 is exposed as the Sap of this NS. The flavour\_id is "simple". In the VNF\_2 node template, it defines a dependency requirement to VNF\_1, which indicates that all the instances of VNF\_1 have to be deployed first before deploy the instances of VNF\_2.

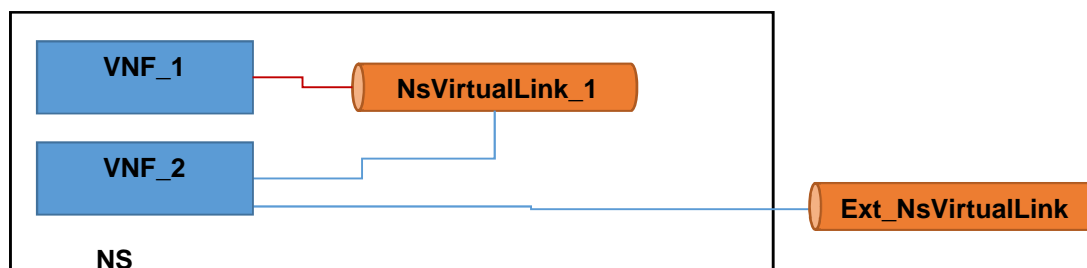


Figure A.8-1: example\_NS (simple)

```

tosca_definitions_version: tosca_simple_yaml_1_1

description: Relational database, simple

imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of NSD related TOSCA types as defined in NFV
    SOL 001 GS

node_types:
  tosca.example_NS:
    derived_from: tosca.nodes.nfv.NS
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177

```

```

designer:
  type: string
  constraints: [ equal: MyCompany ]
  default: MyCompany
name:
  type: string
  constraints: [ equal: ExampleService ]
  default: ExampleService
version:
  type: string
  constraints: [ equal: '1.0' ]
  default: '1.0'
invariant_id:
  type: string
  constraints: [ equal: 1111-2222-aaaa-bbbb ]
  default: 1111-2222-aaaa-bbbb
flavour_id:
  type: string
  constraints: [ equal: simple ]
  default: simple

topology_template:
  substitution_mappings:
    node_type: toska.example_NS
  requirements:
    virtual_link: [ VNF_2, virtual_link_2 ] # the External connection point of
                                           # VNF_2 is exposed as the Sap

node_templates:
  my_service:
    type: toska.example_NS
    properties:
      descriptor_id: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      designer: MyCompany
      name: ExampleService
      version: '1.0'
      invariant_id: 1111-2222-aaaa-bbbb
      flavour_id: simple
    interfaces:
      Nslcm:
        instantiate:
          implementation: instantiate.workflow.yaml
        terminate:
          implementation: terminate.workflow.yaml

  VNF_1:
    type: toska.nodes.nfv.example_VNF1
    properties:
      # no property assignments needed for required properties that have a default
      # value assigned in the node type definition, e.g. descriptor_id
      flavour_id: simple
      vnf_profile:
        instantiation_level: level_1
        min_number_of_instances: 2
        max_number_of_instances: 6
    requirements:
      - virtual_link: NsVirtualLink_1

  VNF_2:
    type: toska.nodes.nfv.example_VNF2

```

```

properties:
  flavour_id: simple
  vnf_profile:
    instantiation_level: level_1
    min_number_of_instances: 1
    max_number_of_instances: 3
requirements:
  - virtual_link_1: NsVirtualLink_1
  - virtual_link_2: # map to virtual_link requirement of the NS node
  - dependency: VNF_1

```

```

NsVirtualLink_1:
  type: toska.nodes.nfv.NsVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ipv4]
      flow_pattern: mesh
  vl_profile:
    max_bitrate_requirements:
      root: 1000
    min_bitrate_requirements:
      root: 1000

```

## A.9 Mapping between NFV IM and TOSCA concepts

### A.9.1 Introduction

This annex describes the mapping between the NFV information model and the TOSCA concepts.

### A.9.2 Mapping between ETSI GS NFV-IFA 011 IM and TOSCA concepts

Table A.9.2-1 illustrates the mapping between the information model as specified in ETSI GS NFV-IFA 011 [1] and the corresponding TOSCA concepts.

**Table A.9.2-1: Mapping between ETSI GS NFV-IFA 011 [1] IM and TOSCA concepts**

ETSI GS NFV-IFA 011 [1] information element	TOSCA concept
VNFD (clause 7.1.2)	TOSCA service template(s) in the VNF package
VnfExtCpd (clause 7.1.4)	node template with type toska.nodes.nfv.VnfExtCp or toska.nodes.nfv.VduCp
VnfLcmOperationsConfiguration (clause 7.1.5)	property of VNF node type with data type toska.datatypes.nfv.VnfConfigurableProperties
Vdu (clause 7.1.6)	node template with type toska.nodes.nfv.Vdu.Compute
VLd (clause 7.1.7)	node template with type toska.nodes.nfv.VnfVirtualLink
DeploymentFlavour (clause 7.1.8)	lower level service template(s) in the VNF package
VduProfile (clause 7.1.8.3)	property of Vdu.Compute node type with data type toska.datatypes.nfv.VduProfile
VirtualLinkProfile (clause 7.1.8.4)	property of VnfVirtualLink node type with data type toska.datatypes.nfv.VIProfile

ETSI GS NFV-IFA 011 [1] information element	TOSCA concept
VirtualLinkDescFlavour (clause 7.1.8.5)	Only qos attribute in VirtualLinkDescFlavour has been defined as property of the VIPProfile data type
InstantiationLevel (clause 7.1.8.7)	policy with type <code>tosca.policies.nfv.InstantiationLevels</code>
VduLevel (clause 7.1.8.9)	policy with type <code>tosca.policies.nfv.VduInstantiationLevels</code>
LocalAffinityOrAntiAffinityRule (clause 7.1.8.11)	policy with type <code>tosca.policies.nfv.AffinityRule</code> or <code>tosca.policies.nfv.AntiAffinityRule</code>
AffinityOrAntiAffinityGroup (clause 7.1.8.12)	policy with type <code>tosca.policies.nfv.AffinityRule</code> or <code>tosca.policies.nfv.AntiAffinityRule</code>
VirtualComputeDesc (clause 7.1.9.2.2)	VirtualCompute capability of the Vdu.Compute node template
VirtualStorageDesc with BlockStorageData (clause 7.1.9.4.3)	node template with type <code>tosca.nodes.nfv.Vdu.VirtualBlockStorage</code>
VirtualStorageDesc with ObjectStorageData (clause 7.1.9.4.4)	node template with type <code>tosca.nodes.nfv.Vdu.VirtualObjectStorage</code>
VirtualStorageDesc with FileStorageData (clause 7.1.9.4.5)	node template with type <code>tosca.nodes.nfv.Vdu.VirtualFileStorage</code>
ScalingAspect (clause 7.1.10.2)	policy with type <code>tosca.policies.nfv.ScalingAspects</code>
ScalingDelta (clause 7.1.10.4)	policy with type <code>tosca.policies.nfv.VduScalingAspectDeltas</code> and/or <code>tosca.policies.nfv.VirtualLinkBitrateScalingAspectDeltas</code>
VnfIndicator (clause 7.1.11.2)	Not supported in this version of the present document
MonitoringParameter (clause 7.1.11.3)	property of the VNF, Vdu.Compute and VnfVirtualLink node types
VnfConfigurableProperties (clause 7.1.12)	property of VNF node template with data type <code>tosca.datatypes.nfv.VnfConfigurableProperties</code>
LifeCycleManagementScript (clause 7.1.13)	interface with type <code>tosca.interfaces.nfv.Vnflcm</code>
VnfInfoModifiableAttributes (clause 7.1.14)	property of VNF node template with data type <code>tosca.datatypes.nfv.VnfInfoModifiableAttributes</code>

### A.9.3 Mapping between ETSI GS NFV-IFA 014 IM and TOSCA concepts

Table A.9.3-1 illustrates the mapping between the information model as specified in ETSI GS NFV-IFA 014 [2] and the corresponding TOSCA concepts.

**Table A.9.3-1: Mapping between ETSI GS NFV-IFA 014 [2] IM and TOSCA concepts**

ETSI GS NFV-IFA 014 [2] information element	TOSCA concept
NSD (clause 6.2.2)	TOSCA service template(s) in the NSD file structure
Sapd (clause 6.2.3)	node template with type <code>tosca.nodes.nfv.Sap</code> or the <code>virtual_link</code> requirement in the <code>substitution_mapping</code>
MonitoredData (clause 6.2.6)	Not supported in this version of the present document
VnfIndicatorData (clause 6.2.7)	Not supported in this version of the present document
LifeCycleManagementScript (clause 6.2.9)	Interface with type <code>Nslcm</code>
NsDf (clause 6.3)	lower level service template(s) in the NSD file structure
VnfProfile (clause 6.3.3)	property of VNF node template with data type <code>tosca.datatypes.nfv.VnfProfile</code>

ETSI GS NFV-IFA 014 [2] information element	TOSCA concept
VirtualLinkProfile (clause 6.3.4)	property of NsVirtualLink node template with data type <code>tosca.datatypes.nfv.VIPProfile</code>
AffinityOrAntiAffinityGroup (clause 6.3.5)	policy with type <code>tosca.policies.nfv.AffinityRule</code> or <code>tosca.policies.nfv.AntiAffinityRule</code>
NsVirtualLinkConnectivity (clause 6.3.7)	requirement and capability between PnfExtCp or VnfExtCp and NsVirtualLink node templates
NsLevel (clause 6.3.9)	Not supported in this version of the present document
NsScaleInfo (clause 6.3.10)	Not supported in this version of the present document
NsProfile (clause 6.3.11)	property of NS node template with data type <code>tosca.datatypes.nfv.NsProfile</code>
Dependencies (clause 6.3.12)	dependency requirements
Vnffgd (clause 6.4.2)	Not supported in this version of the present document
Nfnd (clause 6.4.3)	Not supported in this version of the present document
CpdPool (clause 6.4.4)	Not supported in this version of the present document
NfpPositionDesc (clause 6.4.5)	Not supported in this version of the present document
CpProfile (clause 6.4.6)	Not supported in this version of the present document
NsVirtualLinkDesc (clause 6.5.2)	node template with type <code>tosca.nodes.nfv.NsVirtualLink</code>
VirtualLinkDf (clause 6.5.4)	Only the <code>serviceAvailabilityLevel</code> attribute in <code>VirtualLinkDf</code> has been defined as property of the <code>NsVirtualLink</code> node type
Pnfd (clause 6.6.2)	A standalone TOSCA service template
PnfExtCpd (clause 6.6.4)	node template with type <code>tosca.nodes.nfv.PnfExtCp</code>
NsScalingAspect (clause 6.7.2)	Not supported in this version of the present document
VnfToLevelMapping (clause 6.7.4)	Not supported in this version of the present document
VirtualLinkToLevelMapping (clause 6.7.5)	Not supported in this version of the present document
NsToLevelMapping (clause 6.7.6)	Not supported in this version of the present document

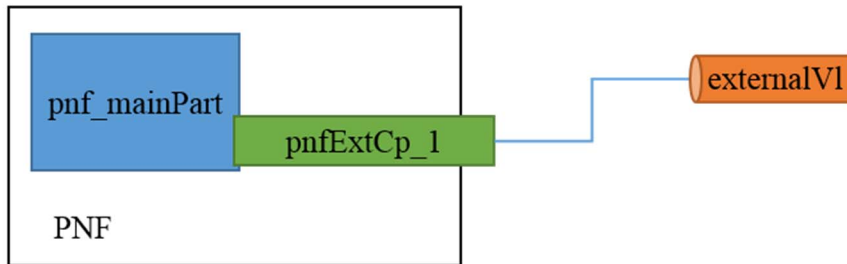
## A.10 PNFD modelling design example

In this example, `TOSCA-Simple-Profile-yaml-v1.1` is used for designing and processing a PNFD TOSCA model.

The service template contains a node template of type `MyCompany.examplePnf.1_0` which represents the main part of the PNF and a node template of type `tosca.nodes.nfv.PnfExtCp` representing the PNF external connection point.

### examplePnf.yaml

This example illustrates a PNF with one external connection point, `pnfExtCp_1`. All the rest parts of the PNF is described as a single box, e.g. called `pnf_mainPart` in the PNFD.



**Figure A.10-1: examplePnf**

```

tosca_definitions_version: tosca_simple_yaml_1_1

description: the service template of a PNFD

imports:
  - etsi_nfv_sol001_pnfd_types.yaml

node_types:
  MyCompany.examplePnf.1_0:
    derived_from: tosca.nodes.nfv.PNF
    properties:
      descriptor_id:
        type: string
        constraints: [ equal: b1bb0ce7-ebca-4fa7-95ed-4840d70a2233 ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a2233
      function_description:
        type: string
        default: an example PNF
    provider:
      type: string
      constraints: [ equal: MyCompany ]
      default: MyCompany
    version:
      type: string
      constraints: [ equal: '1.0' ]
      default: '1.0'
    descriptor_invariant_id:
      type: string
      constraints: [ equal: 1111-2222-ccaa-bbdd ]
      default: 1111-2222-ccaa-bbdd
    name:
      type: string
      constraints: [ equal: ExamplePnf ]
      default: ExamplePnf
    requirements:
      - virtual_link:
          capability: tosca.capabilities.nfv.VirtualLinkable

topology_template:
  substitution_mappings:
    node_type: MyCompany.examplePnf.1_0
  requirements:
    virtual_link: [ pnfExtCp_1, external_virtual_link ]

node_templates:
  pnf_mainPart:
    type: MyCompany.examplePnf.1_0

```

```
properties:
  descriptor_id: b1bb0ce7-ebca-4fa7-95ed-4840d70a2233
  function_description: an example PNF
  provider: MyCompany
  version: 1.0
  descriptor_invariant_id: 1111-2222-ccaa-bbdd
  name: ExamplePnf

pnfExtCp_1:
  type: toasca.nodes.nfv.PnfExtCp
  properties:
    trunk_mode: false
    layer_protocols: [ ipv4 ]
    role: leaf
    description: External connection point to access this pnf
  requirements:
    # - external_virtual_link:
    - dependency: pnf_mainPart
```

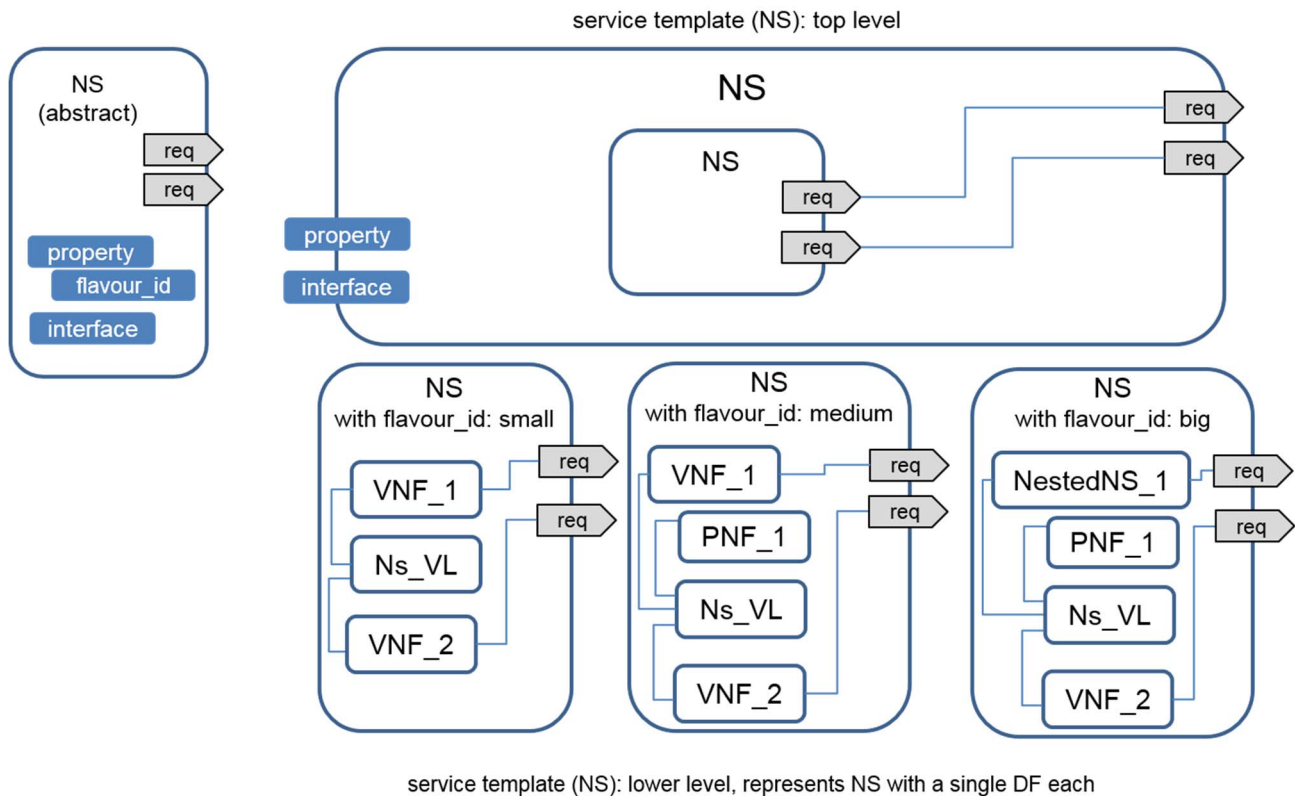
---

## A.11 NSD with Multiple deployment flavour modelling design example

Deployment flavours are represented as deployable TOSCA topology templates. This way one NS service template represents one NS deployment flavour, and different deployment flavours are described by different NS service templates. This is in line with the concept that different deployment flavours can define different topologies of the same NS, with different scaling aspects, different constituent VNFs, PNFs and nested NSs, and different internal connectivity.

In order to represent a NS, a top-level service template is used. This top-level service template contains a topology template with only an abstract NS node which defines the common parts of the different deployment flavours (such as designer, version and parts of the lifecycle management interface definition). It also sets a constraint on the deployment flavour property (the required value of the flavour\_id property); this constraint comes from the NS instantiation request which contains a flavour\_id selected among those available in the NSD.

As a result, the NFVO will look into the available further service templates representing the different NS deployment flavours of the NS and use the one that has the matching flavour\_id property value to substitute for the abstract NS. These are the low-level service templates.



**Figure A.11-1: NSD overview with multiple deployment flavour**

A NSD contains a TOSCA top-level Service Template as entry point in the NSD file structure and one or more TOSCA low-level Service templates representing the different deployment flavours (see figure A.11-1). The NSD is interpreted by an NFVO. In this example, the templates describe two variants of the NS each corresponding to a deployment flavour: a small and a big one. The small NS consists of two VNFs one NS Virtual link and, the big VNF variant consists of three VNFs and one NS Virtual link.

#### NSD-top levelMyExampleNs.yaml

```
tosca_definitions_version: toska_simple_yaml_1_2

description: my service
imports:
  - etsi_nfv_sol001_nsd_types.yaml
  - MyExampleNs_Type.yaml # contains the NS node type definition
  - MyExampleNs_small.yaml
  - MyExampleNs_big.yaml

topology_template:
  inputs:
    nsFlavourId:
      type: string
      description: NS deployment flavour selected by the consumer. It is provided in
the SOL005 API

  node_templates:
    myexampleNs:
      type: toska.MyExampleNS
      properties:
        descriptor_id: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
        designer: MyCompany
        name: ExampleService
        version: '1.0'
```



```

    invariant_id: 1111-2222-aaaa-bbbb
    flavour_id: {get_input: nsFlavourId}
    # get_input function would be used by a TOSCA orchestrator at run time to
access the nsFlavourId. If the deployment is not done by a TOSCA orchestrator above
function may not be needed
    requirements:
    #- virtual_link # mapped in lower-level templates

```

The `MyExampleNs_Type.yaml` file has the following content:

```

tosca_definitions_version: tosca_simple_yaml_1_2

description: type definition of tosca.MyExampleNS

imports:
- etsi_nfv_sol001_nsd_types.yaml # all of TOSCA NSD types as defined in NFV SOL 001
GS

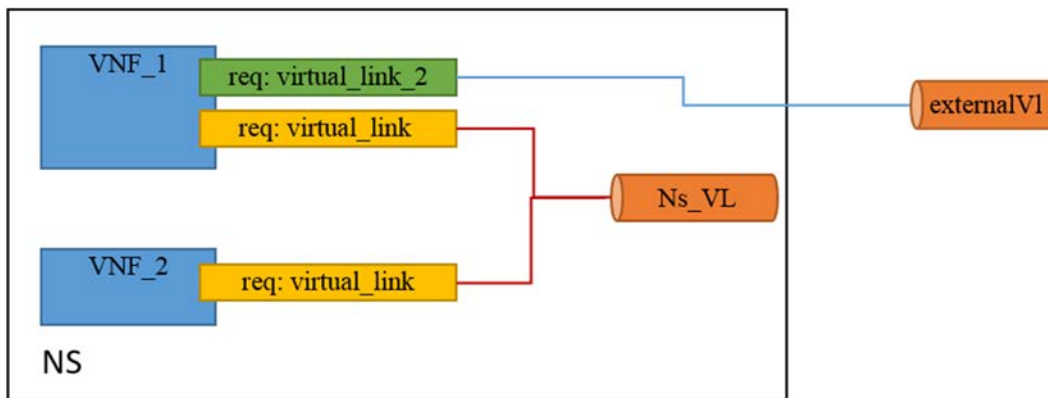
node_types:
  tosca.MyExampleNS:
    derived_from: tosca.nodes.nfv.NS
    properties:
      descriptor_id:
        type: string
        constraints: [ valid_values: [ b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ] ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      designer:
        type: string
        constraints: [ valid_values: [ MyCompany ] ]
        default: MyCompany
      name:
        type: string
        constraints: [ valid_values: [ ExampleService ] ]
        default: ExampleService
      version:
        type: string
        constraints: [ valid_values: [ '1.0' ] ]
        default: '1.0'
      invariant_id:
        type: string
        constraints: [ valid_values: [ 1111-2222-aaaa-bbbb ] ]
        default: 1111-2222-aaaa-bbbb
      flavour_id:
        type: string
        constraints: [ valid_values: [ small, big ] ]
        default: small
    requirements:
    - virtual_link:
        capability: tosca.capabilities.nfv.VirtualLinkable
  interfaces:
    Nslcm:
      type: tosca.interfaces.nfv.Nslcm

```

The NS node template in the `myexample_NS.yaml` file is abstract and is subject to substitution; the lower-level templates in the subsequent sections provide these substitutions. The actual lower-level template is selected based on the node type and a value constraint on the `flavour_id` property.

Each low level service template contains a node template of type `tosca.MyExampleNS` with implementation of the LCM interfaces.

**MyExampleNs** (small): Lower level, contains 2 VNFs and 1 NS virtual link.



**Figure A.11-2: MyExampleNs (simple): Lower level**

### MyExampleNs\_small.yaml

```

tosca_definitions_version: tosca_simple_yaml_1_2

description: myExampleNs with small flavour
imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of TOSCA NSD types as defined in NFV SOL 001
  GS
  - MyExampleNs_Type.yaml # contains the NS node type definition
  - example_VNF1_Type.yaml # contains the tosca.nodes.nfv.example_VNF1 node type
  definition
  - example_VNF2_Type.yaml # contains the tosca.nodes.nfv.example_VNF2 node type
  definition

topology_template:
  substitution_mappings:
    node_type: tosca.MyExampleNS
    properties:
      ..
      flavour_id: small
    requirements:
      virtual_link: [ VNF_1, virtual_link_2 ]

node_templates:
  MyExampleNS:
    type: tosca.MyExampleNS
    properties:
      #
    interfaces:
      Nslcm:
        instantiate:
          implementation: instantiate.workflow.yaml
        terminate:
          implementation: terminate.workflow.yaml

  VNF_1:
    type: tosca.nodes.nfv.example_VNF1
    properties:
      # no property assignments needed for required properties that have a default
      value assigned in the node type definition, e.g. descriptor_id
      flavour_id: simple
      vnf_profile:
        instantiation_level: level_1

```

```

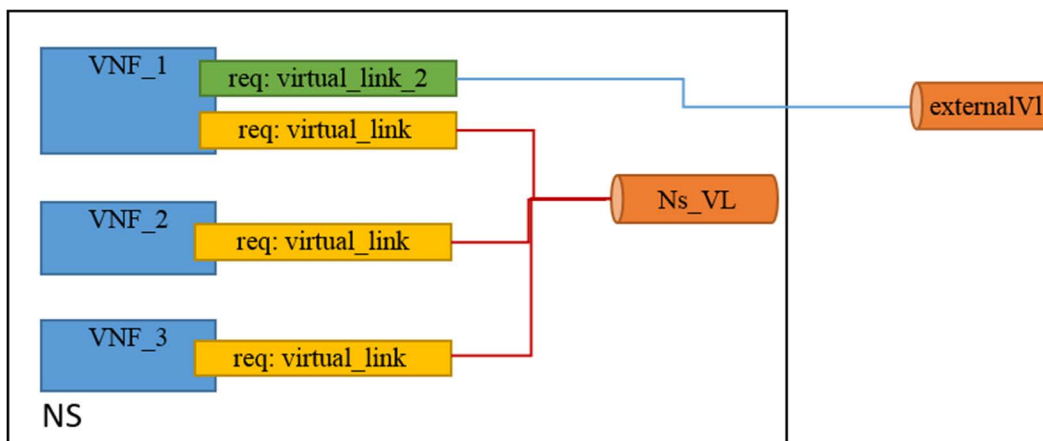
    min_number_of_instances: 2
    max_number_of_instances: 6
  requirements:
    - virtual_link: NsVirtualLink_1
    - virtual_link_2: # map to virtual_link requirement of the NS node

VNF_2:
  type: toasca.nodes.nfv.example_VNF2
  properties:
    flavour_id: simple
    vnf_profile:
      instantiation_level: level_1
      min_number_of_instances: 1
      max_number_of_instances: 3
  requirements:
    - virtual_link_1: NsVirtualLink_1

Ns_VL:
  type: toasca.nodes.nfv.NsVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ipv4]
      flow_pattern: mesh
    vl_profile:
      max_bitrate_requirements:
        root: 1000
      min_bitrate_requirements:
        root: 1000

```

**MyExampleNs (big):** Lower level, contains 3 VNFs and 1 NS virtual link.



**Figure A.11-3: MyExampleNs (big): Lower level**

#### MyExampleNs\_big.yaml

```

tosca_definitions_version: toasca_simple_yaml_1_2

description: myExampleNs with big flavour
imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of TOSCA NSD types as defined in NFV SOL 001
  GS
  - MyExampleNs_Type.yaml # contains the NS node type definition
  - example_VNF1_Type.yaml # contains the toasca.nodes.nfv.example_VNF1 node type
  definition

```

```

- example_VNF2_Type.yaml # contains the toasca.nodes.nfv.example_VNF2 node type
definition
- example_VNF3_Type.yaml # contains the toasca.nodes.nfv.example_VNF3 node type
definition

topology_template:
  substitution_mappings:
    node_type: toasca.MyExampleNS
  properties:
    ..
    flavour_id: small
  requirements:
    virtual_link: [ VNF_1, virtual_link_2 ]

node_templates:
  MyExampleNS:
    type: toasca.MyExampleNS
    properties:
      #
    interfaces:
      Nslcm:
        instantiate:
          implementation: instantiate.workflow.yaml
        terminate:
          implementation: terminate.workflow.yaml
        scale:
          implementation: scale.workbook.yaml

  VNF_1:
    type: toasca.nodes.nfv.example_VNF1
    properties:
      # no property assignments needed for required properties that have a default
value assigned in the node type definition, e.g. descriptor_id
      flavour_id: simple
      vnf_profile:
        instantiation_level: level_1
        min_number_of_instances: 2
        max_number_of_instances: 6
    requirements:
      - virtual_link: NsVirtualLink_1
      - virtual_link_2: # map to virtual_link requirement of the NS node

  VNF_2:
    type: toasca.nodes.nfv.example_VNF2
    properties:
      flavour_id: simple
      vnf_profile:
        instantiation_level: level_1
        min_number_of_instances: 1
        max_number_of_instances: 3
    requirements:
      - virtual_link_1: NsVirtualLink_1

  VNF_3:
    type: toasca.nodes.nfv.example_VNF3
    properties:
      flavour_id: simple
      vnf_profile:
        instantiation_level: level_1
        min_number_of_instances: 1
        max_number_of_instances: 3

```

```

requirements:
  - virtual_link_1: NsVirtualLink_1

Ns_VL:
  type: toasca.nodes.nfv.NsVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ipv4]
      flow_pattern: mesh
    vl_profile:
      max_bitrate_requirements:
        root: 1000
      min_bitrate_requirements:
        root: 1000

```

## A.12 NSD TOSCA service template design

A TOSCA service template representing a NSD may contain a node template of some specific NS node type as one of its constituents. The latter is a nested NS. When the containing NS is deployed, the node template of the nested NS is substituted by the topology template representing the nested NS.

Figure A.12-1 illustrates a network service NS<sub>1</sub> that consists of one VNF (VNF<sub>1</sub>), one NsVirtualLink (NS\_VL<sub>1</sub>) and one nested NS (NS<sub>2</sub>).

The nested NS consists of two VNFs (VNF<sub>3</sub> and VNF<sub>4</sub>) and one NsVirtualLink (NS\_VL<sub>2</sub>).

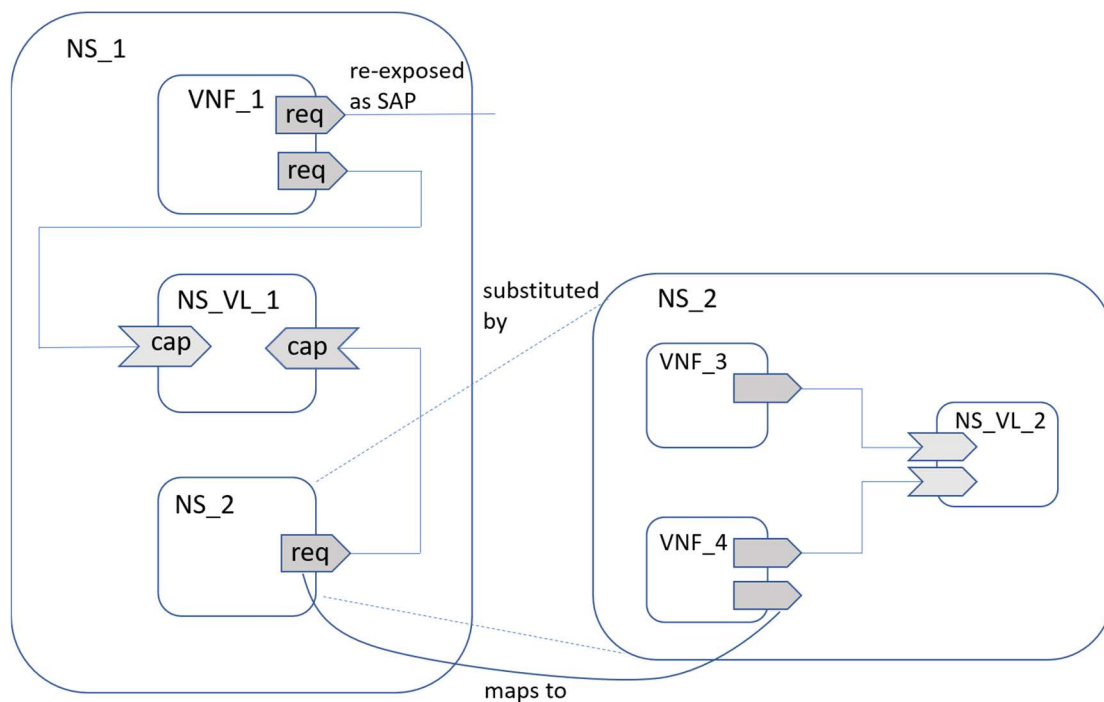


Figure A.12-1: Example of a network service containing a nested network service

The following snippet shows the topology template representing a particular flavour of NS\_1.

```

tosca_definitions_version: tosca_simple_yaml_1_2
description: myExampleNs with small flavour

imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of TOSCA NSD types as defined in NFV SOL 001
GS
  - MyExampleNs_Type.yaml # contains the NS node type definition
  - MyExampleNS_2.yaml

topology_template:
  substitution_mappings:
    node_type: tosca.MyExampleNS
  properties:
    ..
    flavour_id: small
  requirements:
    - virtual_link: [ VNF_1, virtual_link_2 ]

node_templates:
  NS_1:
    type: tosca.MyExampleNS
    interfaces:
      Nslcm:
        instantiate:
          implementation: instantiate.workflow.yaml
        terminate:
          implementation: terminate.workflow.yaml

  VNF_1:
    type: tosca.nodes.nfv.example_VNF1
    properties:
      # no property assignments needed for required properties that have a default
value assigned in the node type definition, e.g. descriptor_id
      flavour_id: simple
      vnf_profile:
        instantiation_level: level_1
        min_number_of_instances: 2
        max_number_of_instances: 6
    requirements:
      - virtual_link_1: NS_VL_1
      - virtual_link_2: # map to virtual_link requirement of the NS node

  NS_2:
    type: tosca.myExample.NS_2
    properties:
      descriptor_id: c1bb0ab8-deab-4fa7-95ed-4840d70a3574
      designer: MyCompany
      version: 1.0.0.0
      name: myExample2Service
      invariant_id: aaaa-bbbb-cccc-dddd
      ns_profile:
        ns_instantiation_level: level_1
        min_number_of_instances: 1
        max_number_of_instances: 3
        flavour_id: simple

  NS_VL_1:
    type: tosca.nodes.nfv.NsVirtualLink

```

```

properties:
  connectivity_type:
    layer_protocols: [ipv4]
    flow_pattern: mesh
  vl_profile:
    max_bitrate_requirements:
      root: 1000
    min_bitrate_requirements:
      root: 1000

```

The contents of MyExampleNs\_Type.yaml file with the node type definition are as follows:

```

tosca_definitions_version: tosca_simple_yaml_1_2

description: type definition of tosca.MyExampleNS

imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of TOSCA types as defined in NFV SOL 001 GS

node_types:
  tosca.MyExampleNS:
    derived_from: tosca.nodes.nfv.NS
    properties:
      descriptor_id:
        type: string
        constraints: [ valid_values: [ b1bb0ce7-ebca-4fa7-95ed-4840d70a1177 ] ]
        default: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      designer:
        type: string
        constraints: [ valid_values: [ MyCompany ] ]
        default: MyCompany
      name:
        type: string
        constraints: [ valid_values: [ ExampleService ] ]
        default: ExampleService
      version:
        type: string
        constraints: [ valid_values: [ '1.0' ] ]
        default: '1.0'
      invariant_id:
        type: string
        constraints: [ valid_values: [ 1111-2222-aaaa-bbbb ] ]
        default: 1111-2222-aaaa-bbbb
      flavour_id:
        type: string
        constraints: [ valid_values: [ small, big ] ]
        default: small
    requirements:
      - virtual_link:
          capability: tosca.capabilities.nfv.VirtualLinkable
    interfaces:
      Nslcm:
        type: tosca.interfaces.nfv.Nslcm

```

The following snippet shows the service template representing the NSD NS\_2. In this example, NS\_2 supports one single deployment flavour.

**MyExampleNS\_2.yaml:**

```

tosca_definitions_version: tosca_simple_yaml_1_1

description: Relational database, simple

imports:
  - etsi_nfv_sol001_nsd_types.yaml # all of NSD related TOSCA types as defined in NFV
    SOL 001 GS

node_types:
  tosca.myExample.NS_2:
    derived_from: tosca.nodes.nfv.NS
    properties:
      descriptor_id:
        type: string
        constraints: [ valid_values: [ c1bb0ab8-deab-4fa7-95ed-4840d70a3574 ] ]
        default: c1bb0ab8-deab-4fa7-95ed-4840d70a3574
      designer:
        type: string
        constraints: [ valid_values: [ MyCompany ] ]
        default: MyCompany
      name:
        type: string
        constraints: [ valid_values: [ myExample2Service ] ]
        default: myExample2Service
      version:
        type: string
        constraints: [ valid_values: [ '1.0.0.0' ] ]
        default: '1.0.0.0'
      invariant_id:
        type: string
        constraints: [ valid_values: [ aaaa-bbbb-cccc-dddd ] ]
        default: aaaa-bbbb-cccc-dddd
      flavour_id:
        type: string
        constraints: [ valid_values: [ simple ] ]
        default: simple

topology_template:
  substitution_mappings:
    node_type: tosca.myExample.NS_2
  requirements:
    - virtual_link: [ VNF_4, virtual_link_2 ] # the External connection point of
      # VNF_2 is exposed as the Sap

node_templates:
  NS_2:
    type: tosca.myExample.NS_2
    interfaces:
      Nslcm:
        instantiate:
          implementation: instantiate.workflow.yaml
        terminate:
          implementation: terminate.workflow.yaml

  VNF_3:
    type: tosca.nodes.nfv.example_VNF3
    properties:
      # no property assignments needed for required properties that have a default
      value assigned in the node type definition, e.g. descriptor_id

```



```
    flavour_id: simple
    vnf_profile:
      instantiation_level: level_1
      min_number_of_instances: 2
      max_number_of_instances: 6
  requirements:
    - virtual_link: NS_VL_2

VNF_4:
  type: toasca.nodes.nfv.example_VNF4
  properties:
    flavour_id: simple
    vnf_profile:
      instantiation_level: level_1
      min_number_of_instances: 1
      max_number_of_instances: 3
  requirements:
    - virtual_link_1: NS_VL_2
    - virtual_link_2: # map to virtual_link requirement of the NS node

NS_VL_2:
  type: toasca.nodes.nfv.NsVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ipv4]
      flow_pattern: mesh
    vl_profile:
      max_bitrate_requirements:
        root: 1000
      min_bitrate_requirements:
        root: 1000
```

---

## Annex B (normative): etsi\_nfv\_sol001\_type definitions

### B.1 Purpose

All type definitions specified in clauses 6, 7 and 8 of the present document are gathered in three definition files.

The file names are structured as follows:

- etsi\_nfv\_sol001\_common\_types.yaml, for the common type definitions which are used by at least two types of deployment templates among those identified in clause 5.1, the content of this yaml file is described in B.5;
- etsi\_nfv\_sol001\_vnfd\_types.yaml, for the definitions provided in clause 6 and only used in a VNFD service template design;
- etsi\_nfv\_sol001\_nsd\_types.yaml, for the definitions provided in clause 7 and only used in a NSD service template design;
- etsi\_nfv\_sol001\_pnfd\_types.yaml, for the definitions provided in clause 8 and only used in a PNFD service template design.

---

### B.2 VNFD type definitions file

The file etsi\_nfv\_sol001\_vnfd\_types.yaml, attached to the present document, contains all type definitions specified in clause 6 of the present document. This file is also available at the following URL:

[https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi\\_nfv\\_sol001\\_vnfd\\_types.yaml](https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi_nfv_sol001_vnfd_types.yaml)

NOTE 1: The file etsi\_nfv\_sol001\_vnfd\_types.yaml includes a TOSCA import definition referencing etsi\_nfv\_sol001\_common\_types.yaml file. If the later file is included in the VNF package, the import definition can reference the local file using appropriate path in the VNF package.

This file is a TOSCA service template that only contains definitions. The template\_version in the metadata section within this template is structured as x.y.z, where x, y and z represent the version of this file and are set respectively to "2", "6" and "1" for this version of the present document. In subsequent versions of the present document, "x", "y" and "z" in the template\_version will be incremented only if there are changes in the VNFD type definitions.

A TOSCA service template representing a VNFD complying with the present document shall contain import statement referencing this file, as defined in clause 5.6.1.

NOTE 2: This file may, but need not, be included in the VNF Package.

---

### B.3 NSD type definitions file

The file etsi\_nfv\_sol001\_nsd\_types.yaml, attached to the present document, contains all type definitions specified in clause 7 of the present document. This file is also available at the following URL:

[https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi\\_nfv\\_sol001\\_nsd\\_types.yaml](https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi_nfv_sol001_nsd_types.yaml)

NOTE 1: The file etsi\_nfv\_sol001\_nsd\_types.yaml includes a TOSCA import definition referencing etsi\_nfv\_sol001\_common\_types.yaml file. If the later file is included in the NSD file archive, the import definition can reference the local file using appropriate path in the NSD file archive.

This file is a TOSCA service template that only contains definitions. The template\_version in the metadata section within this template is structured as x.y.z, where x, y and z represent the version of this file and are set respectively to "2", "6" and "1" for this version of the present document. In subsequent versions of the present document, "x", "y" and "z" in the template\_version will be incremented only if there are changes in the NSD type definitions.

A TOSCA service template representing a NSD complying with the present document shall contain import statement referencing this file, as defined in clause 5.6.2.

NOTE 2: This file may, but need not, be included in the NSD file archive.

---

## B.4 PNFD type definitions file

The file `etsi_nfv_sol001_pnfd_types.yaml`, attached to the present document, contains all type definitions specified in clause 8 of the present document. This file is also available at the following URL:

[https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi\\_nfv\\_sol001\\_pnfd\\_types.yaml](https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi_nfv_sol001_pnfd_types.yaml)

NOTE: The file `etsi_nfv_sol001_pnfd_types.yaml` includes a TOSCA import definition referencing `etsi_nfv_sol001_common_types.yaml` file.

This file is a TOSCA service template that only contains definitions. The `template_version` in the metadata section within this template is structured as `x.y.z`, where `x`, `y` and `z` represent the version of this file and are set respectively to "2", "6" and "1" for this version of the present document. In subsequent versions of the present document, "x", "y" and "z" in the `template_version` will be incremented only if there are changes in the PNFD type definitions.

A TOSCA service template representing a PNFD complying with the present document shall contain import statement referencing this file, as defined in clause 5.6.3.

---

## B.5 Common type definitions file

The file `etsi_nfv_sol001_common_types.yaml`, attached to the present document, contains the type definitions as specified in clause 6 and used by at least two types of deployment templates. This file is also available at the following URL:

[https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi\\_nfv\\_sol001\\_common\\_types.yaml](https://forge.etsi.org/rep/nfv/SOL001/raw/v2.6.1/etsi_nfv_sol001_common_types.yaml)

NOTE 1: This file includes the following type definitions:

- `tosca.datatypes.nfv.L2AddressData`
- `tosca.datatypes.nfv.L3AddressData`
- `tosca.datatypes.nfv.AddressData`
- `tosca.datatypes.nfv.ConnectivityType`
- `tosca.datatypes.nfv.LinkBitrateRequirements`
- `tosca.datatypes.nfv.CpProtocolData`
- `tosca.datatypes.nfv.VnfProfile`
- `tosca.datatypes.nfv.Qos`
- `tosca.capabilities.nfv.VirtualLinkable`
- `tosca.relationships.nfv.VirtualLinksTo`
- `tosca.nodes.nfv.Cp`

This file is a TOSCA service template that only contains definitions. The `template_version` in the metadata section within this template is structured as `x.y.z`, where `x`, `y` and `z` represent the version of this file and are set respectively to "2", "6" and "1" for this version of the present document. In subsequent versions of the present document, "x", "y" and "z" in the `template_version` will be incremented only if there are changes in the common type definitions.

NOTE 2: This file may, but need not, be included in the VNF package or NSD file archive.

## Annex C (normative): Conformance

### C.1 Purpose

The present document specifies a data model for the VNFD, the NSD and the PNFD, by using the grammar defined in the TOSCA Simple Profile in YAML specification [3]. This annex specifies the requirements to be fulfilled for claiming conformance to the present document.

### C.2 NFV TOSCA YAML service template

A VNFD, an NSD or a PNFD conforms to the present document if it complies with all the requirements below:

- 1) A VNFD conformant to the present document shall comply with the requirements in clause 6 of the present document and to the specification of the elements of the TOSCA Simple Profile in YAML specification [3] it uses, unless otherwise stated in clause 6 of the present document.
- 2) A NSD conformant to the present document shall comply with the requirements in clause 7 of the present document and to the specification of the elements of the TOSCA Simple Profile in YAML specification [3] it uses, unless otherwise stated in clause 7 of the present document.
- 3) A PNFD conformant to the present document shall comply with the requirements in clause 8 of the present document and to the specification of the elements of the TOSCA Simple Profile in YAML specification [3] it uses, unless otherwise stated in clause 8 of the present document.
- 4) When using or referring to the TOSCA normative types listed in table C.2-1, it is valid according to the definitions given in clauses 6 and 7 of the present document and to section 5 of the TOSCA Simple Profile in YAML specification [3].

**Table C.2-1: TOSCA normative types used in the present document**

Types	VNFD	NSD
tosca.datatypes.Root	X	X
tosca.artifacts.File	X	
tosca.artifacts.Deployment.Image	X	
tosca.artifacts.Implementation	X	
tosca.capabilities.Root	X	
tosca.relationships.Root	X	
tosca.relationships.DependsOn	X	X
tosca.interfaces.Root	X	X
tosca.nodes.Root	X	X
tosca.groups.Root	X	X
tosca.policies.Root	X	X
tosca.policies.Placement	X	X

- 5) A VNFD conformant to the present document shall comply with VNFD TOSCA service template design specified in clause 6.11 of the present document.
- 6) A VNFD and NSD conformant to the present document shall comply with rules for Type extension defined in clause 5.7 of the present document.

---

## C.3 NFV TOSCA processor

A processor or program conforms to the present document as NFV TOSCA processor for VNFD, NFV TOSCA processor for NSD or NFV TOSCA processor for PNFD if it complies with all the requirements below:

- 1) It can parse and recognize the elements of any VNFD, NSD or PNFD that conform to the present document, and shall generate errors for those documents that fail to conform to the present document.
- 2) It shall comply with all requirements and implement the semantics associated with the definitions specified in clauses 6, 7 and 8 of the present document.
- 3) It shall resolve the import definitions, either explicit or implicit, as described in clause 5.5 of the present document.

## Annex D (informative): Mapping between properties of TOSCA types and API attributes

### D.1 Introduction

This annex provides the mapping between properties of TOSCA types defined in the present document and defined in the following API specifications: ETSI GS NFV-SOL 002 [i.4], ETSI GS NFV-SOL 003 [i.9], and ETSI GS NFV-SOL 005 [i.10].

NOTE: See also annex A "Mapping operations to protocol elements" of ETSI GS NFV-SOL 002 [i.4], ETSI GS NFV-SOL 003 [i.9] and ETSI GS NFV-SOL 005 [i.10] for each operation.

### D.2 VNFD-related constructs

Table D.2-1 provides the mapping between the properties of TOSCA types related to the VNFD and API attributes, which include: resource or notification data type (and referenced structured data type when available), attribute name and type in the resource or notification data type, and the interface operation in which the data type is used.

NOTE: In the "Data model" column of the table, an arrow "->" indicates the navigation through the resource, notification and referenced structured data types.

**Table D.2-1: Mapping of API attributes and TOSCA constructs**

ETSI GS NFV-SOL 001	SOL APIs		
Type and Property or entity name	Type and attribute name	Data model	Operation (see note)
tosca.nodes.nfv.VNF -> descriptor_id	(Identifier) vnfdId	VnfInstance	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		VnfInfoModificationRequest	Modify VNF Information (see SOL002 and SOL003)
		VnfInfoModifications	
		VnfLcmOpOcc -> VnfInfoModifications	Get Operation Status (see SOL002 and SOL003)
		VnfLcmOperationOccurrenceNoti fication -> VnfInfoModifications	Notify about VNF LCM (see SOL002 and SOL003)
		VnfPkgInfo	Create VNF package (see SOL005) Query VNF Package Info (see SOL005)
		VnfPackageOnboardingNotificati on	Notify about VNF Package (see SOL003 and SOL005)
		VnfPackageChangeNotification	Notify about VNF Package (see SOL003 and SOL005)
		NsInstance -> VnfInstance	Query NS (see SOL005)
		UpdateNsRequest -> InstantiateVnfData	Update NS (see SOL005)
		NsLcmOperationOccurrenceNotifi cation -> AffectedVnf	Notify about NS LCM (see SOL005)
		PkgmSubscriptionRequest -> PkgmNotificationsFilter	Subscription about VNF Package (see SOL003 and SOL005)

ETSI GS NFV-SOL 001	SOL APIs		
Type and Property or entity name	Type and attribute name	Data model	Operation (see note)
		PkgmSubscription -> PkgmNotificationsFilter	Subscription about VNF Package (see SOL003 and SOL005)
tosca.nodes.nfv.VNF -> provider	(String) vnfProvider	VnflInstance	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		VnfPkgInfo	Create VNF package (see SOL005) Query VNF Package Info (see SOL005)
		NsInstance -> VnflInstance	Query NS (see SOL005)
		PkgmSubscriptionRequest -> PkgmNotificationsFilter -> vnfProductsFromProviders	Subscription about VNF Package (see SOL003 and SOL005)
		PkgmSubscription -> PkgmNotificationsFilter -> vnfProductsFromProviders	Subscription about VNF Package (see SOL003 and SOL005)
tosca.nodes.nfv.VNF -> product_name	(String) vnfProductName	VnflInstance	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		VnfPkgInfo	Create VNF package (see SOL005) Query VNF Package Info (see SOL003 and SOL005)
		NsInstance -> VnflInstance	Query NS (see SOL005)
		PkgmSubscriptionRequest -> PkgmNotificationsFilter -> vnfProductsFromProviders -> vnfProducts	Subscription about VNF Package (see SOL003 and SOL005)
		PkgmSubscription -> PkgmNotificationsFilter -> vnfProductsFromProviders -> vnfProducts	Subscription about VNF Package (see SOL003 and SOL005)
tosca.nodes.nfv.VNF -> software_version	(Version) vnfSoftwareVersion	VnflInstance	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		VnfPkgInfo	Create VNF package (see SOL005) Query VNF Package Info (see SOL003 and SOL005)
		NsInstance -> VnflInstance	Query NS (see SOL005)
		PkgmSubscriptionRequest -> PkgmNotificationsFilter -> vnfProductsFromProviders -> vnfProducts -> versions	Subscription about VNF Package (see SOL003 and SOL005)
		PkgmSubscription -> PkgmNotificationsFilter -> vnfProductsFromProviders -> vnfProducts -> versions	Subscription about VNF Package (see SOL003 and SOL005)
tosca.nodes.nfv.VNF -> descriptor_version	(Version) vnfdVersion	VnflInstance	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		VnfInfoModifications	Modify VNF Information (see SOL002 and SOL003)

ETSI GS NFV-SOL 001	SOL APIs				
Type and Property or entity name	Type and attribute name	Data model	Operation (see note)		
		VnfLcmOpOcc -> VnfInfoModifications	Get Operation Status (see SOL002 and SOL003)		
		VnfLcmOperationOccurrenceNoti fication -> VnfInfoModifications	Notify about VNF LCM (see SOL002 and SOL003)		
		VnfPkgInfo	Create VNF package (see SOL005) Query VNF Package Info (see SOL003 and SOL005)		
		NsInstance -> VnfInstance	Query NS (see SOL005)		
		PkgmSubscriptionRequest -> PkgmNotificationsFilter -> vnfProductsFromProviders -> vnfProducts -> versions	Subscription about VNF Package (see SOL003 and SOL005)		
		PkgmSubscription -> PkgmNotificationsFilter -> vnfProductsFromProviders -> vnfProducts -> versions	Subscription about VNF Package (see SOL003 and SOL005)		
		tosca.nodes.nfv.VNF -> configurable_properties	(KeyValuePairs) vnfConfigurableProperti es	VnfInstance	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		NsInstance -> VnfInstance	Query NS (see SOL005)		
		VnfInfoModificationRequest VnfInfoModifications	Modify VNF Information (see SOL002 and SOL003)		
		VnfLcmOpOcc -> VnfInfoModifications	Get Operation Status (see SOL002 and SOL003)		
		VnfLcmOperationOccurrenceNoti fication -> VnfInfoModifications	Notify about VNF LCM (see SOL002 and SOL003)		
		tosca.nodes.nfv.VNF ->flavour_id	(IdentifierInVnfd) flavourId	VnfInstance	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		InstantiateVnfRequest	Instantiate VNF (see SOL002 and SOL003)		
		NsInstance -> VnfInstance -> instantiatedVnfInfo	Query NS (see SOL005)		
		(IdentifierInVnfd) vnfFlavourId	Update NS (see SOL005)		
		(IdentifierInVnfd) newFlavourId	Change VNF Flavour (see SOL002 and SOL003)		
		UpdateNsRequest -> ChangeVnfFlavourData	Update NS (see SOL005)		
tosca.datatypes.nfv.ScalingAspect ->name	(IdentifierInVnfd) aspectId	VnfInstance -> instantiatedVnfInfo -> ScaleInfo	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)		
		ScaleVnfRequest	Scale VNF (see SOL002 and SOL003)		
		ScaleVnfToLevelRequest -> ScaleInfo	Scale VNF to level (see SOL002 and SOL003)		
		NsInstance -> VnfInstance -> instantiatedVnfInfo -> ScaleInfo	Query NS (see SOL005)		



ETSI GS NFV-SOL 001	SOL APIs		
Type and Property or entity name	Type and attribute name	Data model	Operation (see note)
tosca.datatypes.nfv.ScaleInfo ->scale_level	(Integer) scaleLevel	VnfInstance -> instantiatedVnfInfo -> ScaleInfo	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		ScaleVnfToLevelRequest -> ScaleInfo	Scale VNF to level (see SOL002 and SOL003)
		NsInstance -> VnfInstance -> instantiatedVnfInfo -> ScaleInfo	Query NS (see SOL005)
Node template name of type tosca.nodes.nfv.VnfExtCp	(IdentifierInVnfd) cpId	VnfInstance -> instantiatedVnfInfo -> VnfExtCpInfo	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		ChangeVnfFlavourRequest -> ExtVirtualLinkData -> VnfExtCpData	Change VNF Flavour (see SOL002 and SOL003)
		ChangeExtVnfConnectivityRequest -> ExtVirtualLinkData -> VnfExtCpData	Change External VNF Connectivity (see SOL002 and SOL003)
		NsInstance -> VnfInstance -> instantiatedVnfInfo -> VnfExtCpInfo	Query NS (see SOL005)
	(IdentifierInVnfd) resourceTemplateId	GrantRequest -> ResourceDefinition	Grant Lifecycle Operation (see SOL003)
Node template name of type tosca.nodes.nfv.VnfVirtualLink	(IdentifierInVnfd) vnfVirtualLinkId	VnfInstance -> instantiatedVnfInfo -> ExtManagedVirtualLinkInfo	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		InstantiateVnfRequest -> ExtManagedVirtualLinkData	Instantiate VNF (see SOL002 and SOL003)
		ChangeVnfFlavourRequest -> ExtManagedVirtualLinkData	Change VNF Flavour (see SOL002 and SOL003)
		VnfLcmOpOcc -> resourceChanges -> AffectedVirtualLink	Get Operation Status (see SOL002 and SOL003)
		NsInstance -> VnfInstance -> instantiatedVnfInfo -> ExtManagedVirtualLinkInfo	Query NS (see SOL005)
	(IdentifierInVnfd) vnfVirtualLinkId	UpdateNsRequest -> InstantiateVnfData -> ExtManagedVirtualLinkData	Update NS (see SOL005)
TBD tosca.datatypes.nfv.VnfMonitoringParameter	(IdentifierInVnfd) id	VnfInstance -> instantiatedVnfInfo -> MonitoringParameter	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		NsInstance -> VnfInstance -> instantiatedVnfInfo -> MonitoringParameter	Query NS (see SOL005)
tosca.datatypes.nfv.VnfMonitoringParameter ->name	(String) name	VnfInstance -> instantiatedVnfInfo -> MonitoringParameter	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		NsInstance -> VnfInstance -> instantiatedVnfInfo -> MonitoringParameter	Query NS (see SOL005)

ETSI GS NFV-SOL 001	SOL APIs		
Type and Property or entity name	Type and attribute name	Data model	Operation (see note)
tosca.nodes.nfv.VNF ->localization_languages	(String) localizationLanguage	VnflInstance -> instantiatedVnflInfo	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		NsInstance -> VnflInstance -> instantiatedVnflInfo	Query NS (see SOL005)
Node template name of type tosca.nodes.nfv.Vdu.Compute	(IdentifierInVnfd)vduId	VnflInstance -> instantiatedVnflInfo -> VnfcResourceInfo	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		VnflInstance -> instantiatedVnflInfo -> VnfcInfo	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		VnflcmOpOcc -> resourceChanges -> AffectedVnfc	Get Operation Status (see SOL002 and SOL003)
		VnflcmOperationOccurrenceNotification -> AffectedVnfc	Notify about VNF LCM (see SOL002 and SOL003)
		NsInstance -> VnflInstance -> instantiatedVnflInfo -> VnfcResourceInfo	Query NS (see SOL005)
		(IdentifierInVnfd) vnfdVirtualComputeDescId	Grant -> vimAssets -> VimComputeResourceFlavour
Node template name of type tosca.nodes.nfv.VduCp	(IdentifierInVnfd) cpId	VnflInstance -> instantiatedVnflInfo -> VnfcResourceInfo -> vnfcCplInfo	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
Node template name of type tosca.nodes.nfv.Vdu.VirtualBlockStorage	(IdentifierInVnfd) virtualStorageDescId	VnflInstance -> instantiatedVnflInfo -> VirtualStorageResourceInfo	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		VnflcmOpOcc -> resourceChanges -> AffectedVirtualStorage	Get Operation Status (see SOL002 and SOL003)
		VnflcmOperationOccurrenceNotification -> AffectedVirtualStorage	Notify about VNF LCM (see SOL002 and SOL003)
Node template name of type tosca.nodes.nfv.Vdu.VirtualFileStorage		NsInstance -> VnflInstance -> instantiatedVnflInfo -> VirtualStorageResourceInfo	Query NS (see SOL005)
tosca.datatypes.nfv.VnflInfoModifiableAttributes ->metadata	(KeyValuePairs) metadata	VnflInstance	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		NsInstance -> VnflInstance	Query NS (see SOL005)
		CreateVnflRequest	Create VNF Identifier (see SOL002 and SOL003)
		VnflInfoModificationRequest	Modify VNF Information (see SOL002 and SOL003)
		VnflInfoModifications	Modify VNF Information (see SOL002 and SOL003)
		VnflcmOpOcc -> VnflInfoModifications	Get Operation Status (see SOL002 and SOL003)
		VnflcmOperationOccurrenceNotification -> VnflInfoModifications	Notify about VNF LCM (see SOL002 and SOL003)

ETSI GS NFV-SOL 001	SOL APIs		
	Type and Property or entity name	Type and attribute name	Operation (see note)
tosca.datatypes.nfv.VnfInfoModifiableAttributes ->extensions	(KeyValuePairs) extensions	VnfInstance	Create VNF Identifier (see SOL002 and SOL003) Query VNF (see SOL002 and SOL003)
		NsInstance -> VnfInstance	Query NS (see SOL005)
		InstantiateVnfRequest	Instantiate VNF (see SOL002 and SOL003)
		VnfInfoModificationRequest	Modify VNF Information (see SOL002 and SOL003)
		VnfLcmOpOcc -> VnfInfoModifications	Get Operation Status (see SOL002 and SOL003)
		VnfLcmOperationOccurrenceNotification -> VnfInfoModifications	Notify about VNF LCM (see SOL002 and SOL003)
		tosca.policies.nfv.InstantiationLevel ->levels[key]	(IdentifierInVnfd) instantiationLevelId
ScaleVnfToLevelRequest	Scale VNF to level (see SOL002 and SOL003)		
ChangeVnfFlavourRequest	Change VNF Flavour (see SOL002 and SOL003)		
UpdateNsRequest -> changeVnfFlavourData	Update NS (see SOL005)		
(IdentifierInVnfd) vnfInstantiationLevelId	UpdateNsRequest -> InstantiateVnfData		Update NS (see SOL005)
	ScaleNsRequest -> ScaleVnfData		Scale NS (see SOL005)
	ScaleToLevelData		
tosca.datatypes.nfv.VnfLcmOperationsConfiguration -> instantiate	(KeyValuePairs) additionalParams	InstantiateVnfRequest	Instantiate VNF (see SOL002 and SOL003)
TBD tosca.datatypes.nfv.VnfLcmOperationsConfiguration -> scale	(KeyValuePairs) additionalParams	ScaleVnfRequest	Scale VNF (see SOL002 and SOL003)
TBD tosca.datatypes.nfv.VnfLcmOperationsConfiguration -> scale_to_level	(KeyValuePairs) additionalParams	ScaleVnfToLevelRequest	Scale VNF to level (see [i.4] and [i.9])!
tosca.datatypes.nfv.VnfLcmOperationsConfiguration -> change_flavour	(KeyValuePairs) additionalParams	ChangeVnfFlavourRequest	Change VNF Flavour (see SOL002 and SOL003)
TBD tosca.datatypes.nfv.VnfLcmOperationsConfiguration -> operate	(KeyValuePairs) additionalParams	OperateVnfRequest	Operate VNF (see SOL002 and SOL003)
TBD tosca.datatypes.nfv.VnfLcmOperationsConfiguration -> heal	(KeyValuePairs) additionalParams	HealVnfRequest	Heal VNF (see SOL002 and SOL003)
tosca.datatypes.nfv.VnfLcmOperationsConfiguration -> change_ext_connectivity	(KeyValuePairs) additionalParams	ChangeExtVnfConnectivityRequest	Change External VNF Connectivity (see SOL002 and SOL003)
TBD tosca.datatypes.nfv.VnfLcmOperationsConfiguration -> terminate	(KeyValuePairs) additionalParams	TerminateVnfRequest	Terminate VNF (see SOL002 and SOL003)
TBD tosca.artifacts.nfv.SwImage	(IdentifierInVnfd) id	VnfPkgInfo -> VnfPackageSoftwareImageInfo	Create VNF Package Info (see SOL005) Query VNF Package Info (see SOL005)

ETSI GS NFV-SOL 001	SOL APIs		
Type and Property or entity name	Type and attribute name	Data model	Operation (see note)
Corresponding property for this attribute is not supported in the present document of SOL001.	(IdentifierInVnfd) Id	VnflIndicator	Get indicator value (see SOL002 and SOL003)
Corresponding property for this attribute is not supported in the present document of SOL001.	(String) Name	VnflIndicator	Get indicator value (see SOL002 and SOL003)
Corresponding property for this attribute is not supported in the present document of SOL001.	(Object) value	VnflIndicator	Get indicator value (see SOL002 and SOL003)
TBD	(IdentifierInVnfd) vnfdSoftwareImageld	Grant -> vimAssets -> VimSoftwareImage	Grant Lifecycle Operation (see SOL003)
NOTE: The entry "SOL002" in the "Operation" column refers to ETSI GS NFV-SOL 002 [i.4], "SOL003" refers to ETSI GS NFV-SOL 003 [i.9] and "SOL005" refers to ETSI GS NFV-SOL 005 [i.10].			

---

## Annex E (informative): Authors & contributors

The following people have contributed to the present document:

**Rapporteur:**

Shitao Li, Huawei

**Other contributors:**

Shitao Li, Huawei

Thinh Nguyenphu, Nokia

Andrei Kojukhov, Amdocs

Arturo Martin De Nicolas, Ericsson

Yuya Kuno, NTTDOCOMO

Gabor Marton, Nokia

Manchang Ju, ZTE Corporation

Bruno Chatras, Orange

Raquel Morera, Verizon

Satish Vasamsetti, Verizon Wireless

Philip Joseph, RIFT.io

Alex Vul, Intel

Priya T G, Netcracker Technologies Ltd

Li xiang, China Mobile US Research Center

Ren rongwei, China Mobile US Research Center

Yuan yannan, China Mobile US Research Center

Uwe Rauschenbach, Nokia Germany

Nagesha Subramanya, Nokia Germany

kaliyaperumal Ramalingam, Verizon UK Ltd

Abhishek Kumar, Verizon UK Ltd

Philip Joseph, Riftio

Wangyaoguang, Huawei

Lijuan Chen, ZTE Corporation

## Annex F (informative): Change History

Date	Version	Information about changes
2016.05	0.0.1	Implemented NFVSOL(16)00005r1_GS_NFV_SOL001_ToC
2016.07	0.02	Implemented NFVSOL(16)000026r1, NFVSOL(16)000027r1, NFVSOL(16)000028r2
2017.09	0.1.0	Implemented NFVSOL(17)000539r3, NFVSOL(17)000540r3, NFVSOL(17)000542r2, NFVSOL(17)000544r1
2017.10	0.2.0	Implemented NFVSOL(17)000543r3, NFVSOL(17)000566, NFVSOL(17)000567r1, NFVSOL(17)000568r1, NFVSOL(17)000569 Editorial modification for clause numbering and format.
2017.11	0.3.0	Clean-up done by <i>editHelp</i> Implemented NFVSOL(17)000545r4, NFVSOL(17)000559r2, NFVSOL(17)000560r2, NFVSOL(17)000570r1, NFVSOL(17)000575r1, NFVSOL(17)000616r1, NFVSOL(17)000641
2017.12	0.4.0	Implemented NFVSOL(17)000642r1, NFVSOL(17)000675r1, NFVSOL(17)000756
2018.01	0.5.0	Implemented NFVSOL(17)000621r6, NFVSOL(17)000676r3, NFVSOL(17)000677r2, NFVSOL(17)000736r1, NFVSOL(18)000004r2
2018.03	0.6.0	Implemented NFVSOL(18)000049, NFVSOL(18)000025r1, NFVSOL(18)00048R1, NFVSOL(18)00040, NFVSOL(18)00023r2, NFVSOL(18)00024r2, NFVSOL(18)00029r1, NFVSOL(18)00063r5, NFVSOL(18)000077, NFVSOL(18)000046r2, NFVSOL(18)000074r2, NFVSOL(17)0000611r5, NFVSOL(18)000041r1, NFVSOL(18)000038r3, NFVSOL(18)000028, NFVSOL(18)000044r2, NFVSOL(18)000055r2, NFVSOL(18)000045r1
2018.03	0.6.1	Implemented NFVSOL(18)000052r1, NFVSOL(18)000094r5, NFVSOL(18)000043r3, NFVSOL(18)0000112r1, NFVSOL(18)0000117r2, NFVSOL(18)0000124r1, NFVSOL(18)0000129r1, NFVSOL(18)000042r2, NFVSOL(18)0000115r2
2018.04	0.6.2	Implemented NFVSOL(18)000113r2, NFVSOL(18)000119r2, NFVSOL(18)000121, NFVSOL(18)000135, NFVSOL(18)000116r1, NFVSOL(18)000157, NFVSOL(18)000158r1
2018.05	0.6.3	Implemented NFVSOL(18)000168, NFVSOL(18)000169r2, NFVSOL(18)000173, NFVSOL(18)000174r2
2018.05	0.7.0	Implemented NFVSOL(18)000156r7, NFVSOL(18)000142r3, NFVSOL(18)000147, NFVSOL(18)000193r2, NFVSOL(18)000201, NFVSOL(18)000202, NFVSOL(18)000203r2, NFVSOL(18)000205r1, NFVSOL(18)000160r1, NFVSOL(18)000199r2, NFVSOL(18)000200r3, NFVSOL(18)000192r1, NFVSOL(18)000194r2, NFVSOL(18)000231r1, NFVSOL(18)000223r1, NFVSOL(18)000183r2
2018.06	0.8.0	Implemented NFVSOL(18)000287, NFVSOL(18)00012r11, NFVSOL(18)000197r2, NFVSOL(18)000198r3, NFVSOL(18)000286, NFVSOL(18)000292, NFVSOL(18)000294r1, NFVSOL(18)000295r1, NFVSOL(18)000301r2, NFVSOL(18)000302r2, NFVSOL(18)000240r1, NFVSOL(18)000253r3, NFVSOL(18)000254r2, NFVSOL(18)000256r2,
2018.06	0.9.0	Implemented NFVSOL(18)000288r2 Adding etsi_nfv_sol001_vnfd_0_9_0_type.yaml and SOL001 Graphics v0_9_0.pptx in the draft GS zip package Editorial changes for all the TOSCA type definitions.
2018.08	0.10.0	Implemented NFVSOL(18)000331r4, NFVSOL(18)000335r2, NFVSOL(18)000351r1, NFVSOL(18)000374r1, NFVSOL(18)000376, NFVSOL(18)000382, NFVSOL(18)000395r1, NFVSOL(18)000402r2, NFVSOL(18)000404, NFVSOL(18)000405r2, NFVSOL(18)000406r1, NFVSOL(18)000408r2, NFVSOL(18)000409r1, NFVSOL(18)000411r1, NFVSOL(18)000413, NFVSOL(18)000416r2, NFVSOL(18)000422r4, NFVSOL(18)000423r4, NFVSOL(18)000424r2, NFVSOL(18)000425r2, NFVSOL(18)000430r1, NFVSOL(18)000336r4, NFVSOL(18)000380, NFVSOL(18)000387r2, NFVSOL(18)000394r1, NFVSOL(18)000410r2, NFVSOL(18)000447r3, NFVSOL(18)000427, NFVSOL(18)000403r3, NFVSOL(18)000420r4, NFVSOL(18)000393r1, NFVSOL(18)000415, NFVSOL(18)000398r1, NFVSOL(18)000399r3, NFVSOL(18)000414r2, NFVSOL(18)000379r3, NFVSOL(18)000428r5, NFVSOL(18)000429r4, NFVSOL(18)000465r1, NFVSOL(18)000479, NFVSOL(18)000375r4, NFVSOL(18)000488r2, NFVSOL(18)000492r1

Date	Version	Information about changes
2018.09	0.11.0	Implemented NFVSOL(18)000486r7, NFVSOL(18)000495r1, NFVSOL(18)000497r3, NFVSOL(18)000498, NFVSOL(18)000500r3, NFVSOL(18)000503r1, NFVSOL(18)000504r1, NFVSOL(18)000505r1, NFVSOL(18)000508, NFVSOL(18)000514r2, NFVSOL(18)000515r1, NFVSOL(18)000516r1, NFVSOL(18)000524r1, NFVSOL(18)000529r2, NFVSOL(18)000530r1, NFVSOL(18)000536r1, NFVSOL(18)000538r1, NFVSOL(18)000541r3, NFVSOL(18)000544r1, NFVSOL(18)000545r1,
2018.10	0.12.0	Implemented NFVSOL(18)000507r3, NFVSOL(18)000621r4, NFVSOL(18)000547r1, NFVSOL(18)000567r1, NFVSOL(18)000574, NFVSOL(18)000579, NFVSOL(18)000590r1
2018.11	0.13.0	Implemented NFVSOL(18)000563r5, NFVSOL(18)000575r5, NFVSOL(18)000586r2, NFVSOL(18)000587r1, NFVSOL(18)000589r2, NFVSOL(18)000591, NFVSOL(18)000592, NFVSOL(18)000604r2, NFVSOL(18)000606r1, NFVSOL(18)000607r5, NFVSOL(18)000608, NFVSOL(18)000609, NFVSOL(18)000612, NFVSOL(18)000614r1, NFVSOL(18)000615, NFVSOL(18)000616r2, NFVSOL(18)000617r3, NFVSOL(18)000619r3, NFVSOL(18)000620r5, NFVSOL(18)000628r2, NFVSOL(18)000629r1, NFVSOL(18)000630, NFVSOL(18)000631r1, NFVSOL(18)000632, NFVSOL(18)000634, NFVSOL(18)000635, NFVSOL(18)000636r3, NFVSOL(18)000637, NFVSOL(18)000655r2, NFVSOL(18)000658r3, NFVSOL(18)000659, NFVSOL(18)000660, NFVSOL(18)000665r1, NFVSOL(18)000666, NFVSOL(18)000678r1, NFVSOL(18)000679, NFVSOL(18)000682, NFVSOL(18)000684r1, NFVSOL(18)000687
2019.03	2.5.2	Implemented NFVSOL(19)000063r4, NFVSOL(19)000068r1, NFVSOL(19)000069r2, NFVSOL(19)000070r1, NFVSOL(19)000080, NFVSOL(19)000107r1, NFVSOL(19)000120r1, NFVSOL(19)000121, NFVSOL(19)00039r5, NFVSOL(19)00087r2, NFVSOL(19)00067r5, NFVSOL(19)00077, NFVSOL(19)00082r3, NFVSOL(19)00085, NFVSOL(19)00086, NFVSOL(19)000106r1
2019.03	2.5.3	Implemented NFVSOL(19)000084r4, NFVSOL(19)0000101r2, NFVSOL(19)0000165, NFVSOL(19)0000166, NFVSOL(19)0000167r1, NFVSOL(19)0000170r4, NFVSOL(19)0000173, NFVSOL(19)0000163, NFVSOL(19)0000119
2019.03.22	2.5.4	Editorial modification made by ETSI Secretariat allowing to structure the SOL repository on ETSI Forge in a future proof and maintainable way: <ul style="list-style-type: none"> <li>• Forge structure updated: "v2.6.1" tag created.</li> <li>• Yaml filenames updated: version numbers removed from filenames (still included in file header).</li> <li>• Import statements updated: version number removed from imported filenames.</li> <li>• Draft updated: <ul style="list-style-type: none"> <li>– Updated all references to yaml files</li> <li>– Updated the forge URLs</li> <li>– In B.1: removed the sentence explaining the meaning of x_y_z_ in the filename structure</li> </ul> </li> </ul>
2019.04.23	2.5.5	2 comments were raised during the Remote Consensus approval: both requesting to implement the WG SOL approved Change Request in NFVSOL(19)000229r3 onto the final SOL001 draft (see these 2 comments in the RC report). The present version implements NFVSOL(19)000229r3: adding machine readable meta info inside the yaml file indicating the SOL001 release version to which they apply + other editorial changes.

---

## History

<b>Document history</b>		
V2.5.1	December 2018	Publication
V2.6.1	May 2019	Publication