Composition Service

Redfish composability provides a data model to describe composable hardware, as well as an interface for clients to manage their composition. Composition Service is the top-level resource for all composability related resources. Within the Composition Service we can find the inventory of Resource Blocks, Resource Zones and Collection Capabilities. Resource Blocks contains inventory of all components that can be used for composability. Resource Zones are descriptors containing the binding restrictions of the different components. Collection Capabilities are annotations that describe how to form composition requests.

ODIM composition service implements the Redfish composition model. The composition service provides APIs and framework to build the composable infrastructure using ODIM resources. Initially, Composition service offers the features like resource blocks and resource zones. Any supported actions on these resources will be added. Composition service requires additional API support from other redfish resources to create a composition, update resources, and delete the composition.

Create a Composed Resource: Redfish Composability model has defined Specific Composition, Constrained Composition. ODIM composition service includes all composition models, in phase-by-phase releases.

Update a composed Resource: Composition service enables updating an existing composition like adding a new resource or removing a attached resource etc.

Delete a composed Resource: Composition service retire or decompose an already composed resource.

References:

Redfish Composition Service Mockup

Redfish Schema Supplement

Redfish v1.12 Specification

API	URI	HTTP Methods	Remarks
Composition Service	/redfish/v1/CompositionService	GET	
Resource Blocks	/redfish/v1/CompositionService /ResourceBlocks	GET, POST	POST to add a resource block manually. Not in Redfish specification.
Resource Zones	/redfish/v1/CompositionService /ResourceZones	GET, POST, PATCH, DELETE	

Dependencies

Composition Service depends on additional APIs to be supported in other ODIM modules. Listed down the APIs required for Computer system Composition. Later this list will be extended for other composable resources.

API	URI	HTTP Methods	Remarks	Jeff H Comments
Computer system Composition	/redfish/v1/Systems	POST	ODIM Systems collection need to have this action	POST is the wrong answer. Ask DMTF to make it possible to insert and delete from the collection
Add resource block	/redfish/v1/Systems/{{id}}/Actions /ComputerSystem.AddResourceBlock	POST	System instance	
Remove resource block	/redfish/v1/Systems/{{id}}/Actions /ComputerSystem.RemoveResourceBlock	POST	Systems instance	
Delete a composed resource	/redfish/v1/Systems/{{id}}	DELETE	Resources will be deleted and available for next composition	

ODIM kind of aggregation services includes servers and hardware resources identified using different technologies and operates on a heterogeneous environment. Building the composition resources in a standard way is not sufficient. We are listing some of the real time use cases below, on the possible options to build the composition resources.

There are different ways we can build the composition resources, in a heterogeneous environment.

Use Case 1: API to add Composition resources.

Redfish has support to add resource blocks. Redfish specification does not support to add a resource block directly. ODIM Composition Service can exposes APIs to add resource blocks. Admin can add a resource block using this API.

We recommend that this shall be proposed as a feature enhancement to DMTF.

After the discussion in TSC review meeting, it is decided to add a POST action in resource blocks. Later, this enhancement will be proposed to DMTF Redfish Composability TF.

Computer System Resource Block - Sample

POST /redfish/v1/CompositionService/ResourceBlocks


```
{
   "Name": "ComputerSystemBlock",
   "Description": "Computer System Block",
   "ComputerSystems": [
      {
         "@odata.id": "/redfish/v1/Systems/System1"
      }
   ],
   "ResourceBlockType": "ComputerSystem",
   "ResourceBlockLimits": {
      "MaxComputerSystem": 1,
      "MinComputerSystem": 1
   },
   "CompositionStatus": {
      "CompositionState": "Unused",
      "MaxCompositions": 1,
      "NumberOfCompositions": 0,
      "Reserved": false,
      "SharingCapable": false,
      "SharingEnabled": false
   }
}
```


Storage Resource Block - Sample

POST /redfish/v1/CompositionService/ResourceBlocks


```
{
   "Name": "Drive Block",
   "Description": "Drive Block",
   "Drives": [
      {
         "@odata.id": "/redfish/v1/Chassis/chassis1/Drives/drive1"
      }
   ],
   "NetworkInterfaces": [],
   "SimpleStorage": [],
   "Storage": [],
   "ResourceBlockType": "Storage",
   "ResourceBlockLimits": {
      "MaxStorage": 1,
      "MinStorage": 1
   },
   "CompositionStatus": {
      "CompositionState": "Unused",
      "MaxCompositions": 3,
      "NumberOfCompositions": 0,
      "Reserved": false,
      "SharingCapable": true,
      "SharingEnabled": true
   }
}
```


Use Case 2: Build the composition resources automatically.

Composition service will have a back end service running that triggers the building of composition service resources like resource blocks and resource zones. For this Composition service need to be notified when there is a new server added to ODIM.

When a new server is added to ODIM, the composition service gets notified. Then composition service collect the new servers resource information and build the composition resources. There is a limitation here that the composition service does not recognize the underlying connectivity across the disaggregated resources. This could be helpful build the resource blocks and resource zones for the integrated resources like compute, processor, memory etc.

After discussion in TSC proposal review meeting, following suggestions provided. Composition service can build the resource blocks and allow administrators to create the resource zones and add the resource blocks.

Use Case 3: Utilizes composition resources from underlying services.

When a new server added to the ODIM, ODIM knows the underlying service (like redfish service) that provides the information. If the underlying service is redfish and exposing the composition service resources that can be consumed in the ODIM composition service.

We must check the possibilities of doing this, as composition service may not have visibility to underlying services that provides the resource data.

Development Plan

Composition Service will be implemented in python 3.x. It has following components.

1. Composition service - A back end service that builds and monitor the composition resources. Running as a service
2. API server - API server exposes the Composition service APIs. Running as a service
3. DBA library - Includes DB abstraction. Will be included in both applications.
4. Utility library - Provides helper functions needed. Included in both applications
5. Service scripts - To run the applications as a service
6. Build script - To build the environment and install services

Initial implementation of ODIM Composition service will have support for resource blocks, resource zones and computer system composition. The initial version shall add support to specific composition.

The constrained composition, and other resource composition will be added later versions. Also, future versions of composition service adopts new features added by DMTF Redfish Composability task force.

Questions to DMTF

1. Redfish Composition service schema is not allowing POST action for new resource blocks creation.

Resource block collection schema defines resource blocks are not insertable.

```
"deletable": false,
"insertable": false,
"updatable": false,
```

We should be able to support CRUD operations on the resource blocks.

To TSC:

2. System capabilities are global to the the system collection. ODIM collects resources from many redfish services that has its own capabilities. How to handle this in the multi vendor environment like ODIM.