T2 Network Hardware Decommissioning behavior for (VM-VM / VM-Disk) colocation scenarios

# Problem Statement

"Proximity Placement Group (VM-VM)" and "XIO (VM-Disk) colocation" scenarios – introduce new colocation constraints which require the containers to be pinned to a specific T1 / T2 network spine in case of "Proximity Placement Group" scenarios and are pinned to a specific T2 network spine in case "XIO colocation" scenarios. Proximity Placement Group scenarios ensure that all the VMs (containers) within the same Proximity Placement Group are pinned to the same T1 / T2 network spine. XIO colocation scenarios ensure that the VMs (containers) and their associated Premium disks are pinned to the same T2 network spine.

Now, there have been a few scenarios identified in which T2 Network Spine IDs (which Compute and Storage depend on for these colocation scenarios) can change:

* Hardware level network changes:
  + Two T2 sets are merged into one existing T2 set (SG and SYD)
  + Two small T2 sets are merged into one large (Tokyo)
  + One small T2 set will be replaced by a larger T2 set (India Central)
* Changes from “invalid” T2 IDs to the “correct” T2 IDs – since there can be bugs in the existing T2 ID values.

During these T2 Network Spine ID changes, there would be no Data Plane impact to the customers. Moreover, since these T2 Network Spine IDs are used extensively throughout the stack (e.g. Fabric, CRP, DiskRP and Storage) to ensure colocation guarantees – there arises a need for all of these components to handle this metadata change and ensure no customer downtime.

As a result, there arises a need for coming up with a new workflow / solution w.r.t handling metadata change of T2 network spine IDs during T2 network hardware decommissioning process.

# Goal

1. No data plane downtime for the customers during T2 network hardware decommissioning process.
2. Cover changes required (across all components) and customer experience w.r.t T2 network hardware upgrade scenario

# References

1. [Proximity Placement Groups PM Spec](https://microsoft.sharepoint.com/:w:/r/teams/AZCompute/_layouts/15/Doc.aspx?sourcedoc=%7BE35DB4B1-4142-4D55-AFE4-12CCDCF7591A%7D&file=Placement%20Groups%20-%20PM%20Spec.docx&action=edit&mobileredirect=true)
2. [VM-Disk Colocation PM PPT](https://microsoft-my.sharepoint.com/:p:/r/personal/chagarw_microsoft_com/_layouts/15/Doc.aspx?sourcedoc=%7BE31D5266-4E76-457D-9E0C-61E570753640%7D&file=Colocationv2.pptx&action=edit&mobileredirect=true)
3. [CRP Design Doc for PPG and VM-Disk Colocation](https://microsoft.sharepoint.com/:w:/r/teams/AzureIDC/AzureIDC_CRP/_layouts/15/Doc.aspx?sourcedoc=%7BCAA0A085-50DA-4FFB-8336-3EB253612175%7D&file=ColocationAllocatorActivity%20and%20ColocationPipeline%20Design%20Document.docx&action=default&mobileredirect=true)

# Approaches:

Approach 1:

* Fabric maintains the map of old T2 IDs vs latest T2 ID which Fabric obtains from NSM via either:
  1. A new API exposed via NSM which provides this map directly
  2. Inferring change of T2 ID based on the previous value and the new value obtained from NSM
* SRP (and thus in-turn DiskRP) exposes the details about old as well as the new T2 IDs via Get-RsrpFaultDomains Api taking into consideration of publishing appropriate scores of these T2 spines (based on the storage capacity distribution and ultimately returning 0 score for the older T2 ID)
* During the next refresh of computeStampCache, CRP retrieves new mapping from fabric as part of GetExtendedDataCenterUtilization or GetClusterInformation API (TBD by fabric) for each fabric cluster.
* During the next refresh of storageCapabilitiesCache, CRP retrieves new mapping from DiskRP as part of GetStorageCapabilities API.
* Till the time, all the stakeholders are not aware of the new mapping (Either DiskRP or all the fabric clusters are not aware of the new mapping or CRP side cache is yet to be refreshed), CRP will reconcile at its end and will keep sending the old T2 spine values for the allocation calls and they will keep honouring the old spine values as well.
* Once, CRP gets mapping from both Fabric as well as DiskRP, it knows that both stakeholders now have the knowledge of T2 spine change. Going forward, all the new tenant allocation calls will have new spine values.
* For existing tenant, during next update triggered on the VM (by the customer), CRP retrieves the container’s runtime network information – via ContainerNetwork packet, in which Fabric would return the old as well as the new T2 ID for that container.

**NOTE: There wouldn’t be a background thread / process which will pro-actively go and update customer’s VMs / Disks metadata to the newer T2 in CRP, due to the following reasons:**

* ***There is no real customer impact due to T2 migration scenario and hence we don’t want to pro-actively trigger a Fabric UpdateTenant call (which may get blocked probably due to some RootHE upgrade / etc.) and thus impacting the customer’s VM’s provisioning state.***
* ***There is currently no way to scan across all subscriptions in a region to find out all the impacted containers. Even if we find a way, it is a very costly operation and thus doesn’t seem to be worth it (less ROI), considering no real end-customer experience impact***
* Fabric would also return the information about why the T2 ID got changed for a container (for CRP to differentiate between Decom scenario vs. T2 network hardware upgrade scenario in order to show correct colocation status as part of GET calls) as part of container’s runtime network information.
* CRP would treat the change in the T2 ID as T2 Migration / T2 change scenario and perform the following:
  1. Update the NetworkSpineId of the disks allocated on the older T2 spine to the newer T2 spine
  2. Update the T2 spine of all the containers within that tenant from older T2 spine to the newer T2 spine.
* During the time period when the spine id is not updated for the container (due to opportunistic update based on customer’s request):
  1. Fabric would ensure that ServiceHealing of such containers do not relax the network spine constraint provided in the SVD but rather reallocate the container to the newer T2 spine instead.
  2. DiskRP and SRP would ensure that there is no impact to the disks/storage accounts running on the older T2.
* Since, for existing tenants, we are following the model of opportunistic update based on customer’s request, all the stakeholders will be required to maintain the mapping at their end indefinitely. As we will never know if all the existing tenants have been updated with new T2 spine values or not.

**Limitations with this approach:**

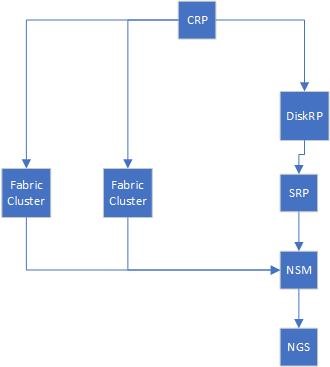
* There can be scenarios where CRP gets new mapping info from random partition in DiskRP but not all the partitions in DiskRP aware of the new spine mapping. As a result, even if cache from other components has new spine values , there may still occur scenarios where CRP may end up making allocation call on a DiskRP partition which does not know about new mapping resulting in allocation failure for customer.

# Approach 2:

**Without AzPubSub**:

The proposed "updated spine mapping info" propagation workflow dictates that all the stakeholders will get the new mapping info from their low level components.

Please refer below for clarity:



* 1. NGS makes the infra changes and persists the spine mapping info at its end indefinitely
  2. NSM polls the spine mapping info from NGS
  3. SRP polls the spine mapping info from NSM
  4. Fabric polls the spine mapping info from NSM
  5. DiskRP polls the spine mapping info from SRP
  6. 1. CRP polls the spine mapping info from both DiskRP as part of GetStorageCapabilities API
     2. CRP polls the spine mapping info from Fabric as part of GetExtendedDataCenterUtilization or GetClusterInformation API (TBD by fabric) for each fabric cluster

## Approach 2a:

**[Without AzPubSub] Timestamp based approach**

The intent of this approach is to rely on a given configurable time interval at CRP end to cater scenarios where not all the components (DiskRP, SRP, Fabric clusters) are yet aware of the new mapping info. In order to avoid any allocation failures that may arise due to DiskRP (which is part of a partition that does not yet have new mapping info) getting an allocation call with a new T2 spine value or skewed capacity scenarios where a fabric cluster hasn't yet refreshed its spine mapping info and as a result corresponding stampInfo recieved by CRP through GetDataCenterUtilization API has old T2 spine values along with their separate available capacities, CRP will keep sending the old T2 spine values as part of allocation calls until the time interval is over. After that, CRP will start sending the new T2 spine values as part of allocation calls.

Instead of waiting for a certain amount of time, we can have a config setting in Dynamic config stating whether to use the given spine’s new alias or not.

It will be a manual activity where CRP will first restrict the use of new mapping for some time through dynamic config and after checking that all the components have latest mapping, CRP will flip the switch allowing the use of new spine aliases.

Details:

* 1. Once CRP gets the new mapping from either DiskRP or Fabric, it will get the waiting period value from the config. (this value will be configurable through dynamic config). The intent behind this waiting period is to give time to all the stakeholders to update their respective spine mapping info. These include all the fabric clusters, DiskRP|SRP|CRP all partitions.
  2. Until the waiting period is over, all the allocation calls by CRP will have old T2 spine values. If any updated component gets old T2 spine value, it should continue to honour it.
  3. After the waiting period is over, CRP will start sending new T2 spine value as part of allocation calls.
  4. For edge case scenarios where any component was not able to update its spine mapping info until the waiting period expires (most likely some issue with the component), customer facing allocation failures can occur and manual intervention may be required. There needs to be monitoring mechanism in place which can flag such components so that they can be addressed.

Until all the components are aware of new mapping, following scenarios can occur from CRP POV:

|  |  |  |
| --- | --- | --- |
| **SpineMapping in CRP** | **Cache from other components (namely, storagecapabilities from DiskRP and computeStampInfo from Fabric)**  **These cache are required for colocation, allocation algorithms in CRP for choosing the best spine|cluster** | **Behaviour <any issues highlighted>** |
| Updated | Cache is in partial state i.e.,  Either Fabric cache is updated but DiskRP cache is not Updated and Vice Versa  Or all the fabric cluster level stampInfos are not updated with latest T2 spine value | From CRP POV, there will be 3 separate T2 spines values  Two older values and one new value. As a result, the available capacity alongwith other params can be divided into these three values.  This may result in selection of less optimal spine for sometime until waiting period is over.    One way to avoid this scenario could be to have reconciliation logic at CRP end to map the new spine values to old spine values during algorithm responsible for choosing the best spine but since this is not a customer facing issue and will be transient in nature. The current proposal is not to introduce any reconciliation at CRP end. |
| Updated | Cache at CRP end is refreshed but  All the partitions of DiskRP are not updated with new spine mapping info | [Issue]  There can be scenarios where CRP gets new mapping info from random partition in DiskRP but not all the partitions in DiskRP aware of the new spine mapping. As a result, even if cache from other components has new spine values , there may still occur scenarios where CRP may end up making allocation call on a DiskRP partition which does not know about new mapping resulting in allocation failure for customer.    Fix:  In order to avoid these scenarios, CRP will let the allocation algorithm run as is (without any mapping involvement) and if the final selected spine is the new T2 spine value, CRP will find the old T2 spine value from mapping and make subsequent allocate calls with the old spine value.  In case, there are multiple old T2 spine aliases for a given spine, CRP will choose the alias which supports premium storage. |
| Updated | Both Fabric and DiskRP cache are not yet refreshed | Since CRP is expecting to get mapping from either DiskRP or Fabric. This scenario is not possible with this approach. |
| NotUpdated | Both Fabric and DiskRP cache are not yet refreshed | CRP will continue to send old T2 spine values which will be honored by other components |
| NotUpdated | Cache is in partial state i.e.,  Either Fabric cache is updated but DiskRP cache is not Updated and Vice Versa  Or all the fabric cluster level stampInfos are not updated with latest T2 spine value | Since CRP is expecting to get mapping from either DiskRP or Fabric. This scenario is not possible with this approach. |
| NotUpdated | Cache at CRP end is refreshed but  All the partitions of DiskRP are not updated with new spine mapping info | Since CRP is expecting to get mapping from either DiskRP or Fabric. This scenario is not possible with this approach. |

Pros

* Since T2 migration will be a rare infra change scenario, having the TimeStamp approach avoids maintaining/honouring mapping versioning across all the components.

Cons

* Since, we are relying on time period to ensure all the DiskRP partitions have latest mapping. This fix will only be specific to spine mapping and will not fix the current stale cache issue in DiskRP

## Approach 2b:

**[Without AzPubSub] Spine versioning approach**

The intent of this approach is to introduce versioning for spine mappings. For each new T2 migration, the version will keep on increasing. For the intermediate scenarios where all the components are not yet on the latest version of mapping info. If any component receives an allocation call with new T2 spine value with new mappingVersion, if it does not have latest mapping info, it will force refresh its mapping info.

Details:

1. Once CRP gets the new mapping from both DiskRP and Fabric, It will start honoring the new T2 spine values immediately. As part of future allocation calls, CRP will also send mappingVersion which will be used by other components in order to check whether they have the latest mapping info by comparing the input mappingVersion with their current mappingVersion.
   * 1. If the input mappingVersion is less than current mappingVersion, keep honouring the old T2 spine value.
     2. If the input mappingVersion is greater than current mappingVersion, it means that component does not have latest mapping info and it will force refresh the mapping info. Since, CRP got the new spine mapping info from both Fabric as well as DiskRP, force refresh should result in latest mapping info.
     3. If the current mappingVersion is Null (i.e, the scenario for the first ever T2 migration, in this scenario the current mappingVersion will be null and input mappingVersion will be v1), the component will force refresh the mapping info.
2. For edge case scenarios where any component was not able to update its spine mapping info even after force refresh (most likely some issue with the component), customer facing allocation failures can occur and manual intervention may be required. There needs to be monitoring mechanism in place which can flag such components so that they can be addressed.

Until all the components are aware of new mapping, following scenarios can occur from CRP POV:

|  |  |  |
| --- | --- | --- |
| **SpineMapping in CRP** | **Cache from other components (namely, storagecapabilities from DiskRP and computeStampInfo from Fabric)**  **These cache are required for colocation, allocation algorithms in CRP for choosing the best spine|cluster** | **Behaviour <any issues highlighted>** |
| Updated | Cache is in partial state i.e.,  Either Fabric cache is updated but DiskRP cache is not Updated and Vice Versa  Or all the fabric cluster level stampInfos are not updated with latest T2 spine value | From CRP POV, there will be 3 separate T2 spines values  Two older values and one new value. As a result, the available capacity alongwith other params can be divided into these three values.  This may result in selection of less optimal spine for sometime until waiting period is over.    One way to avoid this scenario could be to have reconciliation logic at CRP end to map the new spine values to old spine values during algorithm responsible for choosing the best spine but since this is not a customer facing issue and will be transient in nature. The current proposal is not to introduce any reconciliation at CRP end. |
| Updated | Cache at CRP end is refreshed but  All the partitions of DiskRP are not updated with new spine mapping info | In case the DiskRP partition which does not have the new mapping info gets an allocation call with new T2 spine, DiskRP will force refresh the mapping and get the new mapping |
| Updated | Both Fabric and DiskRP cache are not yet refreshed | Since CRP is expecting to get mapping from either DiskRP or Fabric. This scenario is not possible with this approach. |
| NotUpdated | Both Fabric and DiskRP cache are not yet refreshed | CRP will continue to send old T2 spine values which will be honored by other components |
| NotUpdated | Cache is in partial state i.e.,  Either Fabric cache is updated but DiskRP cache is not Updated and Vice Versa  Or all the fabric cluster level stampInfos are not updated with latest T2 spine value | Since CRP is expecting to get mapping from either DiskRP or Fabric. This scenario is not possible with this approach. |
| NotUpdated | Cache at CRP end is refreshed but  All the partitions of DiskRP are not updated with new spine mapping info | Since CRP is expecting to get mapping from either DiskRP or Fabric. This scenario is not possible with this approach. |

Pros

* Spine versioning approach is more definitive in nature than timestamp based approach, also there will be no convoluted reconciliation logic required to map new spines back into old spines.

Cons

* NGS will need to maintain the mapping version at its end and other components will require to pass the mappingVersion as part of allocation calls resulting in possible TDPR for allocation.
* No guarantee that force refresh cache will succeed.

Approach 2c:

NGS provides the spine mapping sometime (1 day/rollout) prior to start returning new T2 spine related info. Since all the components get spine mapping in advance, they get time to sync new mapping in their respective cache (all the DiskRP partitions, SRP, Fabric, CRP, NSM). As a result, there will never be any scenario where any component starts getting new T2 spineId as part of allocation calls before it actually has new mapping info.

|  |  |  |
| --- | --- | --- |
| **SpineMapping in CRP** | **Cache from other components (namely, storagecapabilities from DiskRP and computeStampInfo from Fabric)**  **These cache are required for colocation, allocation algorithms in CRP for choosing the best spine|cluster** | **Behaviour <any issues highlighted>** |
| Updated | Cache is in partial state i.e.,  Either Fabric cache is updated but DiskRP cache is not Updated and Vice Versa  Or all the fabric cluster level stampInfos are not updated with latest T2 spine value | From CRP POV, there will be 3 separate T2 spines values  Two older values and one new value. As a result, the available capacity alongwith other params can be divided into these three values.  This may result in selection of less optimal spine for sometime until waiting period is over.    One way to avoid this scenario could be to have reconciliation logic at CRP end to map the new spine values to old spine values during algorithm responsible for choosing the best spine but since this is not a customer facing issue and will be transient in nature. The current proposal is not to introduce any reconciliation at CRP end. |
| Updated | Cache at CRP end is refreshed but  All the partitions of DiskRP are not updated with new spine mapping info | Not possible as we are ensuring that all the components have new mapping before |
| Updated | Both Fabric and DiskRP cache are not yet refreshed | Since CRP is expecting to get mapping from either DiskRP or Fabric. This scenario is not possible with this approach. |
| NotUpdated | Both Fabric and DiskRP cache are not yet refreshed | CRP will continue to send old T2 spine values which will be honored by other components |
| NotUpdated | Cache is in partial state i.e.,  Either Fabric cache is updated but DiskRP cache is not Updated and Vice Versa  Or all the fabric cluster level stampInfos are not updated with latest T2 spine value | Since CRP is expecting to get mapping from either DiskRP or Fabric. This scenario is not possible with this approach. |
| NotUpdated | Cache at CRP end is refreshed but  All the partitions of DiskRP are not updated with new spine mapping info | Since CRP is expecting to get mapping from either DiskRP or Fabric. This scenario is not possible with this approach. |

Pros

* No reconciliation logic required at components end in order to support scenarios where a component may end up sending new T2 spineId as part of allocation call to the component which does not have latest mapping.
* With the monitoring mechanism in place, we can preemptively check for issues where a particular component does not get latest mapping even after cache refresh.

Cons

* T2 migration workflow will be a phased approach for NGS [Phase1: Update the mapping info | Phase2: Start sending the new T2 spineId]
* The force referesh of DiskRP cache is still required as storage capabilities are still outdated

Approach 2d: **[Accepted Approach]**

**[Without AzPubSub] Force Refresh Cache in DiskRP with a configurable failover at CRP end**

[Open Question] [DiskRP] –

Can you please provide details regarding the DiskRP behaviour for the current stale cache scenarios.

For e.g.,

Currently, if a new T2 spine is added in the infrastructure, SRP makes a config change and roll out the config change

Then, DiskRP will refresh its cache as per its polling interval and update its cache with new T2 info

Since, each DiskRP partition will refresh its cache during its own polling interval, that means there may occur scenario where all the diskRP partitions are not aware of new T2 spine info.

CRP refreshes its storagecapabilites cache from a random DiskRP partition which has new T2 spine info.

During this race condition, if CRP ends up making allocateDisk call with new T2 spine to DiskRP which is not yet aware of new T2 spine.

[As mentioned by Animesh] – Currently, DiskRP will fail the allocateDisks call. DiskRP can have force refresh cache mechanism to cater these issues.

One more such scenario:

SRP team updated the FD count for a particular T2 spine (changes the supported FD count from 2 to 3).

During the same race condition, CRP may end up calling allocateDisk call with this updated FD count T2 spine on a DiskRP partition which doesn't yet know about the updated FD count info.

Since, for T2 migration as well, we are facing the similar stale cache issue.

It would be great if we can check the existing behaviour of DiskRP for the above mentioned scenarios and how do we currently address them.

[DiskRP] – DiskRP will have a force refresh mechanism in place to figure out the current cache is outdated and hence, force refresh the cache.

Since, the stale cache in a partition is a generic problem, instead of having a T2 migration specific logic, intent is to have a generic logic to handle these cases.

Details:

* 1. Once CRP gets the new mapping from both DiskRP and Fabric, it will also get the updated metadata from both components (namely, DataCenterUtilization from Fabric and Storage Capabilites from DiskRP).
  2. As CRP now has new T2 spine related metadata, CRP may end up choosing the new T2 spine as the spine to collocate resources to.
  3. If a DiskRP partition receives an allocate disks request with the new spineId but it has not yet refreshed its cache, it will force refresh its cache and will honour the allocate Disks request.
  4. For certain scenarios where CRP may end up sending old T2 spine as part of allocate calls even if both the components have latest mapping, they will continue to honour the old T2 spines as well.
  5. For existing tenants, as per the discussion, we are going ahead with not updating the metadata at all. Since, the mapping will have to exist indefinitely, the existing tenants will continue to be honoured.
  6. Fabric will keep honouring the mapping in case of scenarios like SH, Update tenant, new deployment.
  7. For scenarios, where the force refresh of DiskRP returned in stale cache, CRP will have a configurable failsafe setting exposed via dynamic config which can be used to ensure that CRP does not send new T2 spine as part of allocate calls.
  8. CRP would treat the change in the T2 ID as T2 Migration / T2 change scenario and perform the following:

a. Update the NetworkSpineId of the disks allocated on the older T2 spine to the newer T2 spine

b. Update the T2 spine of all the containers within that tenant from older T2 spine to the newer T2 spine.

Until all the components are aware of new mapping, following scenarios can occur from CRP POV:

|  |  |  |
| --- | --- | --- |
| **SpineMapping in CRP** | **Cache from other components (namely, storagecapabilities from DiskRP and computeStampInfo from Fabric)**  **These cache are required for colocation, allocation algorithms in CRP for choosing the best spine|cluster** | **Behaviour <any issues highlighted>** |
| Updated | Cache is in partial state i.e.,  Either Fabric cache is updated but DiskRP cache is not Updated and Vice Versa  Or all the fabric cluster level stampInfos are not updated with latest T2 spine value | From CRP POV, there will be 3 separate T2 spines values  Two older values and one new value. As a result, the available capacity alongwith other params can be divided into these three values.  This may result in selection of less optimal spine for sometime until waiting period is over.    One way to avoid this scenario could be to have reconciliation logic at CRP end to map the new spine values to old spine values during algorithm responsible for choosing the best spine but since this is not a customer facing issue and will be transient in nature. The current proposal is not to introduce any reconciliation at CRP end. |
| Updated | Cache at CRP end is refreshed but  All the partitions of DiskRP are not updated with new spine mapping info | Since, DiskRP has force refresh cache mechanism. This scenario will be addressed. |
| Updated | Both Fabric and DiskRP cache are not yet refreshed | Since CRP is expecting to get mapping from either DiskRP or Fabric. This scenario is not possible with this approach. |
| NotUpdated | Both Fabric and DiskRP cache are not yet refreshed | CRP will continue to send old T2 spine values which will be honored by other components |
| NotUpdated | Cache is in partial state i.e.,  Either Fabric cache is updated but DiskRP cache is not Updated and Vice Versa  Or all the fabric cluster level stampInfos are not updated with latest T2 spine value | Since CRP is expecting to get mapping from either DiskRP or Fabric. This scenario is not possible with this approach. |
| NotUpdated | Cache at CRP end is refreshed but  All the partitions of DiskRP are not updated with new spine mapping info | Since CRP is expecting to get mapping from either DiskRP or Fabric. This scenario is not possible with this approach. |

For the first T2 migration activity, in order to ensure that there are no code/scenario related issues, CRP will do a slice by slice rollout for T2 migration using the configurable setting.

Cons

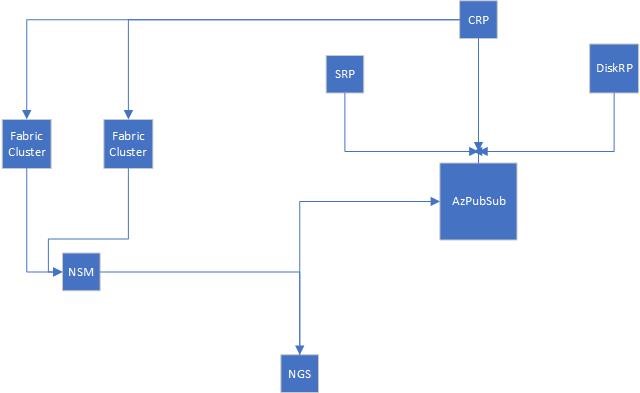
* Since, SRP is also partition based. There may be scenarios where force refresh of DiskRP cache can still result in stale cache.

# Approach 3:

**With AzPubSub:**

Instead of relying on immediate low level components for mapping propagation, NGS will publish the new mapping info on AzPubSub and the other components can subscribe to AzPubSub for getting new mapping info. The only exception to this approach will be Fabric which will continue to follow previous mentioned model. As discussed with Mandar [@Anurag Soni](mailto:Anurag.Soni@microsoft.com) , there is no immediate plan to switch on AzPubSub.

Please refer below for clarity:



* 1. NGS makes the infra changes and persists the spine mapping info at its end indefinitely
  2. NSM polls the spine mapping info from NGS
  3. Fabric polls the spine mapping info from NSM
  4. NGS publishes the spine mapping on AzPubSub
  5. CRP, DiskRP, SRP subscribes to the AzPubSub for getting the mapping

## Approach 3a:

**[With AzPubSub] Timestamp based approach**

* 1. Once CRP gets the new mapping , it will get the waiting period value from the config. (this value will be configurable through dynamic config). The intent behind this waiting period is to give time to all the stakeholders to update their respective spine mapping info. These include all the fabric clusters, DiskRP|SRP|CRP all partitions.
  2. Until the waiting period is over, all the allocation calls by CRP will have old T2 spine values. If any updated component gets old T2 spine value, it should continue to honour it.
  3. After the waiting period is over, CRP will start sending new T2 spine value as part of allocation calls.
  4. For edge case scenarios where any component was not able to update its spine mapping info until the waiting period expires (most likely some issue with the component), customer facing allocation failures can occur and manual intervention may be required. There needs to be monitoring mechanism in place which can flag such components so that they can be addressed.

Until all the components are aware of new mapping, following scenarios can occur from CRP POV:

|  |  |  |
| --- | --- | --- |
| **SpineMapping in CRP** | **Cache from other components (namely, storagecapabilities from DiskRP and computeStampInfo from Fabric)**  **These cache are required for colocation, allocation algorithms in CRP for choosing the best spine|cluster** | **Behaviour <any issues highlighted>** |
| Updated | Cache is in partial state i.e.,  Either Fabric cache is updated but DiskRP cache is not Updated and Vice Versa  Or all the fabric cluster level stampInfos are not updated with latest T2 spine value | From CRP POV, there will be 3 separate T2 spines values  Two older values and one new value. As a result, the available capacity alongwith other params can be divided into these three values.  This may result in selection of less optimal spine for sometime until waiting period is over.    One way to avoid this scenario could be to have reconciliation logic at CRP end to map the new spine values to old spine values during algorithm responsible for choosing the best spine but since this is not a customer facing issue and will be transient in nature. The current proposal is not to introduce any reconciliation at CRP end. |
| Updated | Cache at CRP end is refreshed but  All the partitions of DiskRP are not updated with new spine mapping info | [Issue]  There can be scenarios where CRP gets new mapping info from random partition in DiskRP but not all the partitions in DiskRP aware of the new spine mapping. As a result, even if cache from other components has new spine values , there may still occur scenarios where CRP may end up making allocation call with new spine value on a DiskRP partition which does not know about new mapping resulting in allocation failure for customer.    Fix:  In order to avoid these scenarios, CRP will let the allocation algorithm run as is (without any mapping involvement) and if the final selected spine is the new T2 spine value, CRP will find the old T2 spine value from mapping and make subsequent allocate calls with the old spine value.  In case, there are multiple old T2 spine aliases for a given spine, CRP will choose the alias which supports premium storage. |
| Updated | Both Fabric and DiskRP cache are not yet refreshed | CRP will continue to send old T2 spine values which will be honored by other components |
| NotUpdated | Both Fabric and DiskRP cache are not yet refreshed | CRP will continue to send old T2 spine values which will be honored by other components |
| NotUpdated | Cache is in partial state i.e.,  Either Fabric cache is updated but DiskRP cache is not Updated and Vice Versa  Or all the fabric cluster level stampInfos are not updated with latest T2 spine value | From CRP POV, there will be 3 separate T2 spines values  Two older values and one new value. As a result, the available capacity alongwith other params can be divided into these three values.  This may result in selection of less optimal spine for sometime until waiting period is over.    One way to avoid this scenario could be to have reconciliation logic at CRP end to map the new spine values to old spine values during algorithm responsible for choosing the best spine but since this is not a customer facing issue and will be transient in nature. The current proposal is not to introduce any reconciliation at CRP end. |
| NotUpdated | Cache at CRP end is refreshed but  All the partitions of DiskRP are not updated with new spine mapping info | [Issue]  There can be scenarios where CRP gets new mapping info from random partition in DiskRP but not all the partitions in DiskRP aware of the new spine mapping. As a result, even if cache from other components has new spine values , there may still occur scenarios where CRP may end up making allocation call with new spine value on a DiskRP partition which does not know about new mapping resulting in allocation failure for customer.    Since CRP does not yet have the new mapping, there is no suitable fix for this. |

Pros

* With AzPubSub, all the components (except fabric) will get the mapping info directly from the source of truth i.e., NGS
* Since T2 migration will be a rare infra change scenario, having the TimeStamp approach avoids maintaining/honoring mapping versioning across all the components.

Cons

* Currently, not all the components are integrated with AzPubSub and as mentioned above Fabric is not planning to integrate AzPubSub for its communication with NSM/NGS in near future. For the other components as well, given the timeline (next T2 migration for JapanEast is planned in July), as mentioned by Steve, it is highly unlikely that these timelines will shift further. Also, considering that the “Without AzPubSub” approach will require no extra API than “With AzPubSub” approach, it seems more feasible to go with “Without AzPubSub” approach.

## 

## Approach 3b:

**[With AzPubSub] Spine versioning approach**

1. Once CRP gets the new mapping from both DiskRP and Fabric, It will start honoring the new T2 spine values immediately. As part of future allocation calls, CRP will also send mappingVersion which will be used by other components in order to check whether they have the latest mapping info by comparing the input mappingVersion with their current mappingVersion.
   * 1. If the input mappingVersion is less than current mappingVersion, keep honouring the old T2 spine value.
     2. If the input mappingVersion is greater than current mappingVersion, it means that component does not have latest mapping info and it will force refresh the mapping info. Since, CRP got the new spine mapping info from both Fabric as well as DiskRP, force refresh should result in latest mapping info.
     3. If the current mappingVersion is Null (i.e, the scenario for the first ever T2 migration, in this scenario the current mappingVersion will be null and input mappingVersion will be v1), the component will force refresh the mapping info.
2. For edge case scenarios where any component was not able to update its spine mapping info even after force refresh (most likely some issue with the component), customer facing allocation failures can occur and manual intervention may be required. There needs to be monitoring mechanism in place which can flag such components so that they can be addressed.

Until all the components are aware of new mapping, following scenarios can occur from CRP POV:

|  |  |  |
| --- | --- | --- |
| **SpineMapping in CRP** | **Cache from other components (namely, storagecapabilities from DiskRP and computeStampInfo from Fabric)**  **These cache are required for colocation, allocation algorithms in CRP for choosing the best spine|cluster** | **Behaviour <any issues highlighted>** |
| Updated | Cache is in partial state i.e.,  Either Fabric cache is updated but DiskRP cache is not Updated and Vice Versa  Or all the fabric cluster level stampInfos are not updated with latest T2 spine value | From CRP POV, there will be 3 separate T2 spines values  Two older values and one new value. As a result, the available capacity alongwith other params can be divided into these three values.  This may result in selection of less optimal spine for sometime until waiting period is over.    One way to avoid this scenario could be to have reconciliation logic at CRP end to map the new spine values to old spine values during algorithm responsible for choosing the best spine but since this is not a customer facing issue and will be transient in nature. The current proposal is not to introduce any reconciliation at CRP end. |
| Updated | Cache at CRP end is refreshed but  All the partitions of DiskRP are not updated with new spine mapping info | In case the DiskRP partition which does not have the new mapping info gets an allocation call with new T2 spine, DiskRP will force refresh the mapping and get the new mapping |
| Updated | Both Fabric and DiskRP cache are not yet refreshed | CRP will continue to send old T2 spine values which will be honored by other components |
| NotUpdated | Both Fabric and DiskRP cache are not yet refreshed | CRP will continue to send old T2 spine values which will be honored by other components |
| NotUpdated | Cache is in partial state i.e.,  Either Fabric cache is updated but DiskRP cache is not Updated and Vice Versa  Or all the fabric cluster level stampInfos are not updated with latest T2 spine value | From CRP POV, there will be 3 separate T2 spines values  Two older values and one new value. As a result, the available capacity alongwith other params can be divided into these three values.  This may result in selection of less optimal spine for sometime until waiting period is over.    One way to avoid this scenario could be to have reconciliation logic at CRP end to map the new spine values to old spine values during algorithm responsible for choosing the best spine but since this is not a customer facing issue and will be transient in nature. The current proposal is not to introduce any reconciliation at CRP end. |
| NotUpdated | Cache at CRP end is refreshed but  All the partitions of DiskRP are not updated with new spine mapping info | In case the DiskRP partition which does not have the new mapping info gets an allocation call with new T2 spine, DiskRP will force refresh the mapping and get the new mapping |

Pros

* With AzPubSub, all the components (except fabric) will get the mapping info directly from the source of truth i.e., NGS

Cons

* NGS will need to maintain the mapping version at its end and other components will require to pass the mappingVersion as part of allocation calls.
* Currently, not all the components are integrated with AzPubSub and as mentioned above Fabric is not planning to integrate AzPubSub for its communication with NSM/NGS in near future. For the other components as well, given the timeline (next T2 migration for JapanEast is planned in July), as mentioned by Steve, it is highly unlikely that these timelines will shift further. Also, considering that the “Without AzPubSub” approach will require no extra API than “With AzPubSub” approach, it seems more feasible to go with “Without AzPubSub” approach

Approach 3c:

Same as Approach 2c with AzPubSub as delivery mechanism

# Possible changes required across components

## NGS

**TO BE FILLED BY NGS Team**

## NSM

**TO BE FILLED BY NSM Team**

## Fabric

* *Retrieve and maintain “old vs. new” T2 IDs for each Node*

1. NGS / NSM keeps the data of change in the T2 IDs (old vs. new) and exposes a new API “Changed T2”, which will be used by Fabric to poll these changes periodically and maintain old vs. new T2 values in its state.

**Pros**: Explicit API which ensures a stricter contract on when it is T2 migration scenario

**Cons**: (1) Doesn’t take care of “invalid” to “correct” T2 value change scenario. (2) A new API requirement on NSM/NGS

1. Fabric infers the change in “T2 IDs” based on the value already present in its state and the new value retrieved from NSM and maintains old vs. new values in its state.

**Pros**: (1) Takes care of all scenarios of T2 ID values changing (T2 migration as well as “invalid” to “correct” T2 ID value change) (2) No extra API requirements from NSM/NGS.

**Cons**: In case of any bugs due to which T2 IDs change, then also we will treat those scenarios as T2 Migration scenarios.

* *Return “old as well as new” T2 IDs for each container as part of ContainerNetwork packet and a new way for CRP to identify on the scenario when these T2 IDs change (decom vs. t2 migration)*

*[@Anurag Soni](mailto:Anurag.Soni@microsoft.com) - Can you please check and update if any more changes are required in fabric?*

* *Update ServiceHealing workflow to ensure that for this scenario, the T2 constraint is not relaxed.*
* *Allow update of the T2 IDs in the SVD for a running container from old T2 to new T2.*

[@An](mailto:vikramd@microsoft.com)urag – Can you pls. check and update if more changes would be required in Fabric?

## CRP

* *Retrieve the ContainerNetwork packet and infer T2 change scenario and update the T2 ID for these containers as well as their disks to the newer T2 ID during the update on those containers.*
* *During update on any VM within a PPG, we would update the PPG’s pinned T2 spine to the newer T2 spine*

## DiskRP

* *Allow changes in NetworkSpineId of disks in this scenario i.e., allow the change in network spine id (from old t2 spine to new t2 spine) for a disk attached to a running VM*
* *Change the NetworkSpineId pinning of the storage account on which those disks are allocated to the newer T2 spine id.*

[@Vikram](mailto:vikramd@microsoft.com) / [@Animesh](mailto:Animesh.Podar@microsoft.com) / [@Cosmin](mailto:Cosmin.Corbea@microsoft.com) – Can you pls. check and update if more changes would be required in DiskRP?

## SRP

* *Rollout to include the newer T2 spine id, since SRP has a static configuration for the spine information.*
* *Expose both old as well as new T2 Spine IDs as part of “StorageCapabilities” API*
* *Expose capability to allow updating the pinning of spine id for existing storage accounts.*

[@Chris](mailto:Chris.Ashton@microsoft.com) / [@Anthony](mailto:Anthony.Kunnel@microsoft.com) – Can you pls. check and update if more changes would be required in SRP?

# Open Items

Check with NGS team to confirm old name will never be used again

Keep persisitig the mapping indefinitely

SRP has to keep sending old mapping indefinitely

1. **[NGS/NSM]** - When does NGS / NSM start returning the newer T2 ID – during what phase of the migration? - Do they return the T2 ID mapping first and then start returning the new T2 as per part of other metadata. Or Do they start returning the new T2 mapping and new T2 as part of other metadata at the same time?
2. **[NGS]** - Since we are considering AzPubSub as delivery mechanism in few approaches, does NGS has currently workflow in place to be able to broadcast the new spine mapping via AzPubSub?
3. **[NGS/NSM]** - Does NGS / NSM return both old as well as new T2 IDs during any phase of the migration as part of the API that Fabric calls?
4. **[Fabric/NSM/NGS]** – Close on the approach of whether a new API to return the old <-> new T2 ID map is required.
5. **[Fabric/NSM/NGS]** – Close on whether there is a need to keep a list of old T2 IDs.
6. **[SRP/NSM/NGS]** - SRP would rely on the current way of having a static config and during / post migration will require a rollout to add the new T2 ID in their configs. Is this behaviour acceptable?

**Possible issues with continuing with static config approach:**

* + If there are scenarios where Fabric has been updated to report the newer T2 spine id – where as Storage has not been updated yet, then there can be issues with CRP not choosing this T2 spine id for any allocations. This can lead to skewness in the capacity
  + NSM/NGS have mentioned that these T2 spine ids can change due to bugs as well – and in those cases, depending on this being static data may not help, since it may require a hotfix rollout on SRP side to fix and mitigate the issue.

1. **[Fabric/CRP]** - Fabric exposes a new packet to CRP which contains the old <-> new T2 ID map. Is this required?
2. **[DiskRP/CRP]** – DiskRP needs to allow change in NetworkSpineId of the disks attached to a running VM.
3. **[SRP/DiskRP]** – Storage supports changing the spine of a storage account, and thus if during this call if DiskRP updates the spine information of the storage account from older to newer T2 spine – what happens to the disks allocated on that storage account which belong to a different VM (which from CRP and DiskRP’s POV still are allocated to the older T2 spine id)?

Various scenarios between CRP/DiskRP:

* CRP has new mapping (from fabric) but DiskRP is still returning old spines as part of GetStorageCapabilities because it does not have mapping info yet.

CRP will continue sending old t2 spine as part of allocatedisks API

* CRP does NOT have new mapping (from fabric) but DiskRP starts returning new spines as part of GetStorageCapabilities

CRP will continue sending old t2 spine as part of allocatedisks API and DiskRP will keep honoring the old t2 spine provided by CRP

* CRP does NOT have new mapping (from fabric) but DiskRP starts returning new spines for partial storage skus as part of GetStorageCapabilities as well as new mapping.

CRP will continue sending old t2 spines as part of allocatedisks API and DiskRP will keep honoring the old t2 spine provided by CRP

* CRP has new mapping (from fabric as well as DiskRP) but DiskRP starts returning new spines for few of the skus and old spines for rest of the skus.

CRP will continue sending old t2 spines as part of allocatedisks API and DiskRP will keep honoring the old t2 spines provided by CRP

* CRP has new mapping (from fabric as well as DiskRP) and DiskRP starts returning new spines for all the skus as part of GetStorageCapabilites

CRP will start sending new t2 spines as part of allocatedisks API

* How to validate these changes?

TBD

Various scenarios between CRP/Fabric:

1. Fabric to go through the IMOS rehydration workflow for T2 migration
2. To ensure that mapping always comes before any calls. Explore the feasibility of NSM providing mapping information in advance
3. There are two workflows through which Fabric can provide the mapping info to CRP
   1. As part of DataCenterUtilization API
   2. As part of GetClusterInformation API

Currently, the frequency of cache refresh of GetClusterInformation from Fabric in CRP is 6 hours. As a result, for six hours CRP for new tenants in ppg will fail as CRP will not be having the mapping.

CRP to check if we can increase the frequency of cache refresh to 10 minutes as well make sure GetClusterInformation cache refresh before DataCenterUtilization calls.

Fabric to verify the TDP impact for this as the number of GetClusterInformation calls will increase.

Fabric to check the feasibility for providing spine mapping as part of DataCenterUtilization call.

1. For new tenant scenario, there can be cases where CRP can send old T2 spine as part of new tenant workflow as well. Fabric to honor old T2 spine for new tenant workflows as well. (Refer point 6&7)
2. For existing tenant scenario, CRP will be opportunistic and will update SVD with new T2 spine during the first update tenant after mapping info reaches CRP.
3. For scenarios where not all the related fabric clusters have the new T2 spine or all the related fabric clusters have new T2 spine but not all the storage clusters have the new T2 spine, proposal is to keep using old T2 spine until all the related fabric clusters have new T2 spine and only start using the new T2 spine when all the related fabric as well storage clusters have new T2 spine. If we continue to use the values as is i.e., the clusters which have new T2 spine and the clusters which have old T2 spine as is then there will occur allocation failures which may further result in sev3/sev2
4. For scenario where UtilizationInfo as part of computestampcache  at CRP side is still not updated to reflect new T2 spine (As cache is refreshed every 10 minutes, there may be a race condition where till cache gets refreshed, all the new tenant/existing tenant calls to fabric will contains old T2 spine). For this as well, Fabric needs to honour old T2 spine for both new tenant as well existing tenant scenarios.

* When partial set of compute clusters report on the new T2

(Refer point 6)

* Partial set of storage clusters report on the new T2

(Refer point 6)

* When CRP sends in a score/allocation request using the old T2

Fabric will continue to honor old T2 spine for these as there are few scenarios where this may occur.

(Refer point 6)

* When CRP sends down the score/allocation request using new T2

Once utilization info in CRP is refreshed, these requests will be for new T2

* When service healing happens and the cluster has moved to the new t2

FSH should honor old T2 spine

* When all of the above scenarios happen with AzSM in between fabric and CRP

Fabric to go through AzSM workflow for T2 migration.

* How do we validate these changes? What environment?

TBD

CRP-Fabric Contract

As part of **GetExtendedDatacenterUtilization** API:

Currently it returns ExtendedUtilizationInfo

public class ExtendedUtilizationInfo

{

public UtilizationInfo ClusterUtilizationInfo

public Dictionary<string, UtilizationInfo> T1UtilizationInfo

public Dictionary<string, UtilizationInfo> T2UtilizationInfo

}

Proposed changes:

As part of T2UtilizationInfo, fabric will start returning the list of aliases for that spine.

CRP will parse the aliases for all the spines and create dictionary at its end for further use.

public class UtilizationInfo

{

public IEnumerable<String> ~~T2NetworkSpineAliases~~ Aliases

{

get;

set;

}

}

As part of **PacketType.ContainerNetwork** returned among other packets from **GetTenantInformation** API:

Currently the packet returns:

public class ContainerNetworkPacket : Packet

{

public override PacketType PacketType => PacketType.ContainerNetwork;

public IPAddress CustomerIPAddress

{

get;

set;

}

public ICollection<NetworkInterfaceInfo> SecondaryInterfaces

{

get;

set;

}

public string T1NetworkSpineId

{

get;

set;

}

public string T2NetworkSpineId

{

get;

set;

}

public string RequestedT1NetworkSpine

{

get;

set;

}

public string RequestedT2NetworkSpine

{

get;

set;

}

public override string ToString()

{

return $"{base.ToString()}, CustomerIPAddress='{CustomerIPAddress}', SecondaryInterfaces='{SecondaryInterfaces}', T1NetworkSpineId='{T1NetworkSpineId}', T2NetworkSpineId='{T2NetworkSpineId}', RequestedT1NetworkSpine='{RequestedT1NetworkSpine}', RequestedT2NetworkSpine='{RequestedT2NetworkSpine}'";

}

}

Proposed changes:

Approach 1:

public class ContainerNetworkPacket : Packet

{

public override PacketType PacketType => PacketType.ContainerNetwork;

public IPAddress CustomerIPAddress

{

get;

set;

}

public ICollection<NetworkInterfaceInfo> SecondaryInterfaces

{

get;

set;

}

public string T1NetworkSpineId

{

get;

set;

}

public string T2NetworkSpineId

{

get;

set;

}

public string RequestedT1NetworkSpine

{

get;

set;

}

public string RequestedT2NetworkSpine

{

get;

set;

}

public List<String> T2NetworkSpineAliases

{

get;

set;

}

public override string ToString()

{

return $"{base.ToString()}, CustomerIPAddress='{CustomerIPAddress}', SecondaryInterfaces='{SecondaryInterfaces}', T1NetworkSpineId='{T1NetworkSpineId}', T2NetworkSpineId='{T2NetworkSpineId}', RequestedT1NetworkSpine='{RequestedT1NetworkSpine}', RequestedT2NetworkSpine='{RequestedT2NetworkSpine}'";

}

}

Approach 2:

public class ContainerNetworkPacket : Packet

{

public override PacketType PacketType => PacketType.ContainerNetwork;

public IPAddress CustomerIPAddress

{

get;

set;

}

public ICollection<NetworkInterfaceInfo> SecondaryInterfaces

{

get;

set;

}

public string T1NetworkSpineId

{

get;

set;

}

public string T2NetworkSpineId

{

get;

set;

}

public string RequestedT1NetworkSpine

{

get;

set;

}

public string RequestedT2NetworkSpine

{

get;

set;

}

public bool IsT2MigrationScenario

{

get;

set;

}

public override string ToString()

{

return $"{base.ToString()}, CustomerIPAddress='{CustomerIPAddress}', SecondaryInterfaces='{SecondaryInterfaces}', T1NetworkSpineId='{T1NetworkSpineId}', T2NetworkSpineId='{T2NetworkSpineId}', RequestedT1NetworkSpine='{RequestedT1NetworkSpine}', RequestedT2NetworkSpine='{RequestedT2NetworkSpine}'";

}

}

Approach 3:

public class ContainerNetworkPacket : Packet

{

public override PacketType PacketType => PacketType.ContainerNetwork;

public IPAddress CustomerIPAddress

{

get;

set;

}

public ICollection<NetworkInterfaceInfo> SecondaryInterfaces

{

get;

set;

}

public string T1NetworkSpineId

{

get;

set;

}

public string T2NetworkSpineId

{

get;

set;

}

public string RequestedT1NetworkSpine

{

get;

set;

}

public string RequestedT2NetworkSpine

{

get;

set;

}

public *NetworkSpineMismatchType* T2MismatchReason

{

get;

set;

}

public *NetworkSpineMismatchType* T1MismatchReason

{

get;

set;

}

public override string ToString()

{

return $"{base.ToString()}, CustomerIPAddress='{CustomerIPAddress}', SecondaryInterfaces='{SecondaryInterfaces}', T1NetworkSpineId='{T1NetworkSpineId}', T2NetworkSpineId='{T2NetworkSpineId}', RequestedT1NetworkSpine='{RequestedT1NetworkSpine}', RequestedT2NetworkSpine='{RequestedT2NetworkSpine}'";

}

}

Enum of the Mismatch reason:

*public enum NetworkSpineMismatchType*

*{*

*NoMismatch,*

*Decommisioning,*

*NetworkSpineMigration*

*};*

CRP-DiskRP Contract

Current StorageCapabilities data structure returned by GetStorageCapabilities API of DiskRP:

public class StorageCapabilities

{

public StorageAvailabilitySetSpecifications availabilitySetSpecifications;

/// <summary>

/// The name of the variable is misleading (as zone and availability set do not go together) but

/// changing the name would require a co-ordinated change with DiskRP.

/// </summary>

public StorageAvailabilityZoneSpecifications zoneAndAvailabilitySetSpecifications;

}

The network spine id is specified inside StorageSpine

/// <summary>

/// Specifications for StorageSKUs supported in the given region

/// </summary>

public class StorageAvailabilityZoneSpecifications

{

/// <summary>

/// Specifications for every sku available in the region.

/// </summary>

public List<StorageSku> skus;

}

public class StorageSku

{

/// <summary>

/// Storage account type of the SKU

/// </summary>

public StorageAccountType accountType;

/// <summary>

/// Network spines of the SKU

/// </summary>

public List<StorageSpine> spines { get; set; }

}

public class StorageSpine

{

/// <summary>

/// E.g. "spine\_id\_0".

/// </summary>

public string name { get; set; }

/// <summary>

/// These are physical zones.

/// Note:

/// A network spine can contain multiple physical zones.

/// A physical zone can be in multiple spines.

/// </summary>

public List<StoragePhysicalZone> zones { get; set; }

/// <summary>

/// Fault domain count for this spine.

/// </summary>

public int faultDomainCount { get; set; }

/// <summary>

/// This is health indication of a spine.

/// </summary>

public int score { get; set; }

/// <summary>

/// This will contain a list of scores for each fault domain configuration.This allows Storage to reduce the score for certain configurations within a spine while permitting others.

/// For instance, Storage may set score=0 for faultDomainCount=3 which will cause CRP to filter this spine out for avset allocations with FD=3. However, with non-zero scores

/// for faultDomainCount=2 & faultDomainCount=1, we continue to recommend allocations for availability sets with FDs < 3.

/// </summary>

public List<AvailabilitySetScore> availabilitySetScores { get; set; }

}

Following are the approaches in which the existing data structures can be modified to return the T2 network spine id mappings. Waiting for contract between DiskRP-SRP to know how SRP will provide these mappings, based on that will decide what would be the best approach to provide them to CRP.

### Approach 1

Provide the dictionary of new T2 spine names and their all the aliases (old). It should be nullable.

public class StorageCapabilities

{

public StorageAvailabilitySetSpecifications availabilitySetSpecifications;

/// <summary>

/// The name of the variable is misleading (as zone and availability set do not go together) but

/// changing the name would require a co-ordinated change with DiskRP.

/// </summary>

public StorageAvailabilityZoneSpecifications zoneAndAvailabilitySetSpecifications;

/// <summary>

/// This is wrt T2 migration. This dictionary contains current T2 spines in the region as keys and all the old names for that T2 spine in all the T2 migrations.

/// </summary>

public Dictionary<string, HashSet<string>> t2NetworkSpineAliases;

}

### Approach 2

Provide aliases in the StorageSpine class. Here the info could be redundant. Ensure the set is nullable.

public class StorageSpine

{

/// <summary>

/// E.g. "spine\_id\_0".

/// </summary>

public string name { get; set; }

/// <summary>

/// These are physical zones.

/// Note:

/// A network spine can contain multiple physical zones.

/// A physical zone can be in multiple spines.

/// </summary>

public List<StoragePhysicalZone> zones { get; set; }

/// <summary>

/// Fault domain count for this spine.

/// </summary>

public int faultDomainCount { get; set; }

/// <summary>

/// This is health indication of a spine.

/// </summary>

public int score { get; set; }

/// <summary>

/// This will contain a list of scores for each fault domain configuration.This allows Storage to reduce the score for certain configurations within a spine while permitting others.

/// For instance, Storage may set score=0 for faultDomainCount=3 which will cause CRP to filter this spine out for avset allocations with FD=3. However, with non-zero scores

/// for faultDomainCount=2 & faultDomainCount=1, we continue to recommend allocations for availability sets with FDs < 3.

/// </summary>

public List<AvailabilitySetScore> availabilitySetScores { get; set; }

/// <summary>

/// This is wrt T2 migration. This list contains all the old names for this T2 spine in all the T2 migrations.

/// </summary>

public HashSet<string> aliases { get; set; }

}