

Managing AEM Datastore

Amit Jain | Senior Computer Scientist

1 | Introduction

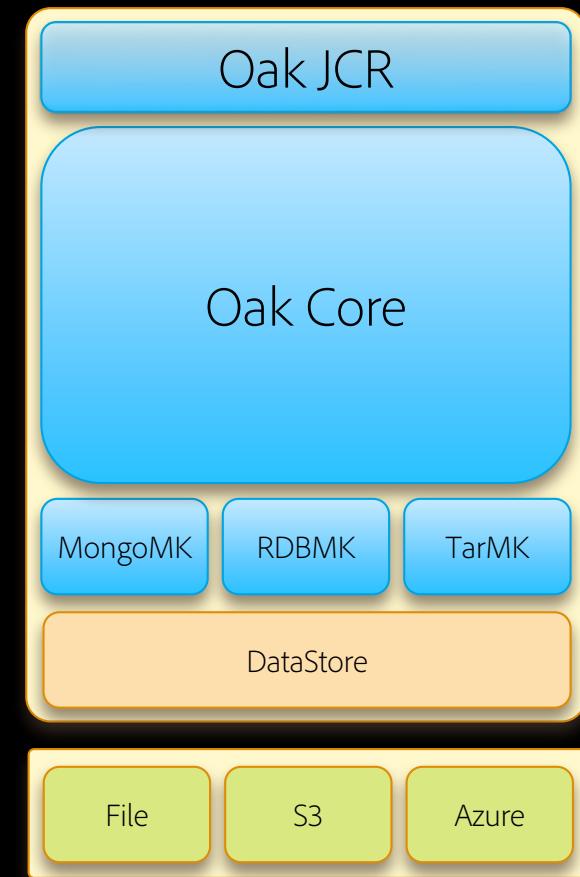
2 | Configuration & Deployment

3 | Garbage Collection

4 | Tooling & Troubleshooting

DataStore

- Apache Jackrabbit/Oak storage abstraction for blobs/binaries



DataStore

- Jackrabbit concept for storage of binaries
- Support for 3 backends
 - File system (FileDataStore)
 - S3 (S3DataStore)
 - Azure (AzureDataStore)

What's stored in the DataStore

- Files and JCR binary properties with size greater than
 - SegmentNodeStore – maximum of
 - *16384 bytes*
 - Below that in-lined in the Segment store
 - *minRecordLength* configured
 - Below that in-lined in the blob ID
 - DocumentNodeStore
 - *minRecordLength* configured
 - Below that in-lined in the blob ID

How content stored in the DataStore

- The content stored (i.e. Blobs) are immutable and de-duplicated
 - SHA-256 digest generated from the content used as identifier (ID)
 - https://en.wikipedia.org/wiki/Content-addressable_storage
- DataStore supports the following operations
 - Create
 - Update
 - Delete
- Blob timestamp updated when re-uploaded

What's stored in the NodeStore

- The unique blob identifier (ID)
- ID suffixed with length of the blob with a '#' delimiter
 - 0bda75655493e6448cdc79a520008b910edd27d0d5319a35f34707d71a2b050f#81920
- Acts as a reference to the underlying blob
- Since de-duplicated the same blob and blob ID can be referenced from multiple node properties

Creating a binary property

- Node node = session.getRootNode().getNode("/content/a");
Node fileNode = node.addNode("filea", "nt:file");
Node resNode = fileNode.addNode("jcr:content", "nt:resource");
resNode.setProperty("jcr:mimeType", "application/pdf");

```
File file = new File("data.pdf");
FileInputStream stream = new FileInputStream(file);
Binary binary = session.getValueFactory().createBinary(stream);
resNode.setProperty("jcr:data", binary);
session.save();
```

Reading a binary property

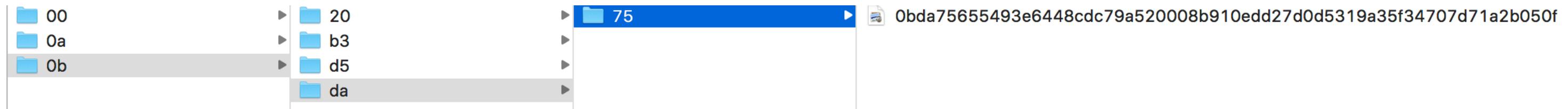
```
Node node = session.getRootNode().getNode("/content/a/filea/jcr:content");
Binary binary = node.getProperty("jcr:data").getBinary();
InputStream stream = binary.getStream();
```

Creating a binary property

The screenshot shows the AEM authoring interface. On the left is a tree view of the site structure, with the path 'a / filea / jcr:content' selected. On the right is a properties editor for a node under 'jcr:content'. The 'Properties' tab is active, displaying the following table:

	Name	Type	Value
1	jcr:data	Binary	view
2	jcr:encoding	String	utf-8
3	jcr:lastModified	Date	2017-10-03T14:58:05.715+05:30
4	jcr:lastModifiedBy	String	Administrator
5	jcr:mimeType	String	application/pdf
6	jcr:primaryType	Name	nt:resource

DataStore – Creating a binary property example



FileDataStore

- 3 level directory structure created under the path covered with each directory from the 2 letter prefix sequentially
 - Blob with ID `0bda75655493e6448cdc79a520008b910edd27d0d5319a35f34707d71a2b050f` stored as
 - `./repository/datastore/0b/da/75/0bda75655493e6448cdc79a520008b910edd27d0d5319a35f34707d71a2b050f`
- `minRecordLength` by default 100 bytes but recommended to be 4096 bytes
- FileDataStore is default blob storage for SegmentNodeStore with AEM 6.3
 - Segment store files memory-mapped and storing blobs inefficient
 - Better memory utilization with only nodes and non-binary properties
 - Less thrashing and better performance for large number of nodes
- If the FDS is stored on NAS and performance slow then can evaluate a local caching wrapper available with
 - `CachingFileDataStore`
 - Prior to AEM 6.3 – `CachingFDS` which uses Jackrabbit caching wrapper

S3/AzureDataStore

- Blobs stored in the container configured
- *minRecordLength* 16 KB by default
- Both DataStores extend the local caching wrapper
 - *AbstractSharedCachingDataStore*
 - Prior to AEM 6.3 – *CachingDataStore* which uses Jackrabbit caching wrapper

Local Caching wrapper for DataStore

- Local file system caching for blobs
- Asynchronous uploads
 - Locally staged and then asynchronously uploaded to S3/Azure/NFS
 - Should be used with caution in the following scenario as asynchronous nature would have latency in uploads to the DataStore and “Blob not found” transiently thrown:
 - In clustered environments
 - On publish systems with binary-less replication
- Some changes/improvements last release
 - Separate local directories for uploads/downloads which enable robust cache eviction
 - Enabled detailed statistics to help in optimization

BlobStore (Introduced with Oak)

- Chunking of binaries into 2 MB block
- Supported backends
 - FileSystem - FileBlobStore
 - Mongo – MongoBlobStore
 - RDB – RDBBlobStore
- Current recommendation is to use DataStore.
 - Performance a concern with native Mongo/RDBBlobStore
 - Legacy reasons for upgraded systems (CQ 5.6)

1 | Introduction

2 | Configuration & Deployment

3 | Garbage Collection

4 | Tooling & Troubleshooting

Topologies

- Standalone/Clustered setups
 - A dedicated DataStore for each instance/cluster
 - Separate FDS or Separate S3/Azure container
 - Separate or same for Cold standby store for SegmentNodeStore
- Shared
 - A little bit of a misnomer
 - Signifies DataStore shared among different/disparate repositories
 - E.g. An author and publish sharing the same DataStore
 - Shared FDS or Shared S3/Azure container

Configuration

Configuring a DataStore for AEM/OSGi setups

- SegmentNodeStoreService/DocumentNodeStoreService configuration
 - org.apache.jackrabbit.oak.segment.SegmentNodeStoreService
 - org.apache.jackrabbit.oak.plugins.document.DocumentNodeStoreService
 - *customBlobStore = true*
 - *blobTrackSnapshotIntervalInSecs*
 - *blobGcMaxAgeInSecs*
- DataStore
 - *FileDataStore (FDS)* - org.apache.jackrabbit.oak.plugins.datastore.FileDataStore.config
 - *S3DataStore (S3)* - org.apache.jackrabbit.oak.plugins.datastore.S3DataStore.config
 - *AzureDataStore (Azure)* – org.apache.jackrabbit.oak.plugins.datastore.AzureDataStore.config

AEM Runmodes

In conjunction with the configuration files

- SegmentNodeStore
 - FDS - *crx3tar* (AEM default)
 - S3/Azure – *crx3tar-nofds*
- DocumentNodeStore
 - *crx3mongo/crx3rdb*

Configuration – FDS Configuration parameters

- *path*: The path of the data store. The default is <AEM install folder>/repository/datastore
- *minRecordLength*: The minimum size of an object that should be stored in the data store. The default is 100 bytes.

Configuration – S3 Configuration parameters

- *accessKey*: The AWS access key
- *secretKey*: The AWS secret access key
- *s3Bucket*: The bucket name
- *s3Region*: The bucket region
- *minRecordLength*: The minimum size of an object that should be stored in the data store. The default is 16KB
- *secret*: Shared secret between instances sharing DataStore for binary-less replication

Configuration – Azure Configuration parameters

- *accessKey*: The AWS access key
- *secretKey*: The AWS secret access key
- *container*: The container name
- *azureSas*: Shared access signature token
- *azureBlobEndpoint*: Azure blob endpoint
- *minRecordLength*: The minimum size of an object that should be stored in the data store. The default is 16KB
- *secret*: Shared secret between instances sharing DataStore for binary-less replication

Configuration – DataStore Caching Wrapper

- *stagingSplitPercentage* - Percentage of cache size configured to be used for staging asynchronous uploads. The default value is **10%**
 - Set *stagingSplitPercentage* = 0 for disabling asynchronous uploads
- *uploadThreads* - The number of uploads threads that are used for asynchronous uploads. The default value is **10**
- *stagingPurgeInterval* - Interval in seconds for purging finished uploads from the staging cache. The default value is **300** seconds
- *stagingRetryInterval* - The retry interval in seconds for failed uploads. The default value is **600** seconds
- *cacheSize*: Size of the cache. The value is specified in bytes. The default is **64GB**
- *path*: The path of the local cache. The default is <AEM install folder>/repository/datastore

Initialization

Along with the DataStore registration

- Repository ID registration
- JMX Mbeans registration
- BlobTracker registration

Initialization – Repository ID

To enable identification of different repositories sharing the DataStore a unique repository ID is registered per repository

- A marker file for each repository registered in the DataStore at
 - Root of the path for the FileDataStore e.g. ./repository/datastore/
 - META folder in the configured bucket for S3DataStore
- Saved as a hidden property for persistence- :clusterConfig/:clusterId
- Need to be reset if instances cloned
 - Use oak-run utility's *resetclusterid* command

Initialization – Mbeans

- *BlobStoreStats* - Useful statistics for monitoring performance of the DataStore
 - Upload/Download rate
 - Upload/Download size
 - Upload/Download count
- *BlobGarbageCollection* – Mbean enabling DataStore GC and consistency check

Initialization - DataStoreCacheStats

- Hit count/ratio download cache
- Hit count/ratio upload cache
- Miss count/ratio download cache
- Miss count/ratio upload cache

Parameters	DownloadCache	StagingCache
<i>elementCount</i>	Number of files cached	Pending file uploads in cache
<i>requestCount</i>	Number of files requested from cache	Number of file uploads requested
<i>hitCount</i>	Number of files served from cache	Number of files uploaded asynchronously
<i>hitRate</i>	Ratio of hits to requests	Ratio of hits to requests
<i>loadCount</i>	Number of files loaded when not in cache	Number of file requests from cache
<i>loadSuccessCount</i>	Number of files successfully loaded	Number of file requests served from cache
<i>loadExceptionCount</i>	Number of load file unsuccessful	Number of file requests not in cache
<i>maxWeight</i>	Max cache size (bytes)	Max cache size (bytes)
<i>totalWeight</i>	Current size of cache (bytes)	Current size of cache (bytes)
<i>totalMemWeight</i>	Approximate size of cache in-memory (bytes)	Approximate size of cache in memory (bytes)

S3DataStore – S3 connector versions

- AEM 6.1
 - Connector version `com.adobe.granite.oaks3connector 1.2.x`
 - Oak 1.2.12
- AEM 6.2
 - Connector version `com.adobe.granite.oaks3connector 1.4.x`
 - Oak 1.4.x
- AEM 6.3
 - Connector version `com.adobe.granite.oaks3connector 1.6.x`
 - Oak 1.6.x

Connector versions not be confused with Oak versions

- Follow separate release cycles but signifies the branch/minor version of Oak embedded

Timeline - Enhancements/Changes

- AEM 6.0
 - minRecordLength - S3/FDS – 100 bytes
 - Digest algorithm - SHA-1
- AEM 6.1
 - SharedDataStore – Yes (Special configuration with Oak 1.2.12)
 - minRecordLength – S3/FDS – 100 bytes
 - Digest algorithm - SHA-1
- AEM 6.2
 - SharedDataStore – Yes
 - minRecordLength – S3/FDS – 100 bytes
 - Digest algorithm - SHA-1

Timeline - Enhancements/Changes

- AEM 6.3
 - minRecordLength
 - S3/Azure – 16 KB
 - FDS – 100 bytes
 - Digest Algorithm – SHA-256 (Oak 1.6.5)
 - BlobTracker
 - New caching wrapper
 - AzureDataStore
- AEM 6.4
 - minRecordLength
 - S3/Azure – 16 KB
 - FDS – 100 bytes
 - Digest algorithm - SHA-256
 - Active deletion lucene blobs

1 | Introduction

2 | Configuration & Deployment

3 | Garbage Collection

4 | Tooling & Troubleshooting

Why Garbage Collection?

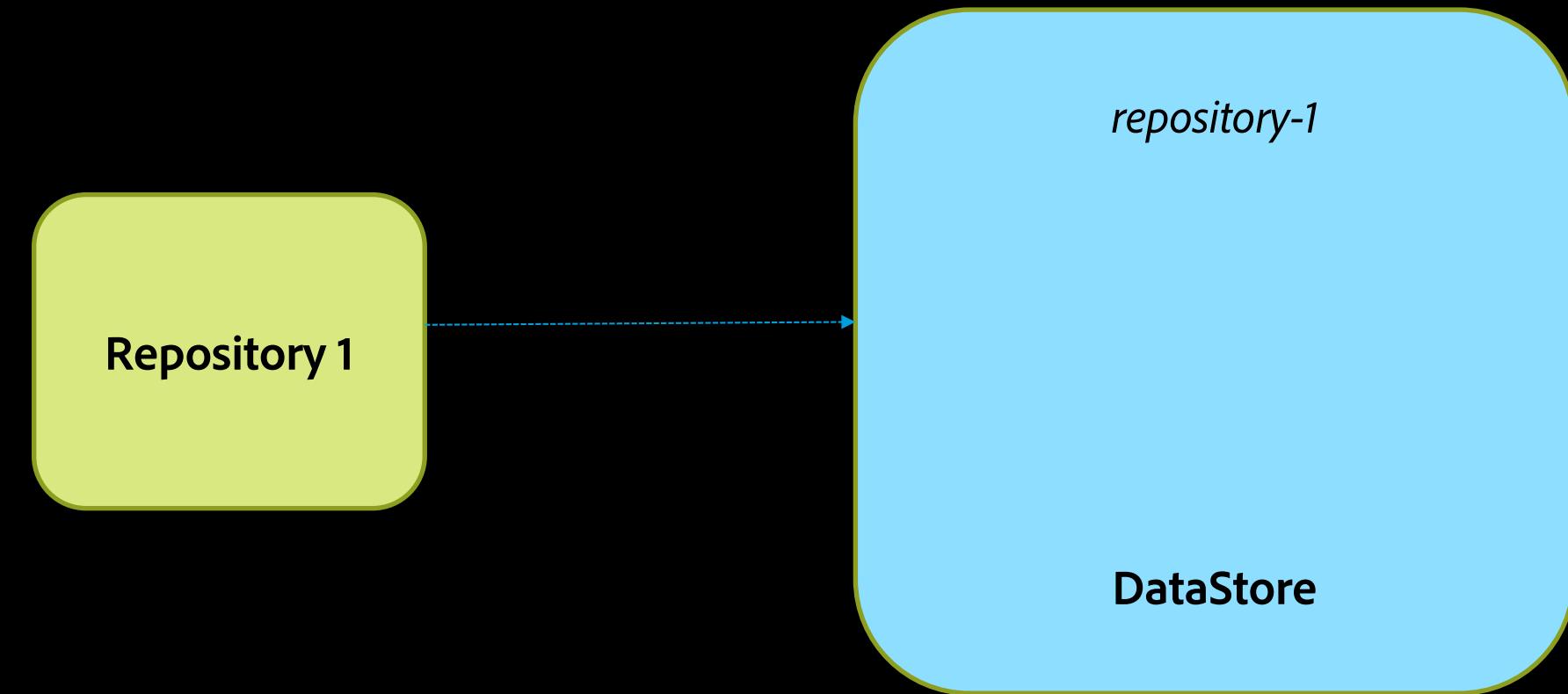
- All references to a particular blob not known when deleting a particular node which contains that blob
 - Tracking references complicated and some sort of a reverse index from blob ids -> node ids needed
 - Another persistent and cluster aware data structure
 - Would adversely affect performance of crucial CUD (Create, Update, Delete) operations

Pre-requisites

- Compaction/cleanup mandatory before GC to remove stale references
- Each repository registers itself in the DataStore
 - Creates a file *repository-xxx-xxxxx-xxxxxx* with a unique identifier

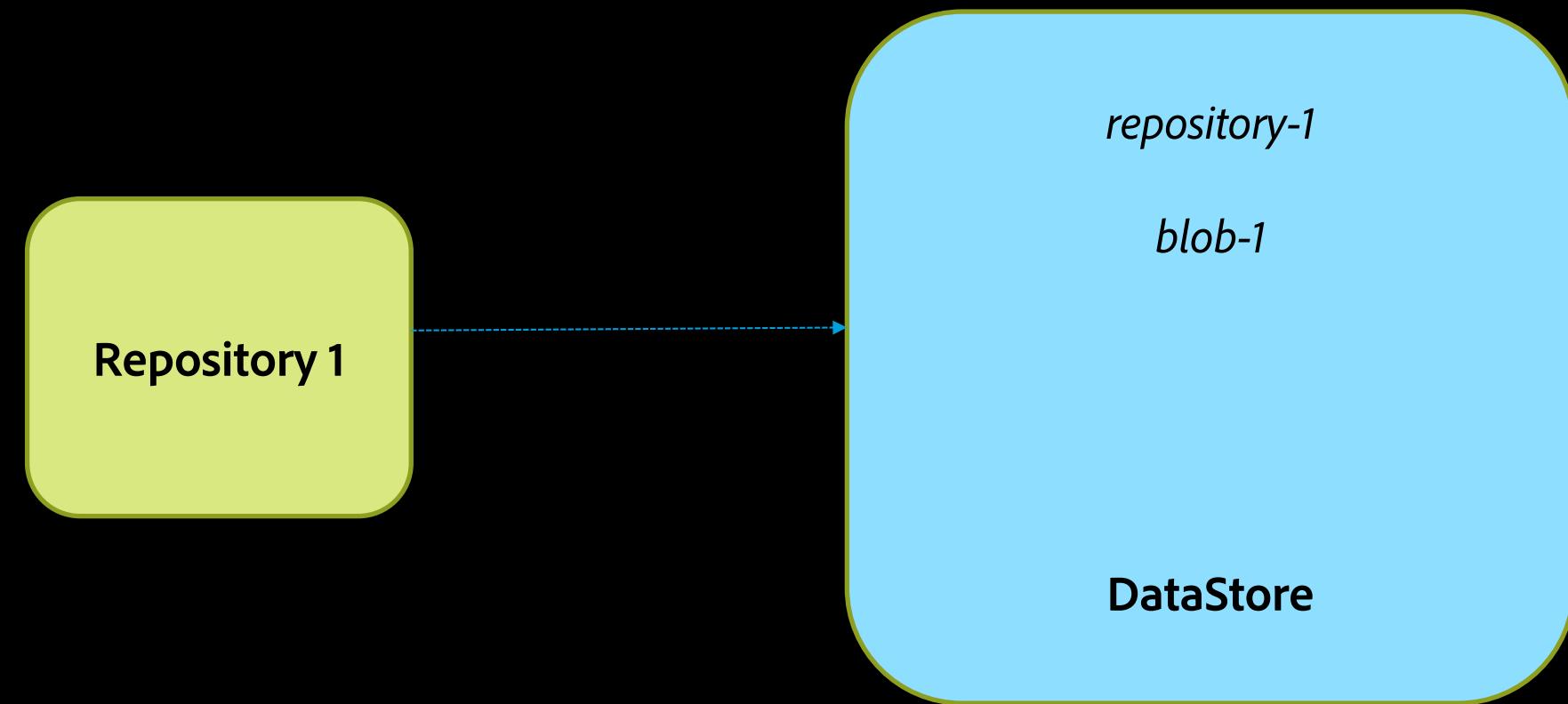
DataStore GC

- On Initialization repository connects to the DataStore and registers



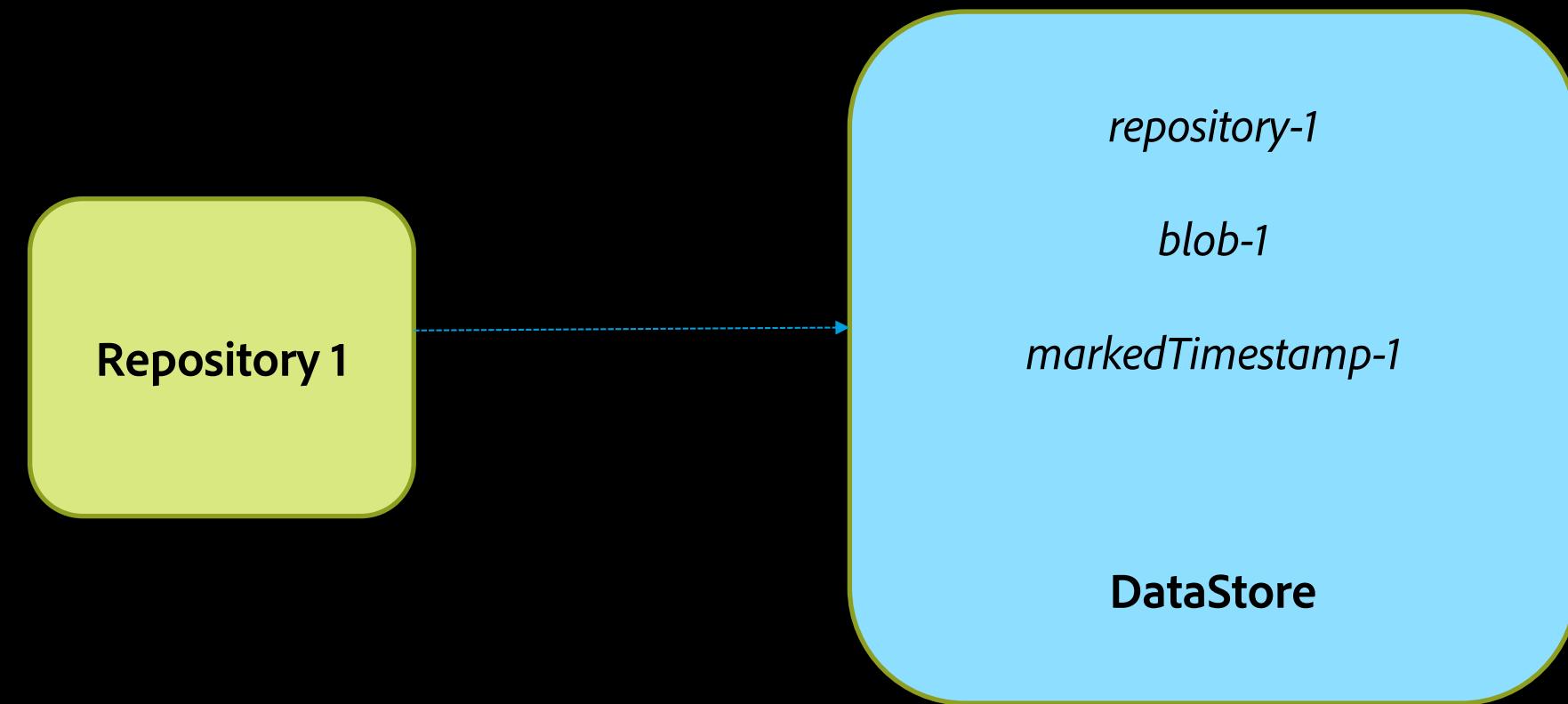
DataStore GC

- Repository uploads cached blob ids at a later time



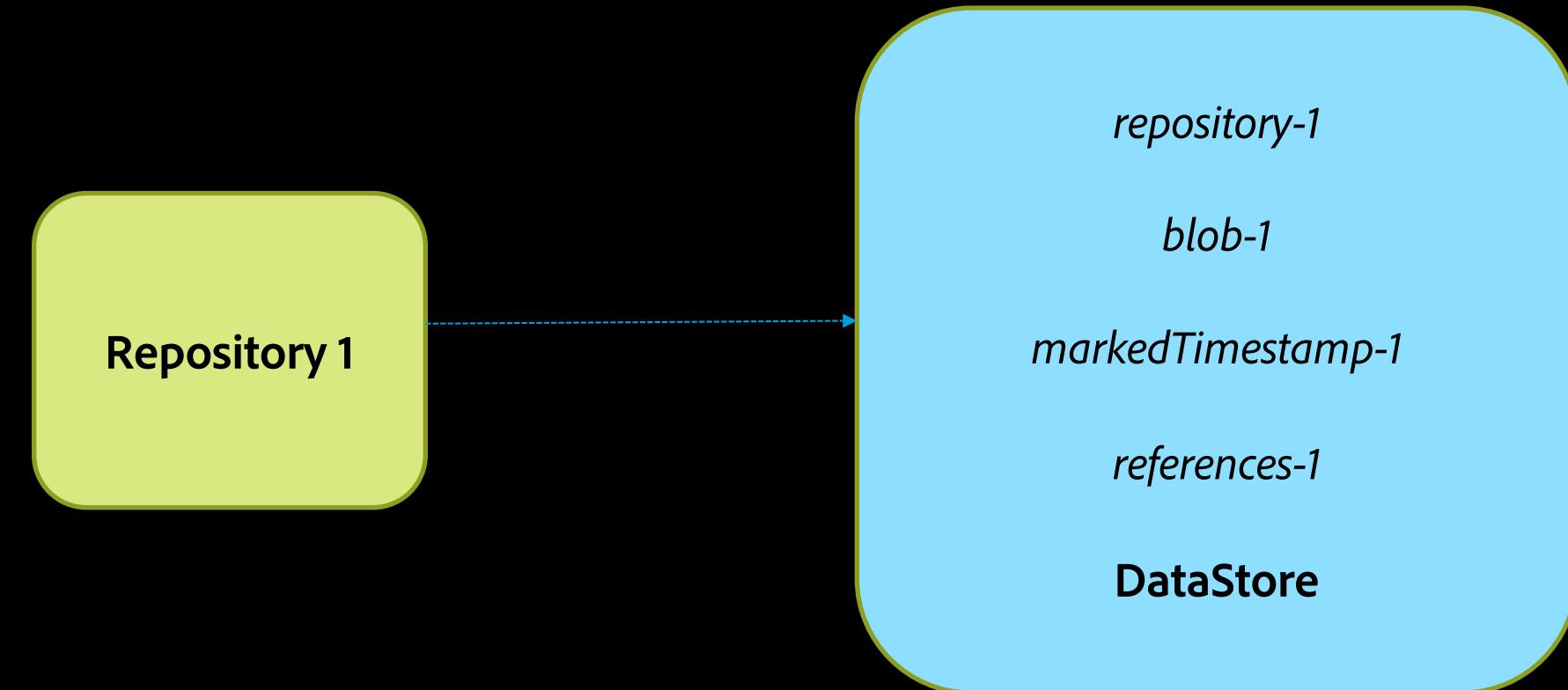
DataStore GC – Mark Phase

- Repository starts the *Mark* phase and adds a starting marker to the DataStore



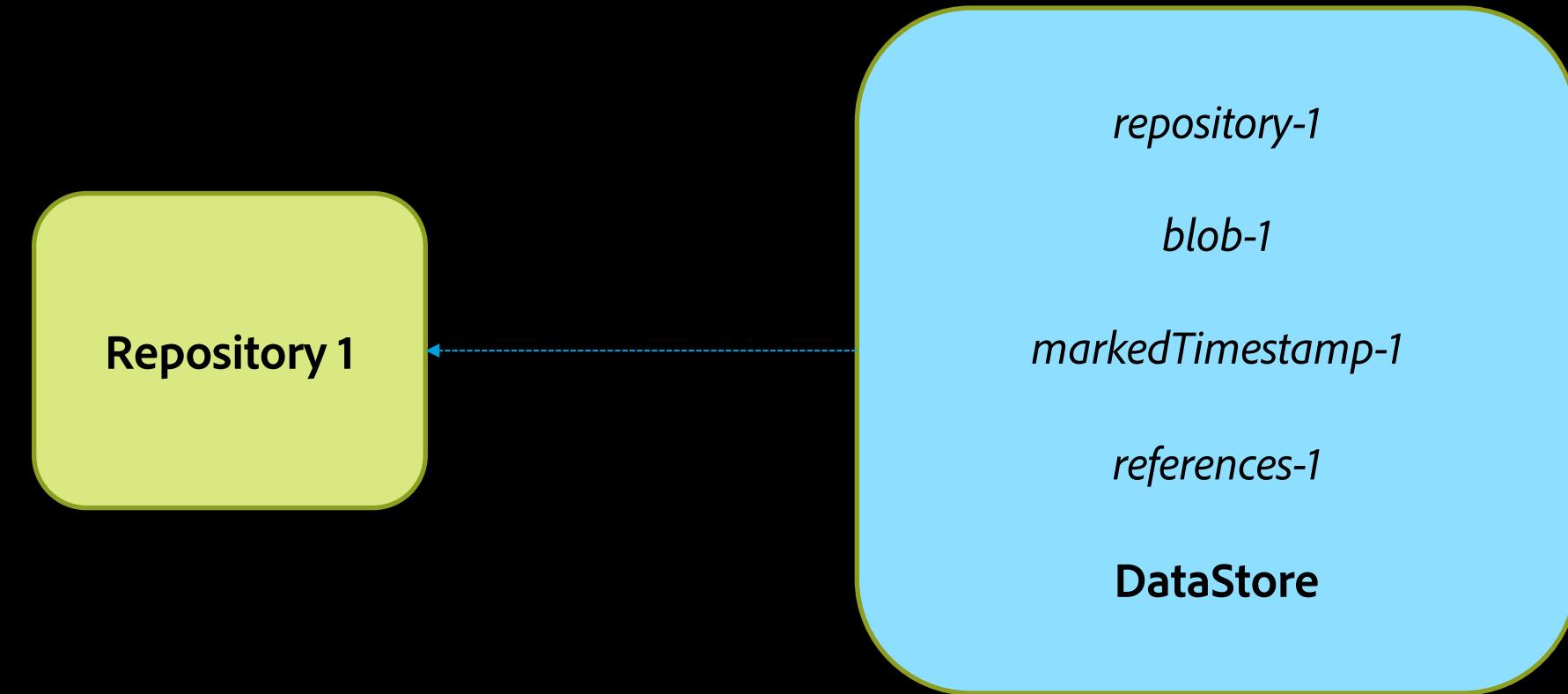
DataStore GC

- Repository add references on Mark phase completion



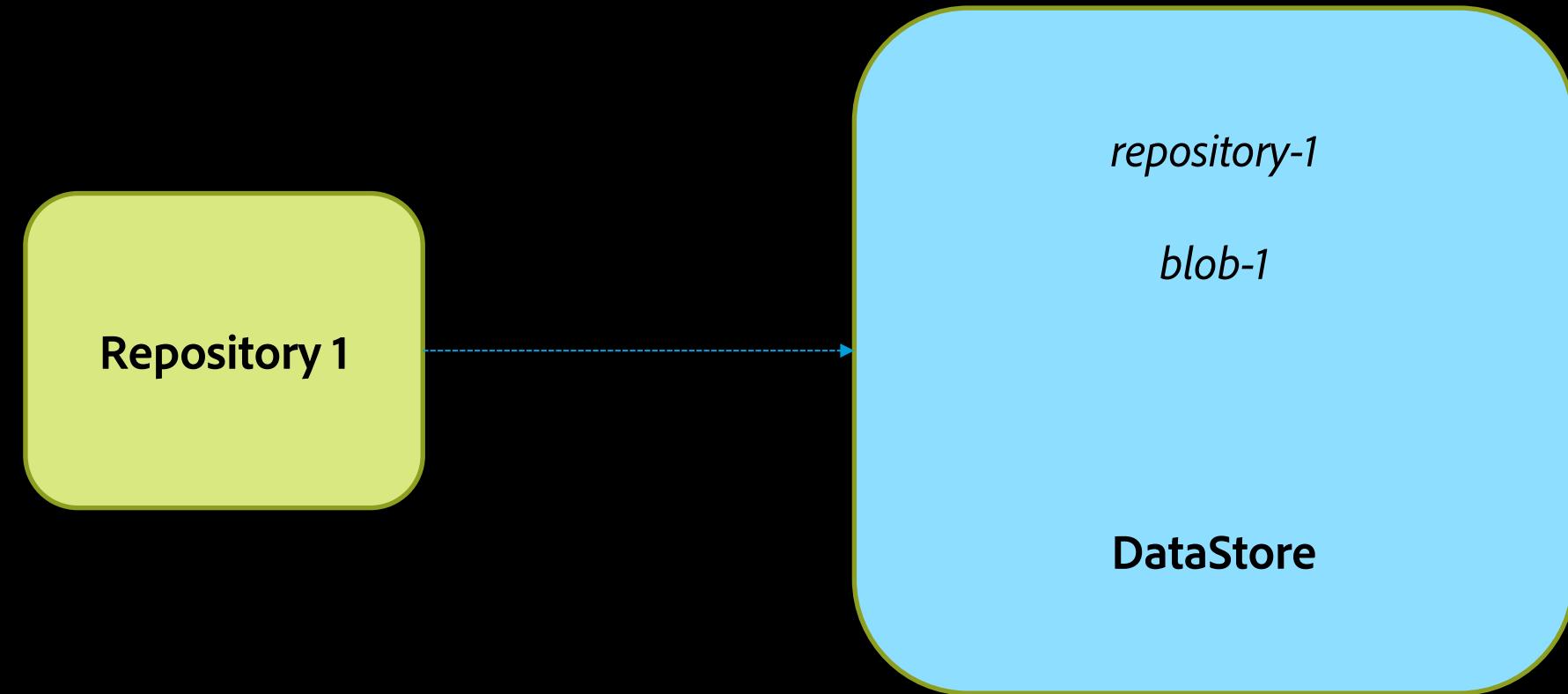
DataStore GC Sweep

- Blob ids collected from the DataStore



DataStore GC Sweep

- Use marked time for ascertaining age to delete blobs and clean up state to finish



Mechanism

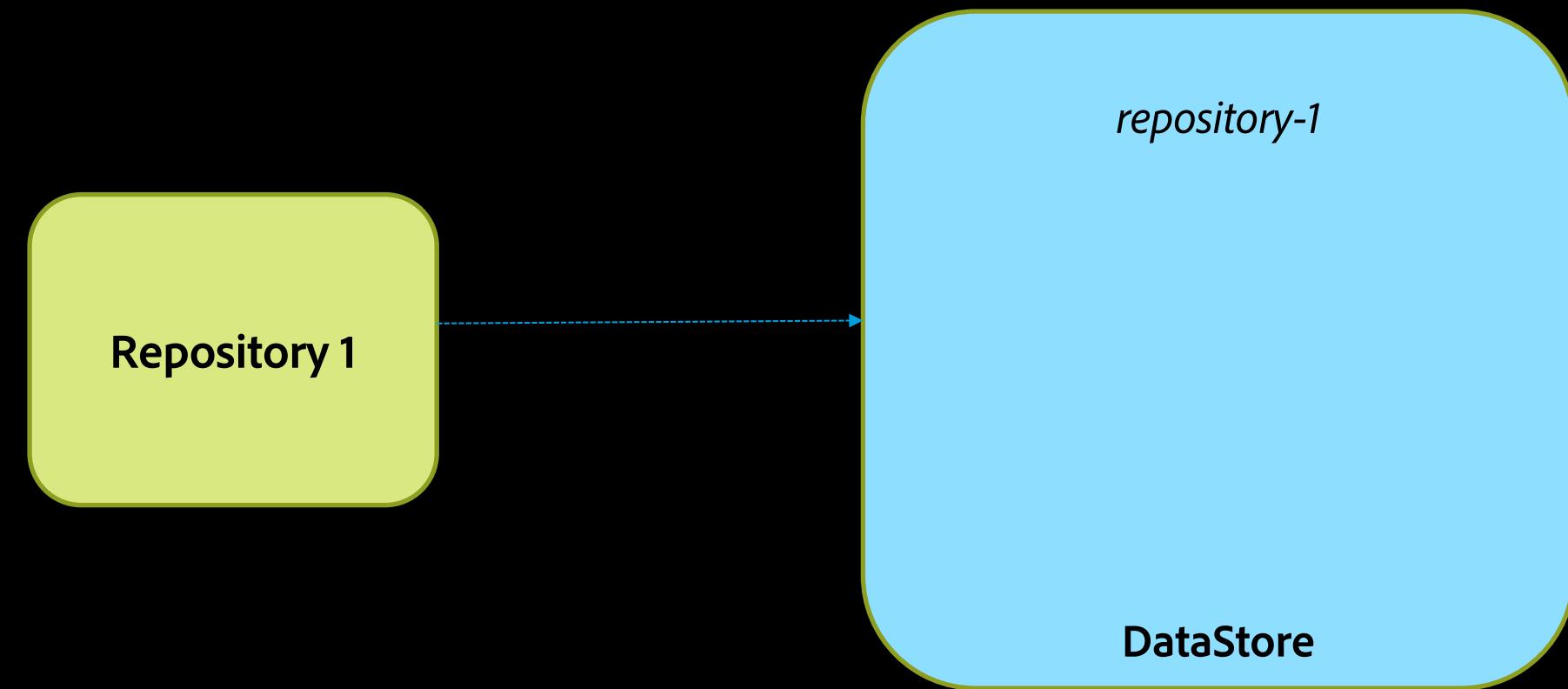
- 2 Phases
 - Mark
 - Retrieve all references stored in the repository by iterating over the node store
 - Mark the starting time by registering the starting timestamp
 - Create the file *markedTimestamp-xxx-xxxxxx-xxxx* in the DataStore
 - Sweep
 - Prepare a candidate list of all garbage blobs by identifying blobs not referred
 - Delete all the candidates older than the configured age from the start time recorded above
 - Default is blobs older than 24 hours

Shared DataStore

- *Shared* refers to 2 different repositories sharing the same DataStore e.g. an author cluster sharing the same DataStore with a publish farm
- Enables binary less replication as the binary already available in the DataStore

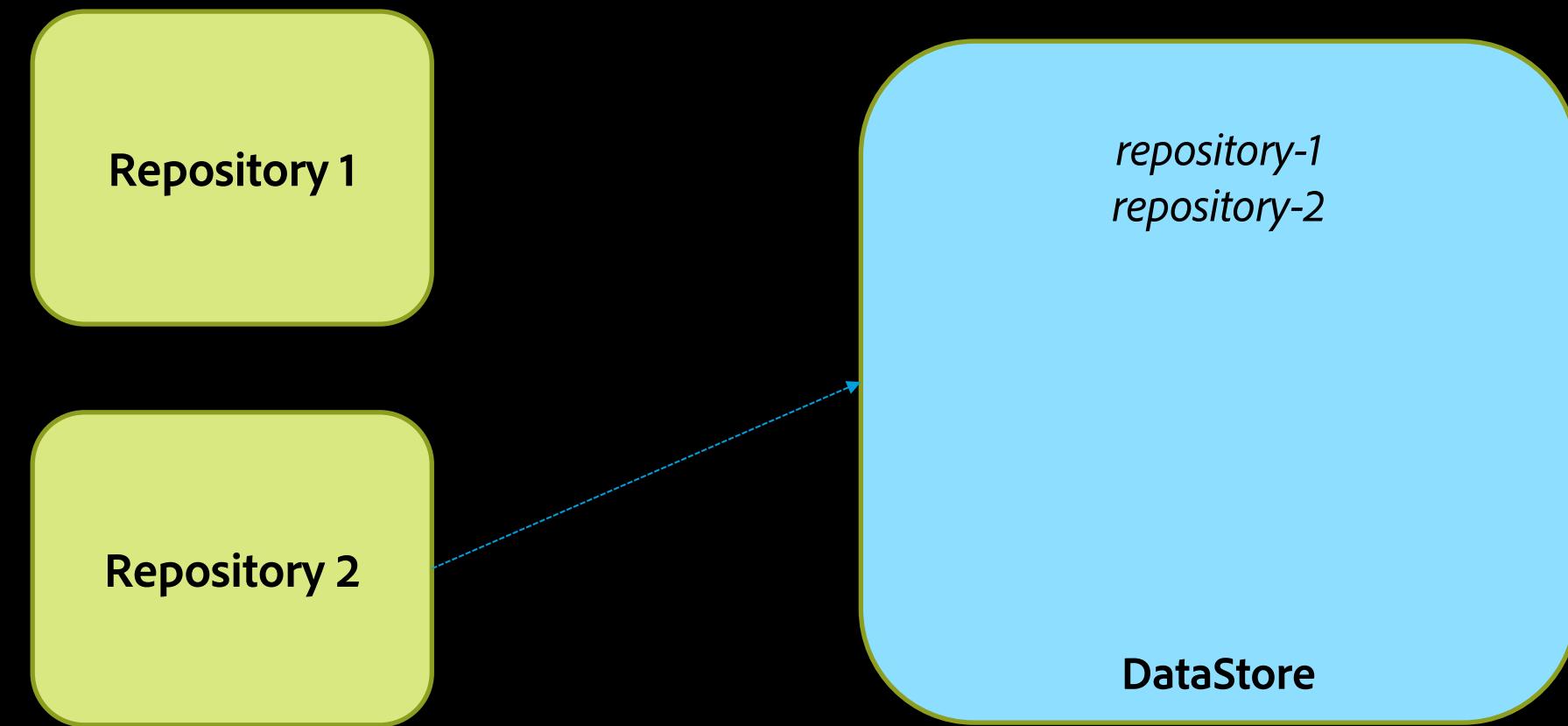
Shared DataStore

- Repository 1 connects to the DataStore and registers



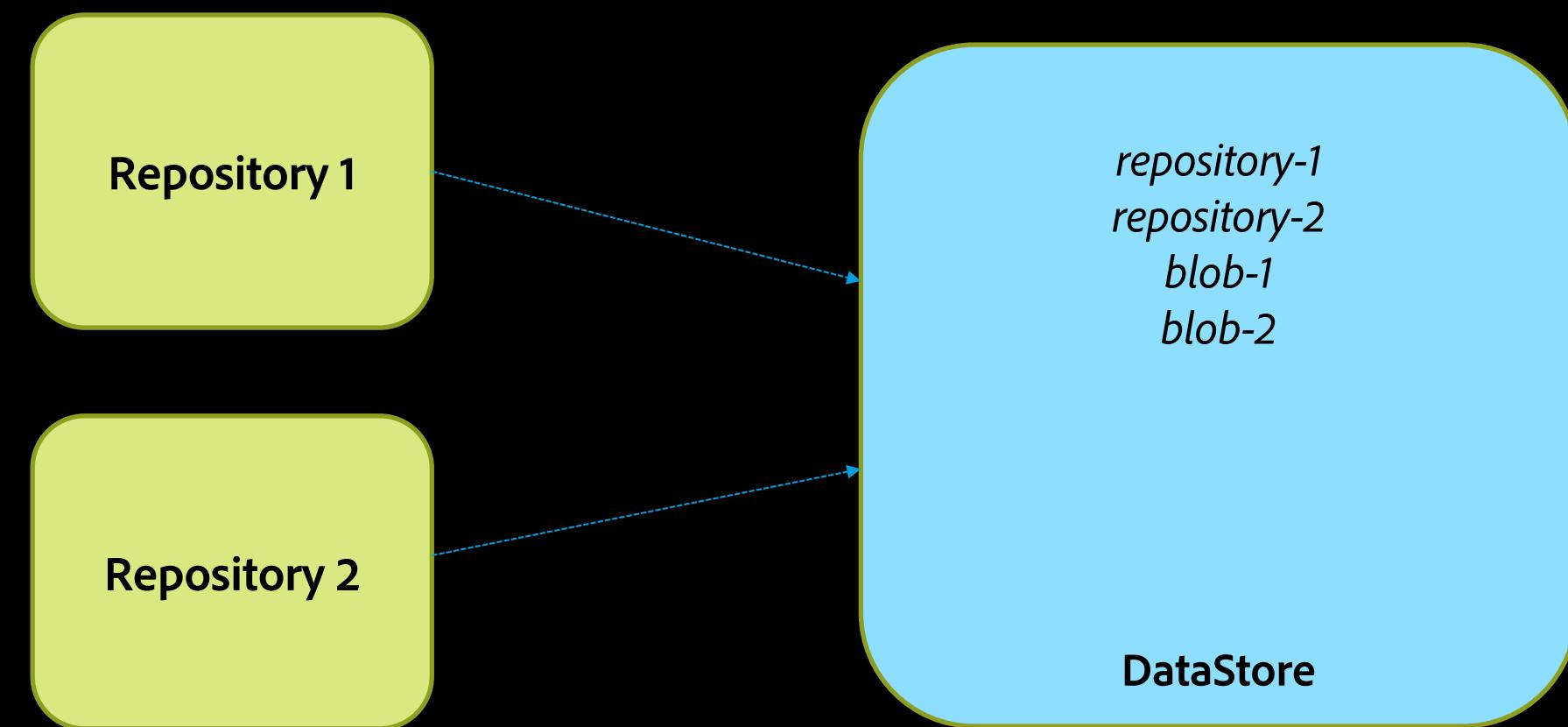
Shared DataStore

- Repository 2 connects to the DataStore and registers



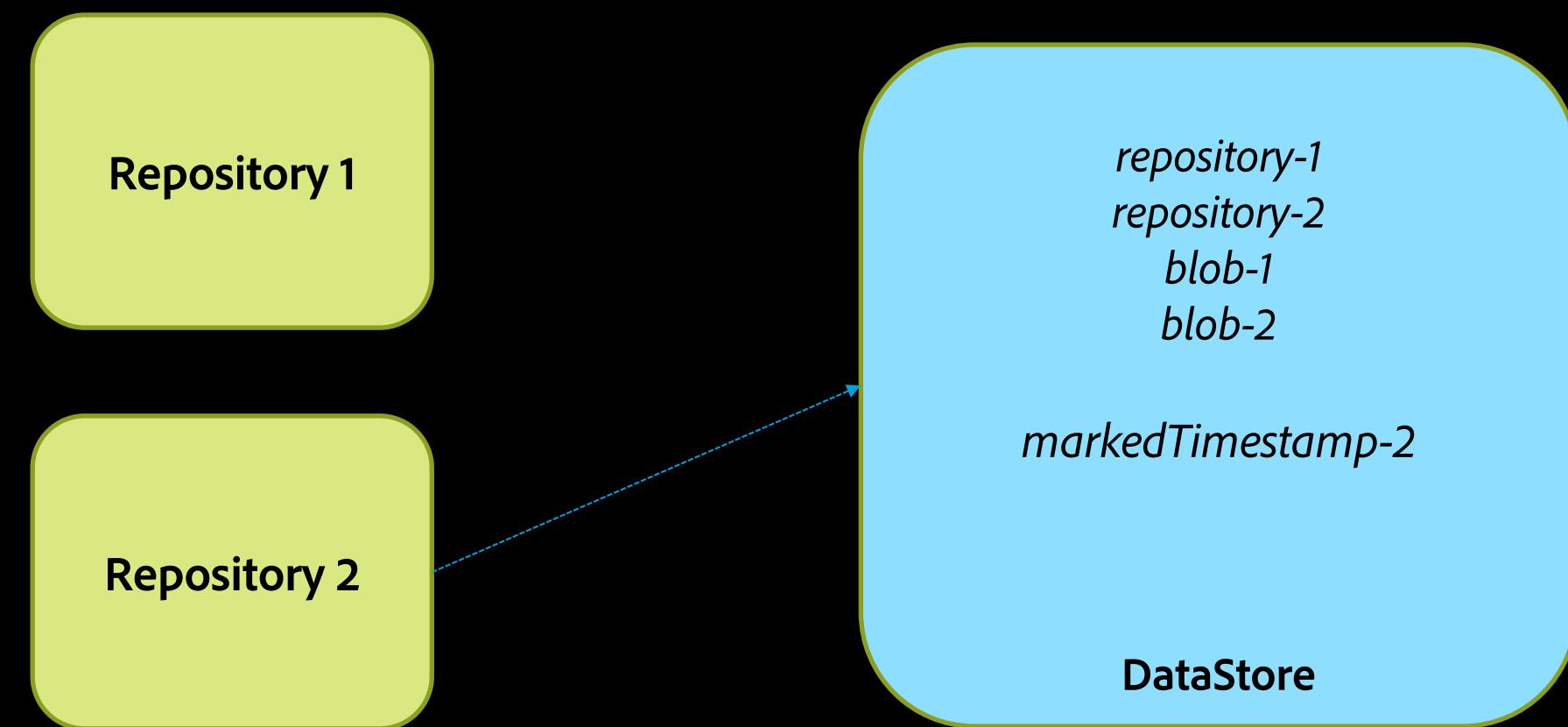
Shared DataStore

- Repository 1 and 2 upload cached blob ids at a later time



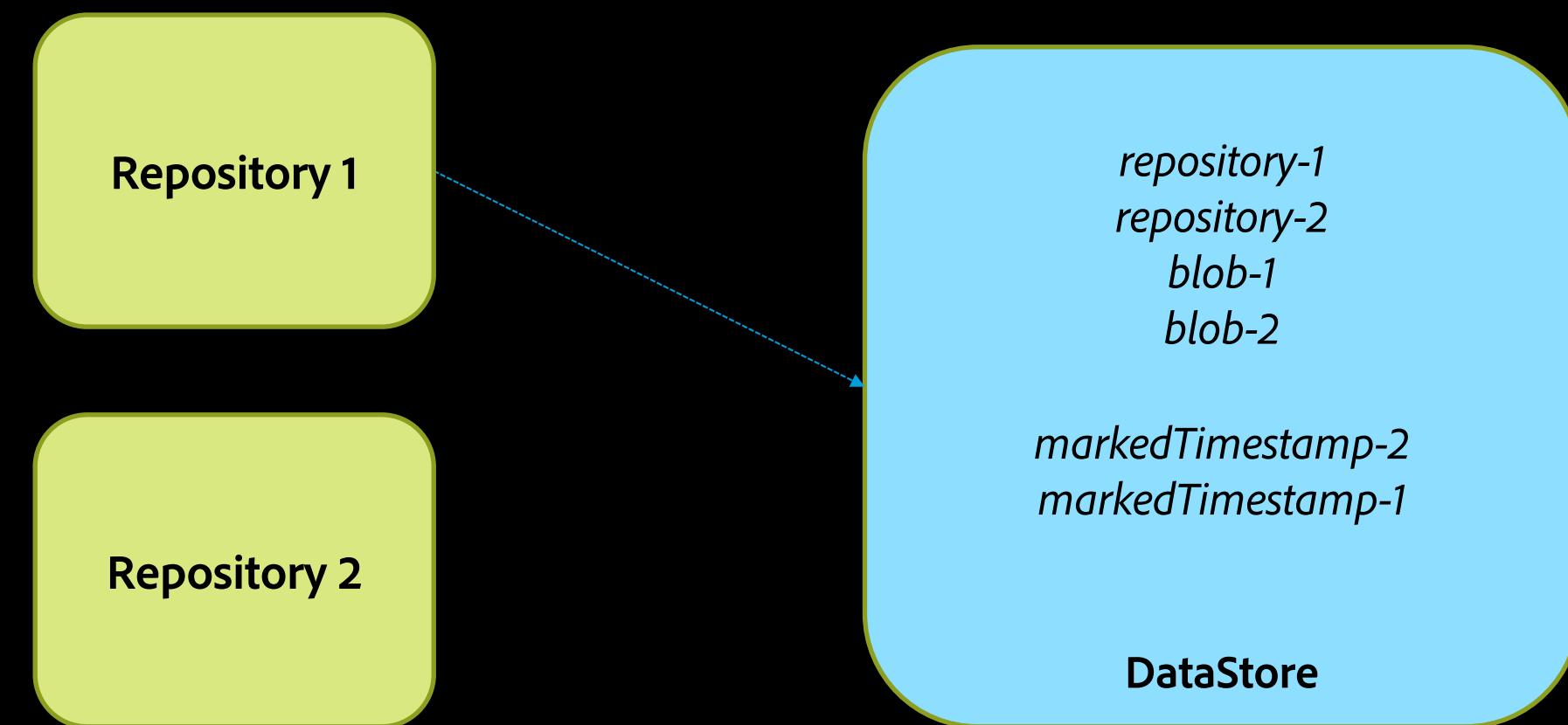
Shared DataStore GC – Mark Phase

- Repository 2 starts the *Mark* phase and adds a starting marker to the DataStore



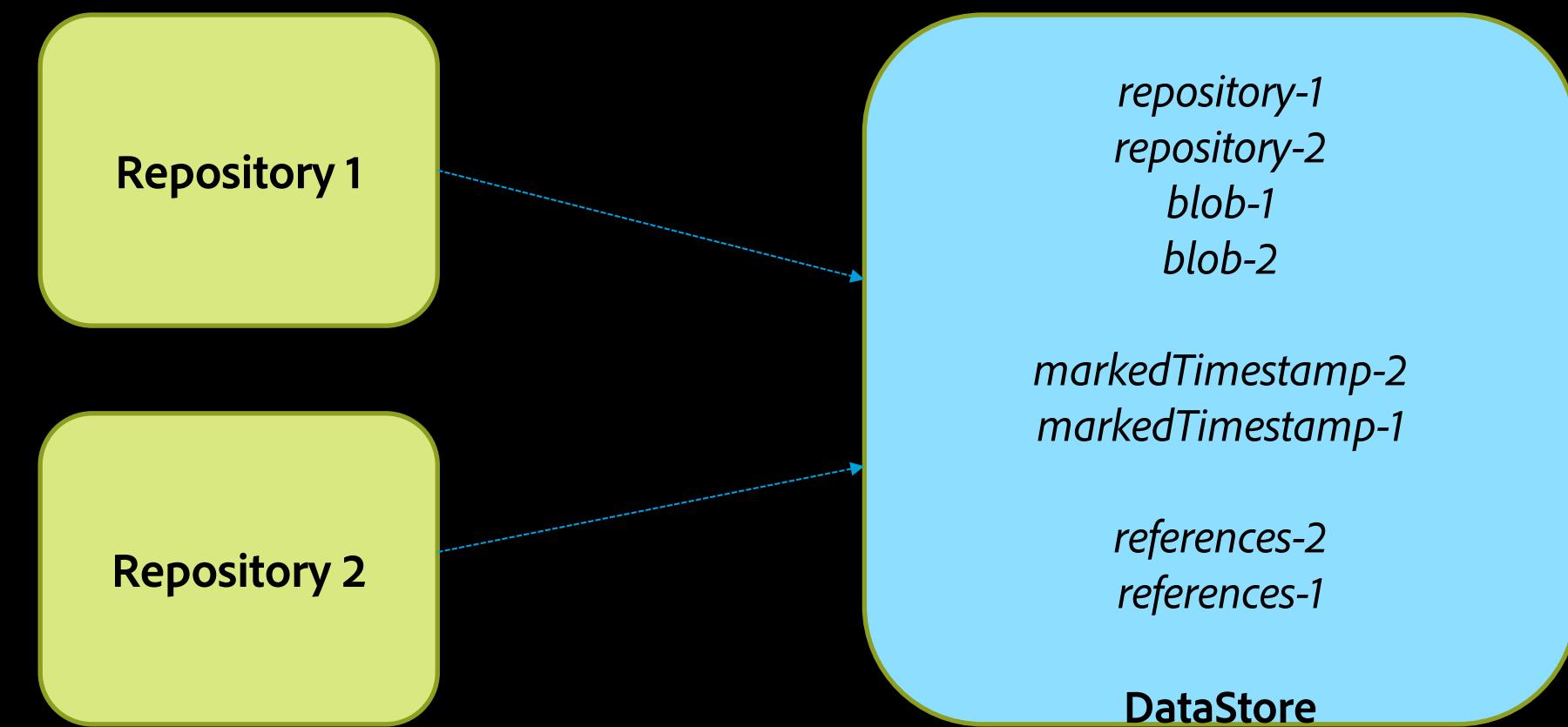
Shared DataStore GC – Mark Phase

- Repository 1 starts the *Mark* phase and adds a starting marker to the DataStore



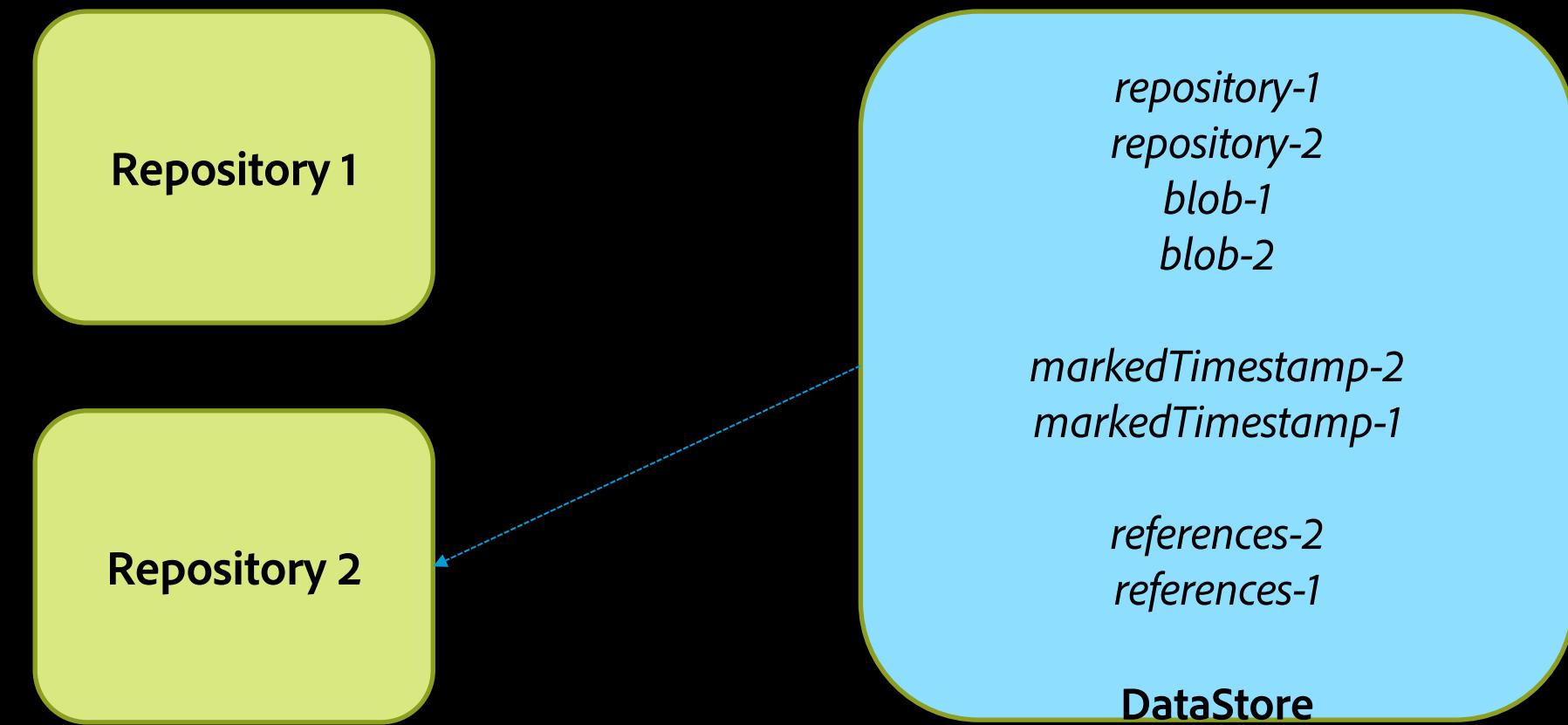
Shared DataStore GC – Mark Phase

- Repository 1 and 2 add references on respective *Mark* phase completion



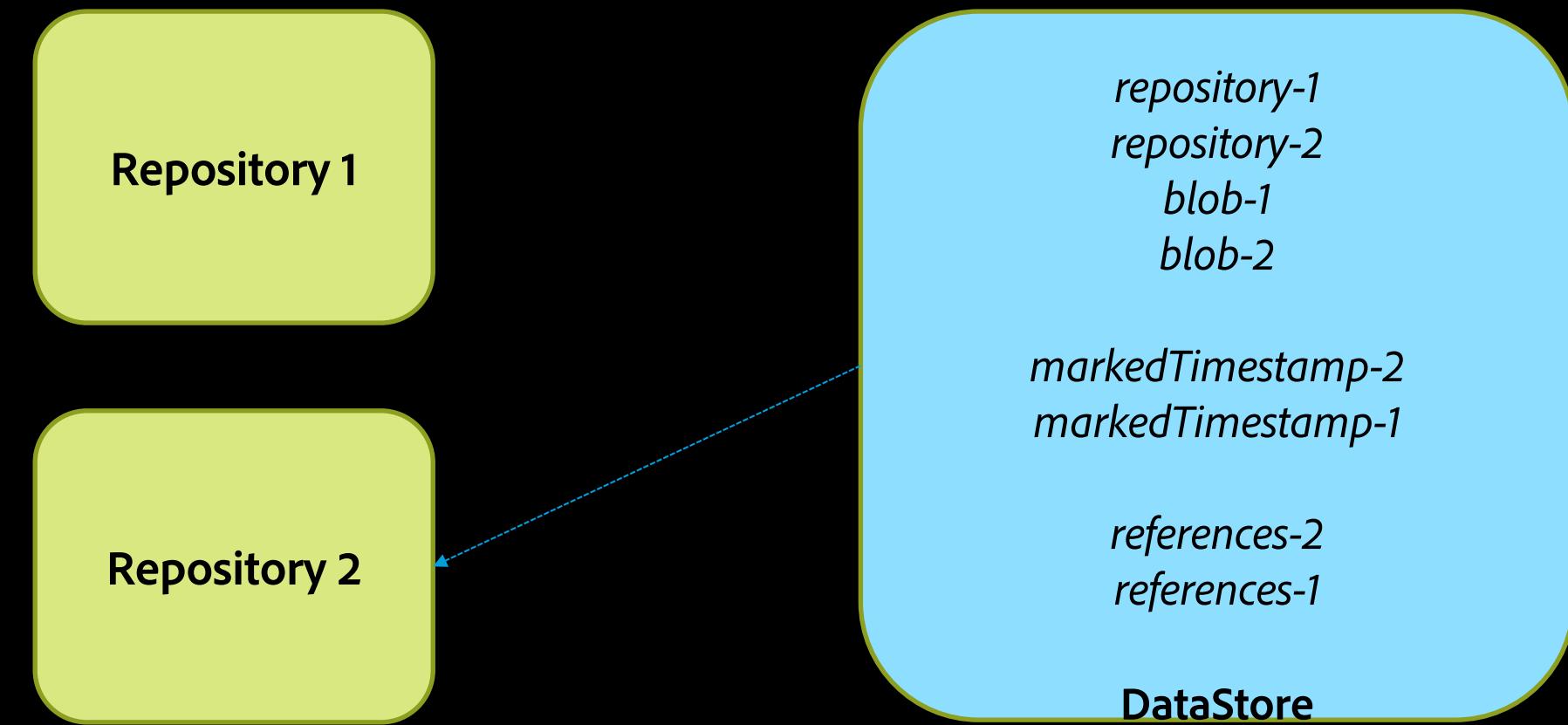
Shared DataStore GC – Sweep Phase

- References collected from the DataStore



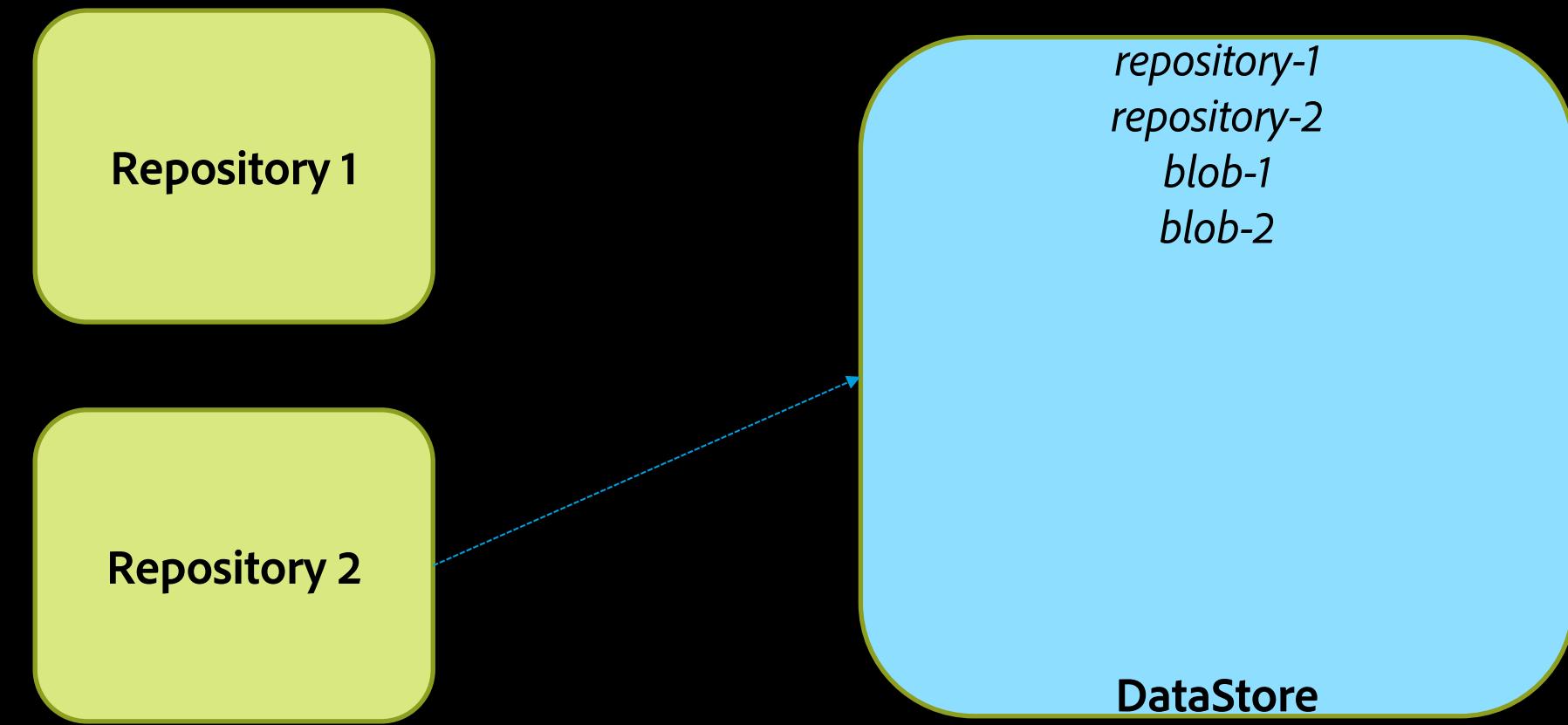
Shared DataStore GC – Sweep Phase

- Blob ids collected from the DataStore



Shared DataStore GC – Sweep Phase

- Use the earliest timestamp to ascertain age to delete blobs and clean up state to finish



Mechanism – Shared DataStore GC

- 2 Phases
 - Mark on each repository
 - Mark the starting time by registering the starting timestamp
 - Create the file *markedTimestamp-xxx-xxxxxx-xxxx* in the DataStore
 - Retrieve all references stored in the repository by iterating over the node store
 - Store the references collected in the DataStore
 - Create the file *references-xxx-xxxxxx-xxxx* in the DataStore
 - Sweep – On a single repository
 - Collect all the references available from the DataStore and aborts if not missing from any repository.
 - Prepare a candidate list of all garbage blobs by identifying blobs not referred
 - Delete all the candidates older than configured age from the earliest start time recorded above (Default 24 hours)

Mechanism – Shared DataStore GC – Important Note

- When cloning publish instances intended to connect to the same DataStore (i.e. *shared*)
 - Reset the repository id (*oak-run resetClusterId*)
- If a repository removed from sharing the DataStore
 - Manually remove the repository id

DataStore GC Log Output

- 11.04.2017 11:30:00.077 *INFO* [sling-oak-observation-1] org.apache.jackrabbit.oak.plugins.blob.MarkSweepGarbageCollector Starting Blob garbage collection with markOnly [false]
- 11.04.2017 11:32:40.513 *INFO* [sling-oak-observation-1] org.apache.jackrabbit.oak.plugins.blob.MarkSweepGarbageCollector Number of valid blob references marked under mark phase of Blob garbage collection [2678506]
- 11.04.2017 11:33:03.021 *INFO* [sling-oak-observation-1] org.apache.jackrabbit.oak.plugins.blob.MarkSweepGarbageCollector Length of blob ids file retrieved from tracker 62939419
- 11.04.2017 11:44:35.278 *WARN* [sling-oak-observation-1] org.apache.jackrabbit.oak.plugins.blob.MarkSweepGarbageCollector Deleted only [669806] blobs entries from the [727556] candidates identified. This may happen if blob modified time is > than the max deleted time (2017-04-10 11:30:00.000)
- 11.04.2017 11:44:35.279 *INFO* [sling-oak-observation-1] org.apache.jackrabbit.oak.plugins.blob.MarkSweepGarbageCollector Blob garbage collection completed in 14.59 min (875201 ms). Number of blobs deleted [669806] with max modification time of [2017-04-10 11:30:00.077]

BlobTracker

- Retrieving available blob ids most expensive operation for larger DataStores during DataStore GC
- BlobTracker added in AEM 6.3 to locally cache blob ids created
- During GC this locally available information is used which greatly reduces the time for GC completion

BlobTracker - Mechanism

- Active writes for fresh blob ids into *blobids/blob-[repositoryId].gen.process*
- On snapshot and initialization the active file gets renamed to *.gen* and a new *.process* file created
- Regular snapshots are taken to merge all the generation (*.gen*) files to a *.refs* file and uploaded to the DataStore
- During DataStore GC when blob ids requested all the *.refs* available in the DataStore are merged locally and ids returned from there
- The deletes from GC are also synchronized to remove stale ids and a snapshot taken

BlobTracker - Mechanism

Support for Shared DataStores or Clustered setups

- The cached blob ids are local and hence have no information on blob ids created through other nodes in a clustered setup or different repositories in a shared setup. To enable synchronization of this information regular snapshot process
- Regular snapshots from all repositories ensures all their information is captured
- Interval configured by *blobTrackSnapshotIntervalInSecs* and by default is 12 hours
- Interval also governed by *blobGcMaxAgeInSecs* which is by default set to 24 hours. Which means with default setting the blob ids created within 12 hours may not be returned but which is Ok as only blobs older than 24 hours are to be deleted

BlobTracker – Coldstart

- Coldstart problems if local ids not available or incomplete
 - On upgraded systems
 - Inadvertent removal of locally tracked files
- Not fatal i.e. would not lead to data loss on running GC but would make GC less effective
- Solution is to force retrieval of blob ids using the BlobGC Mbean
 - Execute *checkConsistency*
 - Execute *startBlobGc (true, true)* – second parameter forces retrieval of blob ids

BlobStoreTracker - continued

javax.management.openmbean.CompositeData startBlobGC(boolean markOnly, boolean forceBlobIdRetrieve) ×

javax.management.openmbean.CompositeData startBlobGC(boolean markOnly, boolean forceBlobIdRetrieve)

Operation exposed for management

boolean markOnly true

Set to true to only mark references and not sweep in the mark and sweep operation. This mode is to be used when the underlying BlobStore is shared between multiple different repositories. For all other cases set it to false to perform full garbage collection

boolean forceBlobIdRetrieve true

Set to true to force retrieve all ids from the datastore bypassing any local tracking

Invoke

Execution

- Weekly maintenance task (Operations Dashboard)
 - Enabled by default at Saturday 1:00 AM
 - Can be configured to a different day and time
 - Though recommended to disable for shared DataStore, will most likely work as chances of repositories finishing mark phase within a few milliseconds remote

Maintenance Task

The screenshot shows the Adobe Experience Manager interface for managing maintenance windows. A modal dialog titled "Configure Maintenance Window" is open in the foreground, overlaid on a list of existing maintenance windows.

The dialog fields are as follows:

- Name:** Weekly Maintenance Window
- Recurrence:** Weekly (radio button selected)
- Start:** Saturday at 01:00
- End:** Saturday at 02:00

At the bottom right of the dialog are "Save" and "Cancel" buttons.

In the background, the main interface shows two maintenance windows listed:

- Daily Maintenance Window:** Daily: 2:00 to 5:00, Next: May 02 2017 02:00 IST
- Weekly Maintenance Window:** Weekly: Saturday 2:00, Next: May 06

Maintenance Task

The screenshot shows the Adobe Experience Manager interface for managing maintenance tasks. The title bar indicates it is the "Weekly Maintenance Window". The interface displays three tasks:

- Data Store Garbage Collection**: Status: Unknown (Did not run yet). Next run: May 06 2017 01:00 IST.
- Workflow Purge**: Status: Unknown (Did not run yet). Next run: May 06 2017 01:00 IST.
- AuditLog Maintenance Task**: Status: Unknown (Did not run yet). Next run: May 06 2017 01:00 IST.

Each task card includes icons for status, execution, information, and settings.

Execution

- Manually executing `startBlobGc` using `BlobGarbageCollection` Mbean from JMX console
 - `markOnly` – If only mark phase to be run for e.g. for shared DataStore scenario
- Also, shows global statistics relevant for shared DataStore scenario

JMX Console

Adobe Experience Manager Web Console

JMX

Main OSGi Sling Status Web Console Log out

org.apache.jackrabbit.oak: Segment node store blob garbage collection (BlobGarbageCollection)

Information on the management interface of the MBean

Attributes

Attribute Name	Attribute Value																		
BlobGCStatus	<table border="1"><tr><td>status</td></tr><tr><td>code 3</td></tr><tr><td>id 7</td></tr><tr><td>message Blob garbage collection running:</td></tr></table>	status	code 3	id 7	message Blob garbage collection running:														
status																			
code 3																			
id 7																			
message Blob garbage collection running:																			
GlobalMarkStats	<table border="1"><tr><td colspan="6">org.apache.jackrabbit.oak.plugins.blob.BlobGC</td></tr><tr><td>markEndTime</td><td>markStartTime</td><td>numReferences</td><td>referenceFileSizeBytes</td><td>referencesFileSize</td><td>repositoryId</td></tr><tr><td>Mon May 01 14:21:30 IST 2017</td><td>Mon May 01 14:21:29 IST 2017</td><td>2564</td><td>121441</td><td>121.4 kB</td><td>bbba77e2-f8bd-4b46-abbe-ab67d9c87b69 *</td></tr></table>	org.apache.jackrabbit.oak.plugins.blob.BlobGC						markEndTime	markStartTime	numReferences	referenceFileSizeBytes	referencesFileSize	repositoryId	Mon May 01 14:21:30 IST 2017	Mon May 01 14:21:29 IST 2017	2564	121441	121.4 kB	bbba77e2-f8bd-4b46-abbe-ab67d9c87b69 *
org.apache.jackrabbit.oak.plugins.blob.BlobGC																			
markEndTime	markStartTime	numReferences	referenceFileSizeBytes	referencesFileSize	repositoryId														
Mon May 01 14:21:30 IST 2017	Mon May 01 14:21:29 IST 2017	2564	121441	121.4 kB	bbba77e2-f8bd-4b46-abbe-ab67d9c87b69 *														

| ConsistencyCheckStatus | | | |------------| | status | | code 1 | | id 3 | | message NA | |

Operations

Return Type	Name
javax.management.openmbean.CompositeData	startBlobGC(boolean markOnly) Operation exposed for management
javax.management.openmbean.CompositeData	startBlobGC(boolean markOnly, boolean forceBlobIdRetrieve) Operation exposed for management
javax.management.openmbean.CompositeData	checkConsistency() Operation exposed for management

1 | Introduction

2 | Configuration & Deployment

3 | Garbage Collection

4 | Tooling & Troubleshooting

Online Consistency check

Using the BlobGarbageCollection Mbean *checkConsistency()* operation

Return Type	Name
javax.management.openmbean.CompositeData	checkConsistency() Operation exposed for management

javax.management.openmbean.CompositeData checkConsistency()

javafx.management.openmbean.CompositeData checkConsistency()
Operation exposed for management

Invoke

Offline Consistency check

Use oak-run *datastorecheck* command

e.g.

```
java -jar oak-run.jar datastorecheck \
--store crx-quickstart/repository/segmentstore --consistency --dump . \
--fds org.apache.jackrabbit.oak.plugins.blob.datastore.FileDataStore.config
```

- Additional options *-id*, *--refs* to output all the blobids and the references from the node store
 - *--refs* option adds the node path information from where the blob ids are referenced for Mongo

Reset Repository ID

Use oak-run *resetclusterid* command

e.g.

```
java -jar oak-run.jar resetclusterid crx-quickstart/repository/segmentstore
```

Backups

Backup should have the following sequence

- Optionally execute DataStore GC
- Ensure that DataStore GC does not run between the NodeStore and DataStore backup
- Backup NodeStore
 - If S3 and asynchronous uploads enabled then wait for all uploads to finish
 - Check the DataStore cache stats Mbean to confirm all uploads finished
- Backup DataStore
 - If FDS then copied to an appropriate location
 - If S3 then either enable versioning for auto backups on blob change or backup to a lower cost storage like Amazon Glacier

Troubleshooting – Missing blobs

Errors like this indicate missing blobs

- `java.lang.RuntimeException: Error occurred while obtaining InputStream for blobId [xxxxxxxxxxxxxxxxxxxx]` at
`org.apache.jackrabbit.oak.plugins.blob.BlobStoreBlob.getNewStream(BlobStoreBlob.java:49)`
- *Possible reasons*
 - Lucene Indexing cycle > 24 hours - Blobs created could be deleted if Blob GC run mid-way
 - Inadvertent sharing of DataStore prior to AEM 6.1 (fixed with 6.2)
 - Cloned systems not having executed `resetclusterid`
- *Mitigation*
 - Disable GC when running a full re-indexing cycle on very large repositories
 - Reset clusterId for cloned systems before starting
- *Identification*
 - Run JMX `BlobGC#consistencyCheck`
 - `oak-run tool datastorecheck` command

Troubleshooting –Rapid DataStore growth

- *Possible Reasons*
 - No DataStore GC for a while
 - Lucene indexes stored in the DataStore may cause repository growth disproportionate to the content
 - Enhancement in AEM 6.4 for active deletion of unused lucene blobs. With regularly scheduled deletes during the day the repository growth would be under check because of lucene blobs.
- *Mitigation*
 - Ensure that DataStore GC is enabled and run weekly
 - Can also increase DataStore GC frequency (say bi-weekly) and schedule during off peak hours for periods when large number of uploads requested
 - Ensure that revision clean-up enabled and working

Troubleshooting – GC Performance - Effectiveness

Too few blobs deleted / Space not reclaimed

- *Possible reasons*
 - Revision garbage collection not executed
 - BlobTracker cold start problem
 - Blobs or older Node revisions not aged enough depending upon the age setting for Revision GC/Compaction and DataStore GC
- *Mitigation*
 - Not recommended to change the age intervals on production systems
 - Specifically, for S3 check if versioning enabled because older versions if not purged can take up significant space
 - Define a policy to purge versions greater than 1 or purge versions older than a configured time
- *Identification*
 - GC logs the candidates identified as garbage and the number of blobs deleted
 - Check with the oak-run datastorecheck utility

Troubleshooting – GC Performance – Slow

GC finishes in days

- *Possible Reasons*
 - Large repository size – GC performance proportional to repository/datastore size
 - DataStore GC executed for the first time or after a long gap
- *Identification*
 - GC has info level logs for each phase and it is easy to identify the phase taking the longest time
 - Empirically, deletions could take the maximum time even spilling over 24 hours if large number of blobs to be deleted
- *Mitigation*
 - Regular DataStore GC
 - Mark phase (collection of blob references used) can affect general repository performance critically so, should be scheduled during off-peak hours
 - Sweep phase should not critically affect system performance so, Ok to have it continue if it spill over to normal working hours

Upcoming Enhancements

- Active deletion of lucene blobs
 - Help in resolving lot of rapid DataStore growth problems due to lucene indexes
- Enhancements to the oak-run *datastorecheck* command
 - Missing blobs reporting node paths from which referenced to make it easier to restore and resolve problems
- CompositeDataStore (wishlist) -
 - Separate storage for binaries (e.g. lucene)
 - Archival storage for automated backup & restore

References

- *Oak Documentation* - <http://jackrabbit.apache.org/oak/docs/plugins/blobstore.html>
- *AEM DataStore Documentation* - <https://docs.adobe.com/docs/en/aem/6-3/deploy/platform/data-store-config.html>
- *Oak Run Readme* - <https://github.com/apache/jackrabbit-oak/tree/trunk/oak-run>

Q&A

