Scenario Analysis for High Availability in NFV

1 Introduction

This scenario analysis document outlines the model and failure modes for NFV systems. Its goal is along with the requirements documents and gap analysis help set context for engagement with various upstream projects. The OPNFV HA project team will continuously evolve these documents.

2 Basic Use Cases

In this section we review some of the basic use cases related to service high availability, that is, the availability of the service or function provided by a VNF. The goal is to understand the different scenarios that need to be considered and the specific requirements to provide service high availability. More complex use cases will be discussed in other sections.

With respect to service high availability we need to consider whether a VNF implementation is statefull or stateless and if it includes or not an HA manager which handles redundancy. For statefull VNFs we can also distinguish the cases when the state is maintained inside of the VNF or it is stored in an external shared storage making the VNF itself virtually stateless.

Managing availability usually implies a fault detection mechanism, which triggers the actions necessary for fault isolation followed by the recovery from the fault. This recovery includes two parts:

- the recovery of the service and
- the repair of the failed entity.

Very often the recovery of the service and the repair actions are perceived to be the same, for example, restarting a failed application repairs the application, which then provides the service again. Such a restart may take significant time causing service outage, for which redundancy is the solution. In cases when the service is protected by redundancy of the providing entities (e.g. application processes), the service is "failed over" to the standby or a spare entity, which replaces the failed entity while it is being repaired. E.g. when an application process providing the service fails, the standby application process takes over providing the service, while the failed one is restarted. Such a failover often allows for faster recovery of the service.

We also need to distinguish between the failed and the faulty entities as a fault may or

may not manifest in the entity containing the fault. Faults may propagate, i.e. cause other entities to fail or misbehave, i.e. an error, which in turn might be detected by a different failure or error detector entity each of which has its own scope. Similarly, the managers acting on these detected errors may have a limited scope. E.g. an HA manager contained in a VNF can only repair entities within the VNF. It cannot repair a failed VM, in fact due to the layered architecture in the VNF it cannot even know whether the VM failed, its hosting hypervisor, or the physical host. But its error detection mechanism will detect the result of such failures - a failure in the VNF - and the service can be recovered at the VNF level.On the other hand, the failure should be detected in the NFVI and the VIM should repair the failed entity (e.g. the VM). Accordingly a failure may be detected by different managers in different layers of the system, each of which may react to the event. This may cause interference. Thus, to resolve the problem in a consistent manner and completely recover from a failure the managers may need to collaborate and coordinate their actions.

Considering all these issues the following basic use cases can be identified (see table 1). These use cases assume that the failure is detected in the faulty entity (VNF component or the VM).

Table 1: VNF high availability use cases

	VNF Statefullness	**VNF Redundancy**	**Failure detection**	**Use Case**
VNF	yes	yes	VNF level only	UC1
			VNF & NFVI levels	UC2
		no	VNF level only	UC3
			VNF & NFVI levels	UC4
	no	yes	VNF level only	UC5
			VNF & NFVI levels	UC6
		no	VNF level only	UC7
			VNF & NFVI levels	UC8

As discussed, there is no guarantee that a fault manifests within the faulty entity. For example, a memory leak in one process may impact or even crash any other process running in the same execution environment. Accordingly, the repair of a failing entity (i.e. the crashed process) may not resolve the problem and soon the same or another process may fail within this execution environment indicating that the fault has remained in the system. Thus, there is a need for extrapolating the failure to a wider scope and perform the recovery at that level to get rid of the problem (at least temporarily till a patch is available for our leaking process). This requires the correlation of repeated failures in a wider scope and the escalation of the recovery action to this wider scope. In the layered architecture this means that the manager detecting the failure may not be the one in charge of the scope at which it can be

resolved, so the escalation needs to be forwarded to the manager in charge of that scope, which brings us to an additional use case UC9.

We need to consider for each of these use cases the events detected, their impact on other entities, and the actions triggered to recover the service provided by the VNF, and to repair the faulty entity.

We are going to describe each of the listed use cases from this perspective to better understand how the problem of service high availability can be tackled the best.

Before getting into the details it is worth mentioning the example end-to-end service recovery times provided in the ETSI NFV REL document [REL] (see table 2). These values may change over time including lowering these thresholds.

Table 2: Service availability levels (SAL)

SAL	Service Recovery Time Threshold	Customer Type	Recommendation
1	5-6 seconds	Network Operator Control Traffic Government/Regulatory Emergency Services	Redundant resources to be made available on-site to ensure fast recovery.
2	10-15 seconds	Enterprise and/or large scale customers Network Operators service traffic	Redundant resources to be available as a mix of on-site and off-site as appropriate: On-site resources to be utilized for recovery of real-time service; Off-site resources to be utilized for recovery of data services.
3	20-25 seconds	General Consumer Public and ISP Traffic	Redundant resources to be mostly available off-site. Real-time services should be recovered before data services.

Note that even though SAL 1 of [REL] allows for 5-6 seconds of service recovery, for many services this is too long and such outage causes a service level reset or the loss of significant amount of data. Also the end-to-end service or network service may be served by multiple VNFs. Therefore for a single VNF the desired service recovery time is sub-second.

Note that failing over the service to another provider entity implies the redirection of the traffic flow the VNF is handling. This could be achieved in different ways ranging from floating IP addresses to load balancers. The topic deserves its own investigation, therefore in these first set of use cases we assume that it is part of the solution without going into the details, which we will address as a complementary set of use cases.

[REL] ETSI GS NFV-REL 001 V1.1.1 (2015-01)

2.1 Use Case 1: VNFC failure in a statefull VNF with redundancy

Use case 1 represents a statefull VNF with redundancy managed by an HA manager, which is part of the VNF (Fig 1). The VNF consists of VNFC1, VNFC2 and the HA Manager. The latter managing the two VNFCs, e.g. the role they play in providing the service named "Provided NF" (Fig 2).

The failure happens in one of the VNFCs and it is detected and handled by the HA manager. On practice the HA manager could be part of the VNFC implementations or it could be a separate entity in the VNF. The point is that the communication of these entities inside the VNF is not visible to the rest of the system. The observable events need to cross the boundary represented by the VNF box.

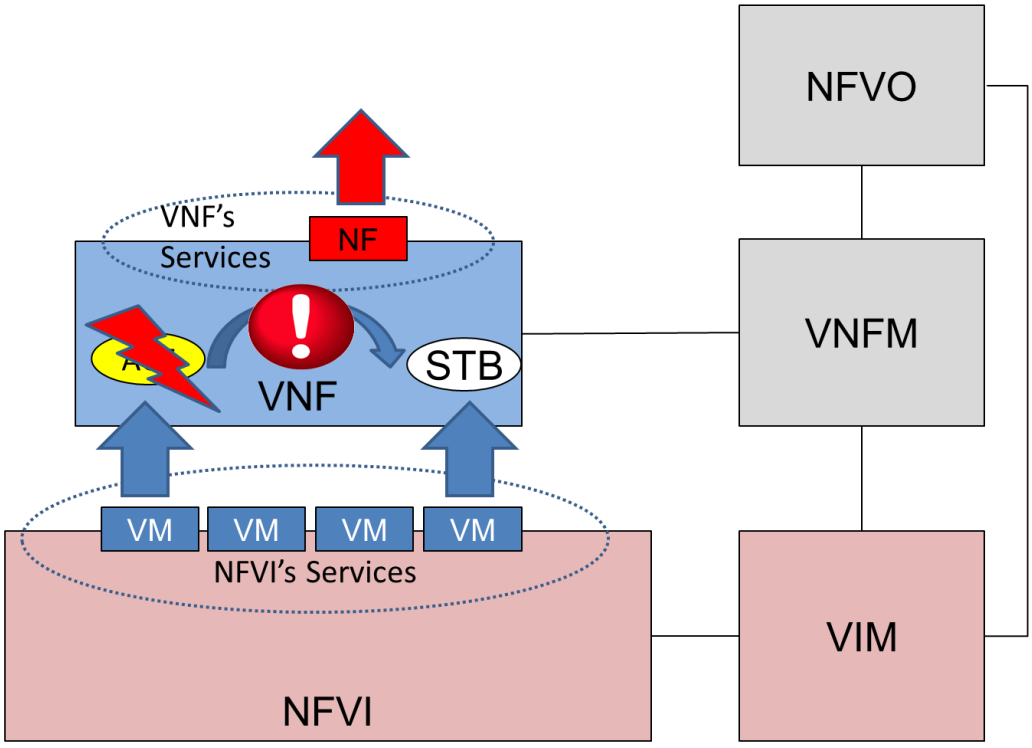

Fig 1. VNFC failure in a statefull VNF with built-in HA manager

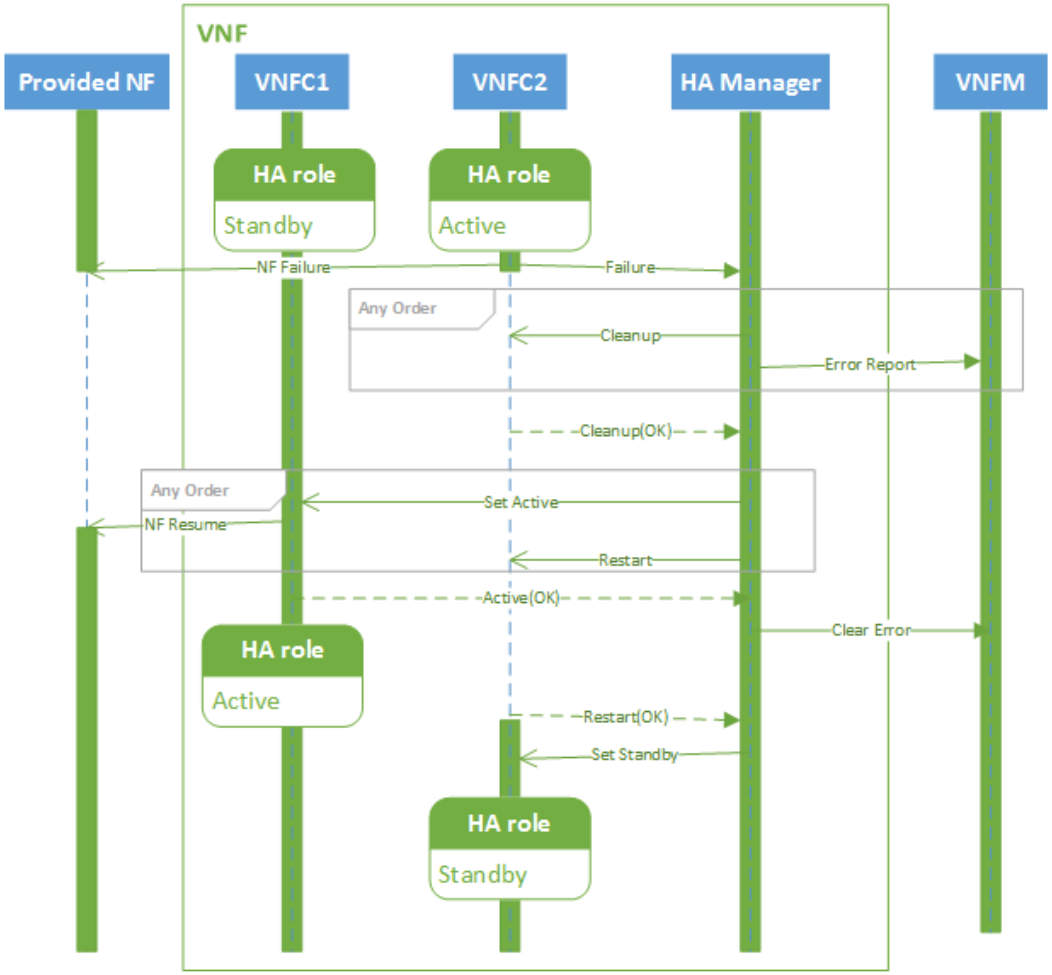

Fig 2. Sequence of events for use case 1

As shown in Fig 2. initially VNFC2 is active, i.e. provides the Provided NF and VNFC1 is a standby. It is not shown, but it is expected that VNFC1 has some means to get the update of the state of the Provided NF from the active VNFC2, so that it is prepared to continue to provide the service in case VNFC2 fails. The sequence of events starts with the failure of VNFC2, which also interrupts the Provided NF. This failure is detected somehow and/or reported to the HA Manager, which in turn may report the failure to the VNFM and simultaneously it tries to isolate the fault by cleaning up VNFC2.

Once the cleanup succeeds (i.e. the OK is received) it fails over the active role to VNFC1 by setting it active. This recovers the service, the Provided NF is indeed provided again. Thus this point marks the end of the outage caused by the failure that need to be considered from the perspective of service availability.

The repair of the failed VNFC2, which might have started at the same time when VNFC1 was assigned the active state, may take longer but without further impact on the availability of the Provided NF service. If the HA Manager reported the interruption of the Provided NF to the VNFM, it should clear the error condition.

The key points in this scenario are:

- The failure of the VNFC2 is not detectable by any other part of the system except the consumer of the Provided NF. The VNFM only knows about the failure because of the error report, and only the information this report provides. I.e. it may or may not include the information on what failed.
- The Provided NF is resumed as soon as VNFC1 is assigned active regardless how long it takes to repair VNFC2.
- The HA manager could be part of the VNFM as well. This requires an interface to detect the failures and to manage the VNFC life-cycle and the role assignments.

2.2 Use Case 2: VM failure in a statefull VNF with redundancy

Use case 2 also represents a statefull VNF with its redundancy managed by an HA manager, which is part of the VNF. The VNFCs of the VNF are hosted on the VMs provided by the NFVI (Fig 3).

The VNF consists of VNFC1, VNFC2 and the HA Manager (Fig 4). The latter managing the role the VNFCs play in providing the service - Provided NF. The VMs provided by the NFVI are managed by the VIM.

In this use case it is one of the VMs hosting the VNF fails. The failure is detected and handled at both the NFVI and the VNF levels simultaneously. The coordination occurs between the VIM and the VNFM.

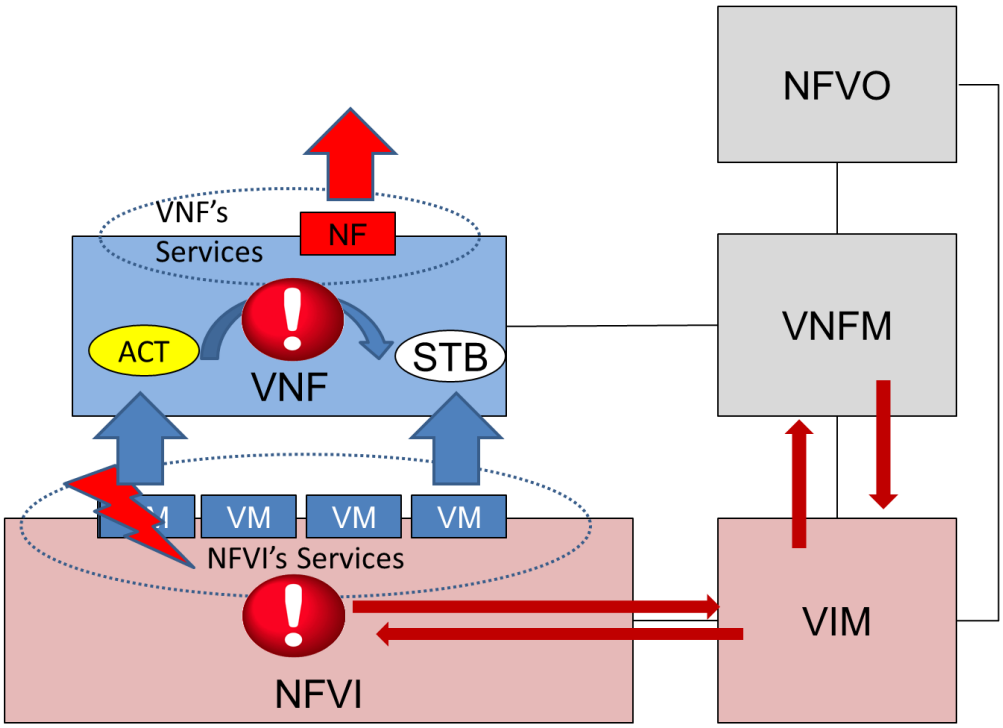

Fig 3. VM failure in a statefull VNF with built-in HA manager

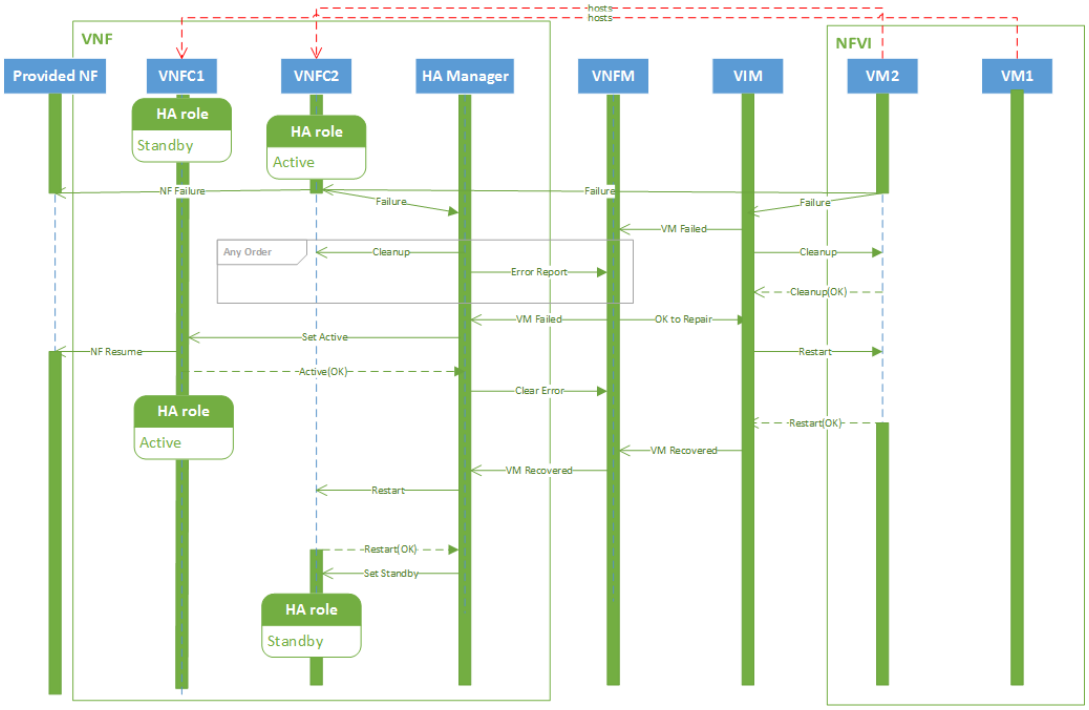

Fig 4. Sequence of events for use case 2

Again initially VNFC2 is active and provides the Provided NF, while VNFC1 is the standby. It is not shown in Fig 4., but it is expected that VNFC1 has some means to learn the state of the Provided NF from the active VNFC2, so that it is able to continue providing the service if VNFC2 fails. VNFC1 is hosted on VM1, while VNFC2 is hosted on VM2 as indicated by the arrows between these objects in Fig 4.

The sequence of events starts with the failure of VM2, which results in VNFC2 failing and interrupting the Provided NF. The HA Manager detects the failure of VNFC2 somehow and tries to handle it the same way as in use case 1. However because the VM is gone the clean up either not initiated at all or interrupted as soon as the failure of the VM is identified. In either case the faulty VNFC2 is considered as isolated.

To recover the service the HA Manager fails over the active role to VNFC1 by setting it active. This recovers the Provided NF. Thus this point marks again the end of the outage caused by the VM failure that need to be considered from the perspective of service availability. If the HA Manager reported the interruption of the Provided NF to the VNFM, it should clear the error condition.

On the other hand the failure of the VM is also detected in the NFVI and reported to the VIM. The VIM reports the VM failure to the VNFM, which passes on this information to the HA Manager of the VNF. This confirms for the VNF HA Manager the VM failure and that it needs to wait with the repair of the failed VNFC2 until the VM is provided again. The VNFM also confirms towards the VIM that it is safe to restart the VM.

The repair of the failed VM may take some time, but since the service has been failed over to VNFC1 in the VNF, there is no further impact on the availability of Provided NF.

When eventually VM2 is restarted the VIM reports this to the VNFM and the VNFC2 can be restored.

The key points in this scenario are:

- The failure of the VM2 is detectable at both levels VNF and NFVI, therefore both the HA manager and the VIM reacts to it. It is essential that these reactions do not interfere, e.g. if the VIM tries to protect the VM state at NFVI level that would conflict with the service failover action at the VNF level.
- While the failure detection happens at both NFVI and VNF levels, the time frame within which the VIM and the HA manager detect and react may be very different. For service availability the VNF level detection, i.e. by the HA manager is the critical one and expected to be faster.
- The Provided NF is resumed as soon as VNFC1 is assigned active regardless how long it takes to repair VM2 and VNFC2.
- The HA manager could be part of the VNFM as well. This requires an interface to detect failures in/of the VNFC and to manage its life-cycle and role assignments.
- The VNFM may not know for sure that the VM failed until the VIM reports it, i.e. whether the VM failure is due to host, hypervisor, host OS failure. Thus the VIM should report/alarm and log VM, hypervisor, and physical host failures. The use cases for these failures are similar with respect to the Provided NF.
- The VM repair also should start with the fault isolation as appropriate for the actual failed entity, e.g. if the VM failed due to a host failure a host may be fenced first.
- The negotiation between the VNFM and the VIM may be replaced by configured repair actions. E.g. on error restart VM in initial state, restart VM from last snapshot, or fail VM over to standby.

2.3 Use Case 3: VNFC failure in a statefull VNF with no redundancy

Use case 3 also represents a statefull VNF, but it stores its state externally on a virtual disk provided by the NFVI. It has a single VNFC and it is managed by the VNFM (Fig 5).

In this use case the VNFC fails and the failure is detected and handled by the VNFM.

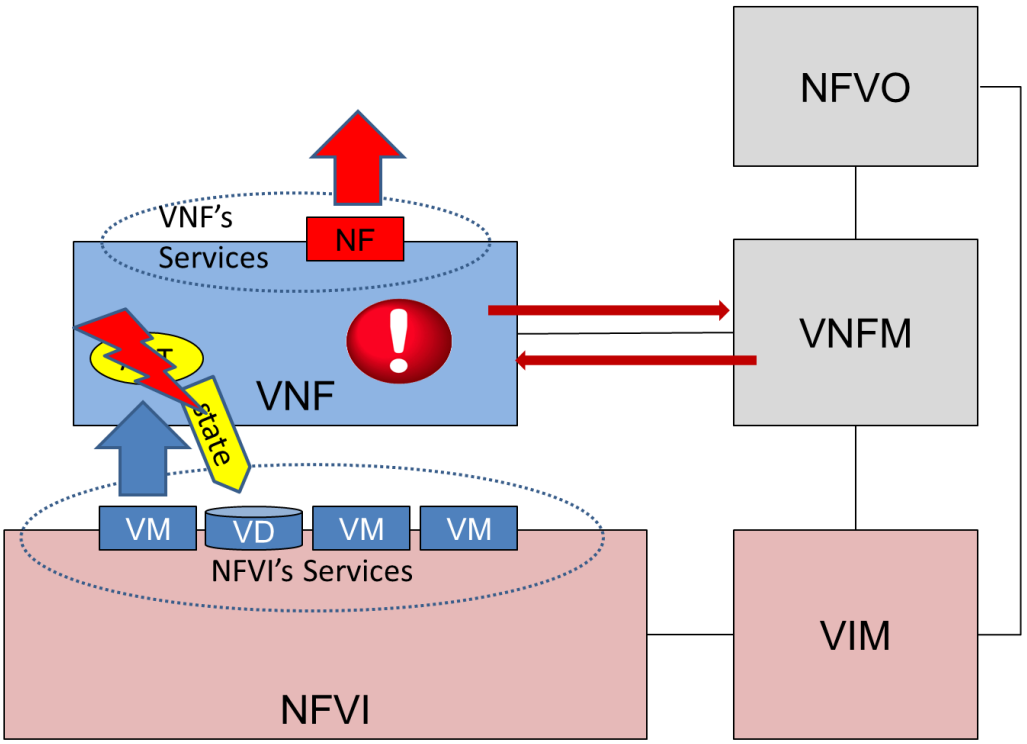

Fig 5. VNFC failure in a statefull VNF with no redundancy

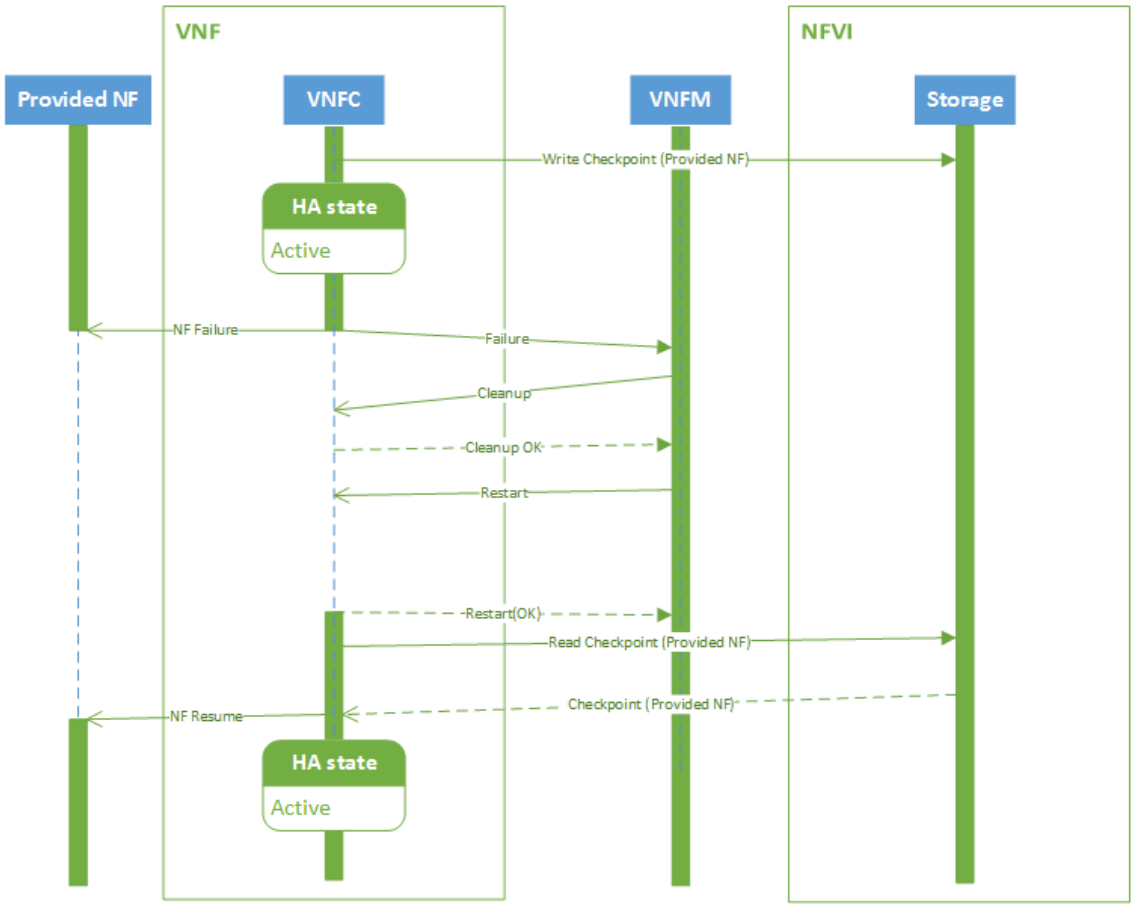

Fig 6. Sequence of events for use case 3

The VNFC periodically checkpoints the state of the Provided NF to the external storage, so that in case of failure the Provided NF can be resumed (Fig 6).

When the VNFC fails the Provided NF is interrupted. The failure is detected by the VNFM somehow, which to isolate the fault first cleans up the VNFC, then if the cleanup is successful it restarts the VNFC. When the VNFC starts up, first it reads the last checkpoint for the Provided NF, then resumes providing it. The service outage lasts from the VNFC failure till this moment.

The key points in this scenario are:

- The service state is saved in an external storage which should be highly available too to protect the service.
- The NFVI should provide this guarantee and also that storage and access network failures are handled seamlessly from the VNF's perspective.
- The VNFM has means to detect VNFC failures and manage its life-cycle appropriately. This is not required if the VNF also provides its availability management.
- The Provided NF can be resumed only after the VNFC is restarted and it has restored the service state from the last checkpoint created before the failure.
- Having a spare VNFC can speed up the service recovery. This requires that the VNFM coordinates the role each VNFC takes with respect to the Provided NF. I.e. the VNFCs do not act on the stored state simultaneously potentially interfering and corrupting it.

2.4 Use Case 4: VM failure in a statefull VNF with no redundancy

Use case 4 also represents a statefull VNF without redundancy, which stores its state externally on a virtual disk provided by the NFVI. It has a single VNFC managed by the VNFM (Fig 7) as in use case 3.

In this use case the VM hosting the VNFC fails and the failure is detected and handled by the VNFM and the VIM simultaneously.

Again, the VNFC regularly checkpoints the state of the Provided NF to the external storage, so that it can be resumed in case of a failure (Fig 8).

When the VM hosting the VNFC fails the Provided NF is interrupted.

On the one hand side, the failure is detected by the VNFM somehow, which to isolate the fault tries to clean the VNFC up which cannot be done because of the VM failure. When the absence of the VM has been determined the VNFM has to wait with restarting the VNFC until the hosting VM is restored. The VNFM may report the problem to the VIM, requesting a repair.

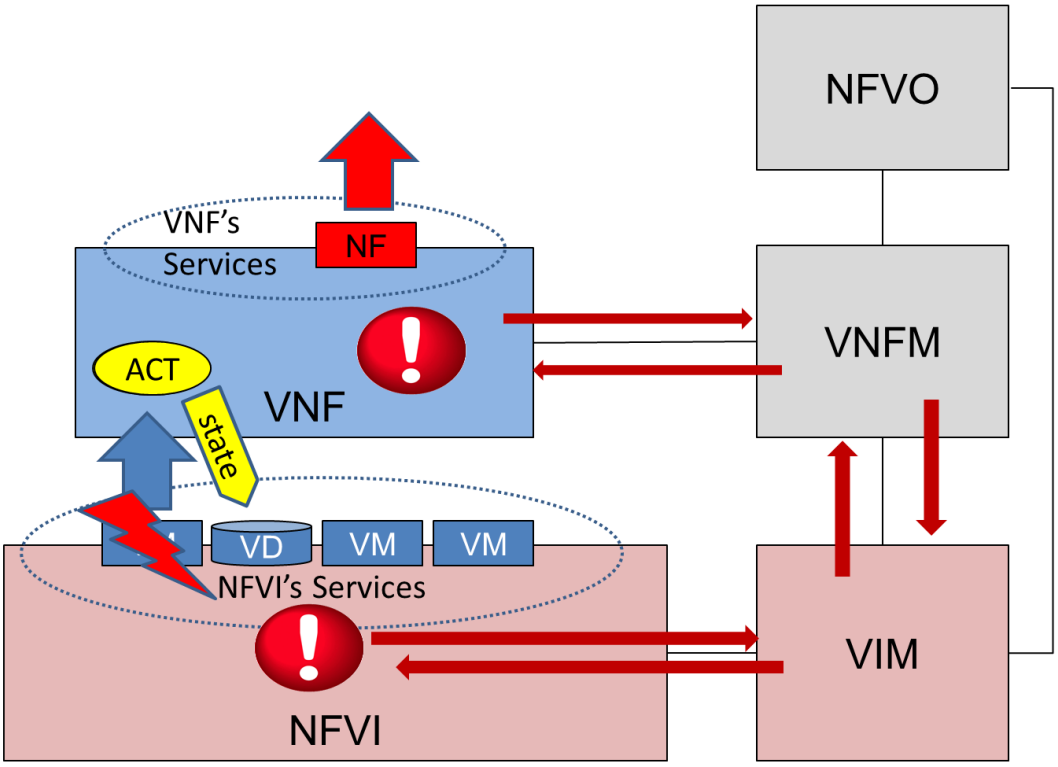

Fig 7. VM failure in a statefull VNF with no redundancy

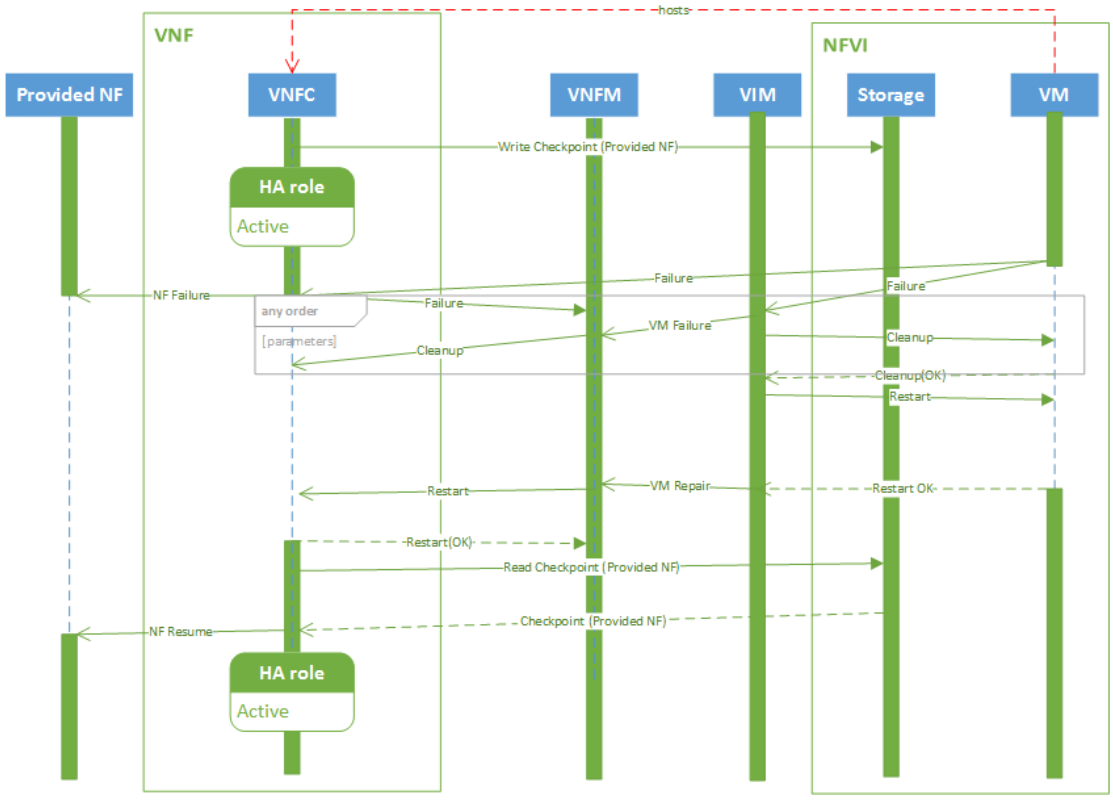

Fig 8. Sequence of events for use case 4

On the other hand the failure is detected in the NFVI and reported to the VIM, which reports it to the VNFM, if the VNFM hasn't reported it yet. If the VNFM has

requested the VM repair or if it acknowledges the repair, the VIM restarts the VM. Once the VM is up the VIM reports it to the VNFM, which in turn can restart the VNFC.

When the VNFC restarts first it reads the last checkpoint for the Provided NF, to be able to resume it. The service outage last until this is recovery completed.

The key points in this scenario are:

- The service state is saved in external storage which should be highly available to protect the service.
- The NFVI should provide such a guarantee and also that storage and access network failures are handled seamlessly from the perspective of the VNF.
- The Provided NF can be resumed only after the VM and the VNFC are restarted and the VNFC has restored the service state from the last checkpoint created before the failure.
- The VNFM has means to detect VNFC failures and manage its life-cycle appropriately. Alternatively the VNF may also provide its availability management.
- The VNFM may not know for sure that the VM failed until the VIM reports this. It also cannot distinguish host, hypervisor and host OS failures. Thus the VIM should report/alarm and log VM, hypervisor, and physical host failures. The use cases for these failures are similar with respect to the Provided NF.
- The VM repair also should start with the fault isolation as appropriate for the actual failed entity, e.g. if the VM failed due to a host failure a host may be fenced first.
- The negotiation between the VNFM and the VIM may be replaced by configured repair actions.
- VM level redundancy, i.e. running a standby or spare VM in the NFVI would allow faster service recovery for this use case, but by itself it may not protect against VNFC level failures. I.e. VNFC level error detection is still required.

2.5 Use Case 5: VNFC failure in a stateless VNF with redundancy

Use case 5 represents a stateless VNF with redundancy, i.e. it is composed of VNFC1 and VNFC2. They are managed by an HA manager within the VNF. The HA manager assigns the active role to provide the Provided NF to one of the VNFCs while the other remains a spare meaning that it has no state information for the Provided NF (Fig 9) therefore it could replace any other VNFC capable of providing the Provided NF service.

In this use case the VNFC fails and the failure is detected and handled by the HA manager.

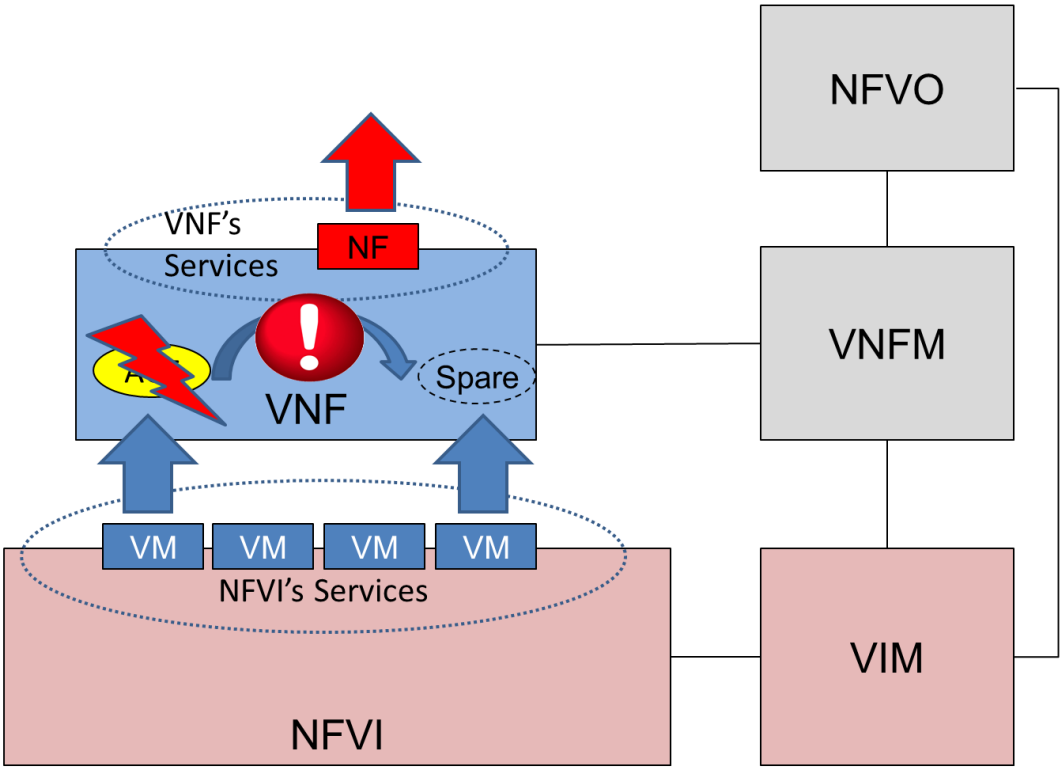

Fig 9. VNFC failure in a stateless VNF with redundancy

Initially VNFC2 provides the Provided NF while VNFC1 is idle or might not even been instantiated yet (Fig 10).

When VNFC2 fails the Provided NF is interrupted. This failure is detected by the HA manager, which as a first reaction cleans up VNFC2 (fault isolation), then it assigns the active role to VNFC1. It may report an error to the VNFM as well.

Since there is no state information to recover, VNFC1 can accept the active role right away and resume providing the Provided NF service. Thus the service outage is over. If the HA manager reported an error to the VNFM it should clear it at this point.

The key points in this scenario are:

- The spare VNFC may be instantiated only once the failure of active VNFC is detected.
- As a result the HA manager's role might be limited to life-cycle management, i.e. no role assignment is needed if the VNFCs provide the service as soon as they are started up.
- Accordingly the HA management could be part of a generic VNFM provided it is capable of detecting the VNFC failures. Besides the service users, the VNFC failure may not be detectable at any other part of the system.
- Also there could be multiple active VNFCs sharing the load of Provided NF

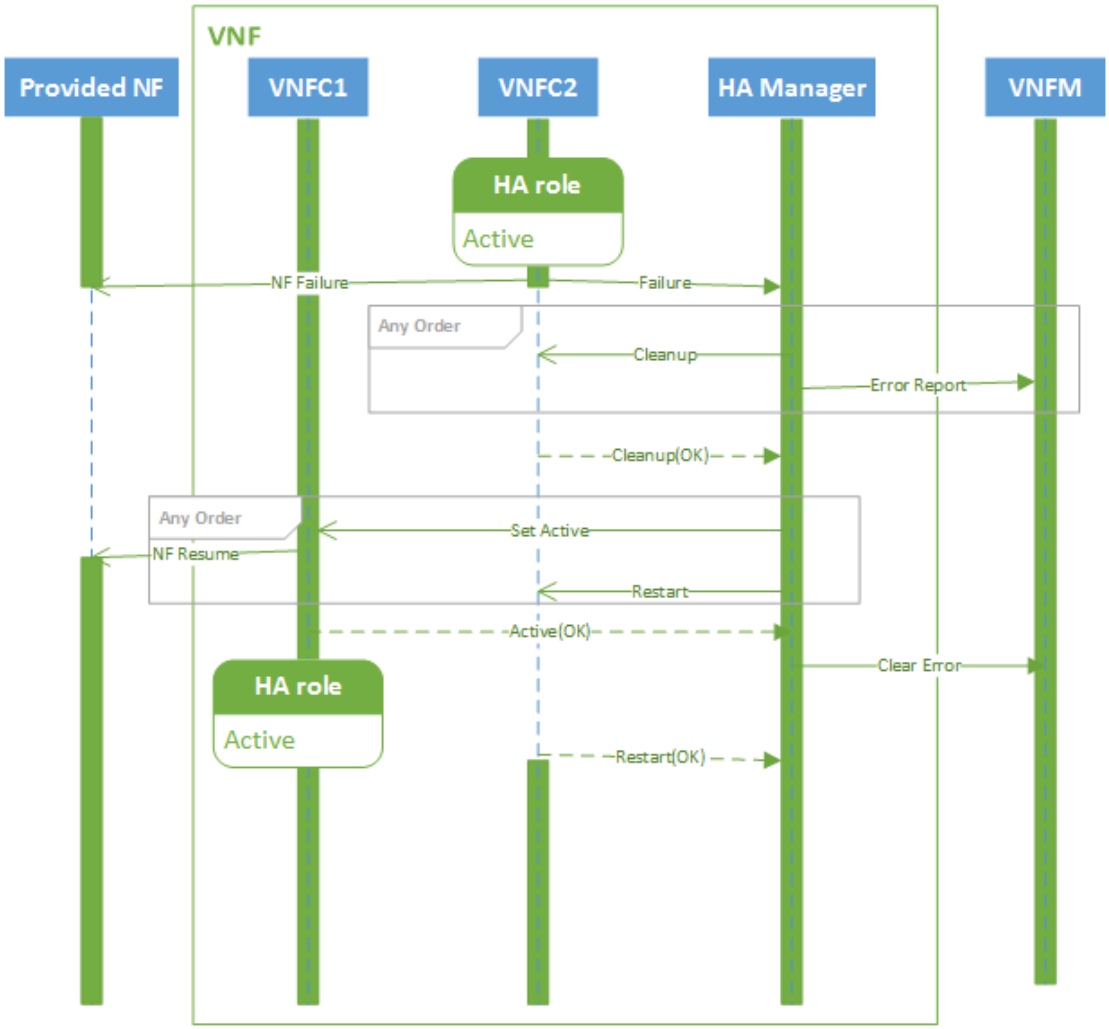

Fig 10. Sequence of events for use case 5

and the spare/standby may protect all of them.

- Reporting the service failure to the VNFM is optional as the HA manager is in charge of recovering the service and it is aware of the redundancy needed to do so.

2.6 Use Case 6: VM failure in a stateless VNF with redundancy

Similarly to use case 5, use case 6 represents a stateless VNF composed of VNFC1 and VNFC2, which are managed by an HA manager within the VNF. The HA manager assigns the active role to provide the Provided NF to one of the VNFCs while the other remains a spare meaning that it has no state information for the Provided NF (Fig 11) and it could replace any other VNFC capable of providing the Provided NF service.

As opposed to use case 5 in this use case the VM hosting one of the VNFCs fails. This failure is detected and handled by the HA manager as well as the VIM.

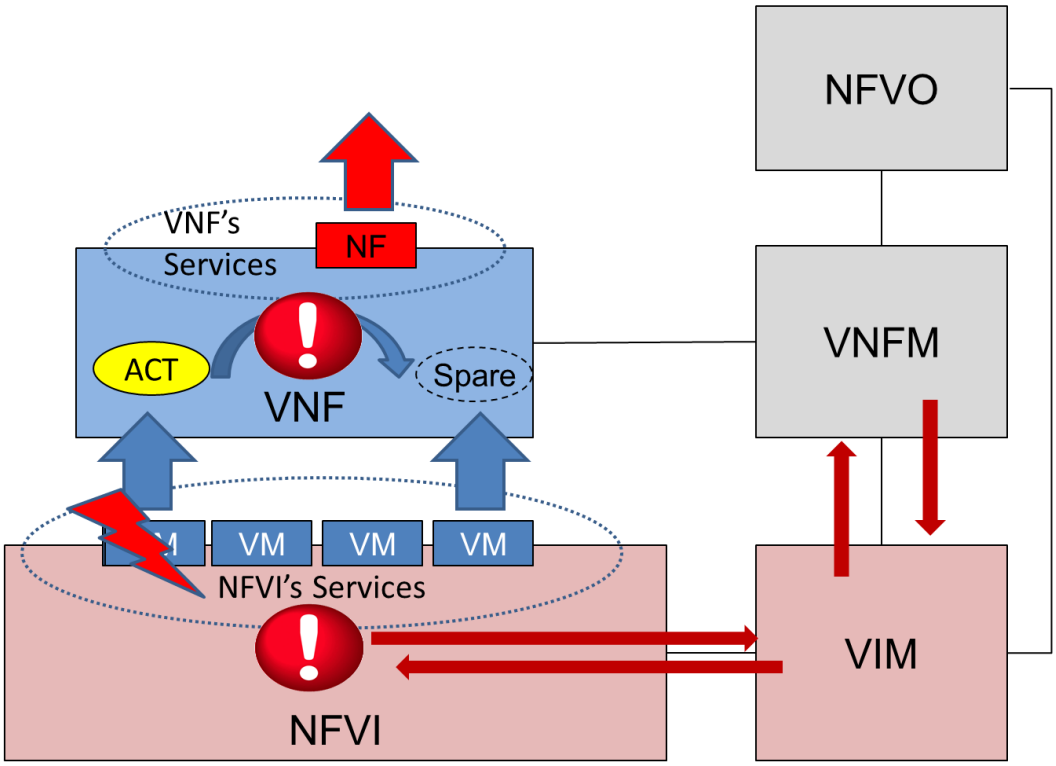

Fig 11. VM failure in a stateless VNF with redundancy

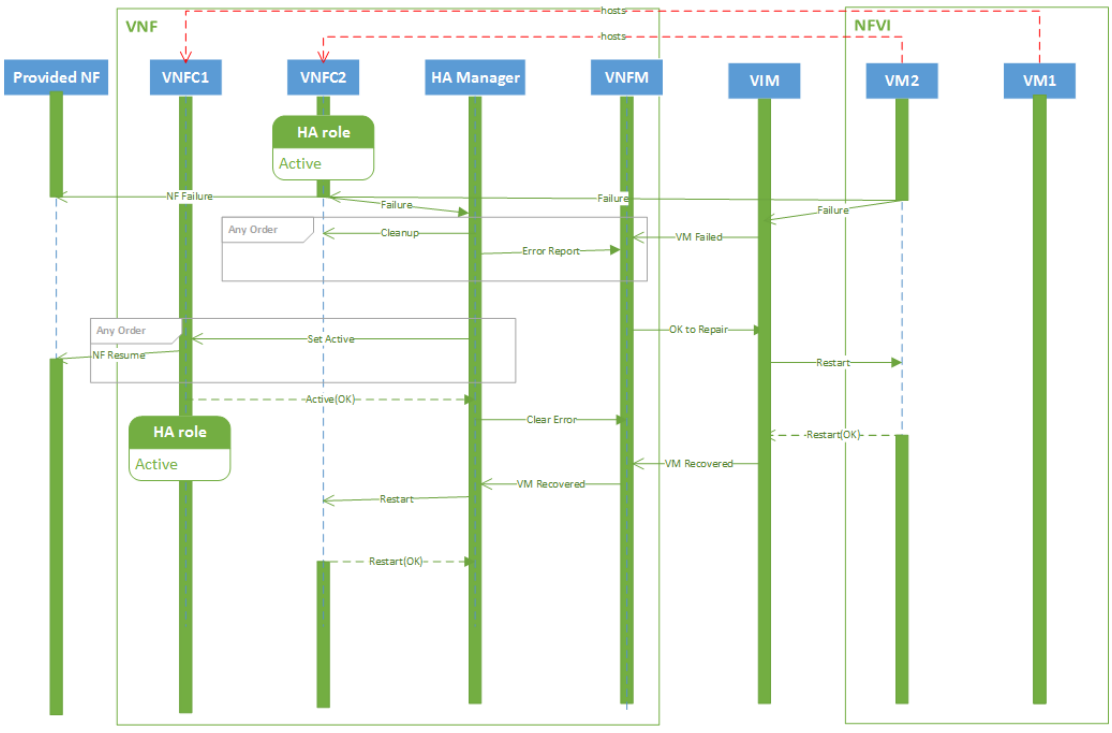

Fig 12. Sequence of events for use case 6

Initially VNFC2 provides the Provided NF while VNFC1 is idle or might not have been instantiated yet (Fig 12) as in use case 5.

When VM2 fails VNFC2 fails with it and the Provided NF is interrupted. The failure is detected by the HA manager and by the VIM simultaneously and independently.

The HA manager's first reaction is trying to clean up VNFC2 to isolate the fault. This is considered to be successful as soon as the disappearance of the VM is confirmed. After this the HA manager assigns the active role to VNFC1. It may report the error to the VNFM as well requesting a VM repair.

Since there is no state information to recover, VNFC1 can accept the assignment right away and resume the Provided NF service. Thus the service outage is over. If the HA manager reported an error to the VNFM for the service it should clear it at this point.

Simultaneously the VM failure is detected in the NFVI and reported to the VIM, which reports it to the VNFM, if the VNFM hasn't requested a repair yet. If the VNFM requested the VM repair or if it acknowledges the repair, the VIM restarts the VM.

Once the VM is up the VIM reports it to the VNFM, which in turn may restart the VNFC if needed.

The key points in this scenario are:

- The spare VNFC may be instantiated only after the detection of the failure of the active VNFC.
- As a result the HA manager's role might be limited to life-cycle management, i.e. no role assignment is needed if the VNFC provides the service as soon as it is started up.
- Accordingly the HA management could be part of a generic VNFM provided if it is capable of detecting failures in/of the VNFC and managing its life-cycle.
- Also there could be multiple active VNFCs sharing the load of Provided NF and the spare/standby may protect all of them.
- The VNFM may not know for sure that the VM failed until the VIM reports this. It also cannot distinguish host, hypervisor and host OS failures. Thus the VIM should report/alarm and log VM, hypervisor, and physical host failures. The use cases for these failures are similar with respect to each Provided NF.
- The VM repair also should start with the fault isolation as appropriate for the actual failed entity, e.g. if the VM failed due to a host failure a host needs to be fenced first.
- The negotiation between the VNFM and the VIM may be replaced by configured repair actions.
- Reporting the service failure to the VNFM is optional as the HA manager is in charge recovering the service and it is aware of the redundancy needed to do so.

2.7 Use Case 7: VNFC failure in a stateless VNF with

no redundancy

Use case 7 represents a stateless VNF composed of a single VNFC, i.e. with no redundancy. The VNF and in particular its VNFC is managed by the VNFM through managing its life-cycle (Fig 13).

In this use case the VNFC fails. This failure is detected and handled by the VNFM. This use case requires that the VNFM can detect the failures in the VNF or they are reported to the VNFM.

The failure is only detectable at the VNFM level and it is handled by the VNFM restarting the VNFC.

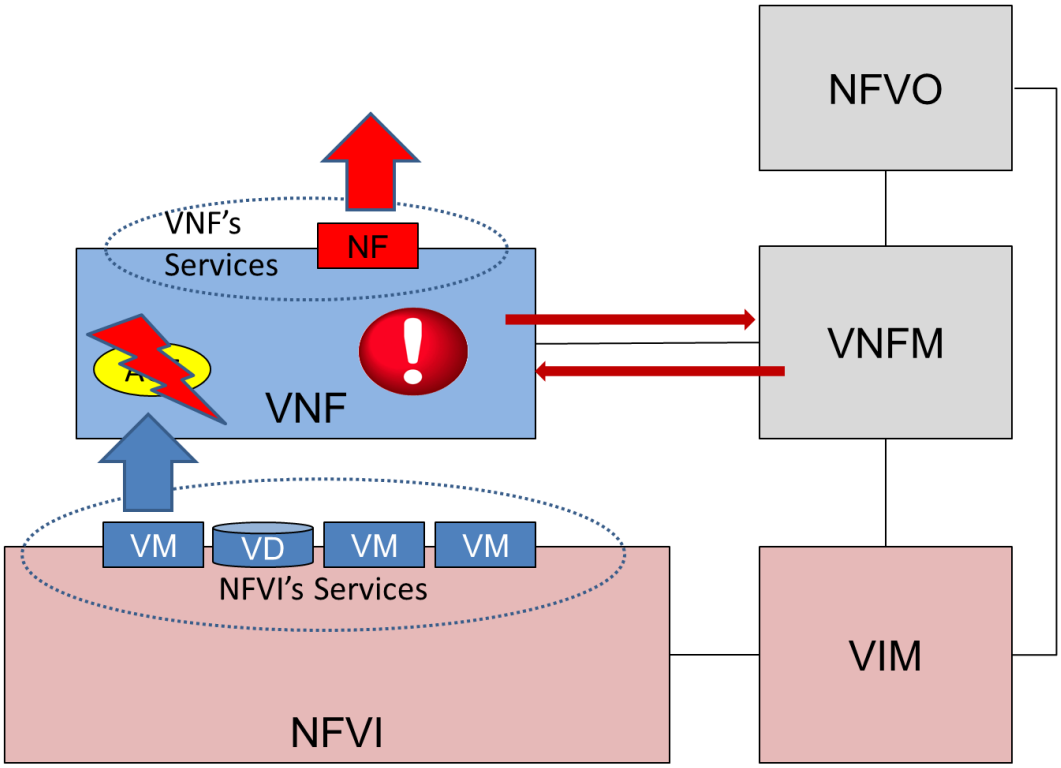

Fig 13. VNFC failure in a stateless VNF with no redundancy

The VNFC is providing the Provided NF when it fails (Fig 14). This failure is detected or reported to the VNFM, which has to clean up the VNFC to isolate the fault. After cleanup success it can proceed with restarting the VNFC, which as soon as it is up it starts to provide the Provided NF as there is no state to recover.

Thus the service outage is over, but it has included the entire time needed to restart the VNFC. Considering that the VNF is stateless this may not be significant still.

The key points in this scenario are:

- The VNFM has to have the means to detect VNFC failures and manage its life-cycle appropriately. This is not required if the VNF comes with its availability management, but this is very unlikely for such stateless VNFs.

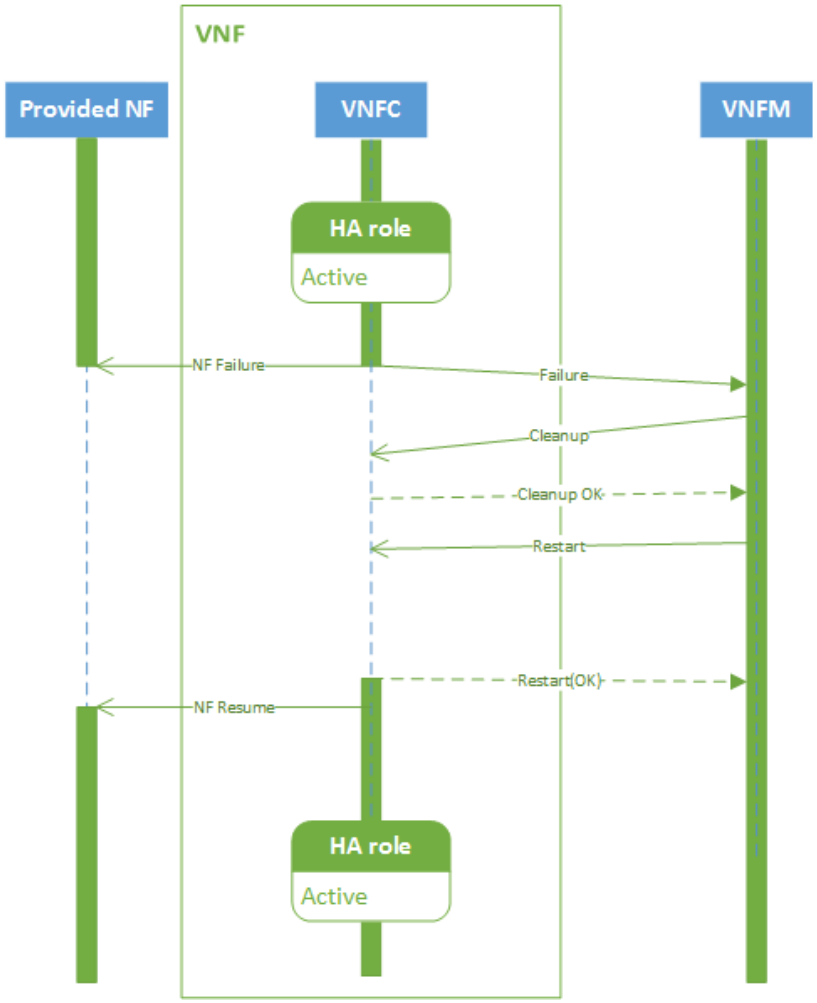

Fig 14. Sequence of events for use case 7

- The Provided NF can be resumed as soon as the VNFC is restarted, i.e. the restart time determines the outage.
- In case multiple VNFCs are used they should not interfere with one another, they should operate independently.

2.8 Use Case 8: VM failure in a stateless VNF with no redundancy

Use case 8 represents the same stateless VNF composed of a single VNFC as use case 7, i.e. with no redundancy. The VNF and in particular its VNFC is managed by the VNFM through managing its life-cycle (Fig 15).

In this use case the VM hosting the VNFC fails. This failure is detected and handled by the VNFM as well as by the VIM.

The VNFC is providing the Provided NF when the VM hosting the VNFC fails (Fig 16). This failure may be detected or reported to the VNFM as a failure of the VNFC. The VNFM may not be aware at this point that it is a VM failure. Accordingly its first

reaction as in use case 7 is to clean up the VNFC to isolate the fault. Since the VM is gone, this cannot succeed and the VNFM becomes aware of the VM failure through this or it is reported by the VIM. In either case it has to wait with the repair of the VMFC until the VM becomes available again.

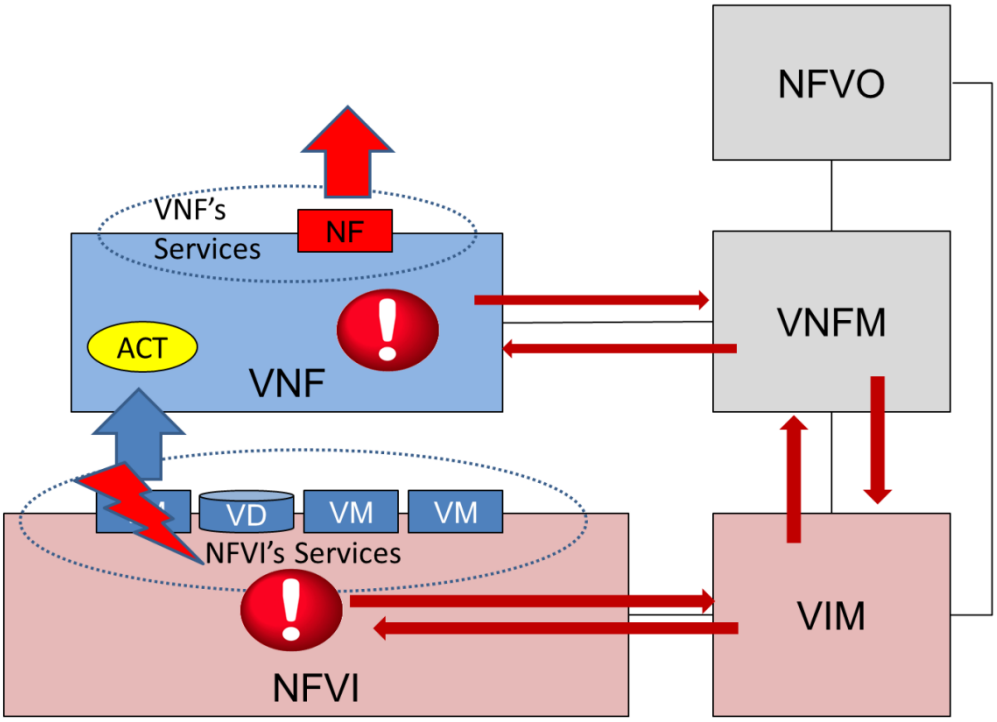

Fig 15. VM failure in a stateless VNF with no redundancy

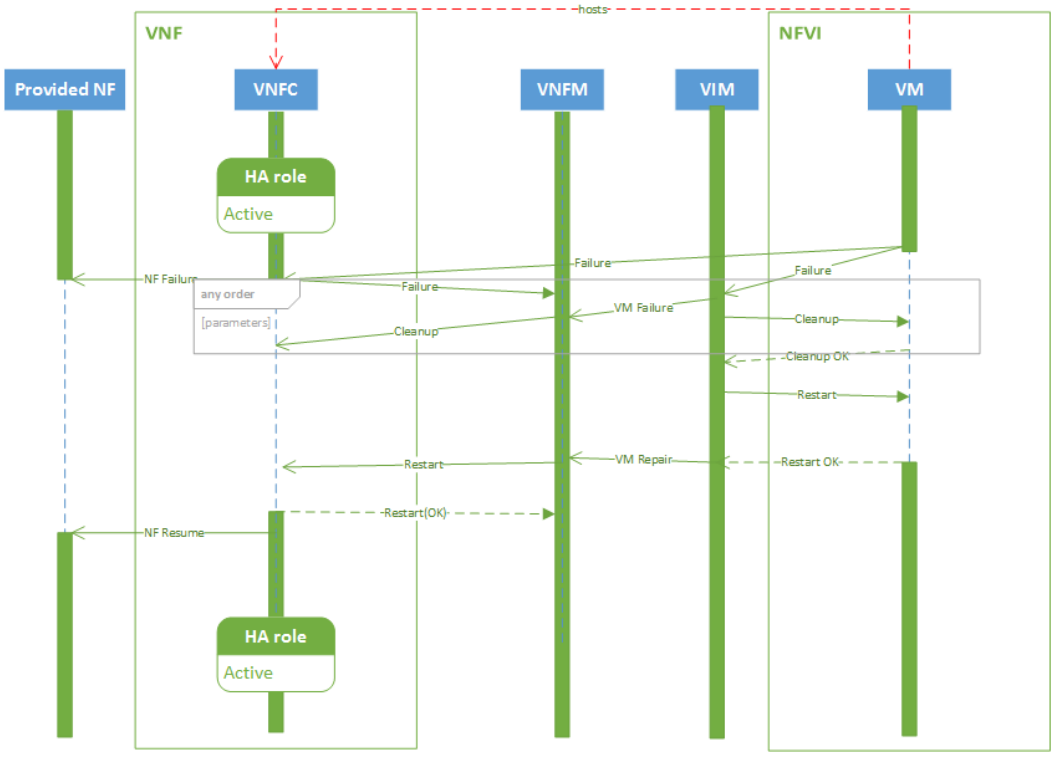

Fig 16. Sequence of events for use case 8

Meanwhile the VIM also detects the VM failure and reports it to the VNFM unless the VNFM has already requested the VM repair. After the VNFM confirming the VM repair the VIM restarts the VM and reports the successful repair to the VNFM, which in turn can start the VNFC hosted on it.

Thus the recovery of the Provided NF includes the restart time of the VM and of the VNFC.

The key points in this scenario are:

- The VNFM has to have the means to detect VNFC failures and manage its life-cycle appropriately. This is not required if the VNF comes with its availability management, but this is very unlikely for such stateless VNFs.
- The Provided NF can be resumed only after the VNFC is restarted on the repaired VM, i.e. the restart time of the VM and the VNFC determines the outage.
- In case multiple VNFCs are used they should not interfere with one another, they should operate independently.
- The VNFM may not know for sure that the VM failed until the VIM reports this. It also cannot distinguish host, hypervisor and host OS failures. Thus the VIM should report/alarm and log VM, hypervisor, and physical host failures. The use cases for these failures are similar with respect to each Provided NF.
- The VM repair also should start with the fault isolation as appropriate for the actual failed entity, e.g. if the VM failed due to a host failure the host needs to be fenced first.
- The repair negotiation between the VNFM and the VIM may be replaced by configured repair actions.
- VM level redundancy, i.e. running a standby or spare VM in the NFVI would allow faster service recovery for this use case, but by itself it may not protect against VNFC level failures. I.e. VNFC level error detection is still required.

2.9 Use Case 9: Repeated VNFC failure in a stateless VNF with no redundancy

Finally use case 9 represents again a stateless VNF composed of a single VNFC as in use case 7, i.e. with no redundancy. The VNF and in particular its VNFC is managed by the VNFM through managing its life-cycle.

In this use case the VNFC fails repeatedly. This failure is detected and handled by the VNFM, but results in no resolution of the fault (Fig 17) because the VNFC is manifesting a fault, which is not in its scope. I.e. the fault is propagating to the VNFC from a faulty VM or host, for example. Thus the VNFM cannot resolve the problem

by itself.

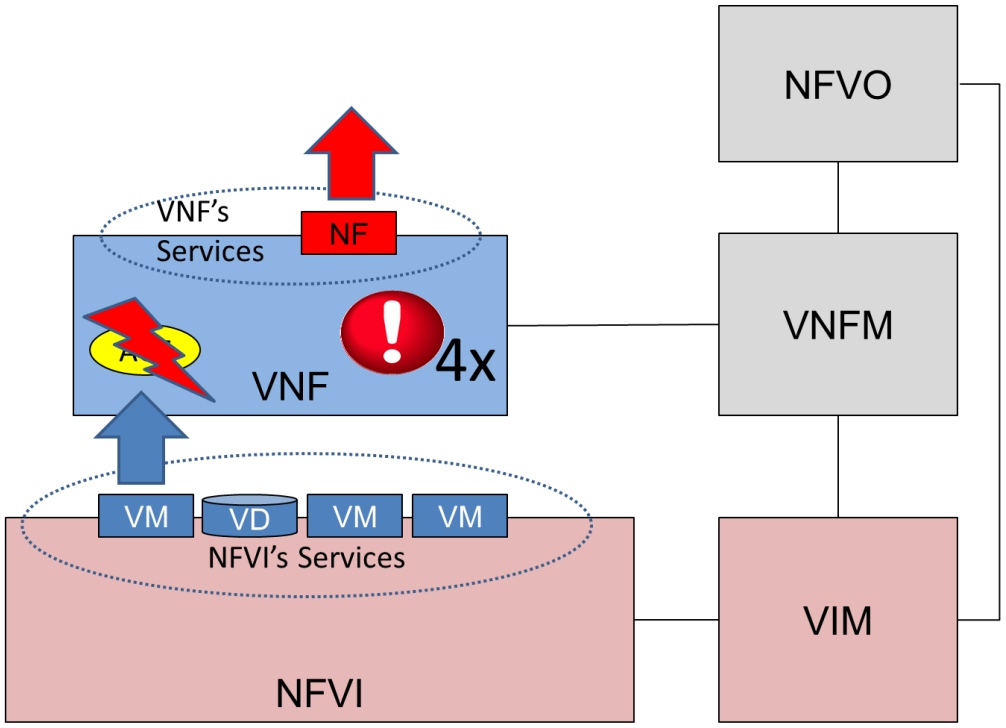

Fig 17. VM failure in a stateless VNF with no redundancy

To handle this case the failure handling needs to be escalated to the a bigger fault zone (or fault domain), i.e. a scope within which the faults may propagate and manifest. In case of the VNF the bigger fault zone is the VM and the facilities hosting it, all managed by the VIM.

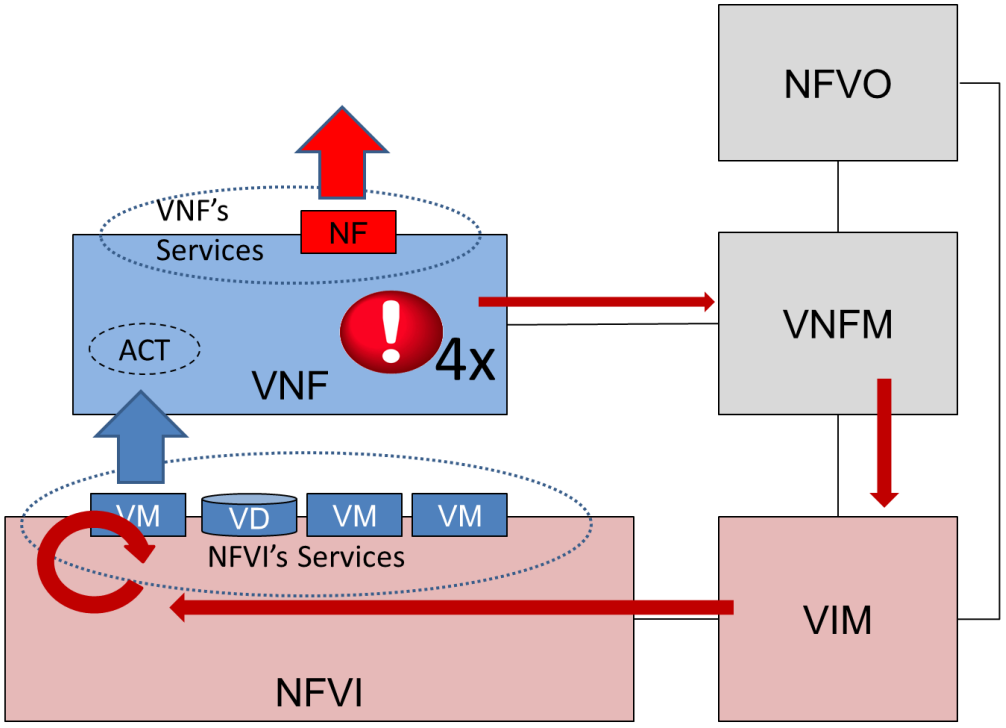

Fig 18. VM failure in a stateless VNF with no redundancy

Thus the VNFM should request the repair from the VIM (Fig 18).

Since the VNFM is only aware of the VM, it needs to report an error on the VM and it is the VIM's responsibility to sort out what might be the scope of the actual fault depending on other failures and error reports in its scope.

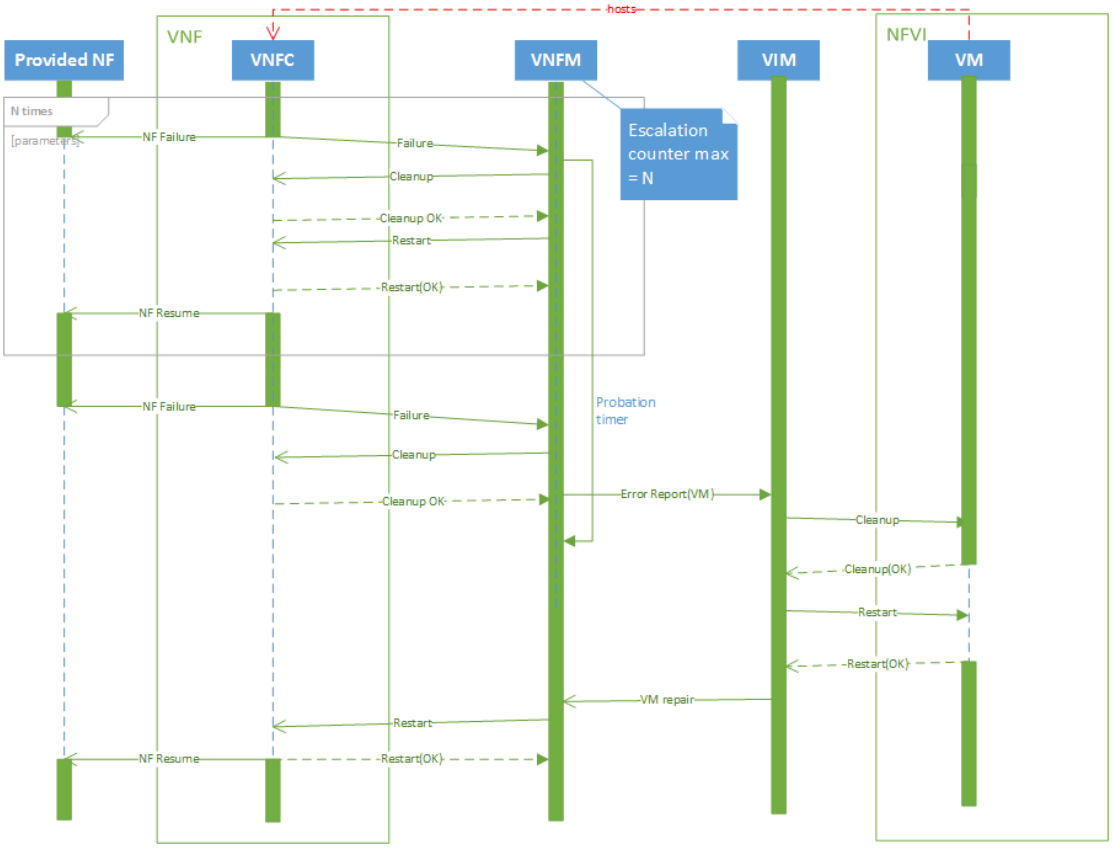

Fig 19. Sequence of events for use case 9

This use case starts similarly to use case 7, i.e. the VNFC is providing the Provided NF when it fails (Fig 17). This failure is detected or reported to the VNFM, which cleans up the VNFC to isolate the fault. After successful cleanup the VNFM proceeds with restarting the VNFC, which as soon as it is up starts to provide the Provided NF again as in use case 7.

However the VNFC failure occurs N times repeatedly within some Probation time for which the VNFM starts the timer when it detects the first failure of the VNFC. When the VNFC fails once more still within the probation time the Escalation counter maximum is exceeded and the VNFM reports an error to the VIM on the VM hosting the VNFC as obviously cleaning up and restarting the VNFC did not solve the problem.

When the VIM receives the error report for the VM it has to isolate the fault by cleaning up at least the VM. After successful cleanup it can restart the VM and once it is up report the VM repair to the VNFM. At this point the VNFM can restart the VNFC, which in turn resumes the Provided VM.

In this scenario the VIM needs to evaluate what may be the scope of the fault to determine what entity needs a repair. For example, if it has detected VM failures on that same host, or other VNFMs reported errors on VMs hosted on the same host, it should consider that the entire host needs a repair.

The key points in this scenario are:

- The VNFM has to have the means to detect VNFC failures and manage its life-cycle appropriately. This is not required if the VNF comes with its availability management, but this is very unlikely for such stateless VNFs.
- The VNFM needs to correlate VNFC failures over time to be able to detect failure of a bigger fault zone. One way to do so is through counting the failures within a probation time.
- The VIM cannot detect all failures caused by faults in the entities under its control. It should be able to receive error reports and correlate these error reports based on the dependencies of the different entities.
- The VNFM does not know the source of the failure, i.e. the faulty entity.
- The VM repair should start with the fault isolation as appropriate for the actual failed entity, e.g. if the VM failed due to a host failure the host needs to be fenced first.

3 Communication Interfaces for VNF HA schemes

This section will discuss some general issues about communication interfaces in the VNF HA schemes. In sections 2, the usecases of both stateful and stateless VNFs are discussed. While in this section, we would like to discuss some specific issues which are quite general for all the usecases proposed in the previous sections.

3.1 VNF External Interfaces

Regardless whether the VNF is stateful or stateless, all the VNFCs should act as a union from the perspective of the outside world. That means all the VNFCs should share a common interface where the outside modules (e.g., the other VNFs) can access the service from. There could be multiple solutions for this share of IP interface. However, all of this sharing and switching of IP address should be ignorant to the outside modules.

There are several approaches for the VNFs to share the interfaces. A few of them are listed as follows and will be discussed in detail.

- IP address of VMs for active/stand-by VM.

- Load balancers for active/active use cases

Note that combination of these two approaches is also feasible.

For active/standby VNFCs, there is a common IP address shared by the VMs hosting the active and standby VNFCs, so that they look as one instance from outside. The HA manager will manage the assignment of the IP address to the VMs. (The HA manager may not be aware of this, I.e. the address may be configured and the active/standby state management is linked to the possession of the IP address, i.e. the active VNFC claims it as part of becoming active.) Only the active one possesses the IP address. And when failover happens, the standby is set to be active and can take possession of the IP address to continue traffic process.

For active/active VNFCs, a LB (Load Balancer) could be used. In such scenario, there could be two cases for the deployment and usage of LB.

Case 1: LB used in front of a cluster of VNFCs to distribute the traffic flow.

In such case, the LB is deployed in front of a cluster of multiple VNFCs. Such cluster can be managed by a separate cluster manager, or can be managed just by the LB, which uses heartbeat to monitor each VNFC. When one of VNFCs fails, the cluster manager should recover the failed one, and should also exclude the failed VNFC from the cluster so that the LB will re-route the traffic to the other VNFCs. In the case when the LB is acting as the cluster manager, it is the LB's responsibility to inform the VNFM to recover the failed VNFC if possible.

Case 2: LB used in front of a cluster of VMs to distribute traffic flow.

In this case, there exists a cluster manager (e.g. Pacemaker) to monitor and manage the VMs in the cluster. The LB sits in front of the VM cluster so as to distribute the traffic. When one of the VM fails, the cluster manager will detect that and will be in charge of the recovery. The cluster manager will also exclude the failed VM out of the cluster, so that the LB won't route traffic to the failed one.

In both two cases, the HA of the LB should also be considered.

3.2 Intra-VNF Communication

For stateful VNFs, data synchronization is necessary between the active and standby VMs. The HA manager is responsible for handling VNFC failover, and do the assignment of the active/standby states between the VNFCs of the VNF. Data synchronization can be handled either by the HA manager or by the VNFC itself.

The state synchronization can happen as.

- direct communication between the active and the standby VNFCs.
- based on the information received from the HA manager on channel or messages using a common queue.

- it could be through a shared storage assigned to the whole VNF.
- through checkpointing of state information via underlying memory and/or database checkpointing services to a separate VM and storage repository.

4 High Availability Scenarios for Network Nodes

4.1 Network nodes and HA deployment

OpenStack network nodes contain: Neutron DHCP agent, Neutron L2 agent, Neutron L3 agent, Neutron LBaaS agent and Neutron Metadata agent. The DHCP agent provides DHCP services for virtual networks. The metadata agent provides configuration information such as credentials to instances. Note that the L2 agent cannot be distributed and highly available. Instead, it must be installed on each data forwarding node to control the virtual network drivers such as Open vSwitch or Linux Bridge. One L2 agent runs per node and controls its virtual interfaces.

A typical HA deployment of network nodes can be achieved in Fig 20. Here shows a two nodes cluster. The number of the nodes is decided by the size of the cluster. It can be 2 or more. More details can be achieved from each agent's part.

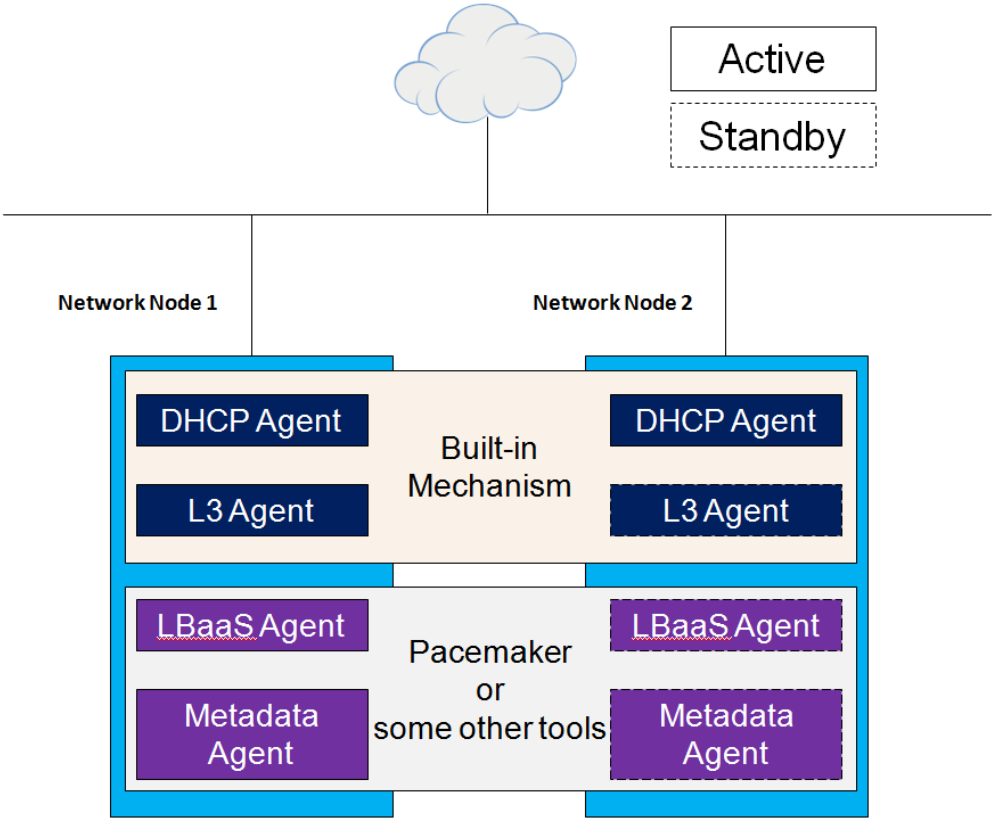

Fig 20. A typical HA deployment of network nodes

4.2 DHCP agent

The DHCP agent can be natively highly available. Neutron has a scheduler which lets you run multiple agents across nodes. You can configure the dhcp_agents_per_network parameter in the neutron.conf file and set it to X (X >=2 for HA, default is 1).

If the X is set to 2, as depicted in Fig 21 three tenant networks (there can be multiple tenant networks) are used as an example, six DHCP agents are deployed in two nodes for three networks, they are all active. Two dhcp1s serve one network, dhcp2s and dhcp3s serve other two different networks. In a network, all DHCP traffic is broadcast, DHCP servers race to offer IP. All the servers will update the lease tables. In Fig 22, when the agent(s) in Node1 doesn't work which can be caused by software failure or hardware failure, the dhcp agent(s) on Node2 will continue to offer IP for the network.

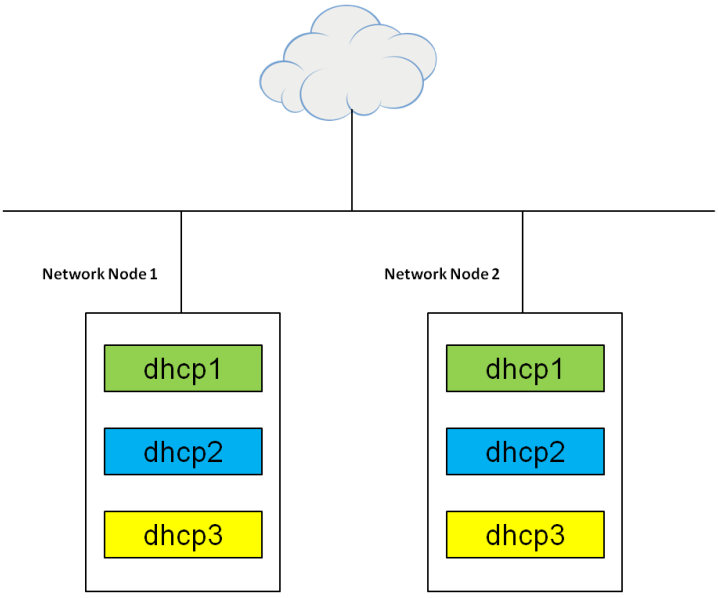

Fig 21. Natively HA deployment of DHCP agents

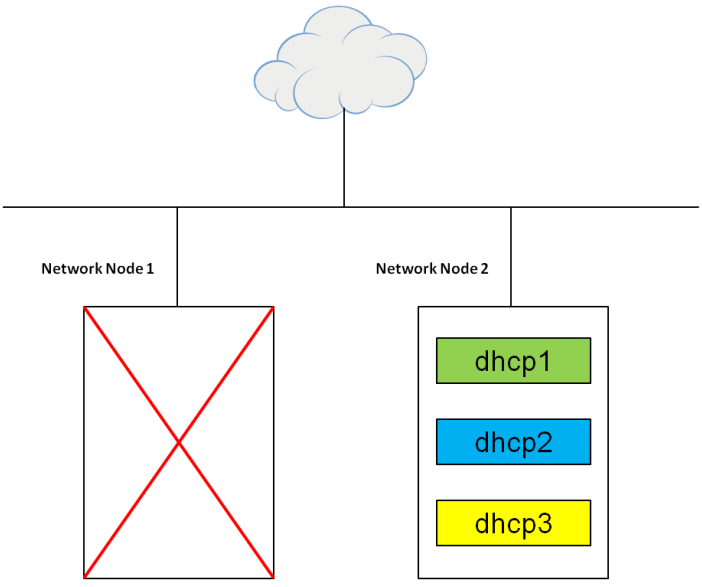

Fig 22. Failure of DHCP agents

4.3 L3 agent

The L3 agent is also natively highly available. To achieve HA, it can be configured in the neutron.conf file.

l3_ha = True # All routers are highly available by default
allow<F7>_automatic_l3agent_failover = True # Set automatic L3 agent failover for routers
max_l3_agents_per_router = 2 # Maximum number of network nodes to use for the HA router
min_l3_agents_per_router = 2 # Minimum number of network nodes to use for the HA router. A new router can be created only if this number of network nodes is available.

According to the neutron.conf file, the L3 agent scheduler supports Virtual Router Redundancy Protocol (VRRP) to distribute virtual routers across multiple nodes (e.g. 2). The scheduler will choose a number between the maximum and the minimum number according scheduling algorithm. VRRP is implemented by Keepalived.

As depicted in Fig 23, both L3 agents in Node1 and Node2 host vRouter 1 and vRouter 2. In Node 1, vRouter 1 is active and vRouter 2 is standby (hot standby). In Node2, vRouter 1 is standby and vRouter 2 is active. For the purpose of reducing the load, two actives are deployed in two Nodes alternatively. In Fig 24, Keepalived will be used to manage the VIP interfaces. One instance of keepalived per virtual router, then one per namespace. 169.254.192.0/18 is a dedicated HA network which is created in order to isolate the administrative traffic from the tenant traffic, each vRouter will be connected to this dedicated network via an HA port. More details can be achieved from the Reference at the bottom.

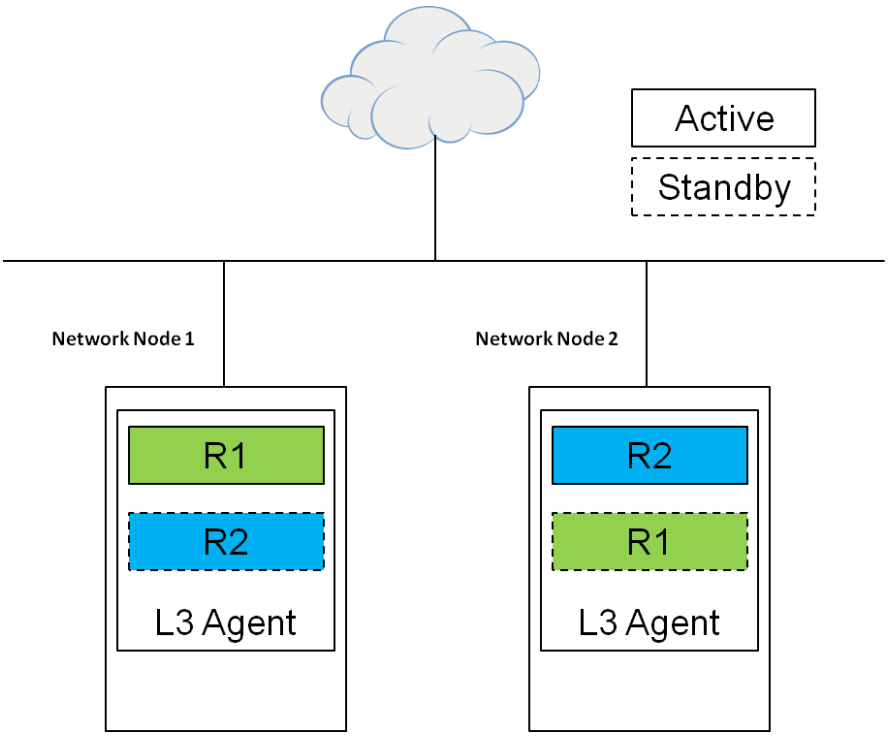

Fig 23. Natively HA deployment of L3 agents

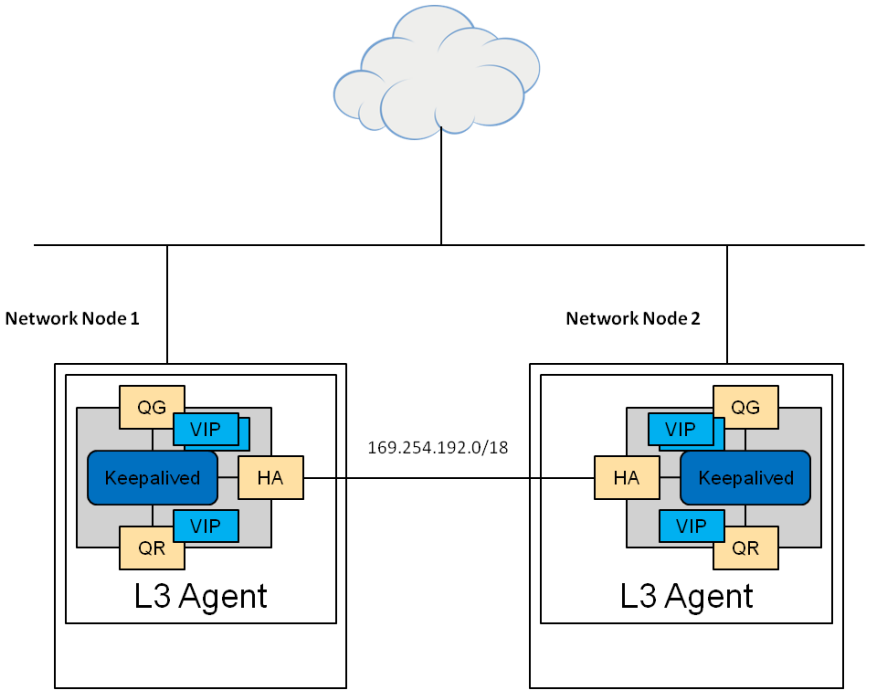

Fig 24. Natively HA principle of L3 agents

In Fig 25, when vRouter 1 in Node1 is down which can be caused by software failure or hardware failure, the Keepalived will detect the failure and the standby will take over to be active. In order to keep the TCP connection, Conntrackd is used to maintain the TCP sessions going through the router. One instance of conntrackd per virtual router, then one per namespace. After then, a rescheduling procedure will be triggered to respawn the failed virtual router to another l3 agent as standby. All the workflows is depicted in Fig 26.

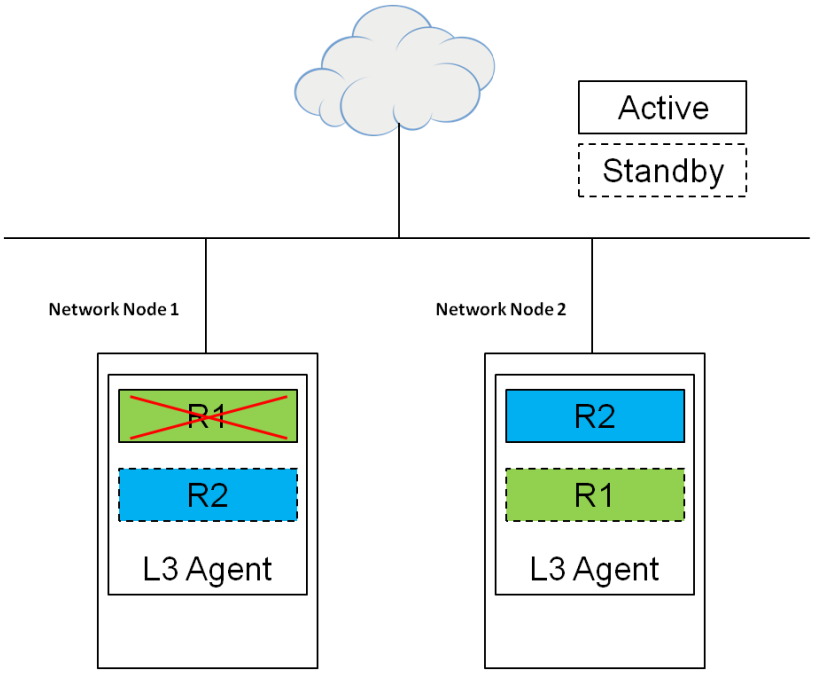

Fig 25. Failure of L3 agents

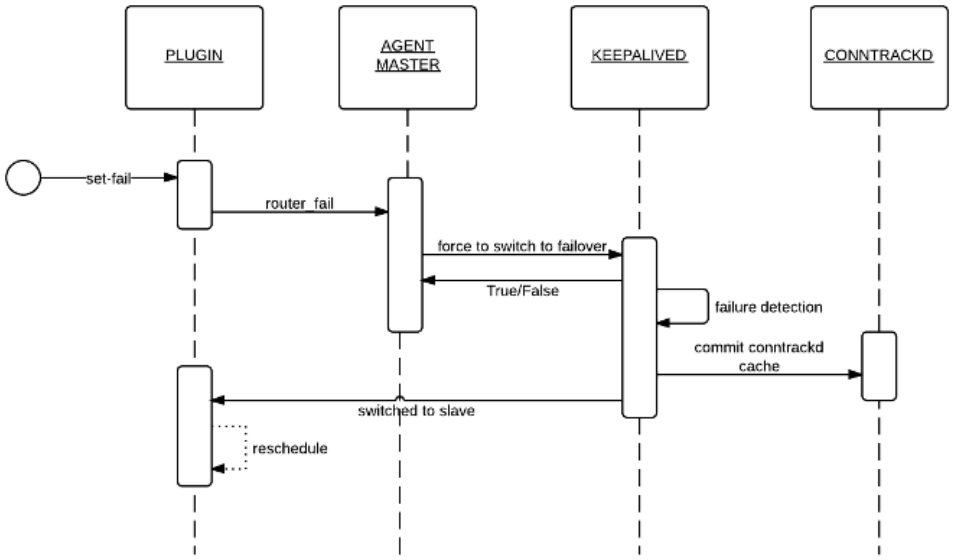

Fig 26. HA workflow of L3 agents

4.4 LBaaS agent and Metadata agent

Currently, no native feature is provided to make the LBaaS agent highly available using the default plug-in HAProxy. A common way to make HAProxy highly available is to use Pacemaker.

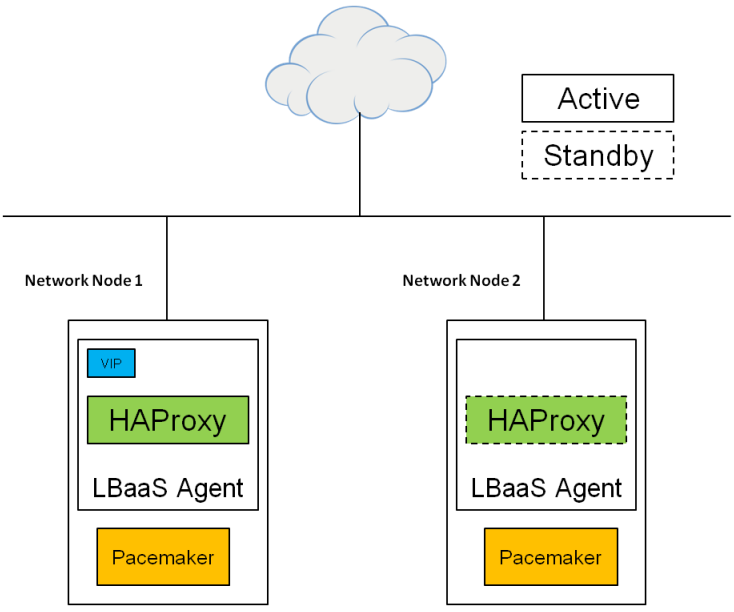

Fig 27. HA deployment of LBaaS agents using Pacemaker

As shown in Fig 27 HAProxy and pacemaker are deployed in both of the network nodes. The number of network nodes can be 2 or more. It depends on your cluster. HAProxy in Node 1 is the master and the VIP is in Node 1. Pacemaker monitors the liveness of HAProxy.

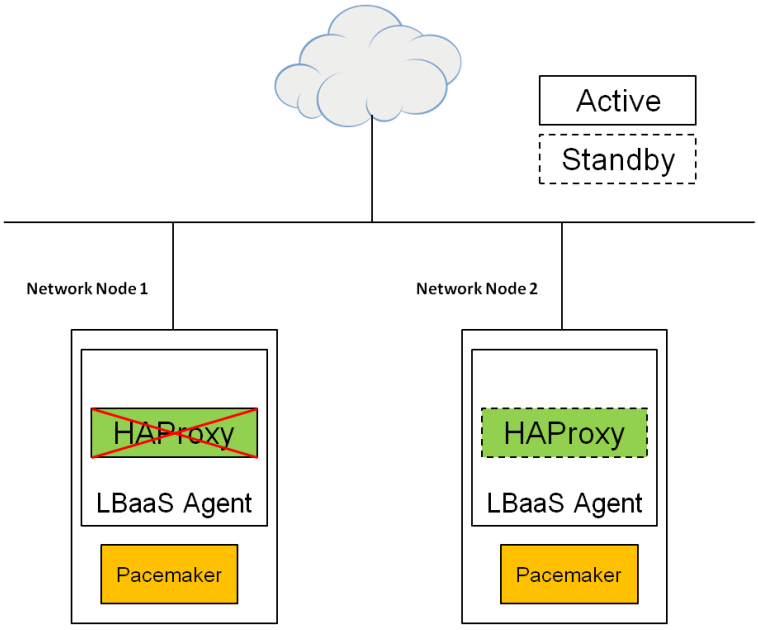

Fig 28. Failure of LBaaS agents

As shown in Fig 28 when HAProxy in Node1 falls down which can be caused by software failure or hardware failure, Pacemaker will fail over HAProxy and the VIP to Node 2.

Note that the default plug-in HAProxy only supports TCP and HTTP.

No native feature is available to make Metadata agent highly available. At this time, the Active/Passive solution exists to run the neutron metadata agent in failover mode with Pacemaker. The deployment and failover procedure can be the same as the case of LBaaS.

5 Storage and High Availability Scenarios

5.1 Elements of HA Storage Management and Delivery

Storage infrastructure, in any environment, can be broken down into two domains: Data Path and Control Path. Generally, High Availability of the storage infrastructure is measured by the occurrence of Data Unavailability and Data Loss (DU/DL) events. While that meaning is obvious as it relates to the Data Path, it is also applicable to Control Path as well. The inability to attach a volume that has data to a host, for example, can be considered a Data Unavailability event. Likewise, the inability to create a volume to store data could be considered Data Loss since it may result in the inability to store critical data.

Storage HA mechanisms are an integral part of most High Availability solutions today. In the first two sections below, we define the mechanisms of redundancy and

protection required in the infrastructure for storage delivery in both the Data and Control Paths. Storage services that have these mechanisms can be used in HA environments that are based on a highly available storage infrastructure.

In the third section below, we examine HA implementations that rely on highly available storage infrastructure. Note that the scope throughout this section is focused on local HA solutions. This does not address rapid remote Disaster Recovery scenarios that may be provided by storage, nor does it address metro active/active environments that implement stretched clusters of hosts across multiple sites for workload migration and availability.

5.2 Storage Failure & Recovery Scenarios: Storage Data Path

In the failure and recovery scenarios described below, a redundant network infrastructure provides HA through network-related device failures, while a variety of strategies are used to reduce or minimize DU/DL events based on storage system failures. This starts with redundant storage network paths, as shown in Fig 29.

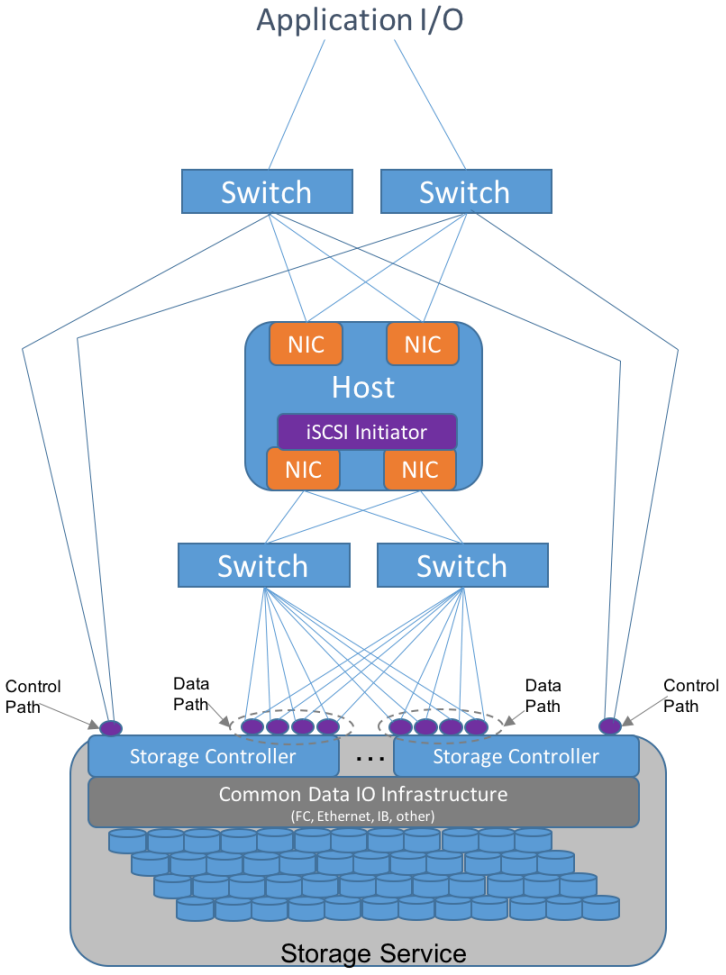

Fig 29.Typical Highly Available Storage Infrastructure

Storage implementations vary tremendously, and the recovery mechanisms for each implementation will vary. These scenarios described below are limited to 1) high level descriptions of the most common implementations since it is unpredictable as to which storage implementations may be used for NFVI; 2) HW- and SW-related failures (and recovery) of the storage data path, and not anything associated with user configuration and operational issues which typically create the most common storage failure scenarios; 3) non-LVM/DAS based storage implementations(managing failure and recovery in LVM-based storage for OpenStack is a very different scenario with less of a reliable track record); and 4) I will assume block storage only, and not object storage, which is often used for stateless applications (at a high level, object stores may include a subset of the block scenarios under the covers).

To define the requirements for the data path, I will start at the compute node and work my way down the storage IO stack and touch on both HW and SW failure/recovery scenarios for HA along the way. I will use Fig 29 as a reference.

1. Compute IO driver: Assuming iSCSI for connectivity between the compute and storage, an iSCSI initiator on the compute node maintains redundant connections to multiple iSCSI targets for the same storage service. These redundant connections may be aggregated for greater throughput, or run independently. This redundancy allows the iSCSI Initiator to handle failures in network connectivity from compute to storage infrastructure. (Fiber Channel works largely the same way, as do proprietary drivers that connect a host's IO stack to storage systems).

2. Compute node network interface controller (NIC): This device may fail, and said failure reported via whatever means is in place for such reporting from the host. The redundant paths between iSCSI initiators and targets will allow connectivity from compute to storage to remain up, though operating at reduced capacity.

3. Network Switch failure for storage network: Assuming there are redundant switches in place, and everything is properly configured so that two compute NICs go to two separate switches, which in turn go to two different storage controllers, then a switch may fail and the redundant paths between iSCSI initiators and targets allows connectivity from compute to storage to operational, though operating at reduced capacity.

4. Storage system network interface failure: Assuming there are redundant storage system network interfaces (on separate storage controllers), then one may fail and the redundant paths between iSCSI initiators and targets allows connectivity from compute to storage to remain operational, though operating at reduced performance. The extent of the reduced performance is dependent upon the storage architecture. See 3.5 for more.

5. Storage controller failure: A storage system can, at a very high level, be described as composed of network interfaces, one or more storage controllers that manage access to data, and a shared Data Path access to the HDD/SSD subsystem. The network interface failure is described in #4, and the HDD/SSD subsystem is described

in #6. All modern storage architectures have either redundant or distributed storage controller architectures. In **dual storage controller architectures**, high availability is maintained through the ALUA protocol maintaining access to primary and secondary paths to iSCSI targets. Once a storage controller fails, the array operates in (potentially) degraded performance mode until the failed storage controller is replaced. The degree of reduced performance is dependent on the overall original load on the array. Dual storage controller arrays also remain at risk of a Data Unavailability event if the second storage controller should fail. This is rare, but should be accounted for in planning support and maintenance contracts.

Distributed storage controller architectures are generally server-based, which may or may not operate on the compute servers in Converged Infrastructure environments. Hence the concept of "storage controller" is abstract in that it may involve a distribution of software components across multiple servers. Examples: Ceph and ScaleIO. In these environments, the data may be stored redundantly, and metadata for accessing the data in these redundant locations is available for whichever compute node needs the data (with authorization, of course). Data may also be stored using erasure coding (EC) for greater efficiency. The loss of a storage controller in this context leads to a discussion of impact caused by loss of a server in this distributed storage controller architecture. In the event of such a loss, if data is held in duplicate or triplicate on other servers, then access is simply redirected to maintain data availability. In the case of EC-based protection, then the data is simply re-built on the fly. The performance and increased risk impact in this case is dependent on the time required to rebalance storage distribution across other servers in the environment. Depending on configuration and implementation, it could impact storage access performance to VNFs as well.

6. HDD/SSD subsystem: This subsystem contains any RAID controllers, spinning hard disk drives, and Solid State Drives. The failure of a RAID controller is equivalent to failure of a storage controller, as described in 5 above. The failure of one or more storage devices is protected by either RAID parity-based protection, Erasure Coding protection, or duplicate/triplicate storage of the data. RAID and Erasure Coding are typically more efficient in terms of space efficiency, but duplicate/triplicate provides better performance. This tradeoff is a common point of contention among implementations, and this will not go into greater detail than to assume that failed devices do not cause Data Loss events due to these protection algorithms. Multiple device failures can potentially cause Data Loss events, and the risk of each method must be taken into consideration for the HA requirements of the desired deployment.

5.3 Storage Failure & Recovery Scenarios: Storage Control Path

As it relates to an NFVI environment, as proposed by OPNFV, there are two parts to

the storage control path.

- The storage system-specific control path to the storage controller
- The OpenStack-specific cloud management framework for managing different storage elements

5.3.1 Storage System Control Paths

High Availability of a storage controller is storage system-specific. Breaking it down to implementation variants is the best approach. However, both variants assume an IP-based management API in order to leverage network redundancy mechanisms for ubiquitous management access.

An appliance style storage array with dual storage controllers must implement IP address failover for the management API's IP endpoint in either an active/active or active/passive configuration. Likewise, a storage array with >2 storage controllers would bring up a management endpoint on another storage controller in such an event. Cluster-style IP address load balancing is also a viable implementation in these scenarios.

In the case of distributed storage controller architectures, the storage system provides redundant storage controller interfaces. E.g., Ceph's RADOS provides redundant paths to access an OSD for volume creation or access. In EMC's ScaleIO, there are redundant MetaData Managers for managing volume creation and access. In the case of the former, the access is via proprietary protocol, in the case of the latter, it is via HTTP-based REST API. Other storage implementations may also provide alternative methods, but any enterprise-class storage system will have built-in HA for management API access.

Finally, note that single server-based storage solutions, such as LVM, do not have HA solutions for control paths. If the server is failed, the management of that server's storage is not available.

5.3.2 OpenStack Controller Management

OpenStack cloud management is comprised of a number of different function-specific management modules such as Keystone for Identity and Access management (IAM), Nova for compute management, Cinder for block storage management, Swift for Object Storage delivery, Neutron for Network management, and Glance as an image repository. In smaller single-cloud environments, these management systems are managed in concert for High Availability; in larger multi-cloud environments, the Keystone IAM may logically stand alone in its own HA delivery across the multiple clouds, as might Swift as a common Object Store. Nova, Cinder, and Glance may have separate scopes of management, but they are more typically managed together as a logical cloud deployment.

It is the OpenStack deployment mechanisms that are responsible for HA deployment of these HA management infrastructures. These tools, such as Fuel, RDO, and others, have matured to include highly available implementations for the database, the API, and each of the manager modules associated with the scope of cloud management domains.

There are many interdependencies among these modules that impact Cinder high availability. For example:

- Cinder is implemented as an Active/Standby failover implementation since it requires a single point of control at one time for the Cinder manager/driver implementation. The Cinder manager/driver is deployed on two of the three OpenStack controller nodes, and one is made active while the other is passive. This may be improved to active/active in a future release.
- A highly available database implementation must be delivered using something like MySQL/Galera replication across the 3 OpenStack controller nodes. Cinder requires an HA database in order for it to be HA.
- A redundant RabbitMQ messaging implementation across the same three OpenStack controller nodes. Likewise, Cinder requires an HA messaging system.
- A redundant OpenStack API to ensure Cinder requests can be delivered.
- An HA Cluster Manager, like PaceMaker for monitoring each of the deployed manager elements on the OpenStack controllers, with restart capability. Keepalived is an alternative implementation for monitoring processes and restarting on alternate OpenStack controller nodes. While statistics are lacking, it is generally believed that the PaceMaker implementation is more frequently implemented in HA environments.

For more information on OpenStack and Cinder HA, see http://docs.openstack.org/ha-guide for current thinking.

While the specific combinations of management functions in these redundant OpenStack controllers may vary with the specific small/large environment deployment requirements, the basic implementation of three OpenStack controller redundancy remains relatively common. In these implementations, the highly available OpenStack controller environment provides HA access to the highly available storage controllers via the highly available IP network.

5.4 The Role of Storage in HA

In the sections above, we describe data and control path requirements and example implementations for delivery of highly available storage infrastructure. In summary:

- Most modern storage infrastructure implementations are inherently highly available. Exceptions certainly apply; e.g., simply using LVM for storage

presentation at each server does not satisfy HA requirements. However, modern storage systems such as Ceph, ScaleIO, XIV, VNX, and many others with OpenStack integrations, certainly do have such HA capabilities.

- This is predominantly through network-accessible shared storage systems in tightly coupled configurations such as clustered hosts, or in loosely coupled configurations such as with global object stores.

Storage is an integral part of HA delivery today for applications, including VNFs. This is examined below in terms of using storage as a key part of HA delivery, the possible scope and limitations of that delivery, and example implementations for delivery of such service. We will examine this for both block and object storage infrastructures below.

5.4.1 VNF, VNFC, and VM HA in a Block Storage HA Context

Several scenarios were described in another section with regard to managing HA at the VNFC level, with variants of recovery based on either VIM- or VNFM-based reporting/detection/recovery mechanisms. In a block storage environment, these differentiations are abstract and meaningless, regardless of whether it is or is not intended to be HA.

In a block storage context, HA is delivered via a logical block device (sometimes called a Logical Unit, or LUN), or in some cases, to a VM. VM and logical block devices are the units of currency.

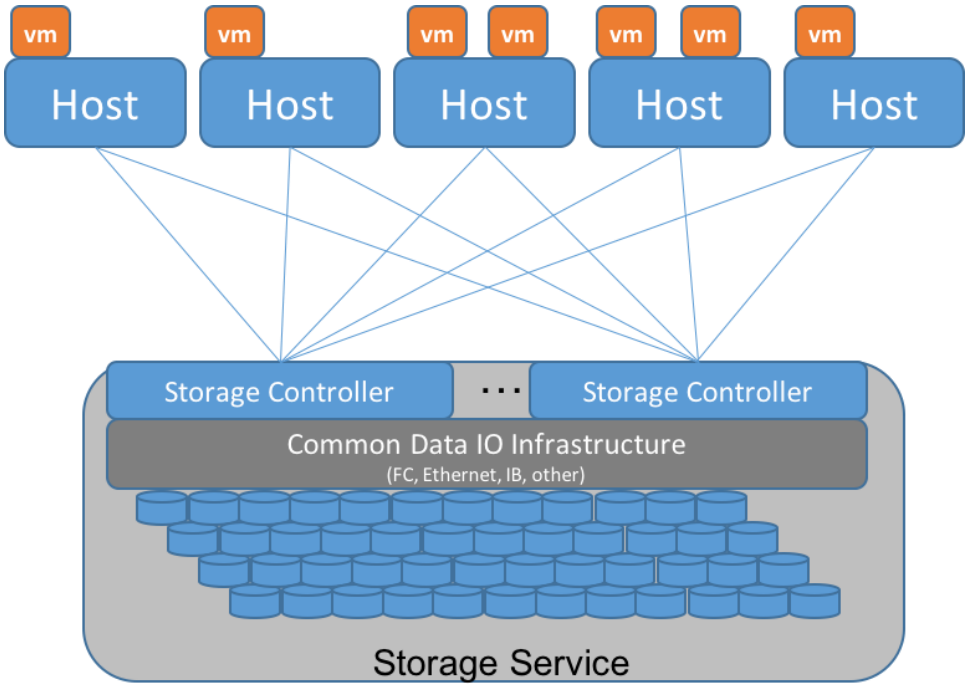

Fig 30. Typical HA Cluster With Shared Storage

In Fig 30, several hosts all share access, via an IP network or via Fibre Channel, to a common set of logical storage devices. In an ESX cluster implementation, these hosts all access all devices with coordination provided with the SCSI Reservation mechanism. In the particular ESX case, the logical storage devices provided by the storage service actually aggregate volumes (VMDKs) utilized by VMs. As a result, multiple host access to the same storage service logical device is dynamic. The vSphere management layer provides for host cluster management.

In other cases, such as for KVM, cluster management is not formally required, per se, because each logical block device presented by the storage service is uniquely allocated for one particular VM which can only execute on a single host at a time. In this case, any host that can access the same storage service is potentially a part of the "cluster". While *potential*access from another host to the same logical block device is necessary, the actual connectivity is restricted to one host at a time. This is more of a loosely coupled cluster implementation, rather than the tightly coupled cluster implementation of ESX.

So, if a single VNF is implemented as a single VM, then HA is provided by allowing that VM to execute on a different host, with access to the same logical block device and persistent data for that VM, located on the storage service. This also applies to multiple VNFs implemented within a single VM, though it impacts all VNFs together.

If a single VNF is implemented across multiple VMs as multiple VNFCs, then all of the VMs that comprise the VNF may need to be protected in a consistent fashion. The storage service is not aware of the distinction from the previous example. However, a higher level implementation, such as an HA Manager (perhaps implemented in a VNFM) may monitor and restart a collection of VMs on alternate hosts. In an ESX environment, VM restarts are most expeditiously handled by using vSphere-level HA mechanisms within an HA cluster for individual or collections of VMs. In KVM environments, a separate HA monitoring service, such as Pacemaker, can be used to monitor individual VMs, or entire multi-VM applications, and provide restart capabilities on separately configured hosts that also have access to the same logical storage devices.

VM restart times, however, are measured in 10's of seconds. This may sometimes meet the SAL-3 recovery requirements for General Consumer, Public, and ISP Traffic, but will never meet the 5-6 seconds required for SAL-1 Network Operator Control and Emergency Services. For this, additional capabilities are necessary.

In order to meet SAL-1 restart times, it is necessary to have: 1. A hot spare VM already up and running in an active/passive configuration 2. Little-to-no-state update requirements for the passive VM to takeover.

Having a spare VM up and running is easy enough, but putting that VM in an appropriate state to take over execution is the difficult part. In shared storage implementations for Fault Tolerance, which can achieve SAL-1 requirements, the VMs share access to the same storage device, and another wrapper function is used to

update internal memory state for every interaction to the active VM.

This may be done in one of two ways, as illustrated in Fig. 3. In the first way, the hypervisor sends all interface interactions to the passive as well as the active VM. The interaction is handled completely by hypervisor-to-hypervisor wrappers, as represented by the purple box encapsulating the VM in Fig 31, and is completely transparent to the VM. This is available with the vSphere Fault Tolerant option, but not with KVM at this time.

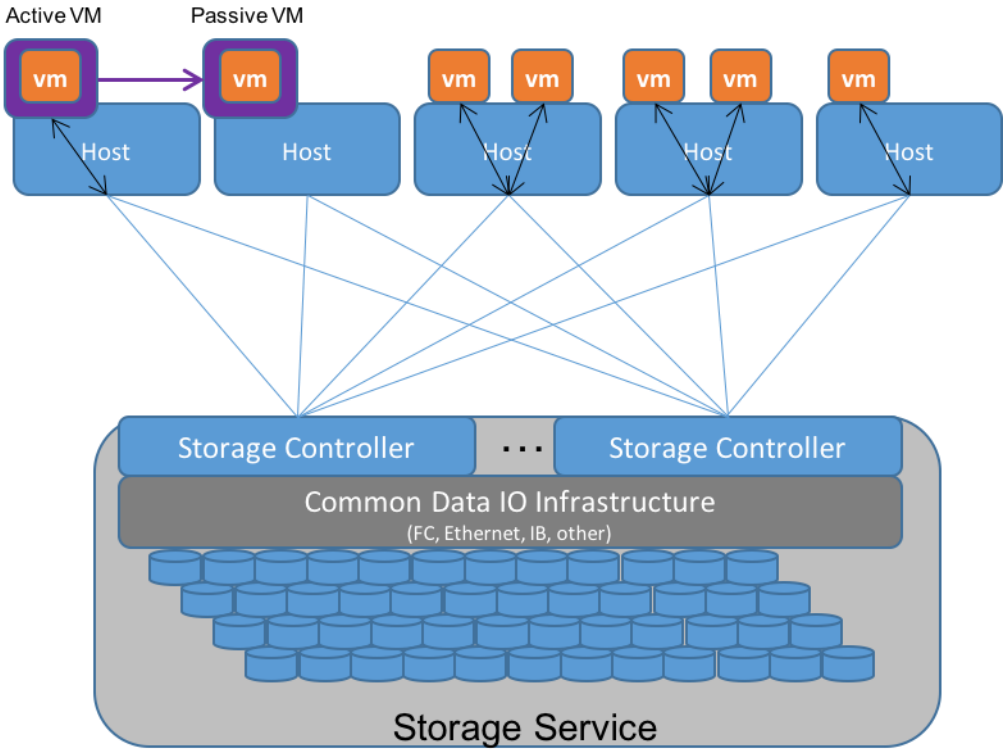

Fig 31. A Fault Tolerant Host/Storage Configuration

In the second way, a VM-level wrapper is used to capture checkpoints of state from the active VM and transfers these to the passive VM, similarly represented as the purple box encapsulating the VM in Fig 31. There are various levels of application-specific integration required for this wrapper to capture and transfer checkpoints of state, depending on the level of state consistency required. OpenSAF is an example of an application wrapper that can be used for this purpose. Both techniques have significant network bandwidth requirements and may have certain limitations and requirements for implementation.

In both cases, the active and passive VMs share the same storage infrastructure. Although the OpenSAF implementation may also utilize separate storage infrastructure as well (not shown in Fig 31).

Looking forward to the long term, both of these may be made obsolete. As soon as 2016, PCIe fabrics will start to be available that enable shared NVMe-based storage systems. While these storage systems may be used with traditional protocols like

SCSI, they will also be usable with true NVMe-oriented applications whose memory state are persisted, and can be shared, in an active/passive mode across hosts. The HA mechanisms here are yet to be defined, but will be far superior to either of the mechanisms described above. This is still a future.

5.4.2 HA and Object stores in loosely coupled compute environments

Whereas block storage services require tight coupling of hosts to storage services via SCSI protocols, the interaction of applications with HTTP-based object stores utilizes a very loosely coupled relationship. This means that VMs can come and go, or be organized as an N+1 redundant deployment of VMs for a given VNF. Each individual object transaction constitutes the duration of the coupling, whereas with SCSI-based logical block devices, the coupling is active for the duration of the VM's mounting of the device.

However, the requirement for implementation here is that the state of a transaction being performed is made persistent to the object store by the VM, as the restartable checkpoint for high availability. Multiple VMs may access the object store somewhat simultaneously, and it is required that each object transaction is made idempotent by the application.

HA restart of a transaction in this environment is dependent on failure detection and transaction timeout values for applications calling the VNFs. These may be rather high and even unachievable for the SAL requirements. For example, while the General Consumer, Public, and ISP Traffic recovery time for SAL-3 is 20-25 seconds, default browser timeouts are upwards of 120 seconds. Common default timeouts for applications using HTTP are typically around 10 seconds or higher (browsers are upward of 120 seconds), so this puts a requirement on the load balancers to manage and restart transactions in a timeframe that may be a challenge to meeting even SAL-3 requirements.

Despite these issues of performance, the use of object storage for highly available solutions in native cloud applications is very powerful. Object storage services are generally globally distributed and replicated using eventual consistency techniques, though transaction-level consistency can also be achieved in some cases (at the cost of performance). (For an interesting discussion of this, lookup the CAP Theorem.)

5.5 Summary

This section addressed several points:

- Modern storage systems are inherently Highly Available based on modern and reasonable implementations and deployments.
- Storage is typically a central component in offering highly available

infrastructures, whether for block storage services for traditional applications, or through object storage services that may be shared globally with eventual consistency.

- Cinder HA management capabilities are defined and available through the use of OpenStack deployment tools, making the entire storage control and data paths highly available.

6 Multisite Scenario

The Multisite scenario refers to the cases when VNFs are deployed on multiple VIMs. There could be three typical usecases for such scenario.

One is in one DC, multiple openstack clouds are deployed. Taking into consideration that the number of compute nodes in one openstack cloud are quite limited (nearly 100) for both opensource and commercial product of openstack, multiple openstack clouds will have to be deployed in the DC to manage thousands of servers. In such a DC, it should be possible to deploy VNFs accross openstack clouds.

Another typical usecase is Geographic Redundancy (GR). GR deployment is to deal with more catastrophic failures (flood, earthquake, propagating software fault, and etc.) of a single site. In the Geographic redundancy usecase, VNFs are deployed in two sites, which are geographically separated and are deployed on NFVI managed by separate VIM. When such a catastrophic failure happens, the VNFs at the failed site can failover to the redundant one so as to continue the service. Different VNFs may have specified requirement of such failover. Some VNFs may need stateful failover, while others may just need their VMs restarted on the redundant site in their initial state. The first would create the overhead of state replication. The latter may still have state replication through the storage. Accordingly for storage we don't want to loose any data, and for networking the NFs should be connected the same way as they were in the original site. We probably want also to have the same number of VMs on the redundant site coming up for the VNFs.

The other usecase is the maintenance. When one site is planning for a maintaining, it should first replicate the service to another site before it stops them. Such replication should not disturb the service, nor should it cause any data loss. In such case, the multisite schemes may be used.

The multisite scenario is also captured by the Multisite project, in which specific requirements of openstack are also proposed for different usecases. However, the multisite project mainly focuses on the requirement of these multisite usecases on openstack. HA requirements are not necessarily the requirement for the approaches discussed in multisite. While the HA project tries to capture the HA requirements in these usecases. The following links are the scenarios and Usecases discussed in the Multisite project.

https://gerrit.opnfv.org/gerrit/#/c/2123/ https://gerrit.opnfv.org/gerrit/#/c/1438/.

7 Concluding remarks

This scenario analysis document outlined the model and some failure modes for NFV systems. These are an initial list. The OPNFV HA project team is continuing to grow the list of scenarios and will issue additional documents going forward. The basic use cases and service availability considerations help define the key considerations for each use case taking into account the impact on the end service. The use case document along with the requirements documents and gap analysis help set context for engagement with various upstream projects.

Reference

- OpenStack HA guide: http://docs.openstack.org/ha-guide/networking-ha.html

- L3 High Availability VRRP:https://wiki.openstack.org/wiki/Neutron/L3_High_Availability_VRRP