Document Identifier: DSP2066

Date: 2025-12-04

Version: 1.1.0

Redfish Fabrics White Paper

Supersedes: 1.0.0

Document Class: Informational

Document Status: Published

Document Language: en-US

CONTENTS

Foreword

The Redfish Fabrics White Paper was prepared by DMTF's Redfish Forum.

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. For information about DMTF, see http://www.dmtf.org.

Acknowledgments

DMTF acknowledges the following individuals for their contributions to this document:

- Martin Halstead — Hewlett Packard Enterprise
- John Mayfield — Hewlett Packard Enterprise
- Slawek Putyrski — Intel Corporation
- Michael Raineri — Dell Technologies
- Shesha Sreenivasamurthy — Marvell Technology, Inc.

1 Introduction

Modern datacenters consist of hundreds to thousands of different server and storage resources used for different purposes. Often a portion of these resources are grouped into clusters that are used to work together on specific workloads. High-speed fabrics are typically used to transfer data among clustered resources to optimally satisfy the needs of the workload. Clustered resources within a datacenter can be used for a single workload or can be utilized for multiple workloads. In order to configure and manage clustered resources, Redfish has a common data model that describes fabrics to management clients. The fabric data model contains methods to configure, manage, and support the lifecycle of intra-fabric connectivity of clusters of resources.

The fabric data model describes the physical topology of the cluster, connectivity constraints, and isolation as well as zones to define sub-domains within a larger fabric. This paper describes the common fabric data model and provides examples for common types of fabrics.

2 Fabric representation

The `Fabrics` property found in the service root contains a set collection of `Fabric` resources. Each `Fabric` resource represents a fabric that is managed by the Redfish service and can contain the following information:

- `FabricType` : Describes the type of protocol sent over the fabric.
- `Switches` : Contains the switches and their connectivity information for the fabric. See Switches and Ports for more information.
- `Endpoints` : Contains the logical representation for devices on a fabric. See Endpoints for more information.
- `Zones` : Contains communication constraints for endpoints within a fabric. See Zones for more information.
- `Connections` : Contains rules for the types of resources an initiator endpoint is allowed to access when connecting to a target endpoint. See Connections for more information.
- `AddressPools` : Contains addressing rules for the fabric. See Address pools for more information.

Example `Fabric` resource:


```
{
    "@odata.id": "/redfish/v1/Fabrics/Ethernet",
    "@odata.type": "#Fabric.v1_4_0.Fabric",
    "Id": "Ethernet",
    "Name": "Ethernet Fabric",
    "FabricType": "Ethernet",
    "Description": "An Ethernet Based Fabric",
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    },
    "Zones": {
        "@odata.id": "/redfish/v1/Fabrics/Ethernet/Zones"
    },
    "Endpoints": {
        "@odata.id": "/redfish/v1/Fabrics/Ethernet/Endpoints"
    },
    "Switches": {
        "@odata.id": "/redfish/v1/Fabrics/Ethernet/Switches"
    },
    "AddressPools": {
        "@odata.id": "/redfish/v1/Fabrics/Ethernet/AddressPools"
    }
}
```


The value of the `FabricType` property contains information about the modelled fabric technology. The `FabricType` property can contain the following values.

Value	Description
`CXL`	A fabric compliant with the Compute Express Link Specification.
`Ethernet`	A fabric compliant with the IEEE 802.3 Ethernet Specification.
`FC`	A fabric compliant with the T11 Fibre Channel Physical and Signaling Interface Specification.
`GenZ`	A fabric compliant with the Gen-Z Core Specification.
`InfiniBand`	A fabric compliant with the InfiniBand Architecture Specification.
`iSCSI`	A fabric compliant with the IETF Internet Small Computer Systems Interface (iSCSI) Specification.
`NVMeOverFabrics`	A fabric compliant with the the NVM Express over Fabrics Specification.
`PCIe`	A fabric compliant with the PCI-SIG PCI Express Base Specification.
`SAS`	A fabric compliant with the T10 SAS Protocol Layer Specification.

3 Endpoints

The `Endpoint` resource represents an addressable entity, a single device, or a set of devices where traffic enters or exits a fabric. Traffic on a fabric flows from one endpoint to another endpoint, and the configuration of the fabric dictates routing and addressability rules for the endpoints. Systems, switches, adapters, and other components connected to a fabric might contain multiple endpoints to represent the different signifying an ingress and egress points on a fabric.

Endpoints can represent varying types of entities on a fabric, some physical and some logical. The `ConnectedEntities` property describes the type of entity on the fabric and how it links to other areas of the Redfish model. `ConnectedEntities` is an array to allow for multiple Redfish resources to be referenced if they work together to form an endpoint on a fabric. Each member of the `ConnectedEntities` array can contain the following properties:

Property	Description
`EntityType`	The type of the entity on the fabric. See the next table for more information on the values.
`EntityRole`	Describes whether the endpoint acts as an initiator, target, or both on the fabric. Initiator endpoints produce traffic to access resources exposed by target endpoints.
`EntityLink`	A link to another resource in the Redfish data model that provides more information about the endpoint. This property might not be present if the `Endpoint` resource represents an entity that is not managed by the service.
`Identifiers`	The globally unique identifier for the entity.

The value of the `EntityType` property gives guidance for the type of resource that can be found with the `EntityLink` property. The `EntityType` property can contain the following values.

Value	Description	`EntityLink` resource
`StorageInitiator`	A storage initiator	A member of `StorageControllers` in `Storage` or a `StorageController` resource
`RootComplex`	A PCI(e) root complex	`ComputerSystem`
`NetworkController`	A network controller	`NetworkDeviceFunction` or `EthernetInterface`
`Drive`	A drive	`Drive`

Value	Description	`EntityLink` resource
`StorageExpander`	A storage expander	`Chassis`
`DisplayController`	A display controller	N/A
`Bridge`	A PCI(e) bridge	N/A
`Processor`	A processor	`Processor`
`Volume`	A volume	`Volume`
`AccelerationFunction`	An acceleration function realized through a device, such as an FPGA.	`AccelerationFunction`
`MemoryChunk`	A memory chunk	`MemoryChunk`
`Switch`	A switch	`Switch`
`FabricBridge`	A fabric bridge	`FabricAdapter`
`Manager`	A manager	`Manager`
`StorageSubsystem`	A storage subsystem	`Storage`
`Memory`	A memory device	`Memory`
`CXLDevice`	A CXL Logical Device	`CXLLogicalDevice`

Endpoints with IP connectivity can report their IP addresses with the `IPTransportDetails` property.

Endpoints that represent a logical entity on a fabric can report an additional `Identifiers` property at the root of the resource to show the globally unique identifier for the endpoint.

The following example shows an `Endpoint` resource for a SAS drive. The `EndpointProtocol` property contains `SAS` to reflect the protocol accepted by the endpoint. `ConnectedEntities` references the `Drive` resource that the endpoint represents. `ConnectedPorts` within `Links` shows the switch ports to which the endpoint is connected.

```
{
    "@odata.id": "/redfish/v1/Fabrics/SAS/Endpoints/Drive1",
    "@odata.type": "#Endpoint.v1_8_2.Endpoint",
    "Id": "Drive1",
    "Name": "SAS Drive",
    "Description": "The SAS Drive in Enclosure 2 Bay 0",
    "EndpointProtocol": "SAS",
    "ConnectedEntities": [
        {
            "EntityType": "Drive",
            "EntityRole": "Target",
            "EntityLink": {
```

```
                "@odata.id": "/redfish/v1/Chassis/2/Drives/0"
            },
            "Identifiers": [
                {
                    "DurableNameFormat": "NAA",
                    "DurableName": "32ADF365C6C1B7C3"
                }
            ]
        }
    ],
    "Links": {
        "ConnectedPorts": [
            {
                "@odata.id": "/redfish/v1/Fabrics/SAS/Switches/Switch1/Ports/8"
            },
            {
                "@odata.id": "/redfish/v1/Fabrics/SAS/Switches/Switch2/Ports/8"
            }
        ]
    }
}
```

4 Connectivity

Redfish describes the physical topology of a fabric using the `Switch` and `Port` resources. The `Port` resources subordinate to the `Switch` resource show how a fabric switch provides connectivity to other devices in the infrastructure. Other resources that represent devices on a fabric, such as `FabricAdapter` and `NetworkAdapter`, contain their own `Port` resources to show connectivity to the fabric. These resources can be used to configure routing information, virtual channel information, virtual LAN information, and congestion information based on the customer's desired topology. A client can verify the topology and port map by following the links contained in the `ConnectedPorts` and `AssociatedEndpoints` properties from the various `Port` resources. Clients can discover these resources from the `Fabrics`, `Chassis`, and `System` properties off the service root. Figure 1 shows an example connectivity diagram with two fabrics named `Ethernet` and `Gen-Z`.

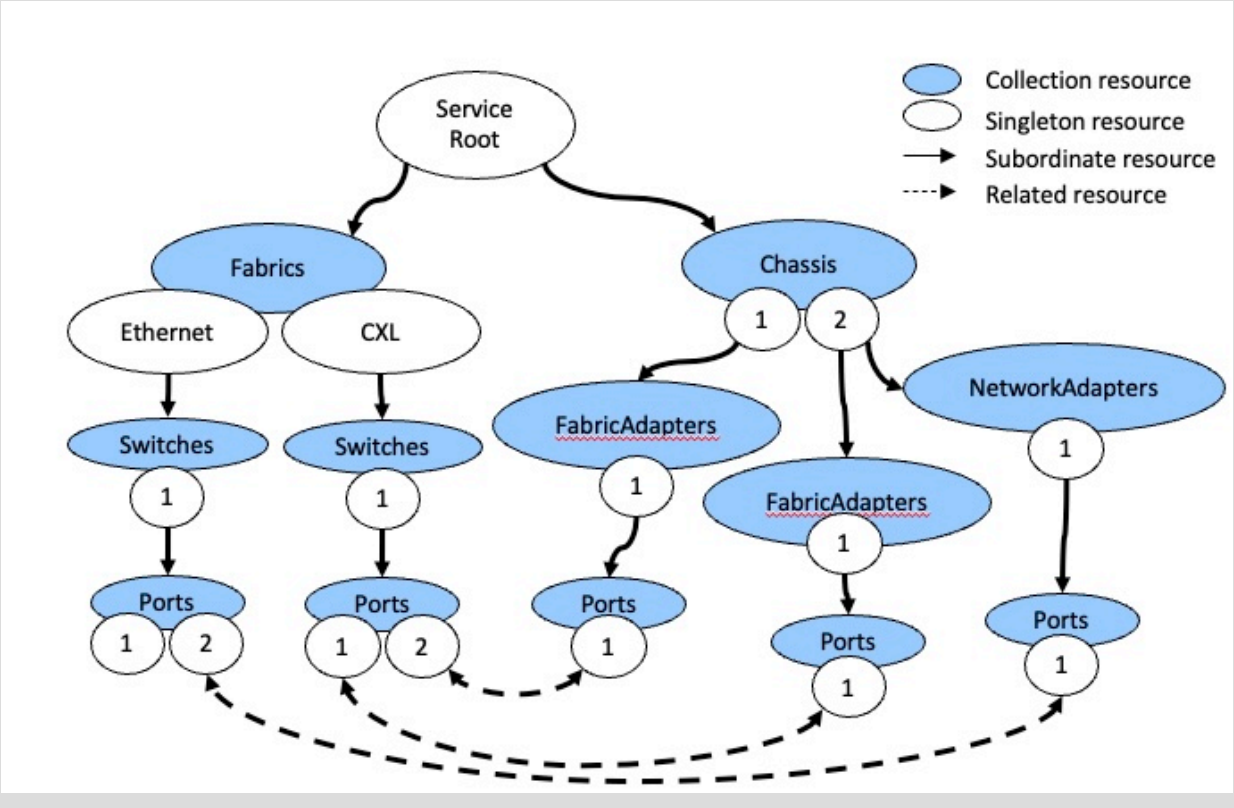

Figure 1 — Fabric connectivity example

4.1 Switches

The `Switch` resource represents a single, generic switch. It contains information about the switch, such as its manufacturer, model, and part number. It also provides a link to the collection of ports on the switch.

Example `Switch` resource:

```
{
    "@odata.id": "/redfish/v1/Fabrics/Ethernet/Switches/Switch1",
    "@odata.type": "#Switch.v1_10_0.Switch",
    "Id": "Switch1",
    "Name": "Ethernet Switch",
    "SwitchType": "Ethernet",
    "Manufacturer": "Contoso",
    "Model": "8320",
    "SKU": "67B",
    "SerialNumber": "2M220100SL",
    "PartNumber": "76-88883",
    "Ports": {
        "@odata.id": "/redfish/v1/Fabrics/Ethernet/Switches/Switch1/Ports"
    }
}
```

4.2 Ports

The `Port` resource represent the physical interface of an adapter or switch to a fabric. Resources such as `Switch`, `FabricAdapter`, `NetworkAdapter`, and `Processor` contain their own collection of `Port` resources to represent their respective interfaces for the device. The `Port` resource describes the port's attributes, such as the interface speed and link status.

The `Port` resource contains several properties within `Links` to describe how the port is connected to other ports or devices in a fabric. Clients can follow the following properties within `Links` for building the topology of a fabric.

Property	Description
`ConnectedSwitches`	Switches connected to the port.
`ConnectedSwitchPorts`	Switch ports connected to the port. The members should be subordinate to the switches found in `ConnectedSwitches`. If the connected port is a device port, use `ConnectedPorts` instead.
`ConnectedPorts`	Device ports connected to the port. The members should not reference ports on switches. If the connected port is a switch port, use `ConnectedSwitchPorts` instead.

Property	Description
`AssociatedEndpoints`	Endpoints connected to the port. This property can be used for fabrics where `Endpoint` resources represent physical devices and `ConnectedPorts` is not applicable.

Example `Port` resource:


```
{
    "@odata.id": "/redfish/v1/Fabrics/Ethernet/Switches/Switch1/Ports/1",
    "@odata.type": "#Port.v1_16_0.Port",
    "Id": "1",
    "Name": "Ethernet Port 1",
    "Description": "Ethernet Port 1",
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    },
    "PortId": "1",
    "PortProtocol": "Ethernet",
    "PortType": "BidirectionalPort",
    "CurrentSpeedGbps": 25,
    "Width": 4,
    "MaxSpeedGbps": 25,
    "Links": {
        "ConnectedPorts": [
            {
                "@odata.id": "/redfish/v1/Chassis/1/NetworkAdapters/3/Ports/1"
            }
        ]
    }
}
```


4.3 Port metrics

Many devices capture the metrics associated with their ports. For example, Ethernet devices can capture metrics such as received frames, transmitted broadcast frames, transmitted multicast frames, frame alignment errors, and collision errors. This provides clients an indication of the health of the port at a particular point in time. The `PortMetrics` resource contains these metrics.

Example `PortMetrics` resource:


```
{
    "@odata.type": "#PortMetrics.v1_7_0.PortMetrics",
    "RXBytes": 567027413,
```

```
    "TXBytes": 6620140961,
    "RXErrors": 0,
    "TXErrors": 0,
    "Networking": {
        "RXFrames": 3356710,
        "RXUnicastFrames": 3354717,
        "RXMulticastFrames": 204,
        "RXBroadcastFrames": 1789,
        "TXFrames": 14358417,
        "TXUnicastFrames": 14357401,
        "TXMulticastFrames": 670,
        "TXBroadcastFrames": 346,
        "RXDiscards": 498455,
        "RXFrameAlignmentErrors": 0,
        "RXFCSErrors": 0,
        "RXFalseCarrierErrors": 0,
        "RXOversizeFrames": 0,
        "RXUndersizeFrames": 0,
        "TXDiscards": 0,
        "TXExcessiveCollisions": 0,
        "TXLateCollisions": 0,
        "TXMultipleCollisions": 0,
        "TXSingleCollisions": 0,
        "RXPFCFrames": 0,
        "TXPFCFrames": 0,
        "RXPauseXOFFFrames": 0,
        "RXPauseXONFrames": 0,
        "TXPauseXOFFFrames": 0,
        "TXPauseXONFrames": 0,
        "RDMARXBytes": 0,
        "RDMARXRequests": 0,
        "RDMAProtectionErrors": 0,
        "RDMAProtocolErrors": 0,
        "RDMATXBytes": 0,
        "RDMATXRequests": 0,
        "RDMATXReadRequests": 0,
        "RDMATXSendRequests": 0,
        "RDMATXWriteRequests": 0
    },
    "Transceivers": [
        {
            "RXInputPowerMilliWatts": 0.06,
            "TXBiasCurrentMilliAmps": 49.01,
            "TXOutputPowerMilliWatts": 1.263,
            "SupplyVoltage": 4.21
        }
    ]
}
```

4.4 Network and fabric adapters

The `NetworkAdapter` and `FabricAdapter` resources contain a collection of `Port` resources to express physical connectivity to a fabric or other adapters. These adapters support specific types of fabrics. For example, a device on an Ethernet fabric would use a `NetworkAdapter` resource and a device on a CXL fabric would use a `FabricAdapter` resource. These adapters describe the physical endpoint of the node on the fabric along with the adapter capabilities.

The `FabricAdapter` resource provides additional fabric-related settings, such as routing information, congestion management, embedded switch configuration, and Virtual Channel/Traffic Class management.

Chassis-level and system-level connectivity is determined using the `Port` resources found on the chassis's or system's related adapters. The `ComputerSystem` resource and `Chassis` resource contain a `FabricAdapters` property to represent the set of available fabric adapters. The `Chassis` resource contains a `NetworkAdapters` property to represent the set of available network adapters. The `ComputerSystem` resource contains `NetworkInterfaces` property to represent the set of available partitions of network adapters.

5 Fabric configuration and routing

The `Zone`, `Connection`, and `AddressPool` resources model communication intent across fabrics. The following sections describe the relationships between these resources and the `Endpoint` resource for managing routing and other networking configurations for a fabric.

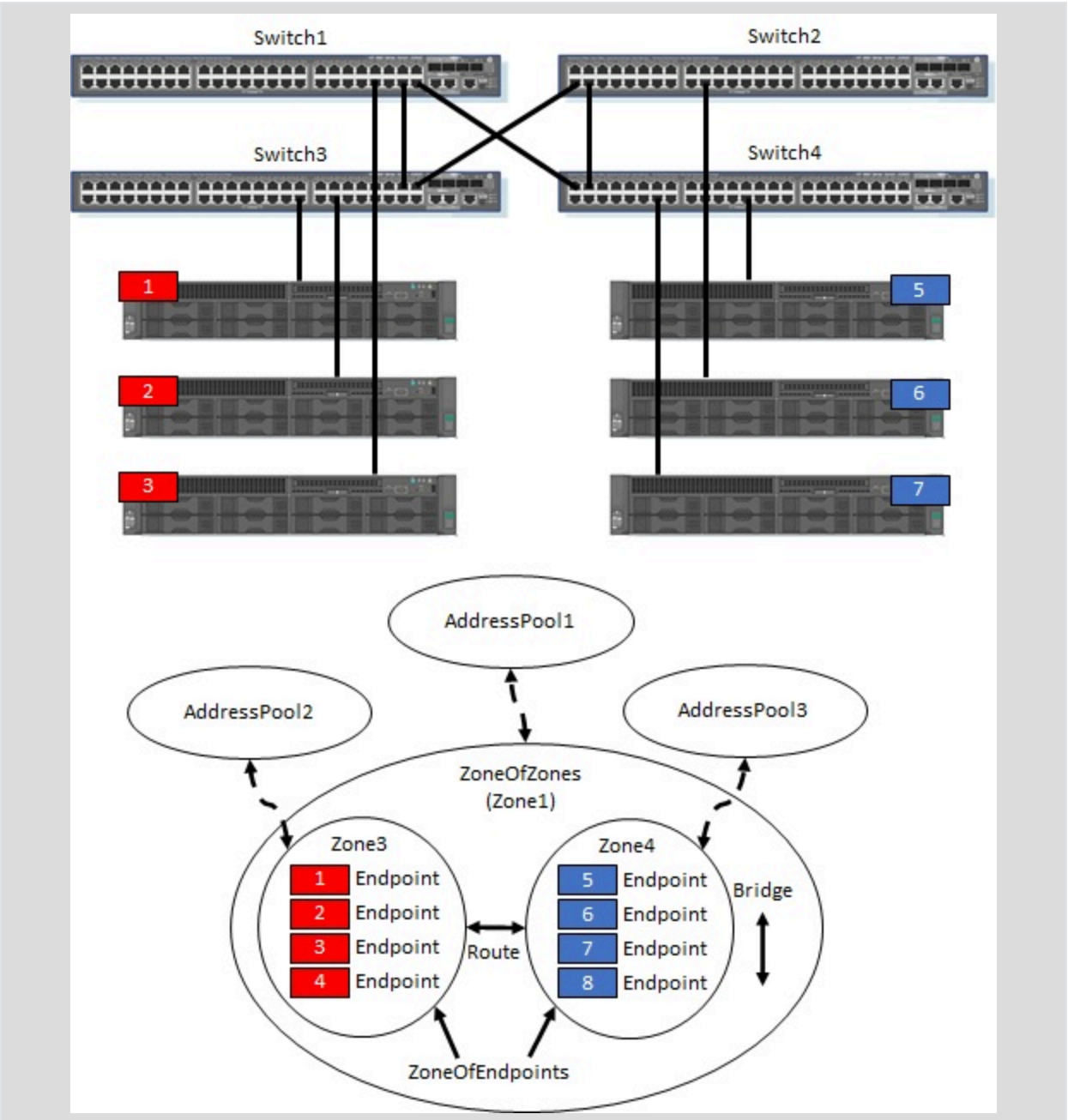

Figure 2 — Fabric configuration

5.1 Zones

The `Zone` resource constrains communications by subdividing a fabric into a series of regions or subnets, either for

the entire data center fabric or for multi-tenant access to it. The `ZoneType` property describes the usage of the zone in the fabric and can contain the following values.

Value	Description
`Default`	The zone is associated with the entire fabric. Newly created endpoints appear in this zone until additional configurations are made by a client to assign the endpoint to another zone.
`ZoneOfEndpoints`	The zone contains a set of endpoints where routing is enabled for traffic to flow between the endpoints.
`ZoneOfZones`	The zone contains other zones for scalability. In Ethernet fabrics, this signifies one or more virtual routing domain or virtual routing (VRF) instances. Zones referenced by this zone will contain `ZoneOfEndpoints` for their `ZoneType` property. Endpoints that are within one zone of type `ZoneOfZones` can overlap with endpoints that are in a different zone of type `ZoneOfZones`. This construct emulates multi-tenancy across fabrics.
`ZoneOfResourceBlocks`	The zone represents a set of resource blocks that can be composed together. This value is specific to composability and does not apply to fabrics.

Each `Fabric` resource should contain a `Zone` resource whose `ZoneType` property contains the value `Default`. This is a well-known location for clients to understand default routing policies for endpoints that have not been assigned to a zone. The `DefaultRoutingEnabled` property controls whether traffic is allowed between the endpoints in this zone.

Zones of type `ZoneOfZones` are prohibited from containing other zones of type `ZoneOfZones`. This is to prevent complexities with nesting zones within zones within zones and also prevents creating circular zoning situations.

The `Links` property can contain the following properties to show additional configuration of the zone and components that are in the zone.

- `Endpoints`: The endpoints that belong in this zone. This only applies when `ZoneType` contains `Default` or `ZoneOfEndpoints`.
- `ContainsZones`: The zones that belong in this zone. This only applies when `ZoneType` contains `ZoneOfZones`.
- `ContainedByZones`: The zone that contains this zone. This only applies when `ZoneType` contains `ZoneOfEndpoints`.
- `AddressPools`: Additional networking configuration for this zone.
- `InvolvedSwitches`: The switches that are used by this zone.
- `ResourceBlocks`: The resource blocks that belong in this zone. This only applies when `ZoneType` contains `ZoneOfResourceBlocks` and does not apply to fabrics.

Figure 3 shows a sample set of zones and endpoints. There are 16 endpoints grouped into four zones of type `ZoneOfEndpoints`. There are also two zones of type `ZoneOfZones` that provide further groupings. The endpoints that belong in the same zone of type `ZoneOfEndpoints`, such as `Zone3`, are hosts on the same network, therefore traffic is bridged across the fabric between these endpoints. Endpoints that belong to different zones of type `ZoneOfEndpoints`, but are contained in the same zone of type `ZoneOfZones`, such as zones `Zone3` and `Zone4`, are on different networks where the fabric is configured to route traffic between the endpoints. Endpoints that do not

belong in any common zones, such as the endpoints in zones `Zone3` and `Zone6`, are not configured to have any routing between the endpoints. If routing is required between `Zone3` and `Zone6`, a new zone of type `ZoneOfZones` would need to be created with `Zone3` and `Zone6` as part of the new zone.

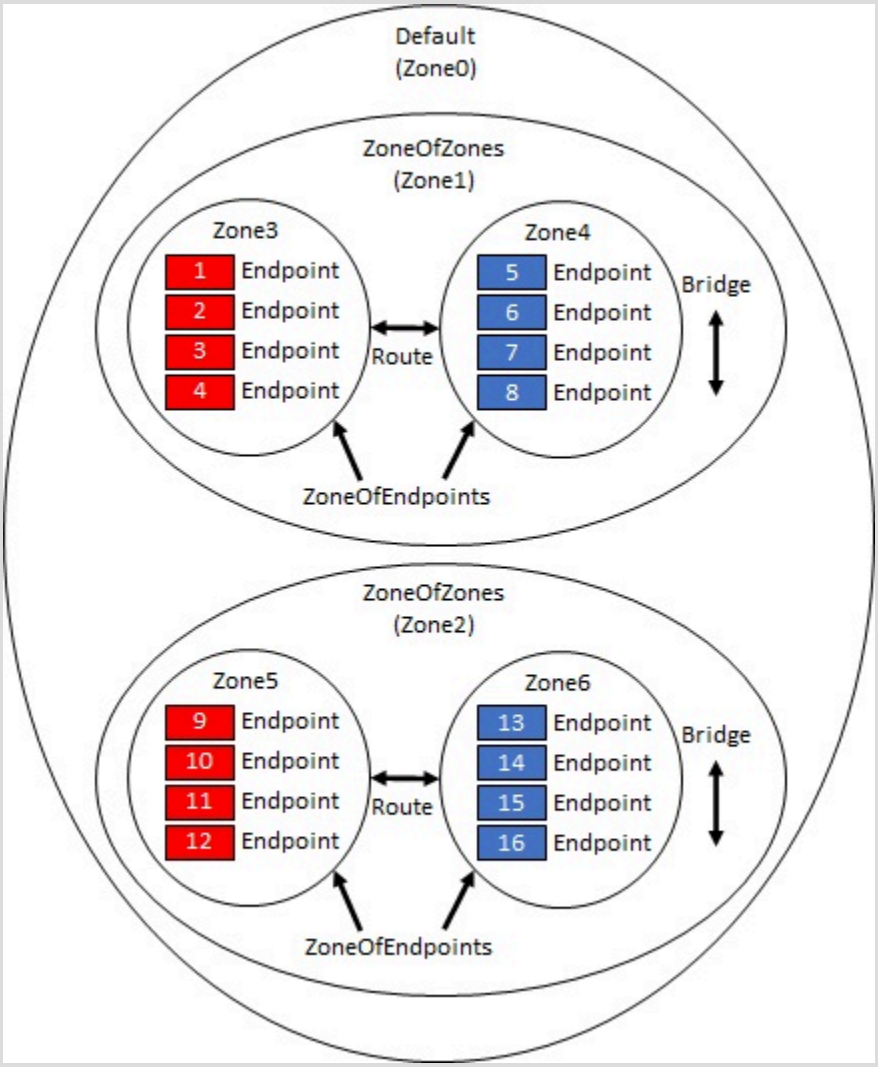

Figure 3 — Redfish zones

By automating these interrelationships, it should be possible to configure any fabric to accommodate those addressing and connectivity requirements. This allows for a multi-vendor and fully standards-based method to uniformly configure distributed fabrics in tandem with compute and storage infrastructure inside the data center.

5.2 Address pools

The `AddressPool` resource constrains control plane specific pools of addressing for setting up fabric-wide communications as well as host network address pools. Address pools can be applied to particular zones in a fabric in order to configure zone-specific networking.

In an Ethernet fabric, an address pool can contain subnet, default gateway, VLAN, BGP underlay, EVPN control plane, and other networking configurations.

In a Gen-Z fabric, an address pool can specify Global Component Identifier information.

In Figure 2, there are three example address pools that belong to an Ethernet fabric. `AddressPool1` contains cross Ethernet fabric addressing settings such as EBGP underlay addressing, EVPN address pools, and BGP timers. `AddressPool2` and `AddressPool3` contain subnet settings such as IP network ranges, gateways, and VLANs for that network.

5.3 Connections

The `Connection` resource contains access permissions for resources accessible via a target endpoint once two endpoints establish a communication channel. This differs from `Zone` resources in that zones describe the routing for the communication channel itself.

`Connection` resources contain a `ConnectionType` property to describe the type of resources target endpoints can expose to the connecting initiator endpoints and can contain the following values.

Value	Description
`Storage`	The target endpoints are able to provide storage-related resources, such as volumes.
`Memory`	The target endpoints are able to provide memory-related resources, such as memory chunks.

Based on the value of `ConnectionType`, one of the following properties will be present to provide the information about the specific resources made available to initiator endpoints.

- `VolumeInfo` : An array containing references to one or more `Volume` resources along with access capabilities, such as whether the volumes are read-only or read-write for the connecting initiators.
- `MemoryInfo` : An array containing references to one or more `MemoryChunk` resources along with access capabilities, such as whether the volumes are read-only or read-write for the connecting initiators.

The `ConnectionKeys` property may also be present for fabrics that require specifying access keys, such as with Gen-Z fabrics.

The `Links` property contains references to the endpoints affected by the connection. Initiators can be referenced with the `InitiatorEndpoints` or `InitiatorEndpointGroups` properties, and targets can be referenced by the `TargetEndpoints` or `TargetEndpointGroups` properties. When a referenced initiator establishes a connection over the fabric with one of the referenced targets, it's able to access the resources specified by other properties, such as `VolumeInfo`.

Figure 4 shows a sample NVMe-oF fabric. The endpoints `Host1` and `Host2` act as initiator endpoints on the fabric, and endpoint `Target1` is the NVMe-oF target. The connection `Conn1` specifies that both `Host1` and `Host2` are given access to the volume `NS1` when they establish a connection with `Target1`.

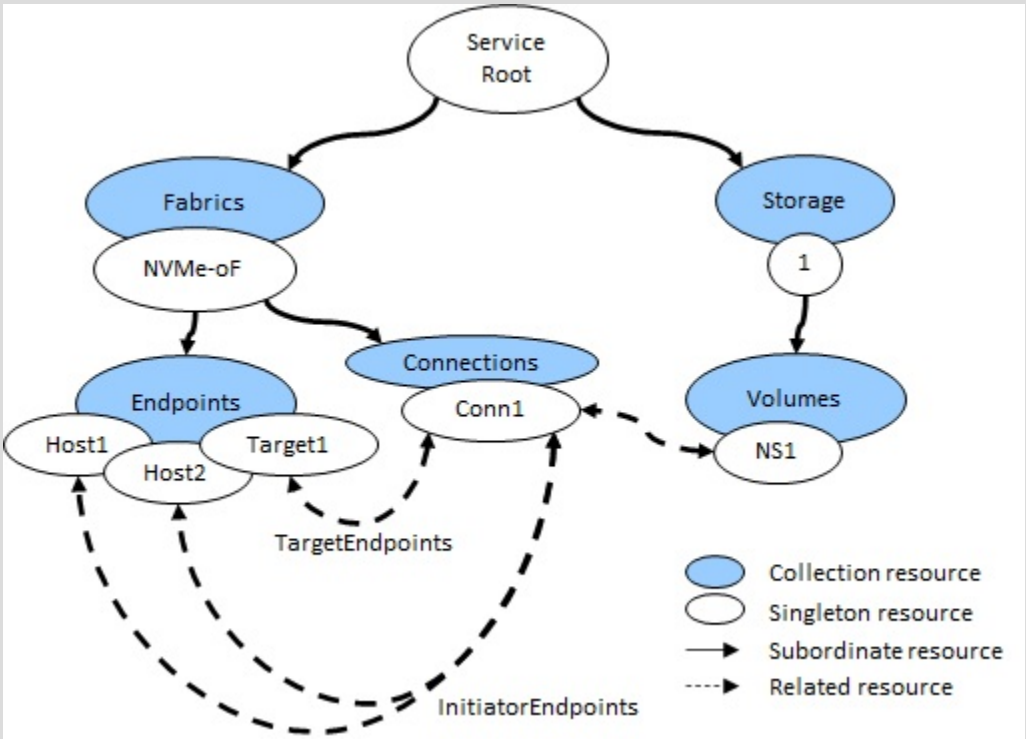

Figure 4 — Redfish connections

6 Fabric management flows

The Redfish fabric model provides a unified, standards-based API for managing a wide range of fabric technologies in data centers, including PCIe, CXL, and NVMe over Fabrics. It enables administrators to perform the following key fabric management operations:

- Systems capabilities discovery: Identify and enumerate the features and capabilities of connected systems across different fabrics.
- Physical connectivity configuration: Define and adjust the physical connections and topologies between fabric connected systems.
- Connection establishment: Initiate and manage the operational connections that enable communication and data flow between fabric endpoints representing resources in initiator and target systems.
- Connection termination: Safely disconnect and decommission operational connections between fabric endpoints, ensuring resources are released and network integrity is maintained.

The following sections outline the management flows for each key operation mentioned above. Step-by-step guidance is provided for systems capabilities discovery, physical connectivity configuration, and connection establishment, helping administrators efficiently manage fabric resources with the Redfish fabric model.

6.1 Generic fabric management flows

This section presents a set of generic management flows that are universally applicable to any fabric technology. By abstracting away technology-specific details, these flows serve as a foundational guide for data center administrators seeking to manage fabric resources efficiently and consistently. The outlined processes include initiator systems discovery, target systems discovery, switch system discovery, connection establishment, and connection termination. Each flow is accompanied by an illustrative diagram to provide a clear, conceptual understanding of the management steps involved.

6.1.1 Initiator systems discovery

Initiator systems discovery involves identifying all endpoints within the fabric that are capable of initiating connections. Administrators use this flow to enumerate initiator devices and assess their readiness and available capabilities for establishing fabric connections. This process ensures that all potential initiator resources are visible and manageable within the fabric management domain.

Figure 5 shows the detailed initiator system fabric capabilities discovery flow.

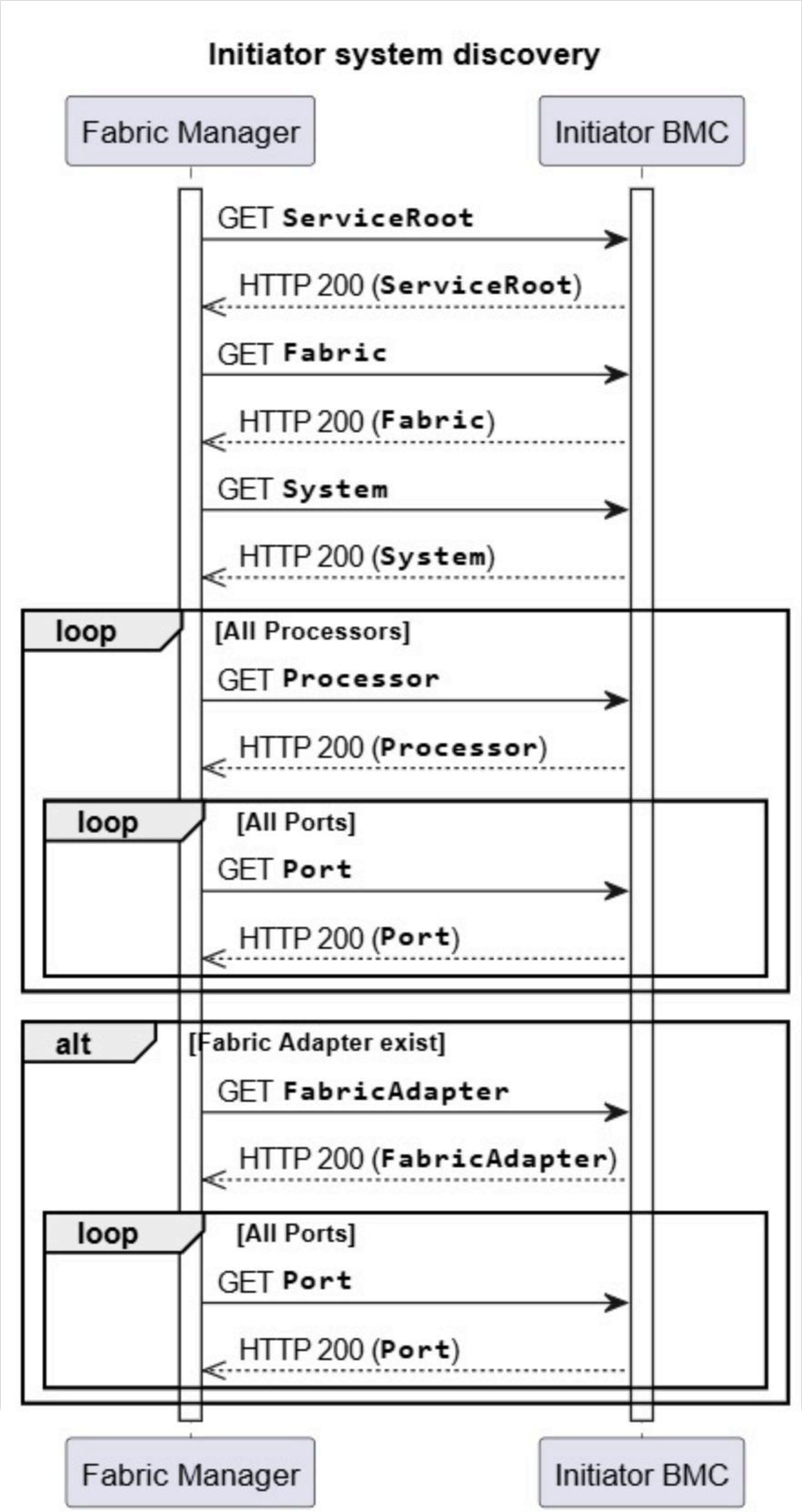
Initiator system discovery
Fabric Manager
Initiator BMC
GET ServiceRoot
HTTP 200 (ServiceRoot)
GET Fabric
HTTP 200 (Fabric)
GET System
HTTP 200 (System)
loop
[All Processors]
GET Processor
HTTP 200 (Processor)
loop
[All Ports]
GET Port
HTTP 200 (Port)
alt
[Fabric Adapter exist]
GET FabricAdapter
HTTP 200 (FabricAdapter)
loop
[All Ports]
GET Port
HTTP 200 (Port)
Fabric Manager
Initiator BMC

Figure 5 — Initiator system fabric capabilities discovery flow

6.1.2 Target systems discovery

Target systems discovery focuses on identifying endpoints that can accept connections from initiators. This management flow allows administrators to enumerate target devices, verify their operational status, and determine their connectivity options. A clear inventory of target systems enables effective resource allocation and connection planning across the fabric.

Figure 6 shows the detailed target system fabric capabilities discovery flow.

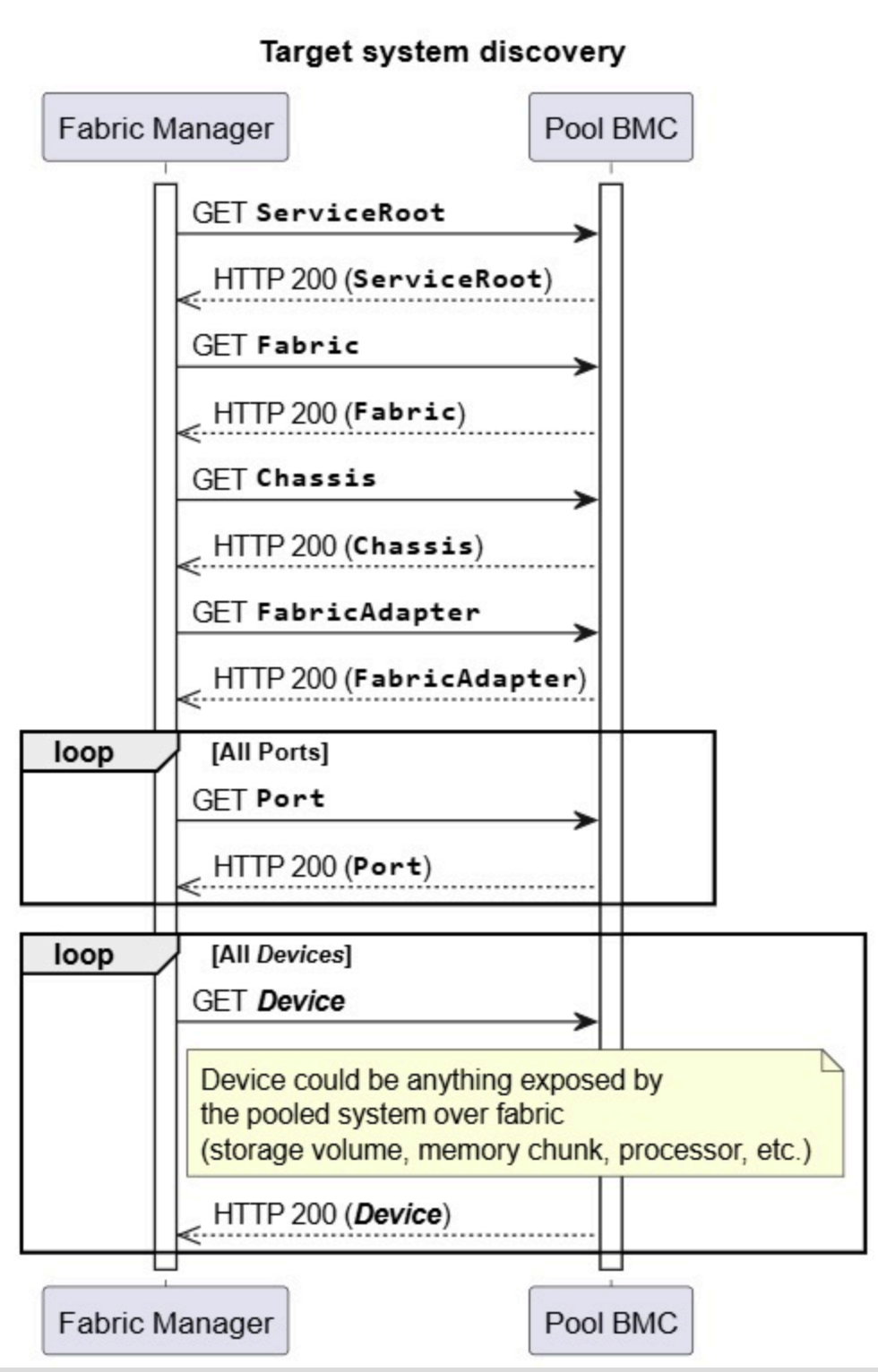
Target system discovery
Fabric Manager
Pool BMC
GET ServiceRoot
HTTP 200 (ServiceRoot)
GET Fabric
HTTP 200 (Fabric)
GET Chassis
HTTP 200 (Chassis)
GET FabricAdapter
HTTP 200 (FabricAdapter)
loop
[All Ports]
GET Port
HTTP 200 (Port)
loop
[All Devices]
GET Device
Device could be anything exposed by
the pooled system over fabric
(storage volume, memory chunk, processor, etc.)
HTTP 200 (Device)
Fabric Manager
Pool BMC

Figure 6 — Target system fabric capabilities discovery flow

6.1.3 Switch system discovery

Switch system discovery is the process of identifying intermediary fabric devices that facilitate data routing between initiator and target systems. Through this flow, administrators can map out available switches, understand their topological roles, and optimize data paths within the fabric. Comprehensive switch discovery supports robust and flexible fabric design.

Figure 7 shows the detailed switch system fabric capabilities discovery flow.

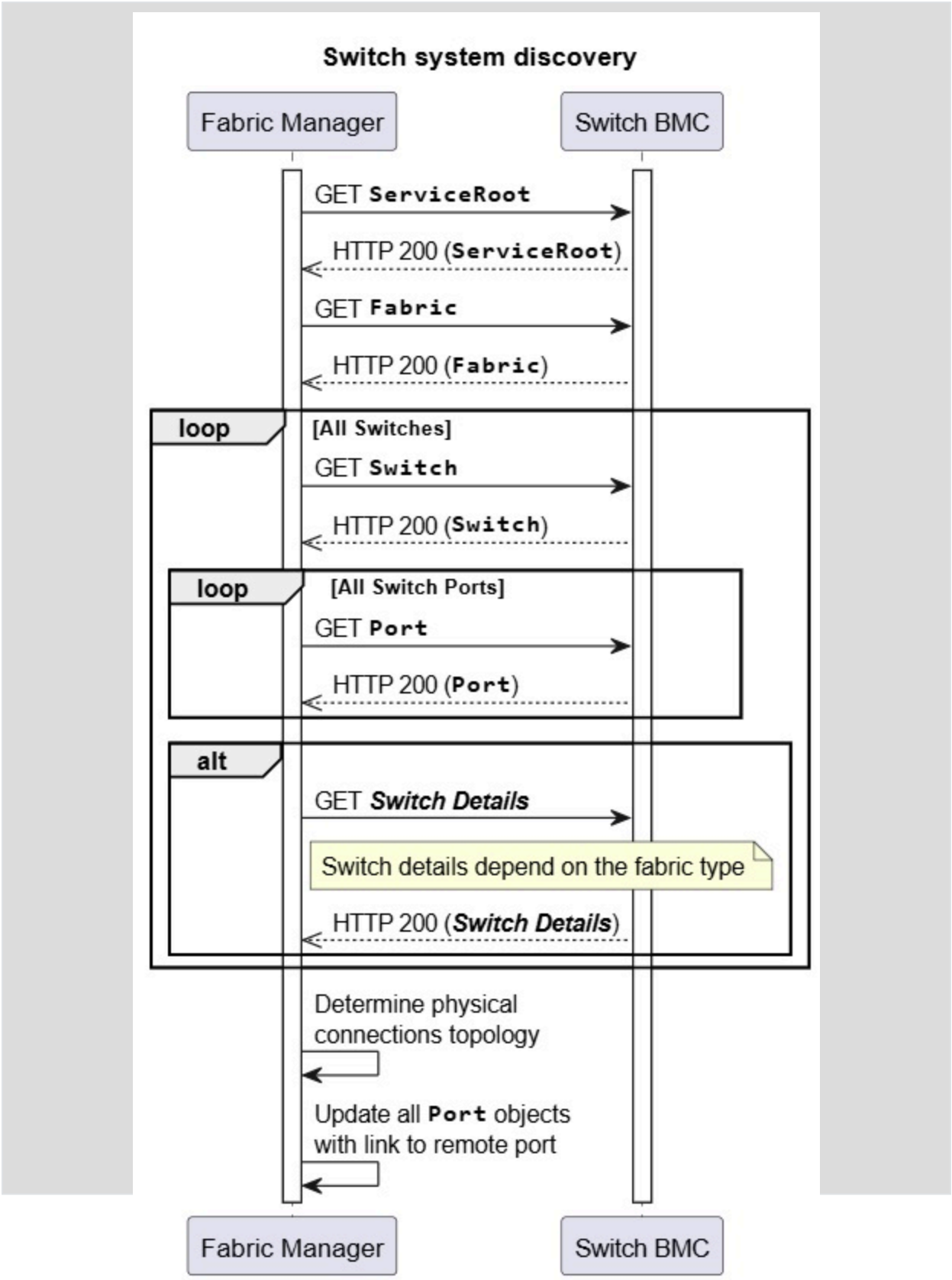
Switch system discovery
Fabric Manager
Switch BMC
GET ServiceRoot
HTTP 200 (ServiceRoot)
GET Fabric
HTTP 200 (Fabric)
loop
[All Switches]
GET Switch
HTTP 200 (Switch)
loop
[All Switch Ports]
GET Port
HTTP 200 (Port)
alt
GET Switch Details
Switch details depend on the fabric type
HTTP 200 (Switch Details)
Determine physical connections topology
Update all Port objects with link to remote port
Fabric Manager
Switch BMC

Figure 7 — Switch system fabric capabilities discovery flow

6.1.4 Connection establishment

Connection establishment consists of two distinct phases: first, the creation of physical connectivity, and second, the establishment of operational links between initiator and target resources. In the physical connectivity creation phase, the necessary pathways—often involving one or more switches—are provisioned to ensure that initiator and target endpoints can communicate. Following this, the process moves to establishing the connection itself, where connection parameters are negotiated, required resources are allocated, and data paths are activated. Efficiently performing both phases is essential for enabling seamless data exchange within the fabric. The following figures show consecutive phases of the detailed fabric connection establishing flow.

Figure 8 shows the flow of the target system setup phase.

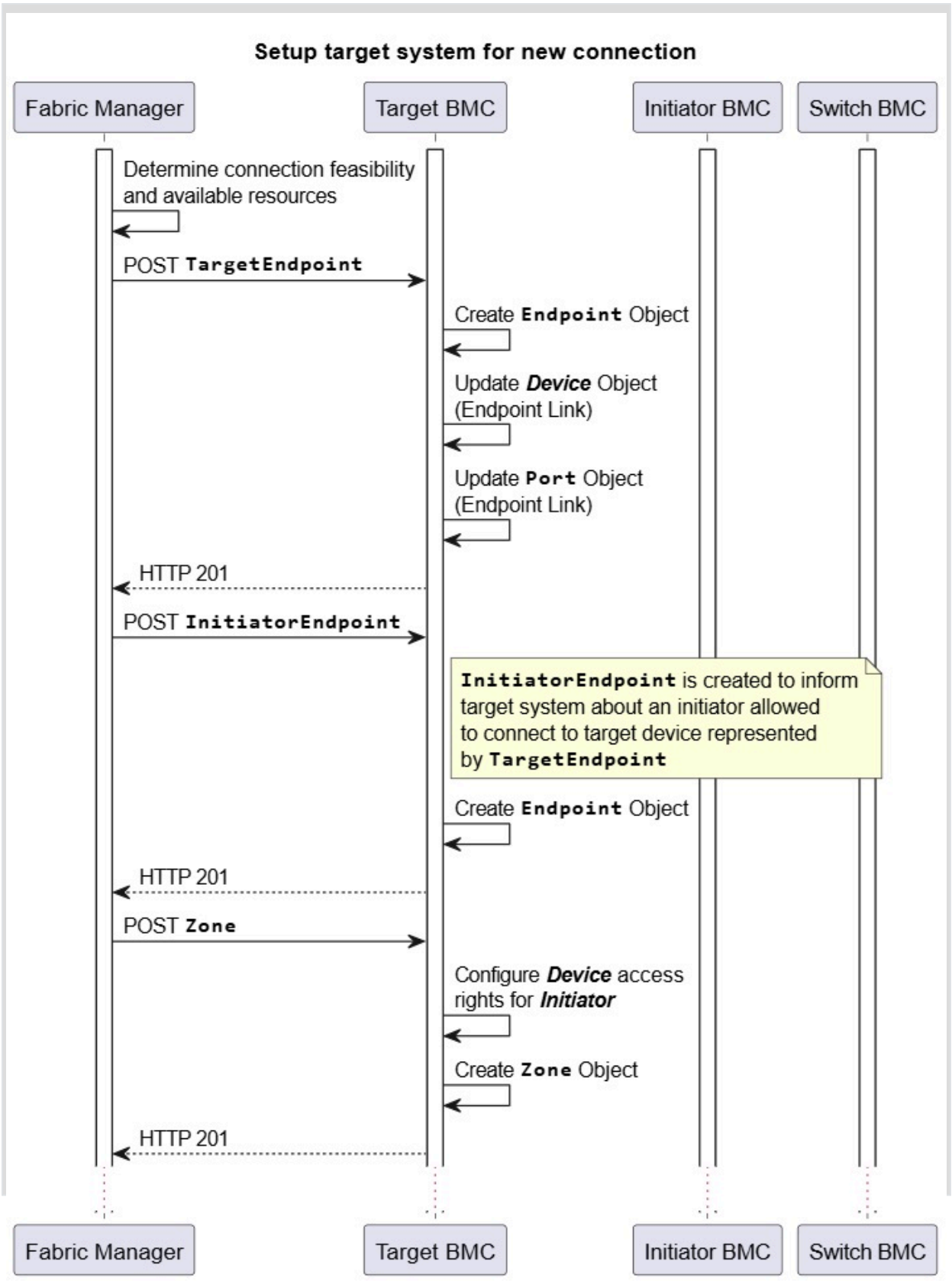
Setup target system for new connection
Fabric Manager
Target BMC
Initiator BMC
Switch BMC
Determine connection feasibility and available resources
POST TargetEndpoint
Create Endpoint Object
Update Device Object (Endpoint Link)
Update Port Object (Endpoint Link)
HTTP 201
POST InitiatorEndpoint
InitiatorEndpoint is created to inform target system about an initiator allowed to connect to target device represented by TargetEndpoint
Create Endpoint Object
HTTP 201
POST Zone
Configure Device access rights for Initiator
Create Zone Object
HTTP 201
Fabric Manager
Target BMC
Initiator BMC
Switch BMC

Figure 8 — Setup target system phase flow

Figure 9 shows the flow of the initiator system setup phase.

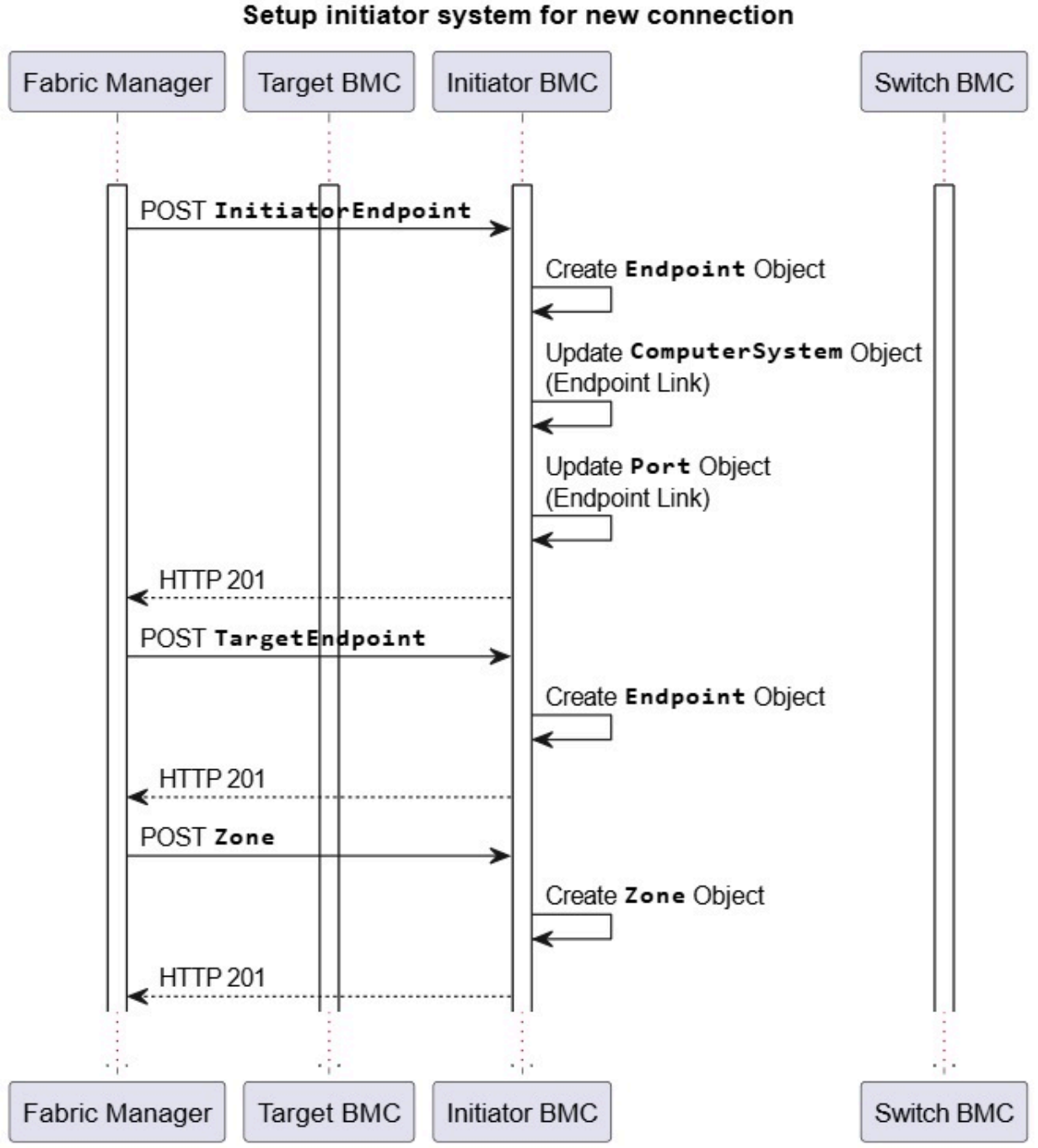

Figure 9 — Setup initiator system phase flow

Figure 10 shows the flow of the switch system setup phase.

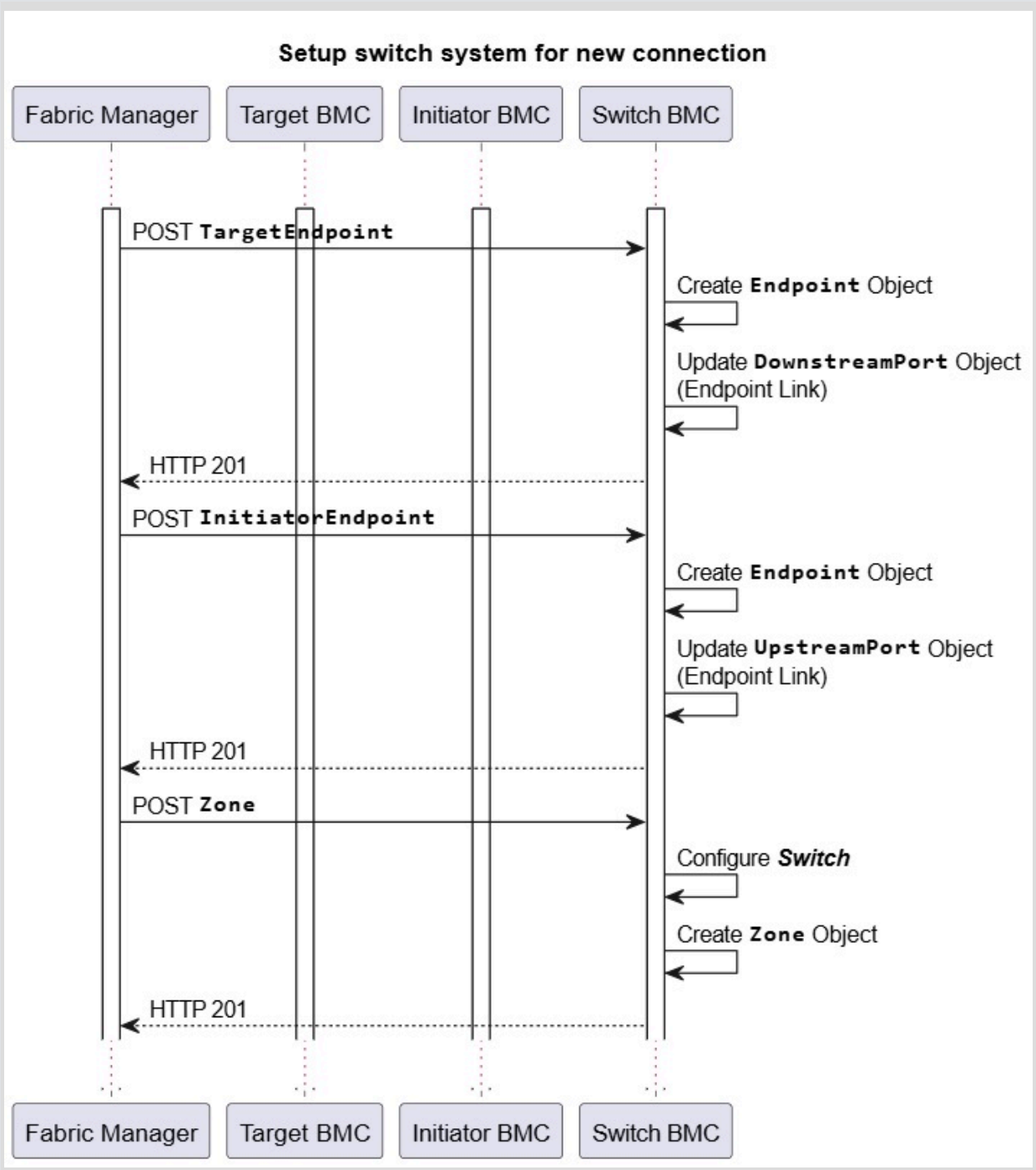

Figure 10 — Setup switch system phase flow

Figure 11 shows the flow of the new connection establishing phase.

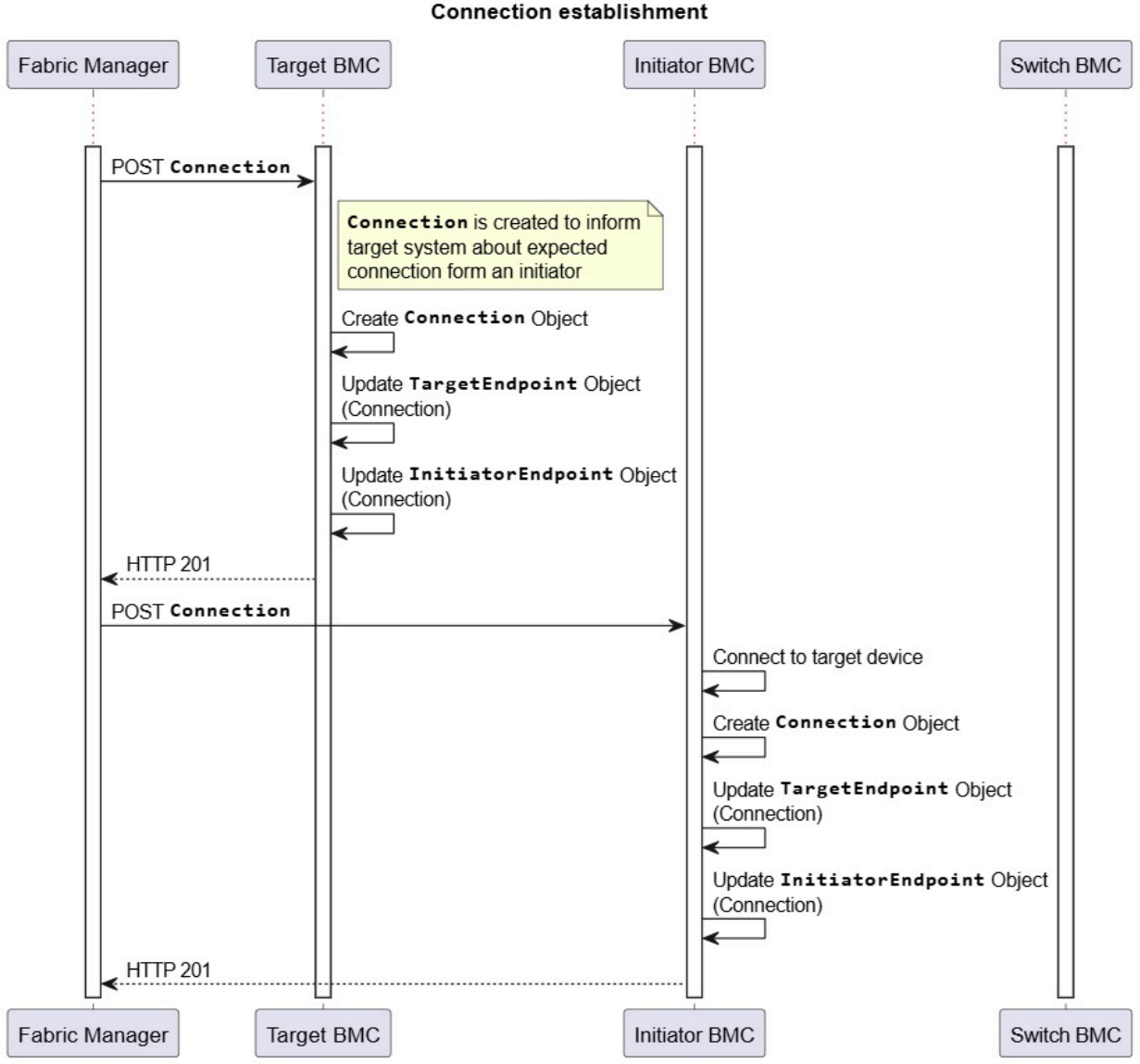

Figure 11 — New connection establishing phase flow

6.1.5 Connection termination

Connection termination involves two main steps to ensure an orderly disconnection of fabric endpoints. First, the administrator deactivates an active connection between resources in initiator and target systems, which includes disabling data paths and releasing resources directly associated with those operational links. Second, administrators proceed to terminate the overall fabric connectivity between systems—this step is performed only when no active

connections remain. At this stage, management systems are updated to reflect the new topology, and any additional resources linked to the underlying fabric pathways are released. Following these steps is crucial for maintaining fabric integrity and ensuring optimal resource utilization. The following figures show consecutive phases of the detailed fabric connection termination flow.

Figure 12 shows the flow of the connection termination phase phase.

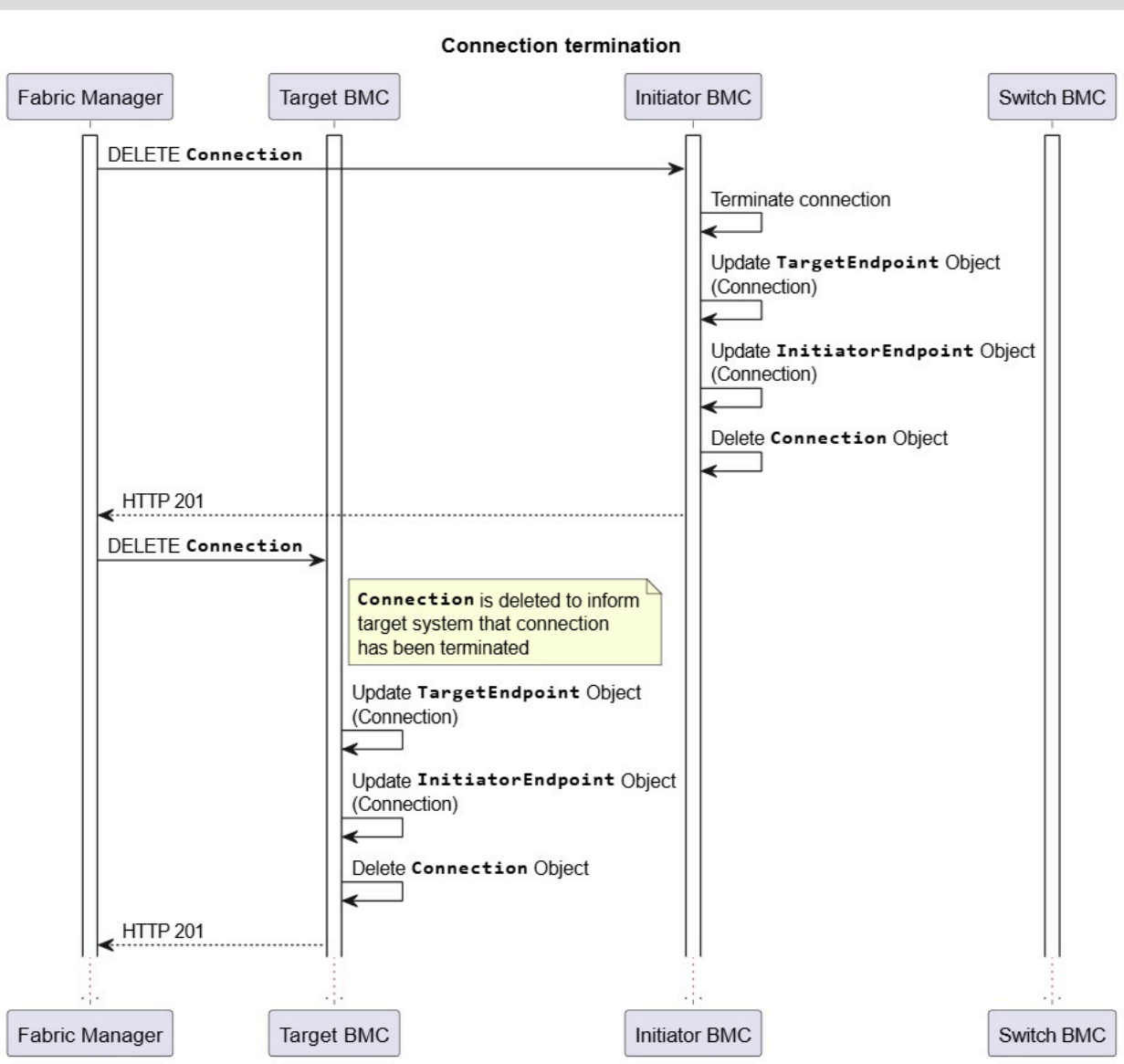

Figure 12 — Existing connection termination phase flow

Figure 13 shows the flow of optional phase of the switch system configuration clearing. This phase is performed when no active connection remains between the initiator system and the target system.

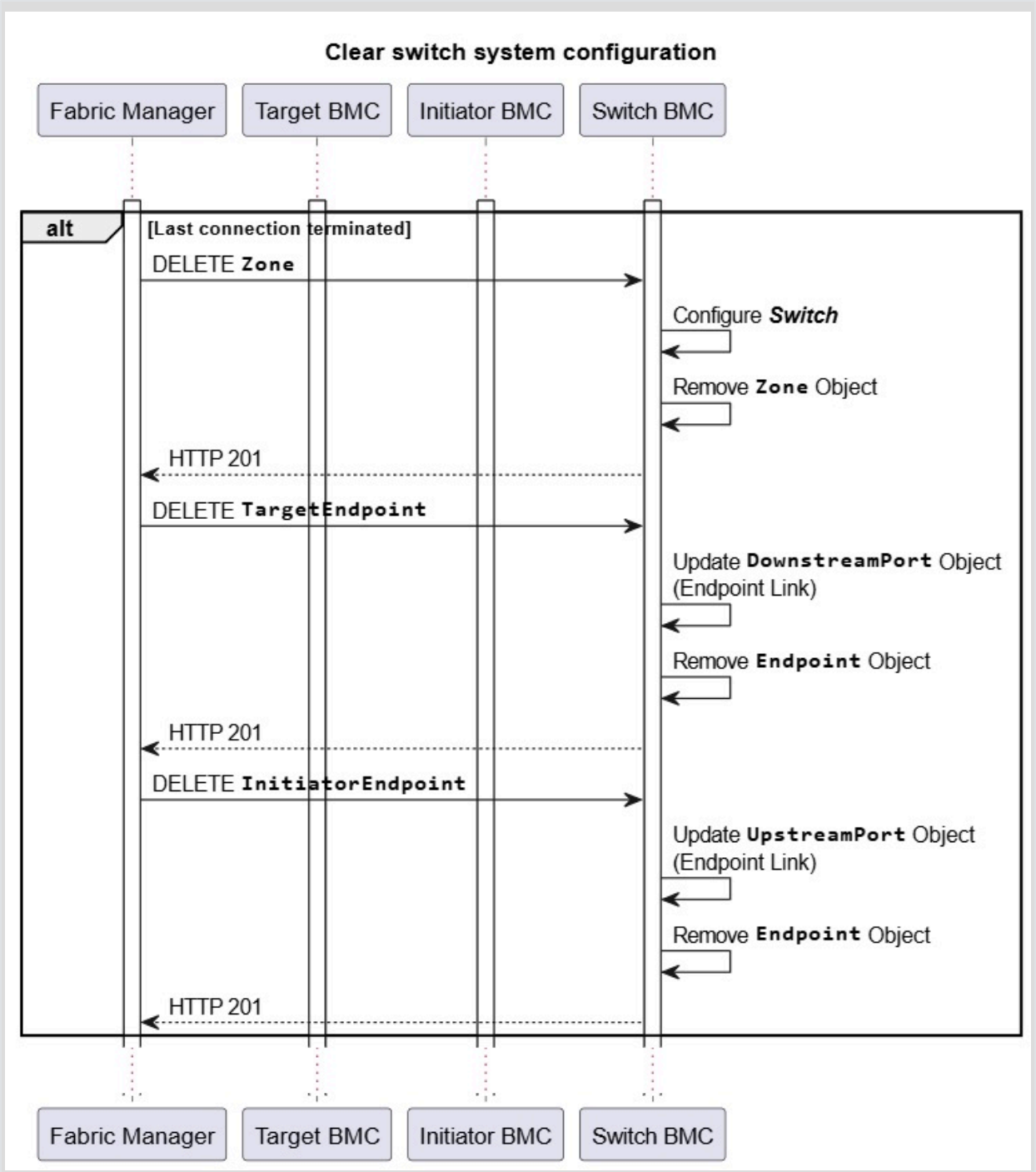

Figure 13 — Clear switch system configuration phase flow

Figure 14 shows the flow of optional phase of the target system configuration clearing. This phase is performed when no active connection remains between the initiator system and the target system.

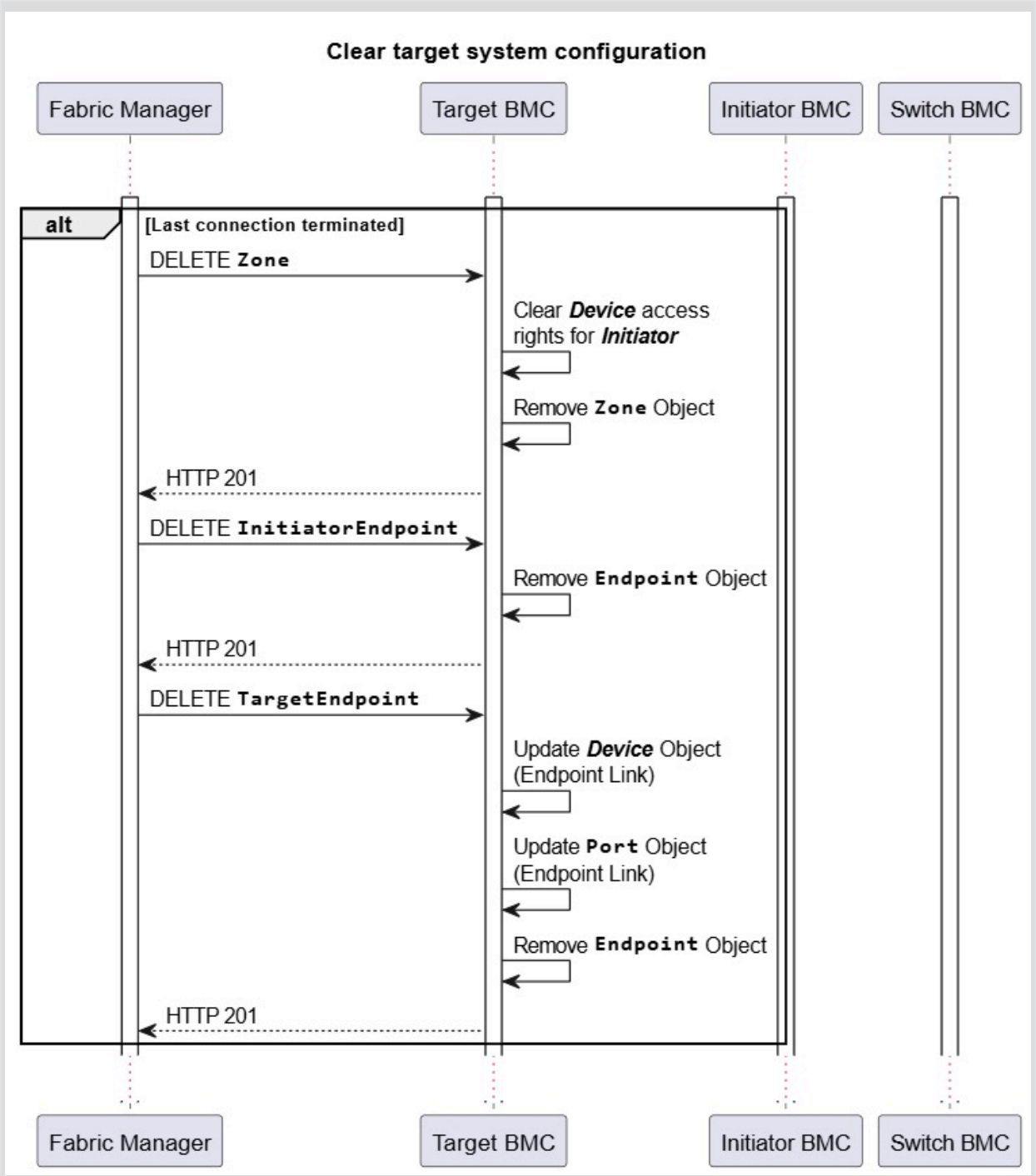

Figure 14 — Clear target system configuration phase flow

Figure 15 shows the flow of optional phase of the initiator system configuration clearing. This phase is performed when no active connection remains between the initiator system and the target system.

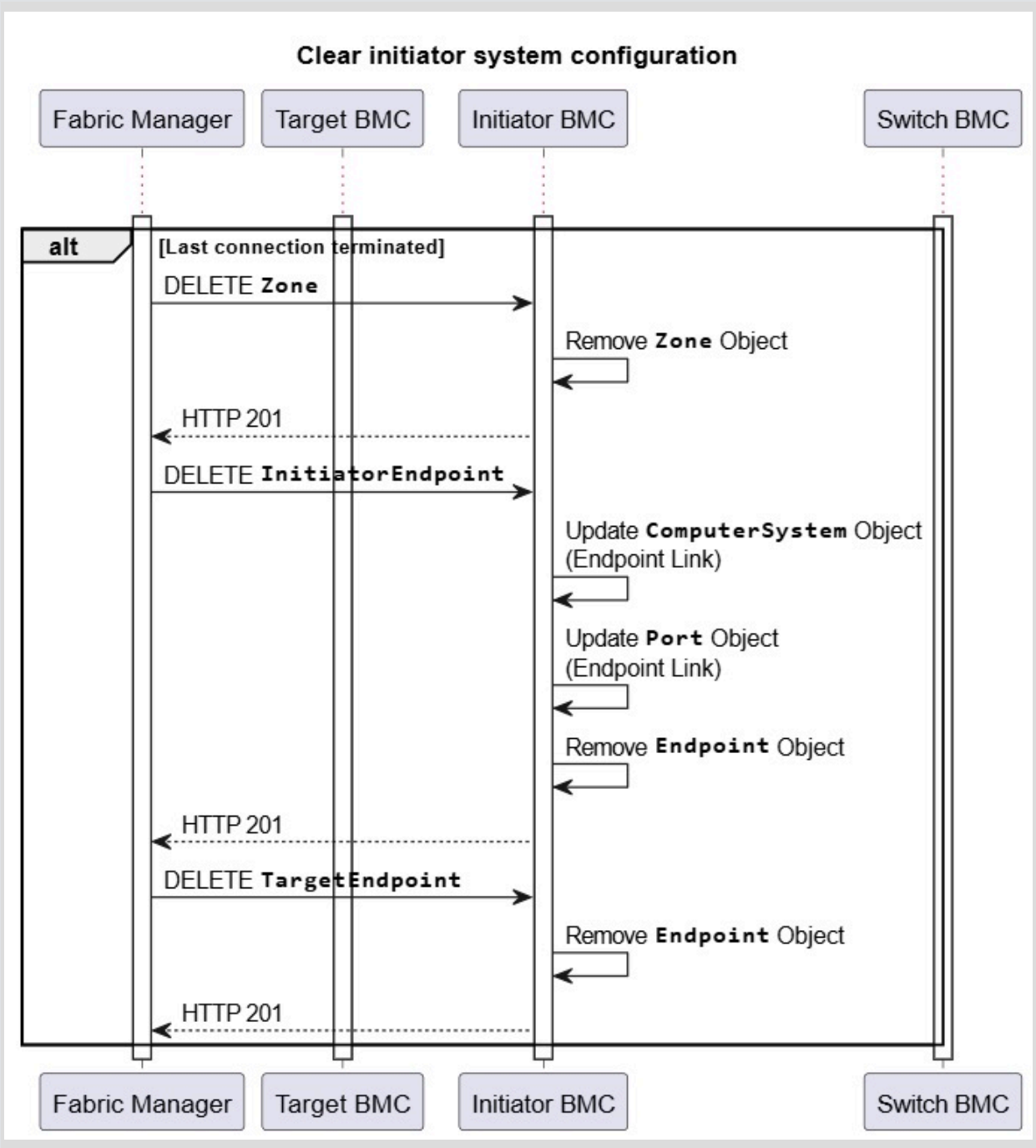

Figure 15 — Clear initiator system configuration phase flow

7 Fabric management flows for CXL Type 3 devices

This section provides a comprehensive overview of the essential management workflows for pooled memory systems, with a particular emphasis on **CXL Type 3** devices operating within a CXL fabric environment. These management flows are critical for ensuring efficient resource utilization, robust connectivity, and secure operations across the memory fabric. The key operations described include:

- **Pooled memory system discovery**: The process of identifying and cataloging all available pooled memory systems, assessing their operational status, and confirming their readiness to offer memory resources for remote access.
- **Host system discovery**: The systematic identification and verification of host systems that are compatible with CXL protocols and capable of mapping and accessing remote memory provided by pooled targets.
- **CXL switch system discovery**: The mapping and assessment of all CXL switches that serve as intermediaries, ensuring that the pathways for memory access between hosts and pooled memory systems are functioning optimally.
- **Connection establishment**: The coordinated steps involved in allocating memory chunks on pooled systems, establishing physical and operational connections via CXL switches, and mapping these resources into the host's address space for seamless remote access.
- **Connection Termination**: The managed process of disconnecting hosts from remote memory, deallocating memory resources as appropriate, and updating the CXL fabric topology to maintain system integrity and resource availability.

Together, these management flows form the foundation for orchestrating resource discovery, connectivity, and lifecycle management in CXL-based pooled memory environments, supporting both performance and scalability objectives.

7.1 Pooled memory system discovery (target system)

In the CXL fabric, the discovery process starts with pooled memory systems, as these targets initiate connections to eligible hosts. Administrators identify and catalog all available pooled memory systems and their associated memory chunks, check their operational states, and ensure readiness for offering remote memory resources. This makes sure all resources capable of establishing CXL connections are visible and manageable.

Figure 16 shows the detailed pooled memory system fabric capabilities discovery flow.

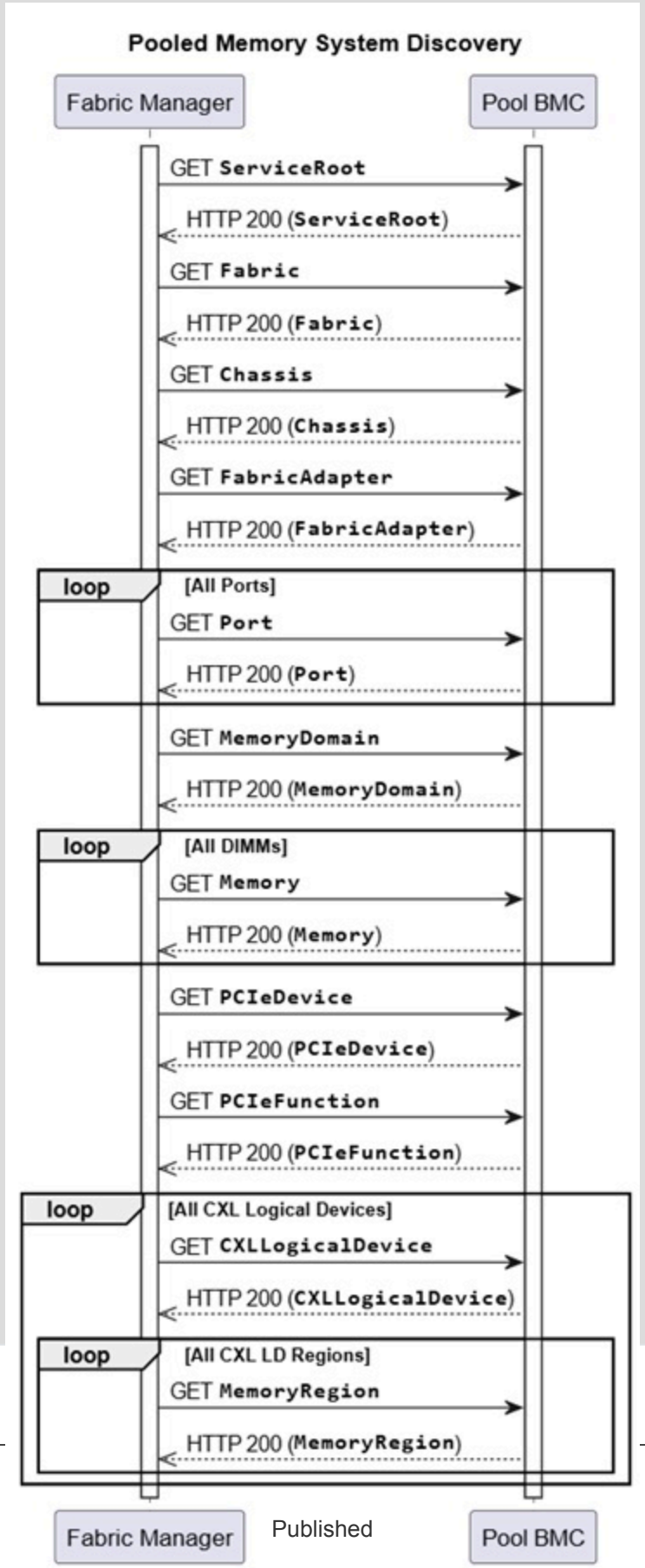
Pooled Memory System Discovery
Fabric Manager
Pool BMC
GET ServiceRoot
HTTP 200 (ServiceRoot)
GET Fabric
HTTP 200 (Fabric)
GET Chassis
HTTP 200 (Chassis)
GET FabricAdapter
HTTP 200 (FabricAdapter)
loop
[All Ports]
GET Port
HTTP 200 (Port)
GET MemoryDomain
HTTP 200 (MemoryDomain)
loop
[All DIMMs]
GET Memory
HTTP 200 (Memory)
GET PCIeDevice
HTTP 200 (PCIeDevice)
GET PCIeFunction
HTTP 200 (PCIeFunction)
loop
[All CXL Logical Devices]
GET CXLLogicalDevice
HTTP 200 (CXLLogicalDevice)
loop
[All CXL LD Regions]
GET MemoryRegion
HTTP 200 (MemoryRegion)
Fabric Manager
Pool BMC

Figure 16 — Pooled memory system CXL fabric capabilities discovery flow

7.2 Host system discovery (initiator system)

Next, discovery focuses on host computer systems whose CPUs can leverage remote memory from pooled targets. Administrators enumerate compatible hosts, verify support for CXL protocols, and ensure their capability to map and access remote CXL memory. This step is crucial for confirming only eligible hosts participate in target-initiated connections.

Figure 17 shows the detailed host system fabric capabilities discovery flow.

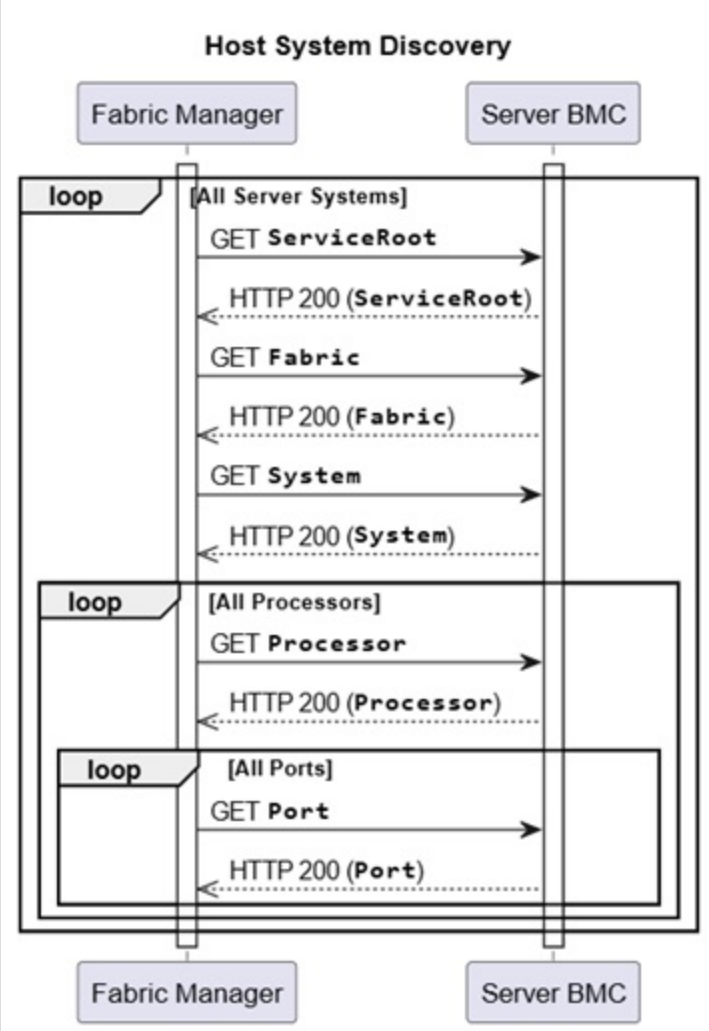

Figure 17 — Host system CXL fabric capabilities discovery flow

7.3 CXL switch system discovery

Discovery of CXL switches involves identifying all intermediary devices that route memory access between pooled memory targets and host CPUs. Administrators map these switches, assess their connectivity and function, and confirm that memory access paths are optimized for both performance and reliability.

Figure 18 shows the detailed fabric switch system configuration discovery flow.

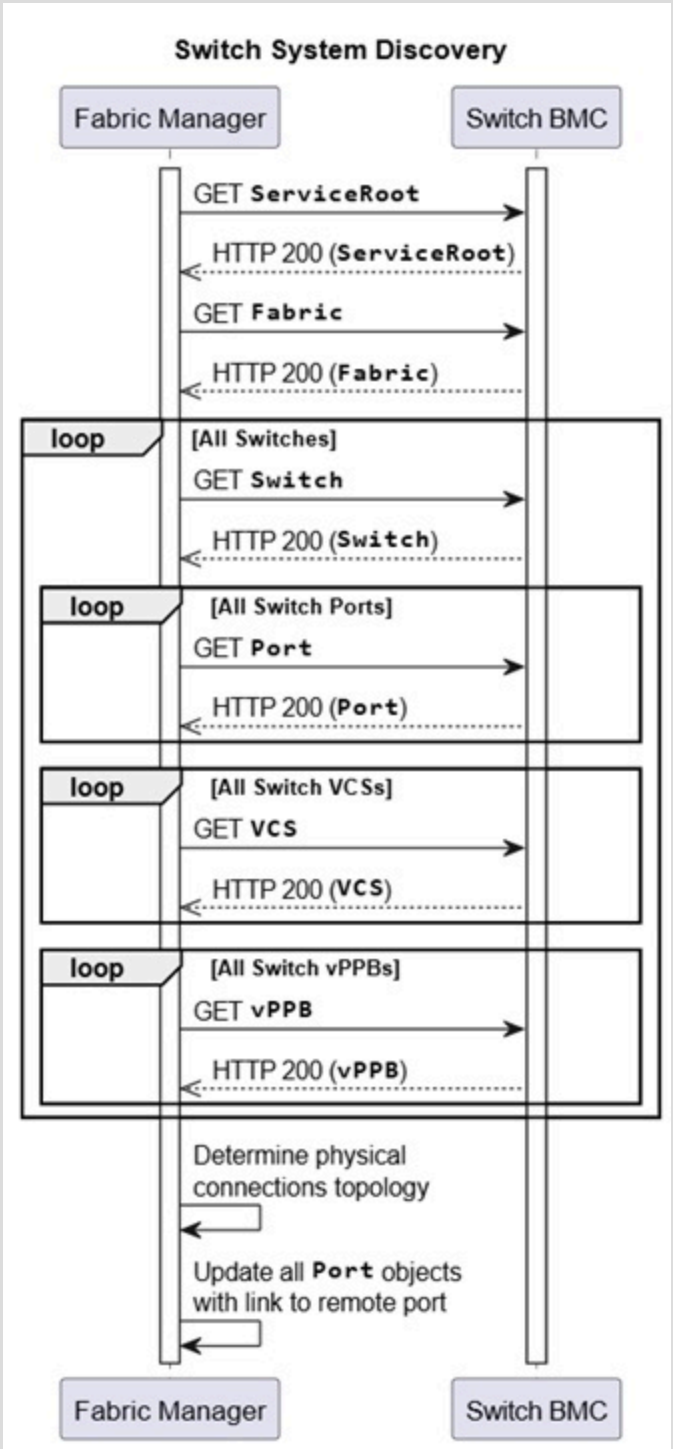

Figure 18 — Switch system CXL fabric capabilities discovery flow

7.4 Connection establishment

In CXL fabrics, establishing a connection involves three main steps:

1. **Memory allocation:** Allocate memory within the pooled memory system, selecting and preparing specific chunks for remote access.
2. **Physical connectivity:** Set up the physical connection between the host and the pooled memory system via a CXL switch, ensuring active data paths.
3. **Operational link:** Establish the operational link by negotiating parameters and mapping the memory region into the host's address space for remote access.

> **Note:** In CXL fabric setup, the connection is initiated by the target system (pooled memory), not the host. This process ensures that only available and eligible hosts get access, supporting efficient resource management and security. The host may accept, partially accept or reject the memory offered by the device during connection establishment.

The following figures show consecutive phases of the detailed CXL fabric connection establishing flow.

Figure 19 shows the flow of the memory system setup phase.

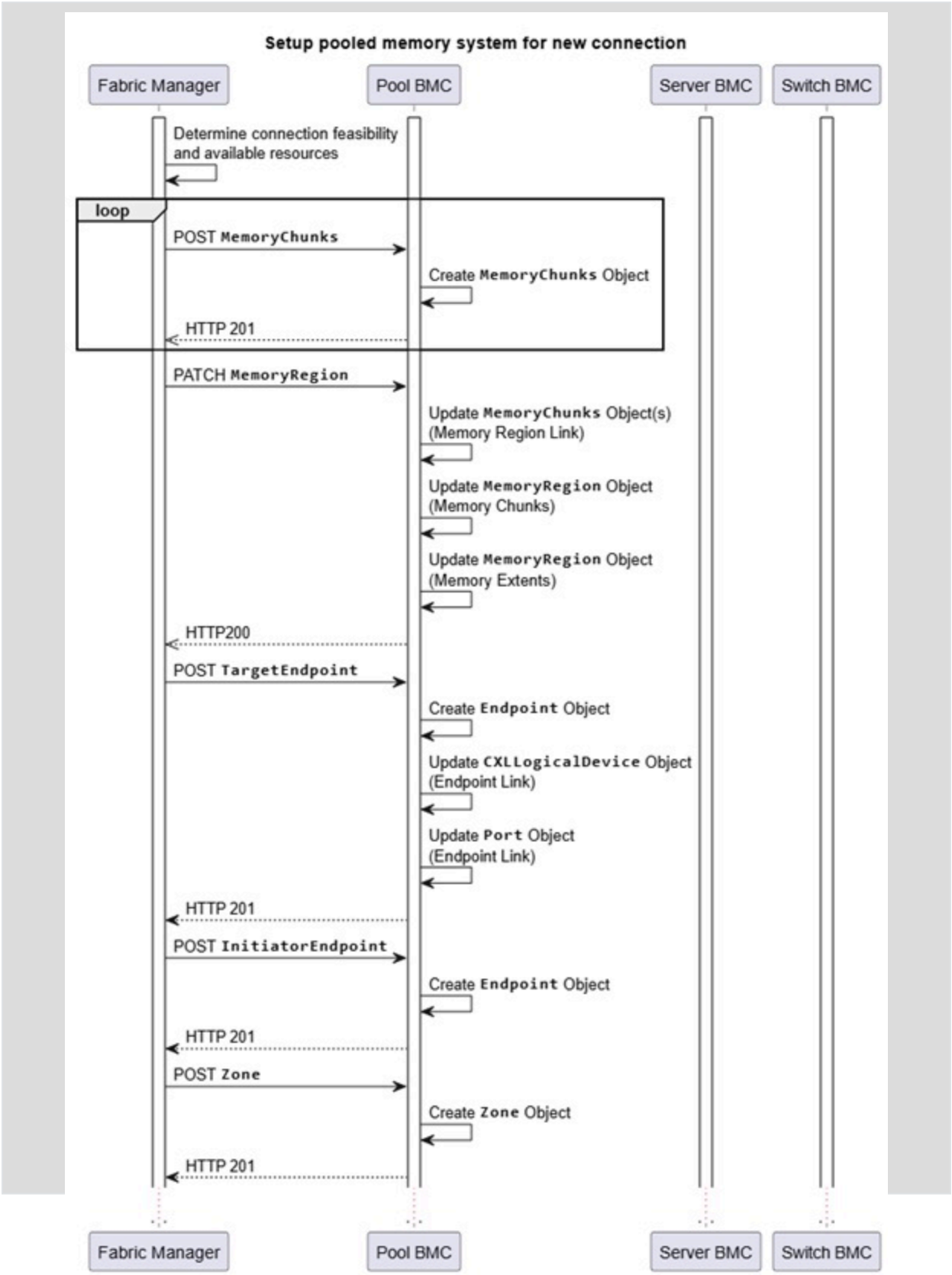
Setup pooled memory system for new connection
Fabric Manager
Pool BMC
Server BMC
Switch BMC
Determine connection feasibility and available resources
loop
POST MemoryChunks
Create MemoryChunks Object
HTTP 201
PATCH MemoryRegion
Update MemoryChunks Object(s) (Memory Region Link)
Update MemoryRegion Object (Memory Chunks)
Update MemoryRegion Object (Memory Extents)
HTTP200
POST TargetEndpoint
Create Endpoint Object
Update CXLLogicalDevice Object (Endpoint Link)
Update Port Object (Endpoint Link)
HTTP 201
POST InitiatorEndpoint
Create Endpoint Object
HTTP 201
POST Zone
Create Zone Object
HTTP 201
Fabric Manager
Pool BMC
Server BMC
Switch BMC

Figure 19 — Pooled memory system setup phase flow

Figure 20 shows the flow of the host system setup phase.

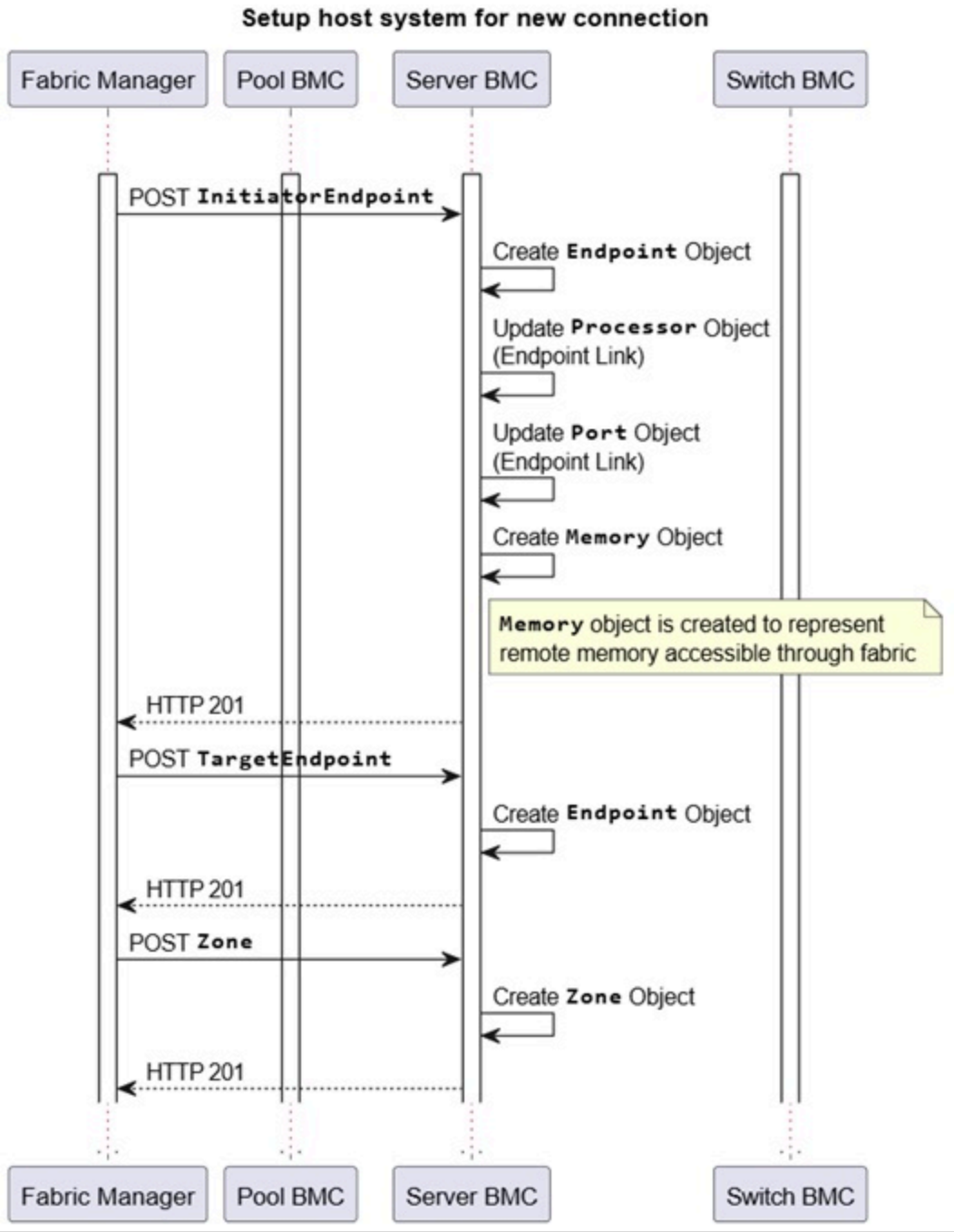

Figure 20 — Host system setup phase flow

Figure 21 shows the flow of the switch system setup phase.

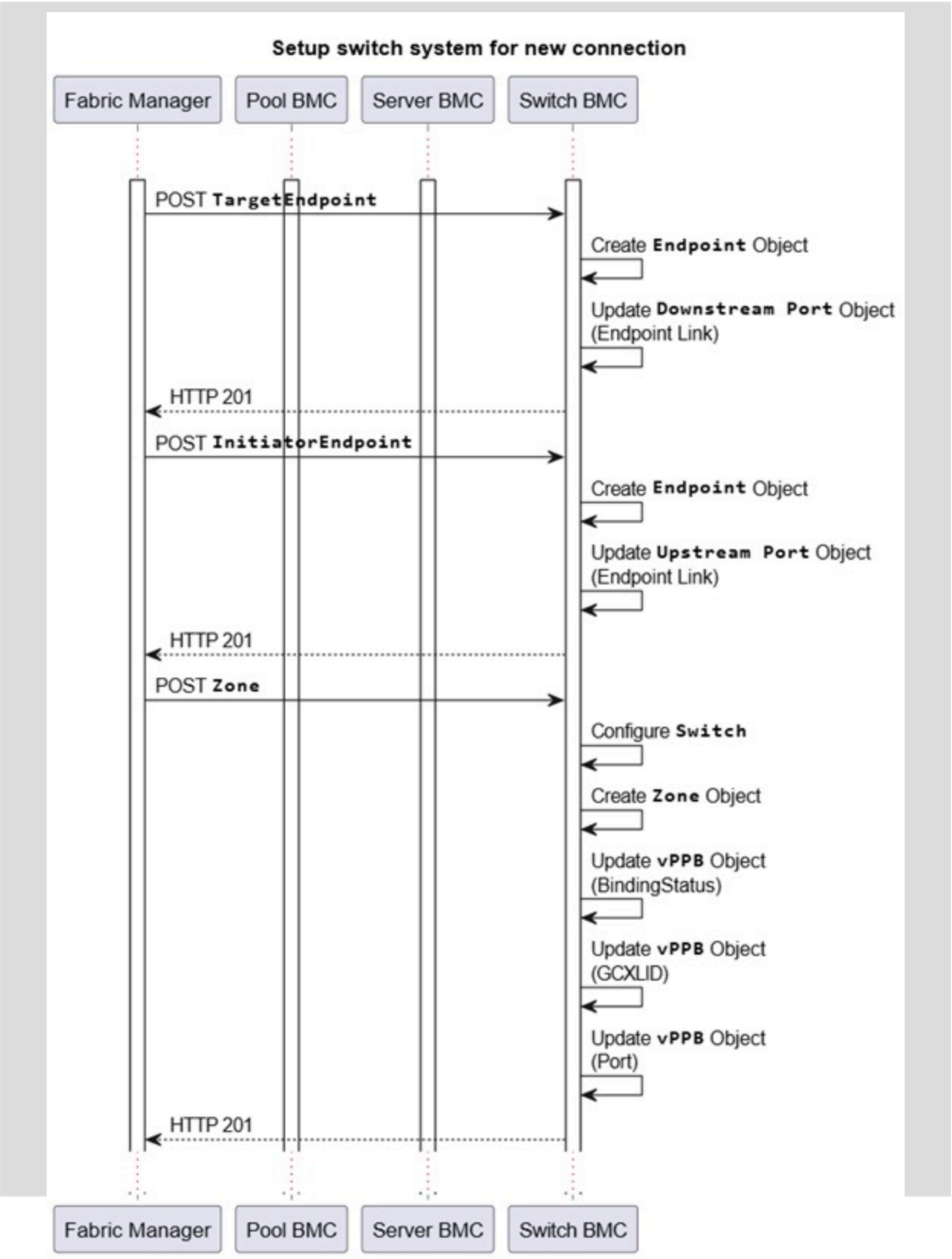
Setup switch system for new connection
Fabric Manager
Pool BMC
Server BMC
Switch BMC
POST TargetEndpoint
Create Endpoint Object
Update Downstream Port Object (Endpoint Link)
HTTP 201
POST InitiatorEndpoint
Create Endpoint Object
Update Upstream Port Object (Endpoint Link)
HTTP 201
POST Zone
Configure Switch
Create Zone Object
Update vPPB Object (BindingStatus)
Update vPPB Object (GCXLID)
Update vPPB Object (Port)
HTTP 201
Fabric Manager
Pool BMC
Server BMC
Switch BMC

Figure 21 — Switch system setup phase flow

Figure 22 shows the flow of the initiator system setup phase.

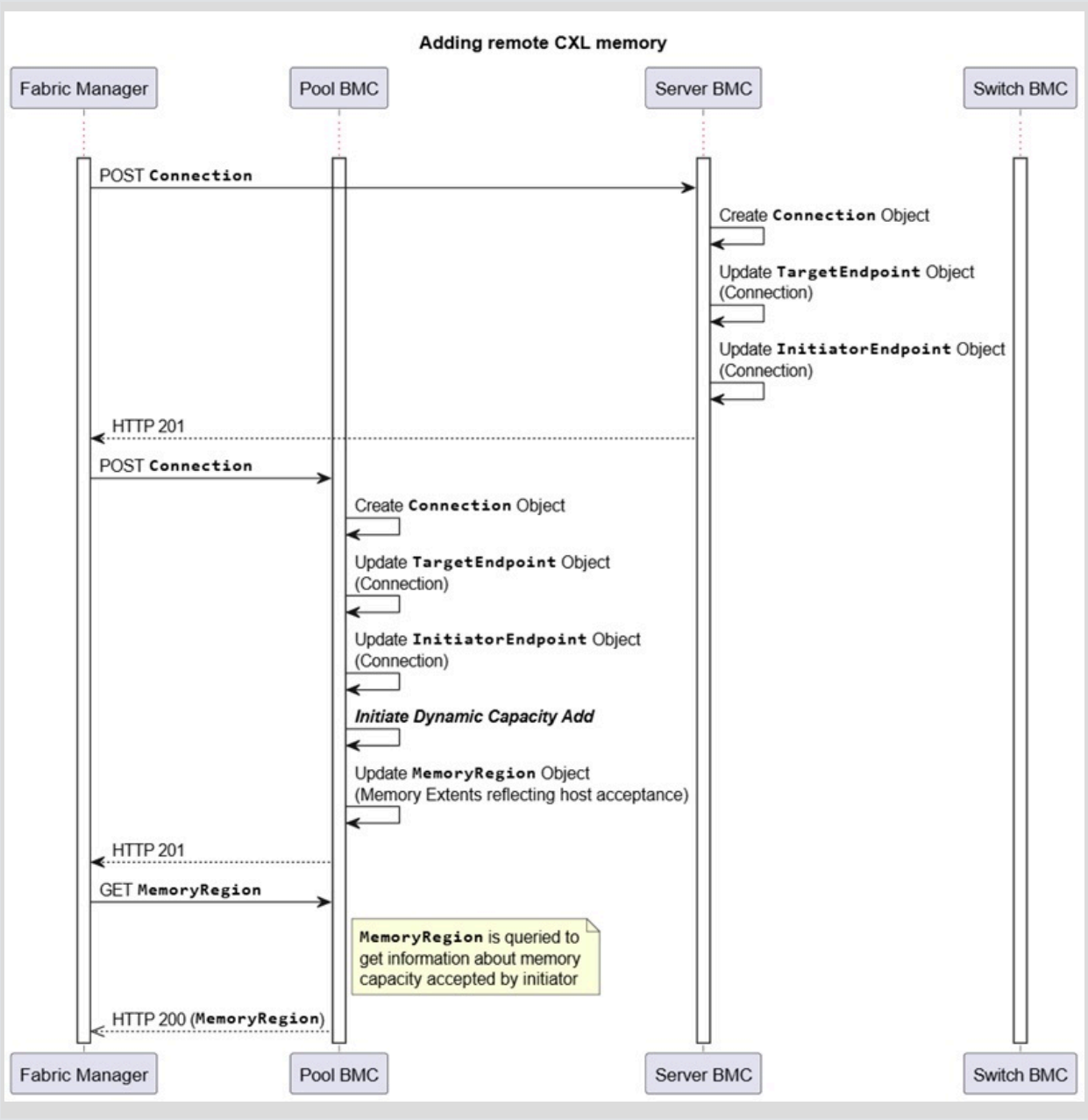

Figure 22 — CXL memory adding phase flow

7.5 Connection termination

Terminating a CXL connection also unfolds in three coordinated steps:

1. **Disconnect host CPU:** The target pooled memory system disconnects the host CPU from the remote memory chunk, releasing access and disabling the operational link.
2. **Deallocate memory chunk:** After disconnection, and if data retention is unnecessary, the pooled memory system can deallocate the memory chunk, erase its contents, and free it for future use.
3. **Update CXL topology:** If no active connections remain, the administrator updates the topology by clearing active paths in the CXL switch, ensuring resources are optimally managed and the fabric maintains its integrity.

> **Note:** In CXL fabric setup, the connection is terminated by the target system (pooled memory), not the host. The host is notified during termination process and may accept, partially accept or reject the release request. The device adjusts next steps to the response of the host.

The following figures show consecutive phases of the detailed fabric connection termination flow.

Figure 23 shows the flow of the CXL memory releasing phase flow.

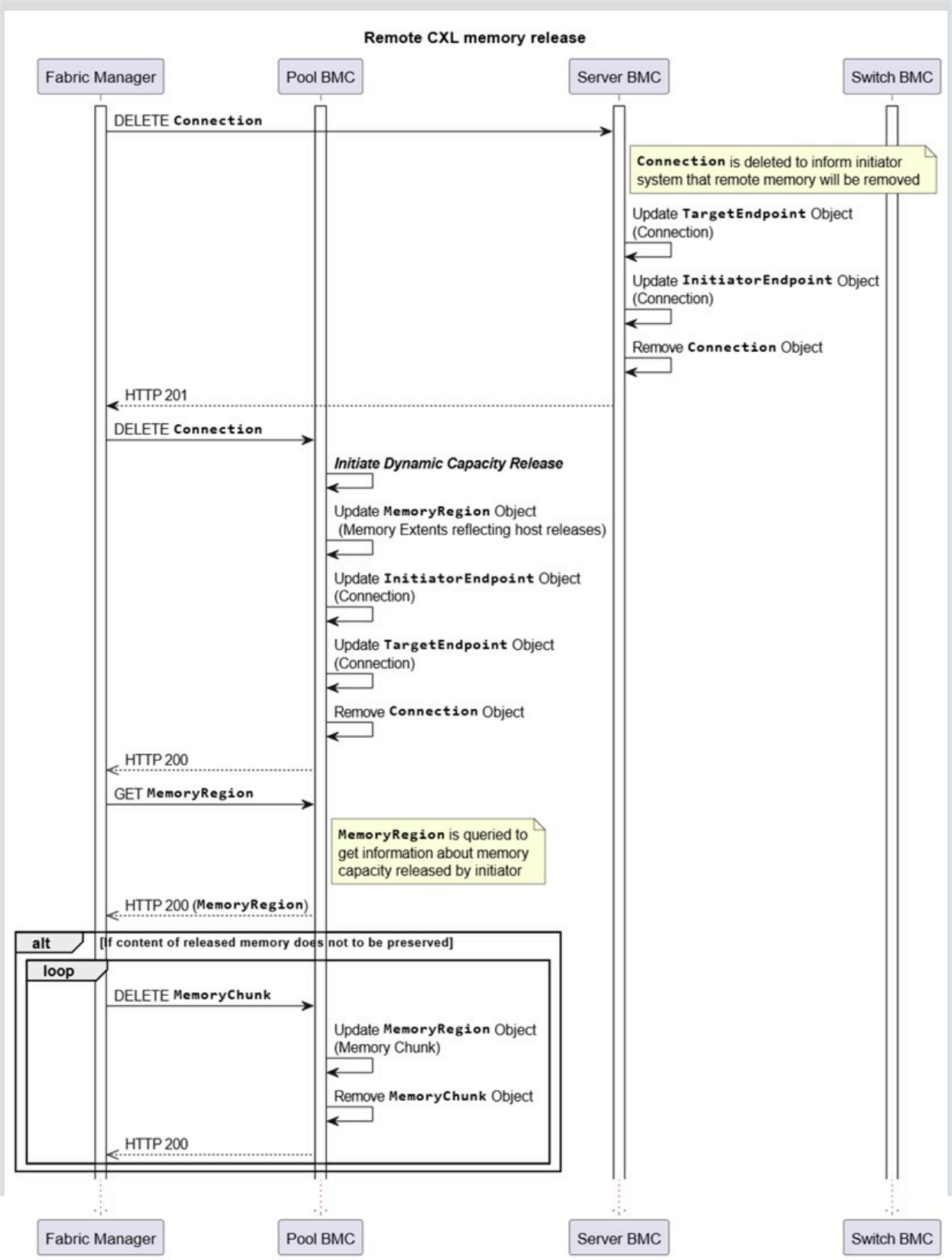
Remote CXL memory release
Fabric Manager
Pool BMC
Server BMC
Switch BMC
DELETE Connection
Connection is deleted to inform initiator system that remote memory will be removed
Update TargetEndpoint Object (Connection)
Update InitiatorEndpoint Object (Connection)
Remove Connection Object
HTTP 201
DELETE Connection
Initiate Dynamic Capacity Release
Update MemoryRegion Object (Memory Extents reflecting host releases)
Update InitiatorEndpoint Object (Connection)
Update TargetEndpoint Object (Connection)
Remove Connection Object
HTTP 200
GET MemoryRegion
MemoryRegion is queried to get information about memory capacity released by initiator
HTTP 200 (MemoryRegion)
alt
[If content of released memory does not to be preserved]
loop
DELETE MemoryChunk
Update MemoryRegion Object (Memory Chunk)
Remove MemoryChunk Object
HTTP 200
Fabric Manager
Pool BMC
Server BMC
Switch BMC

Figure 23 — CXL memory release phase flow

Figure 24 shows the flow of optional phase of the switch system configuration clearing. This phase is performed when no active connection remains between the initiator system and the target system.

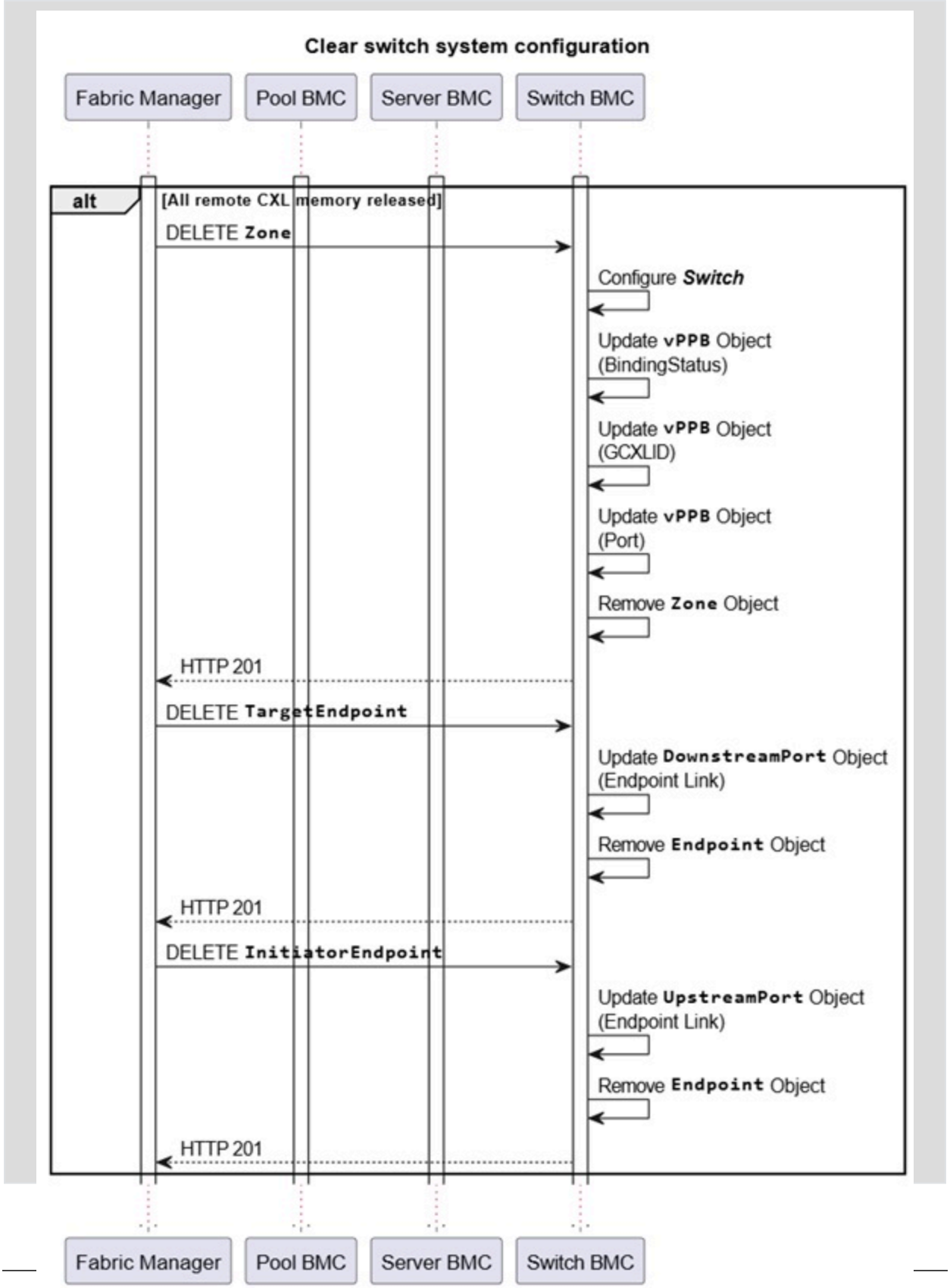
Clear switch system configuration
Fabric Manager
Pool BMC
Server BMC
Switch BMC
alt
[All remote CXL memory released]
DELETE Zone
Configure Switch
Update vPPB Object (BindingStatus)
Update vPPB Object (GCXLID)
Update vPPB Object (Port)
Remove Zone Object
HTTP 201
DELETE TargetEndpoint
Update DownstreamPort Object (Endpoint Link)
Remove Endpoint Object
HTTP 201
DELETE InitiatorEndpoint
Update UpstreamPort Object (Endpoint Link)
Remove Endpoint Object
HTTP 201
Fabric Manager
Pool BMC
Server BMC
Switch BMC

Figure 24 — Clear switch system configuration phase flow

Figure 25 shows the flow of optional phase of the target system configuration clearing. This phase is performed when no active connection remains between the initiator system and the target system.

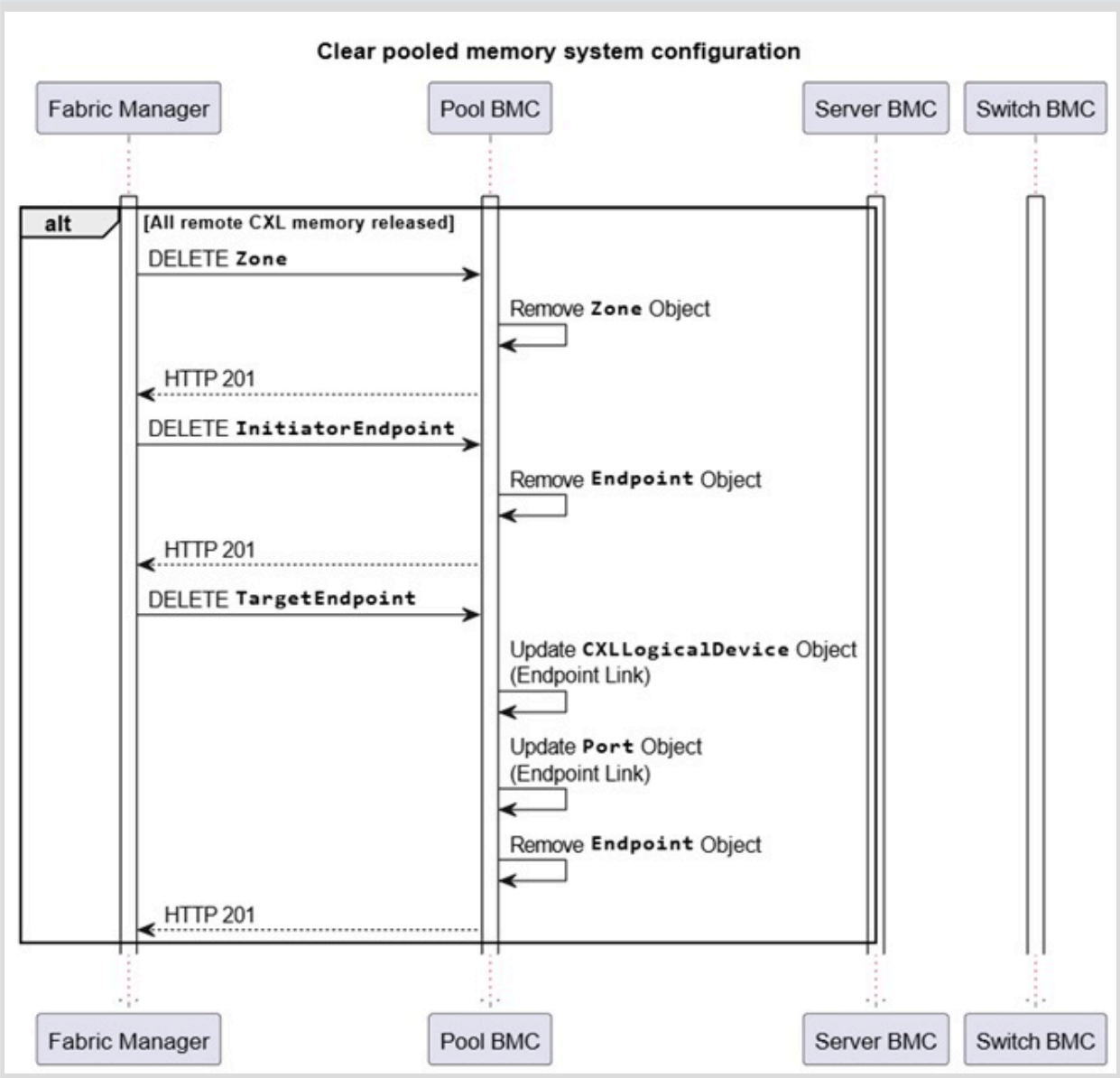

Figure 25 — Clear pooled memory system configuration phase flow

Figure 26 shows the flow of optional phase of the initiator system configuration clearing. This phase is performed when no active connection remains between the initiator system and the target system.

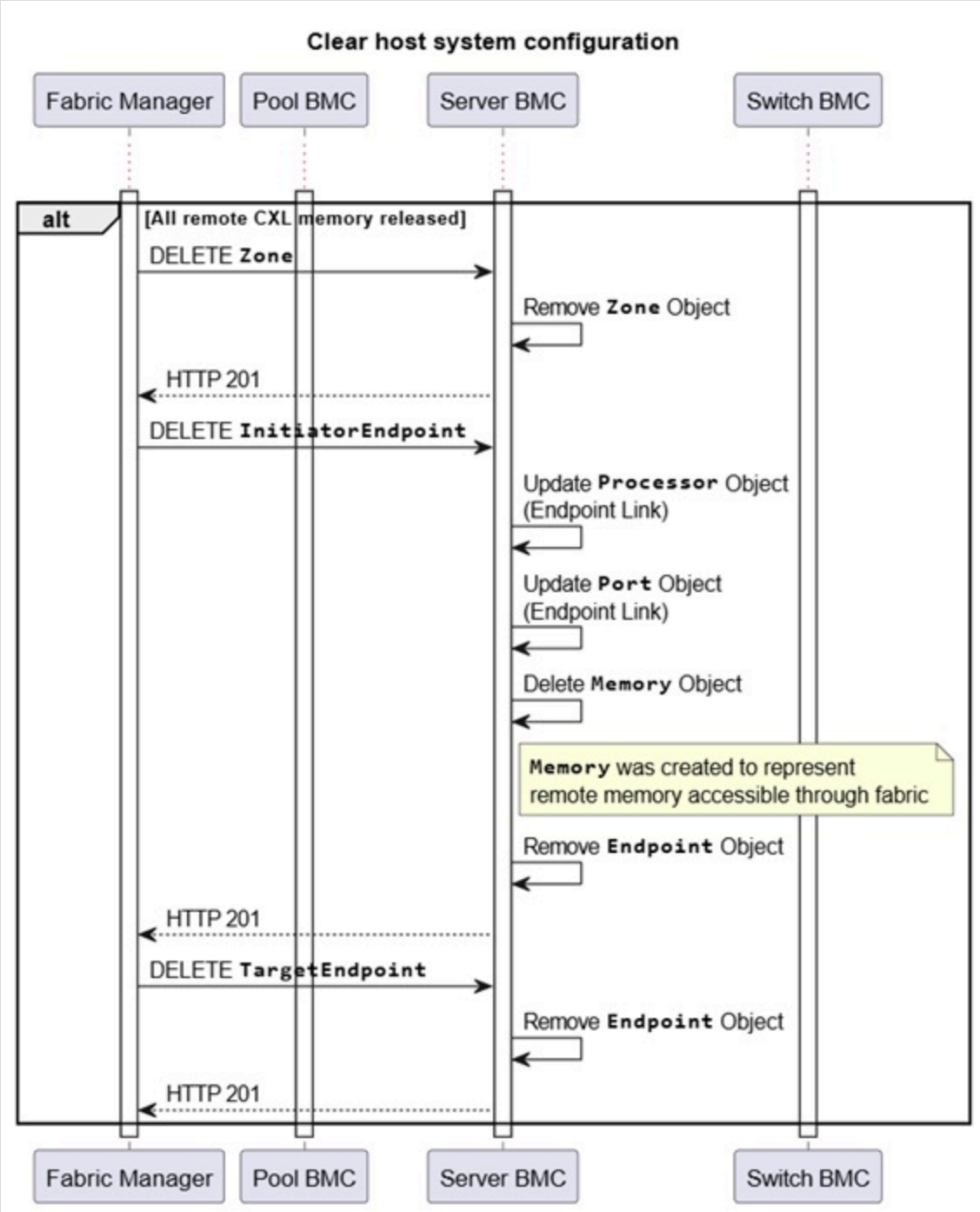

Figure 26 — Clear host system configuration phase flow

8 Representing different types of fabrics

The following sections contain diagrams for representing different types of fabrics with Redfish.

8.1 Ethernet

Figure 27 shows a sample Ethernet fabric and how the `Chassis` and `Fabric` resources are related. The sample fabric contains the following components.

- Chassis `1`: An enclosure with a network adapter containing a single function and port.
- Chassis `2`: A second enclosure with a network adapter containing a single function and port.
- Fabric `Ethernet`: The representation of the Ethernet fabric with its switches and configurations.

The ports for the network adapters in chassis `1` and `2` are connected to the ports on the switch found in fabric `Ethernet`. The ports for each network adapter also show a relationship to a network device function to show the port usage of the functions.

Endpoints `EP1` and `EP2` are within fabric `Ethernet` to represent the fabric-view of the network device functions found in chassis `1` and `2`. Both of these endpoints belong to zone `1`, signifying routing is enabled between them. Zone `1` is also associated with address pool `1`, which contains networking configurations applied to the endpoints in the zone.

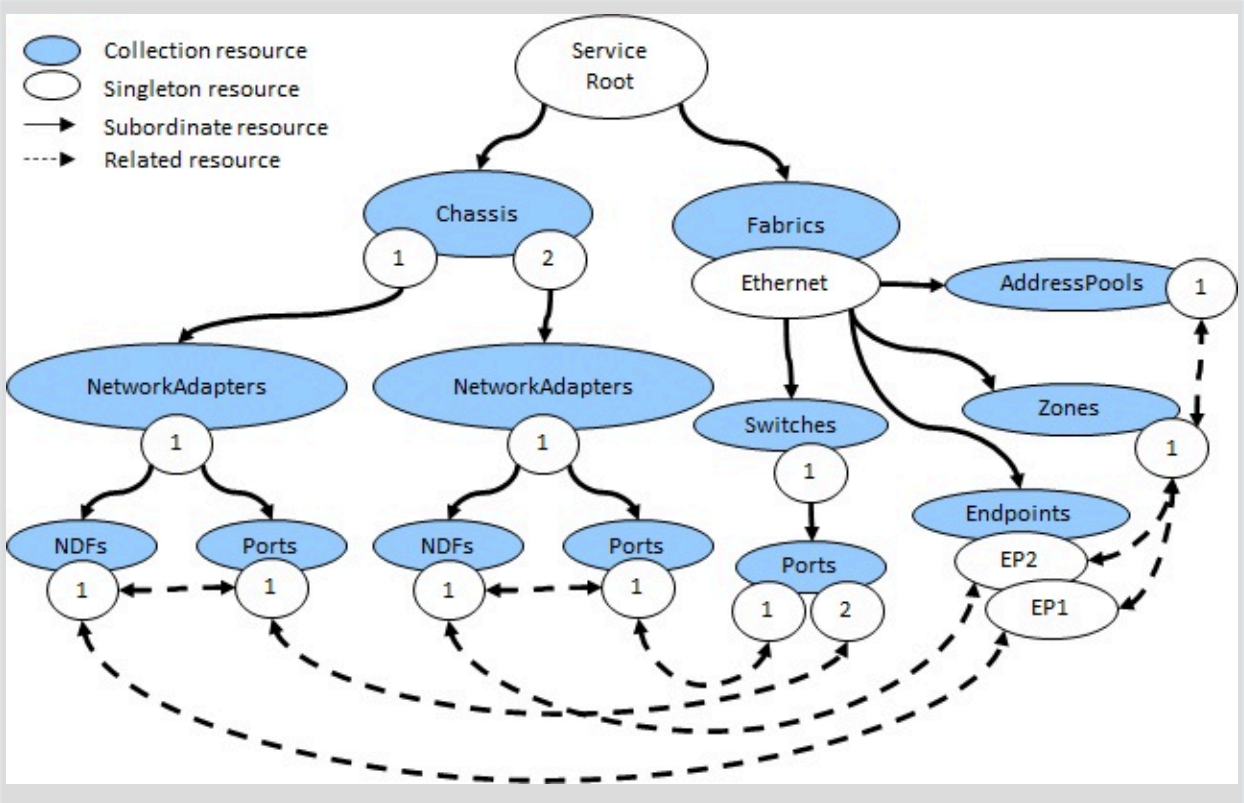

Figure 27 — An example Ethernet fabric

8.2 SAS

Figure 28 shows a sample Serial Attached SCSI (SAS) fabric and how the `ComputerSystem`, `Chassis`, and `Fabric` resources are related. The sample fabric contains the following components.

- System `1`: A system with a storage controller to access the SAS fabric.
- Chassis `1`: An enclosure containing drives that are accessible over a SAS fabric.
- Fabric `SAS`: The representation of the SAS fabric with its switches and configurations.

System `1` contains a single storage subsystem with one storage controller. The storage controller is represented in the fabric as endpoint `Initiator`. The storage controller shows its port is connected to port `3` on the switch found in fabric `SAS`.

Chassis `1` contains two drives, `D1` and `D2`. Each drive is represented in the fabric as endpoints `D1` and `D2`. These endpoints show connectivity to switch ports `1` and `2`.

Fabric `SAS` contains a single zone with all endpoints belonging to the zone. This signifies routing is enabled between all three endpoints.

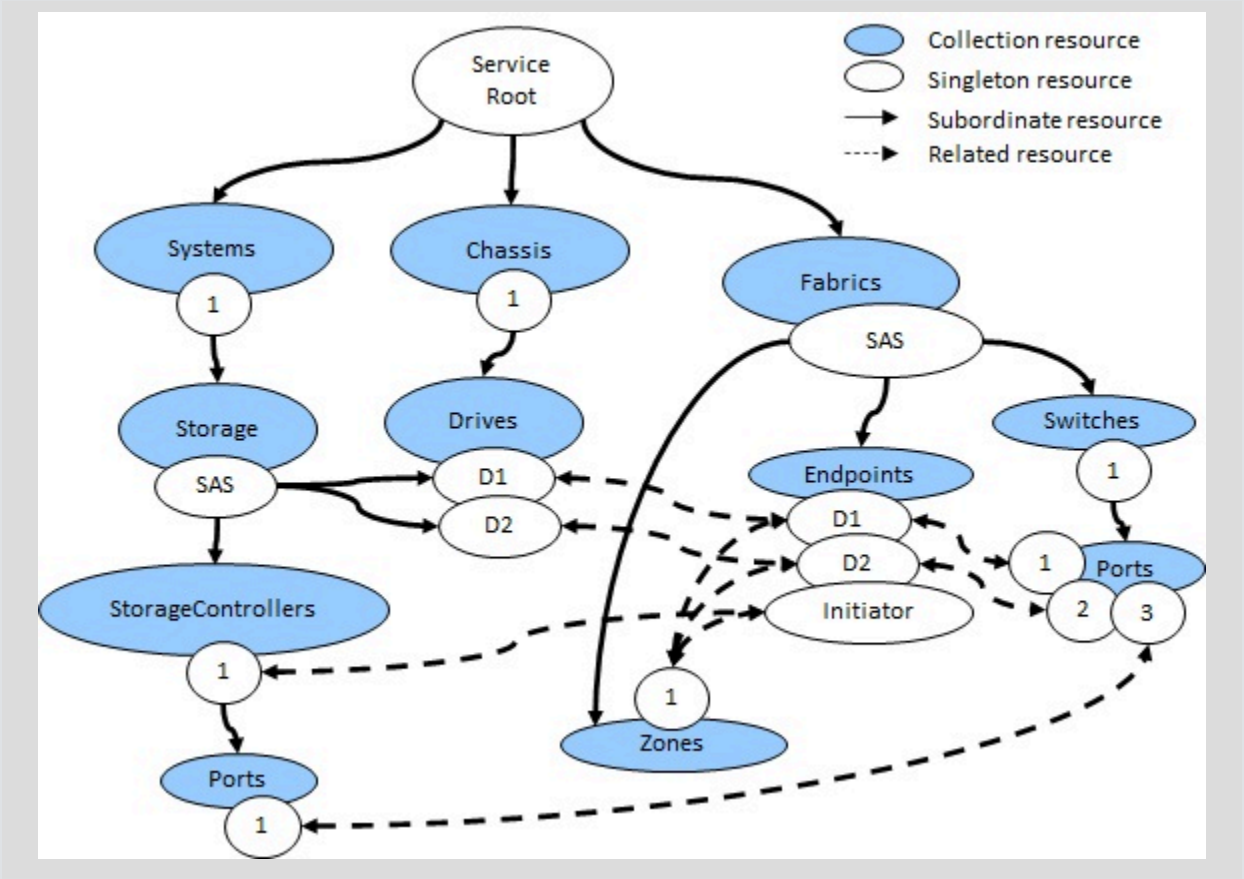

Figure 28 — An example SAS fabric

8.3 PCIe

Figure 29 below shows a sample PCI Express(PCIe) fabric and how the `Fabric` resources are related to other types of resources. The sample fabric contains the following components.

- System `1` : A system with a storage controller and NIC connected to a PCIe switch.
- Fabric `PCIe` : The representation of the PCIe fabric with its switches and configurations.

System `1` contains an Ethernet interface and is represented in the fabric as endpoint `NIC1` . The endpoint shows connectivity to switch port `Down1` .

System `1` also contains a single storage subsystem with one storage controller. The storage controller is represented in the fabric as endpoint `SAS-HBA`. The endpoint shows connectivity to switch port `Down2`.

Fabric `PCIe` contains an additional endpoint named `Root` and represents the PCIe root port for system `1`. The endpoint also shows connectivity to switch port `Up`. The fabric also contains a single zone with all endpoints belonging to the zone. This signifies routing is enabled between all three endpoints.

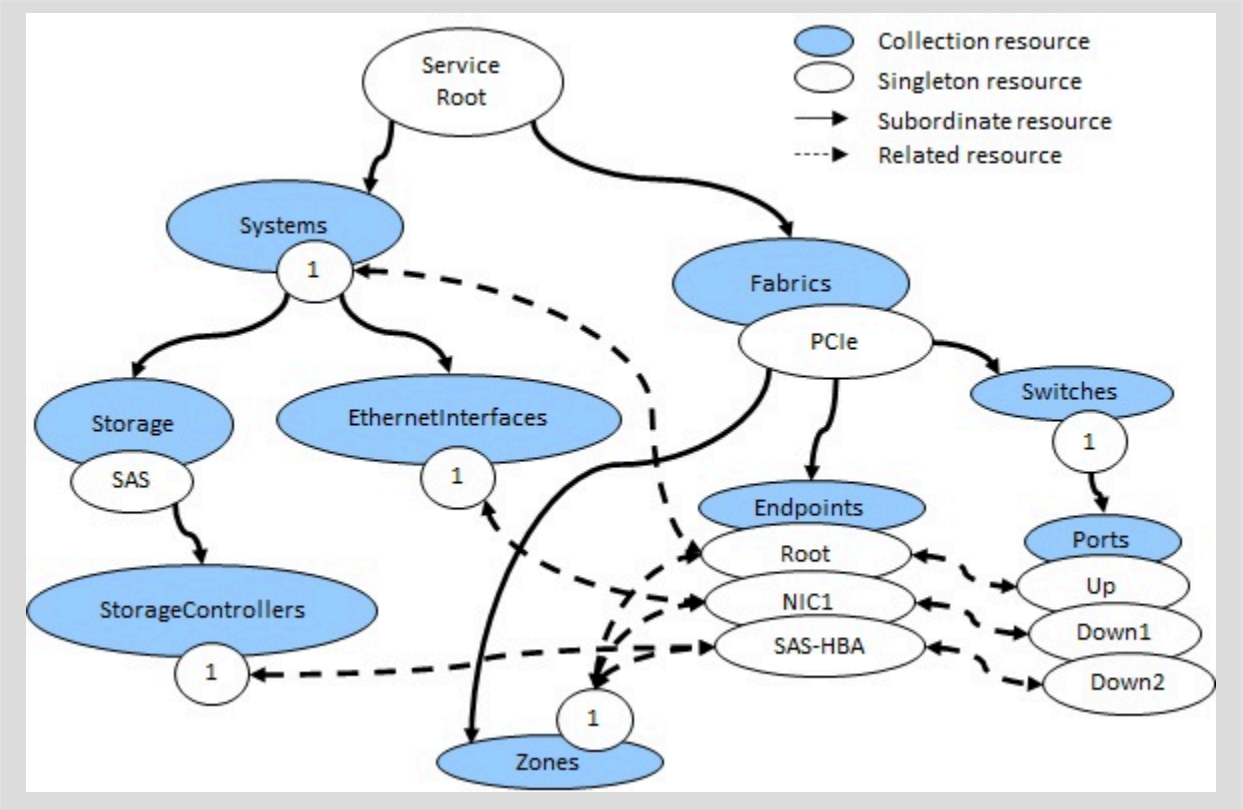

Figure 29 — An example PCIe fabric

8.4 CXL

A Compute Express Link (CXL) fabric is defined in Redfish using the `Fabric`, `Chassis`, and `ComputerSystem` resources. This section shows the physical topology of an example CXL-based system, the routing of packets through a CXL switch, access to resources in a CXL fabric, device identification, logical device creation, and how memory domains and memory chunks are utilized. The CXL to Redfish Mapping Specification provides a mapping of CXL CCI and FMAPI commands to Redfish resources and properties.

8.4.1 Physical topology

The following figure shows an example Compute Express Link (CXL) fabric consisting of a host CPU with CXL capabilities, a CXL switch, and a multi-logical device (MLD) connected to the switch.

- System `1`: A Host system containing processors with CXL ports that enables it to be attached to a CXL fabric.
- Fabric `CXL`: This represents a set of CXL switches and their configurations. The switch can be configured to be in hierarchy-based routing (HBR) mode or port-based routing (PBR) mode.
- Chassis `CXL-MLD1`: A representation of a CXL MLD device with 2 logical devices.

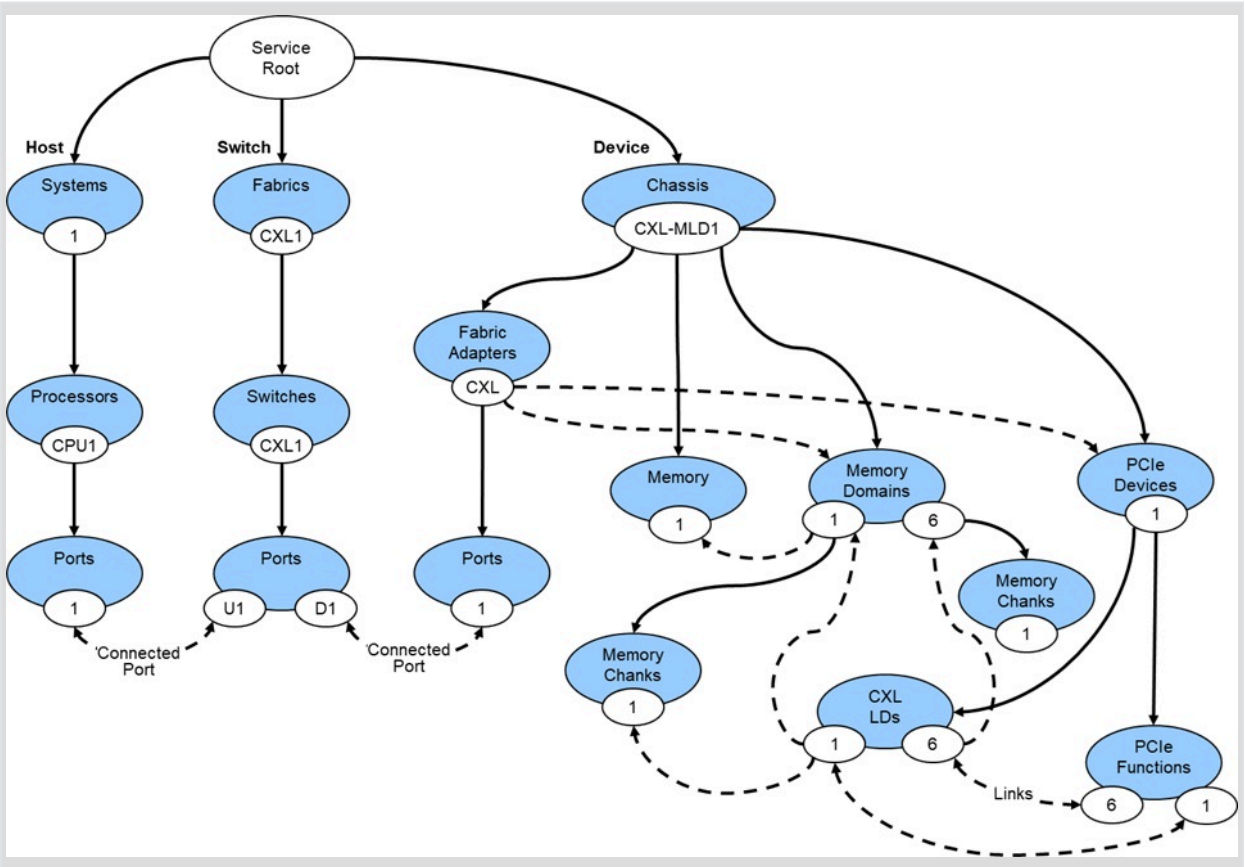

Figure 30 — CXL physical representation

Figure 30 shows the physical connectivity between different components of the CXL fabric. In a CXL fabric, the processors should be CXL capable, i.e., processors should contain CXL ports. Therefore, a fabric adapter is not required to attach the host to the fabric as the processor itself acts as a fabric adapter. This can be observed on the left hand side of Figure 30 under **Host**.

The far right of Figure 30 represents a CXL memory device. Unlike the host, physical connectivity of the memory devices to the fabric require a fabric adapter to connect the device to the fabric as observed under **Device**. The device chassis `CXL-MLD1` represents a multi-logical device (MLD) with two logical devices (LDs), represented by `1` and `6` in the diagram. A CXL device is also a PCIe device and each PCIe function corresponds to an LD as shown in Figure 30. Therefore, the LDs are linked with corresponding PCIe functions. If the `CXL-MLD1` chassis contain multiple MLD devices, there will be multiple `FabricAdapter` resources, `MemoryDomain` resources, and `PCIeDevice` resources associated with each instance of an MLD device. Therefore, the links between the `FabricAdapter`, `MemoryDomain`, and `PCIeDevice` resources represent the memory domains and PCIe devices that can be reached from the fabric adapter.

A CXL device can also be directly attached to a host. However, to build a system with large number of devices, it is typical to use a CXL switch for additional capacity. Additionally, it will be beneficial to use a CXL switch if a CXL device being connected is an MLD or a multi-headed device (MHD). The CXL switch contains a set of upstream ports (`U1`) to which hosts are physically connected, and a set of downstream ports (`D1`) to which CXL devices are physically connected.

To summarize, Figure 30 represents a host CPU with a CXL port (`1`) connected to a CXL switch (`CXL1`) upstream port (`U1`). The CXL switch could have multiple downstream ports of which `D1` shown here is connected to a CXL MLD device `CXL-MLD1`. The CXL device is connected to the CXL fabric via a fabric adapter (`CXL`). The CXL device ia a PCIe device (`1`) that contains logical devices (LDs) `1` and `6`, internally represented as PCIe functions `1` and `6` respectively. The `CXL-MLD1` chassis contains memory domains and memory chunks that are assignable to different hosts.

8.4.2 Routing

Figure 31 below shows the routing configuration for fabric `CXL1`. The fabric `CXL1` can contain two types of endpoints: *initiators* and *targets*. The processors from where a CXL transaction originates can be described as *initiators*. The CXL logical device (LD) where CXL transactions terminate can be described as *targets*. Note that in advanced configurations, a CXL device can also initiate transactions. This is not covered in this white paper. All endpoints belong to a zone of type `ZoneOfEndpoints`. The traffic can flow between any pair of endpoints present within this zone.

Each intermediate CXL switch in the path from the initiator to the target is configured as host-based routing (HBR). This configuration includes binding a virtual PCI-PCI Bridge (vPPB) to a downstream port through which the device is accessible. The vPPB belongs to a virtual CXL switch (VCS) that is part of the virtual PCIe hierarchy to which the host belongs. If the CXL switches are configured as port-based routing (PBR), the appropriate PBR tables in the intermediate CXL switches and access control lists (ACLs) of the global FAM device (GFD) should be configured appropriately.

There is a special endpoint described in the fabric that represents a management device called fabric manager endpoint shown in red as `T3` in Figure 31. This endpoint is used to configure the device and must have LD-ID of `FFFFh`. A similar endpoint is present in the CXL switch and is used to configure the CXL switch, shown in green as `T4`.

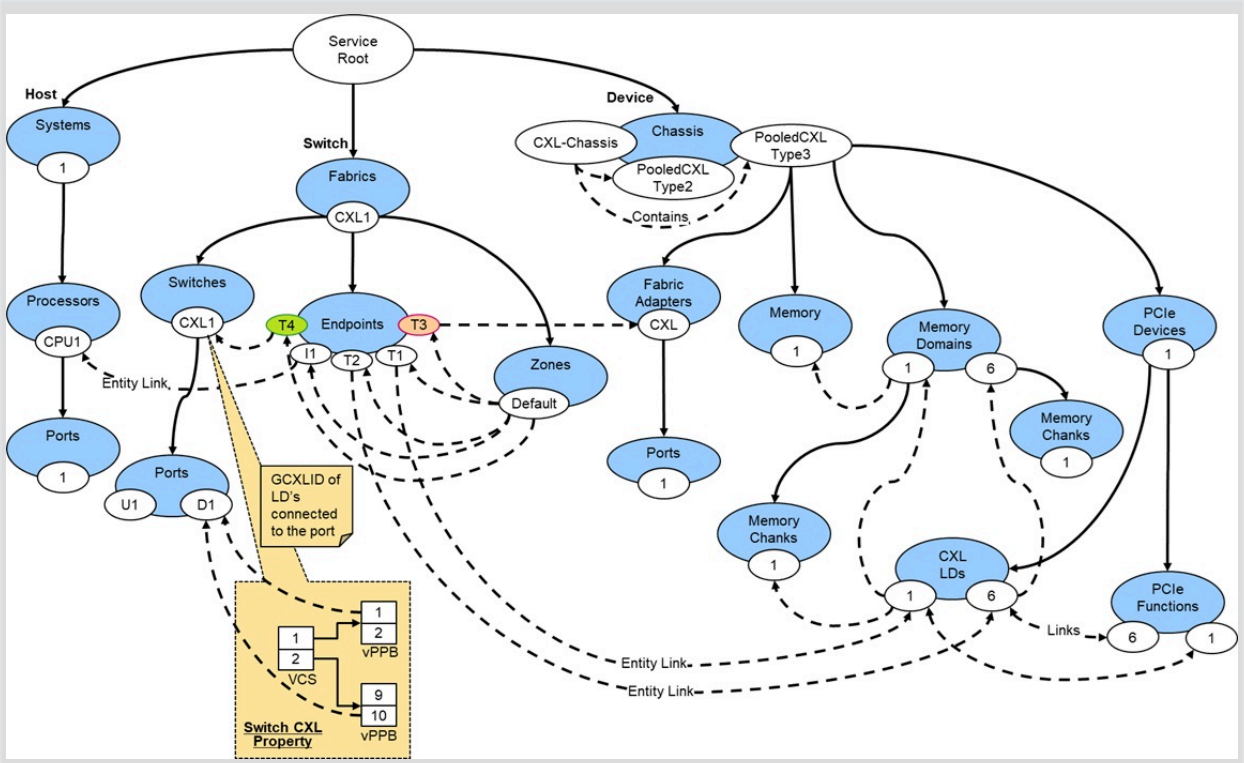

Figure 31 — CXL fabric representation for routing

8.4.3 Access

Figure 32 shows access control to the resources in the `CXL1` fabric. This is achieved by creating a `Connection` resource and adding the two endpoints to be connected. Figure 32 shows this mapping with endpoints `I1` and `T1`. It also assumes the physical connections between CPU and LDs represented in Figure 31 exist. The `Connection` resource creation should be followed by including memory chunk `1` to the `Connection` resource in case of static memory allocation. However, in case of dynamic memory allocation, new memory chunks should be exposed to `CPU1` enabling access the memory and auto populates the memory chunk information using the extents allocated by the device. This is described further in the CXL memory chunks section.

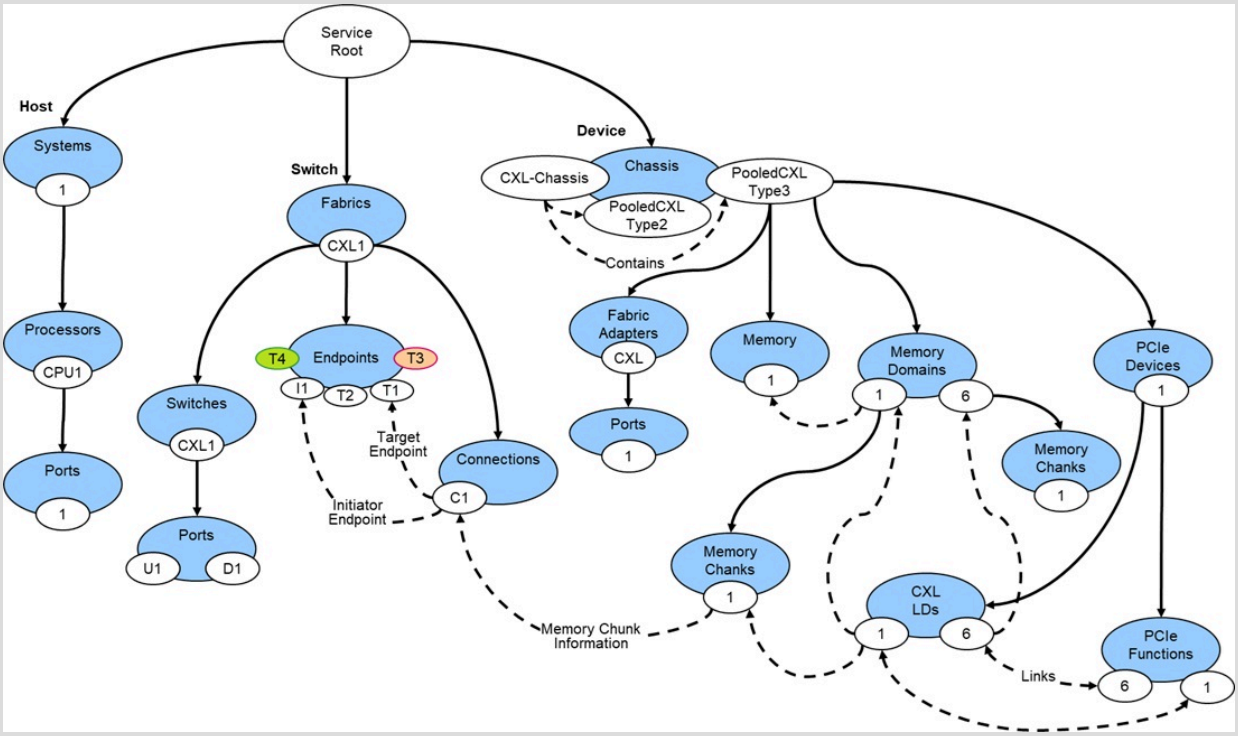

Figure 32 — CXL access control

8.5 Gen-Z

The following figures show a sample Gen-Z fabric and how the `ComputerSystem`, `Chassis`, and `Fabric` resources are related. The sample fabric contains the following components.

- System `1`: A system with a fabric adapter to access resources on the Gen-Z fabric.
- Chassis `Gen-Z`: An enclosure containing memory that is partitioned into chunks that are accessible over a Gen-Z fabric.
- Fabric `Gen-Z`: The representation of the Gen-Z fabric with its switches and configurations.

Figure 33 shows the physical connectivity between the devices on the fabric. System `1` and chassis `Gen-Z` both contain a fabric adapter with one port. Fabric `Gen-Z` contains a single switch with two ports. The ports on system `1` and chassis `Gen-Z` connect to the ports on the switch found in the fabric `Gen-Z`.

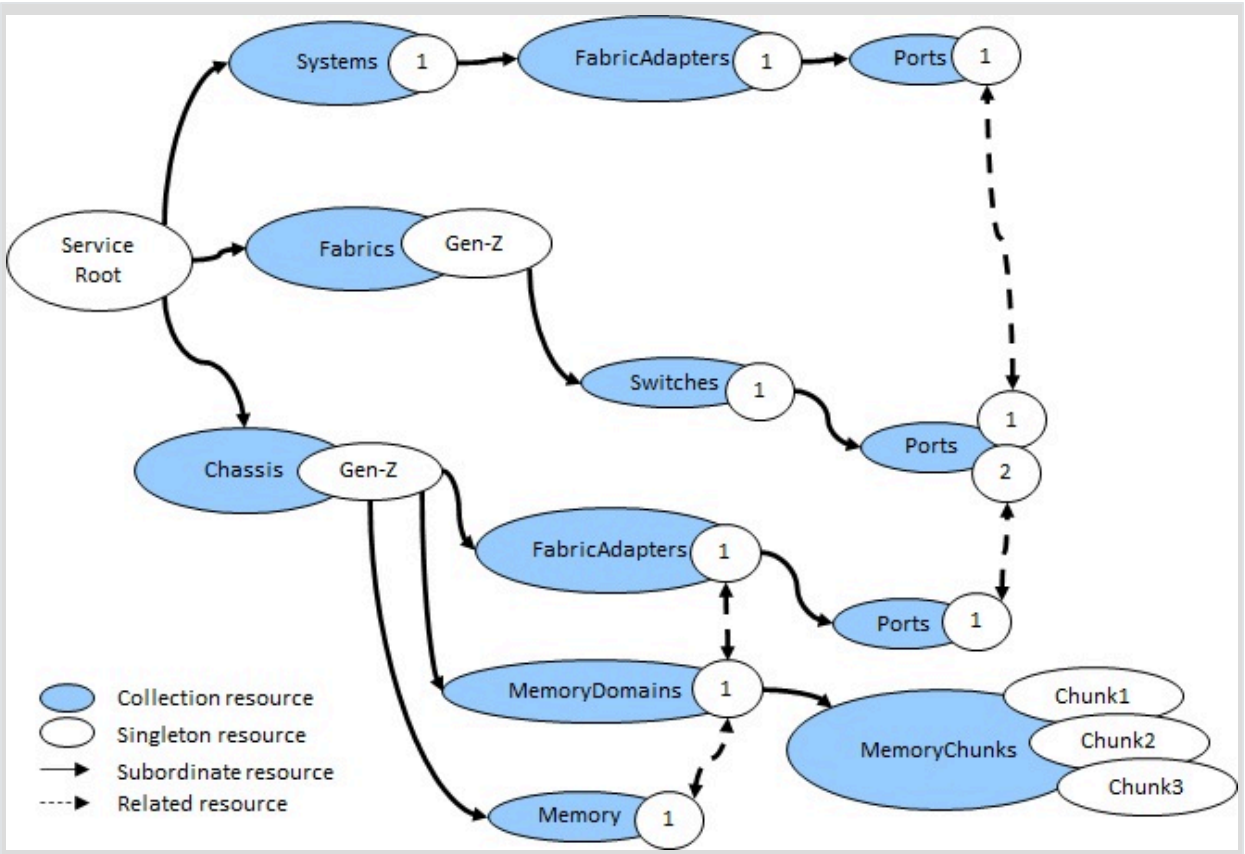

Figure 33 — An example Gen-Z fabric, physical connectivity

Figure 34 shows routing and addressing configurations for the fabric `Gen-Z`. Fabric `Gen-Z` contains two endpoints: one named `Initiator` to represent the connectivity of system `1` to the fabric, and one named `Target` to represent the connectivity of chassis `Gen-Z` to the fabric. Fabric `Gen-Z` contains a single zone where both endpoints belong to the zone to show that traffic is routable between endpoints `Initiator` and `Target`. Fabric `Gen-Z` also contains a single address pool to control Gen-Z Component Identifier and Subnet Identifier assignments to the endpoints `Initiator` and `Target`.

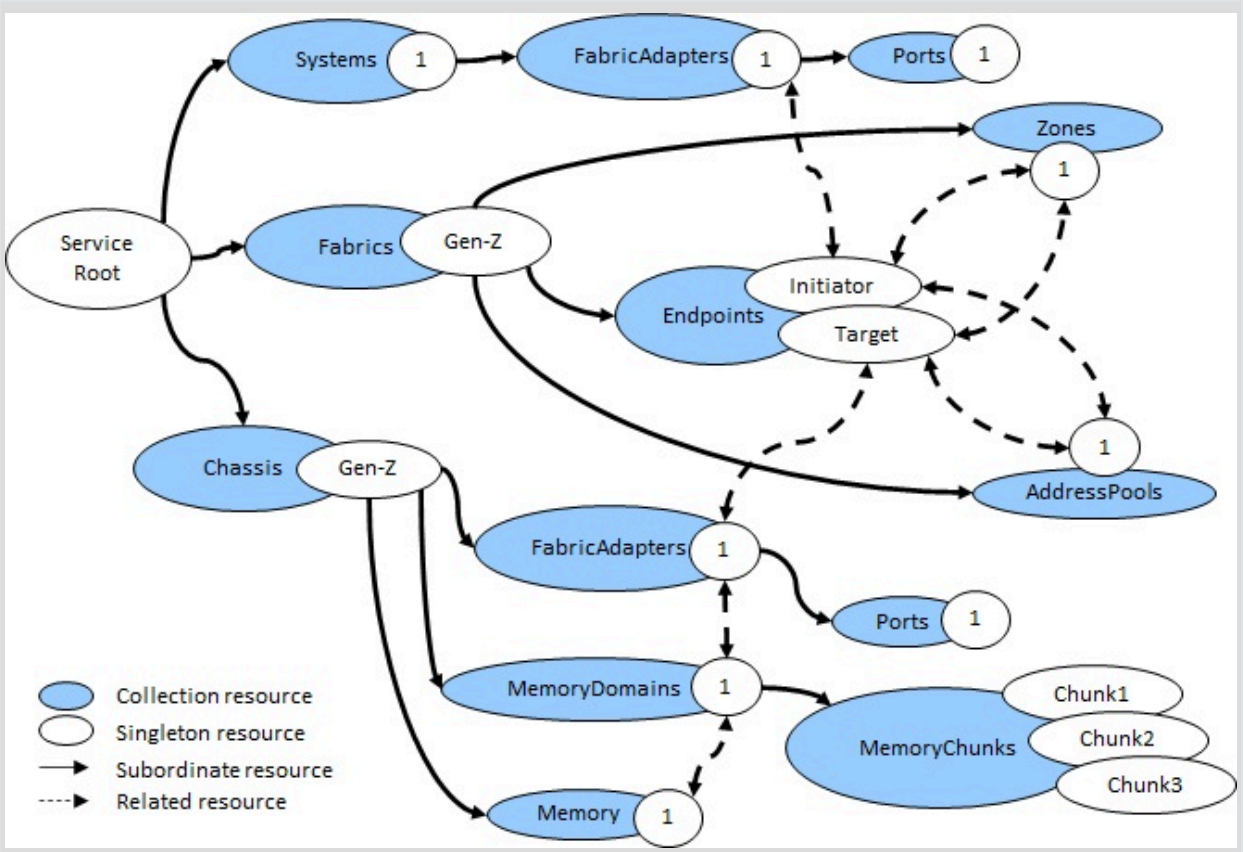

Figure 34 — An example Gen-Z fabric, routing and addressing

Figure 35 shows access control to resources for the fabric `Gen-Z`. Fabric `Gen-Z` contains one connection named `Conn1`. `Conn1` is configured to allow the endpoint `Initiator` to access memory chunks `Chunk1`, `Chunk2`, and `Chunk3` when connecting to endpoint `Target` over the fabric.

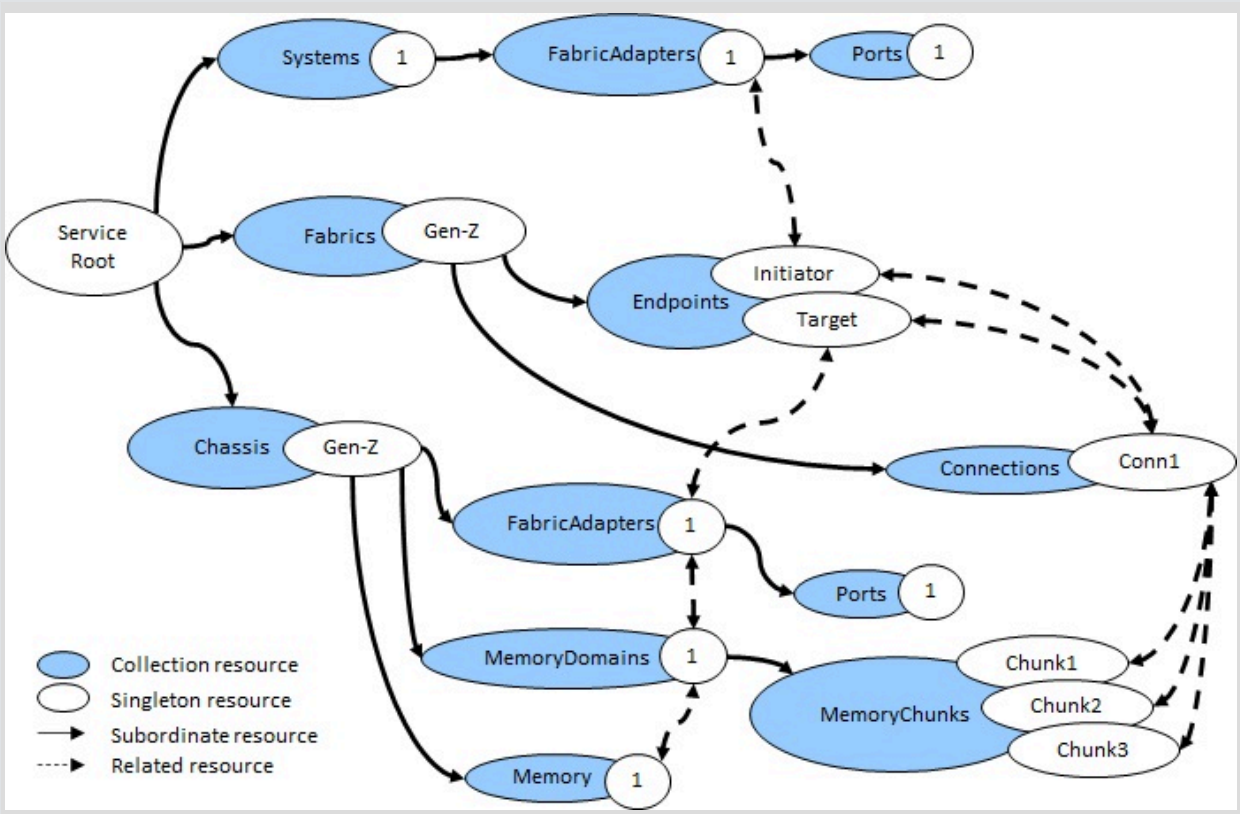

Figure 35 — An example Gen-Z fabric, access control

9 Fabric model and composability

Redfish has a data model for composability where clients are able to specify components in a service and build systems on demand. There are aspects of the fabric model that overlap concepts in composability, such as establishing routing paths between components. While there is overlap in some of the usage, both models can coexist on the same service. In cases where both models are present on the same service, performing requests in one area of the model might have impacts on the other. For example, if a client performs a composition request to build a new system, the service may perform fabric configurations and routing in order to satisfy the client's request, which are then reflected in the fabric model.

10 Appendix A: References

- "Simple SAS Fabric" and "NVMe-oF JBOF" Mockups: redfish.dmtf.org/redfish/v1
- DMTF DSP0288, *CXL to Redfish Mapping Specification*, https://www.dmtf.org/dsp/DSP0288
- DMTF DSP2050, *Redfish Composability White Paper*, https://www.dmtf.org/dsp/DSP2050
- "Compute Express Link Specification": computeexpresslink.org/cxl-specification
- "Serial Attached SCSI Specification": www.opencompute.org/documents/datacenter-sas-sata-device-specification-rev-1-0-pdf
- "PCI Express Specification": pcisig.com/specifications/pciexpress

11 Appendix B: Change log

Version	Date	Description
1.1.0	2025-12-04	Updated to incorporate Compute Express Link (CXL) technology.
1.0.0	2022-04-07	Initial release.