DriveScale Architecture for Data Center Scale Composability

by Brian Pawlowski, Chris Unkel, Jim Hanko, Jean-François Remy
Overview of talk

- Defining composable infrastructure
- What the administrator sees
- DriveScale design and architecture
- Future directions
- Recapitulation
- Questions
Defining composable infrastructure
DriveScale – Disaggregate then Compose

Captive, fixed DAS

Disaggregate

Composable

Right sized

Purchase Time Defined Infrastructure

Software Defined Infrastructure
Composable Infrastructure – blade chassis scale

• Single vendor hardware
• Low scalability

10 Servers
100 Drives
Composable Infrastructure – data center scale

10,000 Servers

100,000 Drives

• Choose optimized dense diskless servers, optimized storage platforms for HDD and SSD
• Multivendor – mix and match
• Multiprotocol Ethernet (iSCSI, ROCEv2, NVME over TCP) for wide variety of use cases
• Composer platform server(s) configuration: dual socket x86, 128GB DRAM, SSD
Composable Infrastructure at scale

Compose any compute to any drive

Ethernet to storage adapters

Logical View

Hadoop  Cassandra  Aerospike
Think Bare Metal Virtualized Infrastructure for Linux, at scale – but without the VMs.
The devil is in the details

- Breaking up (disaggregation) is easy (in my pocket)
- Composable Infrastructure – at scale – is hard
  - Create secure durable bindings between servers and drives
  - Plumb end-to-end Linux storage stack over multiple protocols, multipathing, file systems, RAID configurations, hardware vendors and storage interconnect technologies, and encryption/key mgmt.
  - Persist server/drive bindings through rolling upgrades, multi-version software support, and failed data path component replacement
  - Non-disruptive cluster expansion
  - Retrying configuration changes in face of transient network errors

- Manage and automate complexity at scale
What the administrator sees

See demo at https://www.youtube.com/watch?v=PPxyM-liQo
DriveScale Composer – Inventory

- Servers: 4/8
- Drives: 24/24
- Switches: 3/3
- JBODs: 1/1
- Adapter cards: 2/2
- Composers: 3/3
DriveScale Composer – Quick Cluster
DriveScale Composer – Logical Cluster
DriveScale design and architecture
## Terminology for this talk

<table>
<thead>
<tr>
<th><strong>Disaggregation</strong></th>
<th>Breaking up of compute and storage into discrete elements on an Ethernet network</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Composition</strong></td>
<td>Or composability – the dynamic secure binding of servers to drives</td>
</tr>
<tr>
<td><strong>Logical Cluster</strong></td>
<td>Secure closed (non-overlapping) set of durable server and drive bindings used to host a single scale-out application</td>
</tr>
<tr>
<td><strong>Configuration</strong></td>
<td>The set of logical cluster and element configurations written by the DriveScale composer, read by agents</td>
</tr>
<tr>
<td><strong>Inventory</strong></td>
<td>The set of servers, drives, and storage adapters written by DriveScale agents. Ephemeral status znodes.</td>
</tr>
</tbody>
</table>

### server
In composable deployment, a thin (diskless) compute node or element

### drive
A whole HDD, SSD or an (SSD) slice – also called a *disk*

### HDD
Hard Disk Drive – standard form factors are 2.5" and 3.5"

### SSD
Solid State Drive (form factor irrelevant)

### slice
A slice is a discrete persistent portion of a single SSD that is presented to a server as a single drive

### storage adapter
An Ethernet (iSCSI or NVME) to JBOx adapter – presents drives (12, 24, 60, or 80 drives are typical sizes) – simply provides multipath access to drives for high availability

### JBOx
Just a Bunch Of Disks (HDD) or Flash (SSD)

### EBOx
Ethernet-attached Bunch Of Disks (HDD) or Flash (SSD)

### ZooKeeper
A centralized service for maintaining configuration information, naming, providing distributed synchronization
Design goals

- Scalable (key goal from which much followed)
- Durable
- Highly available
- Correct
- Performant
- Recoverable
DriveScale Composer Platform (control and data plane)

DriveScale Composer Platform (triple replica)

- **3rd Party Orchestration**
- **GUI**
- **API**
- **DriveScale Cloud Central**

Server Agents

- **Server pool**
- **Data Path**
  - + LLDP

Network switches

Data Path
  - + LLDP

Storage Adapter Agents

- **Drive pool**
  - (HDD, SSD, SSD slices)

User level server and storage agents share 90% of their code

Configuration recovery snapshots, alert logs, software updates

DriveScale

Cloud Central

Drive pool

(HDD, SSD, SSD slices)
DriveScale Composer platform functions

- Provides durable bindings of servers to drives (HDD, SSD, SSD slices)
- Stores the auto-discovered inventory from disaggregated server and storage adapters
- Create, expand, manage and report on Logical Clusters
- Health monitoring and alarm notification
  - Notification management of all agents
- Template driven (includes user defined)
  - Templates as expression of each app reference architecture
- Manage role-based administrators
  - Read-only/modify/administer users/groups
  - Fine-grained, design for customer multitenancy (future)
DriveScale Composer platform structure

- DriveScale GUI uses same RESTful APIs as provided for 3rd party integration (see backup)
- Uses open source components
  - Apache Zookeeper is the durable configuration store
  - MongoDB stores auditing (daily push) and monitor logs (hourly push)
- DriveScale Engine implements control logic
- Triple replicas
  - Separate fault domains outside disaggregated pools
  - Provides increased read bandwidth
- Configuration snapshots are pushed daily to DriveScale Cloud Central for disaster recovery
- Monitoring and alert logs are pushed hourly to DriveScale Cloud Central (predictive use)
DriveScale Start Up

**Installation**
- Install DriveScale Composer software on triple replica Linux hosts (can be VMs)
- Install DriveScale agents on servers and storage adapters (/etc/drivescale/conf)

**Discovery**
- DriveScale agents automatically inventory the disaggregated nodes
- Initialize the DriveScale Composer with inventory of available compute and storage

**Operation**
- User creates an *application (logical) cluster* from available servers and storage elements using the GUI or programatically
- User monitors, modifies, expands, destroys clusters
DriveScale platform structure

- The early choice of the ZooKeeper framework reinforced and influenced the product design
  - Fully enabled the “current to desired state” execution model
  - (needed) Alert mechanism came along for free
  - Data structures separated in schema to single writers (owners)/multiple readers
  - We think in terms of consistency points when updating configurations

- User level resource agents and Composer
  - The DriveScale Composer manager contains the DriveScale Engine, ZooKeeper and Mongo DB instances – all deployed as 3-way replicas
  - Server and storage adapter agents are symmetrical (90% of code is common)
  - Agents seek to achieve the desired state defined by the Composer in ZooKeeper

- The DriveScale Engine recovers incomplete server and storage adapter configuration updates after crash
  - End point configuration writes are replayed to ZooKeeper complete transactions before starting service
DriveScale Data Design

DriveScale Composer
- Primary reader of inventory
- Create, manage, modify logical clusters
- Initialize empty node configuration on installation for all element agents
- Add or delete managed elements as physical configuration changes
- Watches the server and storage adapter inventory to detect new or deleted hardware
- A ZooKeeper client

ZooKeeper store

Configuration (Composer domain)
- Persistent znodes
  - All logical clusters
  - All server node configs
  - All storage node configs
  - Network topology

Inventory (Agent domain)
- Persistent znodes of all elements
  - Server node inventory
  - Storage adapter inventory
    - Drive inventory
    - Network topology
- Ephemeral znodes
  - Node status

DriveScale Agent
- Automatically inventory compute or storage adapters
- Write device configuration information to the Inventory
- On detecting changes (mainly drive adds/removes) updates its Inventory
- “Watches” its configuration as defined by the Composer executes desired state action if modified
- Maintains ephemeral status znode (online vs. offline state)
- A ZooKeeper client

The operation of ZooKeeper is similar to a file system, and our ordering of writes allows for recovery from “crashes”.

Atomic write
read
Atomic write
read
DriveScale Inventory – agent write domain (all elements)

**Server Inventory**
- UUID
- network: addresses, speeds, LLDP
- supported transports (iSCSI, RoCE, NVMEoTCP)
- system memory
- # cores
- bogomips per core
- OS

**Storage adapter inventory**
- UUID
- network: addresses, speeds, LLDP
- supported transports

**Drive Inventory**
- WWN or NQN (persistent)
- size
- SSD or spinning RPM
- vendor/model/revision/serial
- ID of enclosing JBOD

**Adapter Status**
- status existence = online
- goal config version #
- latest achieved config version #
- status of goal config (accomplished, trying, retrying)
- errors trying goal config

**Server Status**
- same as adapter, plus
- is it safe to remove drives?

**Persistent znodes**

**Ephemeral znodes**
Ephemeral znodes deleted on heartbeat failure, indicating to the DriveScale Engine that the element has gone offline.

The above data is what we use in composition. Below-the-line data for both servers and adapters includes:
- HW vendor/model/serial
- installed version of our software
- complete SAS topology, etc.

```
/DriveScale/server/config/*
/DriveScale/dengine/config/*
```
States – create logical cluster example

Current state
- Available resource pool
- servers
- storage elements

Desired state
- New logical cluster
- servers
- storage elements
- template (configuration)

- Multiple distinct logical clusters co-exist in a DriveScale data center deployment
  - alongside global available (free) servers and storage elements
- **Drives**, not storage adapters, are composed with **servers**
- DriveScale Composer configurations move from current state to desired state
  - Only affected elements are acted on by respective agents, unused inventory unaffected
- Creating or modifying a logical cluster configuration is non-disruptive to unchanged elements
  - Elements are added or deleted to a logical cluster as defined by the desired state
- The DriveScale Composer defines the “what”, as a desired state, not the “how”, a set of commands. The DriveScale agents determine the steps to achieve the new configuration
DriveScale Logical Cluster etc. – Composer write domain

Logical cluster configuration
- Servers
  - Targets (storage elements)
  - Transports
- Drives
  - Addresses (network)
  - Drive template
- Template - policies
  - File system
  - Media type
  - Encryption
  - mount point, Transports
- Server/Drive association/linkage
- Version (config)

Server configuration
- File system
- Drives
- Encryption
- mount prefix, etc
- Version

Storage adapter configuration
- Drives
- Server endpoint
- (not visible in lcluster, but necessary to push config)
- Version

The Truth

new logical cluster configuration
Restore atomic write

DriveScale Engine

The Derived Truth
restartable recoverable

/DriveScale/lcluster/config/*
root@u39:/opt/drivescale/tools/zookeeper# ./zk_cat.py /DriveScale/lcluster/config/75119458-dfb0-4d65-a994-7a66c74531e8
{
  "annotationsForLogicalNode": {
    "raid-encrypt-xfs-4-00-0000": {
      "ds:raidId": "00qFrgzQI1s"
    }
  },
  "driveShredMethod": "",
  "enonce": "JAv6J5ZdCqW9usEApiinDU0z2zn-n2muCX9j2Yw106PYte4h4osrgzqAOKGLPd3ViWky3ohSFx8Huj-09MTq9Rqadjdu9Ek3dSP0JKESnILktvJU3wx5p-KOLjXPQ",
  "logicalNodeForDrive": {
    "iqn.2013-04.com.drivescale:wwn:0x5000c50058d47e87": "raid-encrypt-xfs-4-00-0000",
    "iqn.2013-04.com.drivescale:wwn:0x5000c50058d47ebb": "raid-encrypt-xfs-4-00-0000",
    "iqn.2013-04.com.drivescale:wwn:0x5000c50058d47ed": "raid-encrypt-xfs-4-00-0000",
    "iqn.2013-04.com.drivescale:wwn:0x5000c50058d47f3": "raid-encrypt-xfs-4-00-0000"
  },
  "logicalNodeForServer": {
    "4c4c4544-0039-3010-8030-c4c04f313437": "raid-encrypt-xfs-4-00-0000"
  },
  "name": "jgh-raid",
  "options": [],
  "templateForLogicalNode": {
    "raid-encrypt-xfs-4-00-0000": "c207f56c-042f-4723-8fa5-47c885ce5d86"
  },
  "templates": {
    "c207f56c-042f-4723-8fa5-47c885ce5d86":{
    "driveTagsExcluded": [],
    "driveTagsRequired": [],
    "drives": 4,
    "encrypt": true,
    "fstype": "xfs",
    "maxInstances": 100000,
    "mediaTransportsAllowed": [],
    "minDriveRpm": 0,
    "minDriveSize": 0,
    "minInstances": 0,
    "minServerBogoMips": 0,
    "minServerCpus": 0,
    "minServerMem": 0,
    "mountPrefix": "",
    "ratio": 1,
    "rotating": true,
    "serverTagsExcluded": [],
    "serverTagsRequired": [],
    "uuid": "c207f56c-042f-4723-8fa5-47c885ce5d86"
  }
  },
  "version": 51
}
root@u39:/opt/drivescale/tools/zookeeper#
DriveScale Composer in operation

- An administrator creates a *logical cluster* from available server and drive elements
  - Rule checked by the Composer and a configuration is created and atomically written to Composer’s Zookeeper key-value domain
- The configuration for each compute server and storage adapter is derived from the cluster configuration and a new element state is atomically written to each element Zookeeper instance
  - Small optimization, storage configuration pushed first as servers tend to aggressively retry – the storage must reach the new state for compute to bind
Achieving *desired state* – storage adapter agents

**Diagram:**

- **ZooKeeper store**
  - **Configuration (Composer domain)**
    - Persistent znodes
      - All logical clusters
      - All server node configs
      - All storage node configs
      - Network topology
  - **Inventory (Agent domain)**
    - Persistent znodes of *all elements*
      - Server node inventory
      - Storage adapter inventory
      - Drive inventory
      - Network topology
    - Ephemeral znodes
      - Node status

- Storage Adapter Agents
  - Agents attempt to achieve new desired state – the "how"
  - Retry forever

- All agents are independent and execute in parallel
DriveScale storage adapter move to desired state (storage adapter logic)

A. What targets am I offering?
   `currentlyOffering = set(activeTargets().get('targets', []))`

B. Is there anything I'm offering that isn't in my inventory?
   `good = devlinks.validTargetIqns(includeSlices=True)
   bad = currentlyOffering - good`

C. Is there anything I'm offering that isn't in my target configuration?
   `toOffer = set(newConfig.get('targets', []))
   withdraw = currentlyOffering - toOffer`

D. Stop offering everything from B and C:
   `withdraw.update(bad)
   failures = stopTargets(withdraw)`

E. Is there anything I'm not offering that is in my target configuration and inventory? Offer it.
   `toStart = set()
   for targetName, config in newConfig['targets'].iteritems():
     # SNIP QUARANTINE LOGIC
     if targetName not in good:
       noDev += 1
       failures.add(targetName)
       continue
     changed = updateTargetConfig(targetName, config, queueDepth)
     # None if failure, true if updated, false if no change
     if changed is None:
       failures.add(targetName)
     elif changed or targetName not in currentlyOffering:
       toStart.add(targetName)
   failures.update(startTargets(newConfig['targets'], toStart))`
## Performance – Aerospike ACT 4 benchmark

<table>
<thead>
<tr>
<th>Flash Device</th>
<th>Speed (tps)</th>
<th>&gt;1ms</th>
<th>&gt;8ms</th>
<th>&gt;64ms</th>
<th>Endurance</th>
<th>ACT</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>SmartIOPS DataEngine 3.2T *</td>
<td>555,000 *</td>
<td>1.8%</td>
<td>0.01%</td>
<td>0.00%</td>
<td>3 DWPD</td>
<td>3</td>
<td>Aerospace</td>
</tr>
<tr>
<td>*Micron 9200 Max 1.6 TB</td>
<td>316.500</td>
<td>4.64%</td>
<td>0.13%</td>
<td>0.00%</td>
<td>3 DWPD</td>
<td>3</td>
<td>Aerospace</td>
</tr>
<tr>
<td>HGST UltraStar SN200 RoCEv2</td>
<td>300,000</td>
<td>2.60%</td>
<td>0.06%</td>
<td>0.00%</td>
<td>3 DWPD</td>
<td>4</td>
<td>DriveScale</td>
</tr>
<tr>
<td>HGST UltraStar SN200 iSCSI</td>
<td>300,000</td>
<td>4.14%</td>
<td>0.06%</td>
<td>0.00%</td>
<td>3 DWPD</td>
<td>4</td>
<td>DriveScale</td>
</tr>
<tr>
<td>Intel P4610 1.6 TB</td>
<td>300,000</td>
<td>4.56%</td>
<td>0.85%</td>
<td>0.00%</td>
<td>3 DWPD</td>
<td>4</td>
<td>Aerospace</td>
</tr>
<tr>
<td>Intel P4610 6.4 TB</td>
<td>300,000</td>
<td>4.98%</td>
<td>0.14%</td>
<td>0.00%</td>
<td>3 DWPD</td>
<td>4</td>
<td>Aerospace</td>
</tr>
<tr>
<td>ScaleFlux CSS1000 3.2T</td>
<td>300,000</td>
<td>4.59%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>5 DWPD</td>
<td>4</td>
<td>ScaleFlux</td>
</tr>
<tr>
<td>Micron 9200 Pro 3.84 TB</td>
<td>283,500</td>
<td>4.54%</td>
<td>0.13%</td>
<td>0.00%</td>
<td>1 DWPD</td>
<td>3</td>
<td>Micron</td>
</tr>
<tr>
<td>Huawei ES3600P V3 1.6T</td>
<td>153,000</td>
<td>4.47%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>3 DWPD</td>
<td>4</td>
<td>Aerospace</td>
</tr>
<tr>
<td>Toshiba PX04PMB320 3.2T</td>
<td>135,500</td>
<td>4.73%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>10 DWPD</td>
<td>3</td>
<td>Toshiba</td>
</tr>
<tr>
<td>Intel P4510 1 TB</td>
<td>120,000</td>
<td>4.31%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>1 DWPD</td>
<td>4</td>
<td>Aerosipe</td>
</tr>
<tr>
<td>Samsung PM983 1.92 TB</td>
<td>108,000</td>
<td>0.09%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>1.3 DWPD</td>
<td>3.1</td>
<td>Aerosipe</td>
</tr>
</tbody>
</table>
Wrap
Future directions

- We understand the network topology, and constrain cluster to avoid obvious configurations that will violate application replica fault separation – really want to do more here
- Additional capabilities on predictive analytics
  - Exploit long term monitoring to provide trending and failure predictions
  - End-to-end data monitoring provide rich data source
- ZooKeeper is not really a database – may restructure non-configuration information out of its data store for more powerful query capabilities
- Composing disaggregated GPUs
- Container framework integration – building on current Kubernetes plug-in
- Application centric analytics
Recapitulation

- Manage and automate complexity of composable infrastructure at scale
  - Including fault domains for rack failure semantic support
  - Distributed key distribution for authentication and date encryption over-the-wire and at rest
- Laser focus on scalability, durability, correctness goals
- Provide a simple, intuitive user experience
Questions
Backup
References

- ZooKeeper watches
  https://zookeeper.apache.org/doc/r3.1.2/zookeeperProgrammers.html#ch_zkWatches
- ZooKeeper design
  https://zookeeper.apache.org/doc/r3.1.2/zookeeperProgrammers.html
- ZooKeeper overview
  https://zookeeper.apache.org
- DriveScale Drive Encryption Overview -
- ZooKeeper use at Twitter
- Aerospike benchmark information
  https://www.aerospike.com/docs/operations/plan/ssd/ssd_certification.html
Determining the available resource pool

ZooKeeper store
- Configuration (Composer domain)
  - Persistent znodes
    - All server node configs
    - All storage node configs
    - Network topology
  - Ephemeral znodes
- Resource inventory (Agent domain)
  - Persistent znodes of *all elements*
    - Server inventory
    - Storage adapter inventory
    - Drive inventory
    - Network topology
  - Ephemeral znodes
    - Node status

read all configured resources

all inventory – configured resources = available pool

read all online inventory
“Carol Danvers (aka Captain Marvel) is very durable, able to survive attacks from the likes of the Hulk and has actually been able to lend her hand in physically fighting Thanos without getting one shotted.”
ZooKeeper description
“ZooKeeper is used at Twitter as the source of truth for storing critical metadata. It serves as a coordination kernel to provide distributed coordination services, such as leader election and distributed locking.”
Apache ZooKeeper: a durable data repository

- A widely used service for maintaining configuration information, naming, providing synchronization in a distributed system
  - Data is stored in a hierarchical key-value store (similar to a file system)
  - Multiple replicas provide availability and increased aggregate read performance
- ZooKeeper implements a leader/follower model
  - A single leader is elected on start up, the remaining nodes becomes followers
  - Only the leader can receive writes (guaranteeing atomicity)
  - Writes are propagated to followers before being acknowledged
  - All ZooKeeper instances can serve reads (design is suited for read mostly deployment)
  - Failure of a leader results in election of a new leader from the remaining followers (quorum on partition defines election group)
- The ZooKeeper data repository resides in memory during operation
  - Fits easily in 128GB server or VM
  - Initialized from persistent storage (SSD) on start up
  - Compressed data znodes (includes directories) are limited to maximum 1MB
Technology trends
Technology Trends – towards composable infrastructure

- Ubiquitous high performance, low latency fabric (Ethernet)
- Massively parallel Big Data and NoSQL etc. applications
  - Hadoop and other apps
- Affordable Solid State Storage enabling web scale real time applications
  - And increases in reliability of both HDD and SSD
- NVME everywhere

DriveScale’s design exploits ubiquitous low-latency high-speed Ethernet at scale
DriveScale Composable Infrastructure
API Summary
REST API structure pt 1

DriveScale API

This provides a RESTful interface (augmented with a WebSocket-based event mechanism) to read and write the state of the (logical and physical) objects that exist within a DriveScale Management Domain (MSD) and the relationships that exist among these objects.

This API also provides a set of endpoints to allow the manipulation of (user-defined) LogicalCluster and LogicalFilesystem Templates, to control the process of collecting and distributing system meta-data (via the "logcon" abstraction), and to allow (authenticated) user sessions to be initiated and ended.

Alerts

Get alert indications

GET /alert Get a list of alerts in the system

GET /alert/classes Get a list of alert classes in the system

GET /alert/{alertId} Retrieve a specific alert given its identifier

PUT /alert/{alertId}/resolve Manually mark a specific alert as resolved

PUT /alert/{alertId}/ignore Ignore an alert, it will still be present in the history.

Audit

Get audit log

GET /audit Get a list of audit log in the system

GET /audit/users Get a list of users appearing in the audit log

GET /audit/{auditId} Retrieve a specific audit event given its identifier

Common

Get information about the main kinds of objects supported by the API

GET /object/kinds Get list of kinds of Objects

GET /object/{objectKind}/types Get list of types of an Object

GET /object/{objectKind}/objectType/properties Get the names of the properties of an Object type

GET /object/{objectKind}/objectType/properties/schema Get the schema of the properties of an Object type

GET /object/{objectKind}/objectType/relationships Get allowable Relationships for a type of Object
REST API structure pt 2

Composer
- Create, modify, get information about, and manage LogicalClusters
  - **POST** /composer/proposals/new: Create LogicalCluster proposal
  - **POST** /composer/proposals/expansion: Create proposal to expand a LogicalCluster
  - **POST** /composer/proposals/reanimation: Create proposal to resume a LogicalCluster
  - **POST** /composer/proposals/replace: Create proposal to replace certain entities in a LogicalCluster
  - **POST** /composer/proposals/resources: Create proposal of drives and/or servers in Bandwidth Domain(s)
  - **POST** /composer/proposals/drives: Find drives for a given set of servers
  - **POST** /composer/transfer: Transfer drives and/or servers between LogicalClusters

Relationships
- Get information on the relationships among Entities
  - **GET** /relationship/{relationshipType}: Get list of properties for current Relationships of the given type
  - **GET** /relationship/{relationshipType}/{srcEntityId}/{srcEntityType}/{tarEntityId}: Get properties of a Relationship

Templates
- Create, modify, and delete LogicalCluster and LogicalNode templates
  - **GET** /template/{templateType}: Get list of properties for current Templates of the given type
  - **POST** /template/{templateType}: Create new Template
  - **DELETE** /template/{templateType}/{templateId}: Delete a Template
  - **GET** /template/{templateType}/{templateId}: Get values of a Template's properties
  - **PATCH** /template/{templateType}/{templateId}: Write properties of a Template
  - **PUT** /template/{templateType}/{templateId}: Overwrite Template
REST API structure pt 3

<table>
<thead>
<tr>
<th>Method</th>
<th>URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td><code>/entity/{entityType}</code></td>
<td>Get list of all instances of a given type of Entity</td>
</tr>
<tr>
<td>POST</td>
<td><code>/entity/{entityType}</code></td>
<td>Create a new Entity</td>
</tr>
<tr>
<td>DELETE</td>
<td><code>/entity/{entityType}/{entityId}</code></td>
<td>Delete an Entity</td>
</tr>
<tr>
<td>GET</td>
<td><code>/entity/{entityType}/{entityId}</code></td>
<td>Get property values for a specific Entity</td>
</tr>
<tr>
<td>PATCH</td>
<td><code>/entity/{entityType}/{entityId}</code></td>
<td>Update 'override' properties of an Entity</td>
</tr>
<tr>
<td>PUT</td>
<td><code>/entity/{entityType}/{entityId}</code></td>
<td>Write 'override' properties of an Entity</td>
</tr>
<tr>
<td>PATCH</td>
<td><code>/entity/{entityType}/{entityId}/annotations</code></td>
<td>Update 'annotations' values of an Entity</td>
</tr>
<tr>
<td>PUT</td>
<td><code>/entity/{entityType}/{entityId}/annotations</code></td>
<td>Write 'annotation' values of an Entity</td>
</tr>
<tr>
<td>GET</td>
<td><code>/entity/{entityType}/{entityId}/annotations</code></td>
<td>Get the 'annotation' values of an Entity</td>
</tr>
<tr>
<td>PATCH</td>
<td><code>/entity/{entityType}/{entityId}/defaults</code></td>
<td>Update 'default' properties of an Entity</td>
</tr>
<tr>
<td>PUT</td>
<td><code>/entity/{entityType}/{entityId}/defaults</code></td>
<td>Write 'default' properties of an Entity</td>
</tr>
<tr>
<td>GET</td>
<td><code>/entity/{entityType}/{entityId}/defaults</code></td>
<td>Get the 'default' properties of an Entity</td>
</tr>
</tbody>
</table>

DriveScale

---

GET /entity/{entityType}/{entityId}/graph - Get graph of an Entity's relatives

POST /entity/{entityType}/{entityId}/unlock - Unlock ManagementDomain access for an entity

GET /entity/LogicalCluster/{logicalClusterId}/config - Get the configuration properties of a LogicalCluster

PATCH /entity/LogicalCluster/{logicalClusterId}/config - Update configuration properties of a LogicalCluster

PUT /entity/LogicalCluster/{logicalClusterId}/config - Configure a LogicalCluster

POST /entity/LogicalCluster/{logicalClusterId}/reconnect - Reconnect a LogicalCluster

POST /entity/LogicalCluster/{logicalClusterId}/reconnectAll - Reconnect a LogicalCluster completely

POST /entity/Drive/{entityId}/enableCarving - Enable a free SSD Drive to be carved into Drive-Slices

POST /entity/Drive/{entityId}/disableCarving - Stop an SSD Drive from being carved into Drive-Slices

POST /entity/Drive/{entityId}/shred - Request shredding of a free drive

POST /entity/Drive/{entityId}/abortShredding - Abort shredding of a drive
REST API structure pt 4
REST API structure pt 5

**Miscellaneous support functions**

- **GET /apiVersion** Get current version of the API
- **GET /currentDms** Get the current master DMS node
- **GET /currentUser** Get the current logged in user
- **GET /currentUser/enroll2FA** Get the setup values to enroll in 2 factor authentication
- **POST /currentUser/enroll2FA** Confirm enrolling of a user with 2FA
- **DELETE /currentUser/enroll2FA** Disable selected 2FA
- **GET /currentUser/check2FA** Get challenge 2FA data if needed
- **POST /currentUser/check2FA** Do 2FA authentication
- **GET /overrides/SAS** Get current SAS override information.
- **PUT /overrides/SAS** Replace current SAS override information.
- **GET /quarantine/Drive** Get current list of quarantined drives.
- **PATCH /quarantine/Drive** Modify the list of quarantined drives.

**Additional endpoints**

- **POST /updatePassword** Update the user's password
- **POST /updateUser** Update some of the properties of the logged-in user
- **POST /checkACL** Check if the user has access to the action/path
- **GET /dmsConfig** Get the DMS node's configuration information
DriveScale Composer Platform (control and data plane)

- DriveScale Composer Platform (control and data plane)
- 3rd Party Orchestration
- GUI
- DriveScale Cloud Central
- Configuration recovery snapshots, alert logs, software updates
- API
- DriveScale Composer Platform (triple replica)
- Server Agents
- Compute pool
- Network switches
- HDD, SSD pools
- Storage Adapter Agents/Software
- User level server and storage agents share 90% of their code
- Data Path + LLDP

DriveScale